



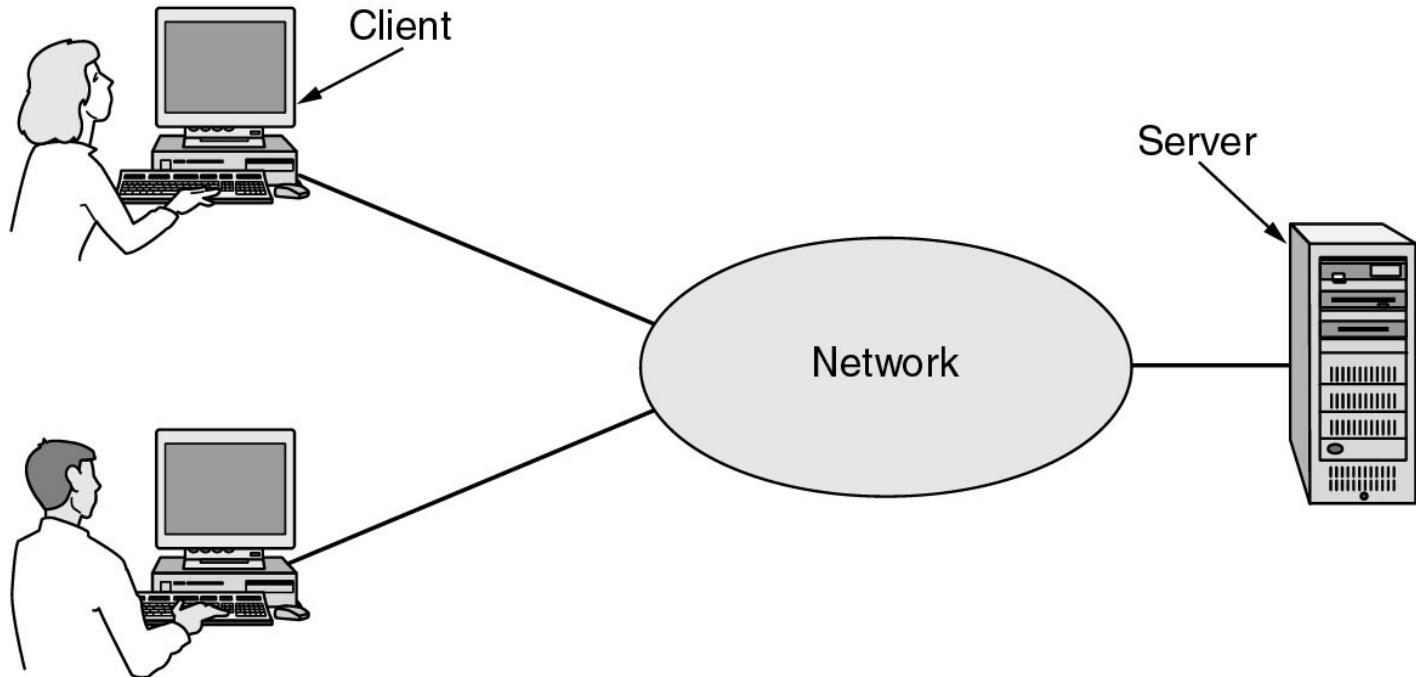
Chapter 1

Introduction

Uses of Computer Networks

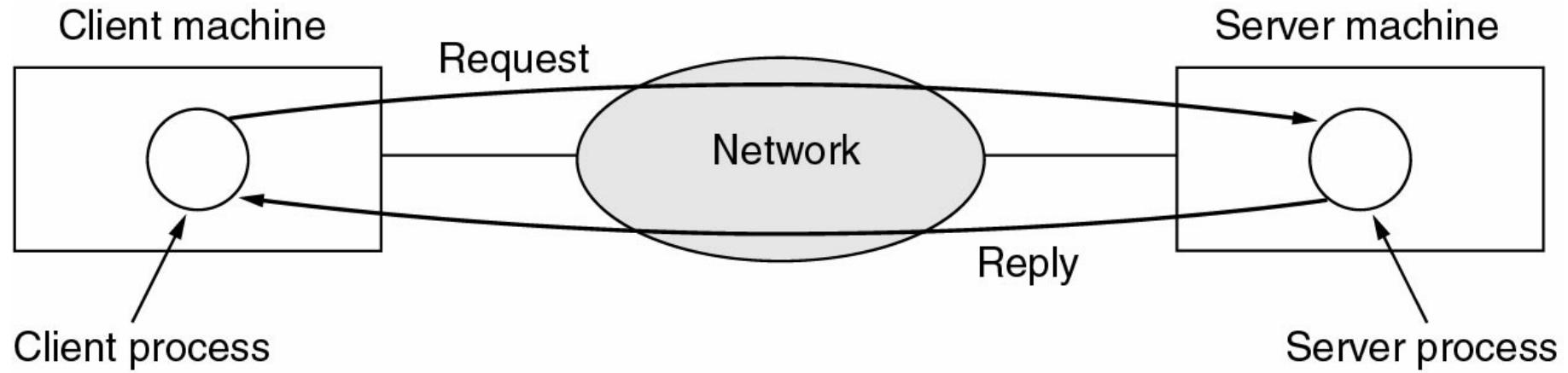
- Business Applications
- Home Applications
- Mobile Users
- Social Issues

Business Applications of Networks



A network with two clients and one server.

Business Applications of Networks (2)

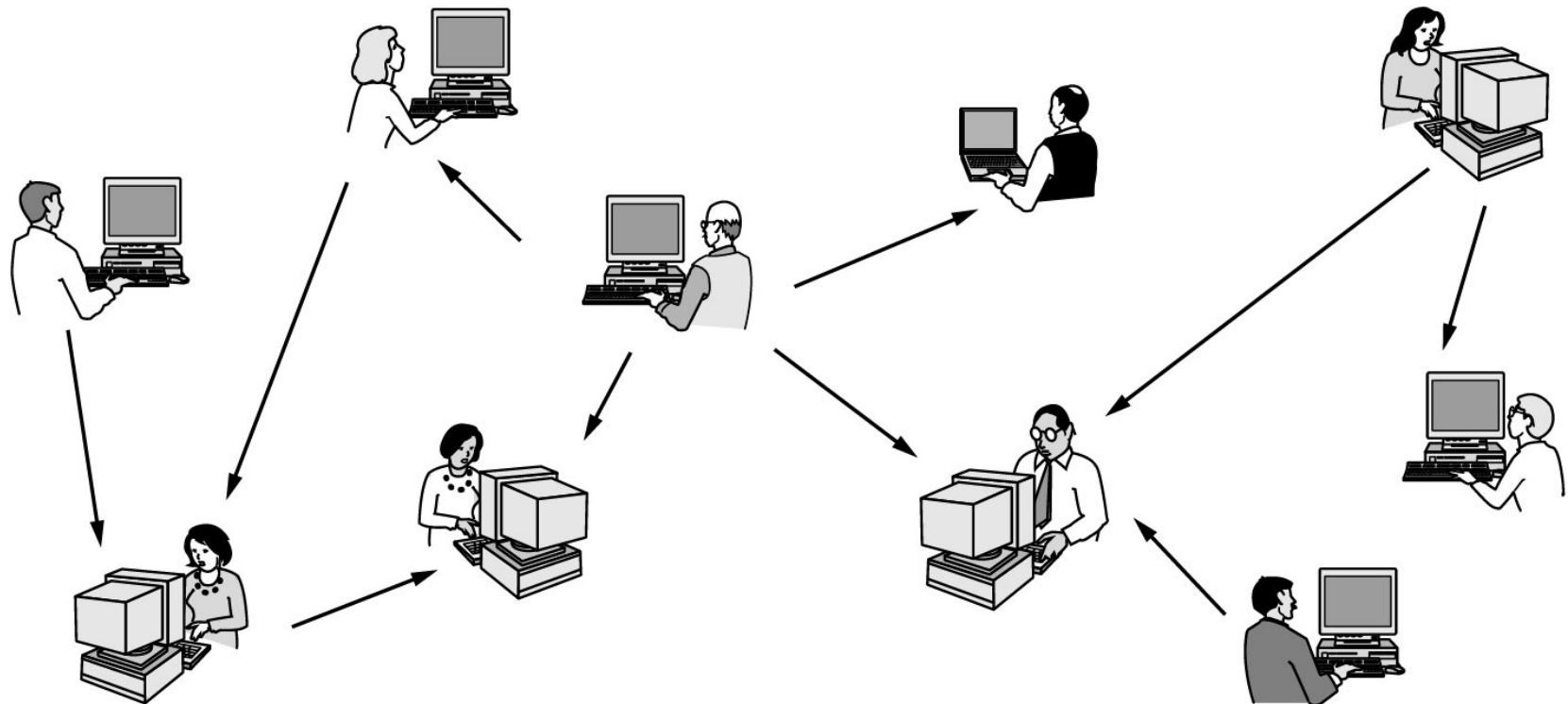


The client-server model involves requests and replies.

Home Network Applications

- Access to remote information
- Person-to-person communication
- Interactive entertainment
- Electronic commerce

Home Network Applications (2)



In peer-to-peer system there are no fixed clients and servers.

Home Network Applications (3)

Tag	Full name	Example
B2C	Business-to-consumer	Ordering books on-line
B2B	Business-to-business	Car manufacturer ordering tires from supplier
G2C	Government-to-consumer	Government distributing tax forms electronically
C2C	Consumer-to-consumer	Auctioning second-hand products on-line
P2P	Peer-to-peer	File sharing

Some forms of e-commerce.

Mobile Network Users

Wireless	Mobile	Applications
No	No	Desktop computers in offices
No	Yes	A notebook computer used in a hotel room
Yes	No	Networks in older, unwired buildings
Yes	Yes	Portable office; PDA for store inventory

Combinations of wireless networks and mobile computing.

Network Hardware

- Local Area Networks
- Metropolitan Area Networks
- Wide Area Networks
- Wireless Networks
- Home Networks
- Internetworks

Broadcast Networks

Types of transmission technology

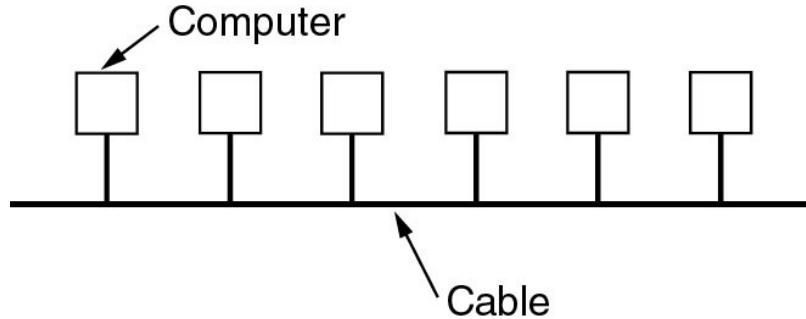
- Broadcast links
- Point-to-point links

Broadcast Networks (2)

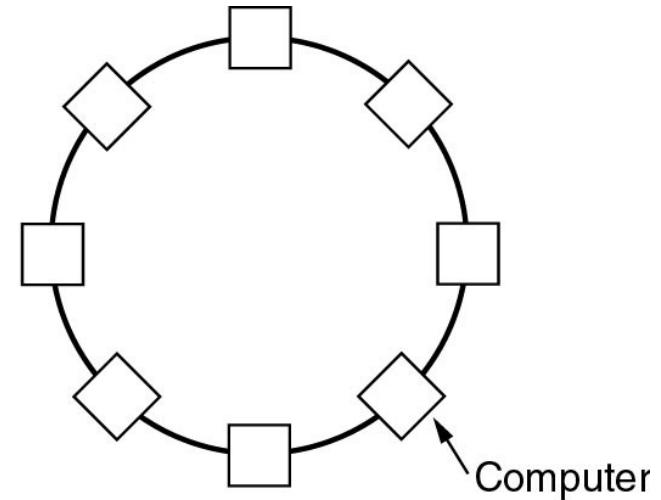
Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	
100 m	Building	Local area network
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	
1000 km	Continent	Wide area network
10,000 km	Planet	The Internet

Classification of interconnected processors by scale.

Local Area Networks



(a)



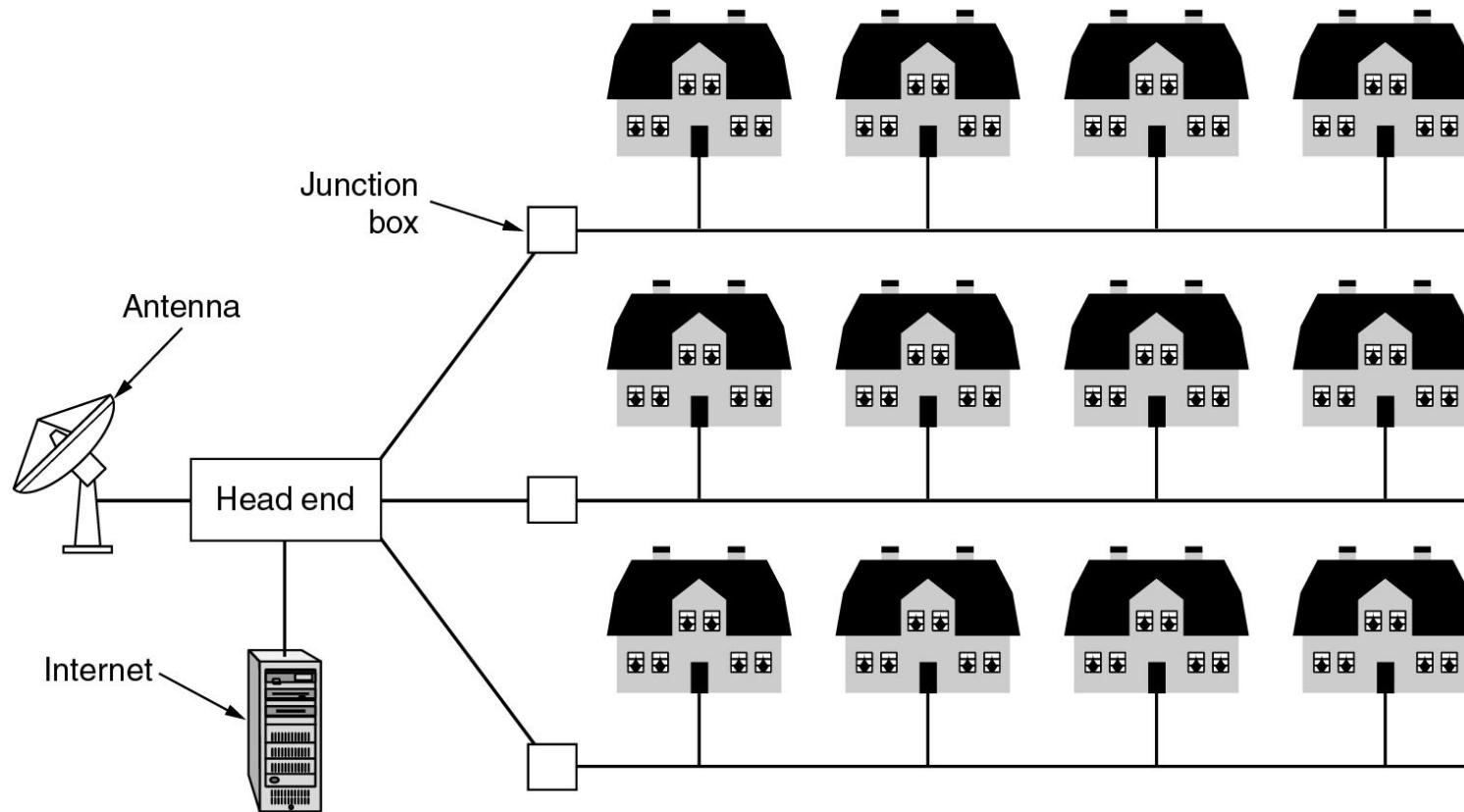
(b)

Two broadcast networks

(a) Bus

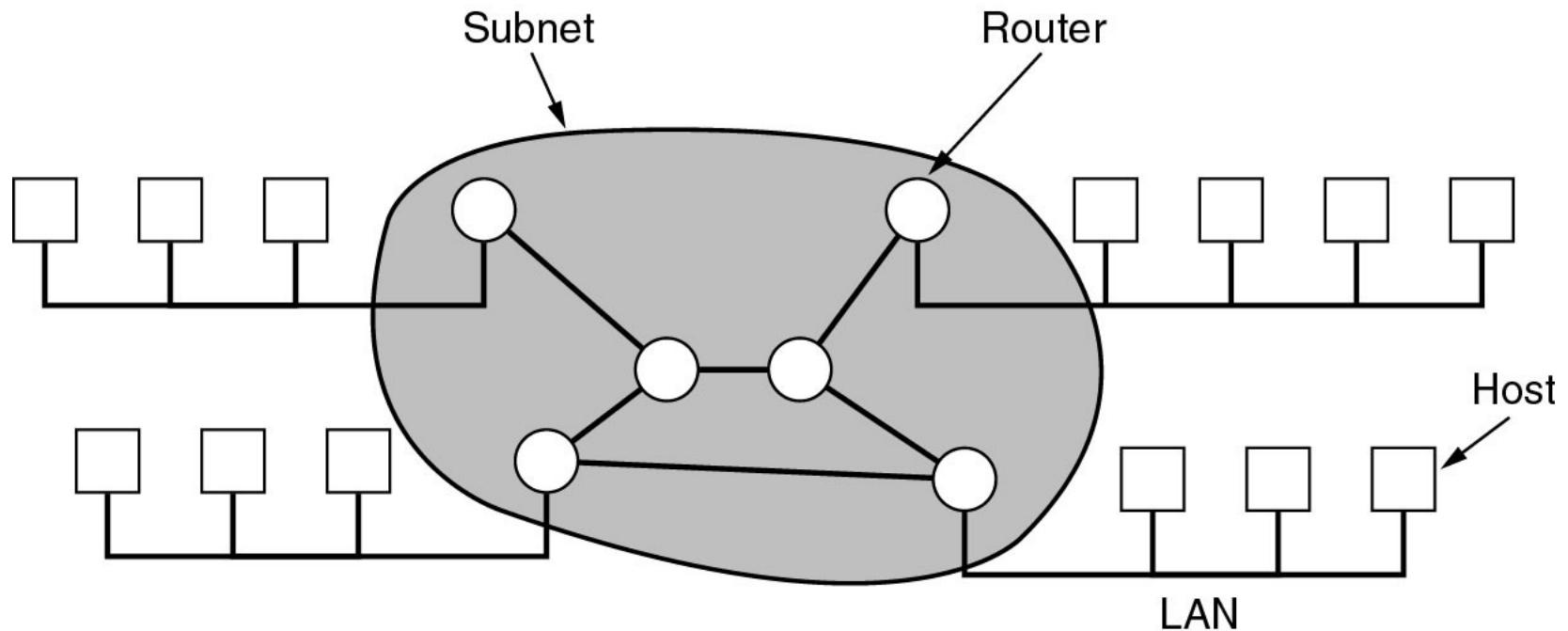
(b) Ring

Metropolitan Area Networks



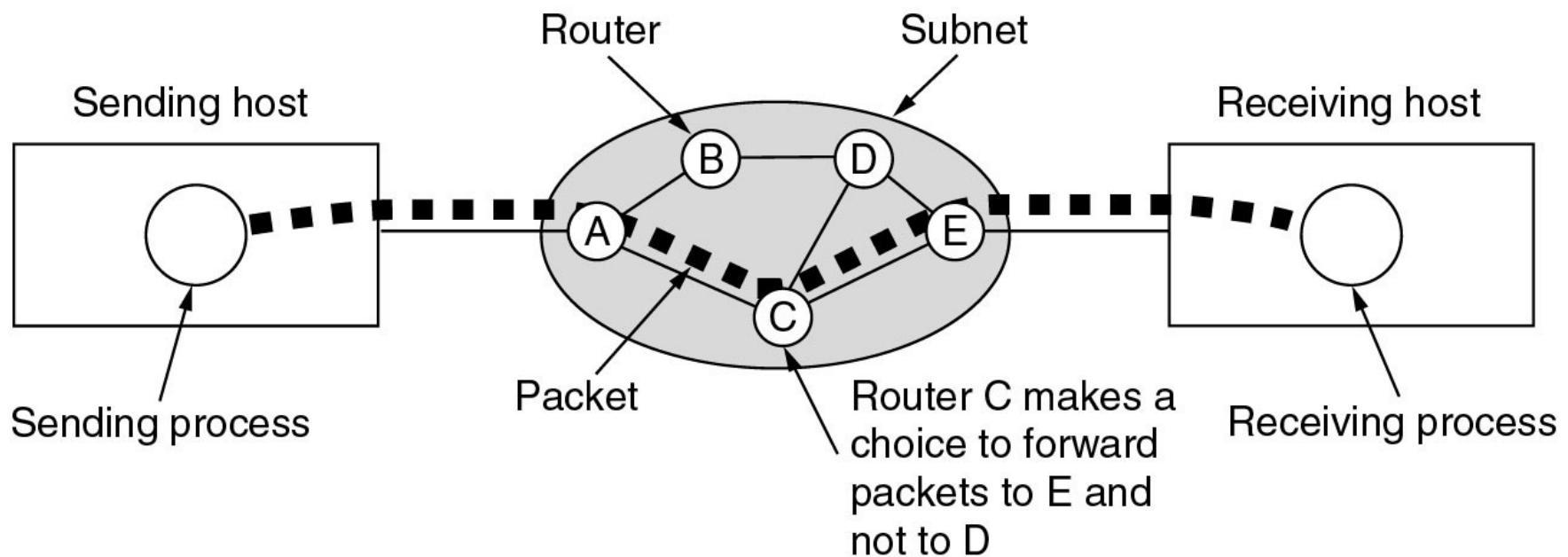
A metropolitan area network based on cable TV.

Wide Area Networks



Relation between hosts on LANs and the subnet.

Wide Area Networks (2)



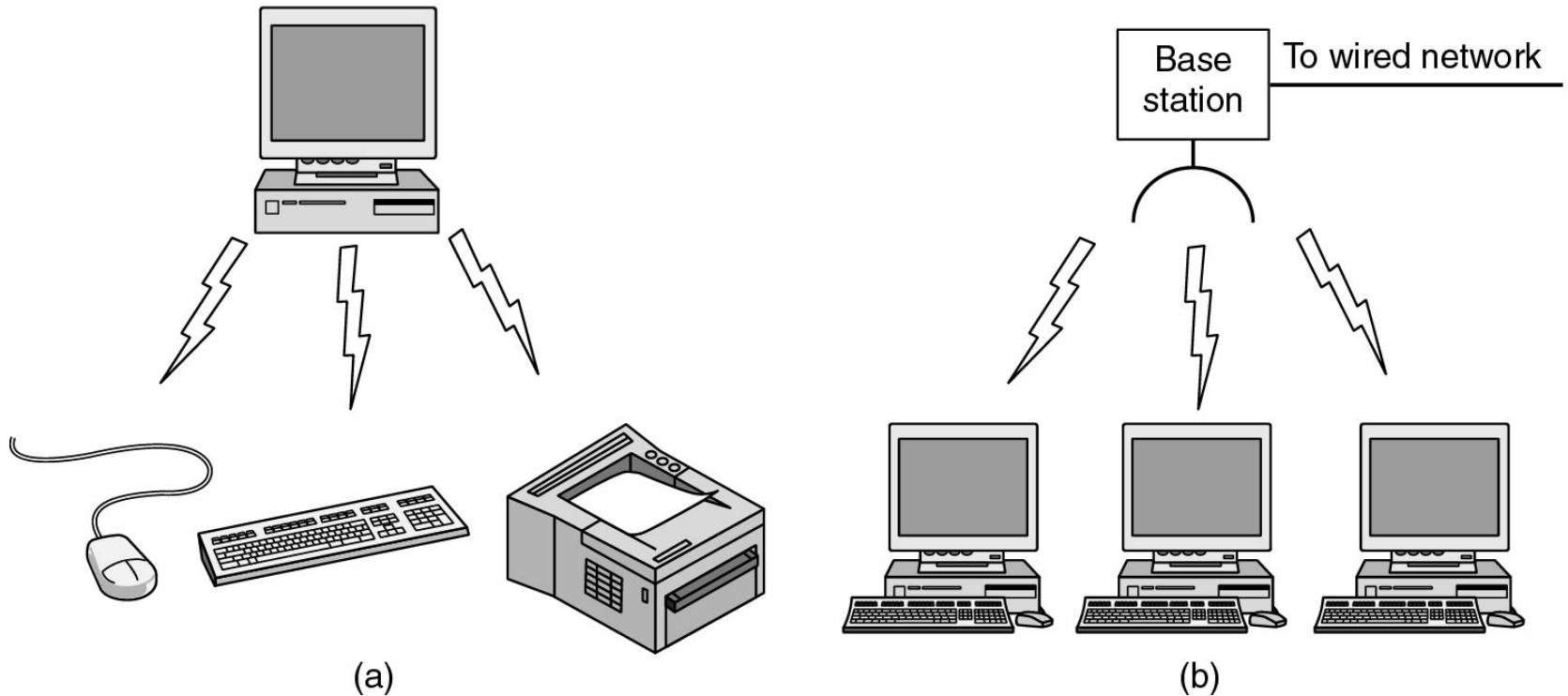
A stream of packets from sender to receiver.

Wireless Networks

Categories of wireless networks:

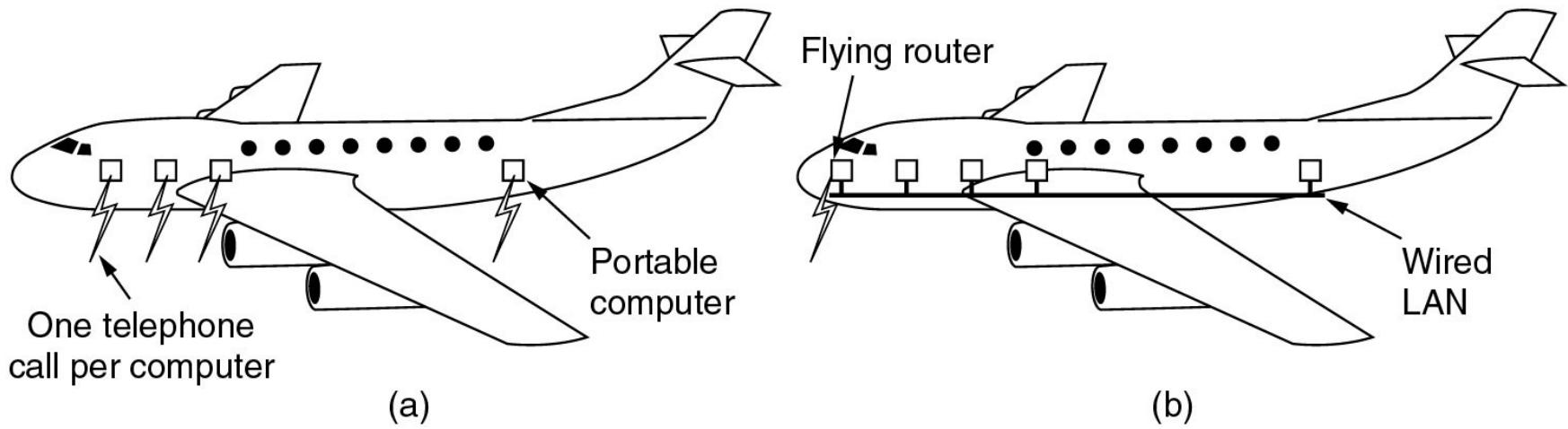
- System interconnection
- Wireless LANs
- Wireless WANs

Wireless Networks (2)



- (a) Bluetooth configuration
- (b) Wireless LAN

Wireless Networks (3)



- (a) Individual mobile computers
- (b) A flying LAN

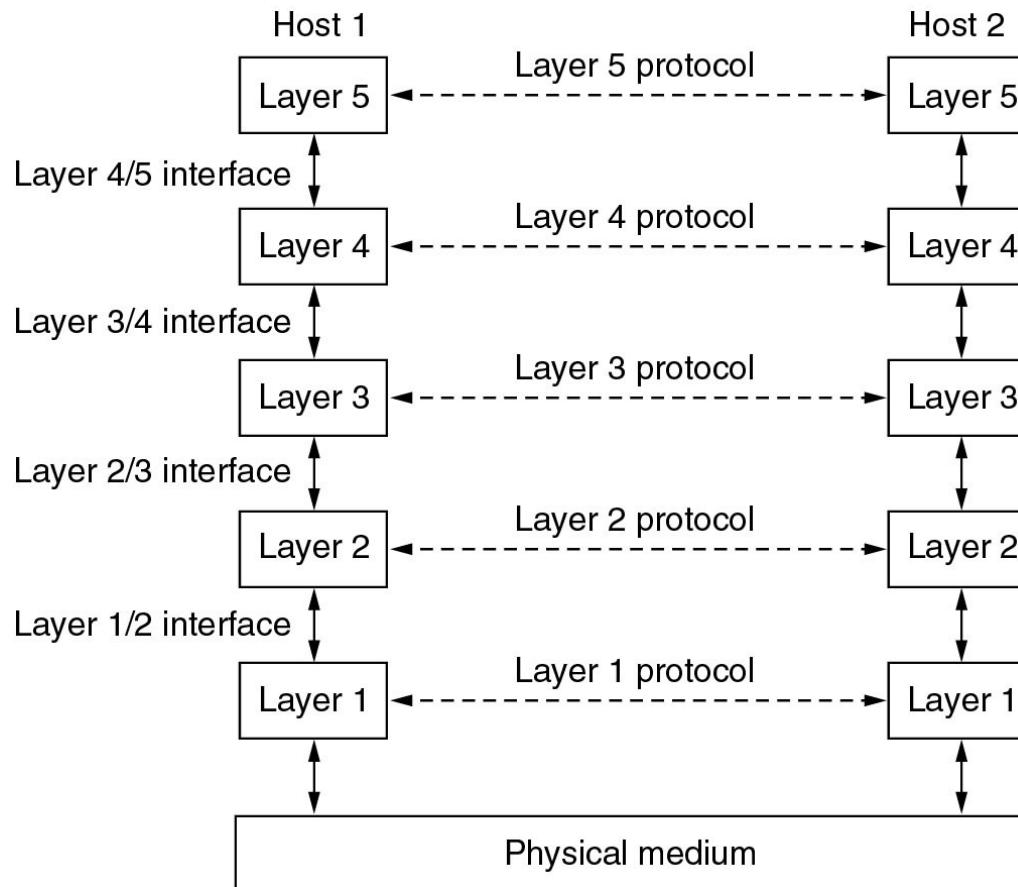
Home Network Categories

- Computers (desktop PC, PDA, shared peripherals)
- Entertainment (TV, DVD, VCR, camera, stereo, MP3)
- Telecomm (telephone, cell phone, intercom, fax)
- Appliances (microwave, fridge, clock, furnace, airco)
- Telemetry (utility meter, burglar alarm, babycam).

Network Software

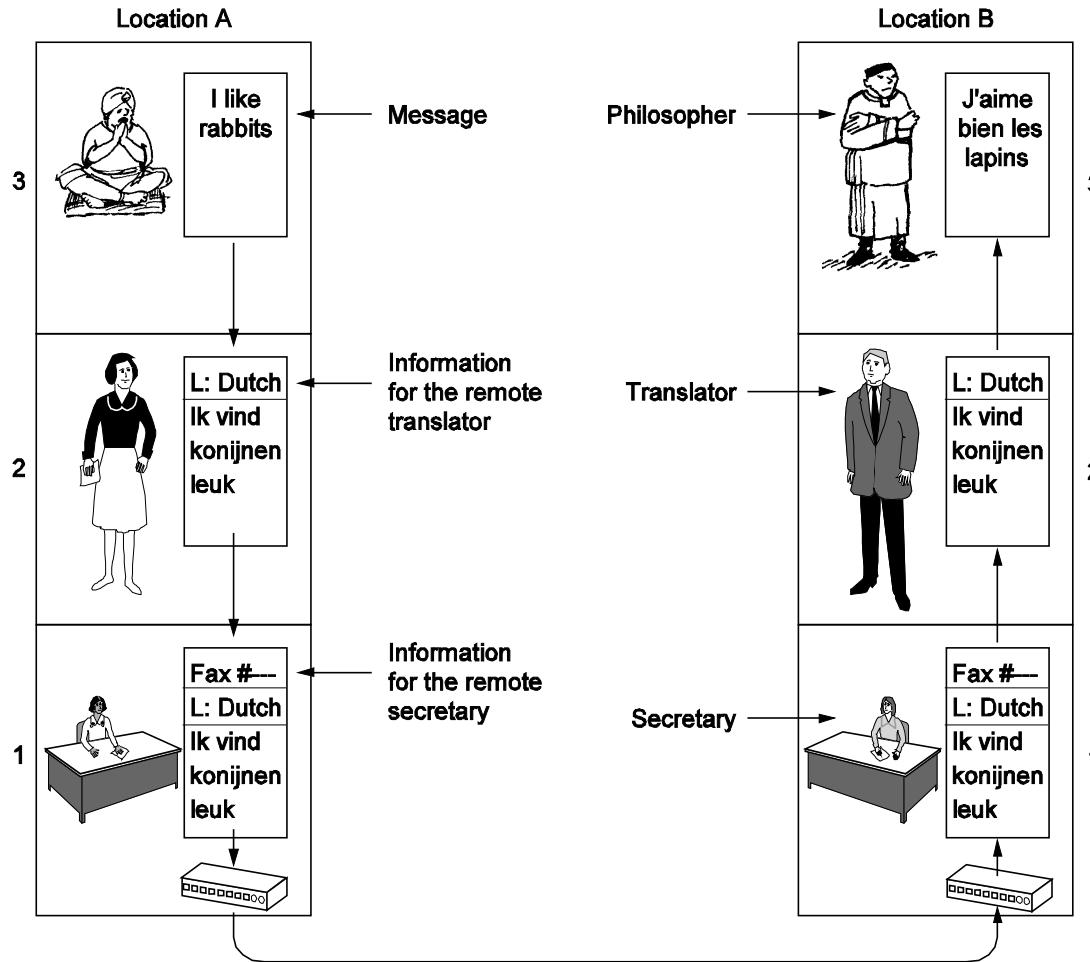
- Protocol Hierarchies
- Design Issues for the Layers
- Connection-Oriented and Connectionless Services
- Service Primitives
- The Relationship of Services to Protocols

Network Software Protocol Hierarchies



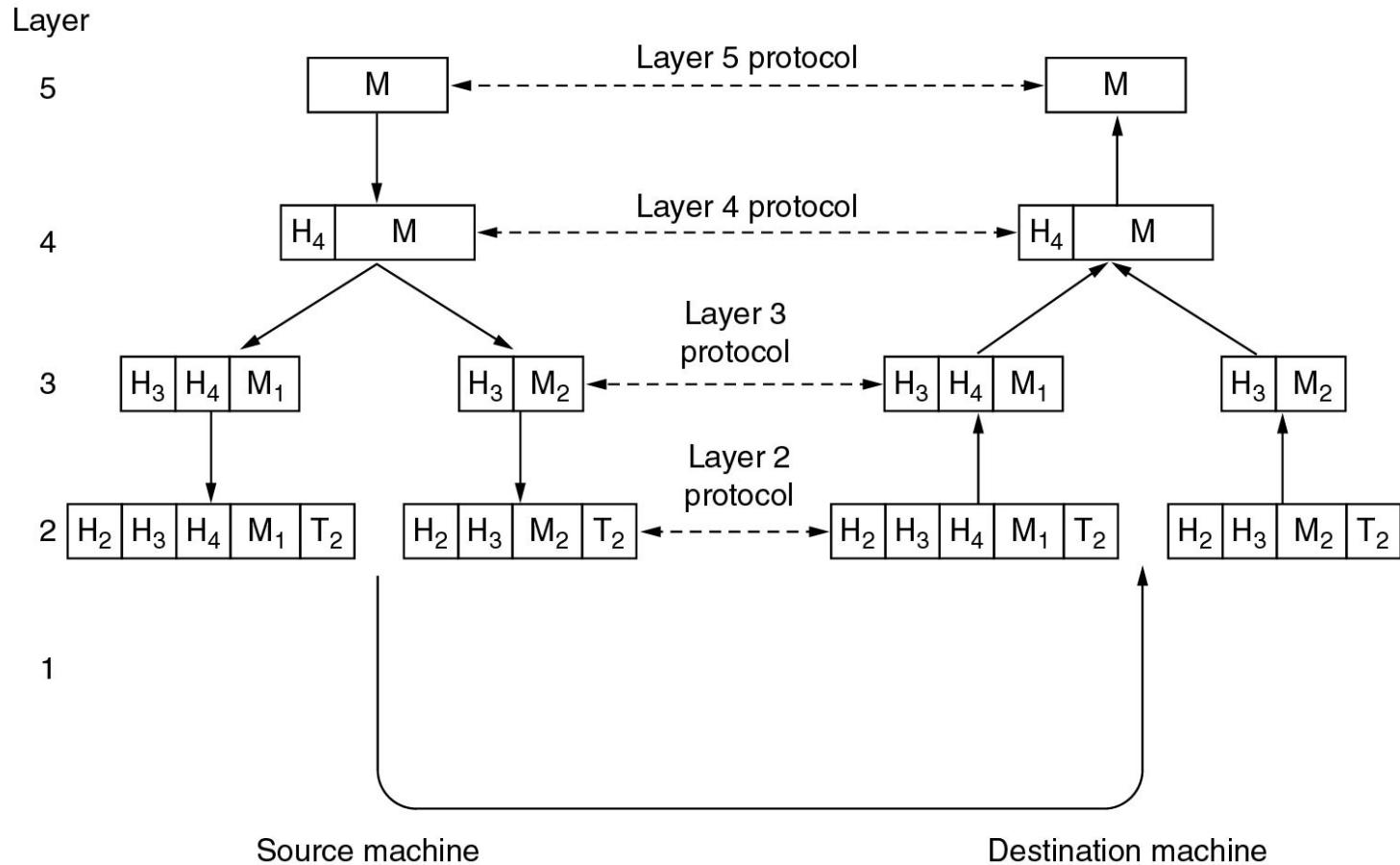
Layers, protocols, and interfaces.

Protocol Hierarchies (2)



The philosopher-translator-secretary architecture.

Protocol Hierarchies (3)



Example information flow supporting virtual communication in layer 5.

Design Issues for the Layers

- Addressing
- Error Control
- Flow Control
- Multiplexing
- Routing

Connection-Oriented and Connectionless Services

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

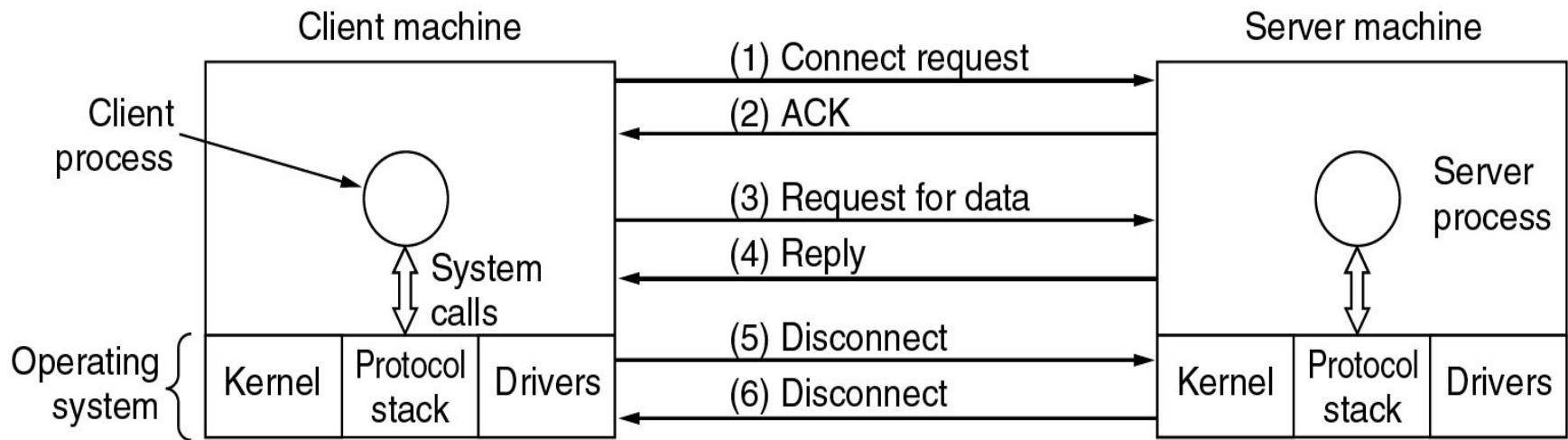
Six different types of service.

Service Primitives

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

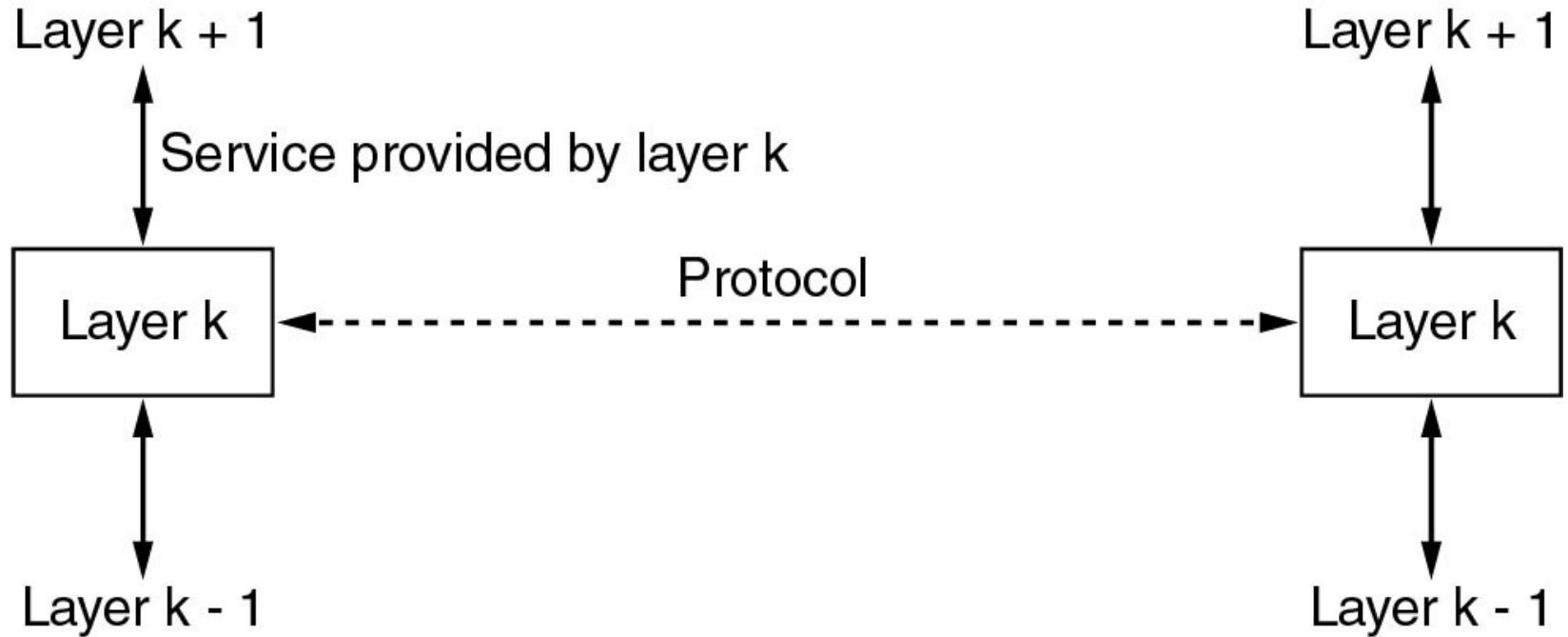
Five service primitives for implementing a simple connection-oriented service.

Service Primitives (2)



Packets sent in a simple client-server interaction on a connection-oriented network.

Services to Protocols Relationship



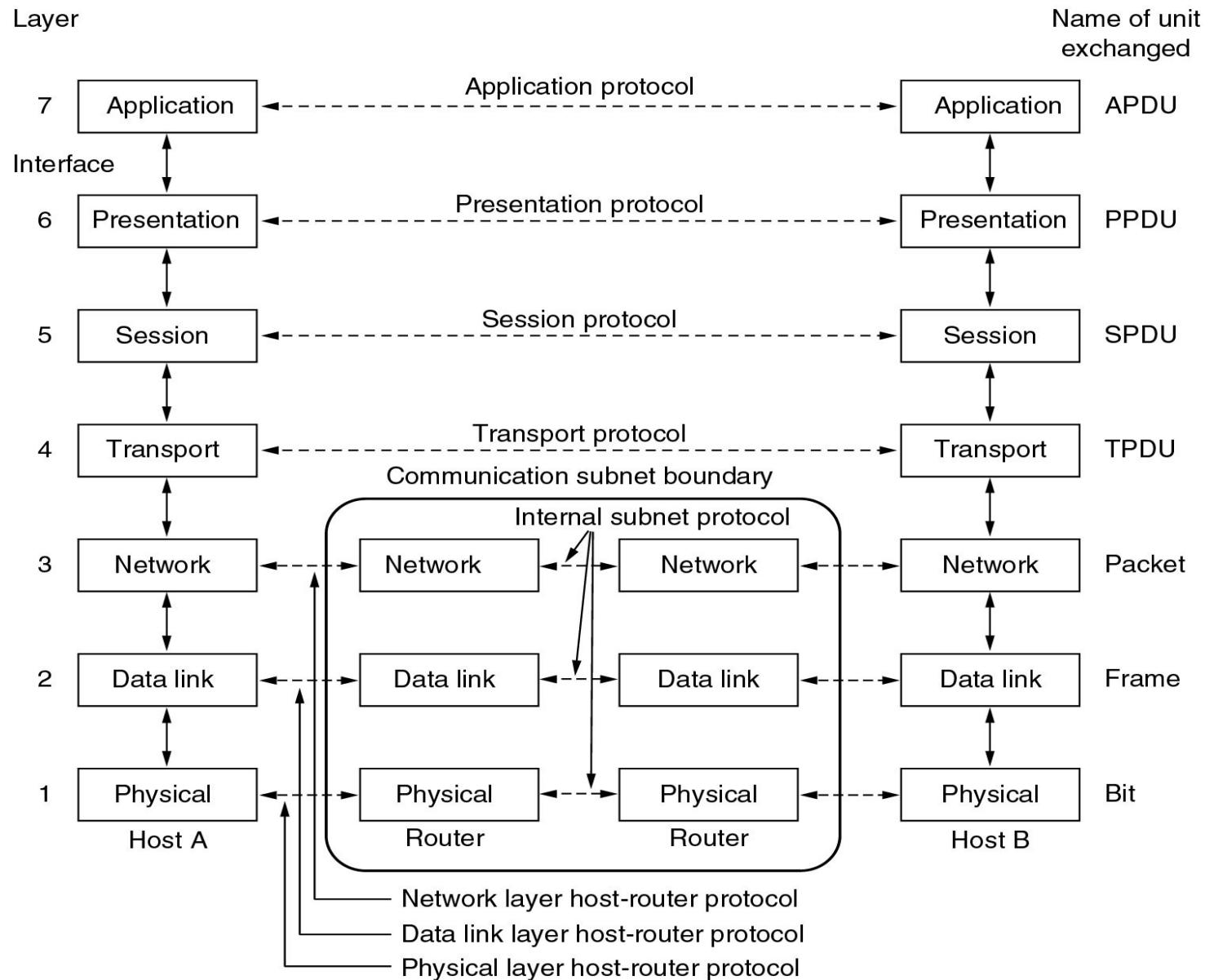
The relationship between a service and a protocol.

Reference Models

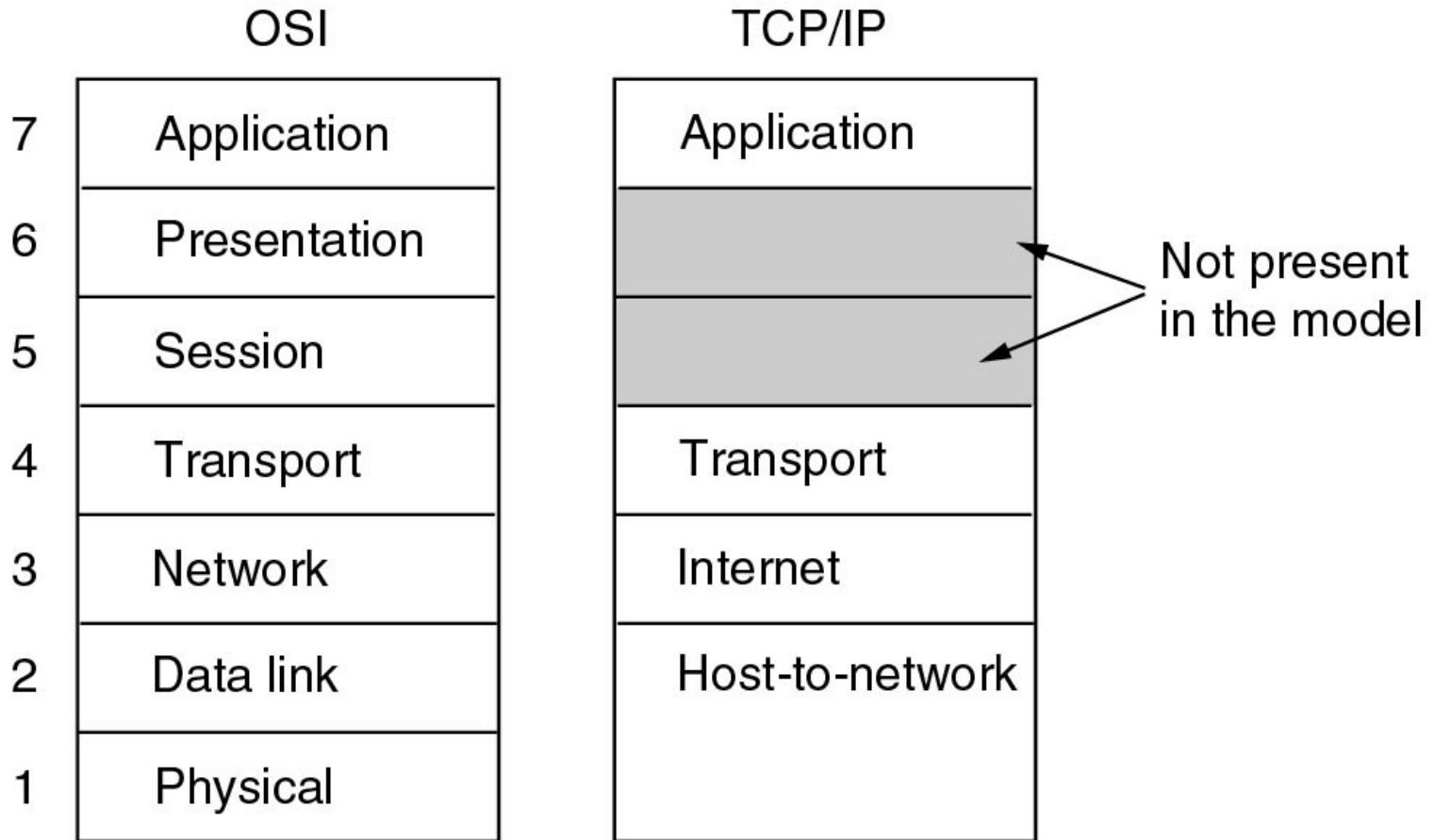
- The OSI Reference Model
- The TCP/IP Reference Model
- A Comparison of OSI and TCP/IP
- A Critique of the OSI Model and Protocols
- A Critique of the TCP/IP Reference Model

Reference Models

The OSI reference model.

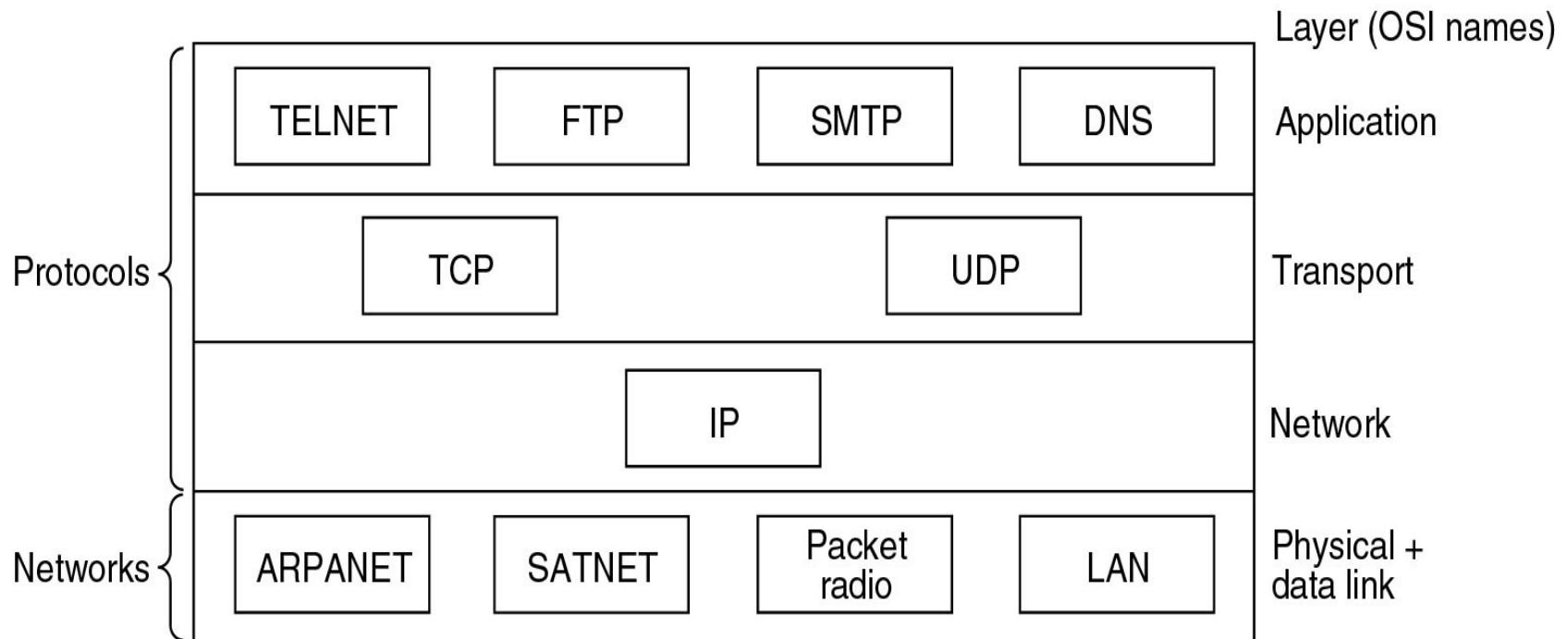


Reference Models (2)



The TCP/IP reference model.

Reference Models (3)



Protocols and networks in the TCP/IP model initially.

Comparing OSI and TCP/IP Models

Concepts central to the OSI model

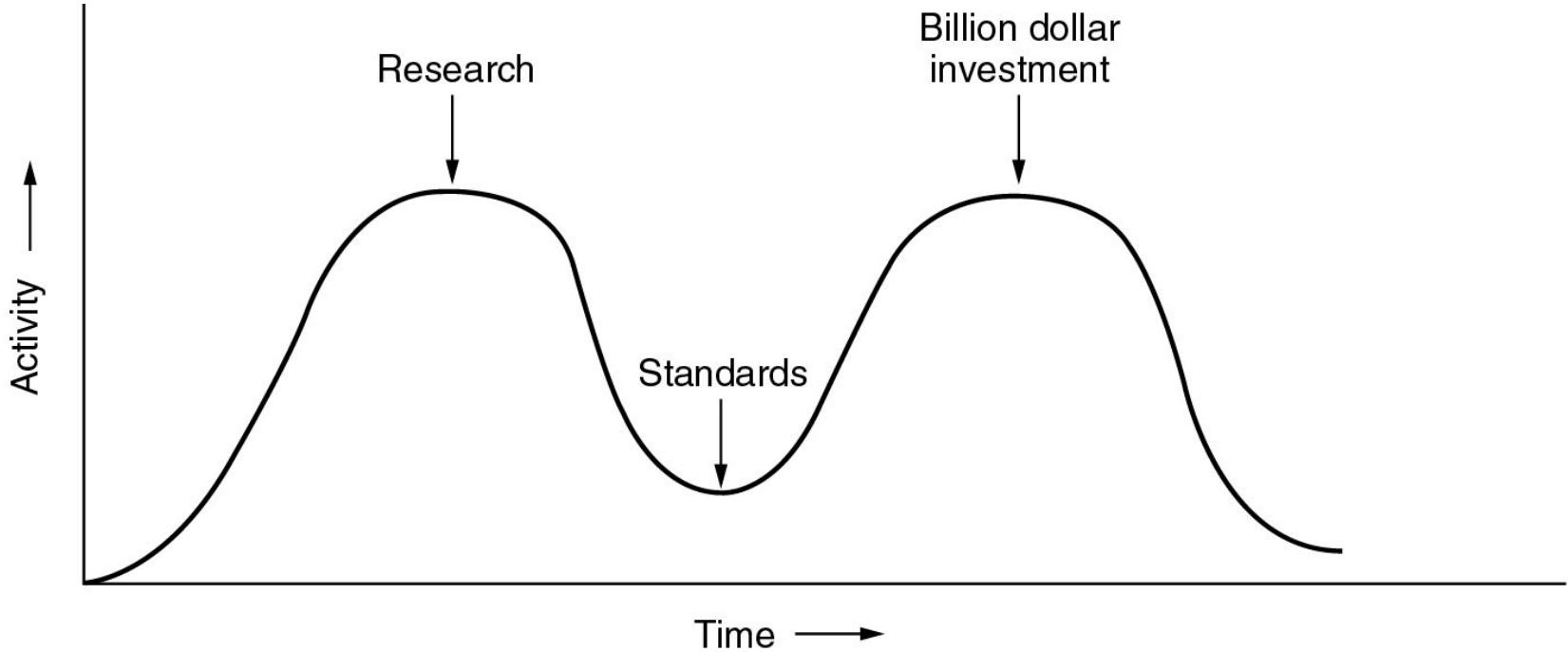
- Services
- Interfaces
- Protocols

A Critique of the OSI Model and Protocols

Why OSI did not take over the world

- Bad timing
- Bad technology
- Bad implementations
- Bad politics

Bad Timing



The apocalypse of the two elephants.

A Critique of the TCP/IP Reference Model

Problems:

- Service, interface, and protocol not distinguished
- Not a general model
- Host-to-network “layer” not really a layer
- No mention of physical and data link layers
- Minor protocols deeply entrenched, hard to replace

Hybrid Model

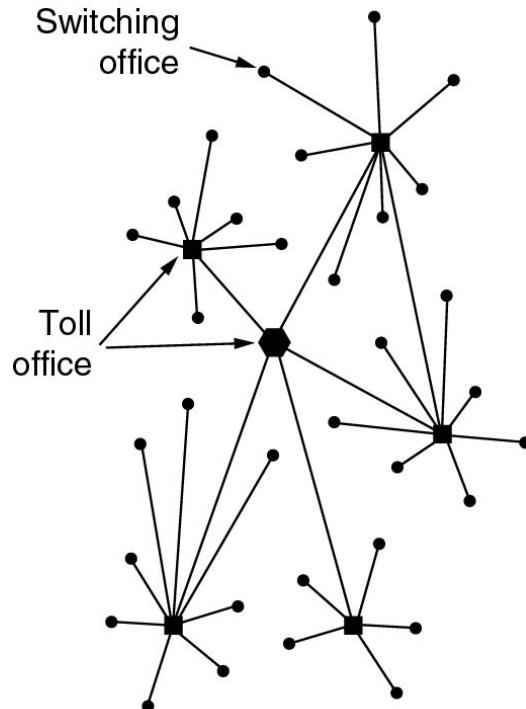
5	Application layer
4	Transport layer
3	Network layer
2	Data link layer
1	Physical layer

The hybrid reference model to be used in this book.

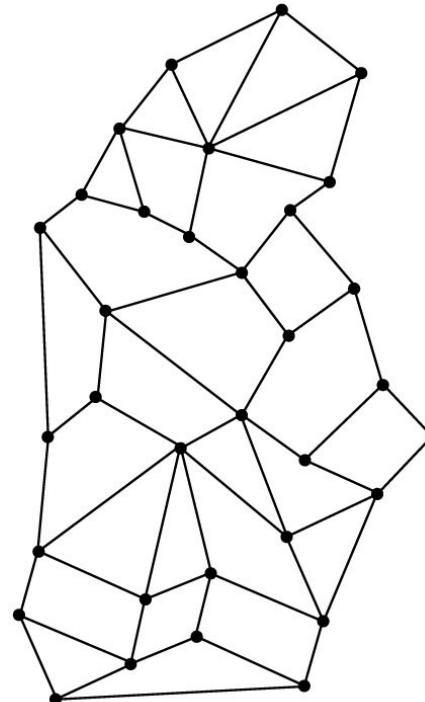
Example Networks

- The Internet
- Connection-Oriented Networks:
X.25, Frame Relay, and ATM
- Ethernet
- Wireless LANs: 802:11

The ARPANET



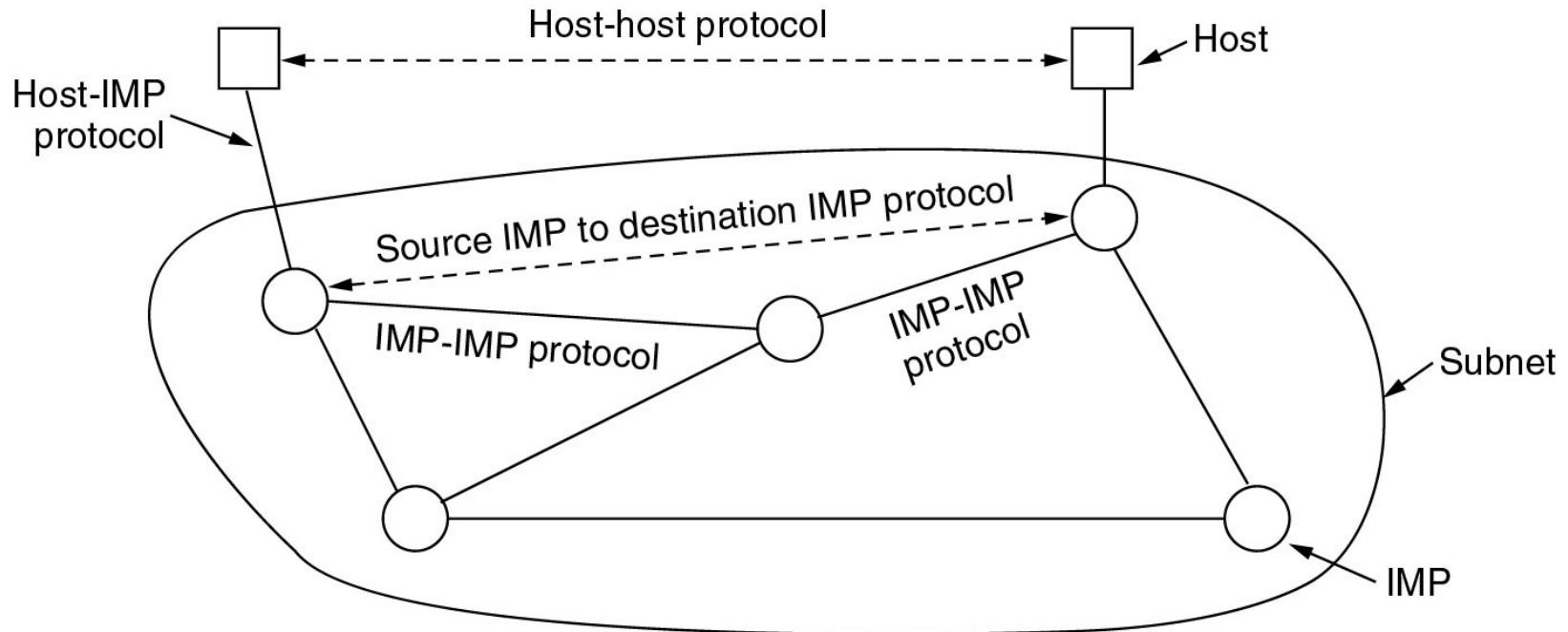
(a)



(b)

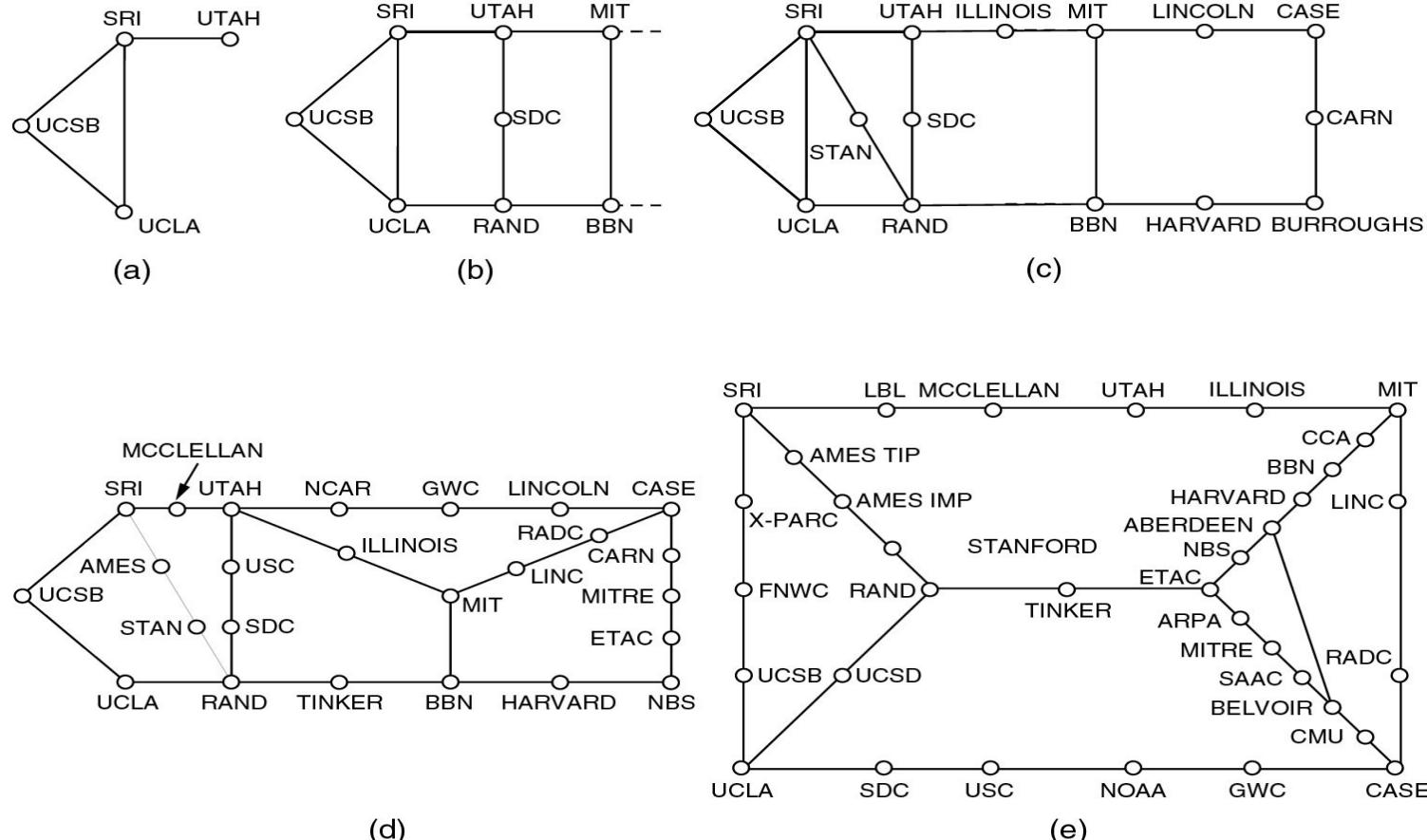
- (a) Structure of the telephone system.
- (b) Baran's proposed distributed switching system.

The ARPANET (2)



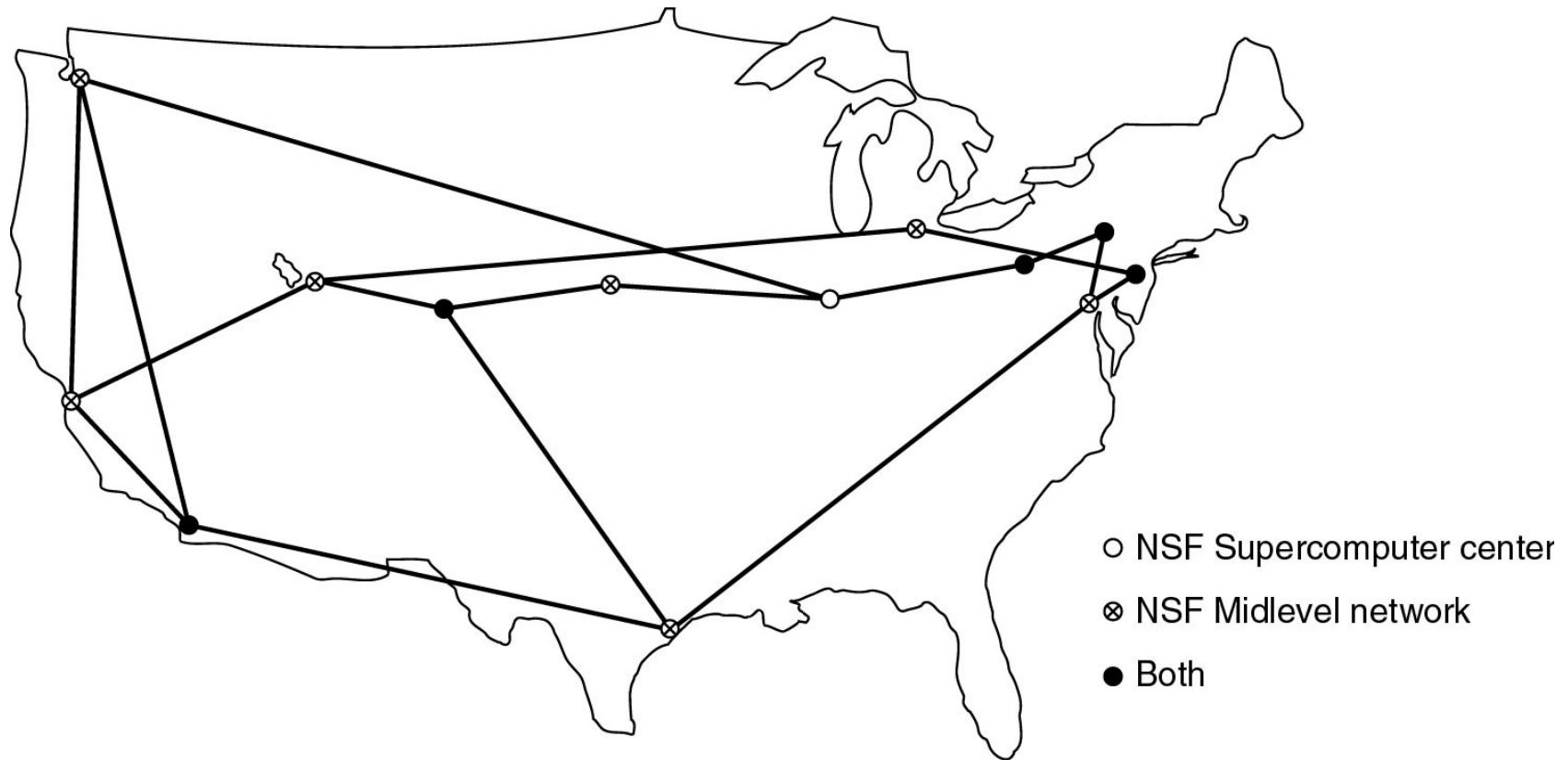
The original ARPANET design.

The ARPANET (3)



Growth of the ARPANET **(a)** December 1969. **(b)** July 1970.
(c) March 1971. **(d)** April 1972. **(e)** September 1972.

NSFNET



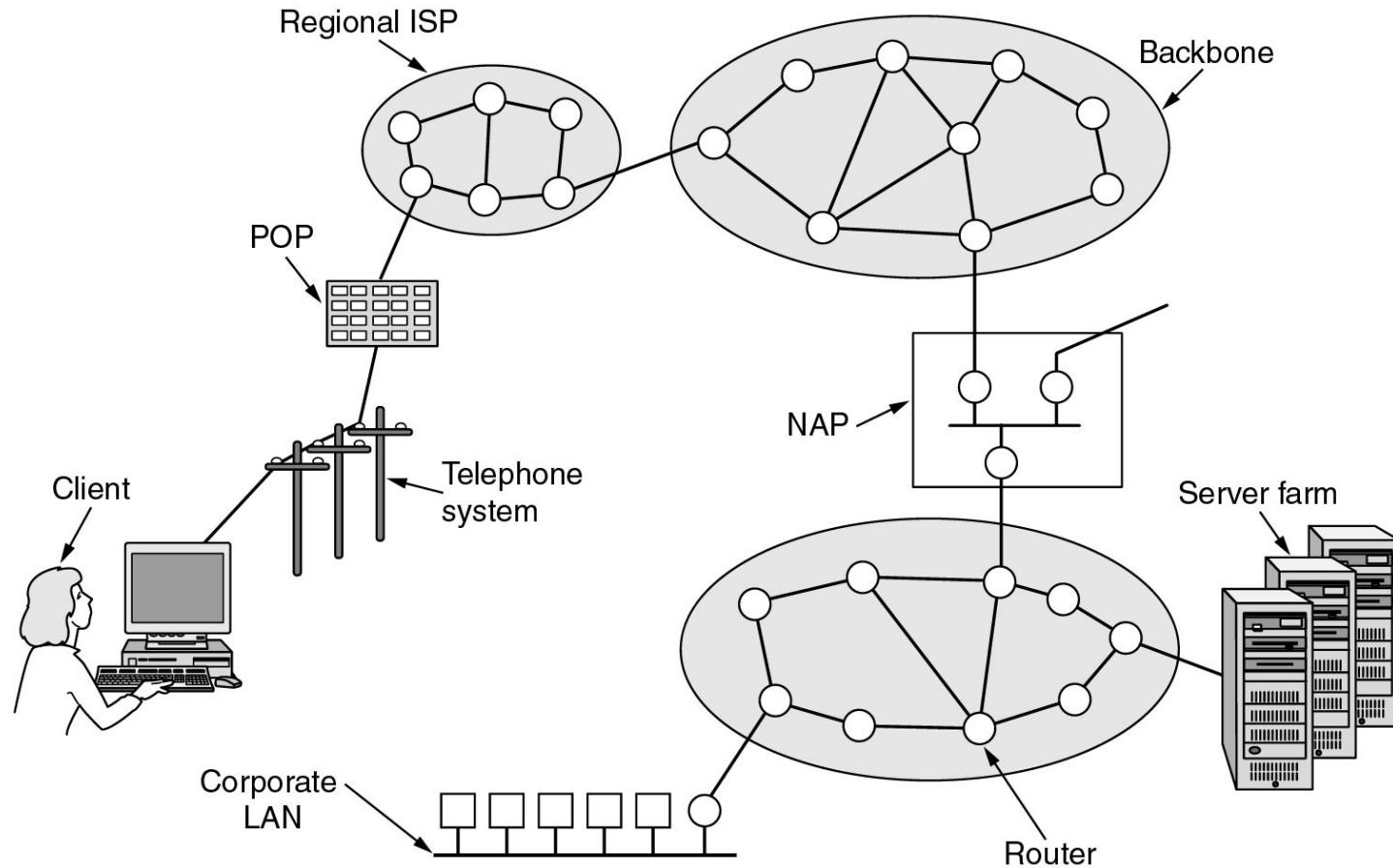
The NSFNET backbone in 1988.

Internet Usage

Traditional applications (1970 – 1990)

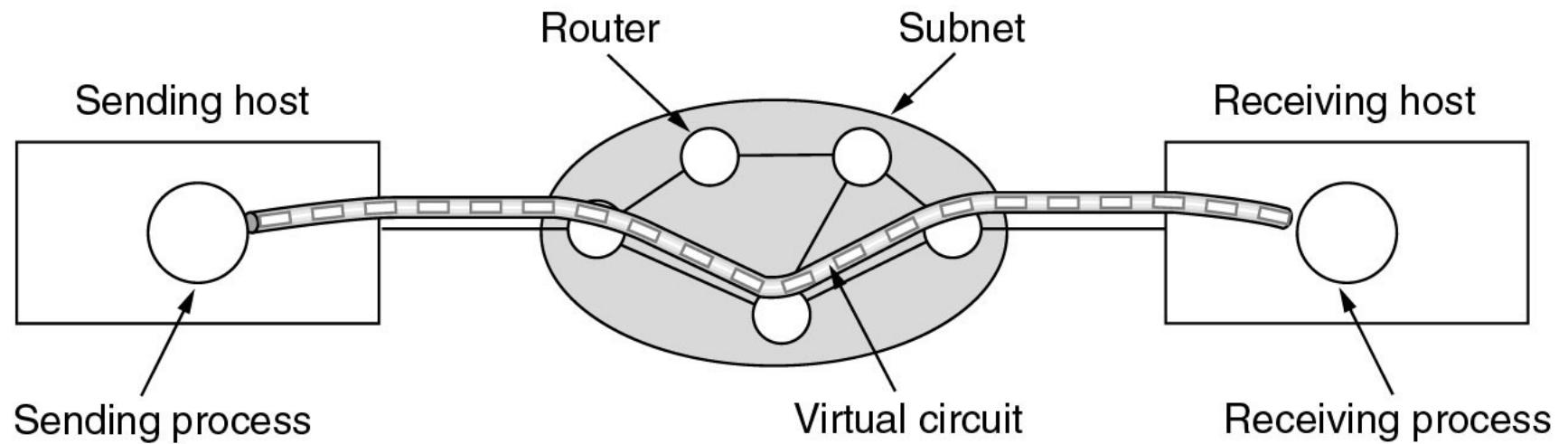
- E-mail
- News
- Remote login
- File transfer

Architecture of the Internet



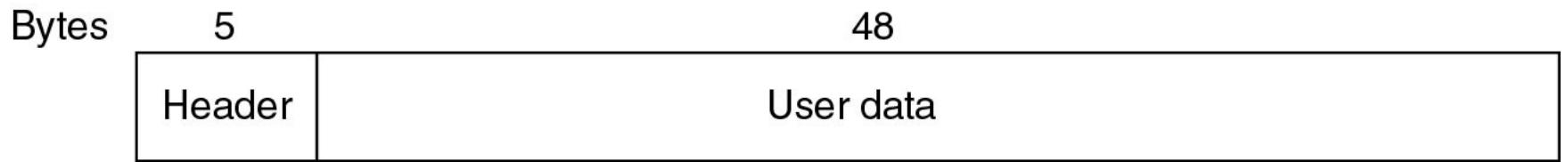
Overview of the Internet.

ATM Virtual Circuits



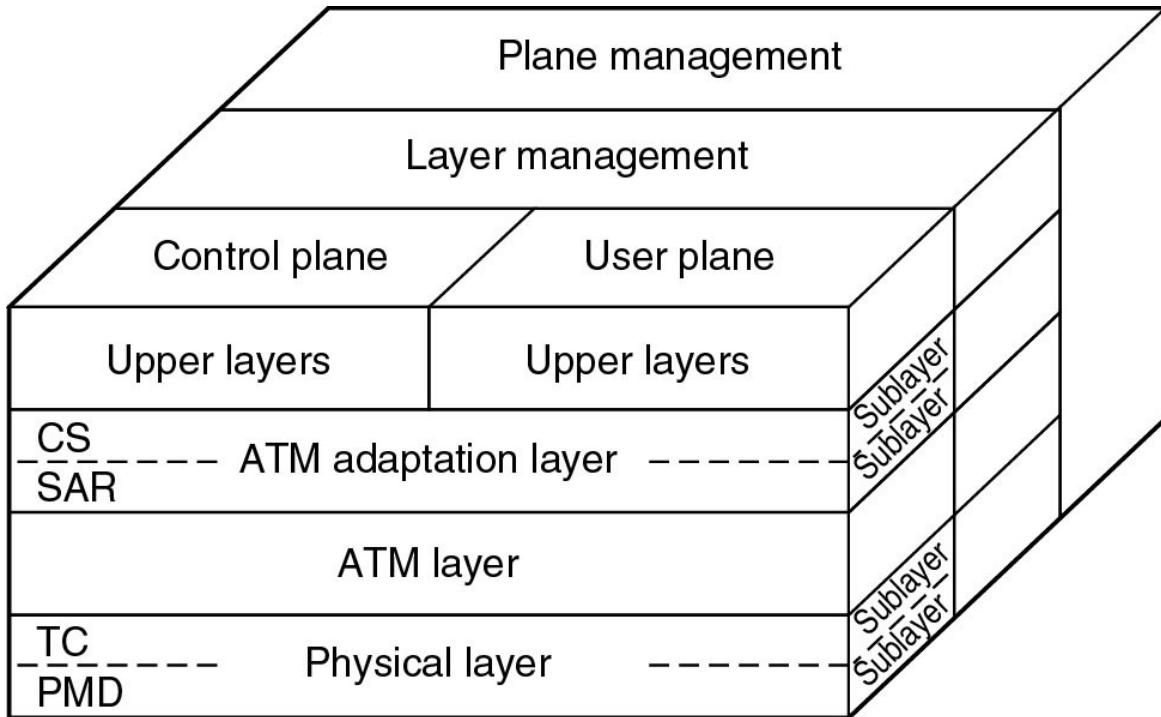
A virtual circuit.

ATM Virtual Circuits (2)



An ATM cell.

The ATM Reference Model



CS: Convergence sublayer
SAR: Segmentation and reassembly sublayer
TC: Transmission convergence sublayer
PMD: Physical medium dependent sublayer

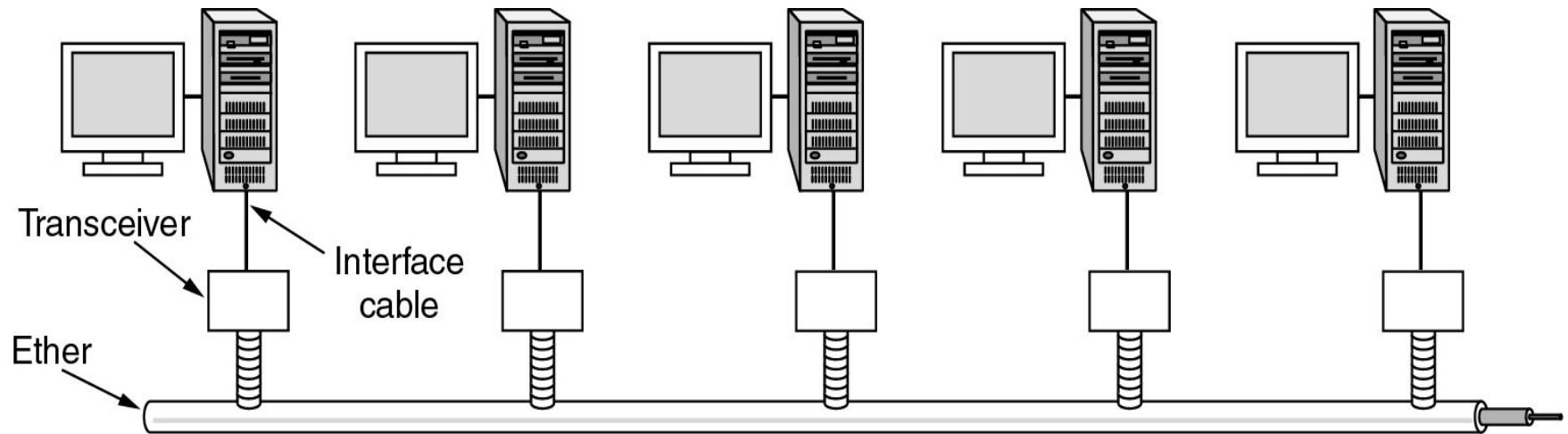
The ATM reference model.

The ATM Reference Model (2)

OSI layer	ATM layer	ATM sublayer	Functionality
3/4	AAL	CS	Providing the standard interface (convergence)
		SAR	Segmentation and reassembly
2/3	ATM		Flow control Cell header generation/extraction Virtual circuit/path management Cell multiplexing/demultiplexing
2	Physical	TC	Cell rate decoupling Header checksum generation and verification Cell generation Packing/unpacking cells from the enclosing envelope Frame generation
			Bit timing Physical network access

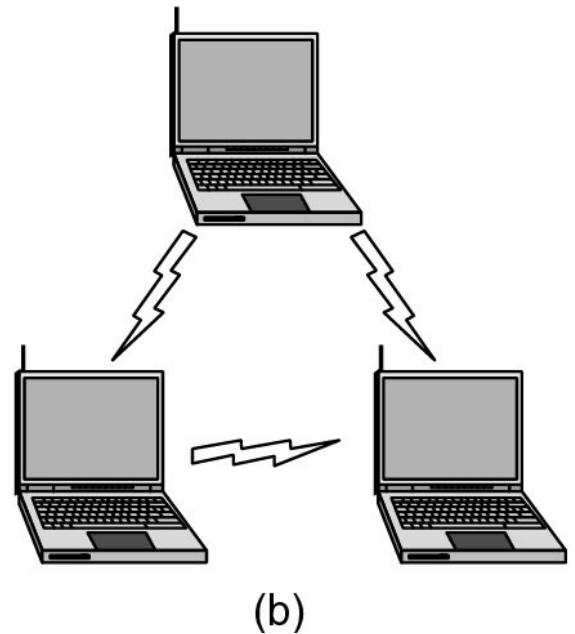
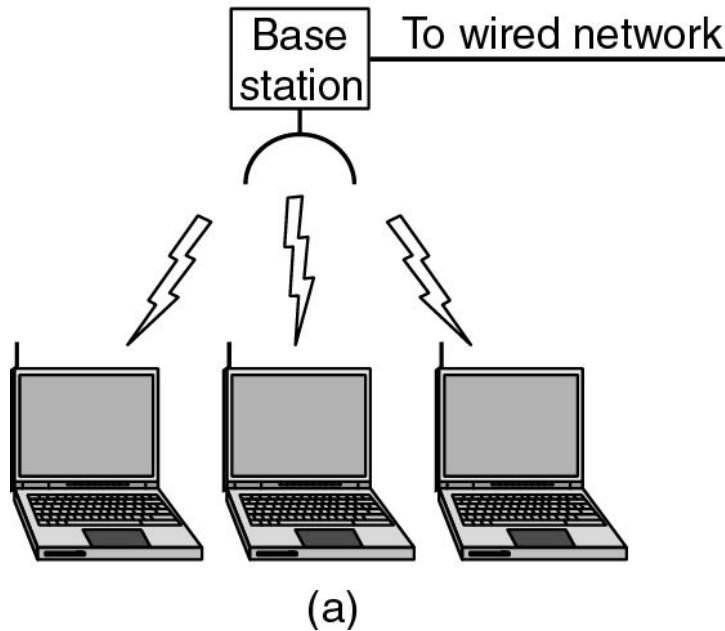
The ATM layers and sublayers and their functions.

Ethernet



Architecture of the original Ethernet.

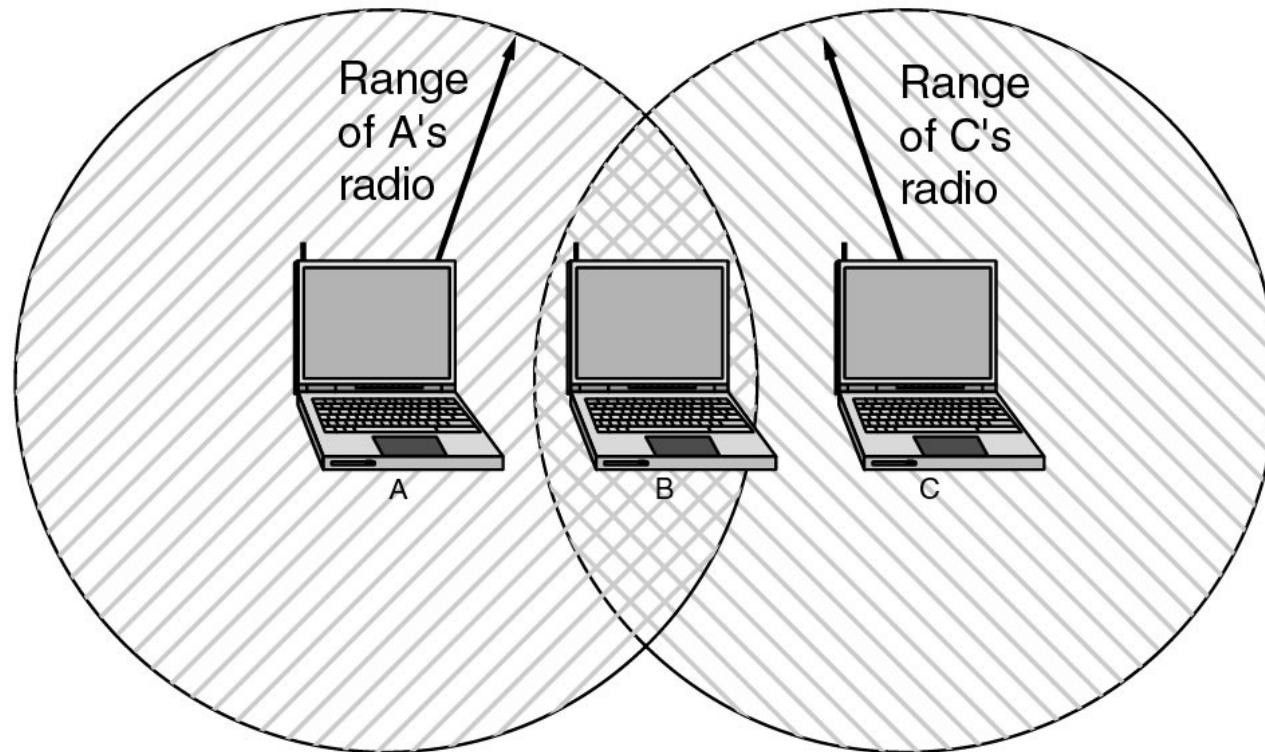
Wireless LANs



(a) Wireless networking with a base station.

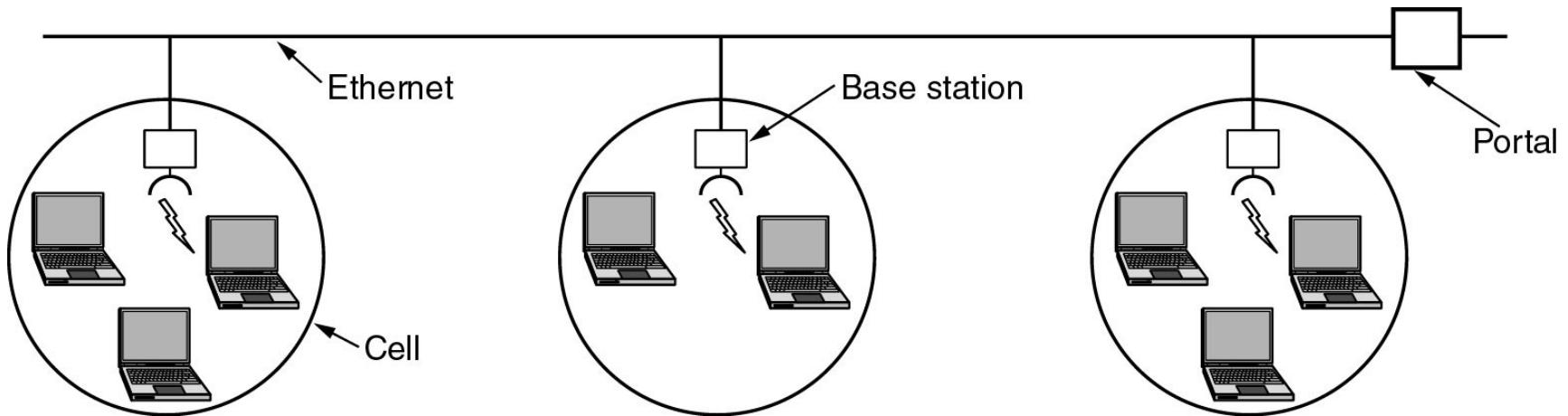
(b) Ad hoc networking.

Wireless LANs (2)



The range of a single radio may not cover the entire system.

Wireless LANs (3)



A multicell 802.11 network.

Network Standardization

- Who's Who in the Telecommunications World
- Who's Who in the International Standards World
- Who's Who in the Internet Standards World

ITU

- Main sectors
 - Radiocommunications
 - Telecommunications Standardization
 - Development
- Classes of Members
 - National governments
 - Sector members
 - Associate members
 - Regulatory agencies

IEEE 802 Standards

Number	Topic
802.1	Overview and architecture of LANs
802.2 ↓	Logical link control
802.3 *	Ethernet
802.4 ↓	Token bus (was briefly used in manufacturing plants)
802.5	Token ring (IBM's entry into the LAN world)
802.6 ↓	Dual queue dual bus (early metropolitan area network)
802.7 ↓	Technical advisory group on broadband technologies
802.8 †	Technical advisory group on fiber optic technologies
802.9 ↓	Isochronous LANs (for real-time applications)
802.10 ↓	Virtual LANs and security
802.11 *	Wireless LANs
802.12 ↓	Demand priority (Hewlett-Packard's AnyLAN)
802.13	Unlucky number. Nobody wanted it
802.14 ↓	Cable modems (defunct: an industry consortium got there first)
802.15 *	Personal area networks (Bluetooth)
802.16 *	Broadband wireless
802.17	Resilient packet ring

The 802 working groups. The important ones are marked with *. The ones marked with ↓ are hibernating. The one marked with † gave up.

Metric Units

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^3	1,000	Kilo
10^{-6}	0.000001	micro	10^6	1,000,000	Mega
10^{-9}	0.000000001	nano	10^9	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.0000000000000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000,000	Yotta

The principal metric prefixes.



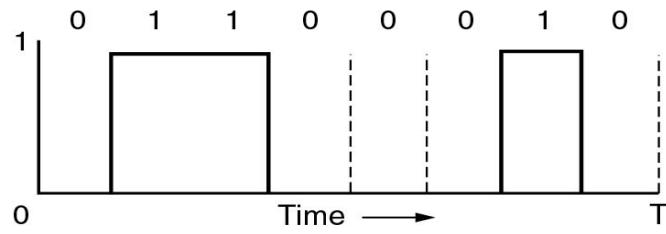
Chapter 2

The Physical Layer

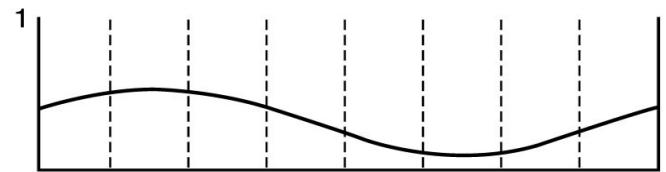
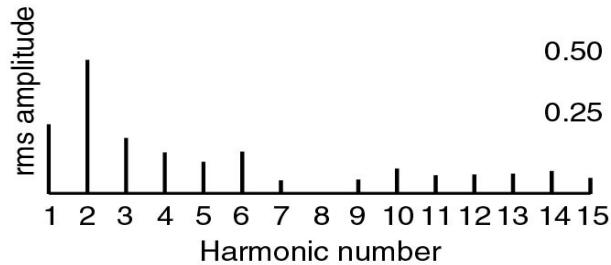
The Theoretical Basis for Data Communication

- Fourier Analysis
- Bandwidth-Limited Signals
- Maximum Data Rate of a Channel

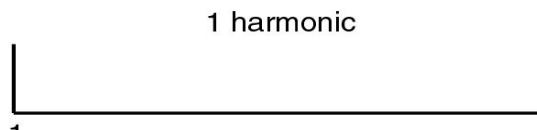
Bandwidth-Limited Signals



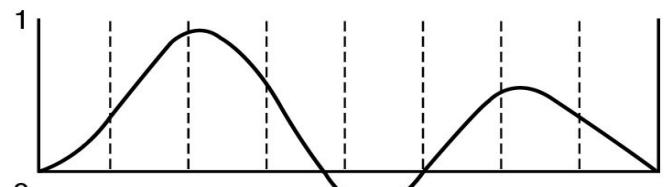
(a)



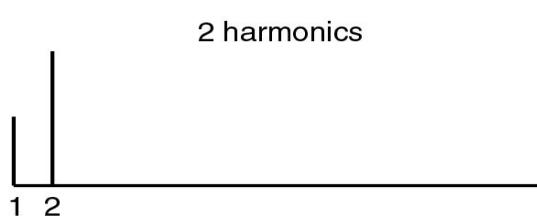
(b)



1 harmonic



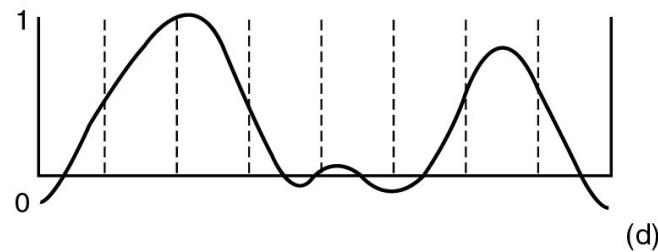
(c)



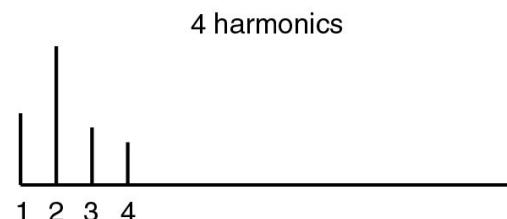
2 harmonics

A binary signal and its root-mean-square Fourier amplitudes.
(b) – (c) Successive approximations to the original signal.

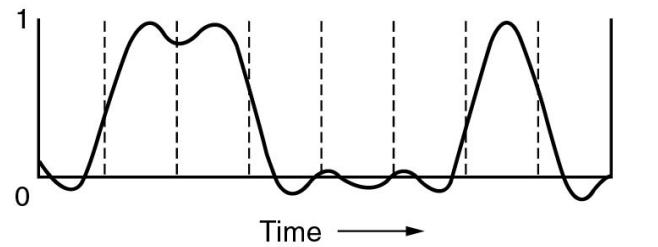
Bandwidth-Limited Signals (2)



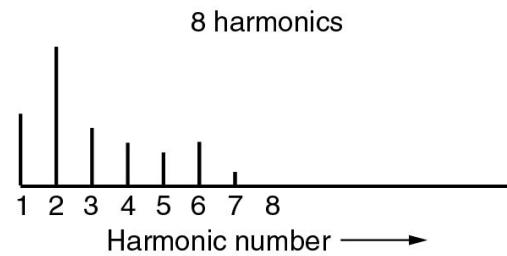
(d)



4 harmonics



(e)



8 harmonics

(d) – (e) Successive approximations to the original signal.

Bandwidth-Limited Signals (3)

Bps	T (msec)	First harmonic (Hz)	# Harmonics sent
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

Relation between data rate and harmonics.

Guided Transmission Data

- Magnetic Media
- Twisted Pair
- Coaxial Cable
- Fiber Optics

Twisted Pair



(a)

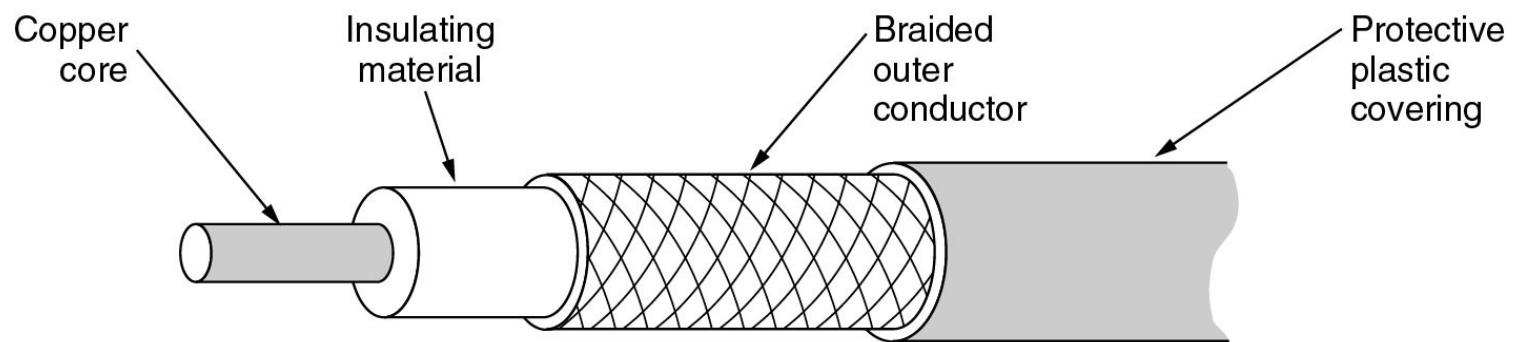


(b)

(a) Category 3 UTP.

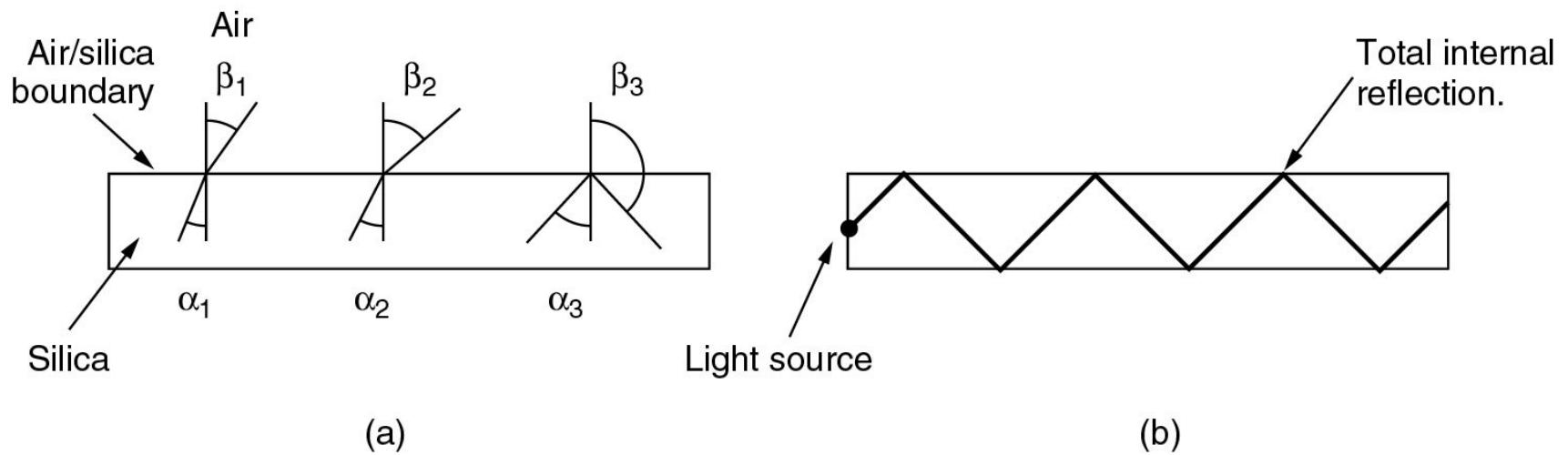
(b) Category 5 UTP.

Coaxial Cable



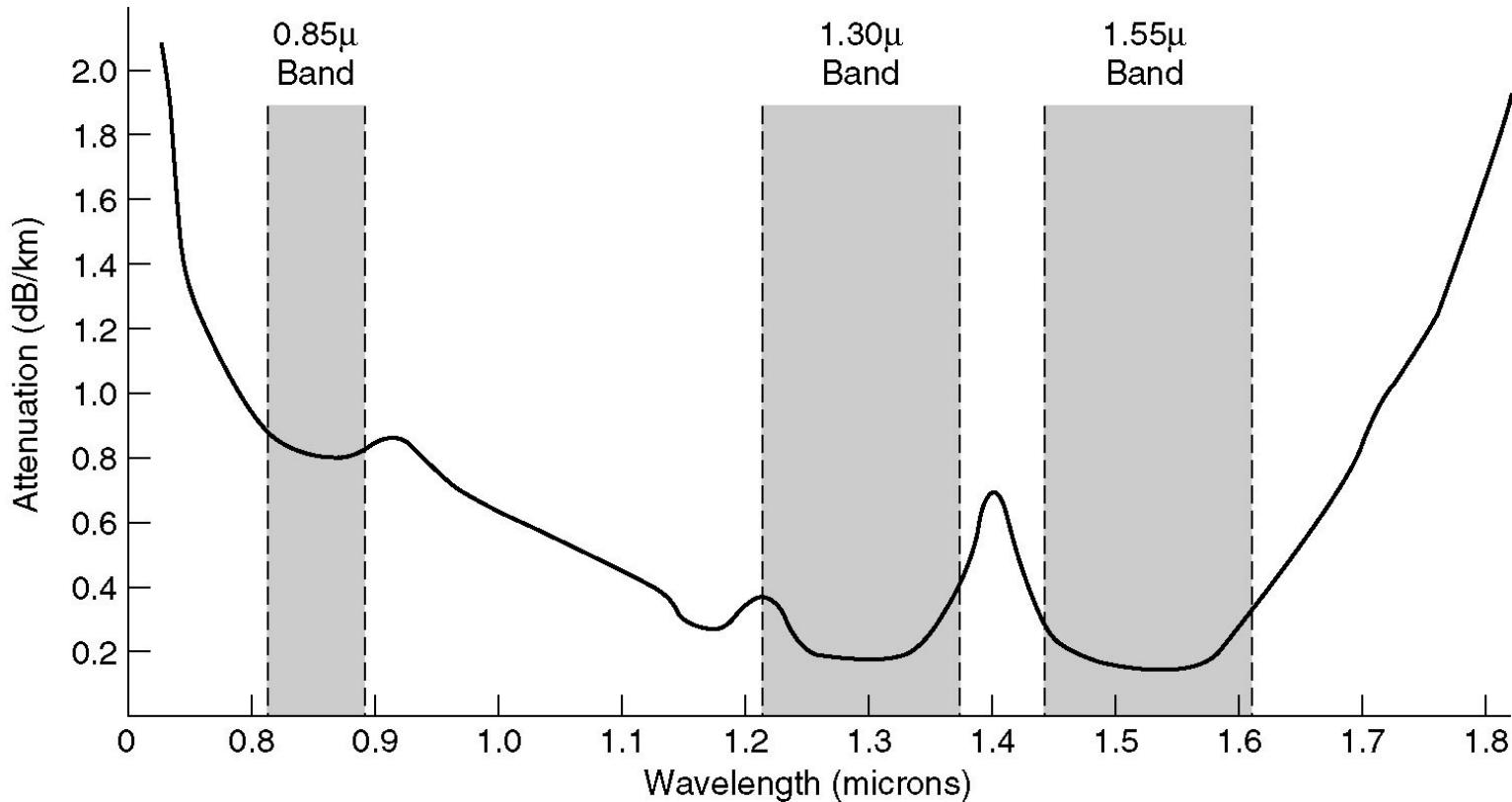
A coaxial cable.

Fiber Optics



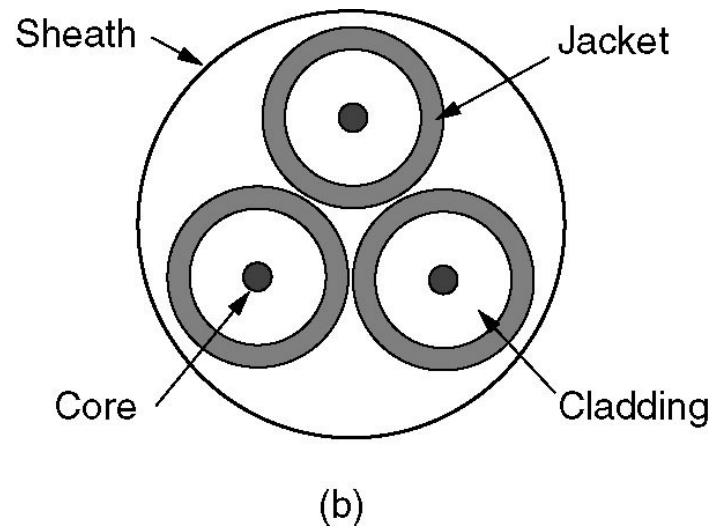
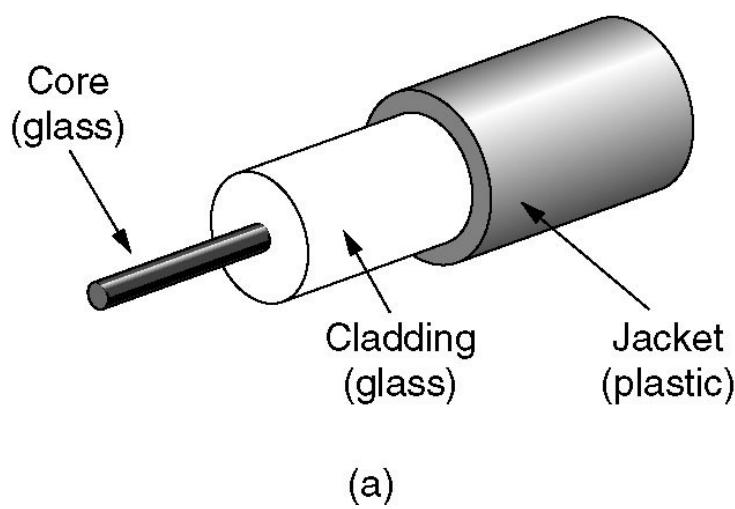
- (a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles.
- (b) Light trapped by total internal reflection.

Transmission of Light through Fiber



Attenuation of light through fiber in the infrared region.

Fiber Cables



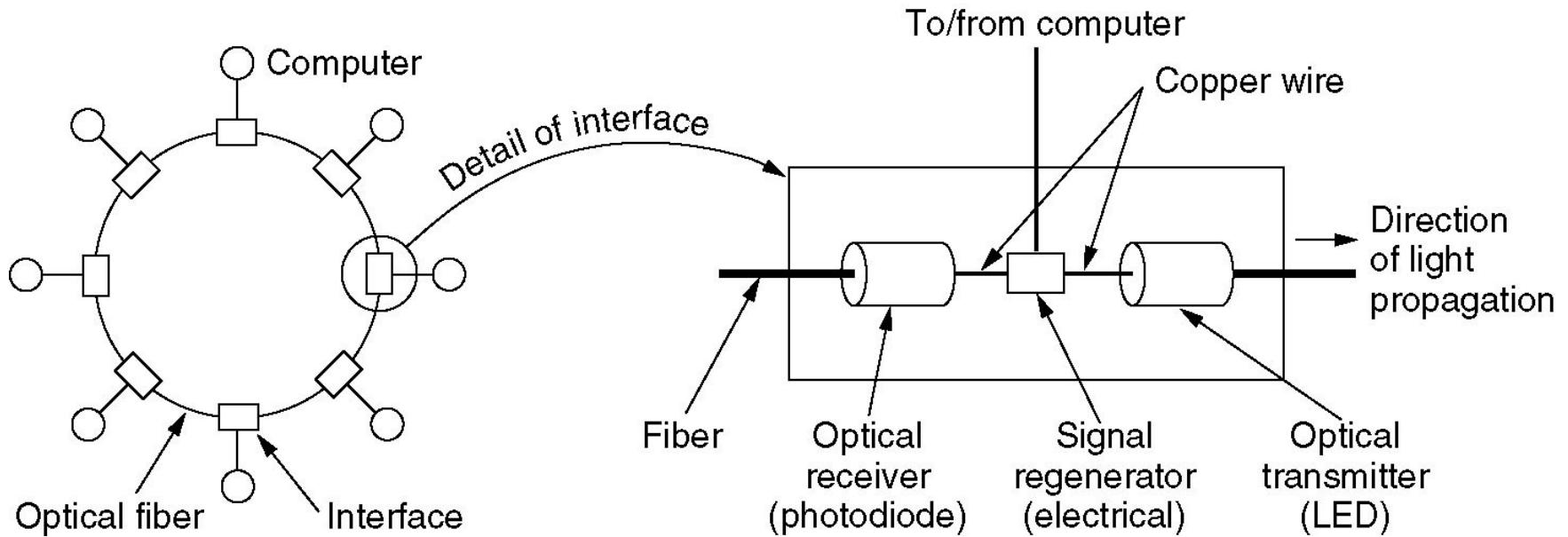
- (a) Side view of a single fiber.
- (b) End view of a sheath with three fibers.

Fiber Cables (2)

Item	LED	Semiconductor laser
Data rate	Low	High
Fiber type	Multimode	Multimode or single mode
Distance	Short	Long
Lifetime	Long life	Short life
Temperature sensitivity	Minor	Substantial
Cost	Low cost	Expensive

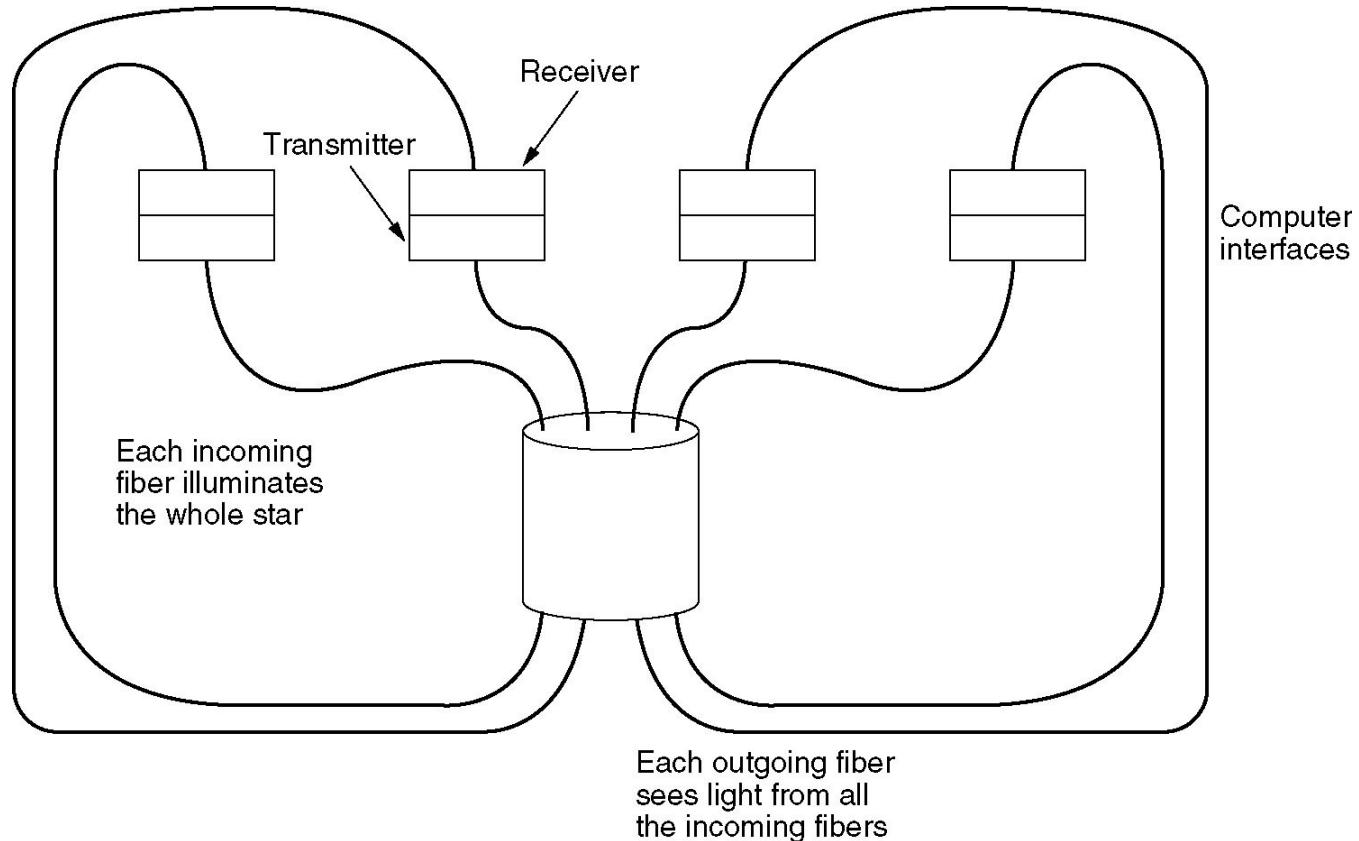
A comparison of semiconductor diodes and LEDs as light sources.

Fiber Optic Networks



A fiber optic ring with active repeaters.

Fiber Optic Networks (2)

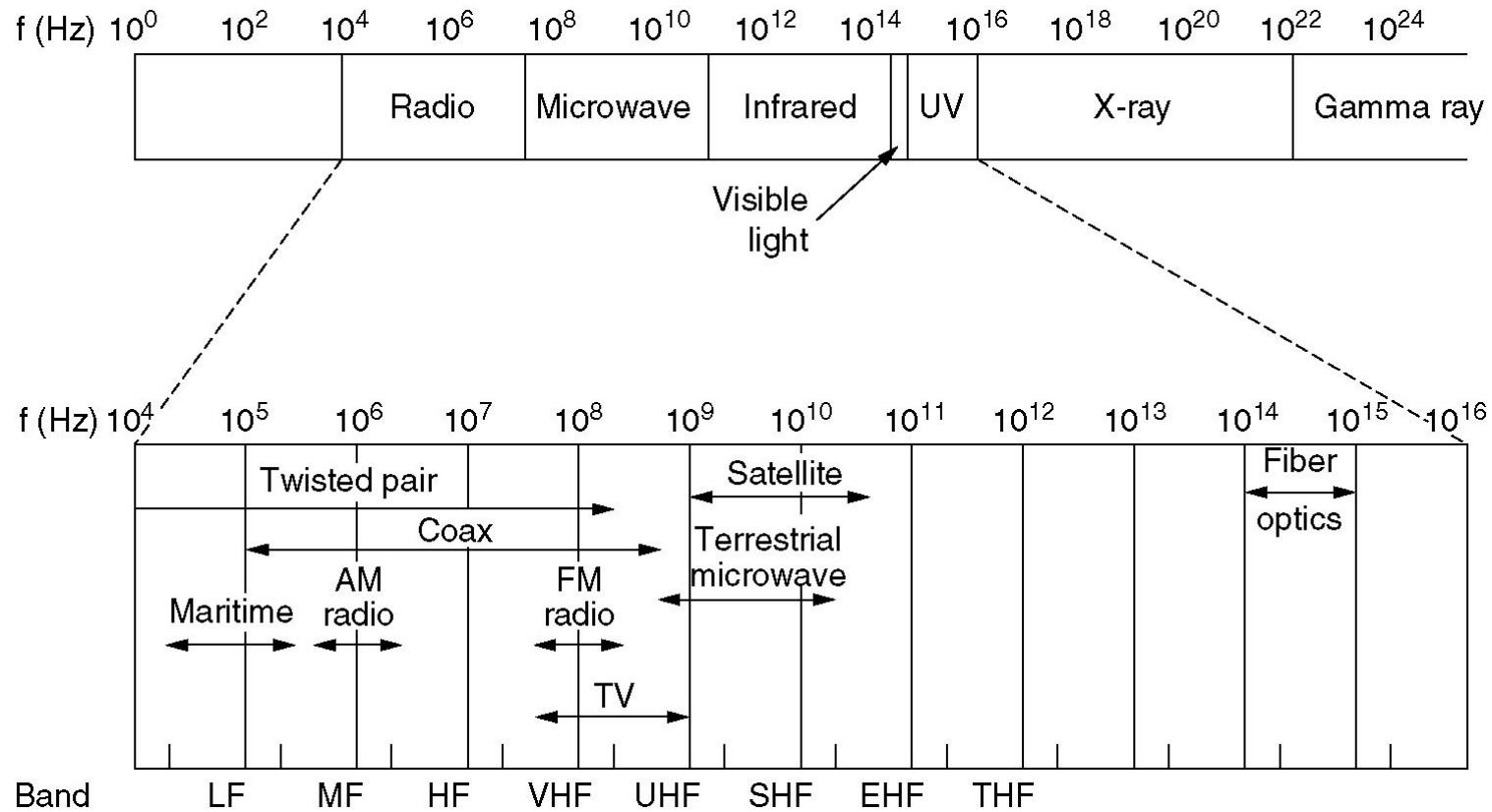


A passive star connection in a fiber optics network.

Wireless Transmission

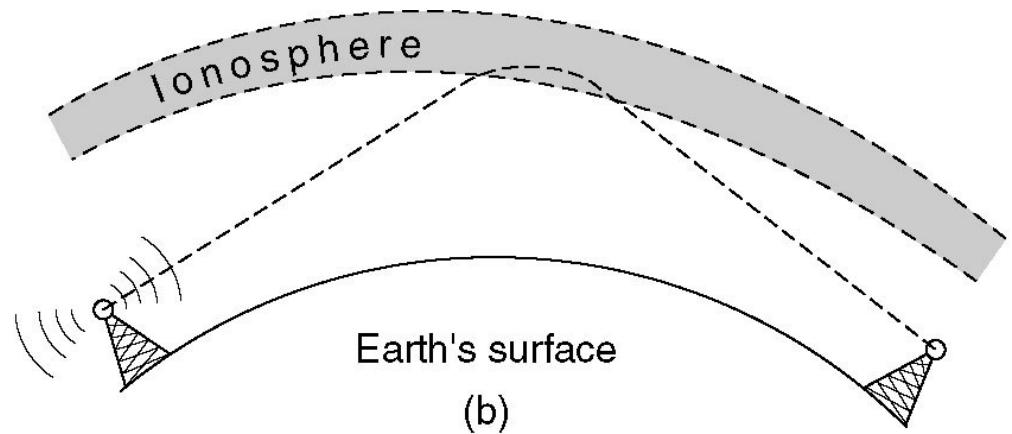
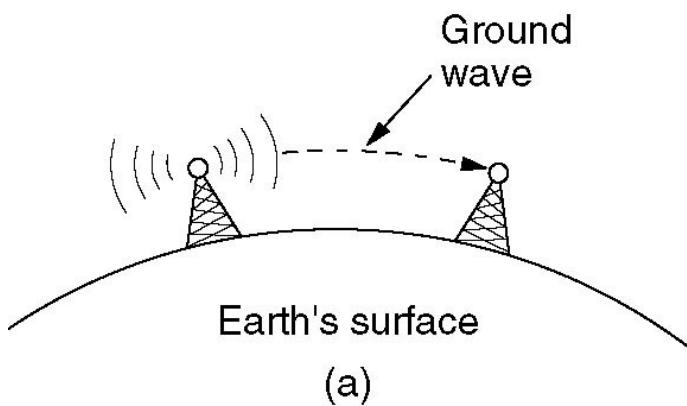
- The Electromagnetic Spectrum
- Radio Transmission
- Microwave Transmission
- Infrared and Millimeter Waves
- Lightwave Transmission

The Electromagnetic Spectrum



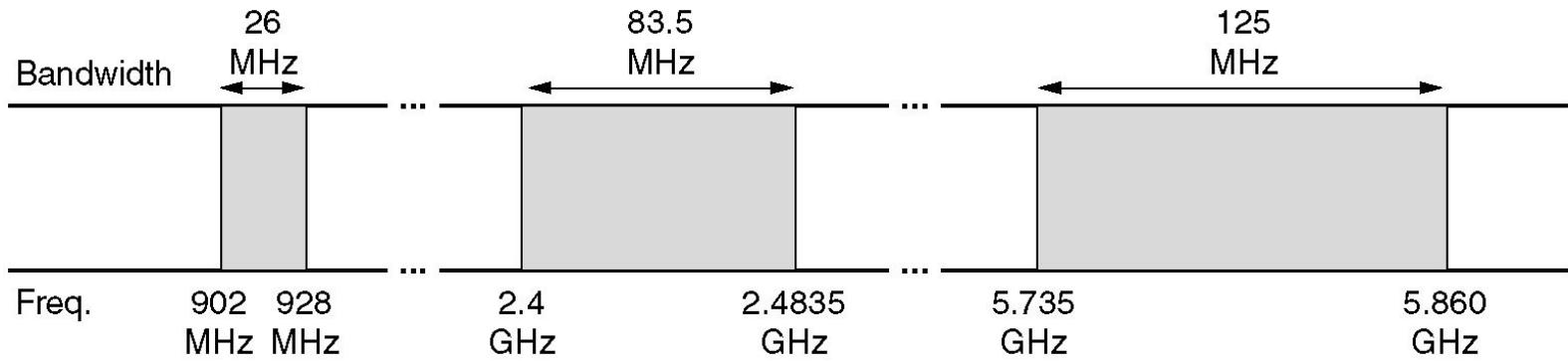
The electromagnetic spectrum and its uses for communication.

Radio Transmission



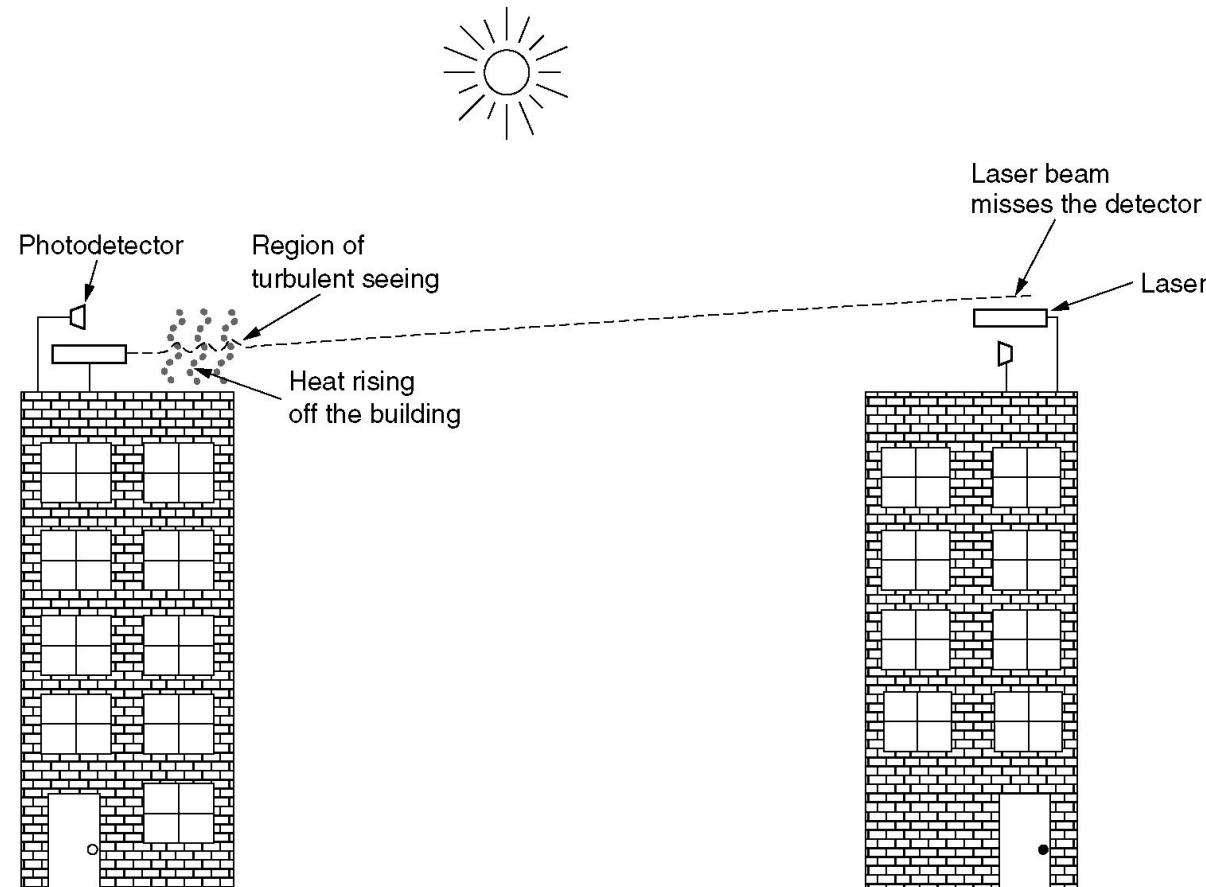
- (a) In the VLF, LF, and MF bands, radio waves follow the curvature of the earth.
- (b) In the HF band, they bounce off the ionosphere.

Politics of the Electromagnetic Spectrum



The ISM bands in the United States.

Lightwave Transmission

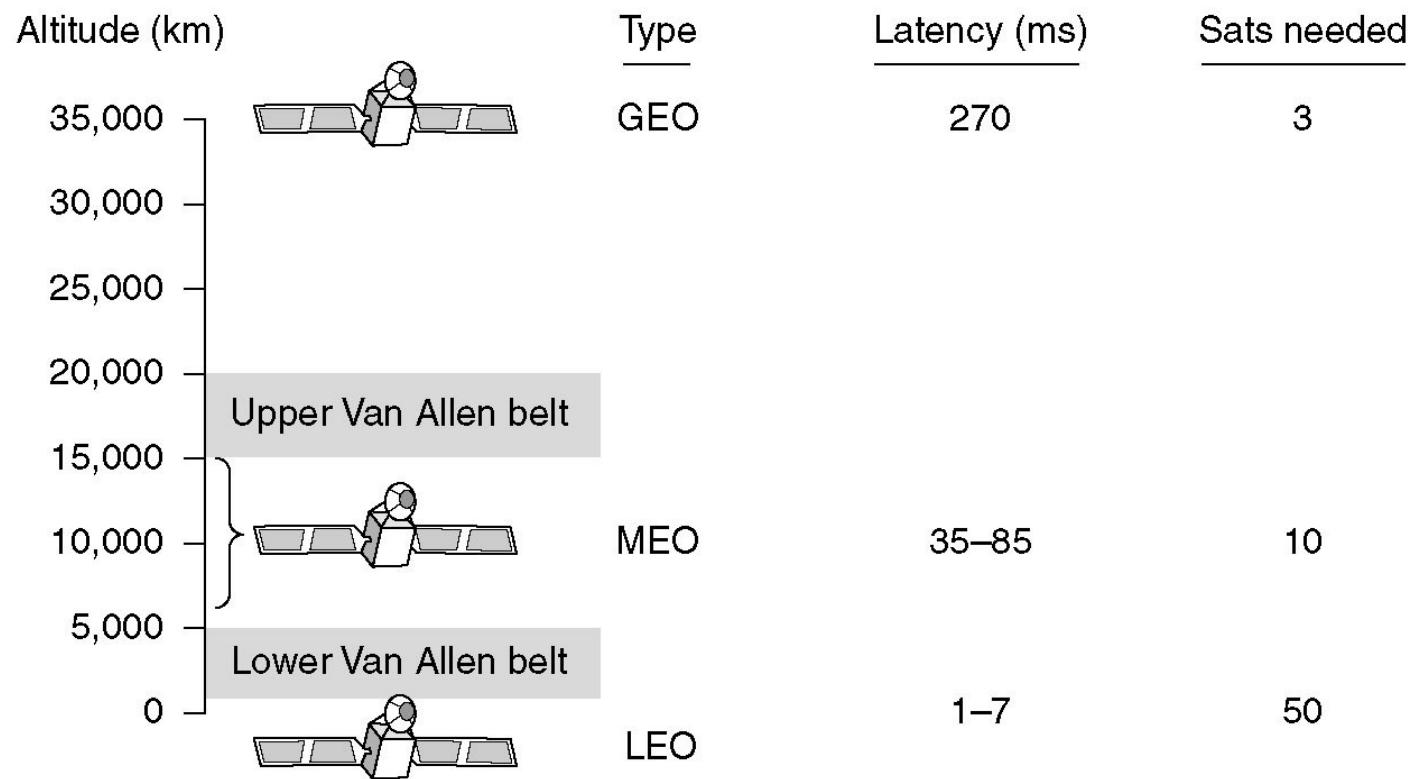


Convection currents can interfere with laser communication systems.
A bidirectional system with two lasers is pictured here.

Communication Satellites

- Geostationary Satellites
- Medium-Earth Orbit Satellites
- Low-Earth Orbit Satellites
- Satellites versus Fiber

Communication Satellites



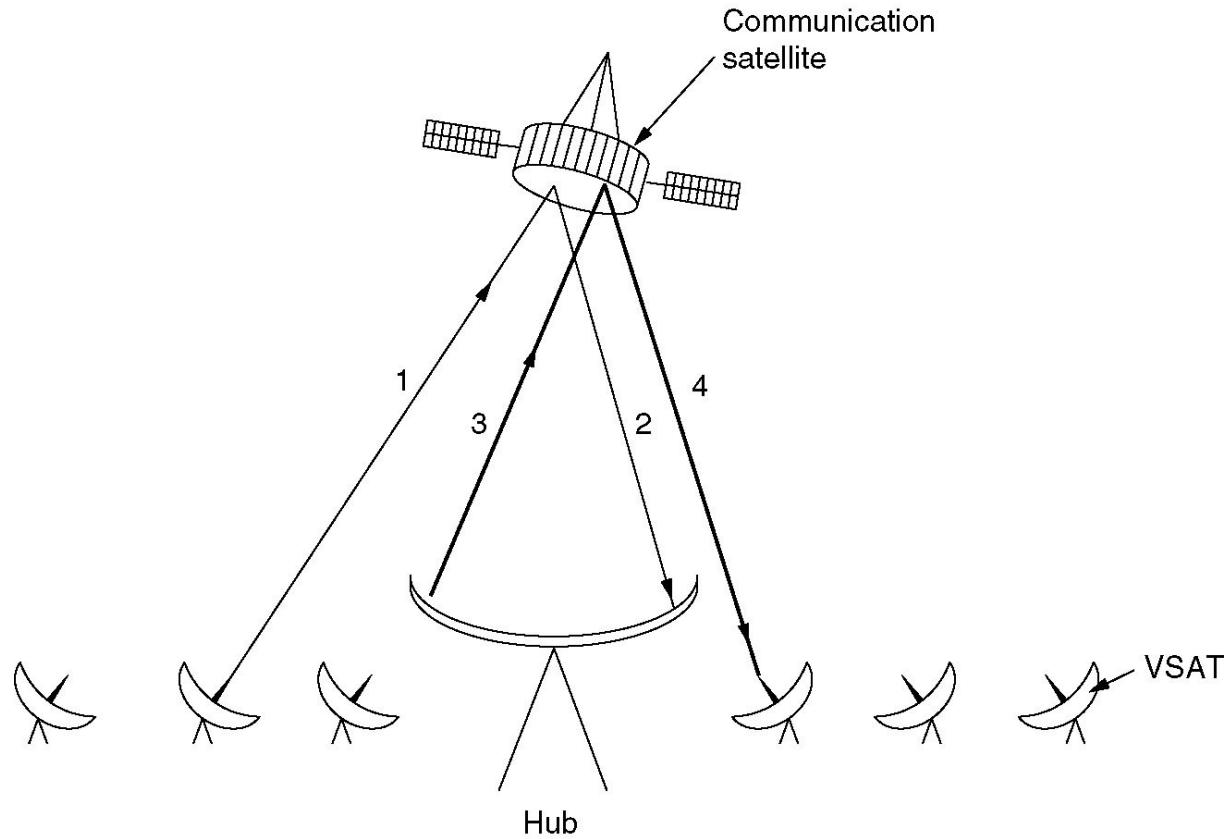
Communication satellites and some of their properties, including altitude above the earth, round-trip delay time and number of satellites needed for global coverage.

Communication Satellites (2)

Band	Downlink	Uplink	Bandwidth	Problems
L	1.5 GHz	1.6 GHz	15 MHz	Low bandwidth; crowded
S	1.9 GHz	2.2 GHz	70 MHz	Low bandwidth; crowded
C	4.0 GHz	6.0 GHz	500 MHz	Terrestrial interference
Ku	11 GHz	14 GHz	500 MHz	Rain
Ka	20 GHz	30 GHz	3500 MHz	Rain, equipment cost

The principal satellite bands.

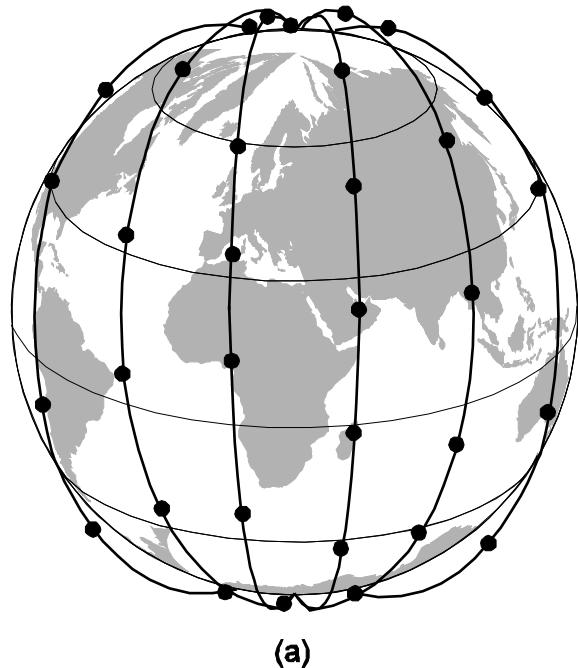
Communication Satellites (3)



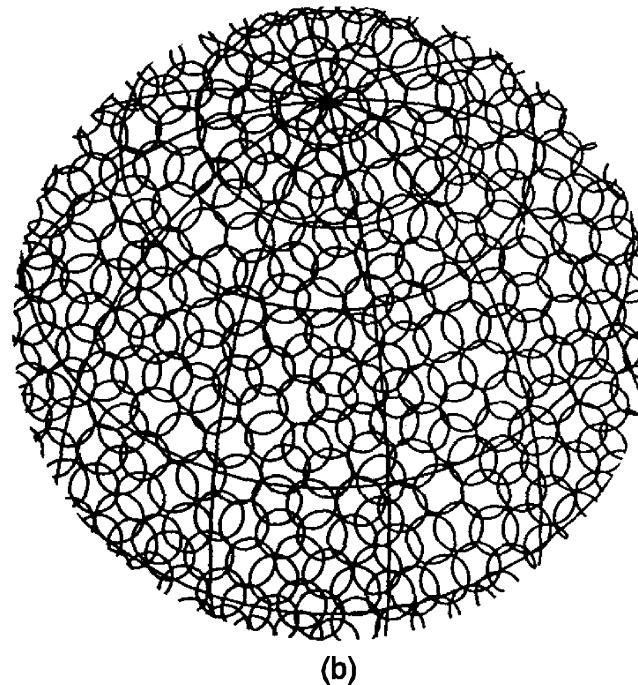
VSATs using a hub.

Low-Earth Orbit Satellites

Iridium



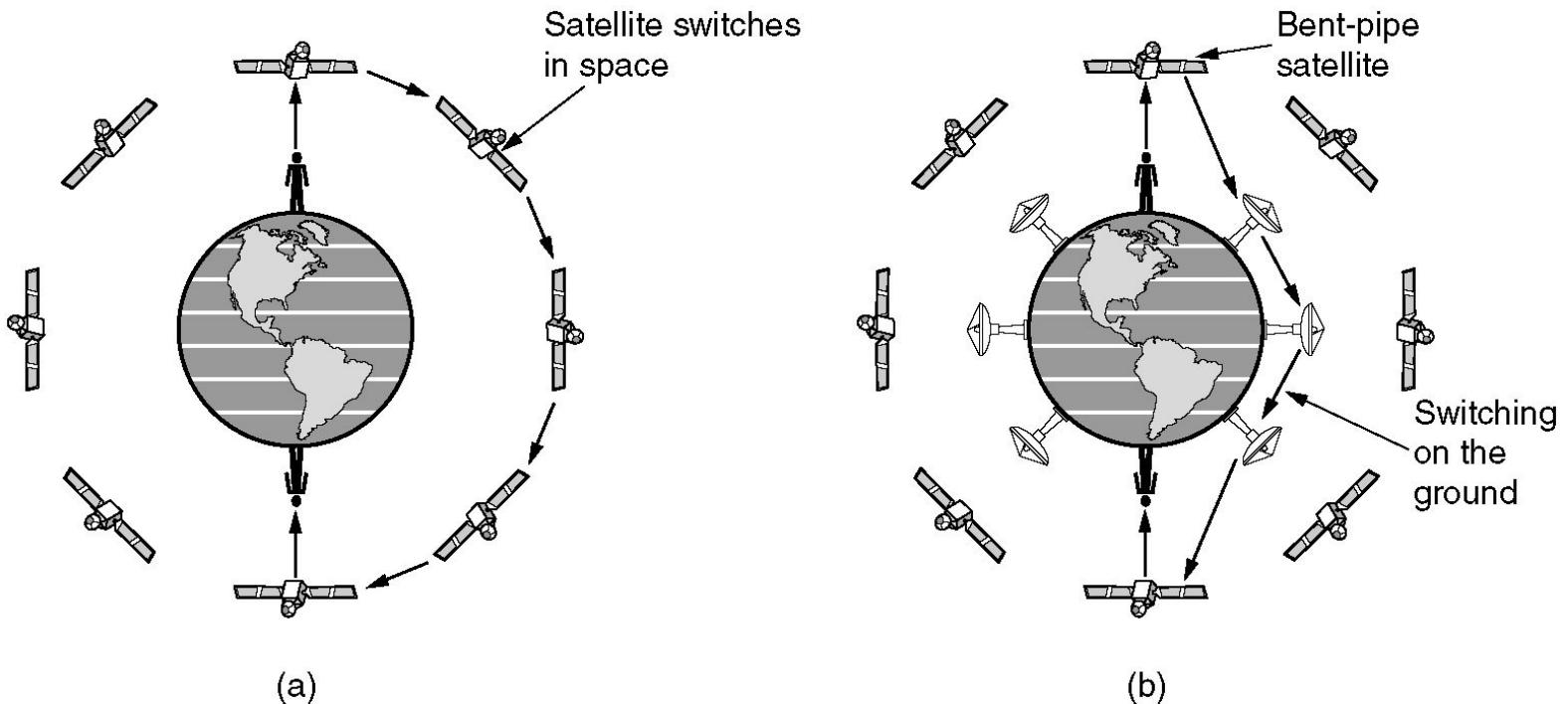
(a)



(b)

- (a) The Iridium satellites from six necklaces around the earth.
- (b) 1628 moving cells cover the earth.

Globalstar

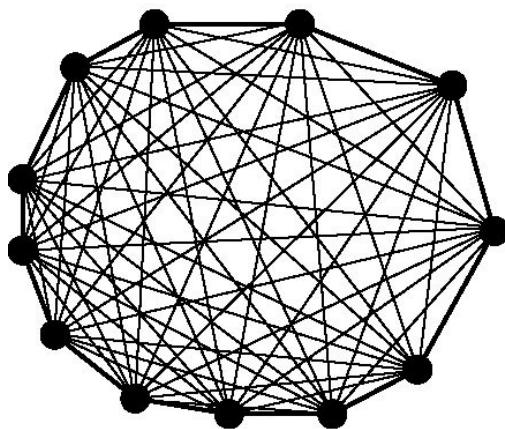


- (a) Relaying in space.
- (b) Relaying on the ground.

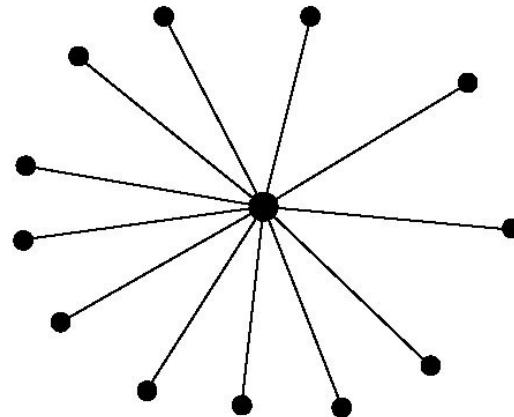
Public Switched Telephone System

- Structure of the Telephone System
- The Politics of Telephones
- The Local Loop: Modems, ADSL and Wireless
- Trunks and Multiplexing
- Switching

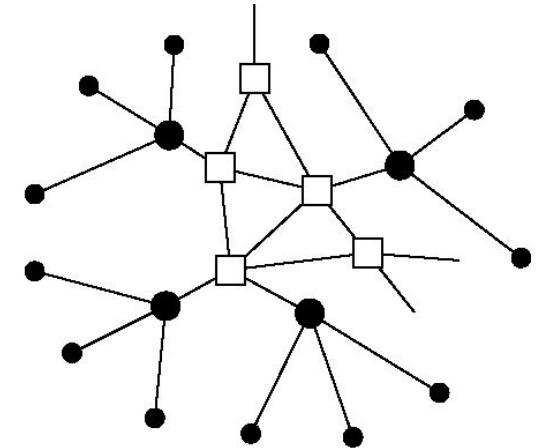
Structure of the Telephone System



(a)



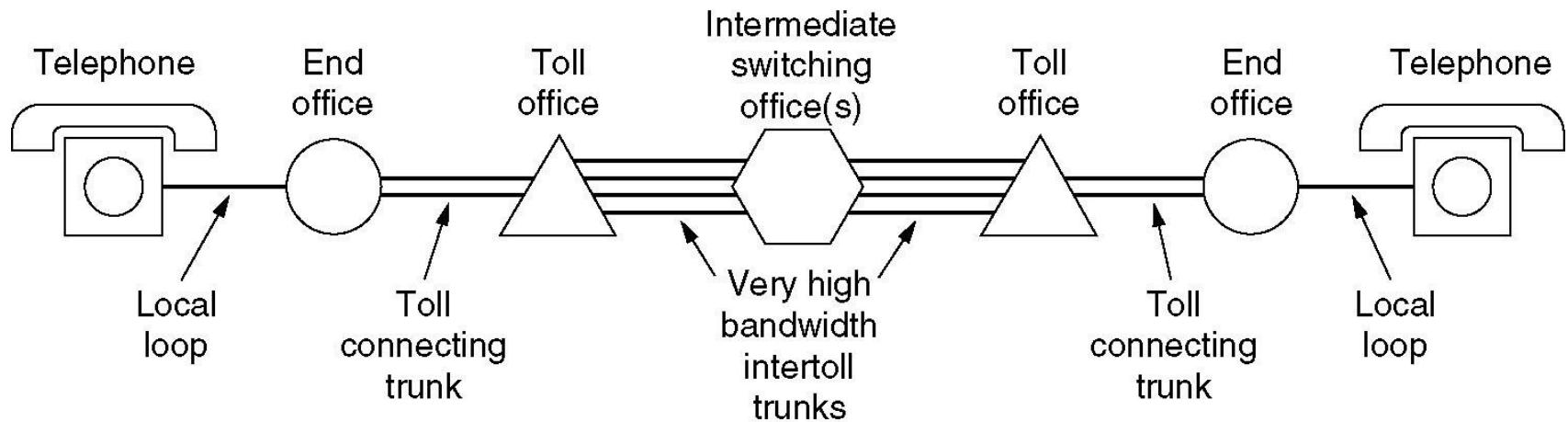
(b)



(c)

- (a) Fully-interconnected network.
- (b) Centralized switch.
- (c) Two-level hierarchy.

Structure of the Telephone System (2)

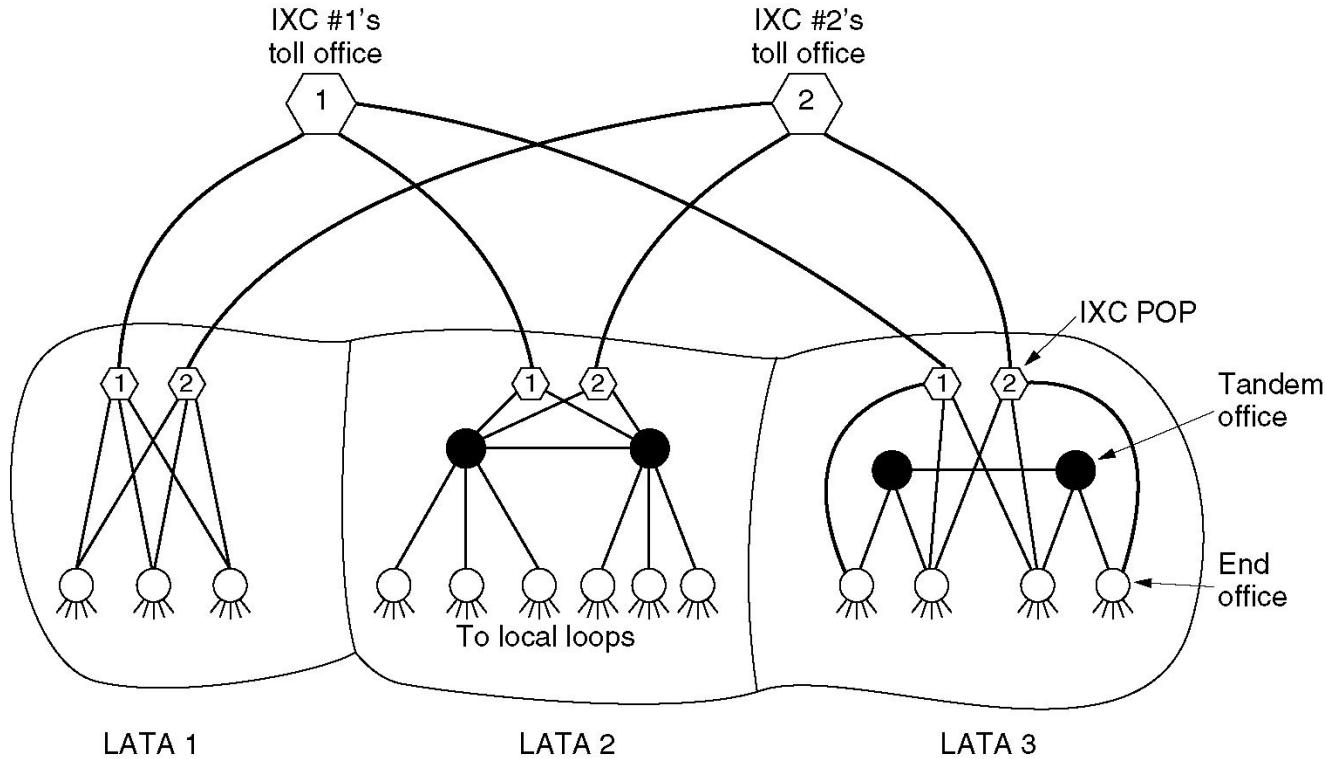


A typical circuit route for a medium-distance call.

Major Components of the Telephone System

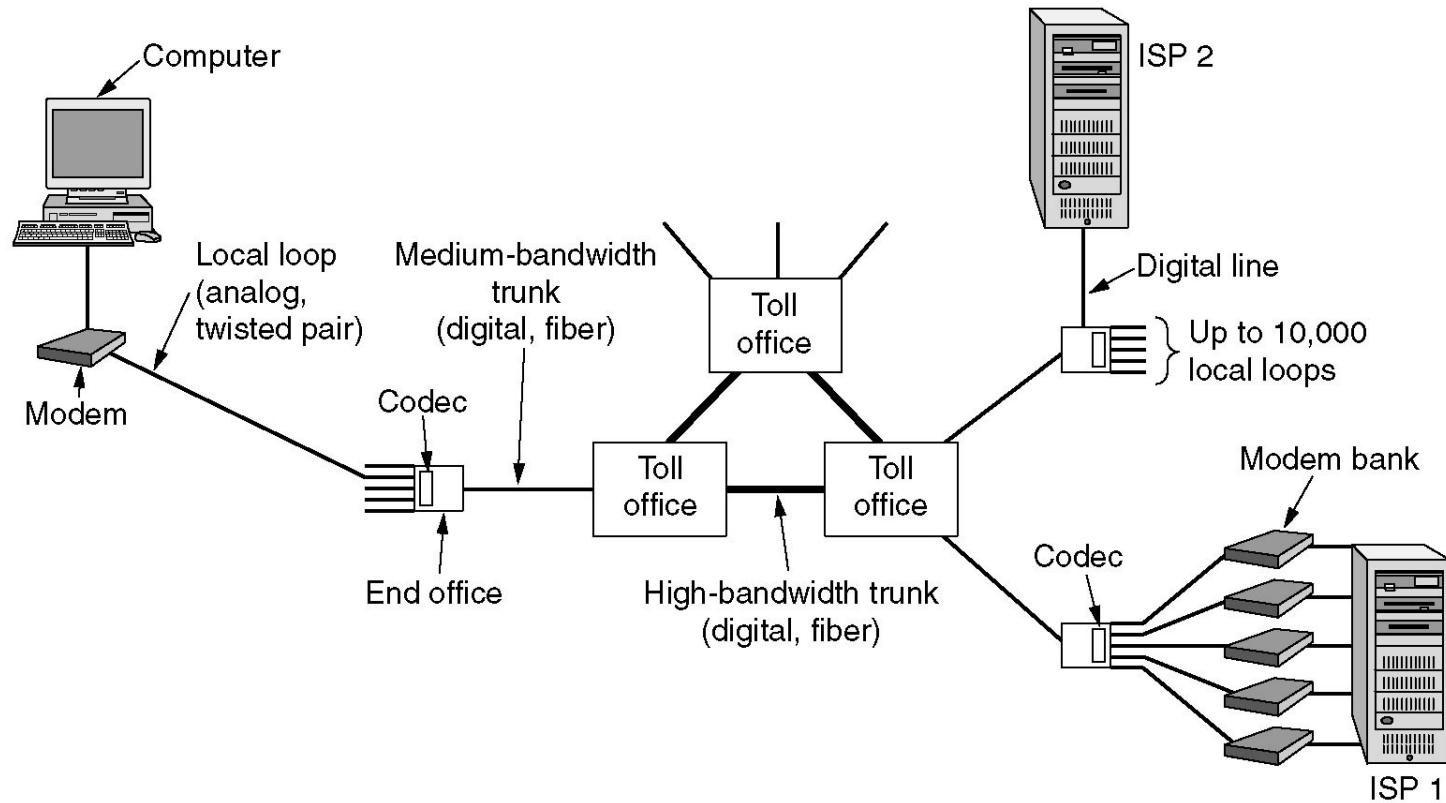
- Local loops
 - Analog twisted pairs going to houses and businesses
- Trunks
 - Digital fiber optics connecting the switching offices
- Switching offices
 - Where calls are moved from one trunk to another

The Politics of Telephones



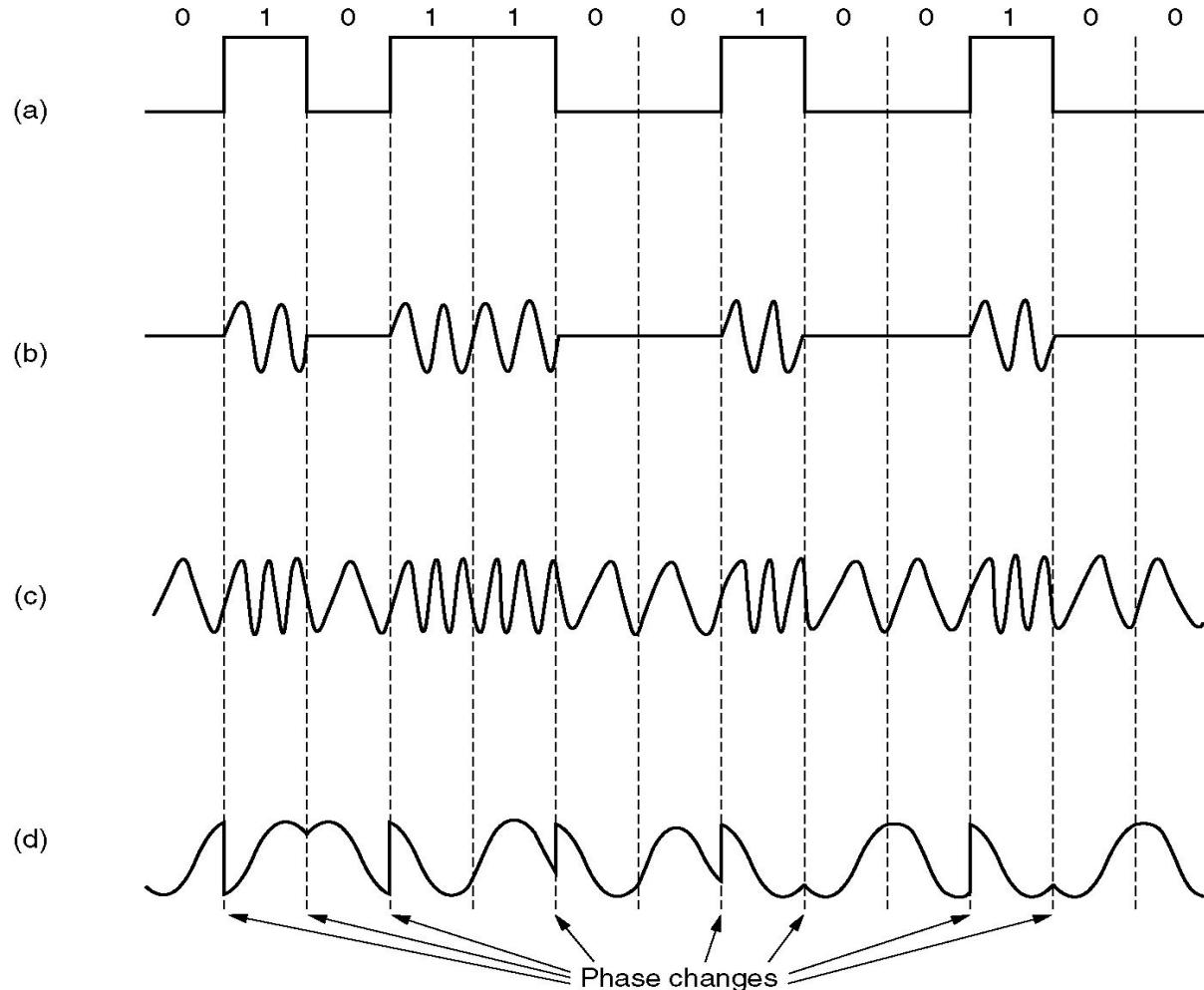
The relationship of LATAs, LECs, and IXCs. All the circles are LEC switching offices. Each hexagon belongs to the IXC whose number is on it.

The Local Loop: Modems, ADSL, and Wireless



The use of both analog and digital transmissions for a computer to computer call. Conversion is done by the modems and codecs.

Modems



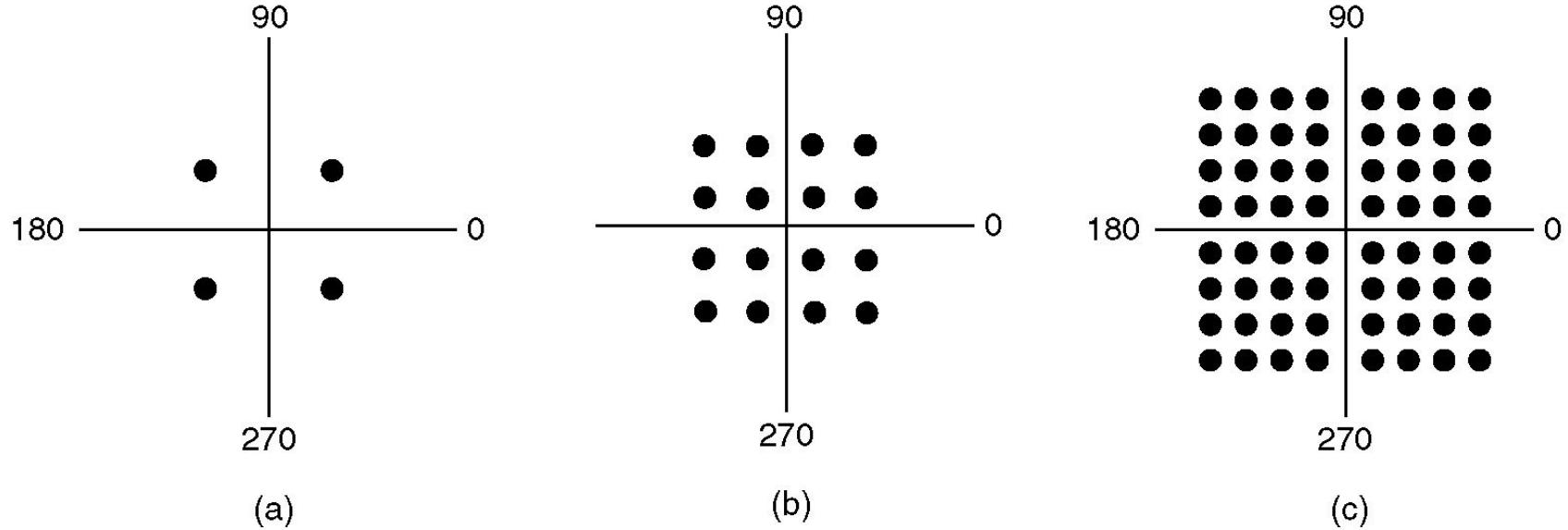
(a) A binary signal

(b) Amplitude modulation

(c) Frequency modulation

(d) Phase modulation

Modems (2)

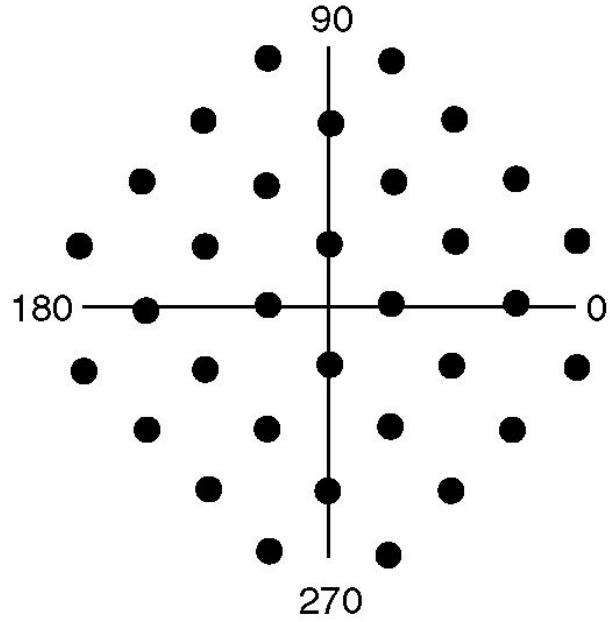


(a) QPSK.

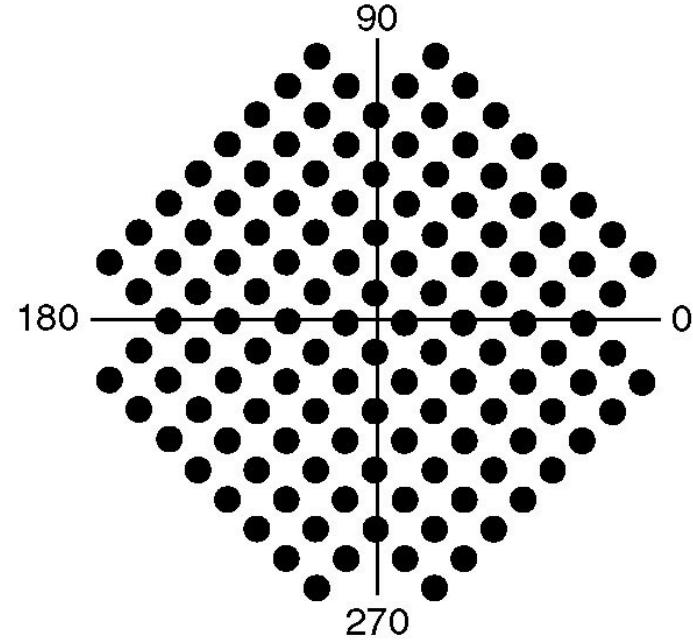
(b) QAM-16.

(c) QAM-64.

Modems (3)



(a)

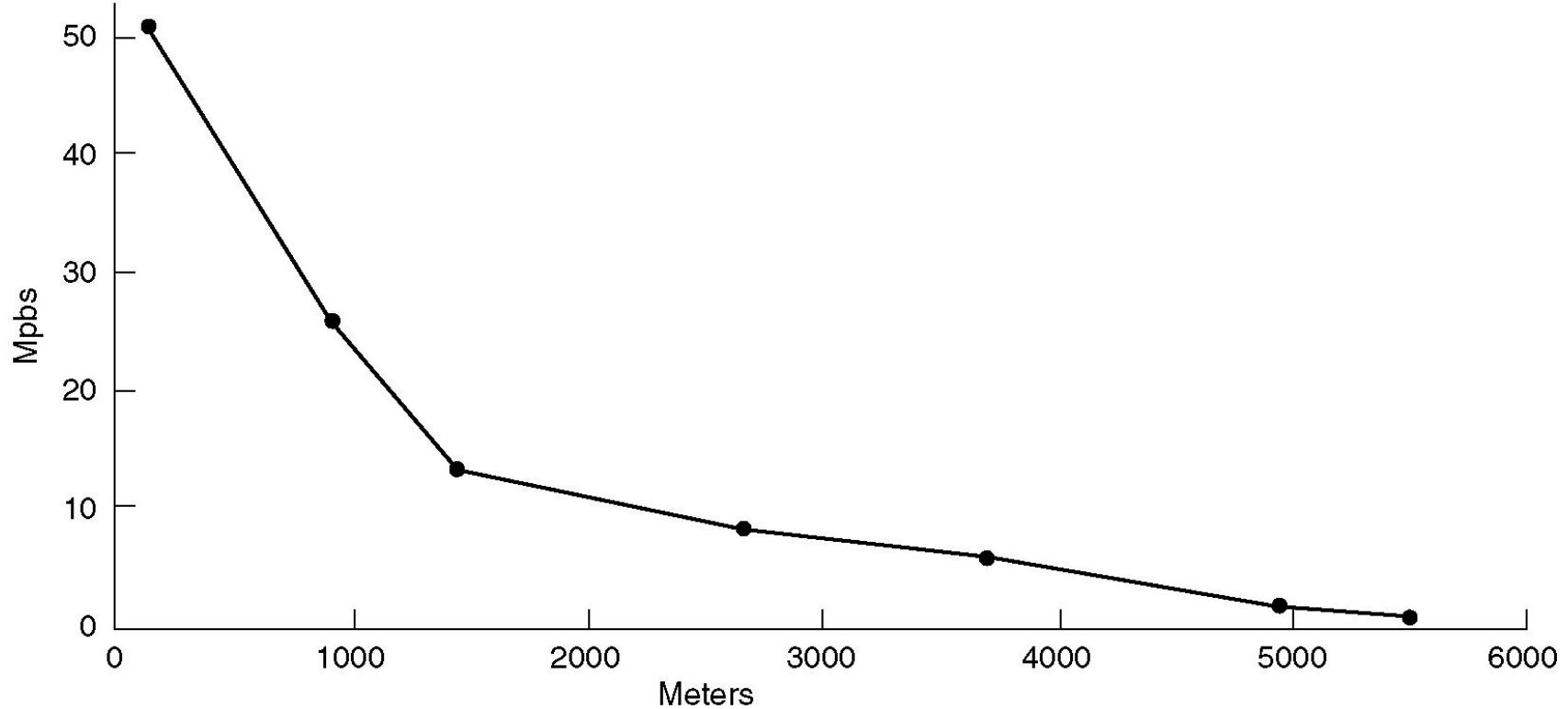


(b)

(a) V.32 for 9600 bps.

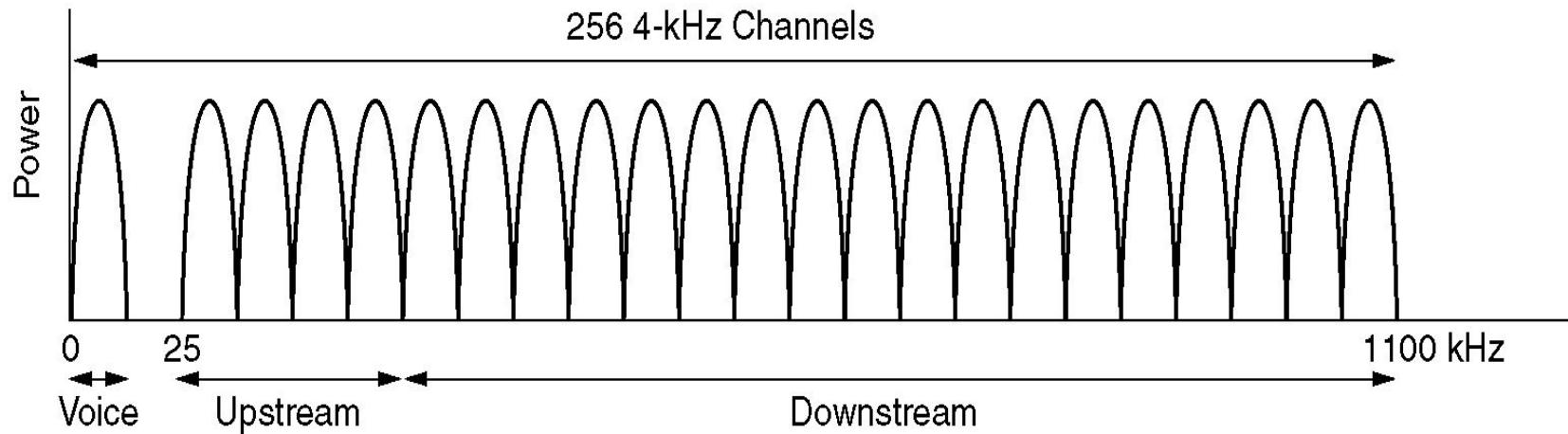
(b) V32 bis for 14,400 bps.

Digital Subscriber Lines



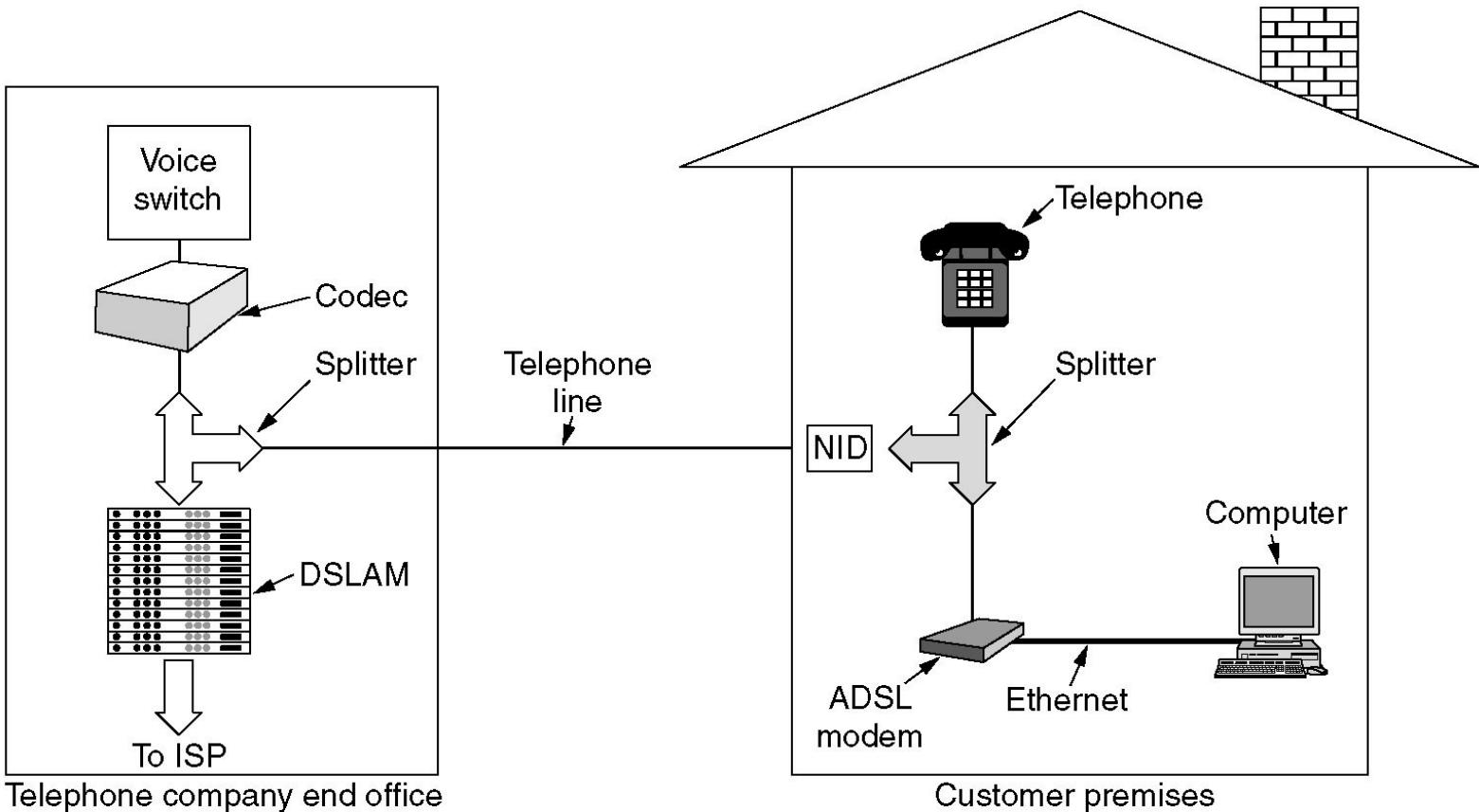
Bandwidth versus distance over category 3 UTP for DSL.

Digital Subscriber Lines (2)



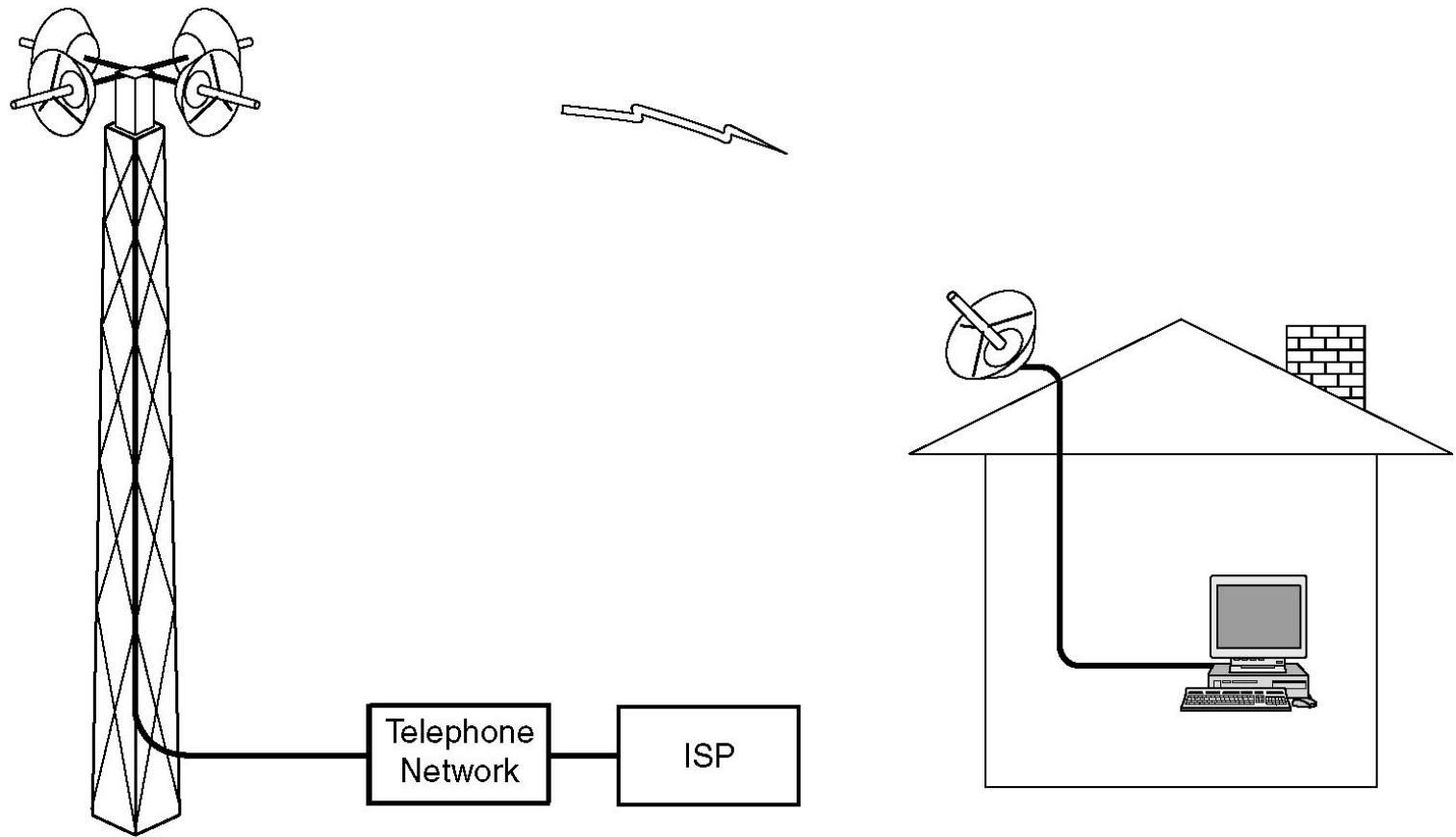
Operation of ADSL using discrete multitone modulation.

Digital Subscriber Lines (3)



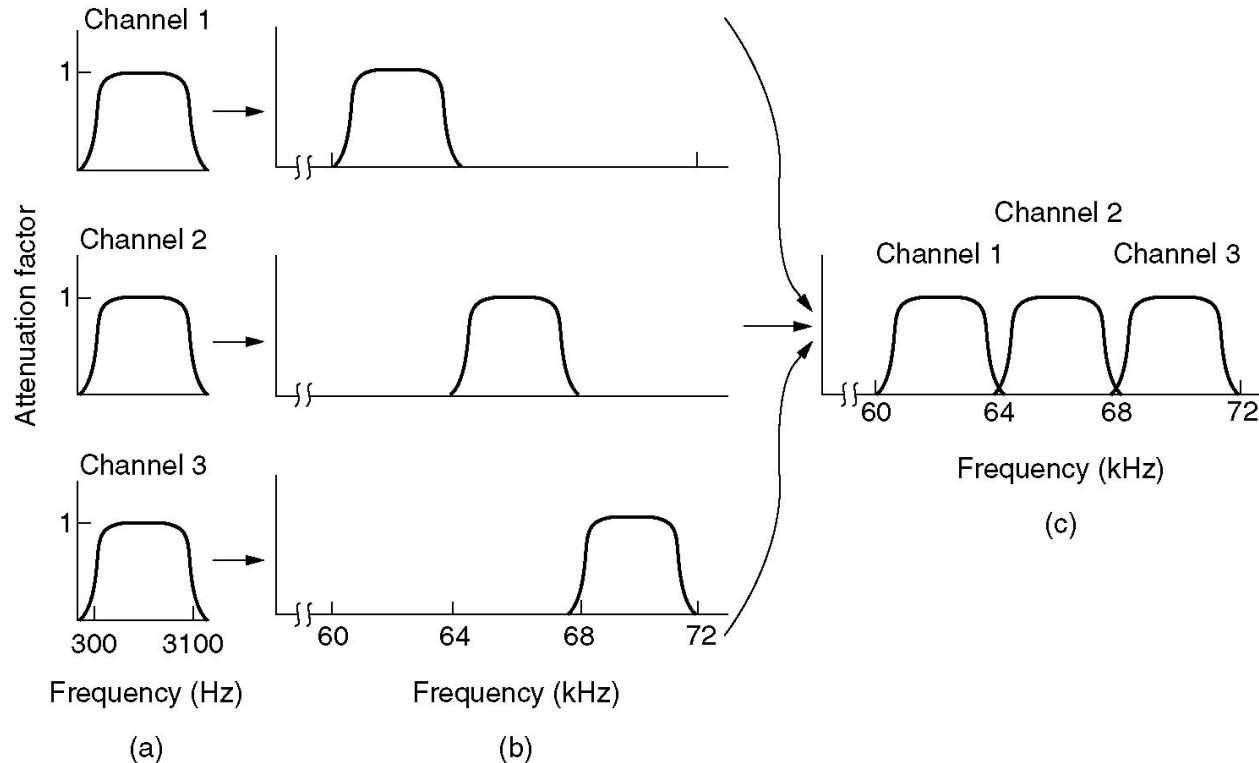
A typical ADSL equipment configuration.

Wireless Local Loops



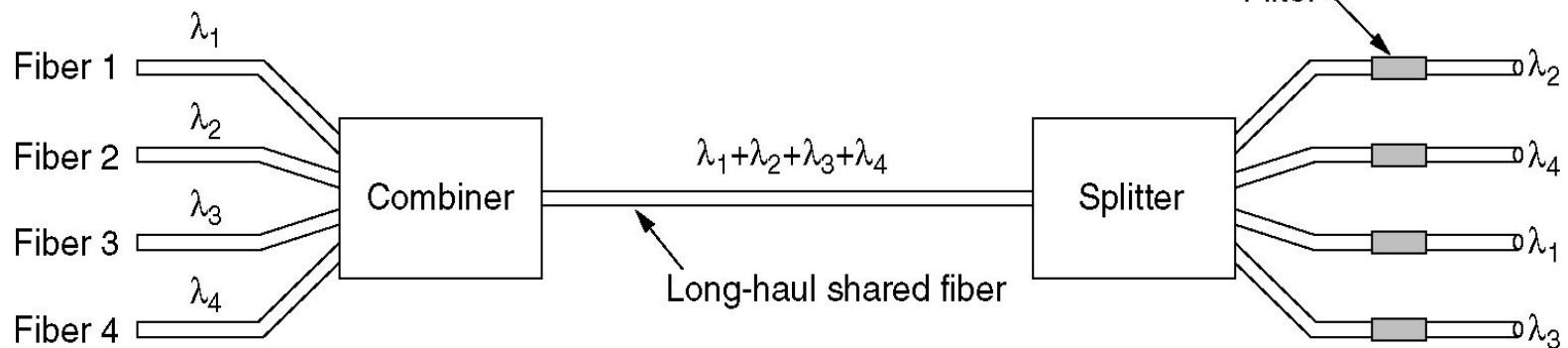
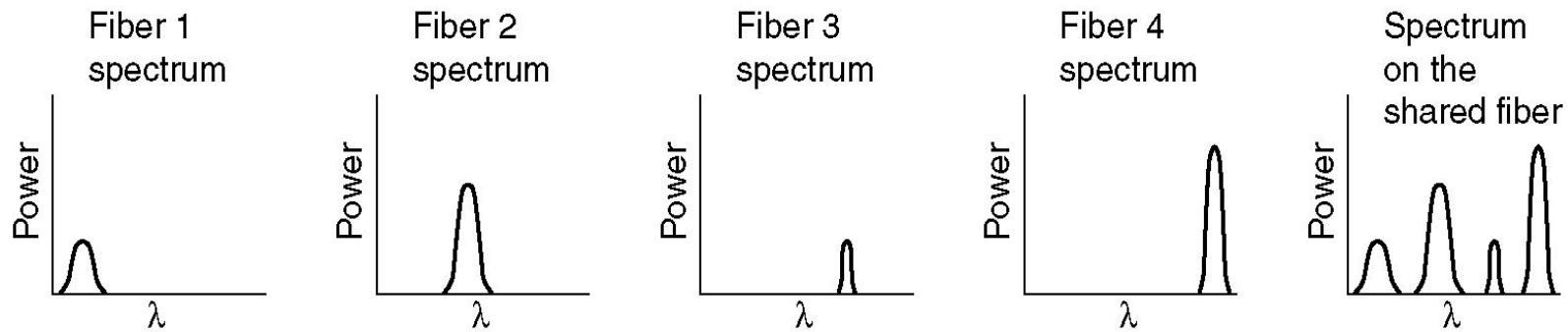
Architecture of an LMDS system.

Frequency Division Multiplexing



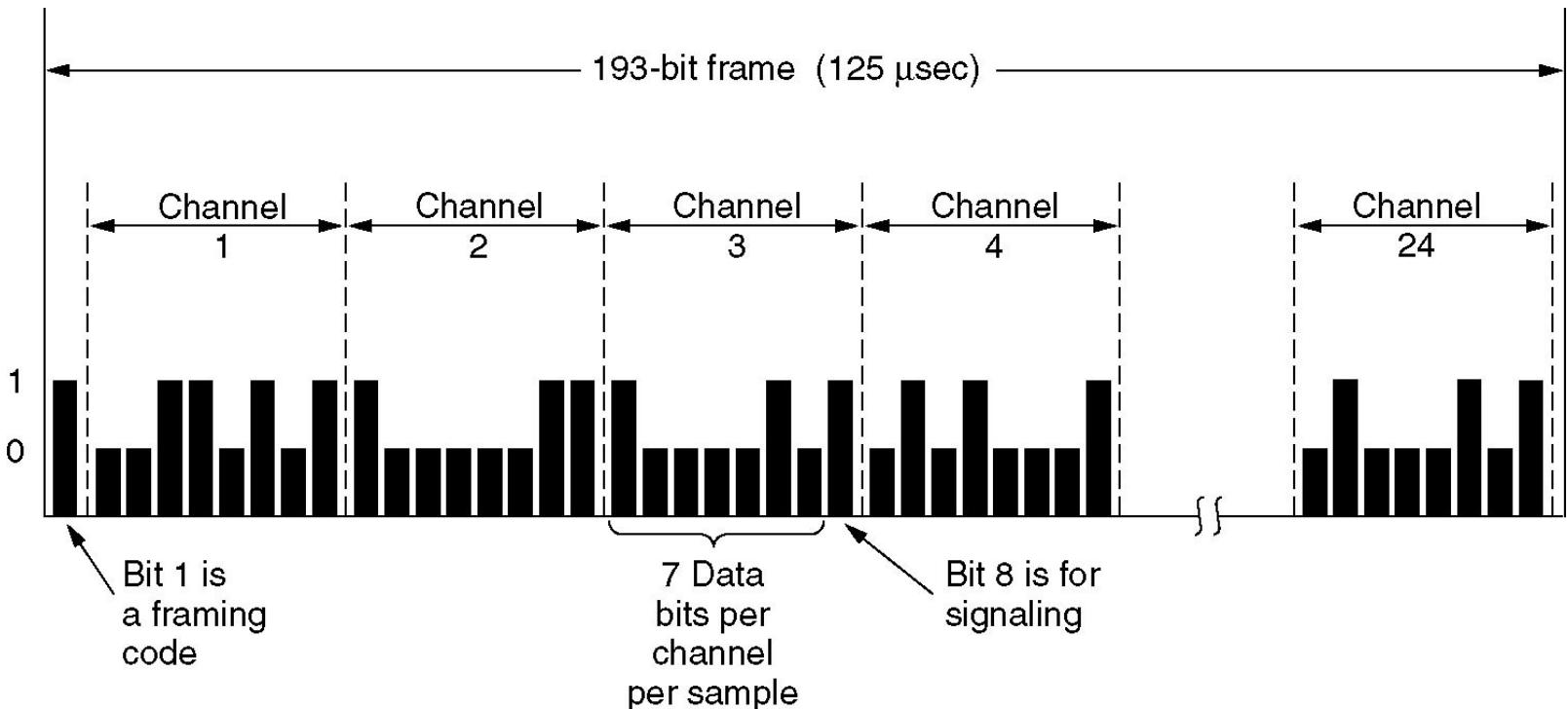
- (a) The original bandwidths.
- (b) The bandwidths raised in frequency.
- (b) The multiplexed channel.

Wavelength Division Multiplexing



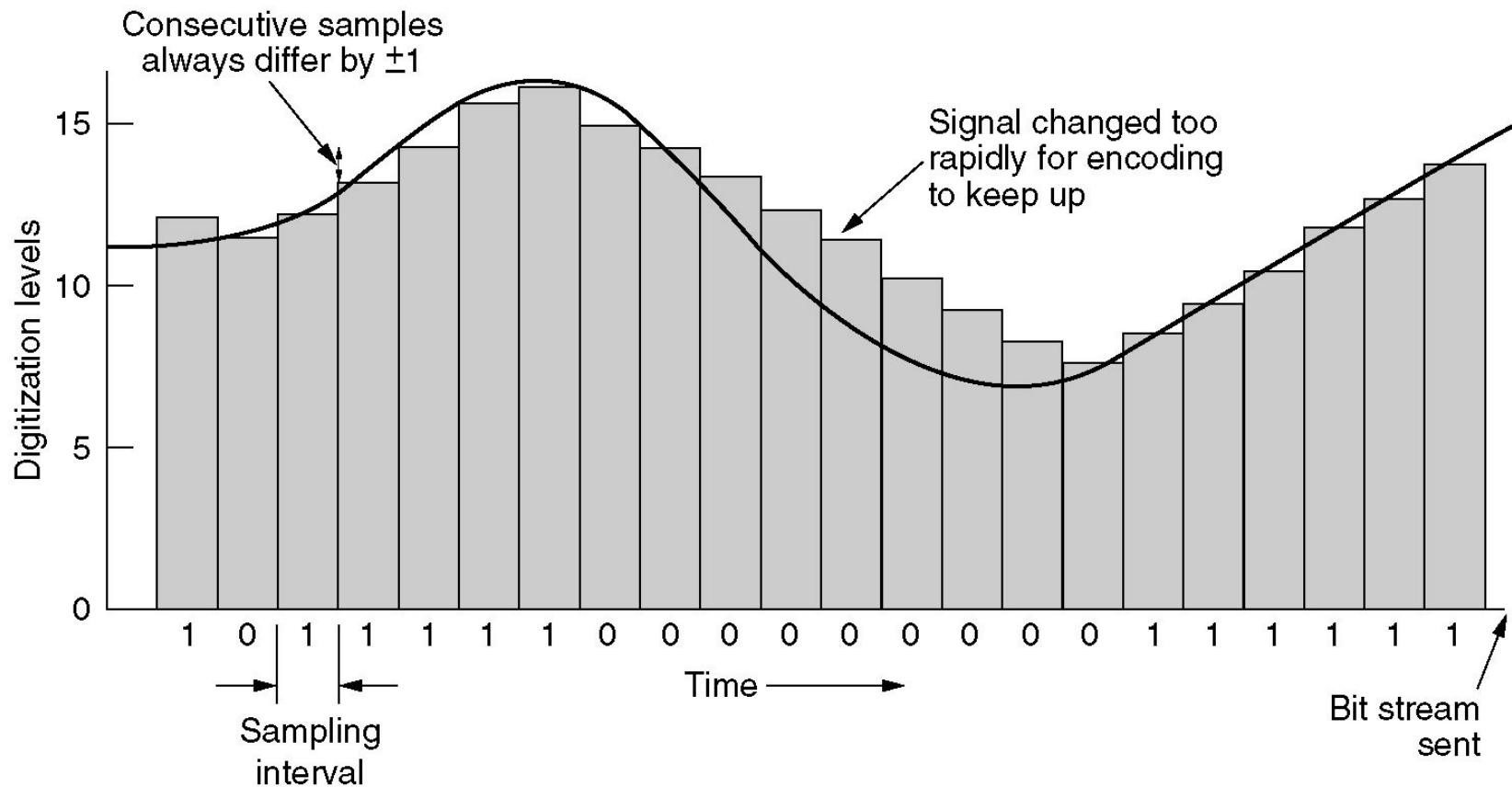
Wavelength division multiplexing.

Time Division Multiplexing



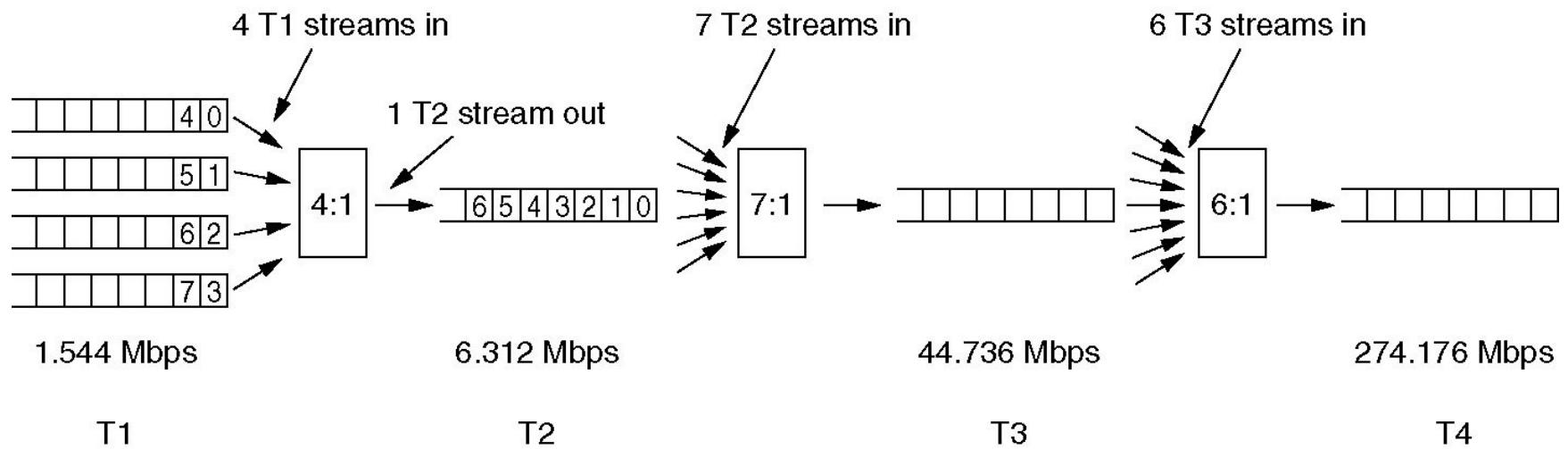
The T1 carrier (1.544 Mbps).

Time Division Multiplexing (2)



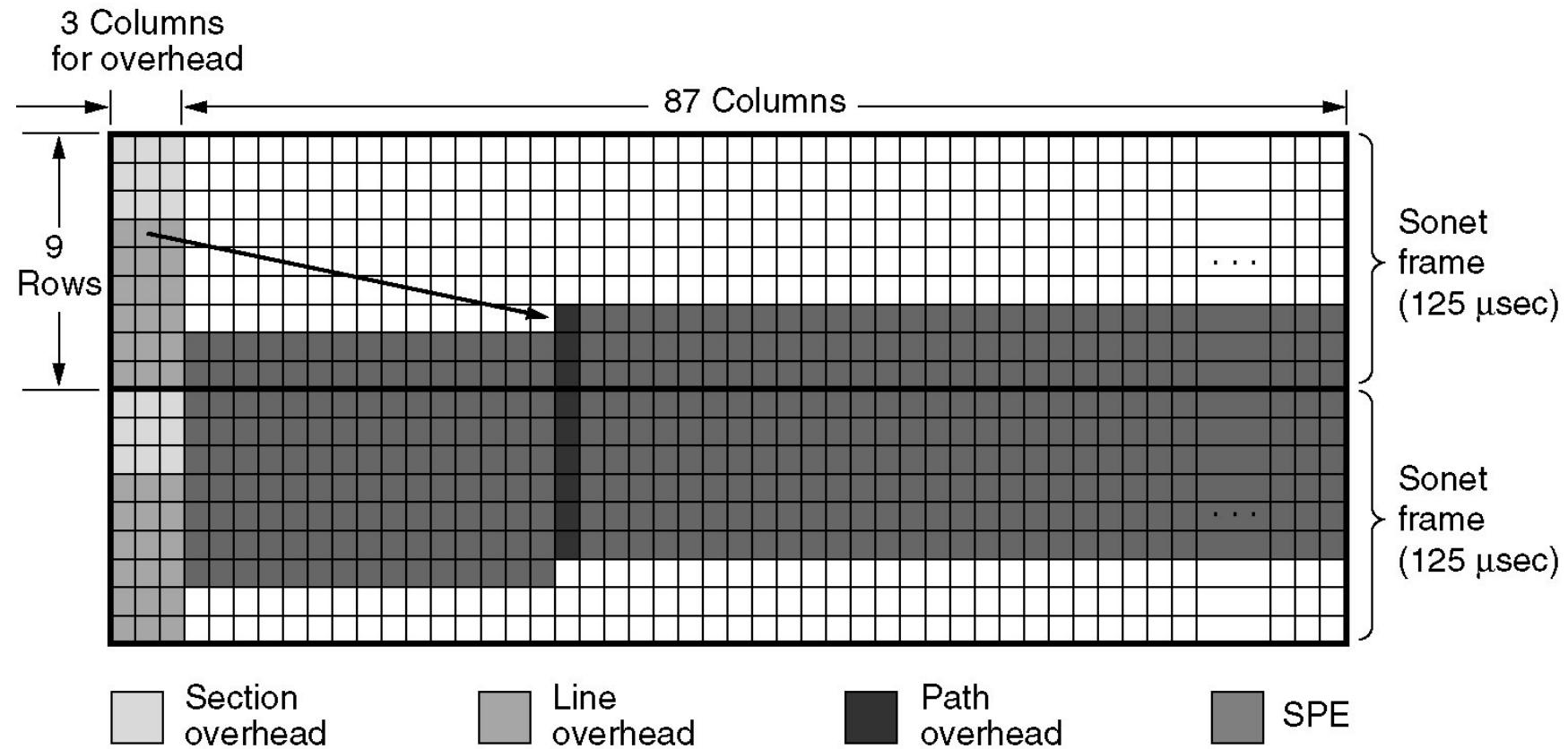
Delta modulation.

Time Division Multiplexing (3)



Multiplexing T1 streams into higher carriers.

Time Division Multiplexing (4)



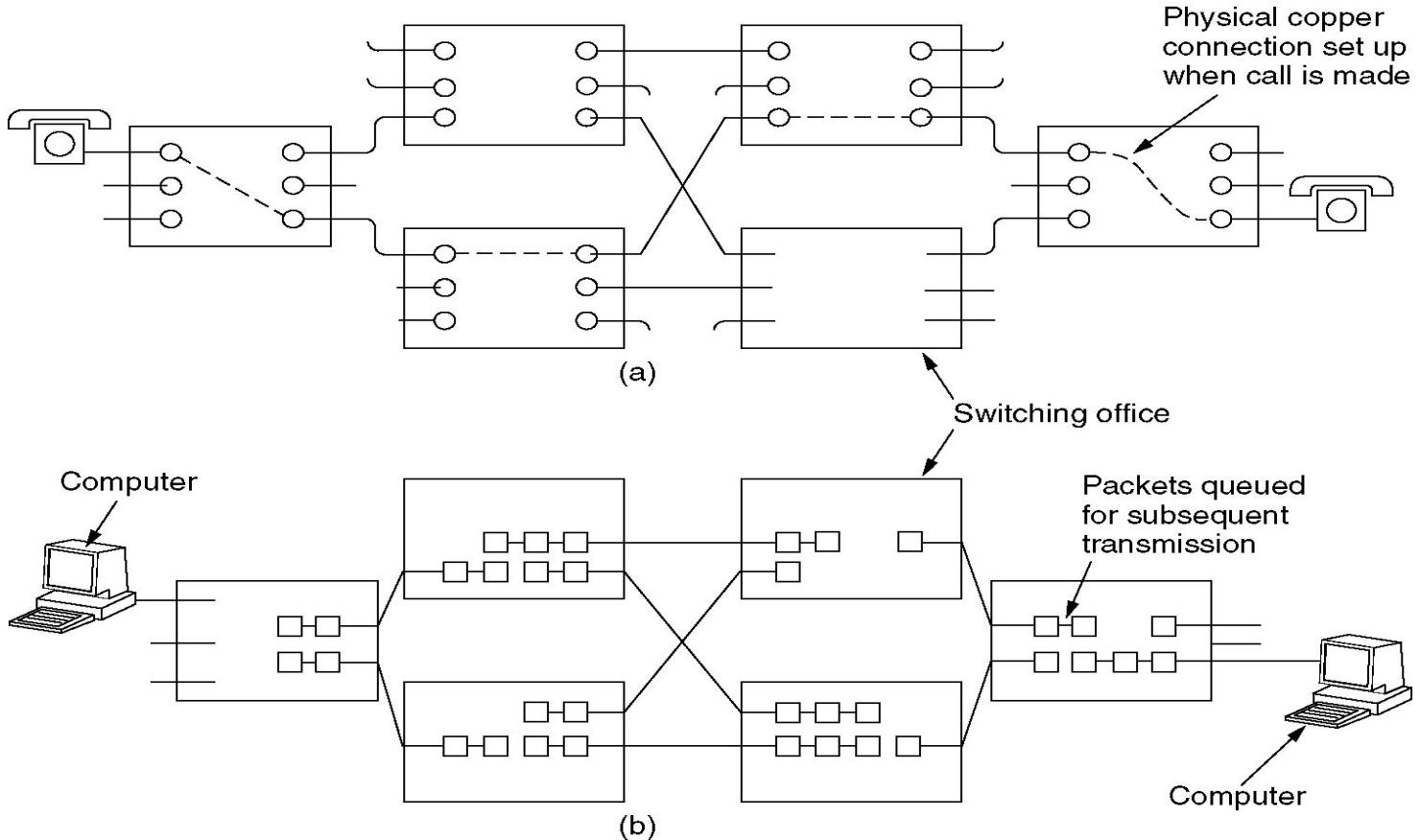
Two back-to-back SONET frames.

Time Division Multiplexing (5)

SONET		SDH	Data rate (Mbps)		
Electrical	Optical	Optical	Gross	SPE	User
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-9	OC-9	STM-3	466.56	451.008	445.824
STS-12	OC-12	STM-4	622.08	601.344	594.432
STS-18	OC-18	STM-6	933.12	902.016	891.648
STS-24	OC-24	STM-8	1244.16	1202.688	1188.864
STS-36	OC-36	STM-12	1866.24	1804.032	1783.296
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728
STS-192	OC-192	STM-64	9953.28	9621.504	9510.912

SONET and SDH multiplex rates.

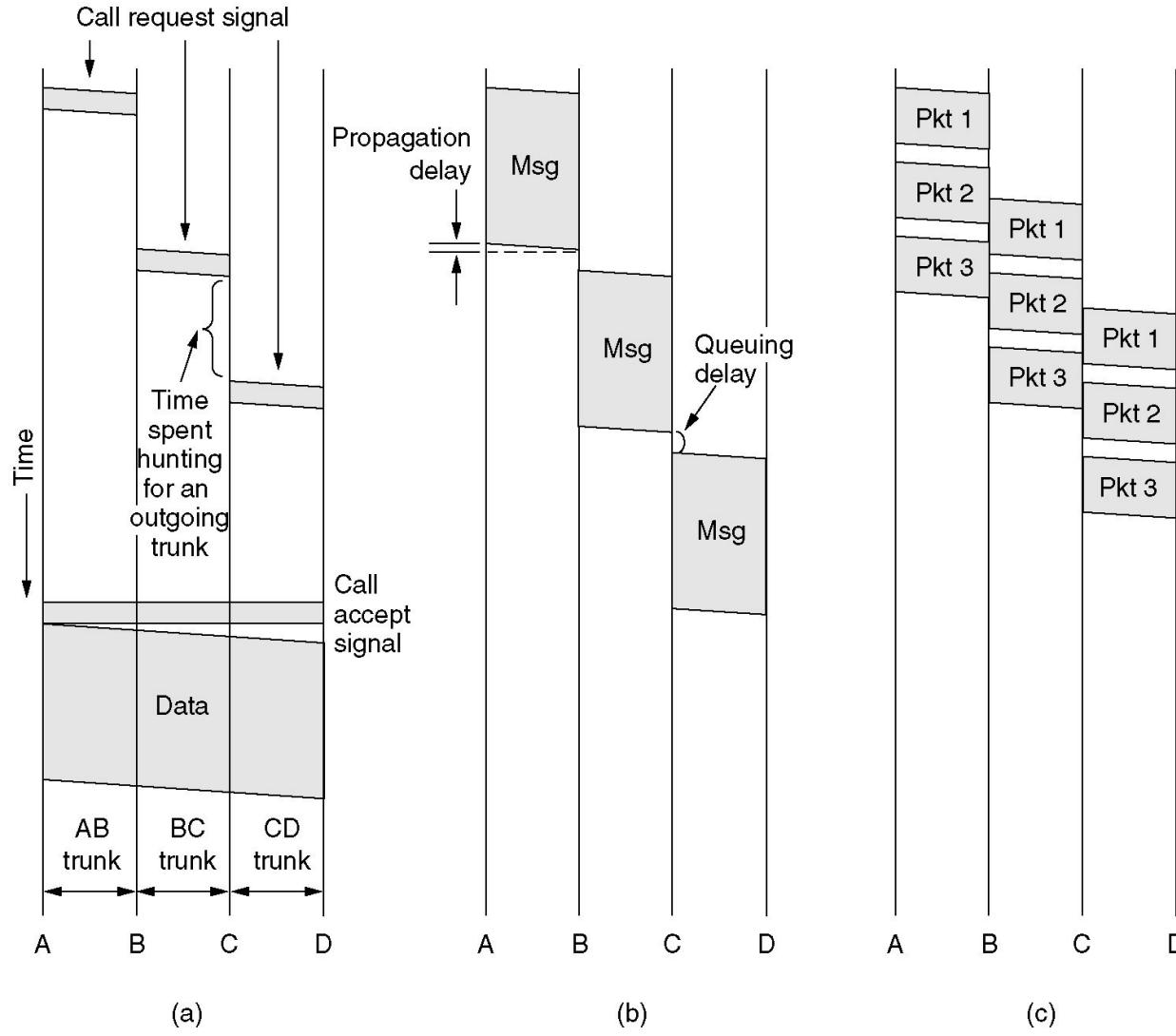
Circuit Switching



(a) Circuit switching.

(b) Packet switching.

Message Switching



(a) Circuit switching (b) Message switching (c) Packet switching

Packet Switching

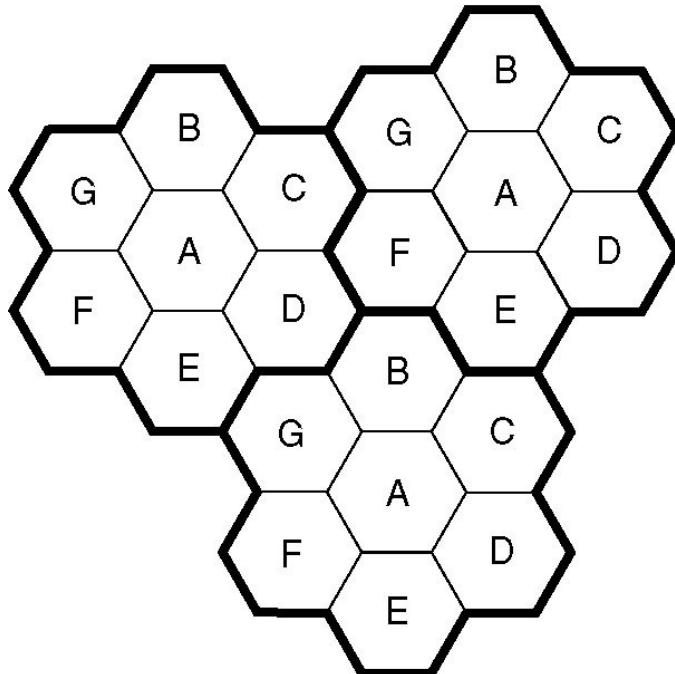
Item	Circuit-switched	Packet-switched
Call setup	Required	Not needed
Dedicated physical path	Yes	No
Each packet follows the same route	Yes	No
Packets arrive in order	Yes	No
Is a switch crash fatal	Yes	No
Bandwidth available	Fixed	Dynamic
When can congestion occur	At setup time	On every packet
Potentially wasted bandwidth	Yes	No
Store-and-forward transmission	No	Yes
Transparency	Yes	No
Charging	Per minute	Per packet

A comparison of circuit switched and packet-switched networks.

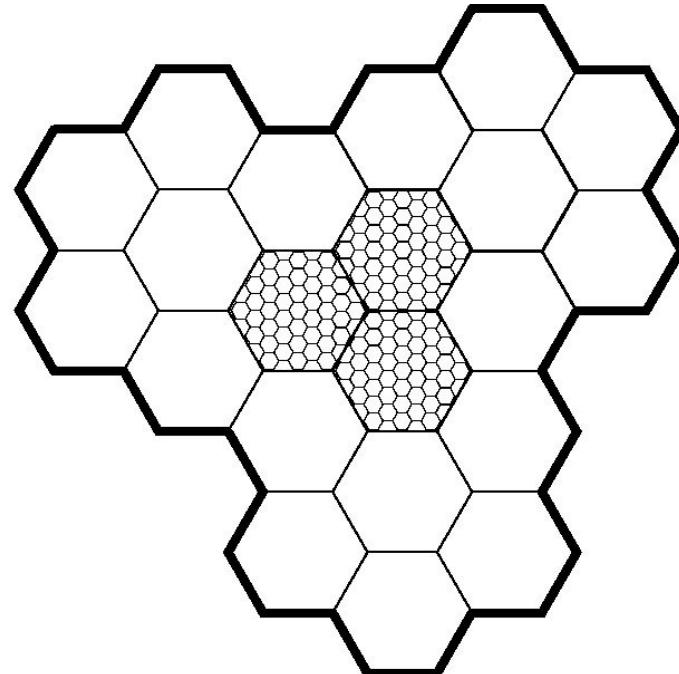
The Mobile Telephone System

- First-Generation Mobile Phones:
Analog Voice
- Second-Generation Mobile Phones:
Digital Voice
- Third-Generation Mobile Phones:
Digital Voice and Data

Advanced Mobile Phone System



(a)



(b)

- (a) Frequencies are not reused in adjacent cells.
- (b) To add more users, smaller cells can be used.

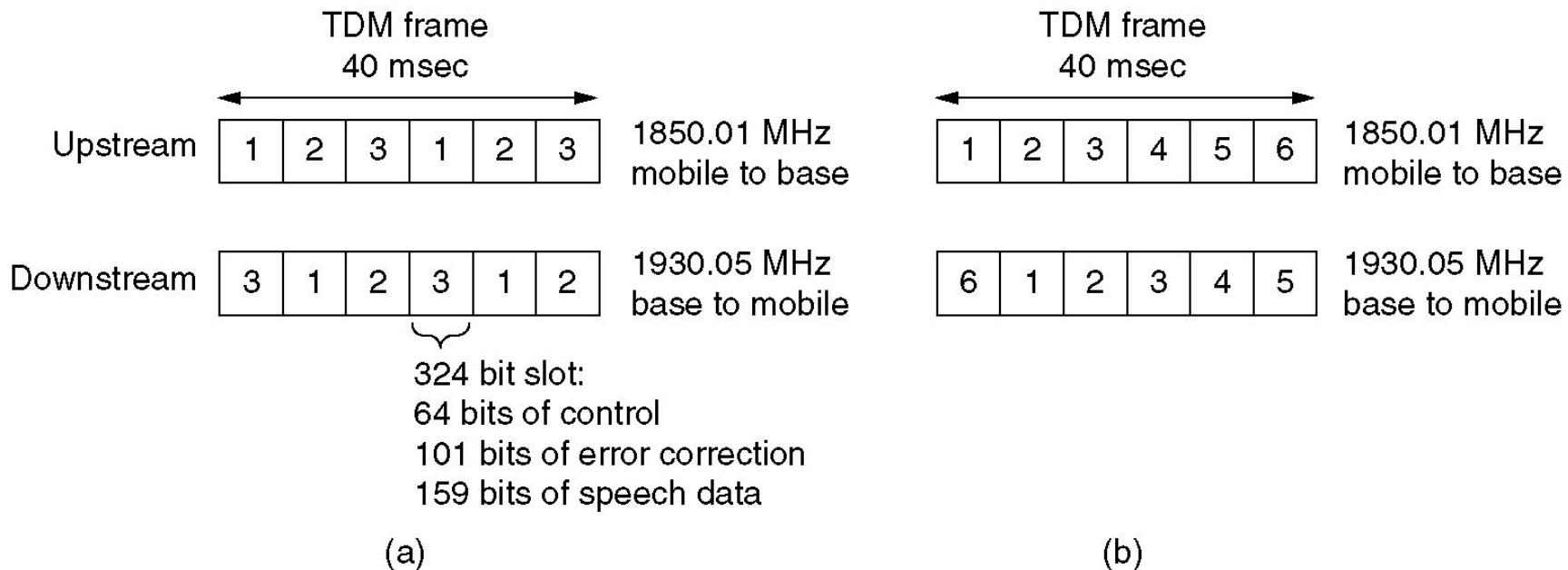
Channel Categories

The 832 channels are divided into four categories:

- Control (base to mobile) to manage the system
- Paging (base to mobile) to alert users to calls for them
- Access (bidirectional) for call setup and channel assignment
- Data (bidirectional) for voice, fax, or data

D-AMPS

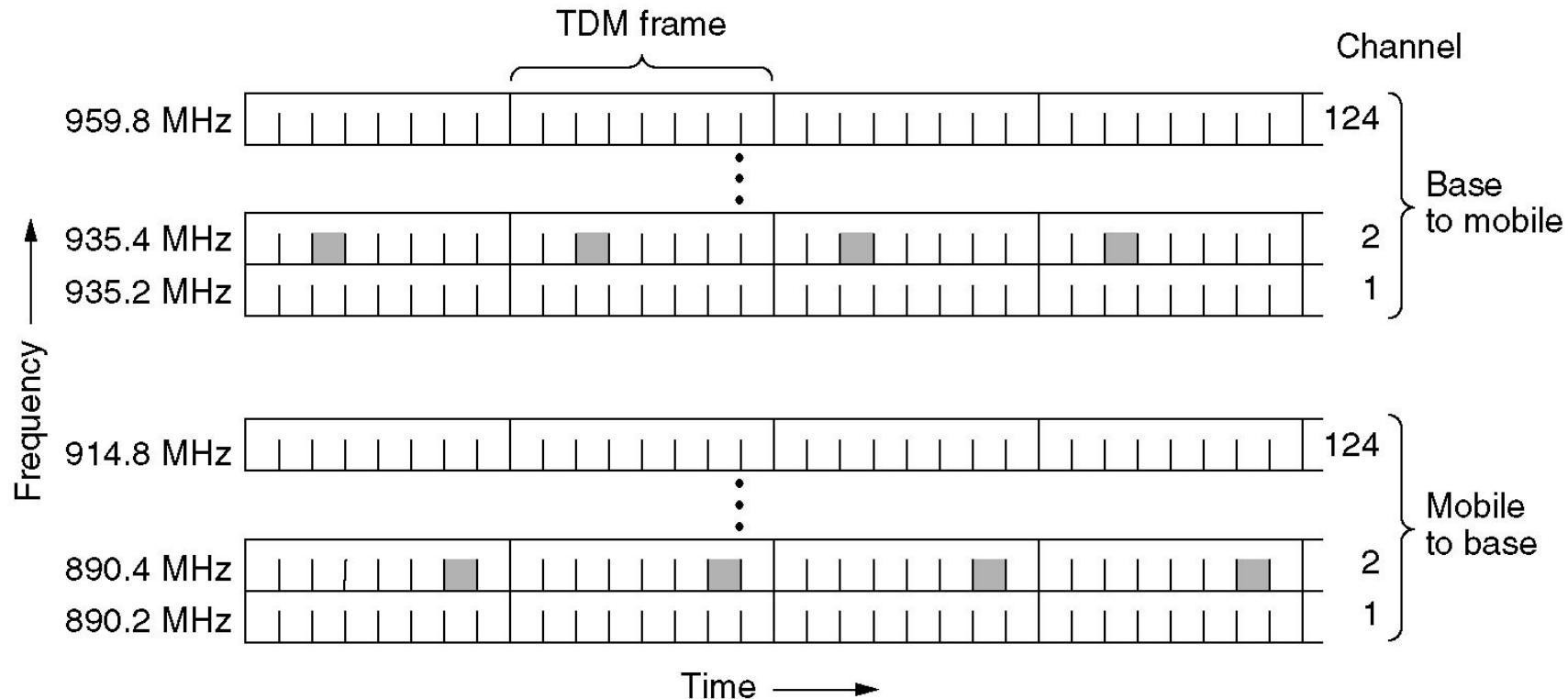
Digital Advanced Mobile Phone System



- (a) A D-AMPS channel with three users.
(b) A D-AMPS channel with six users.

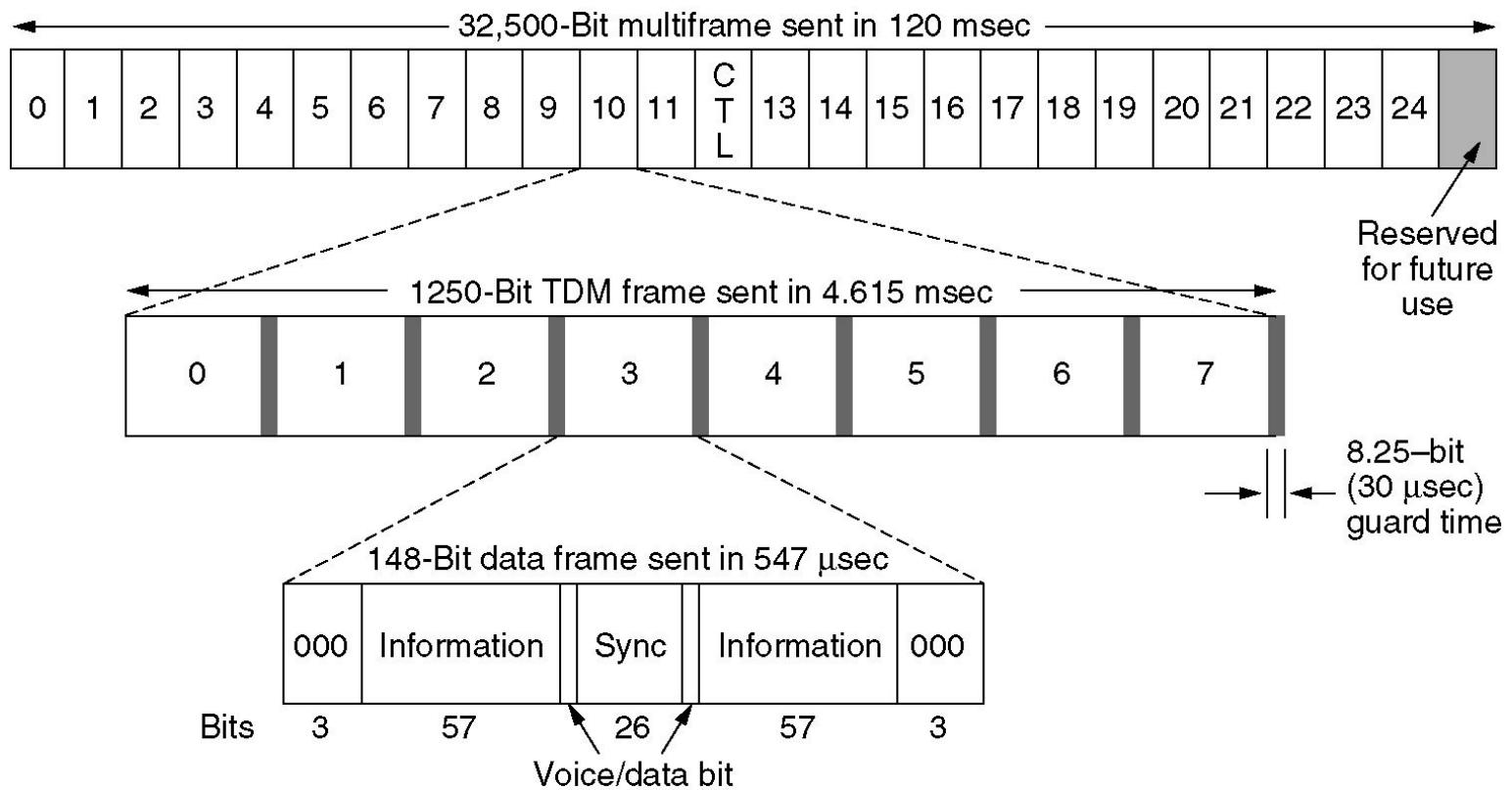
GSM

Global System for Mobile Communications



GSM uses 124 frequency channels, each of which uses an eight-slot TDMA system

GSM (2)



A portion of the GSM framing structure.

CDMA – Code Division Multiple Access

A: 0 0 0 1 1 0 1 1
B: 0 0 1 0 1 1 1 0
C: 0 1 0 1 1 1 0 0
D: 0 1 0 0 0 0 1 0

(a)

A: (-1 -1 -1 +1 +1 -1 +1 +1)
B: (-1 -1 +1 -1 +1 +1 +1 -1)
C: (-1 +1 -1 +1 +1 +1 -1 -1)
D: (-1 +1 -1 -1 -1 -1 +1 -1)

(b)

Six examples:

-- 1 -- **C**
- 1 1 -- **B** + **C**
1 0 -- **A** + **B**
1 0 1 -- **A** + **B** + **C**
1 1 1 1 **A** + **B** + **C** + **D**
1 1 0 1 **A** + **B** + **C** + **D**

$S_1 = (-1 +1 -1 +1 +1 +1 -1 -1)$
 $S_2 = (-2 \ 0 \ 0 \ +2 \ +2 \ 0 \ -2)$
 $S_3 = (\ 0 \ 0 \ -2 \ +2 \ 0 \ -2 \ 0 \ +2)$
 $S_4 = (-1 +1 -3 +3 +1 -1 -1 +1)$
 $S_5 = (-4 \ 0 \ -2 \ 0 \ +2 \ 0 \ +2 \ -2)$
 $S_6 = (-2 \ -2 \ 0 \ -2 \ 0 \ -2 \ +4 \ 0)$

(c)

$$\begin{aligned}S_1 \bullet C &= (1 +1 +1 +1 +1 +1 +1)/8 = 1 \\S_2 \bullet C &= (2 +0 +0 +0 +2 +2 +0 +2)/8 = 1 \\S_3 \bullet C &= (0 +0 +2 +2 +0 -2 +0 -2)/8 = 0 \\S_4 \bullet C &= (1 +1 +3 +3 +1 -1 +1 -1)/8 = 1 \\S_5 \bullet C &= (4 +0 +2 +0 +2 +0 -2 +2)/8 = 1 \\S_6 \bullet C &= (2 -2 +0 -2 +0 -2 -4 +0)/8 = -1\end{aligned}$$

(d)

- (a) Binary chip sequences for four stations
- (b) Bipolar chip sequences
- (c) Six examples of transmissions
- (d) Recovery of station C's signal

Third-Generation Mobile Phones: Digital Voice and Data

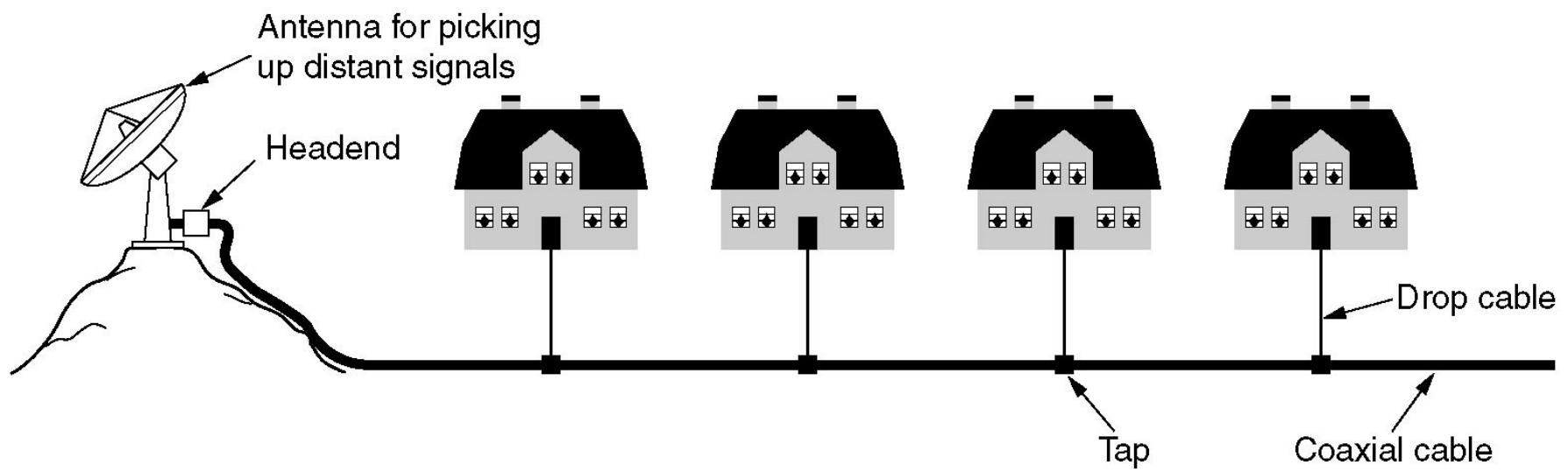
Basic services an IMT-2000 network should provide

- High-quality voice transmission
- Messaging (replace e-mail, fax, SMS, chat, etc.)
- Multimedia (music, videos, films, TV, etc.)
- Internet access (web surfing, w/multimedia.)

Cable Television

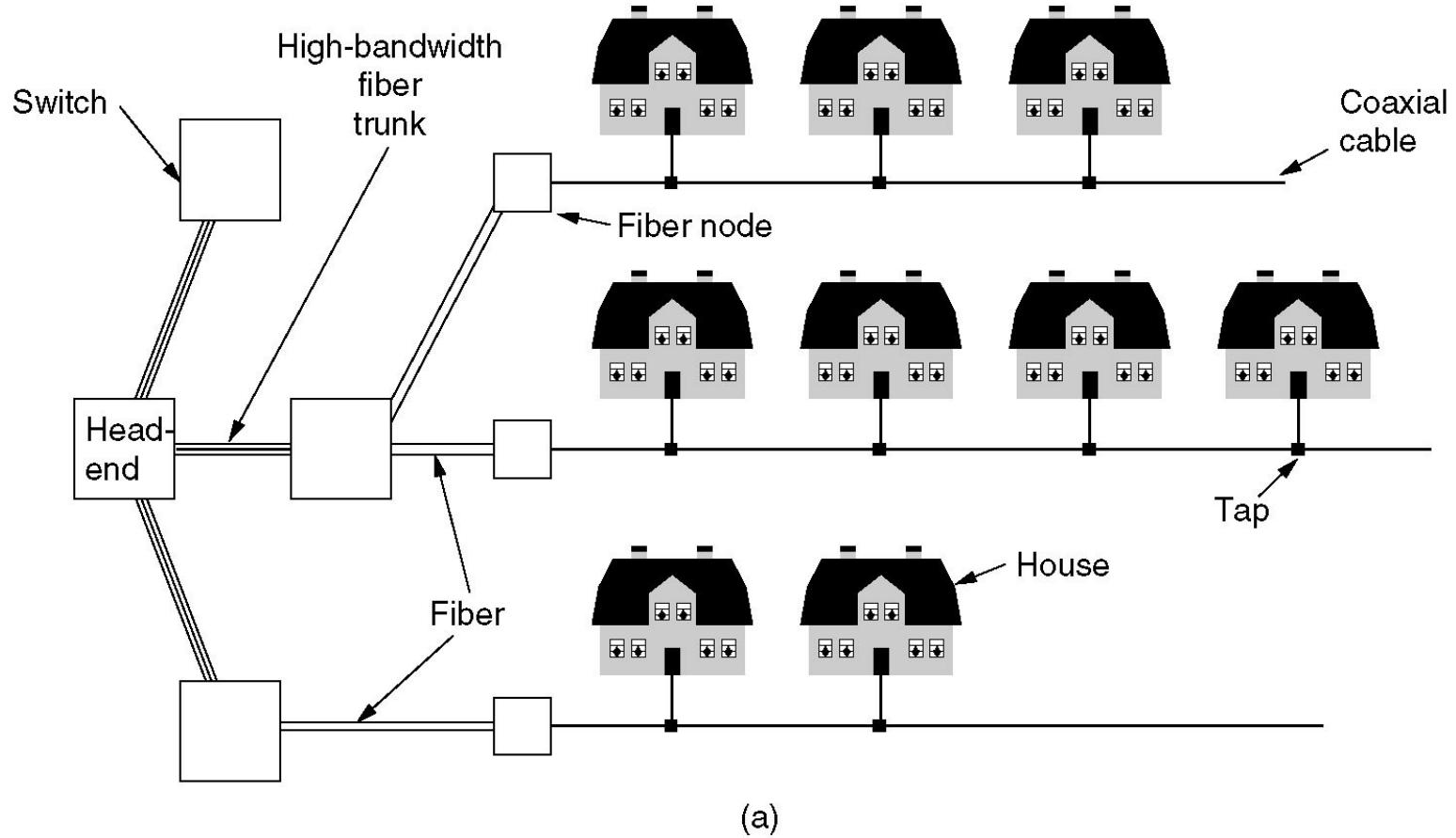
- Community Antenna Television
- Internet over Cable
- Spectrum Allocation
- Cable Modems
- ADSL versus Cable

Community Antenna Television



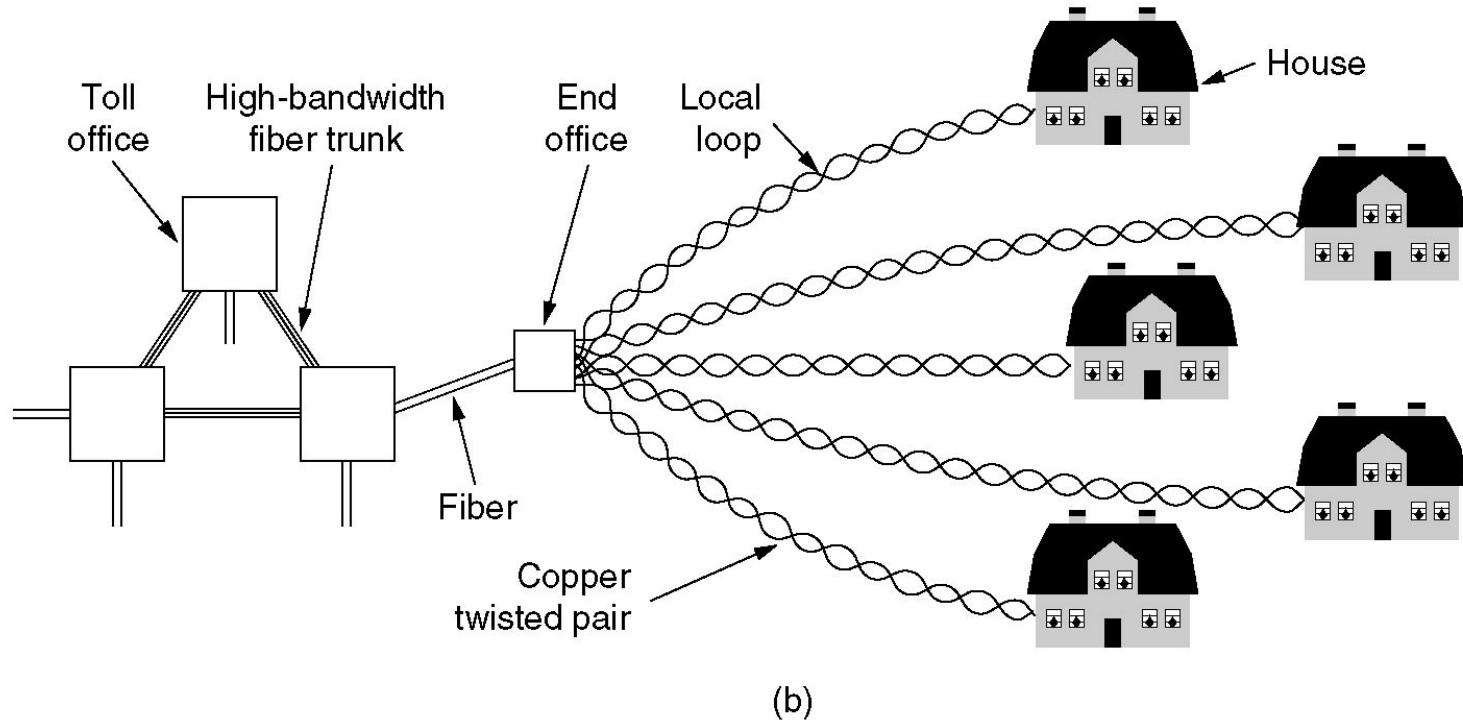
An early cable television system.

Internet over Cable



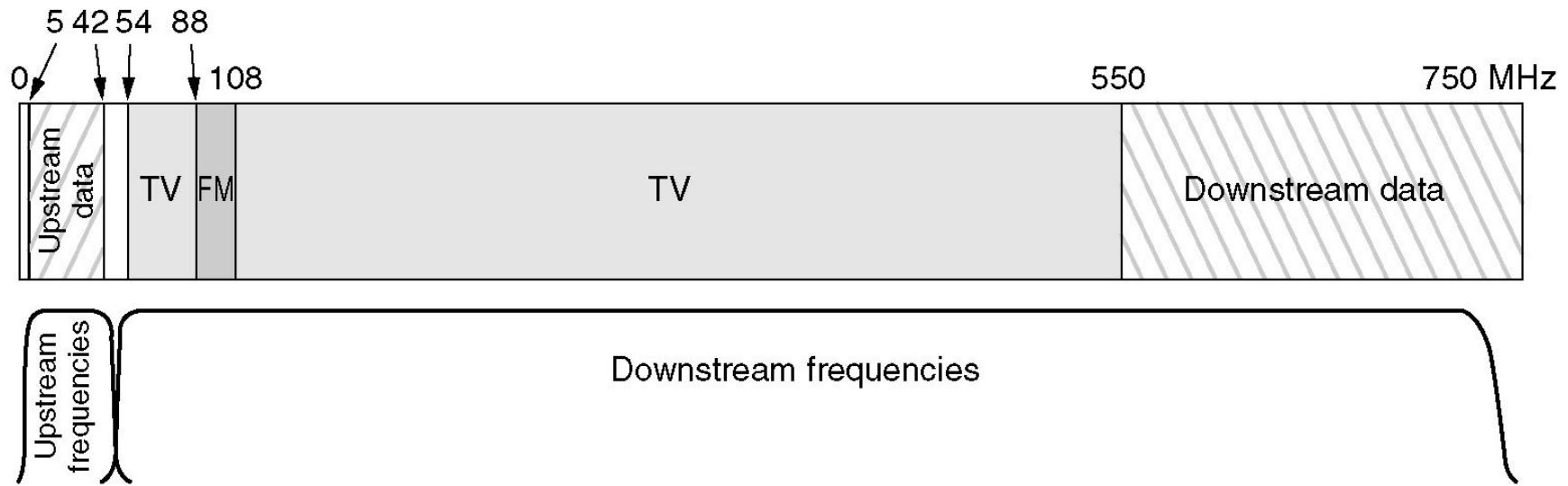
Cable television

Internet over Cable (2)



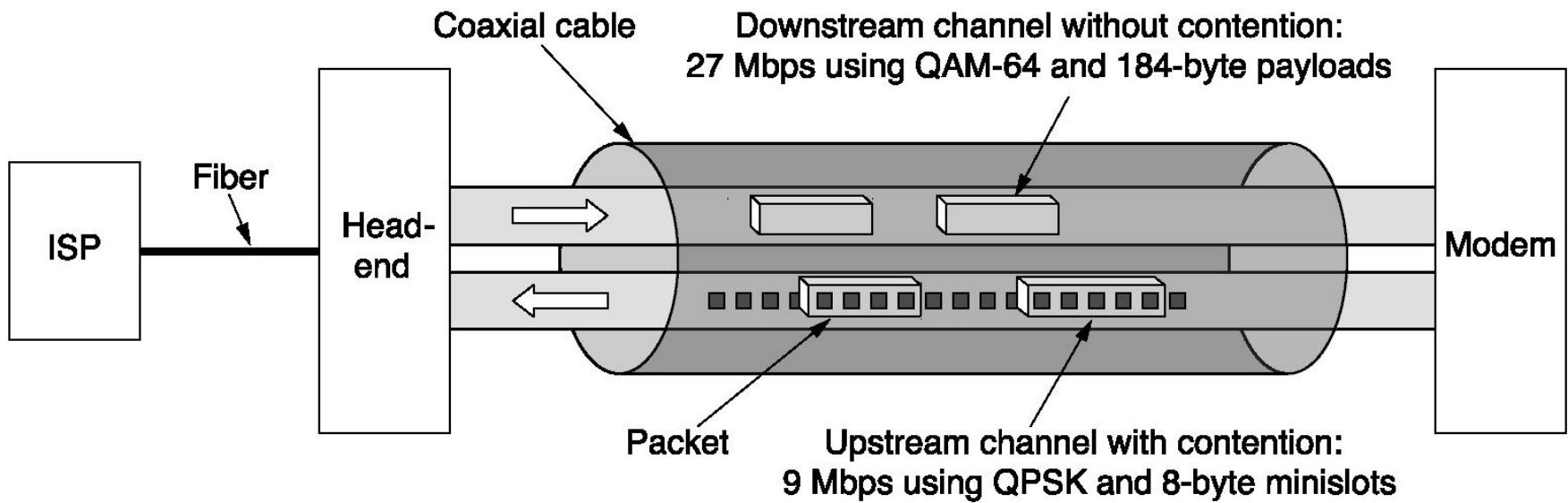
The fixed telephone system.

Spectrum Allocation



Frequency allocation in a typical cable TV system
used for Internet access

Cable Modems



Typical details of the upstream and downstream channels in North America.



Chapter 3

The Data Link Layer

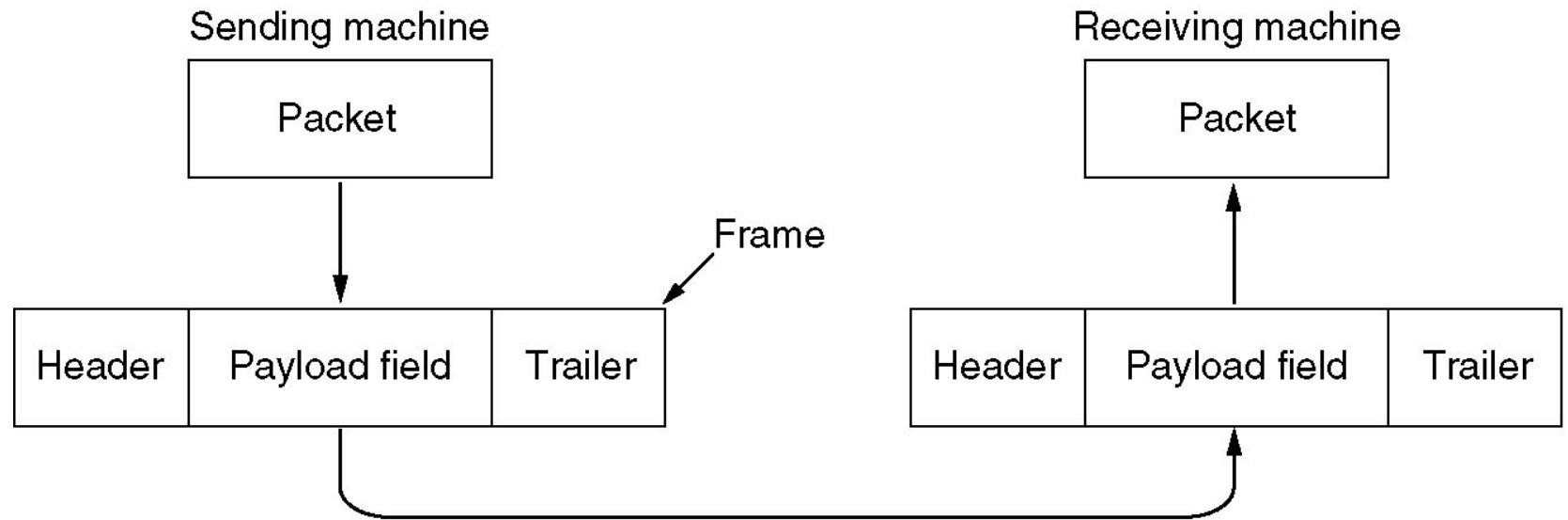
Data Link Layer Design Issues

- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control

Functions of the Data Link Layer

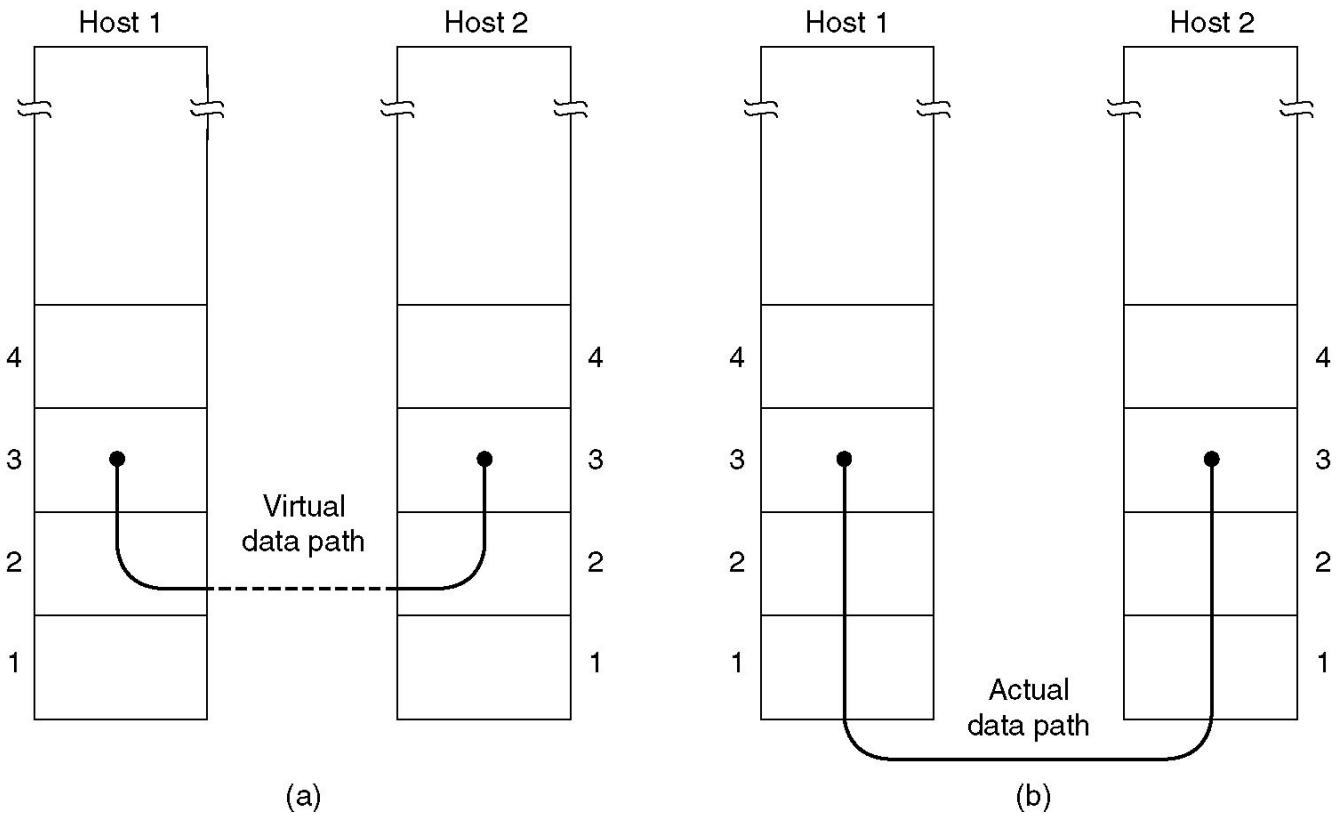
- Provide service interface to the network layer
- Dealing with transmission errors
- Regulating data flow
 - Slow receivers not swamped by fast senders

Functions of the Data Link Layer (2)



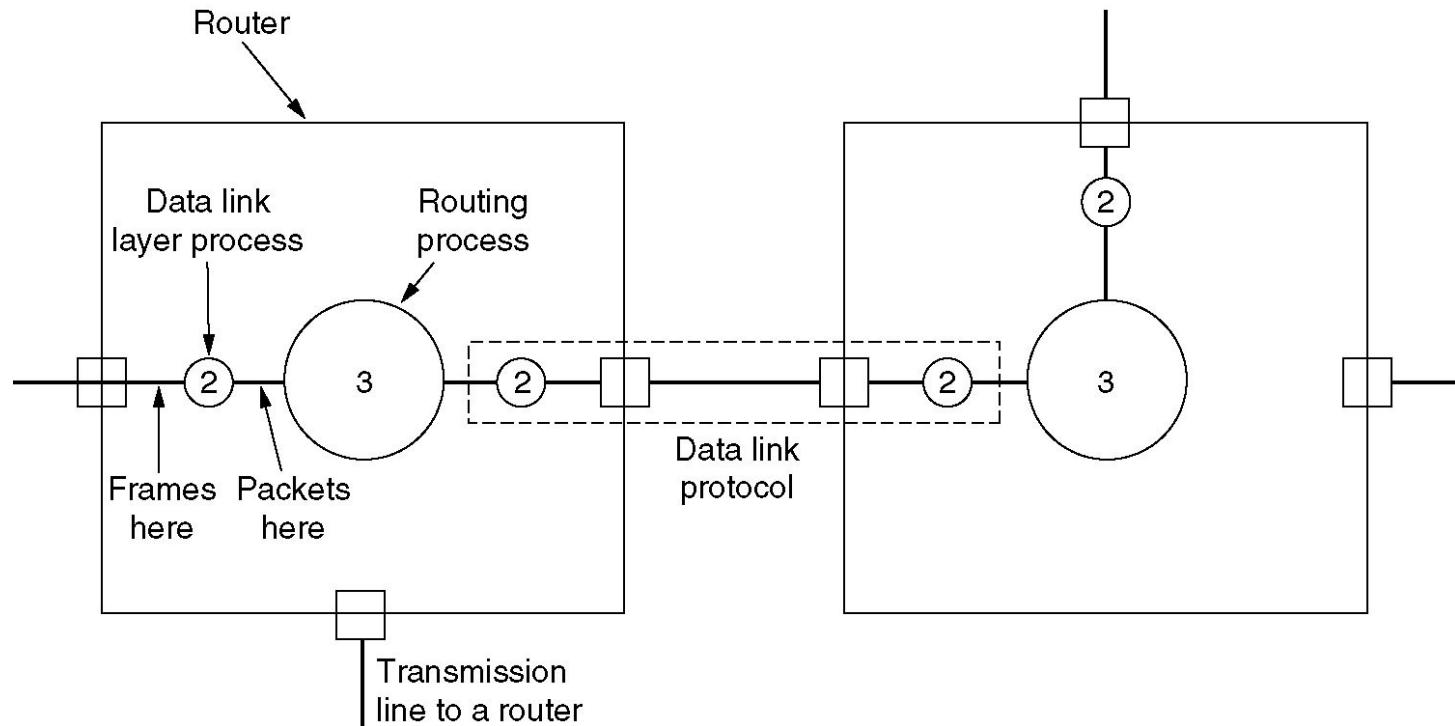
Relationship between packets and frames.

Services Provided to Network Layer



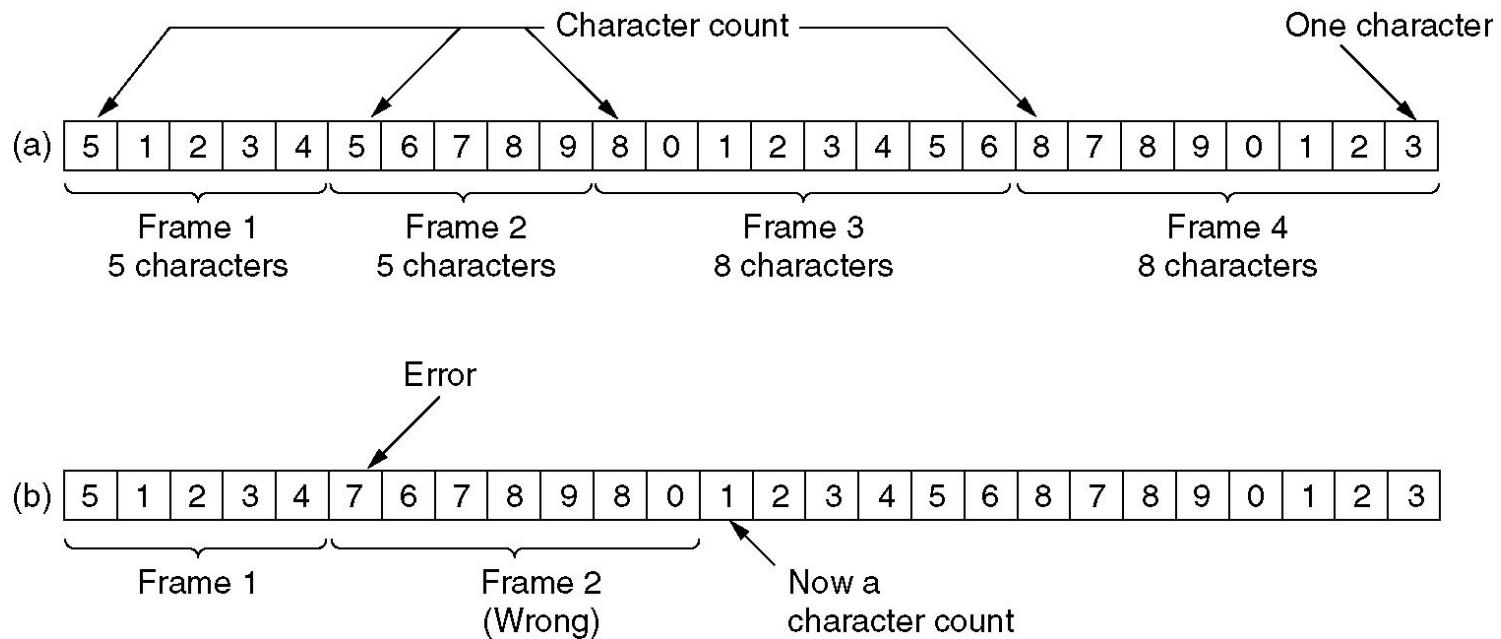
- (a) Virtual communication.
- (b) Actual communication.

Services Provided to Network Layer (2)



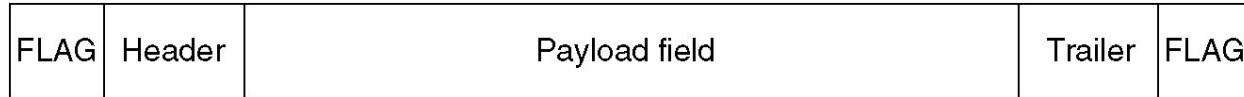
Placement of the data link protocol.

Framing

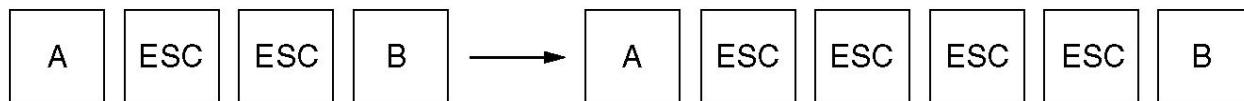
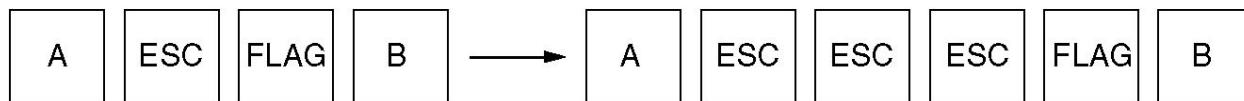
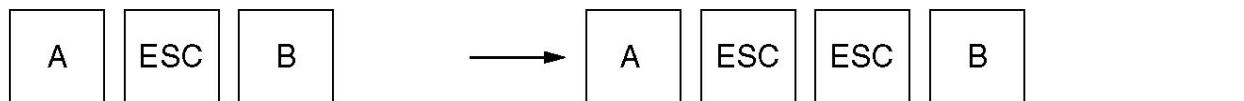
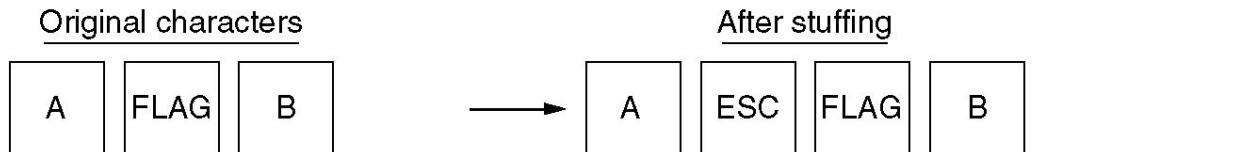


A character stream. (a) Without errors. (b) With one error.

Framing (2)



(a)



(b)

(a) A frame delimited by flag bytes.

(b) Four examples of byte sequences before and after stuffing.

Framing (3)

(a) 011011111111111111110010

(c) 0110111111111111110010

Bit stuffing

- (a) The original data.
 - (b) The data as they appear on the line.
 - (c) The data as they are stored in receiver's memory after destuffing.

Error Detection and Correction

- Error-Correcting Codes
- Error-Detecting Codes

Error-Correcting Codes

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

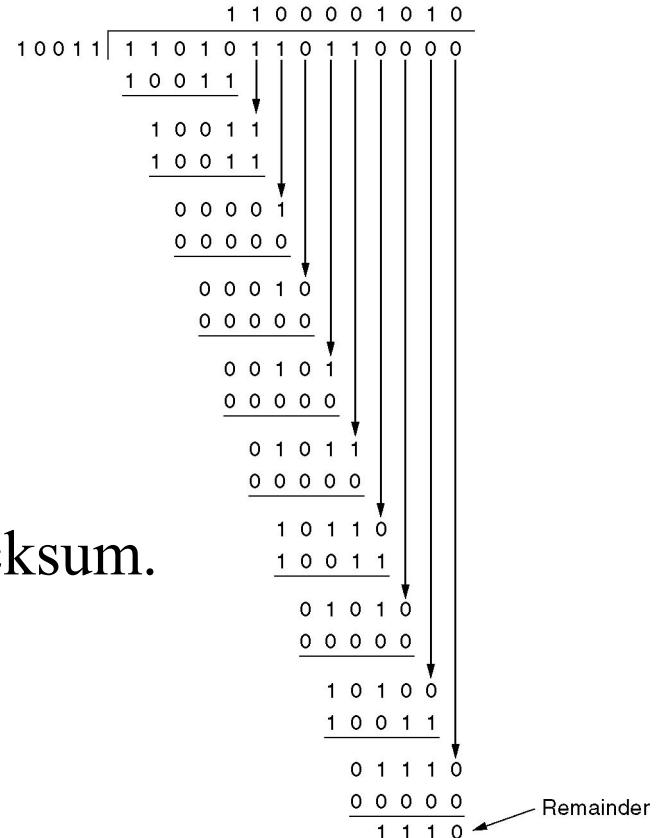
Use of a Hamming code to correct burst errors.

Error-Detecting Codes

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Calculation of the polynomial code checksum.

Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 0

Elementary Data Link Protocols

- An Unrestricted Simplex Protocol
- A Simplex Stop-and-Wait Protocol
- A Simplex Protocol for a Noisy Channel

Protocol Definitions

```
#define MAX_PKT 1024                                /* determines packet size in bytes */

typedef enum {false, true} boolean;                  /* boolean type */
typedef unsigned int seq_nr;                        /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind;           /* frame_kind definition */

typedef struct {
    frame_kind kind;                               /* frames are transported in this layer */
    seq_nr seq;                                    /* what kind of a frame is it? */
    seq_nr ack;                                    /* sequence number */
    packet info;                                   /* acknowledgement number */
} frame;                                         /* the network layer packet */
```

Continued →

Some definitions needed in the protocols to follow.
These are located in the file protocol.h.

Protocol Definitions (ctd.)

Some definitions
needed in the
protocols to follow.
These are located in
the file protocol.h.

```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Unrestricted Simplex Protocol

/* Protocol 1 (utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free, and the receiver is assumed to be able to process all the input infinitely quickly. Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. */

```
typedef enum {frame arrival} event type;
#include "protocol.h"

void sender1(void)
{
    frame s;                                /* buffer for an outbound frame */
    packet buffer;                          /* buffer for an outbound packet */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;                /* copy it into s for transmission */
        to_physical_layer(&s);         /* send it on its way */
    }                                         /* * Tomorrow, and tomorrow, and tomorrow,
                                                Creeps in this petty pace from day to day
                                                To the last syllable of recorded time
                                                - Macbeth, V, v */
}

void receiver1(void)
{
    frame r;
    event_type event;                      /* filled in by wait, but not used here */

    while (true) {
        wait_for_event(&event);          /* only possibility is frame_arrival */
        from_physical_layer(&r);        /* go get the inbound frame */
        to_network_layer(&r.info);      /* pass the data to the network layer */
    }
}
```

Simplex Stop-and-Wait Protocol

/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time, the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. */

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
    frame s;
    packet buffer;
    event_type event;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}

void receiver2(void)
{
    frame r, s;
    event_type event;
    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

/* buffer for an outbound frame */
/* buffer for an outbound packet */
/* frame_arrival is the only possibility */

/* go get something to send */
/* copy it into s for transmission */
/* bye bye little frame */
/* do not proceed until given the go ahead */

/* buffers for frames */
/* frame_arrival is the only possibility */

/* only possibility is frame_arrival */
/* go get the inbound frame */
/* pass the data to the network layer */
/* send a dummy frame to awaken sender */

A Simplex Protocol for a Noisy Channel

A positive acknowledgement with retransmission protocol.

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                                /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
    seq_nr next_frame_to_send;                  /* seq number of next outgoing frame */
    frame s;                                    /* scratch variable */
    packet buffer;                            /* buffer for an outbound packet */

    next_frame_to_send = 0;                     /* initialize outbound sequence numbers */
    from_network_layer(&buffer);             /* fetch first packet */

    while (true) {
        s.info = buffer;
        s.seq = next_frame_to_send;            /* construct a frame for transmission */
        to_physical_layer(&s);                /* insert sequence number in frame */
        start_timer(s.seq);                  /* send it on its way */
        wait_for_event(&event);              /* if answer takes too long, time out */
        if (event == frame_arrival) {          /* /* frame_arrival, cksum_err, timeout */
            from_physical_layer(&s);
            if (s.ack == next_frame_to_send) { /* get the acknowledgement */
                stop_timer(s.ack);
                from_network_layer(&buffer); /* turn the timer off */
                inc(next_frame_to_send);    /* get the next one to send */
                /* invert next_frame_to_send */
            }
        }
    }
}
```

Continued →

A Simplex Protocol for a Noisy Channel (ctd.)

```
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}
```

/* possibilities: frame_arrival, cksum_err */
/* a valid frame has arrived. */
/* go get the newly arrived frame */
/* this is what we have been waiting for. */
/* pass the data to the network layer */
/* next time expect the other sequence nr */

/* tell which frame is being acked */
/* send acknowledgement */

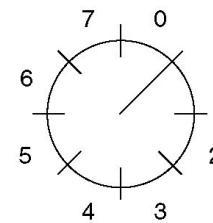
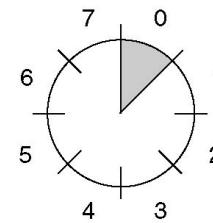
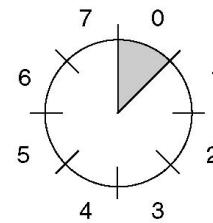
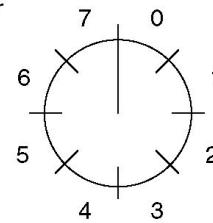
A positive acknowledgement with retransmission protocol.

Sliding Window Protocols

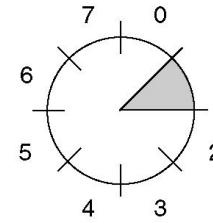
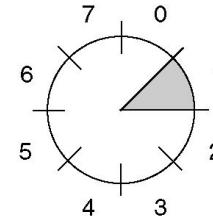
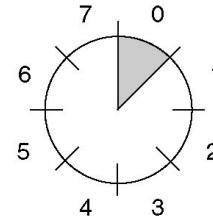
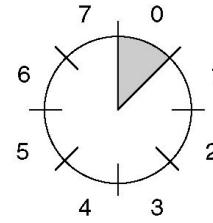
- A One-Bit Sliding Window Protocol
- A Protocol Using Go Back N
- A Protocol Using Selective Repeat

Sliding Window Protocols (2)

Sender



Receiver



(a)

(b)

(c)

(d)

A sliding window of size 1, with a 3-bit sequence number.

(a) Initially.

(b) After the first frame has been sent.

(c) After the first frame has been received.

(d) After the first acknowledgement has been received.

A One-Bit Sliding Window Protocol

```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
    seq_nr next_frame_to_send;                  /* 0 or 1 only */
    seq_nr frame_expected;                     /* 0 or 1 only */
    frame r, s;                               /* scratch variables */
    packet buffer;                            /* current packet being sent */
    event_type event;

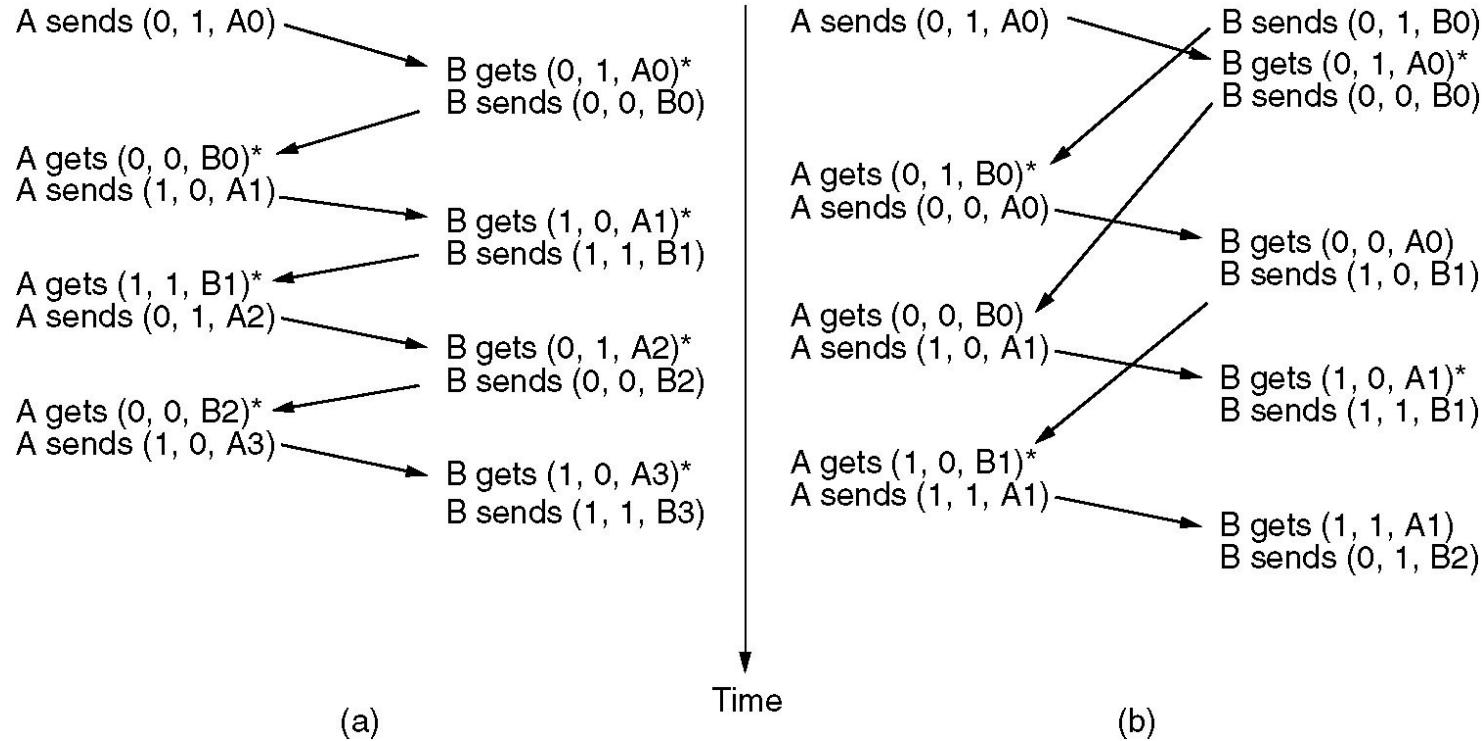
    next_frame_to_send = 0;                    /* next frame on the outbound stream */
    frame_expected = 0;                      /* frame expected next */
    from_network_layer(&buffer);            /* fetch a packet from the network layer */
    s.info = buffer;                          /* prepare to send the initial frame */
    s.seq = next_frame_to_send;               /* insert sequence number into frame */
    s.ack = 1 - frame_expected;              /* piggybacked ack */
    to_physical_layer(&s);                 /* transmit the frame */
    start_timer(s.seq);                     /* start the timer running */
```

Continued →

A One-Bit Sliding Window Protocol (ctd.)

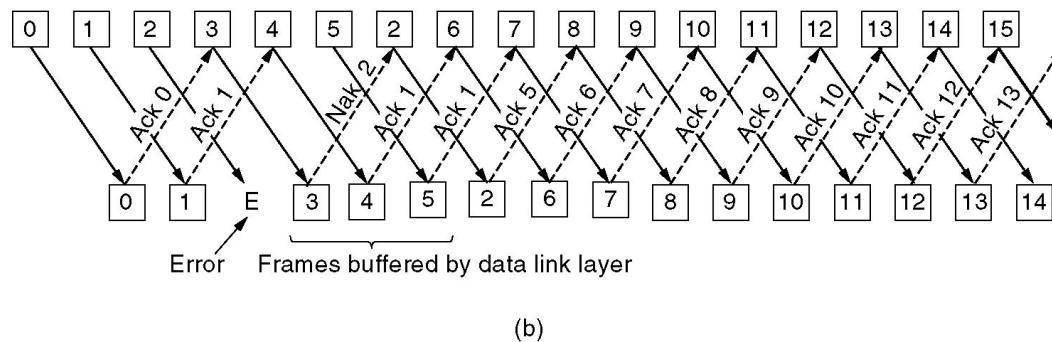
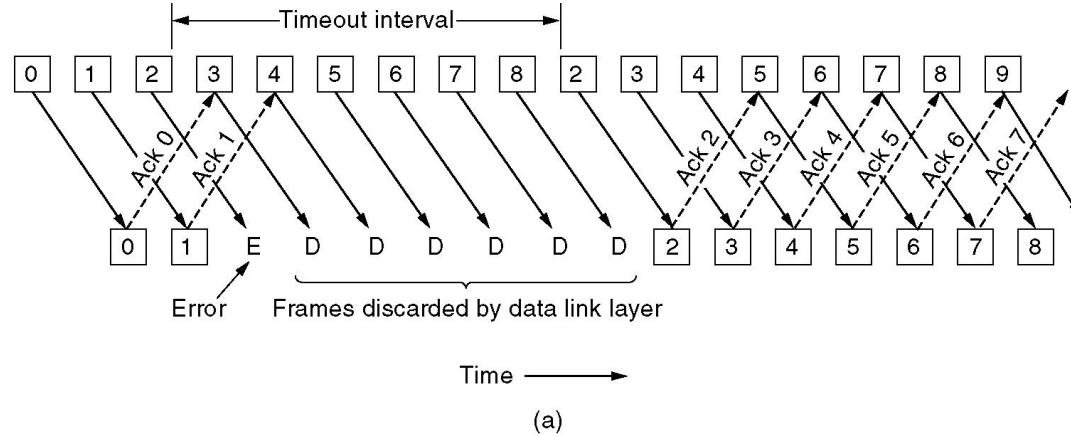
```
while (true) {
    wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&r);
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }
        if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
            stop_timer(r.ack); /* turn the timer off */
            from_network_layer(&buffer); /* fetch new pkt from network layer */
            inc(next_frame_to_send); /* invert sender's sequence number */
        }
        s.info = buffer; /* construct outbound frame */
        s.seq = next_frame_to_send; /* insert sequence number into it */
        s.ack = 1 - frame_expected; /* seq number of last received frame */
        to_physical_layer(&s); /* transmit a frame */
        start_timer(s.seq); /* start the timer running */
    }
}
```

A One-Bit Sliding Window Protocol (2)



Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

A Protocol Using Go Back N



Pipelining and error recovery. Effect on an error when

- (a) Receiver's window size is 1.
- (b) Receiver's window size is large.

Sliding Window Protocol Using Go Back N

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7           /* should be 2^n - 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
    /* Return true if a <= b < c circularly; false otherwise. */
    if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
    /* Construct and send a data frame. */
    frame s;                      /* scratch variable */

    s.info = buffer[frame_nr];      /* insert packet into frame */
    s.seq = frame_nr;               /* insert sequence number into frame */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1); /* piggyback ack */
    to_physical_layer(&s);         /* transmit the frame */
    start_timer(frame_nr);          /* start the timer running */
}
```

Continued →

Sliding Window Protocol Using Go Back N

```
void protocol5(void)
{
    seq_nr next_frame_to_send;          /* MAX_SEQ > 1; used for outbound stream */
    seq_nr ack_expected;               /* oldest frame as yet unacknowledged */
    seq_nr frame_expected;             /* next frame expected on inbound stream */
    frame r;                          /* scratch variable */
    packet buffer[MAX_SEQ + 1];        /* buffers for the outbound stream */
    seq_nr nbuffered;                 /* # output buffers currently in use */
    seq_nr i;                         /* used to index into the buffer array */

    enable_network_layer();           /* allow network_layer_ready events */
    ack_expected = 0;                 /* next ack expected inbound */
    next_frame_to_send = 0;            /* next frame going out */
    frame_expected = 0;               /* number of frame expected inbound */
    nbuffered = 0;                   /* initially no packets are buffered */
```

Continued →

Sliding Window Protocol Using Go Back N

```
while (true) {
    wait_for_event(&event);           /* four possibilities: see event_type above */

    switch(event) {
        case network_layer_ready:    /* the network layer has a packet to send */
            /* Accept, save, and transmit a new frame. */
            from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
            nbuffered = nbuffered + 1; /* expand the sender's window */
            send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
            inc(next_frame_to_send); /* advance sender's upper window edge */
            break;

        case frame_arrival:          /* a data or control frame has arrived */
            from_physical_layer(&r); /* get incoming frame from physical layer */

            if (r.seq == frame_expected) {
                /* Frames are accepted only in order. */
                to_network_layer(&r.info); /* pass packet to network layer */
                inc(frame_expected); /* advance lower edge of receiver's window */
            }
    }
}
```

Continued →

Sliding Window Protocol Using Go Back N

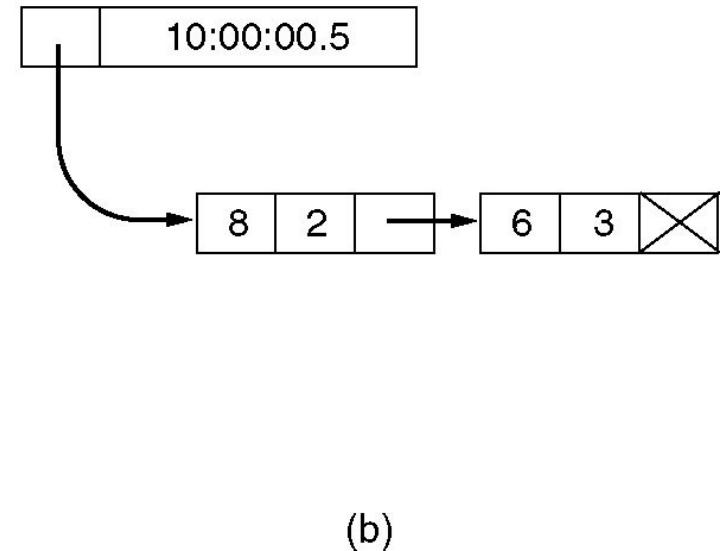
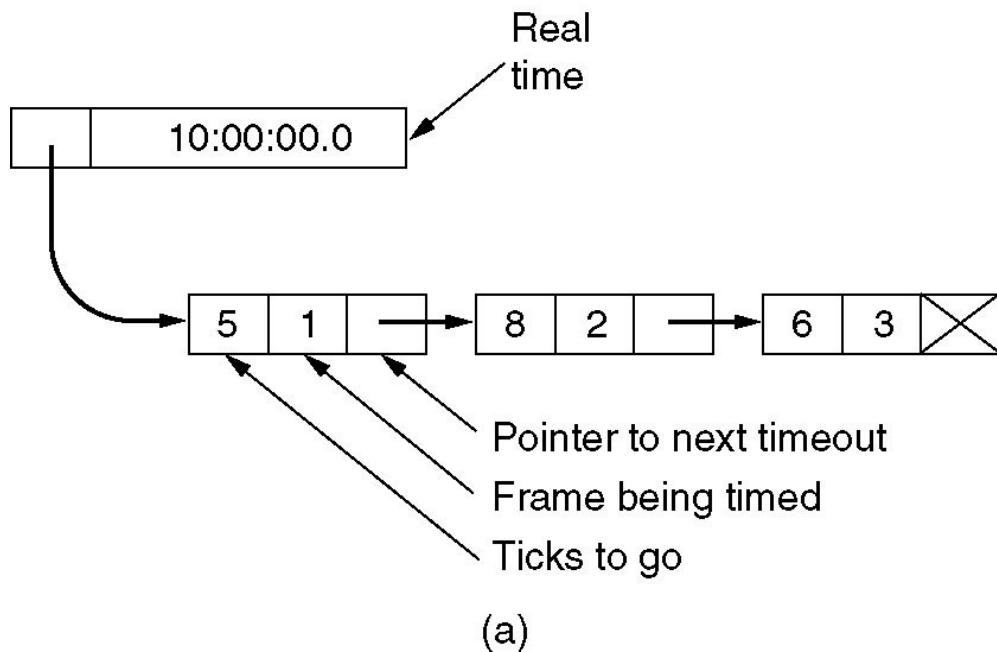
```
/* Ack n implies n - 1, n - 2, etc. Check for this. */
while (between(ack_expected, r.ack, next_frame_to_send)) {
    /* Handle piggybacked ack. */
    nbuffered = nbuffered - 1; /* one frame fewer buffered */
    stop_timer(ack_expected); /* frame arrived intact; stop timer */
    inc(ack_expected);      /* contract sender's window */
}
break;

case cksum_err: break;          /* just ignore bad frames */

case timeout:                  /* trouble; retransmit all outstanding frames */
    next_frame_to_send = ack_expected; /* start retransmitting here */
    for (i = 1; i <= nbuffered; i++) {
        send_data(next_frame_to_send, frame_expected, buffer); /* resend 1 frame */
        inc(next_frame_to_send); /* prepare to send the next one */
    }
}

if (nbuffered < MAX_SEQ)
    enable_network_layer();
else
    disable_network_layer();
}
```

Sliding Window Protocol Using Go Back N (2)



Simulation of multiple timers in software.

A Sliding Window Protocol Using Selective Repeat

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                /* should be 2^n - 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                         /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;             /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
    return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
    frame s;                                     /* scratch variable */

    s.kind = fk;                                  /* kind == data, ack, or nak */
    if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
    s.seq = frame_nr;                            /* only meaningful for data frames */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    if (fk == nak) no_nak = false;                /* one nak per frame, please */
    to_physical_layer(&s);                      /* transmit the frame */
    if (fk == data) start_timer(frame_nr % NR_BUFS);
    stop_ack_timer();                            /* no need for separate ack frame */
}
```

Continued →

A Sliding Window Protocol Using Selective Repeat (2)

```
void protocol6(void)
{
    seq_nr ack_expected;                                /* lower edge of sender's window */
    seq_nr next_frame_to_send;                          /* upper edge of sender's window + 1 */
    seq_nr frame_expected;                             /* lower edge of receiver's window */
    seq_nr too_far;                                    /* upper edge of receiver's window + 1 */
    int i;                                            /* index into buffer pool */
    frame r;                                         /* scratch variable */
    packet out_buf[NR_BUFS];                           /* buffers for the outbound stream */
    packet in_buf[NR_BUFS];                            /* buffers for the inbound stream */
    boolean arrived[NR_BUFS];                          /* inbound bit map */
    seq_nr nbuffered;                                 /* how many output buffers currently used */

    enable_network_layer();                           /* initialize */
    ack_expected = 0;                                  /* next ack expected on the inbound stream */
    next_frame_to_send = 0;                            /* number of next outgoing frame */
    frame_expected = 0;
    too_far = NR_BUFS;
    nbuffered = 0;                                    /* initially no packets are buffered */
    for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

Continued →

A Sliding Window Protocol Using Selective Repeat (3)

```
while (true) {
    wait_for_event(&event);                                /* five possibilities: see event_type above */
    switch(event) {
        case network_layer_ready:                         /* accept, save, and transmit a new frame */
            nbuffered = nbuffered + 1;                      /* expand the window */
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
            send_frame(data, next_frame_to_send, frame_expected, out_buf); /* transmit the frame */
            inc(next_frame_to_send);                          /* advance upper window edge */
            break;

        case frame_arrival:                               /* a data or control frame has arrived */
            from_physical_layer(&r);                     /* fetch incoming frame from physical layer */
            if (r.kind == data) {
                /* An undamaged frame has arrived. */
                if ((r.seq != frame_expected) && no_nak)
                    send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
                if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                    /* Frames may be accepted in any order. */
                    arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
                    in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
                    while (arrived[frame_expected % NR_BUFS]) {
                        /* Pass frames and advance window. */
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);    /* advance lower edge of receiver's window */
                        inc(too_far);          /* advance upper edge of receiver's window */
                        start_ack_timer();    /* to see if a separate ack is needed */
                    }
                }
            }
        }
    }
}
```

Continued →

A Sliding Window Protocol Using Selective Repeat (4)

```
if((r.kind==nak) && between(ack_expected,(r.ack+1)%MAX_SEQ+1),next frame to send))
    send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);
```

```
while (between(ack_expected, r.ack, next_frame_to_send)) {
    nbuffered = nbuffered - 1;           /* handle piggybacked ack */
    stop_timer(ack_expected % NR_BUFS);  /* frame arrived intact */
    inc(ack_expected);                  /* advance lower edge of sender's window */
}
break;
```

```
case cksum_err:
    if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
    break;
```

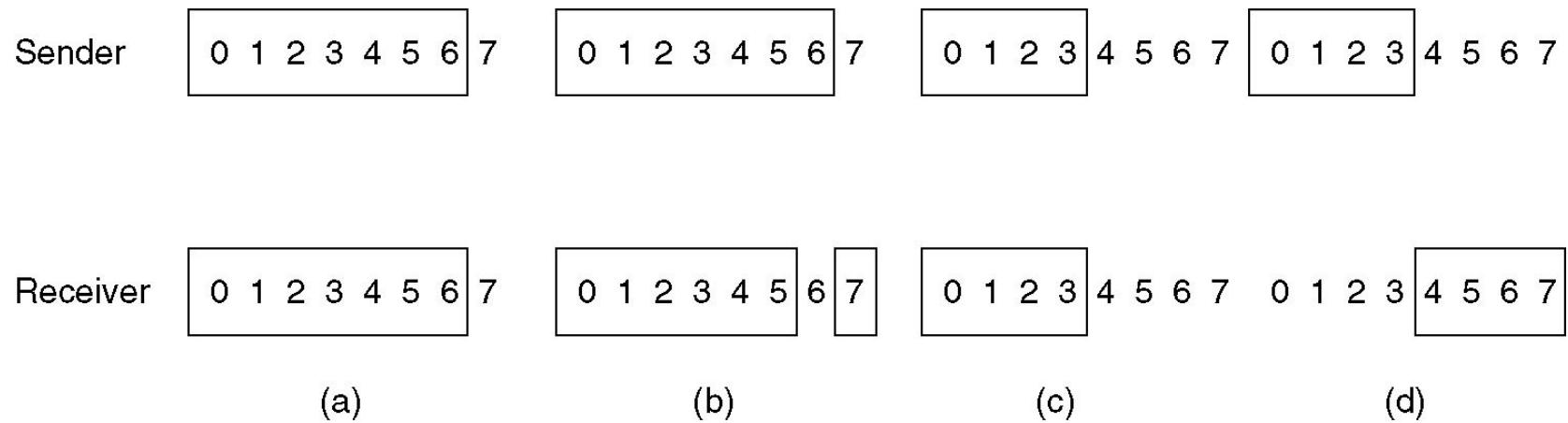
```
case timeout:
    send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
    break;
```

```
case ack_timeout:
    send_frame(ack,0,frame_expected, out_buf);    /* ack timer expired; send ack */
}
```

```
if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
```

```
}
```

A Sliding Window Protocol Using Selective Repeat (5)

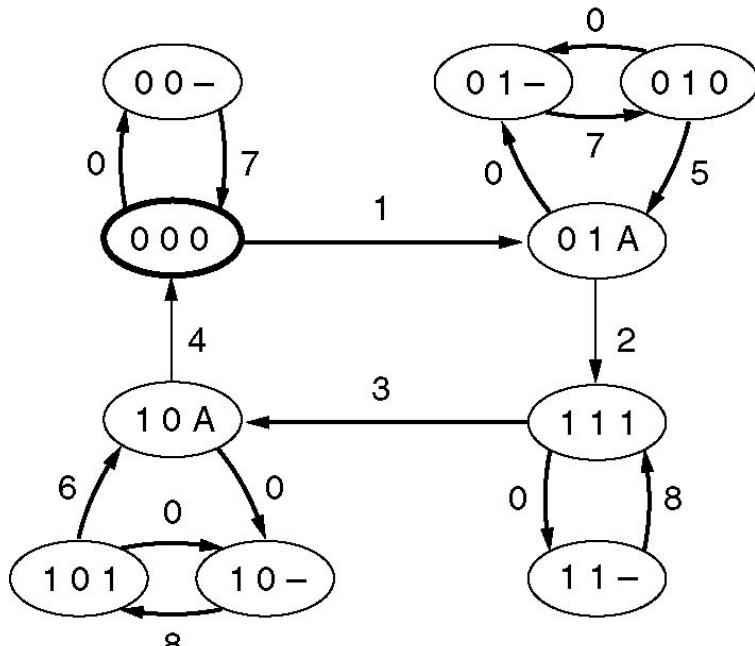


- (a) Initial situation with a window size seven.
- (b) After seven frames sent and received, but not acknowledged.
- (c) Initial situation with a window size of four.
- (d) After four frames sent and received, but not acknowledged.

Protocol Verification

- Finite State Machined Models
- Petri Net Models

Finite State Machined Models



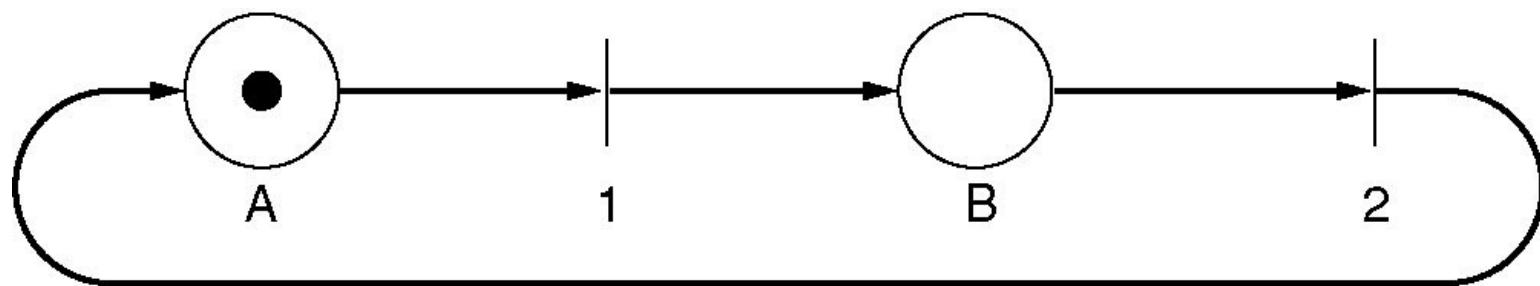
(a)

Transition	Who runs?	Frame accepted	Frame emitted	To network layer
0	-	(frame lost)		-
1	R	0	A	Yes
2	S	A	1	-
3	R	1	A	Yes
4	S	A	0	-
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	-
8	S	(timeout)	1	-

(b)

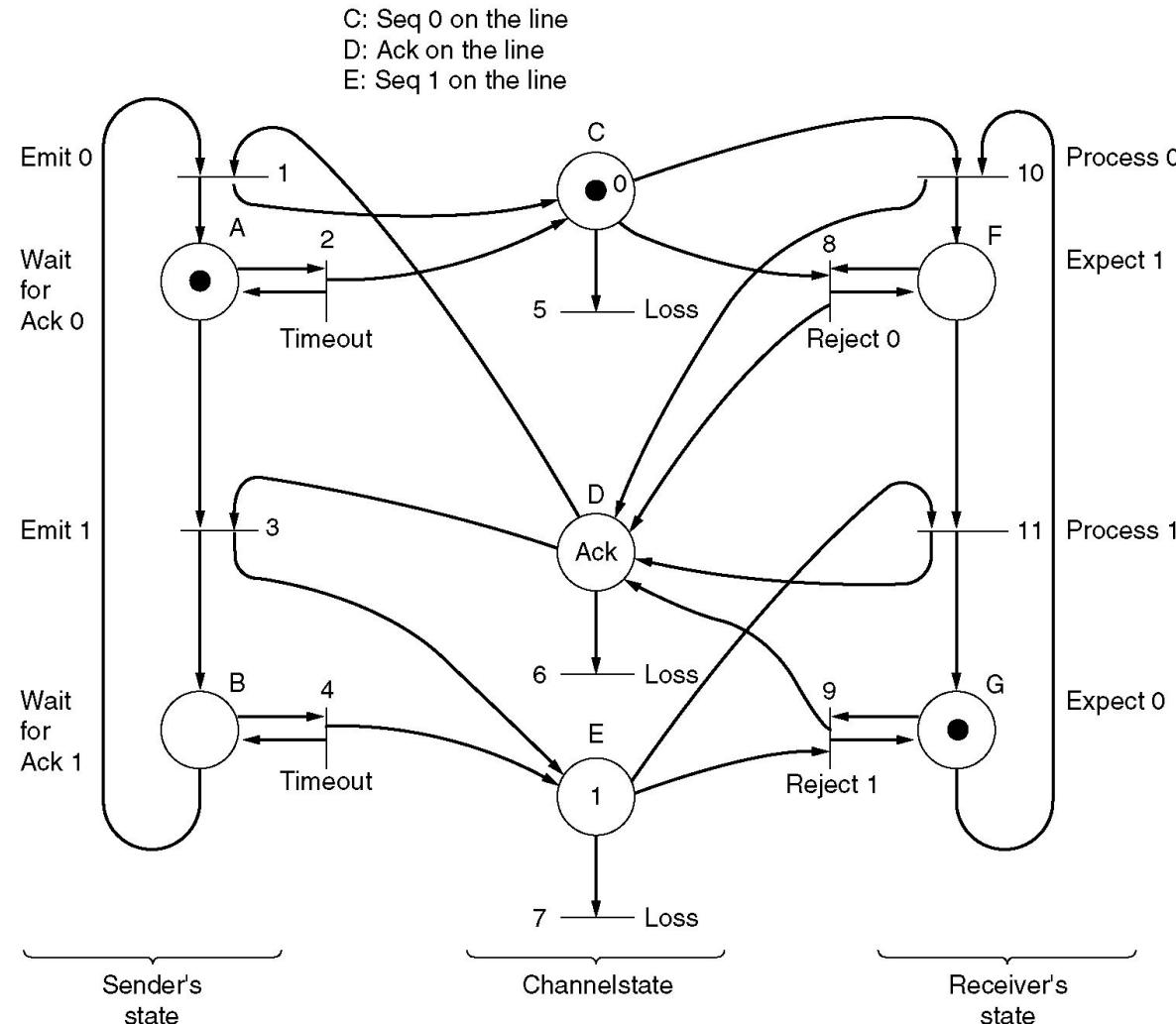
(a) State diagram for protocol 3. (b) Transmissions.

Petri Net Models



A Petri net with two places and two transitions.

Petri Net Models (2)



A Petri net model for protocol 3.

Example Data Link Protocols

- HDLC – High-Level Data Link Control
- The Data Link Layer in the Internet

High-Level Data Link Control

Bits	8	8	8	≥ 0	16	8
	0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0

Frame format for bit-oriented protocols.

High-Level Data Link Control (2)

Bits	1	3	1	3
(a)	0	Seq	P/F	Next

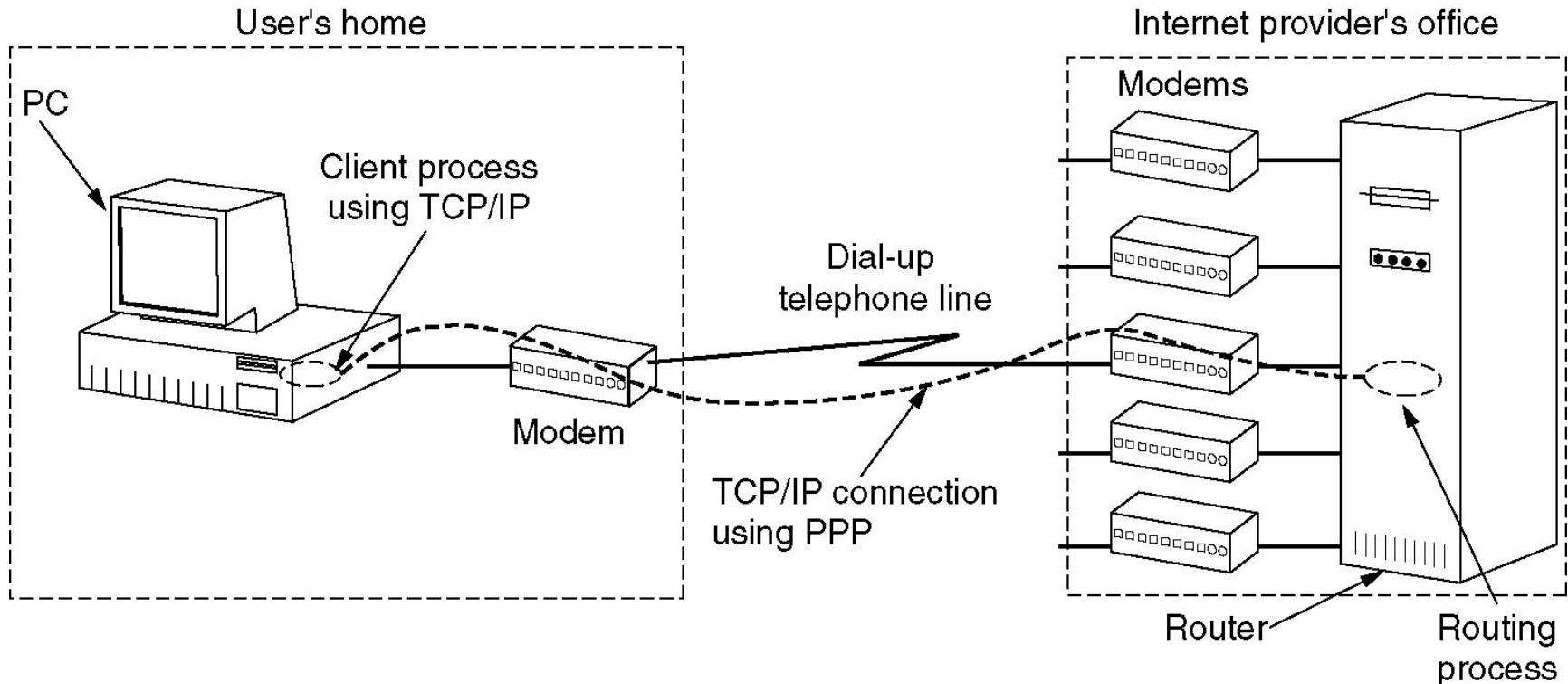
(b)	1	0	Type	P/F	Next
-----	---	---	------	-----	------

(c)	1	1	Type	P/F	Modifier
-----	---	---	------	-----	----------

Control field of

- (a) An information frame.
- (b) A supervisory frame.
- (c) An unnumbered frame.

The Data Link Layer in the Internet



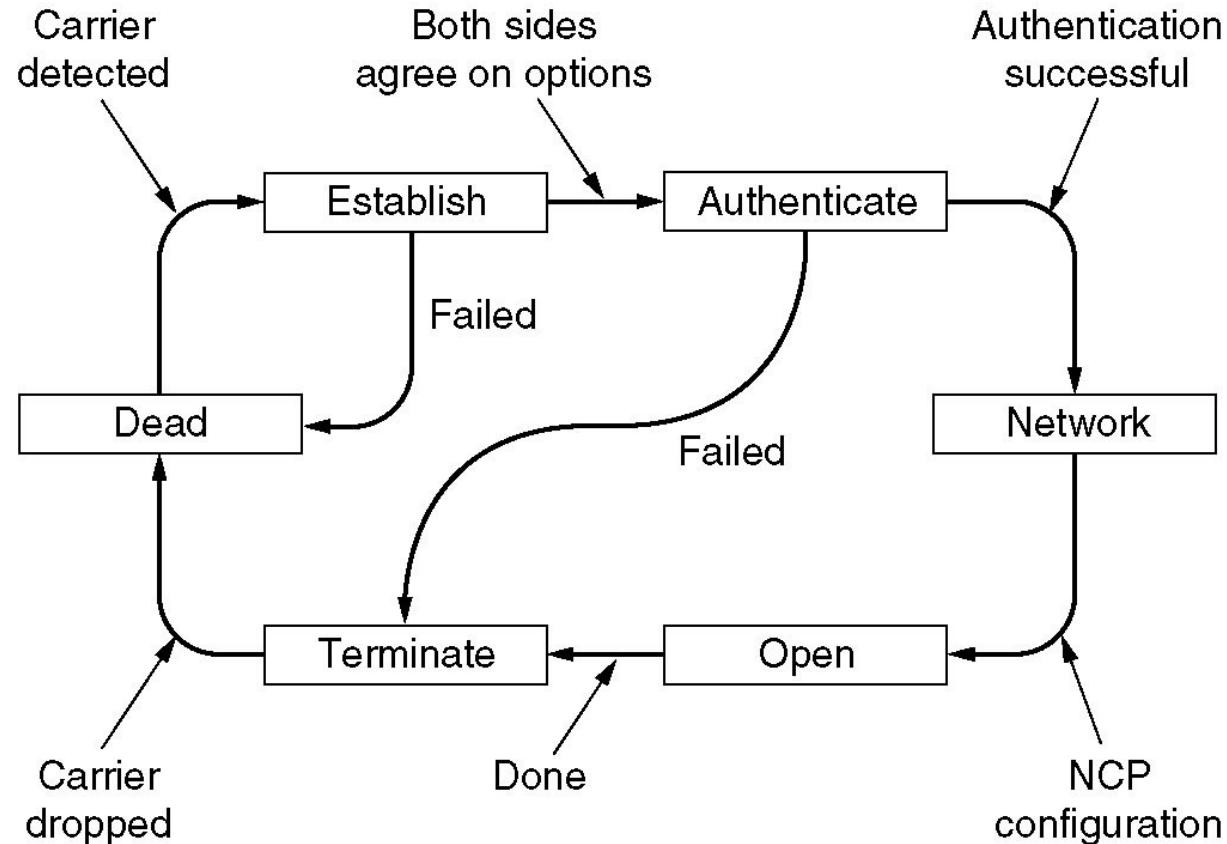
A home personal computer acting as an internet host.

PPP – Point to Point Protocol

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload }}	Checksum	Flag 01111110

The PPP full frame format for unnumbered mode operation.

PPP – Point to Point Protocol (2)



A simplified phase diagram for bringing a line up and down.

PPP – Point to Point Protocol (3)

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

The LCP frame types.



Chapter 4

The Medium Access Control Sublayer

The Channel Allocation Problem

- Static Channel Allocation in LANs and MANs
- Dynamic Channel Allocation in LANs and MANs

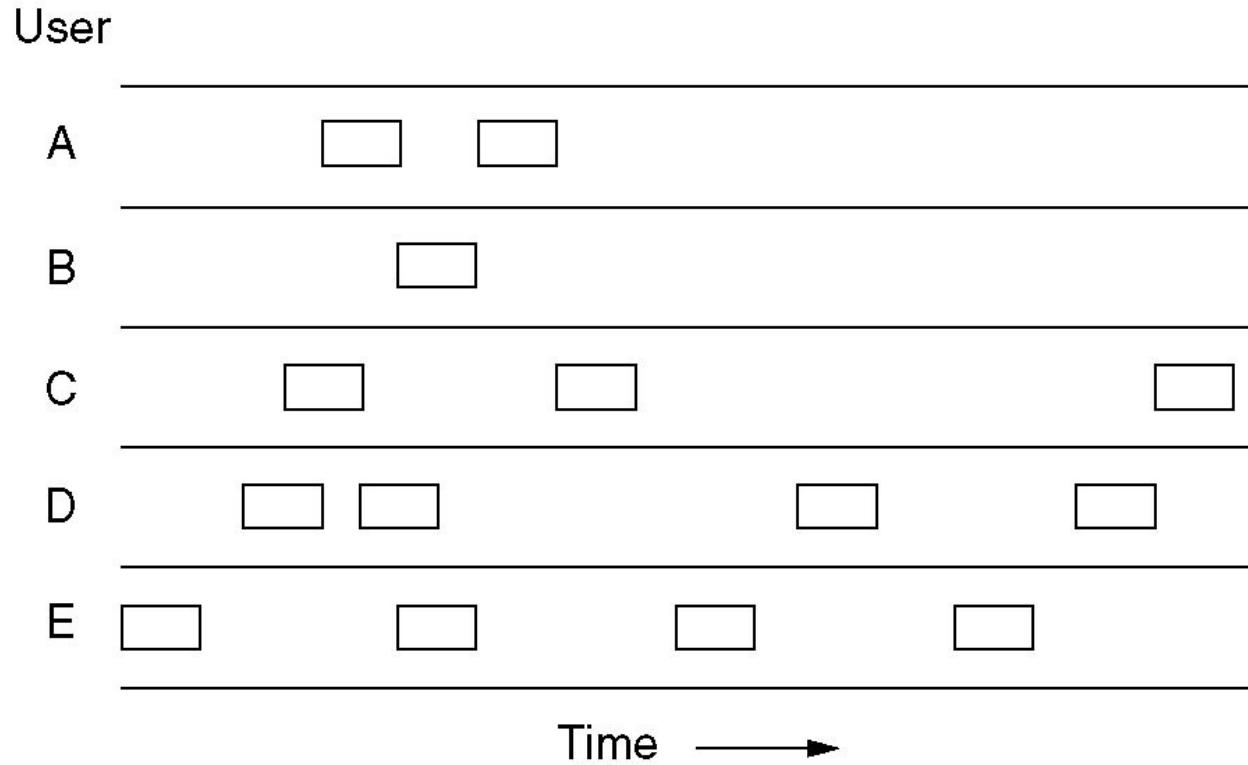
Dynamic Channel Allocation in LANs and MANs

- Station Model.
- Single Channel Assumption.
- Collision Assumption.
- (a) Continuous Time.
(b) Slotted Time.
- (a) Carrier Sense.
(b) No Carrier Sense.

Multiple Access Protocols

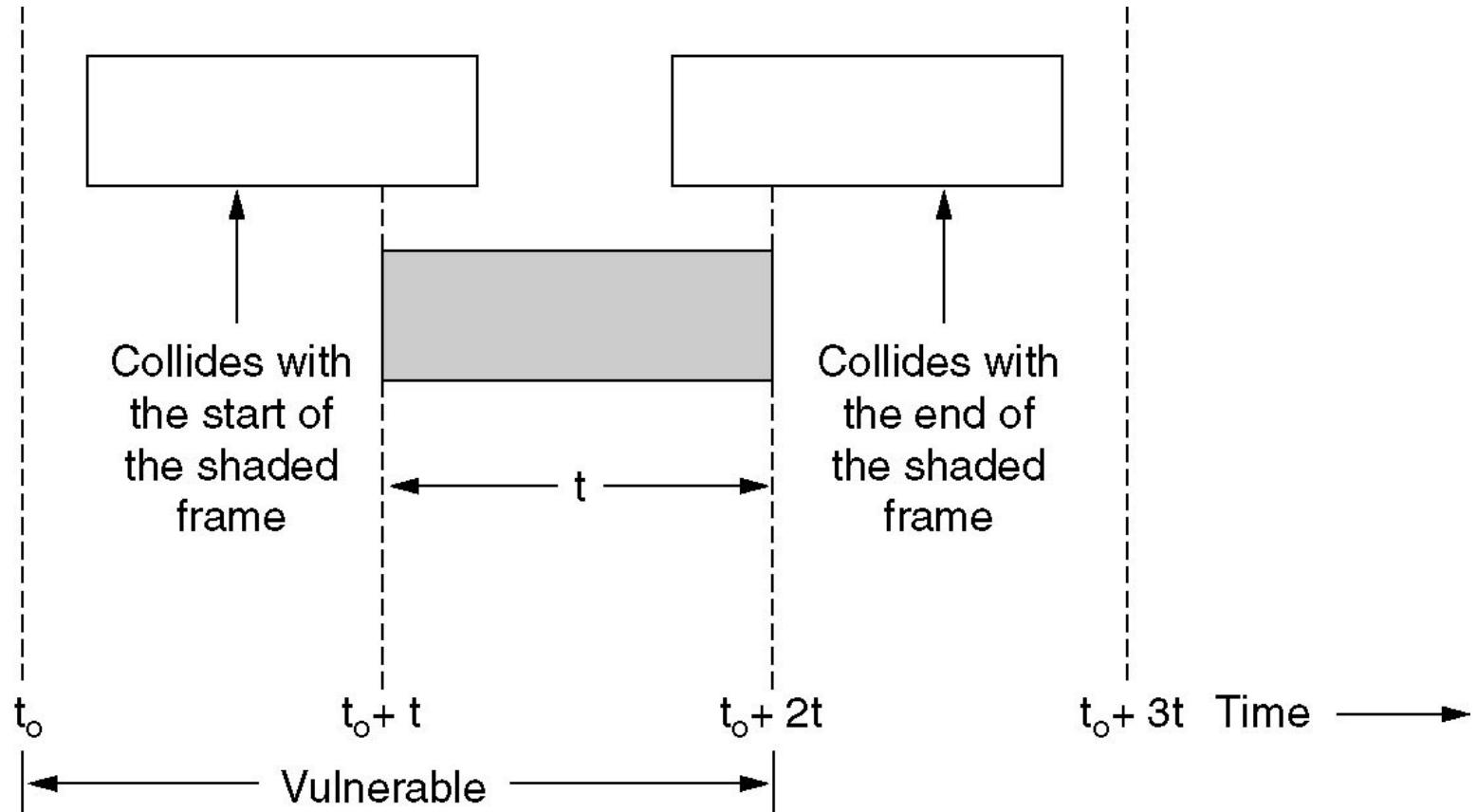
- ALOHA
- Carrier Sense Multiple Access Protocols
- Collision-Free Protocols
- Limited-Contention Protocols
- Wavelength Division Multiple Access Protocols
- Wireless LAN Protocols

Pure ALOHA



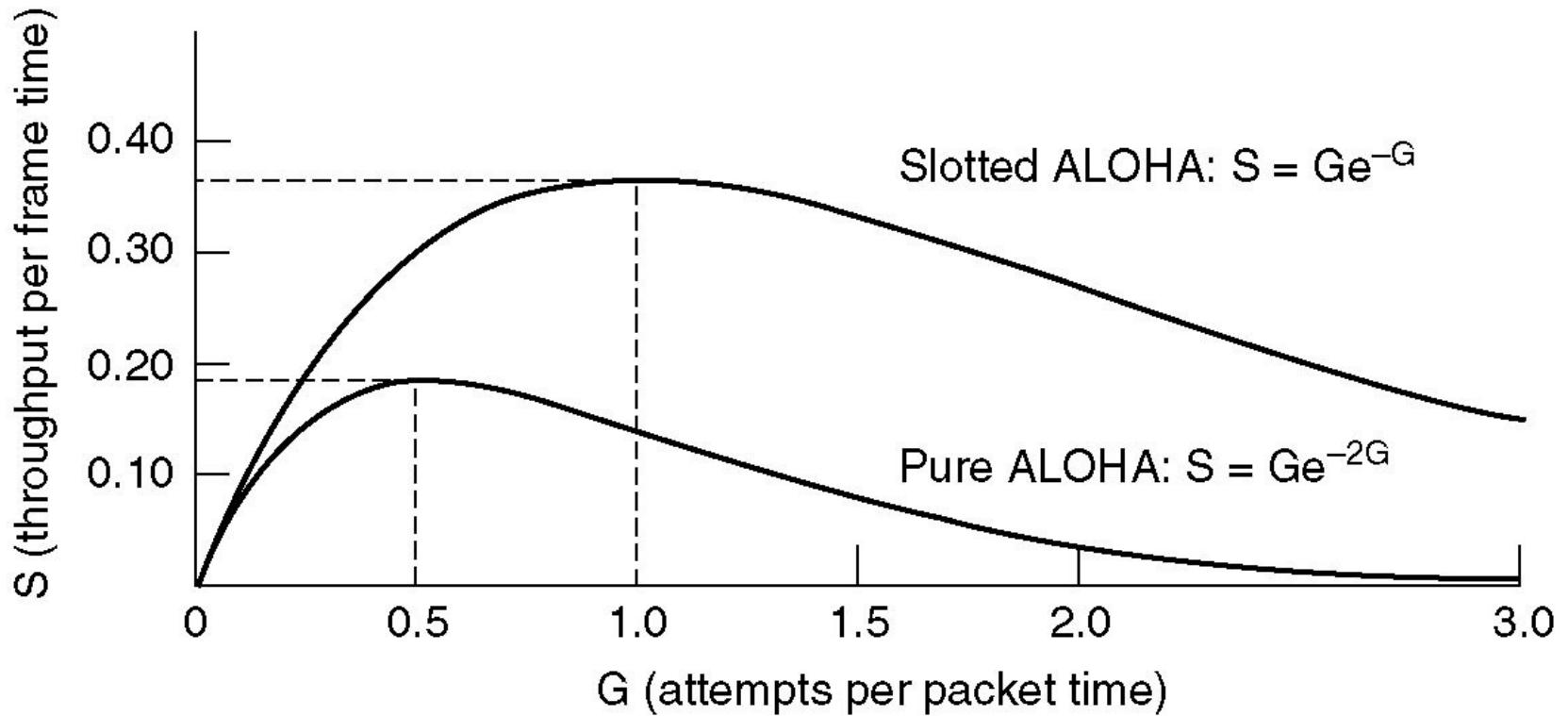
In pure ALOHA, frames are transmitted at completely arbitrary times.

Pure ALOHA (2)



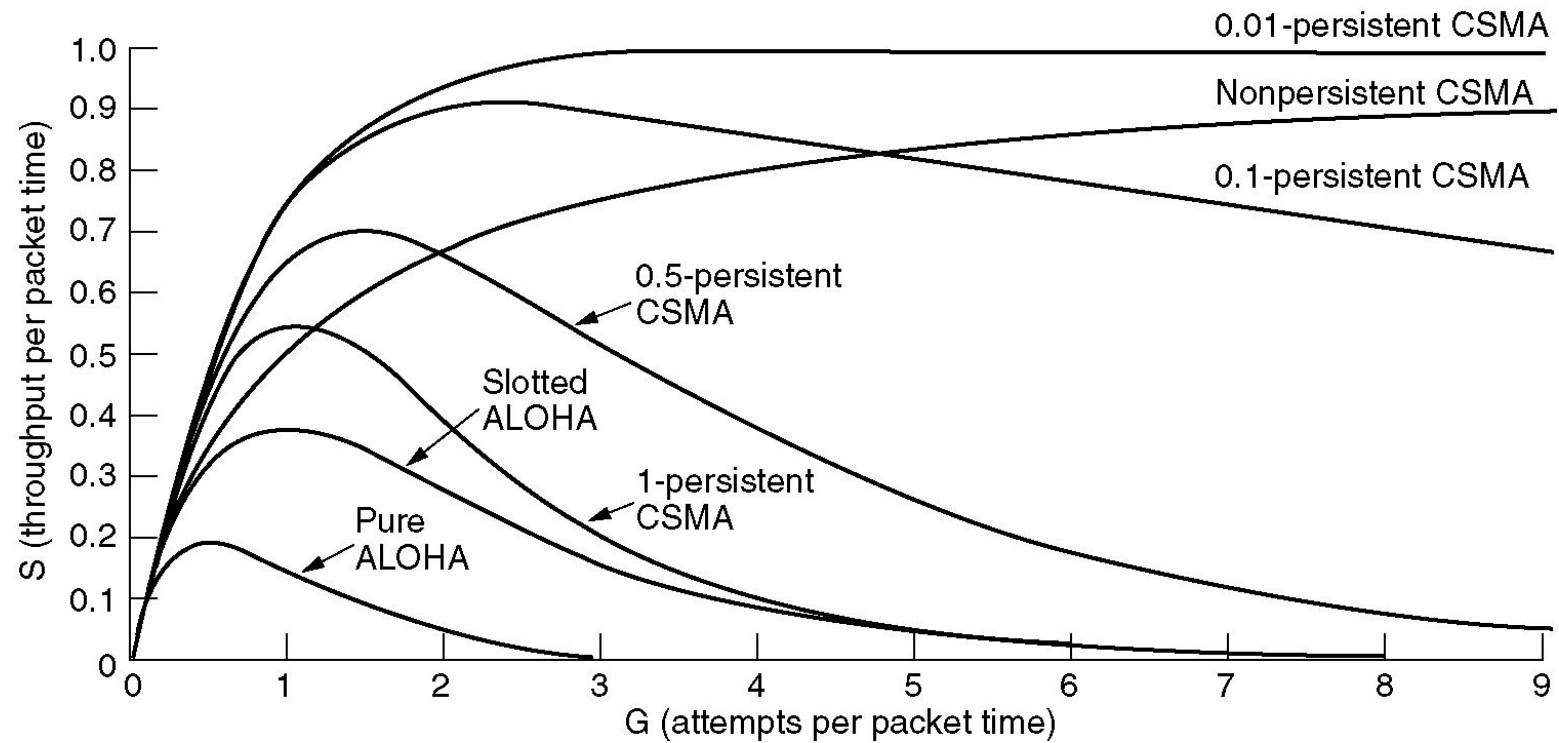
Vulnerable period for the shaded frame.

Pure ALOHA (3)



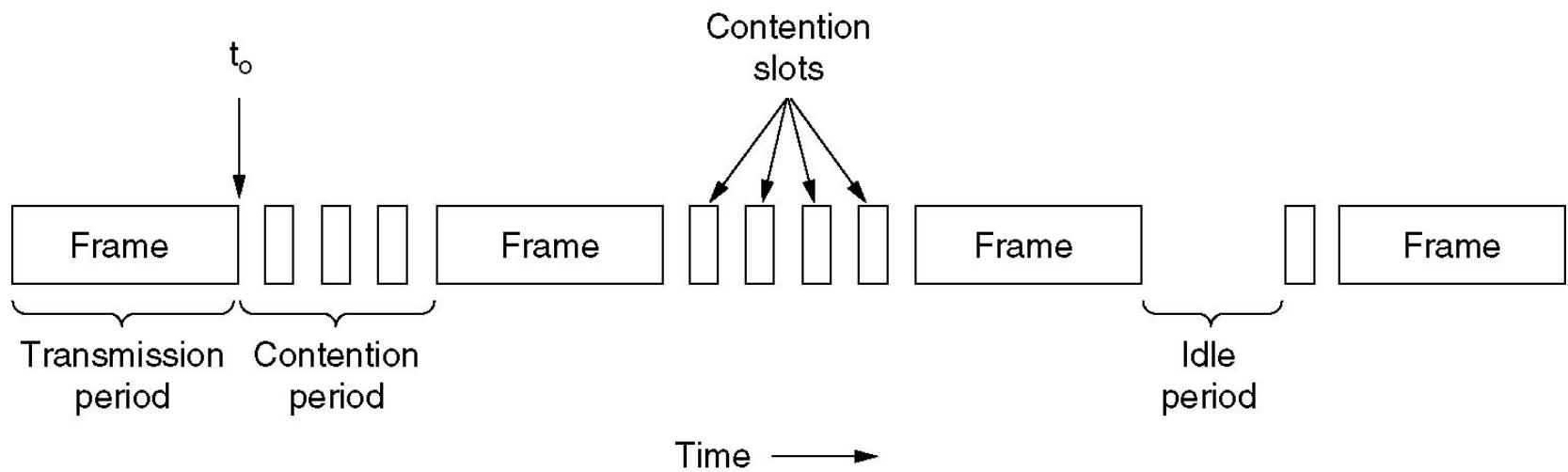
Throughput versus offered traffic for ALOHA systems.

Persistent and Nonpersistent CSMA



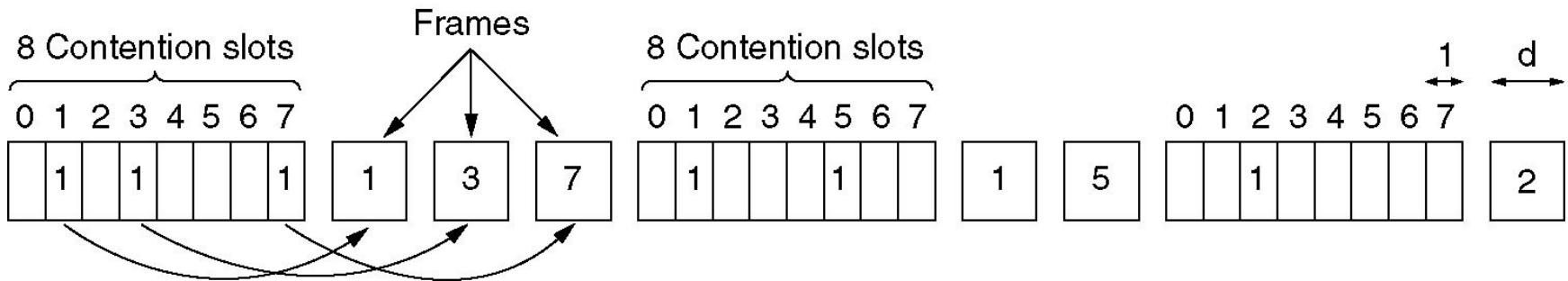
Comparison of the channel utilization versus load for various random access protocols.

CSMA with Collision Detection



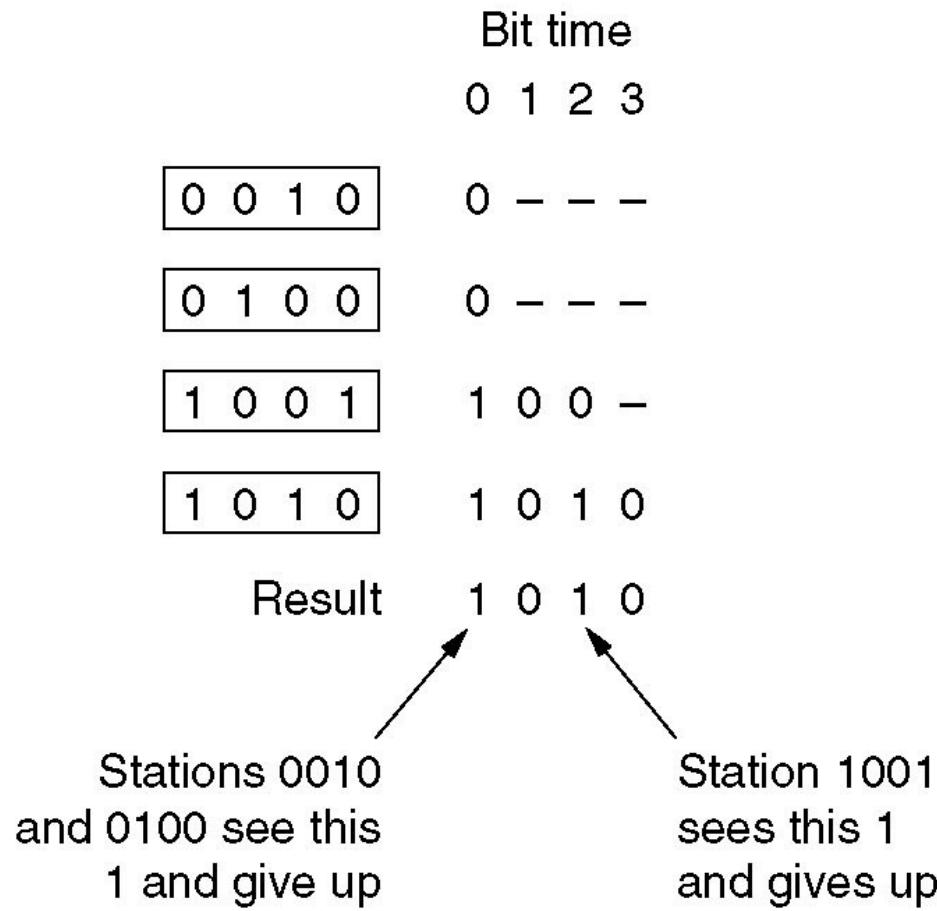
CSMA/CD can be in one of three states: contention, transmission, or idle.

Collision-Free Protocols



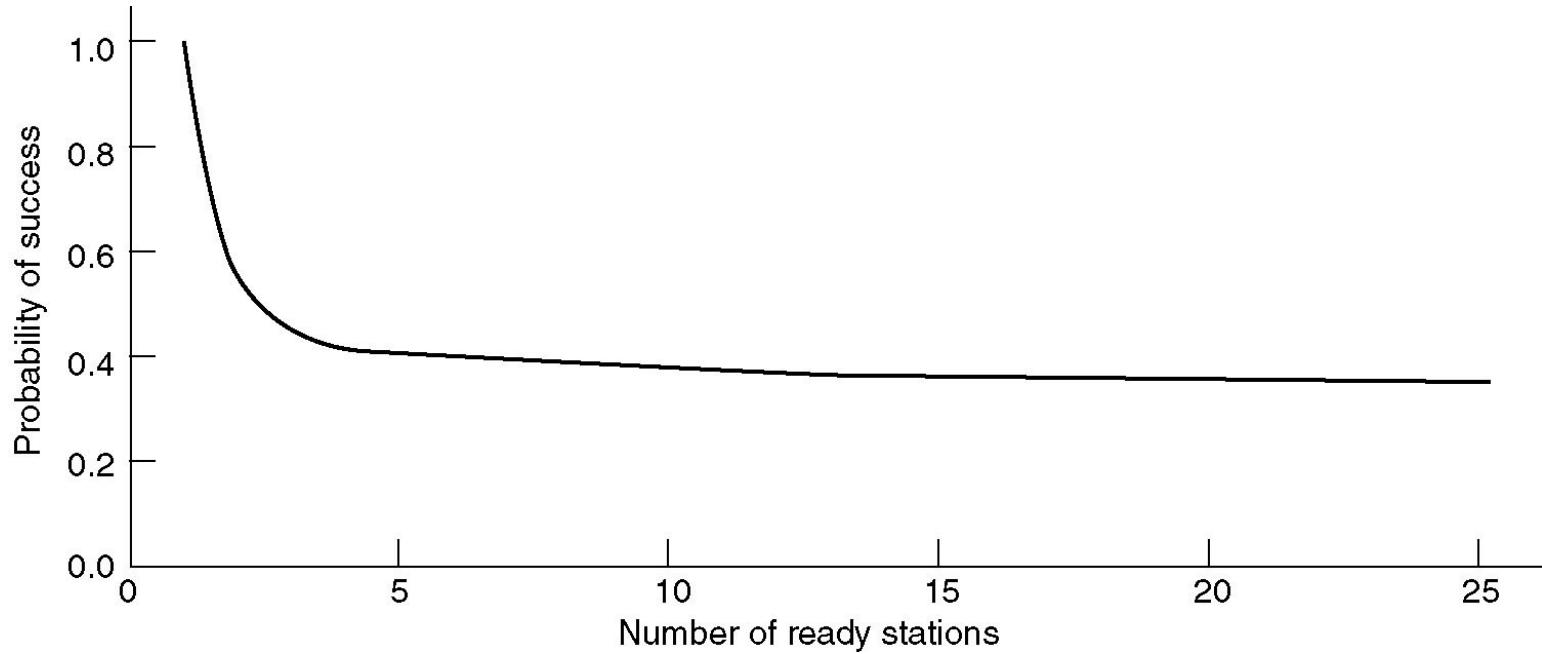
The basic bit-map protocol.

Collision-Free Protocols (2)



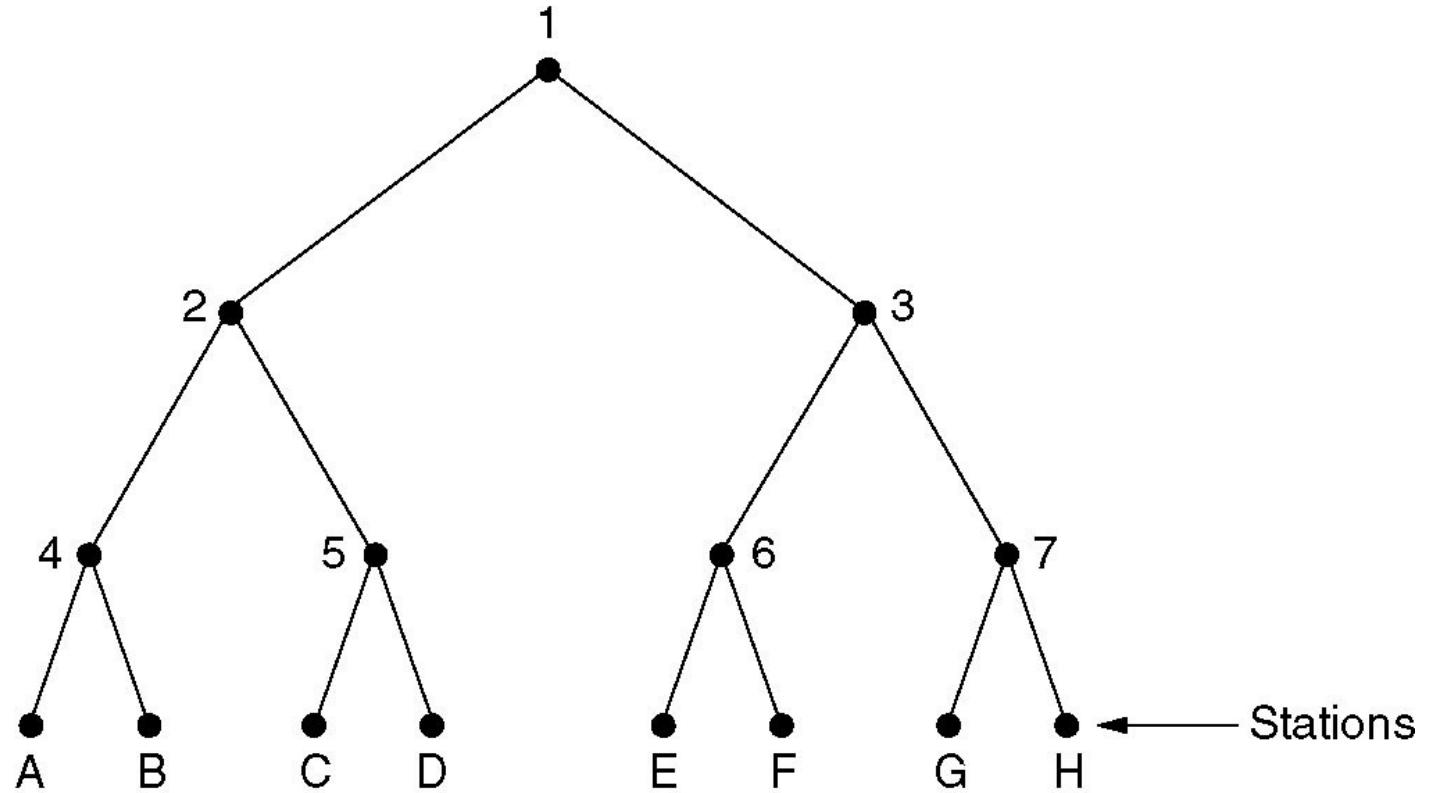
The binary countdown protocol. A dash indicates silence.

Limited-Contention Protocols



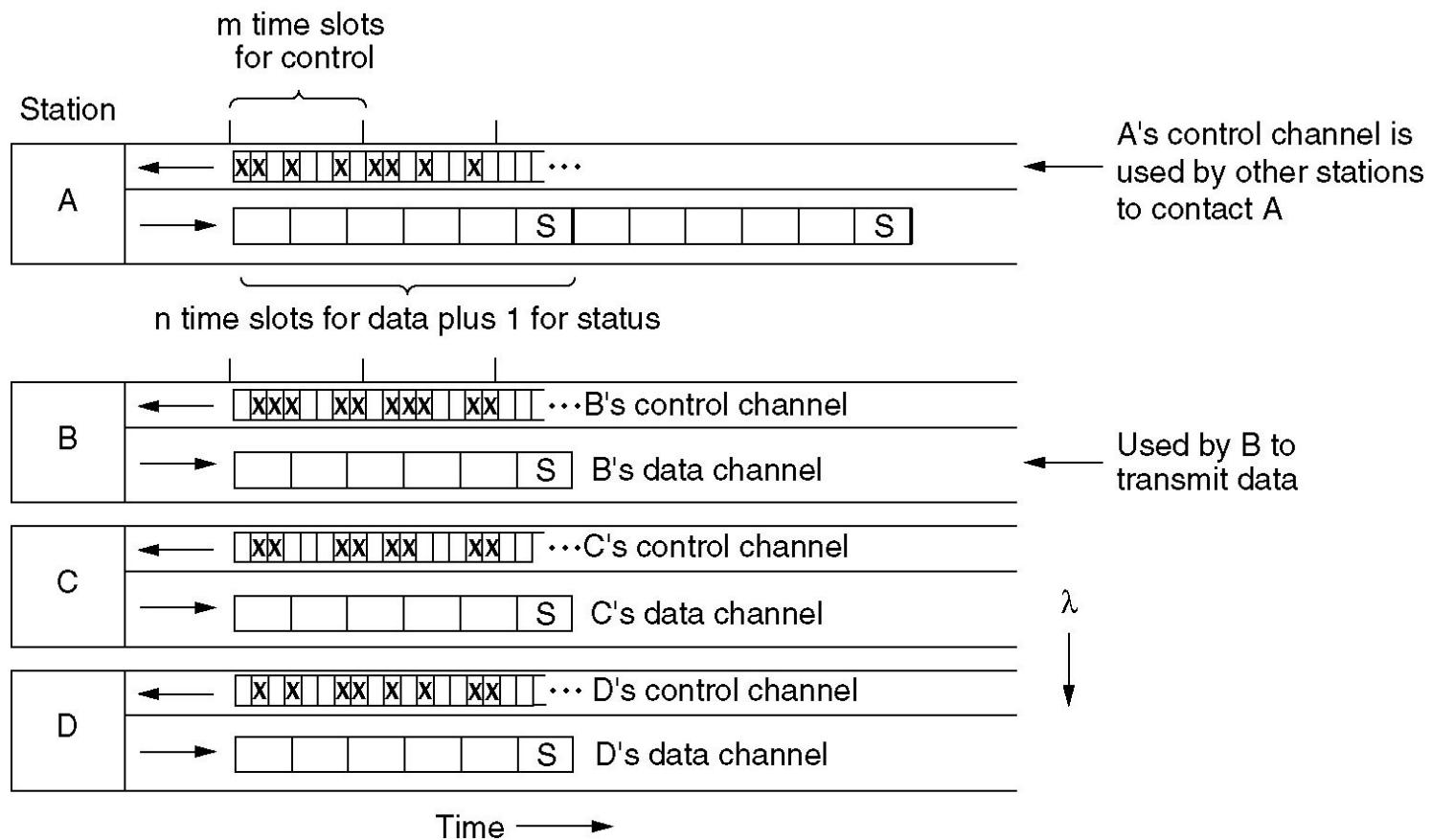
Acquisition probability for a symmetric contention channel.

Adaptive Tree Walk Protocol



The tree for eight stations.

Wavelength Division Multiple Access Protocols



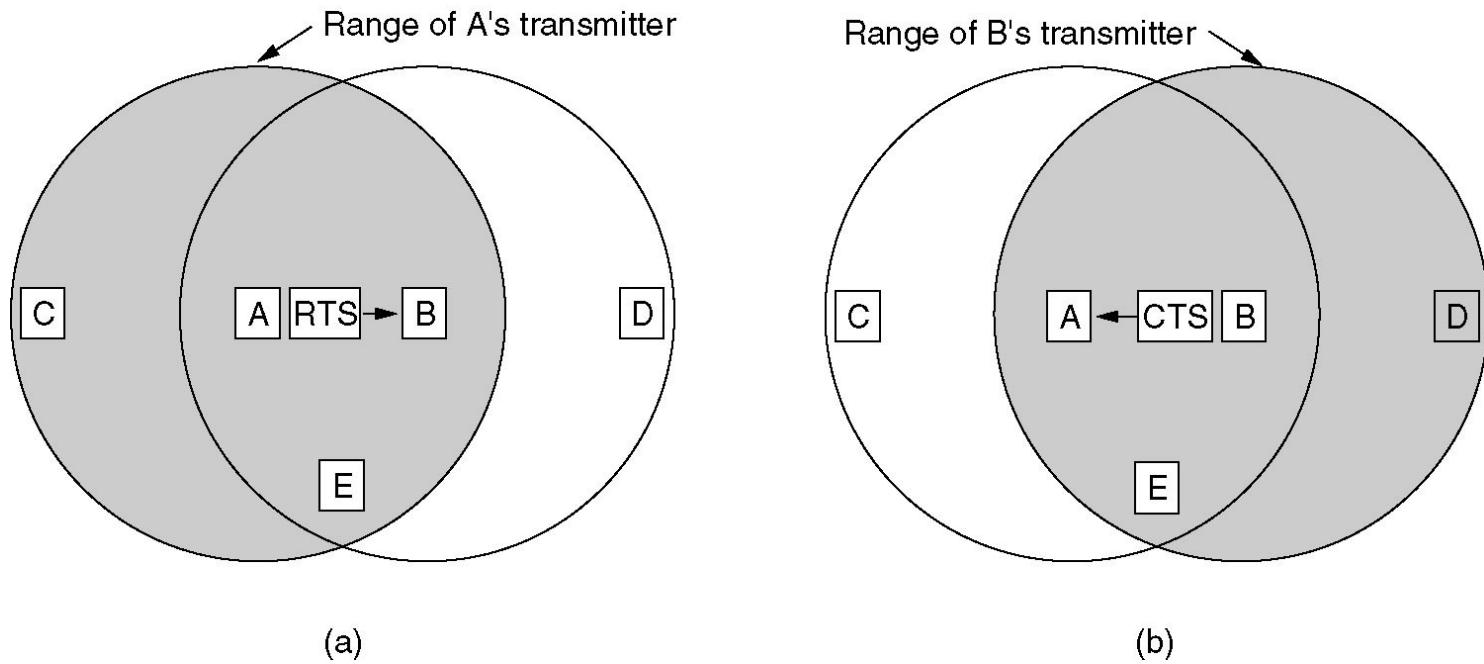
Wavelength division multiple access.

Wireless LAN Protocols



A wireless LAN. (a) A transmitting. (b) B transmitting.

Wireless LAN Protocols (2)



The MACA protocol. (a) A sending an RTS to B.
(b) B responding with a CTS to A.

Ethernet

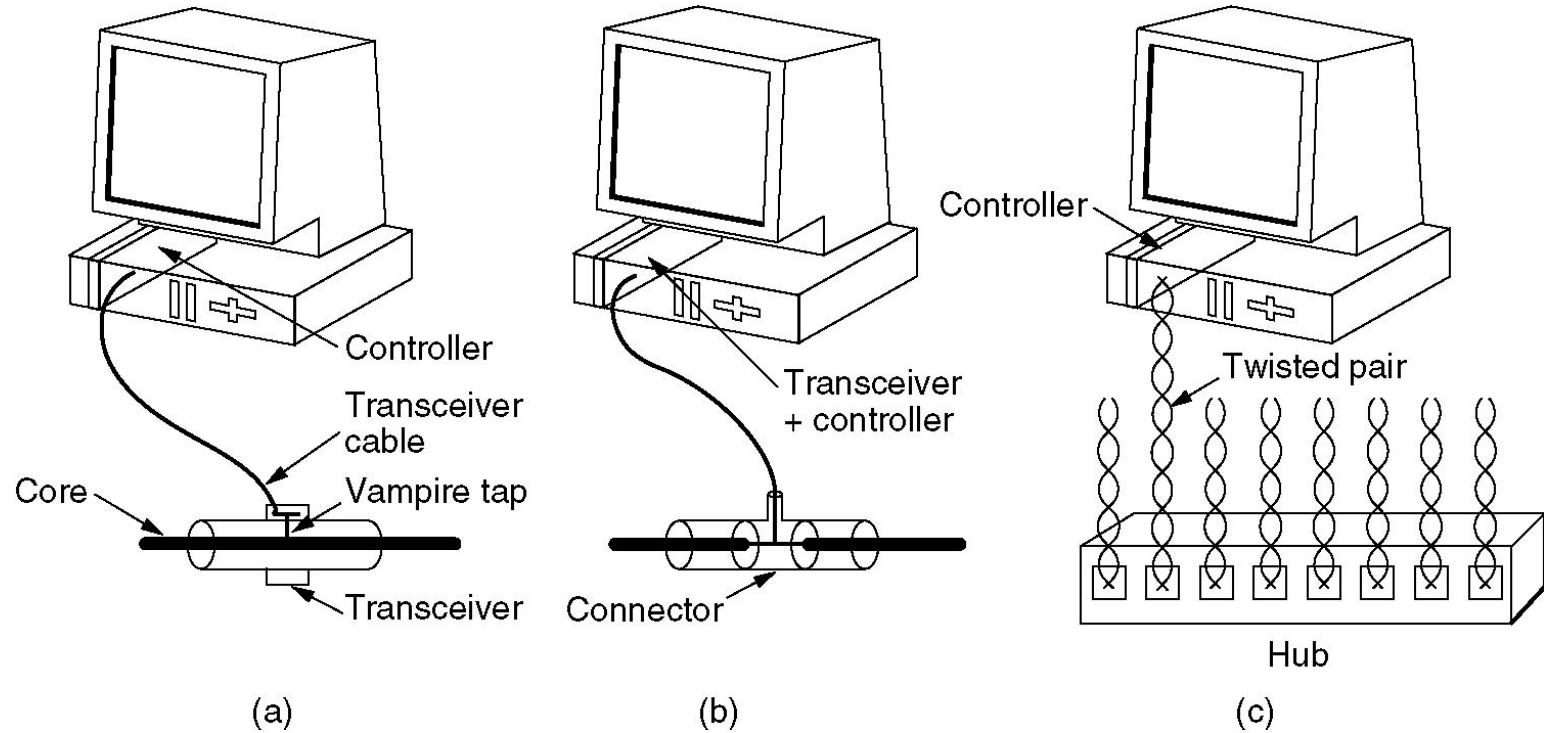
- Ethernet Cabling
- Manchester Encoding
- The Ethernet MAC Sublayer Protocol
- The Binary Exponential Backoff Algorithm
- Ethernet Performance
- Switched Ethernet
- Fast Ethernet
- Gigabit Ethernet
- IEEE 802.2: Logical Link Control
- Retrospective on Ethernet

Ethernet Cabling

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

The most common kinds of Ethernet cabling.

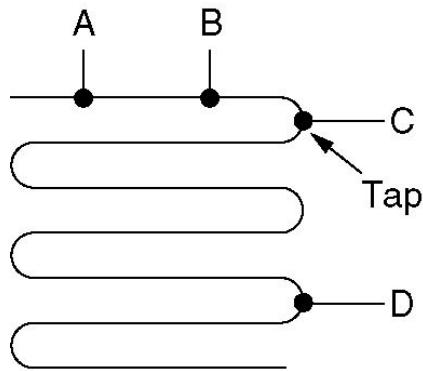
Ethernet Cabling (2)



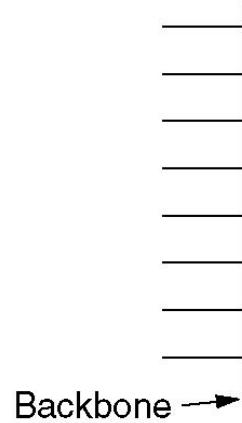
Three kinds of Ethernet cabling.

(a) 10Base5, (b) 10Base2, (c) 10Base-T.

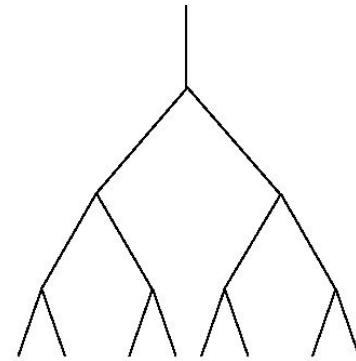
Ethernet Cabling (3)



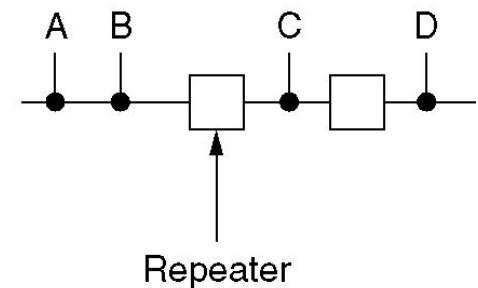
(a)



(b)



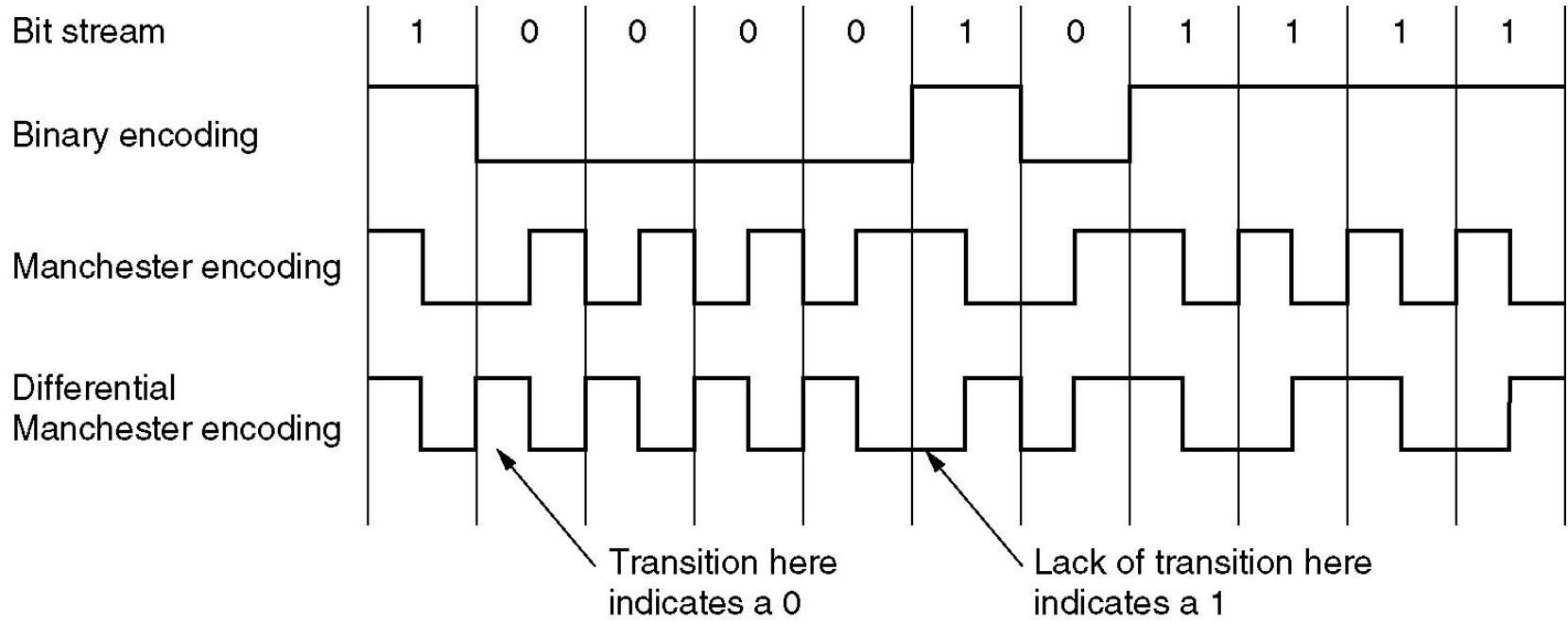
(c)



(d)

Cable topologies. (a) Linear, (b) Spine, (c) Tree, (d) Segmented.

Ethernet Cabling (4)



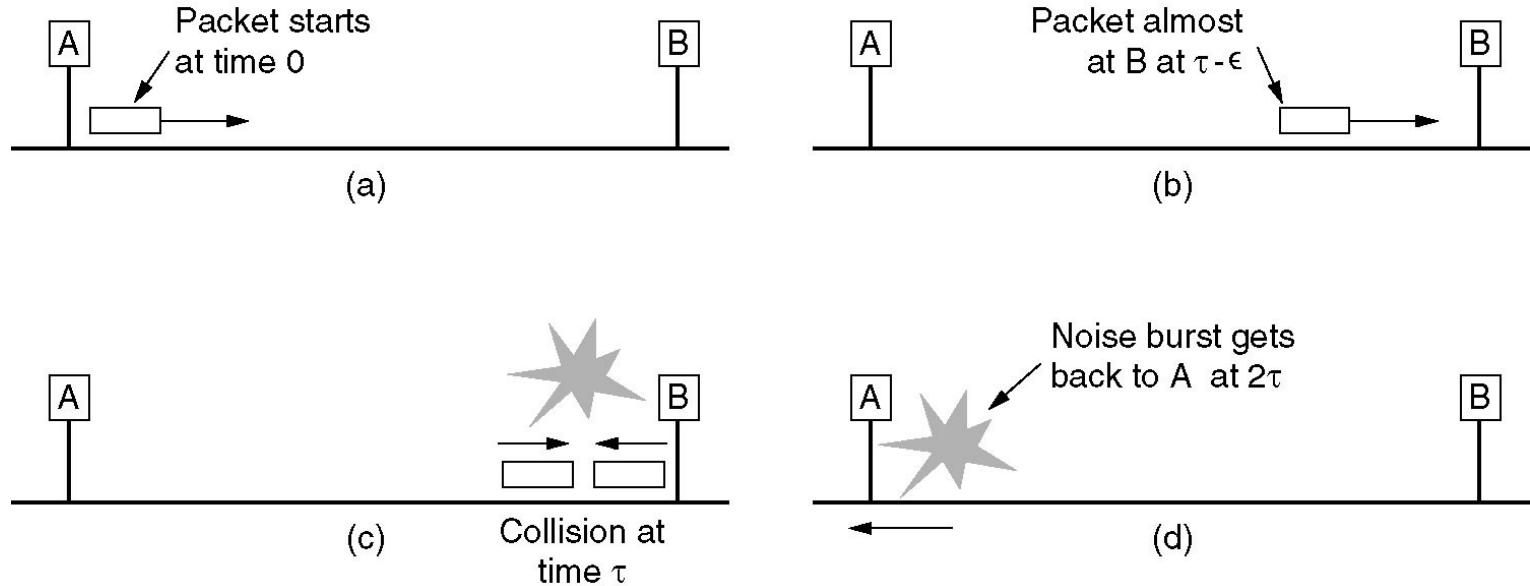
- (a) Binary encoding, (b) Manchester encoding,
(c) Differential Manchester encoding.

Ethernet MAC Sublayer Protocol

Bytes	8	6	6	2	0-1500	0-46	4	
(a)	Preamble	Destination address	Source address	Type	Data << >>	Pad	Check-sum	
(b)	Preamble	S o F	Destination address	Source address	Length	Data << >>	Pad	Check-sum

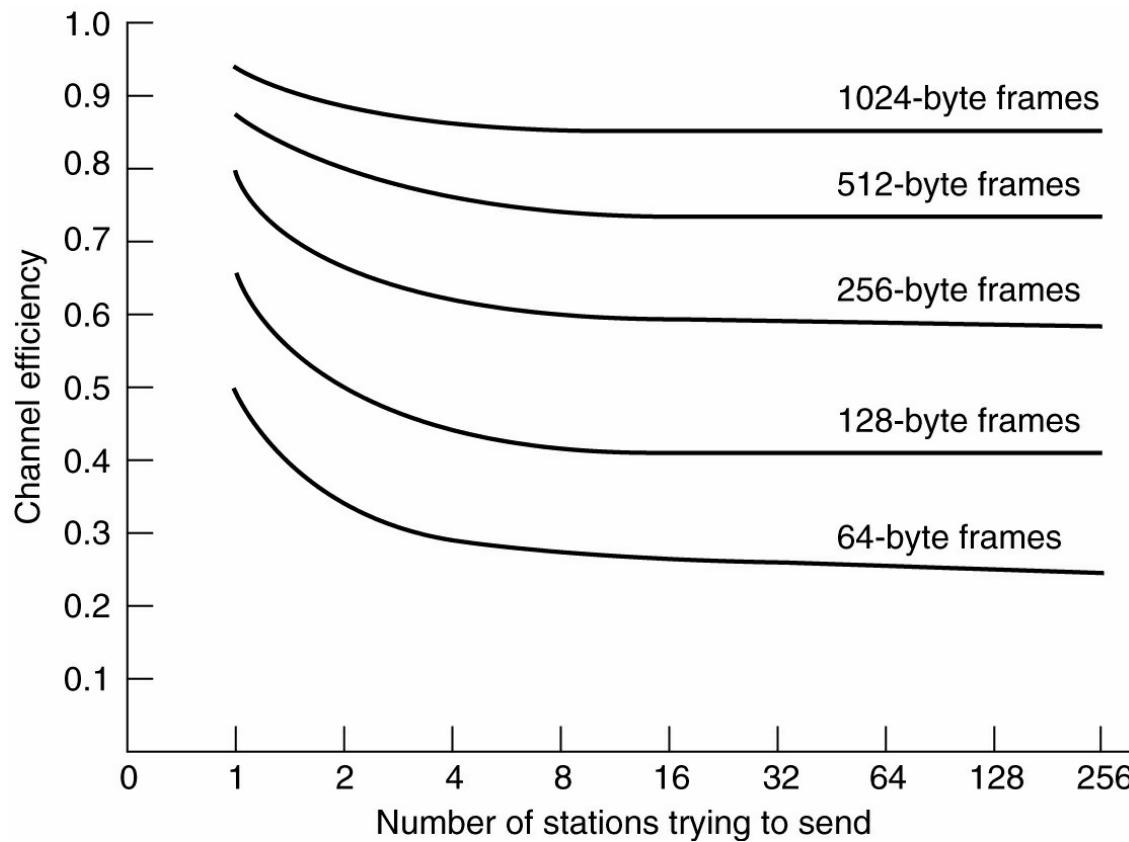
Frame formats. (a) DIX Ethernet, (b) IEEE 802.3.

Ethernet MAC Sublayer Protocol (2)



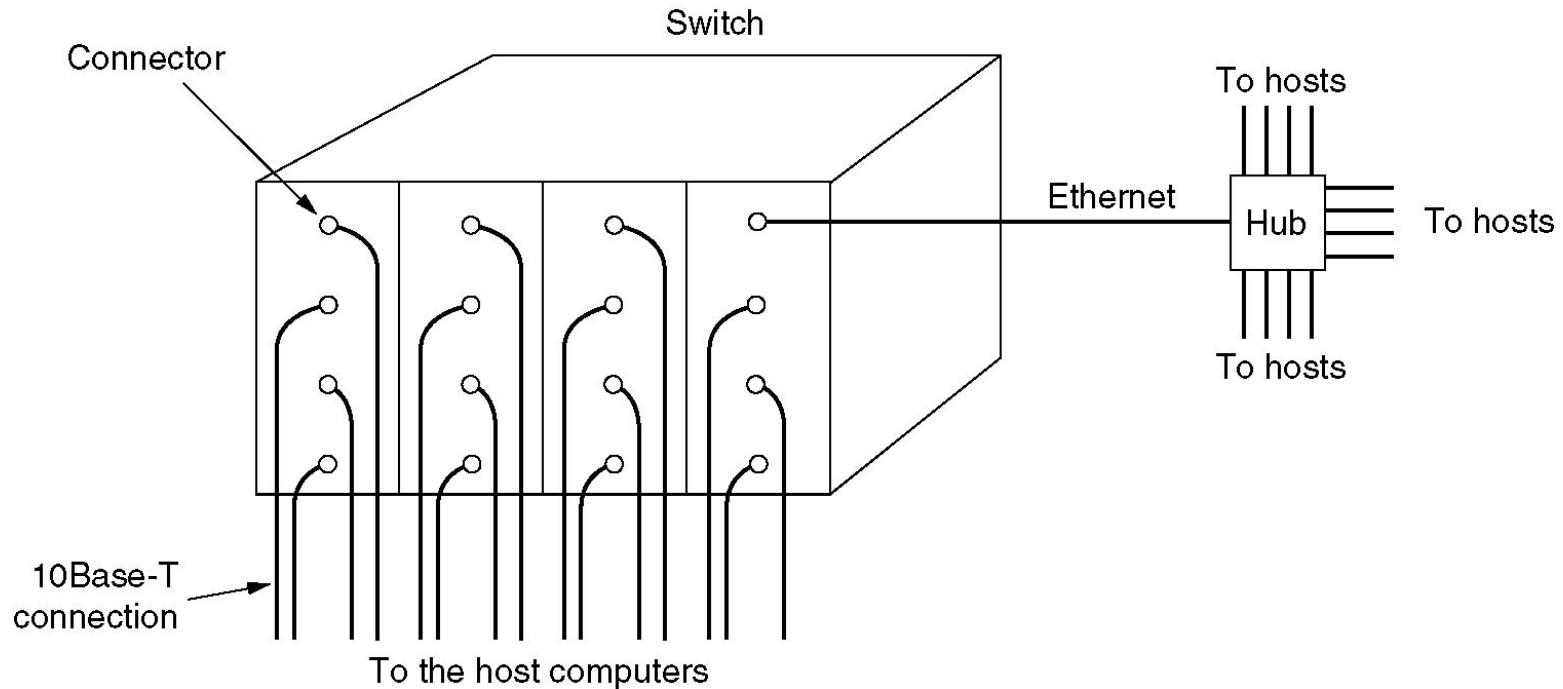
Collision detection can take as long as 2τ .

Ethernet Performance



Efficiency of Ethernet at 10 Mbps with 512-bit slot times.

Switched Ethernet



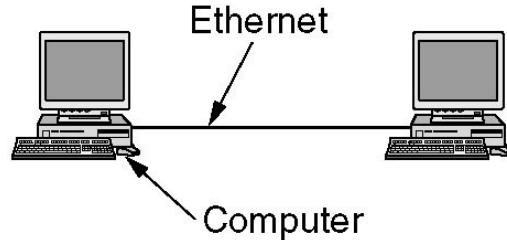
A simple example of switched Ethernet.

Fast Ethernet

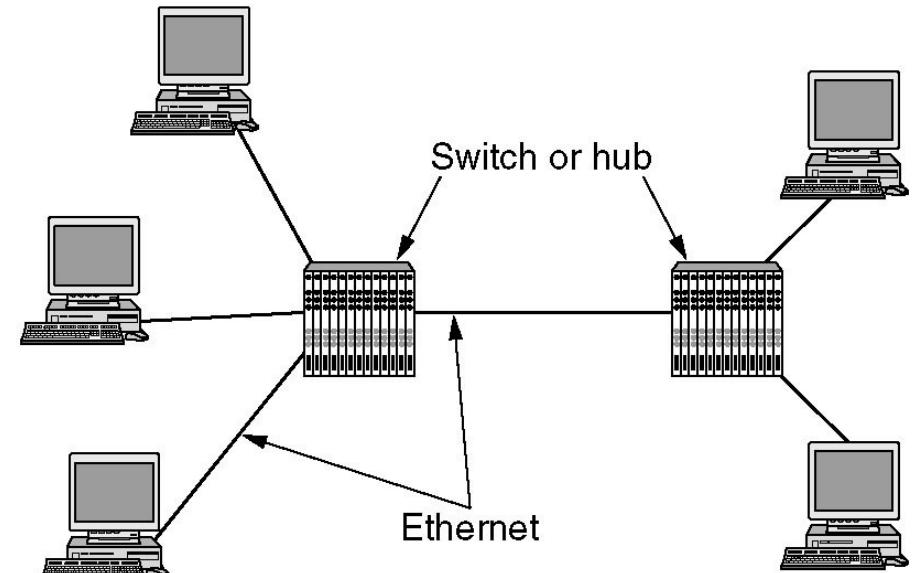
Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

The original fast Ethernet cabling.

Gigabit Ethernet



(a)



(b)

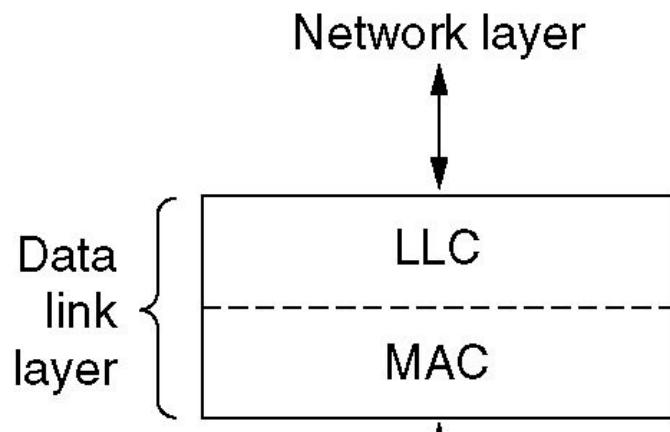
(a) A two-station Ethernet. (b) A multistation Ethernet.

Gigabit Ethernet (2)

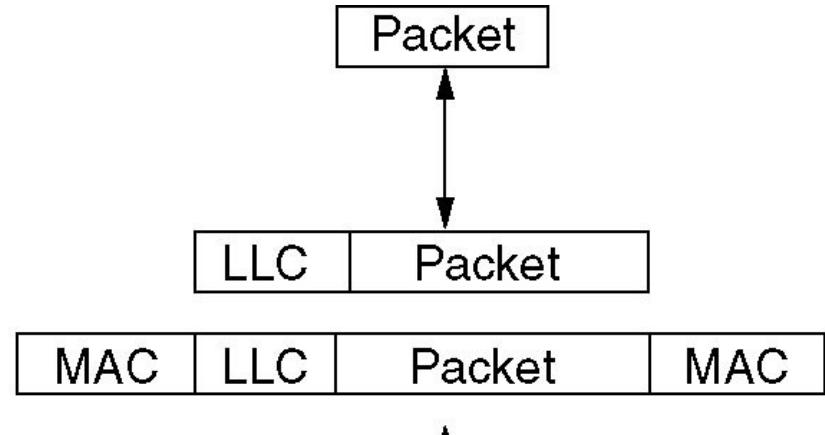
Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 μ) or multimode (50, 62.5 μ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

Gigabit Ethernet cabling.

IEEE 802.2: Logical Link Control



(a)



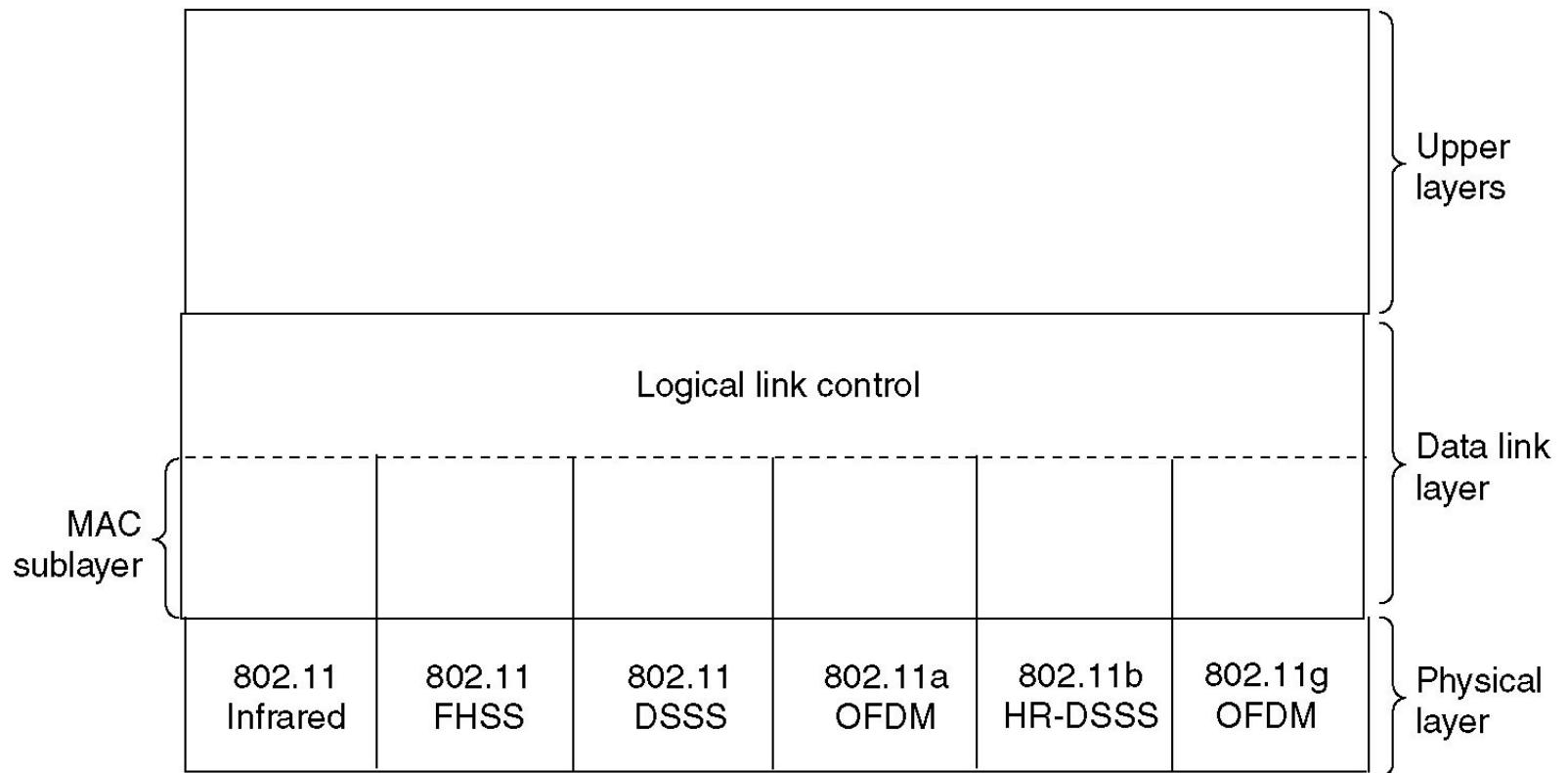
(b)

(a) Position of LLC. (b) Protocol formats.

Wireless LANs

- The 802.11 Protocol Stack
- The 802.11 Physical Layer
- The 802.11 MAC Sublayer Protocol
- The 802.11 Frame Structure
- Services

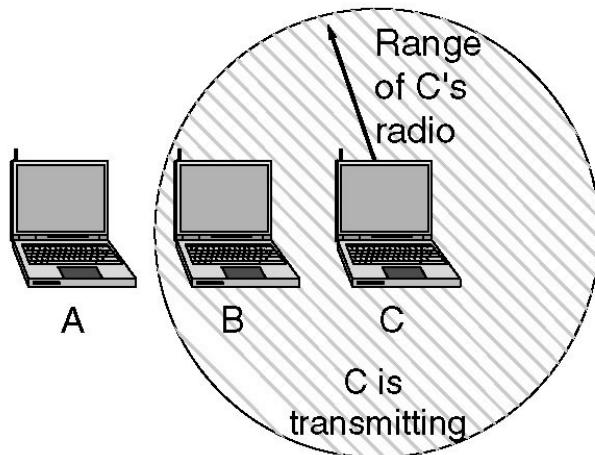
The 802.11 Protocol Stack



Part of the 802.11 protocol stack.

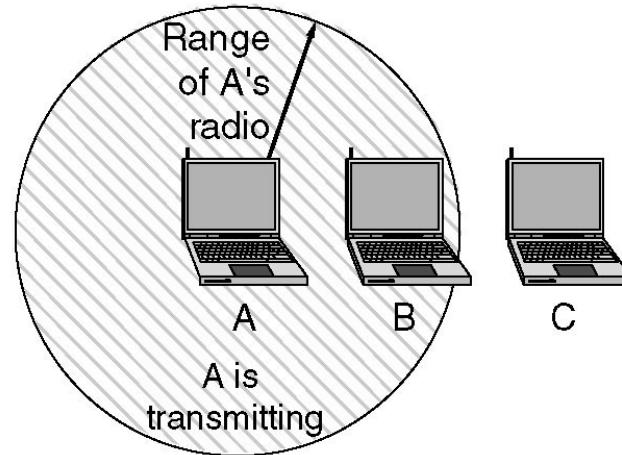
The 802.11 MAC Sublayer Protocol

A wants to send to B
but cannot hear that
B is busy



(a)

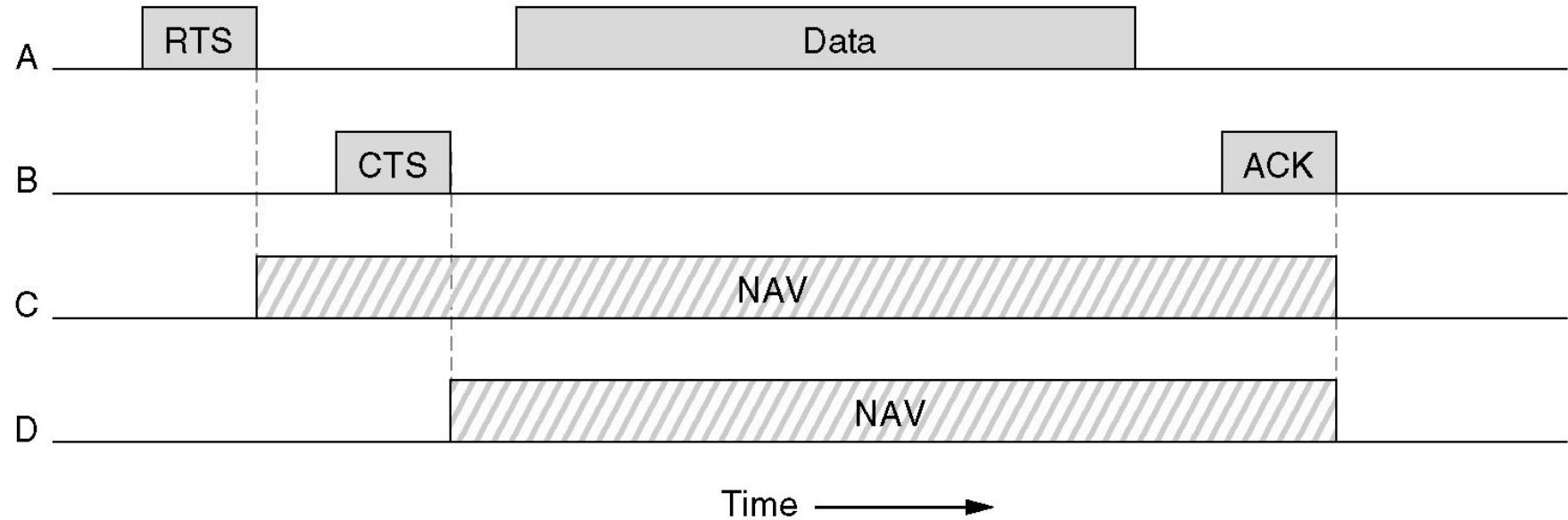
B wants to send to C
but mistakenly thinks
the transmission will fail



(b)

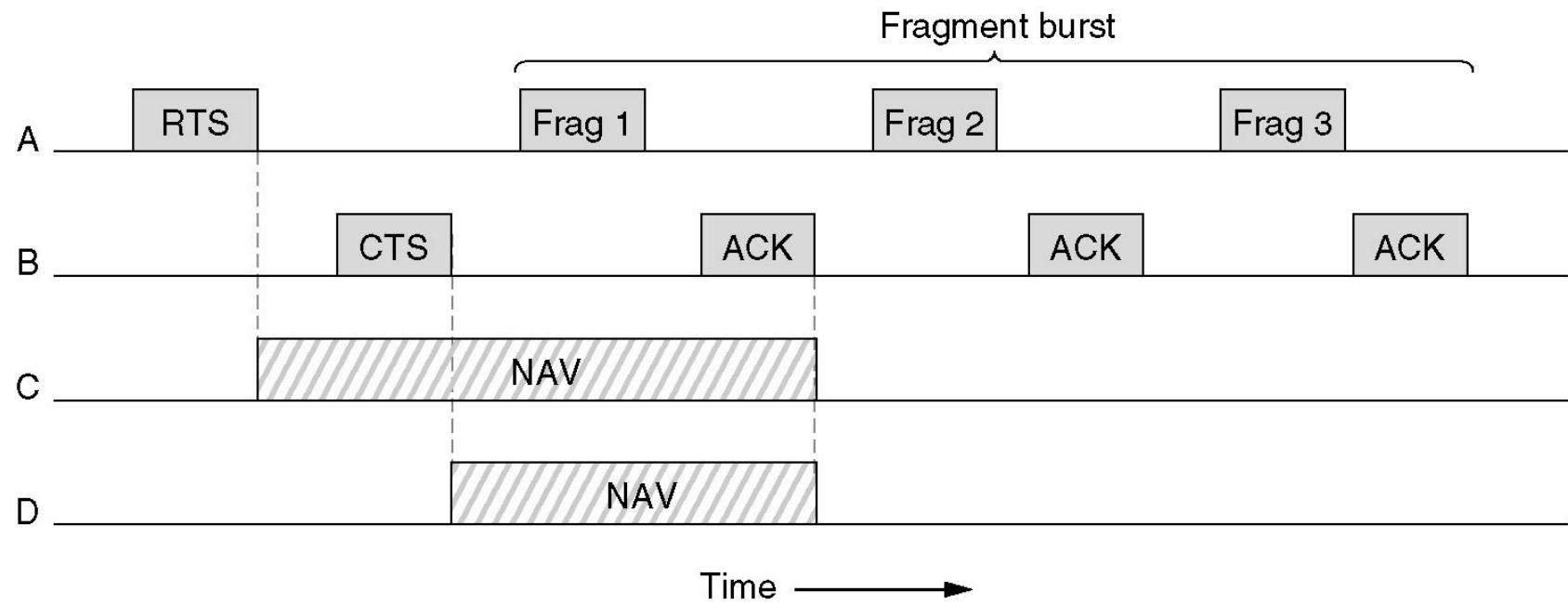
- (a) The hidden station problem.
- (b) The exposed station problem.

The 802.11 MAC Sublayer Protocol (2)



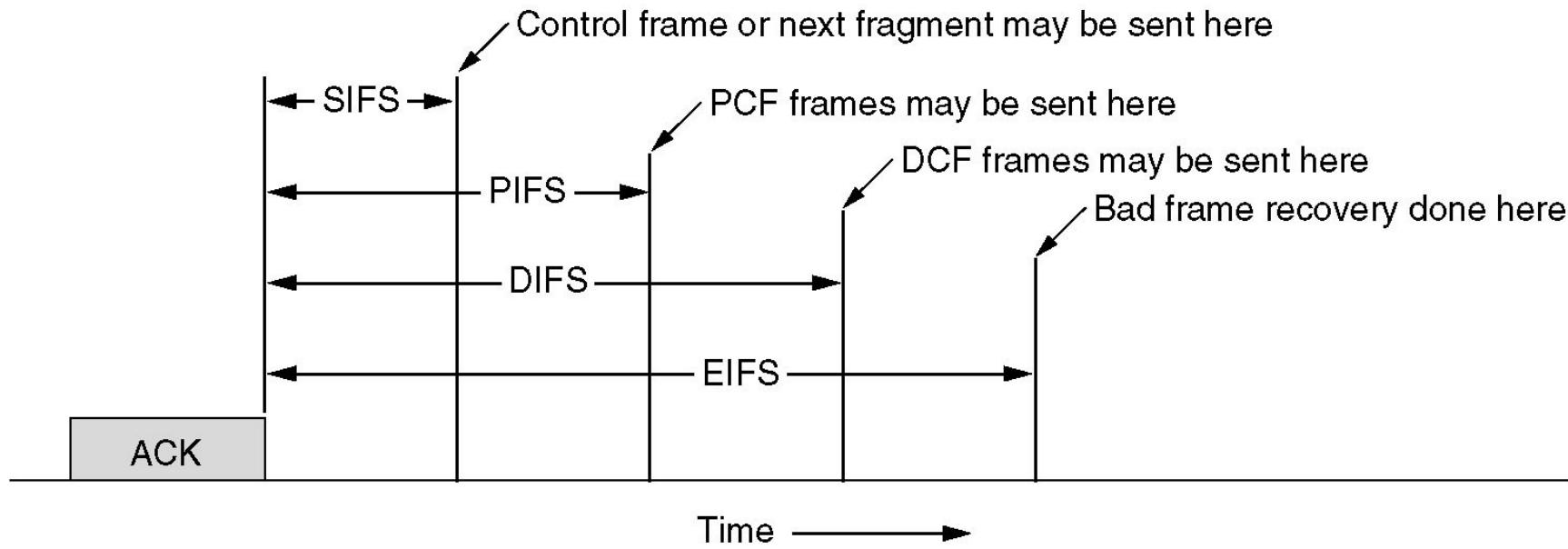
The use of virtual channel sensing using CSMA/CA.

The 802.11 MAC Sublayer Protocol (3)



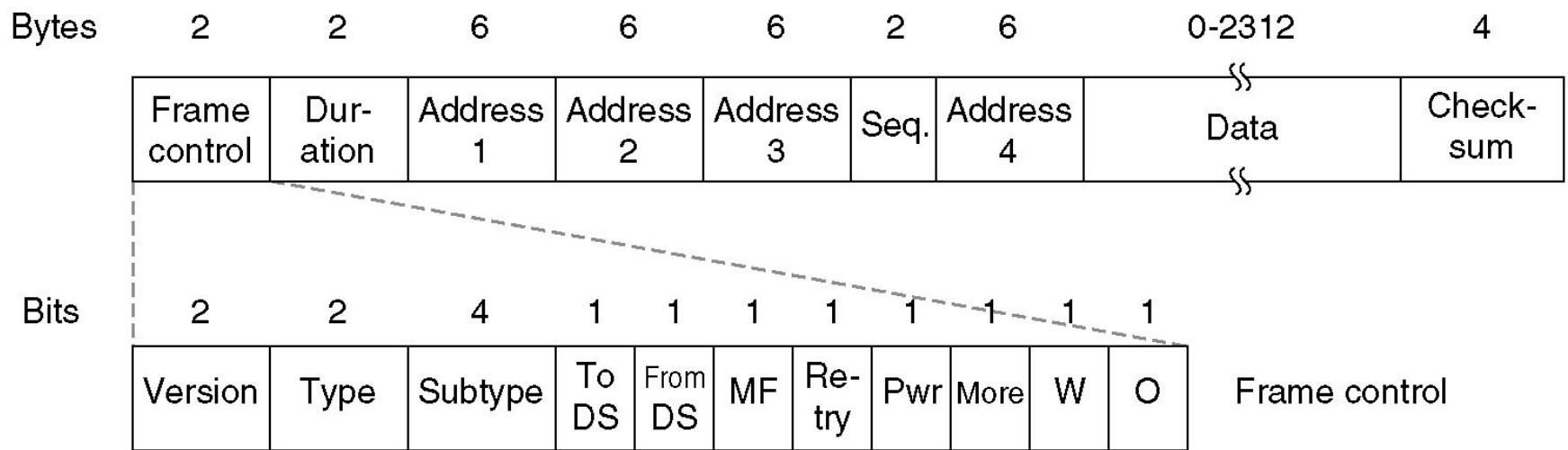
A fragment burst.

The 802.11 MAC Sublayer Protocol (4)



Interframe spacing in 802.11.

The 802.11 Frame Structure



The 802.11 data frame.

802.11 Services

Distribution Services

- Association
- Disassociation
- Reassociation
- Distribution
- Integration

802.11 Services

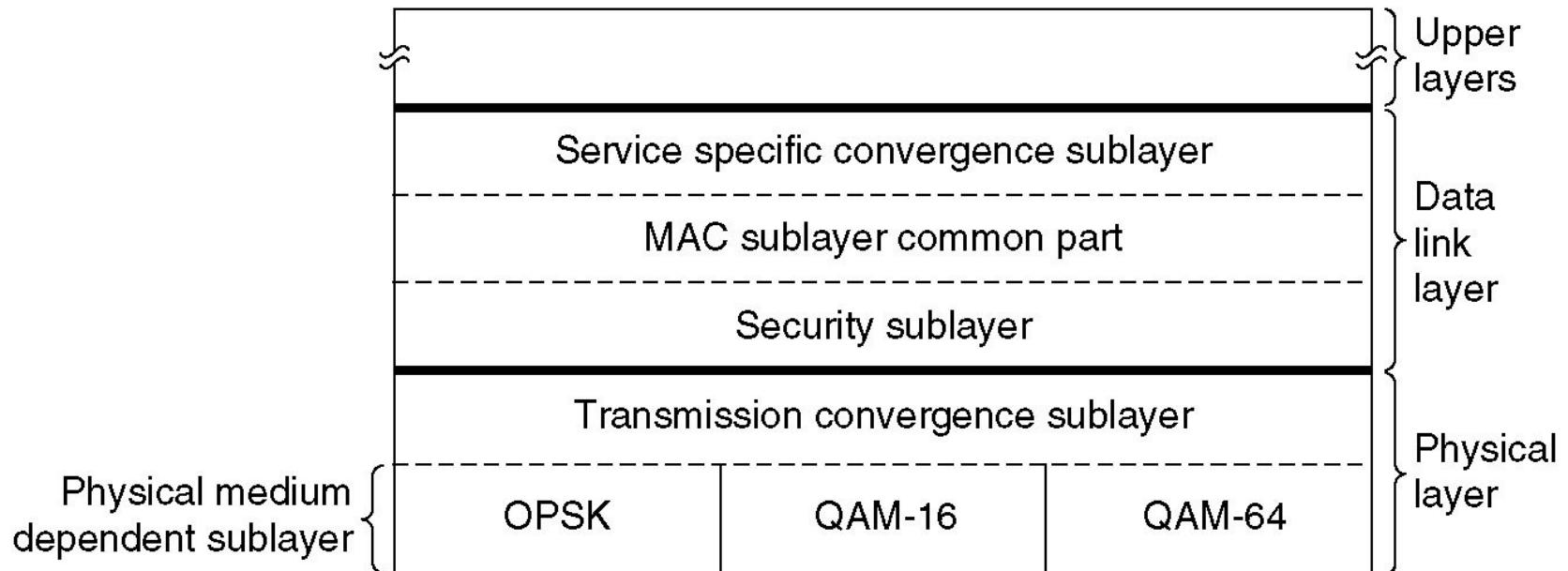
Intracell Services

- Authentication
- Deauthentication
- Privacy
- Data Delivery

Broadband Wireless

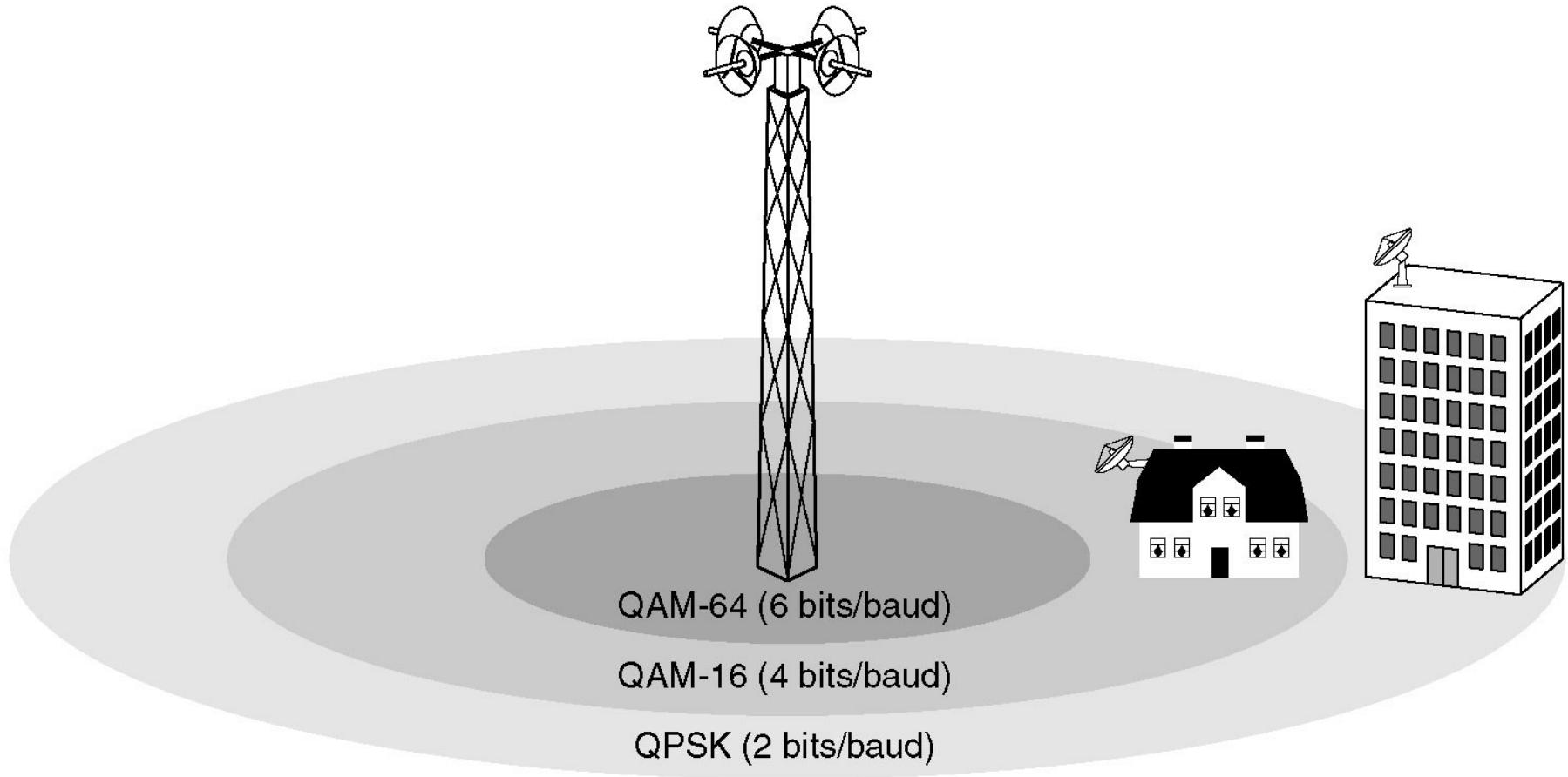
- Comparison of 802.11 and 802.16
- The 802.16 Protocol Stack
- The 802.16 Physical Layer
- The 802.16 MAC Sublayer Protocol
- The 802.16 Frame Structure

The 802.16 Protocol Stack



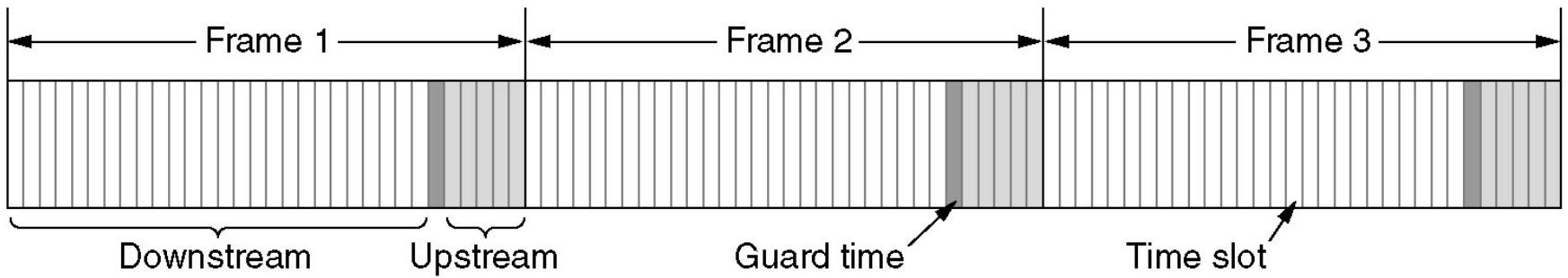
The 802.16 Protocol Stack.

The 802.16 Physical Layer



The 802.16 transmission environment.

The 802.16 Physical Layer (2)



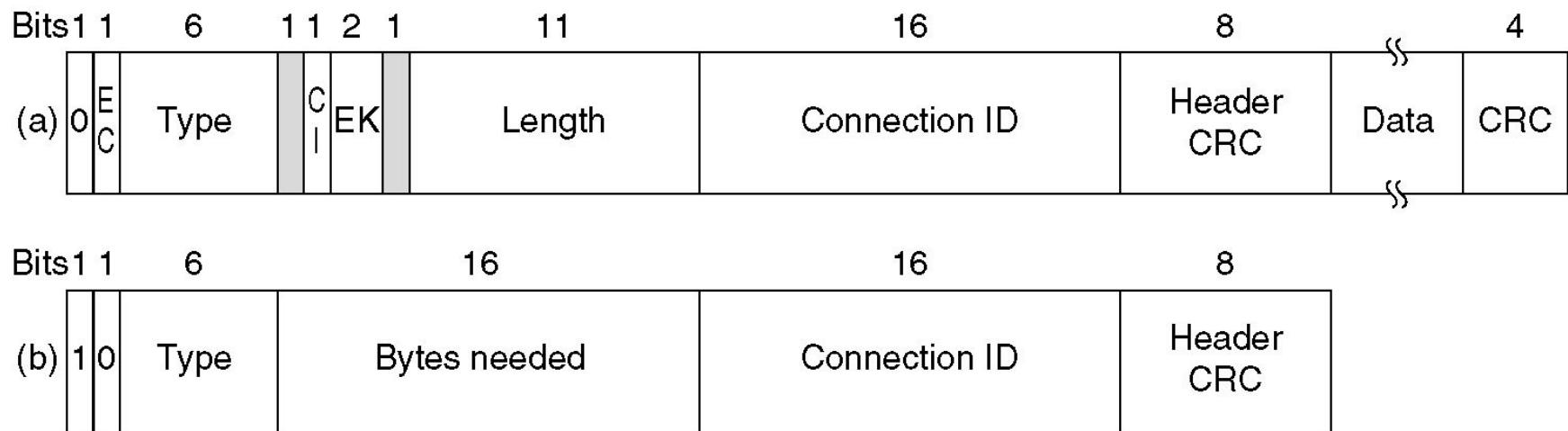
Frames and time slots for time division duplexing.

The 802.16 MAC Sublayer Protocol

Service Classes

- Constant bit rate service
- Real-time variable bit rate service
- Non-real-time variable bit rate service
- Best efforts service

The 802.16 Frame Structure

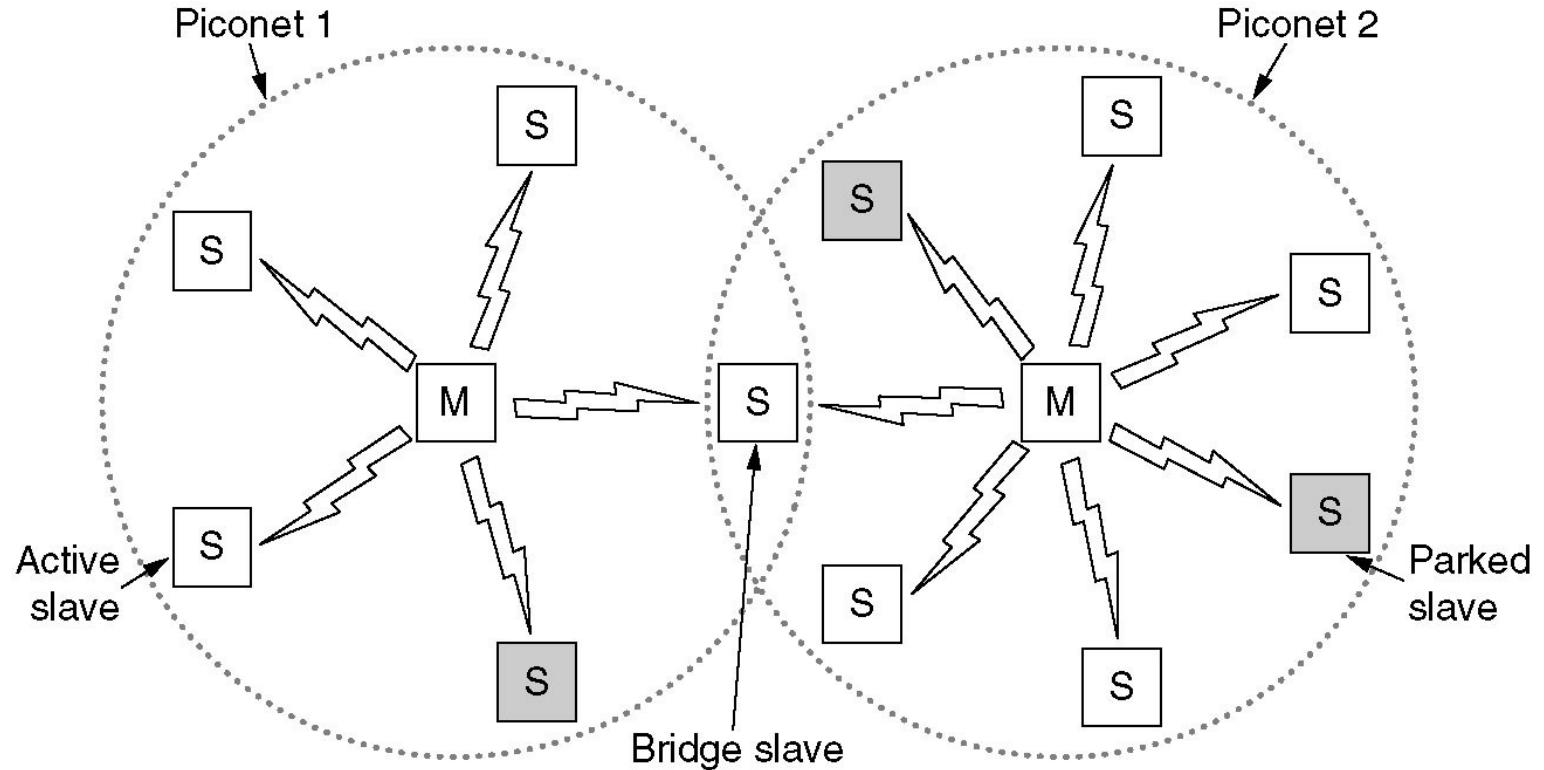


(a) A generic frame. (b) A bandwidth request frame.

Bluetooth

- Bluetooth Architecture
- Bluetooth Applications
- The Bluetooth Protocol Stack
- The Bluetooth Radio Layer
- The Bluetooth Baseband Layer
- The Bluetooth L2CAP Layer
- The Bluetooth Frame Structure

Bluetooth Architecture



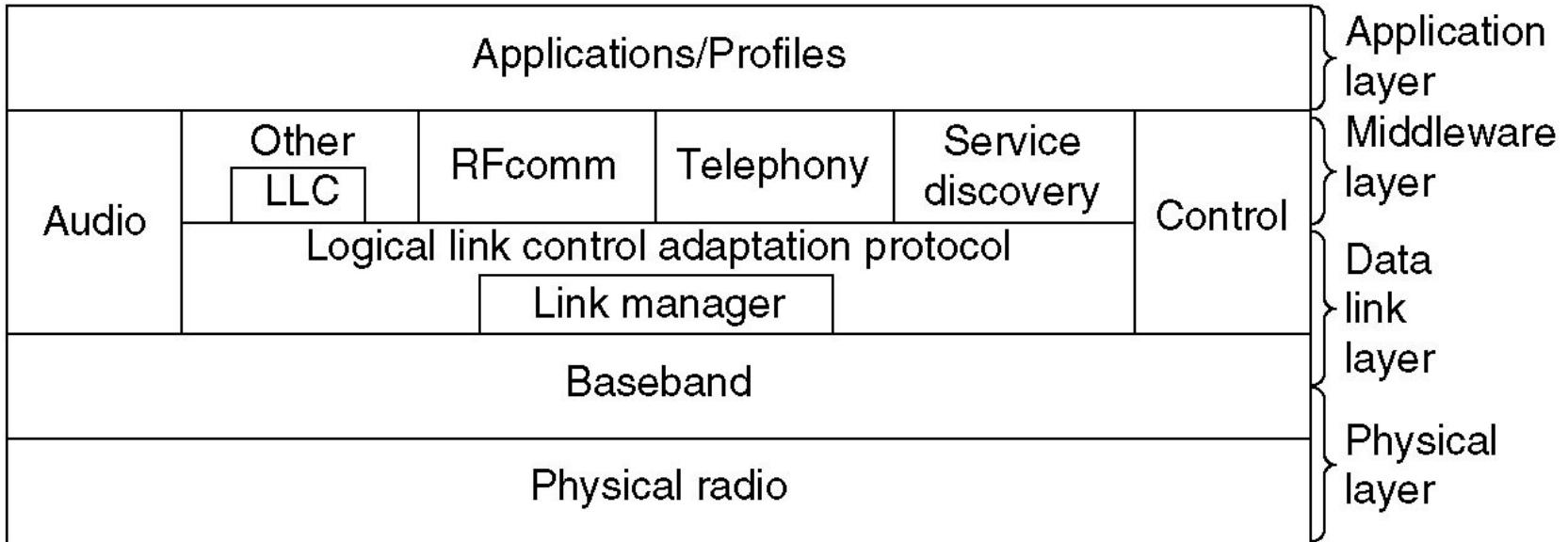
Two piconets can be connected to form a scatternet.

Bluetooth Applications

Name	Description
Generic access	Procedures for link management
Service discovery	Protocol for discovering offered services
Serial port	Replacement for a serial port cable
Generic object exchange	Defines client-server relationship for object movement
LAN access	Protocol between a mobile computer and a fixed LAN
Dial-up networking	Allows a notebook computer to call via a mobile phone
Fax	Allows a mobile fax machine to talk to a mobile phone
Cordless telephony	Connects a handset and its local base station
Intercom	Digital walkie-talkie
Headset	Intended for hands-free voice communication
Object push	Provides a way to exchange simple objects
File transfer	Provides a more general file transfer facility
Synchronization	Permits a PDA to synchronize with another computer

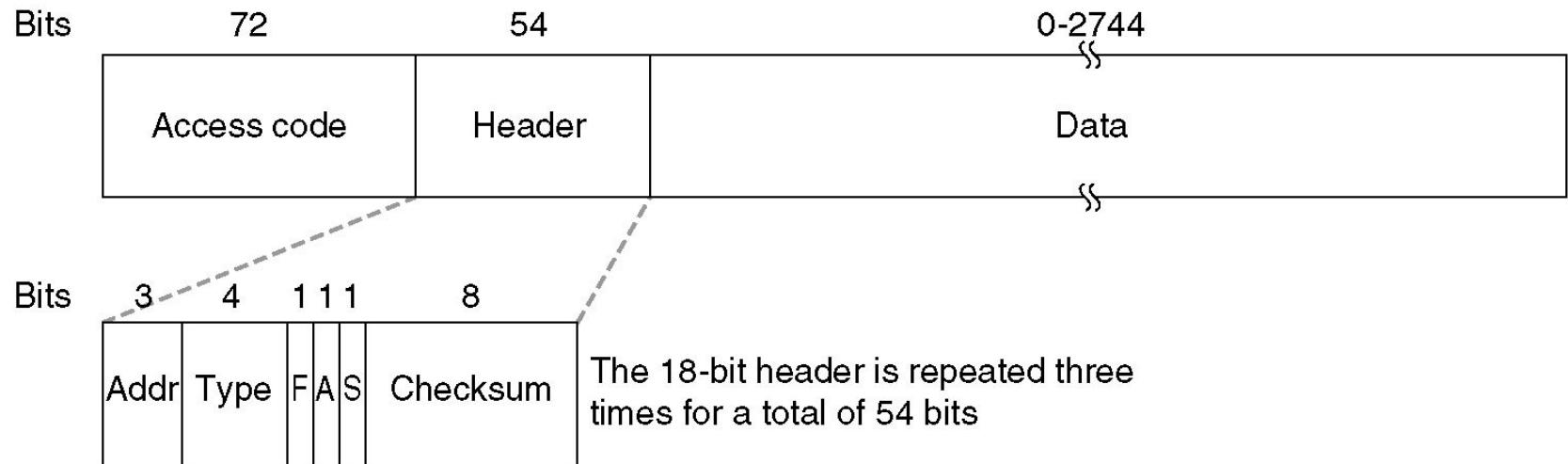
The Bluetooth profiles.

The Bluetooth Protocol Stack



The 802.15 version of the Bluetooth protocol architecture.

The Bluetooth Frame Structure

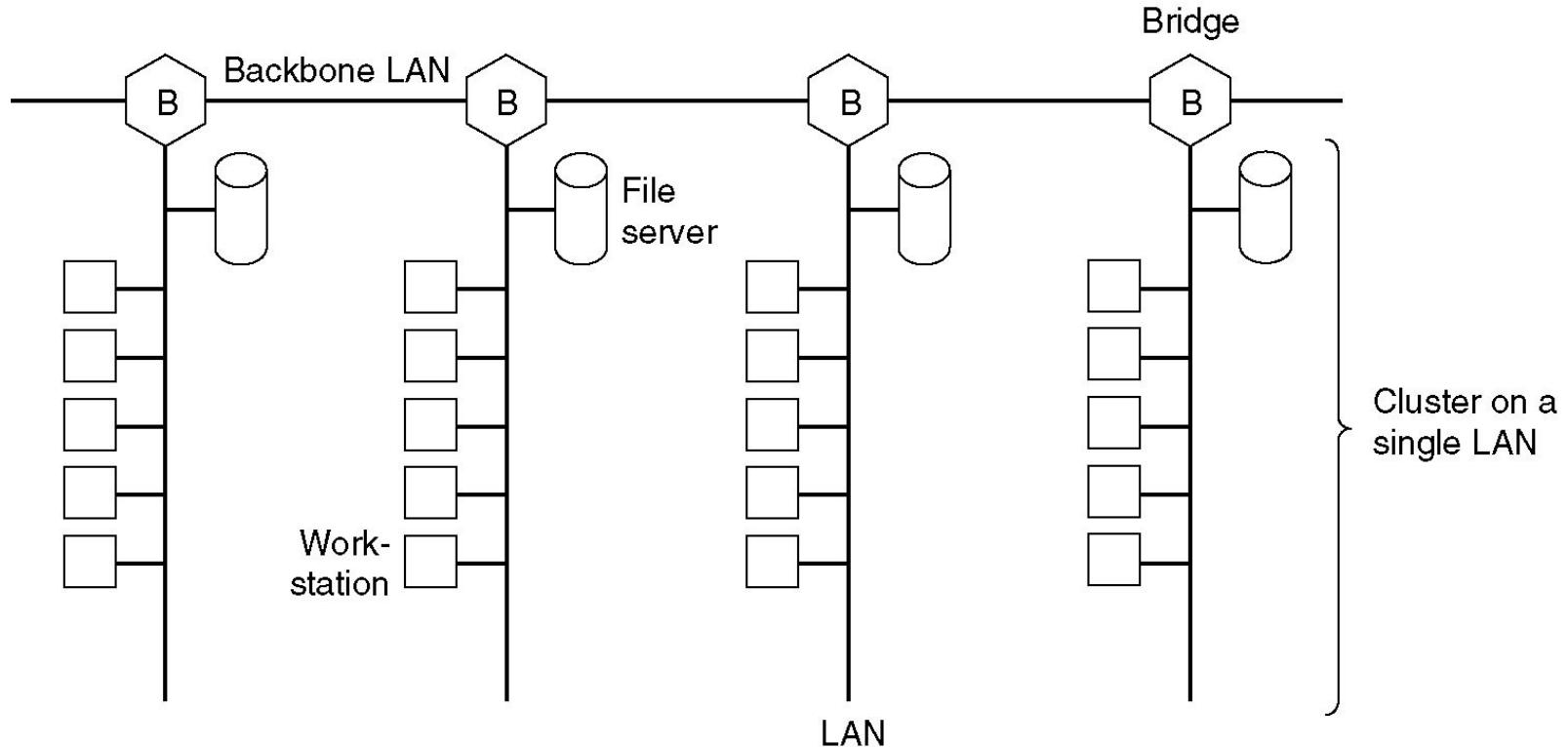


A typical Bluetooth data frame.

Data Link Layer Switching

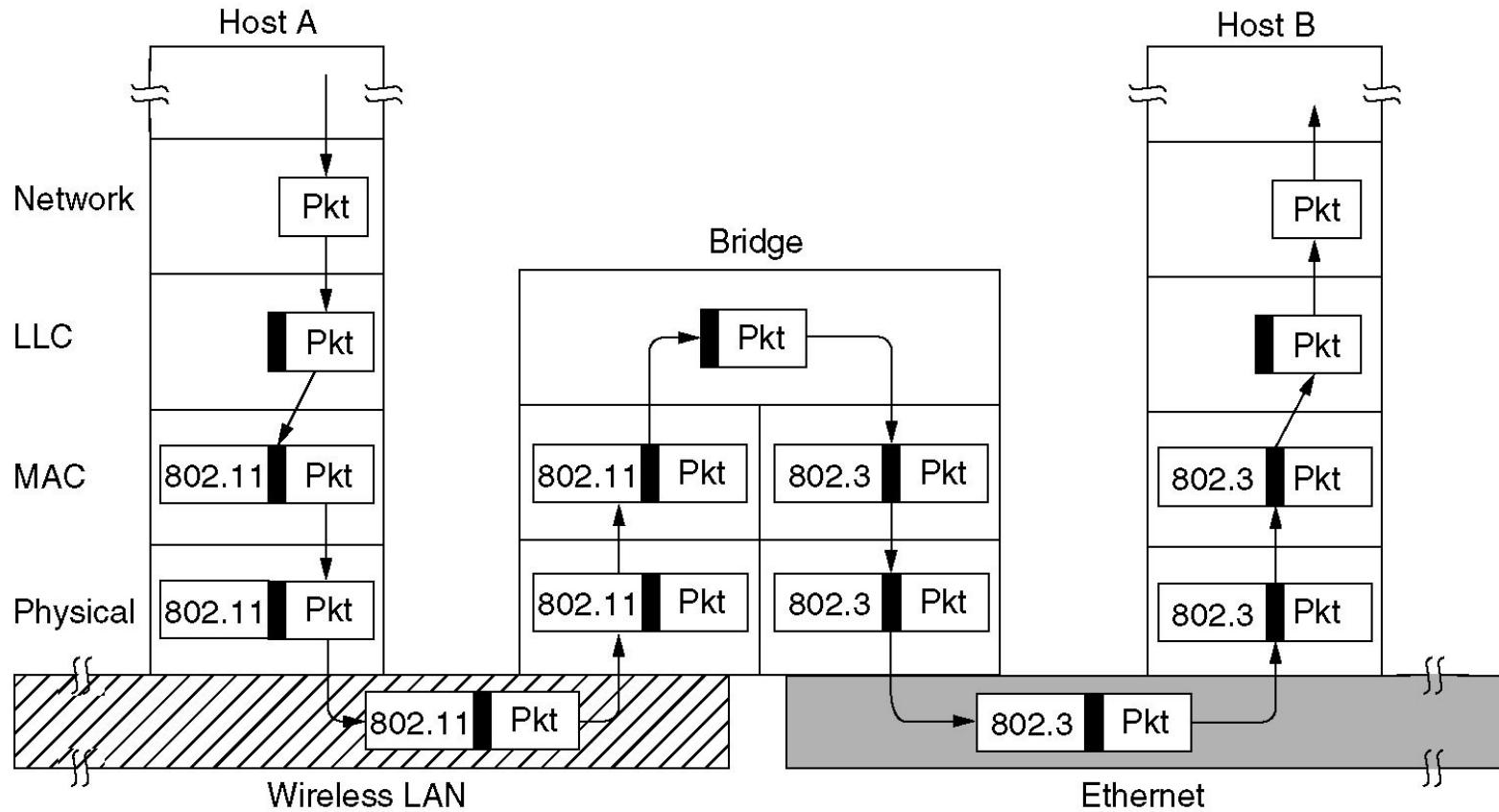
- Bridges from 802.x to 802.y
- Local Internetworking
- Spanning Tree Bridges
- Remote Bridges
- Repeaters, Hubs, Bridges, Switches, Routers, Gateways
- Virtual LANs

Data Link Layer Switching



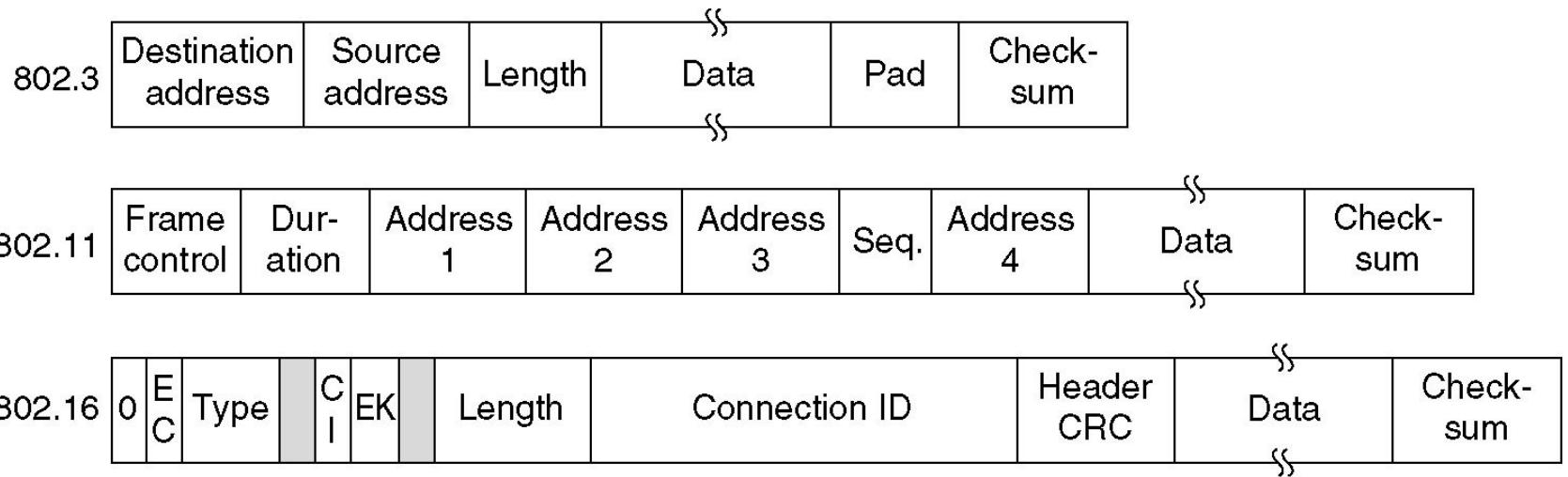
Multiple LANs connected by a backbone to handle a total load higher than the capacity of a single LAN.

Bridges from 802.x to 802.y



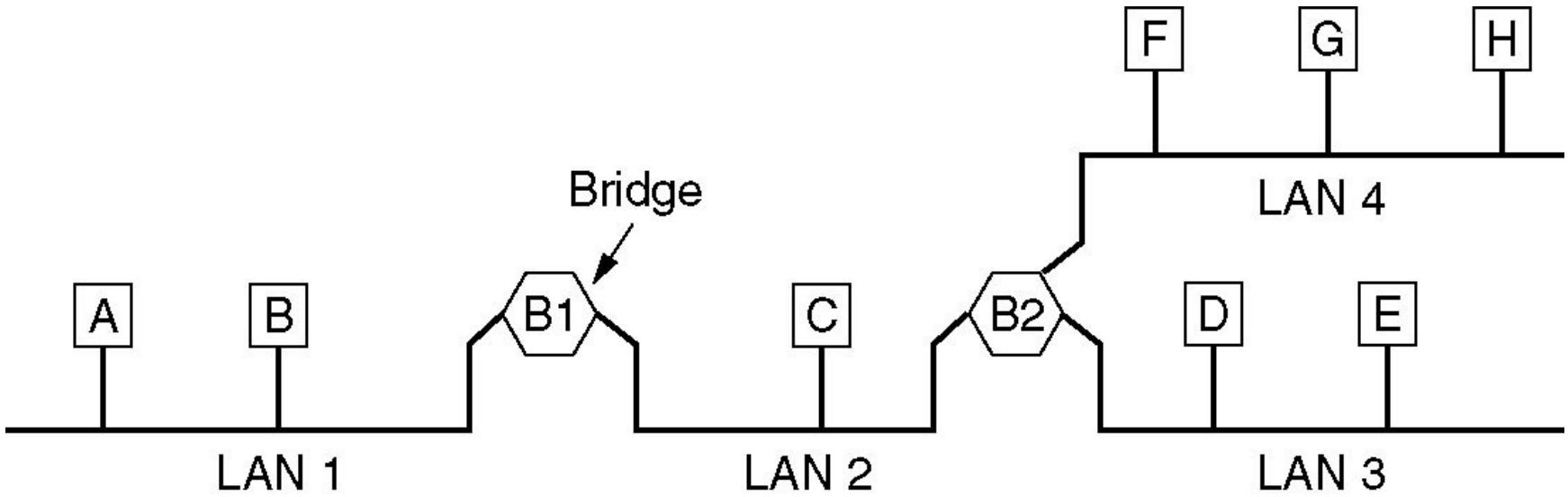
Operation of a LAN bridge from 802.11 to 802.3.

Bridges from 802.x to 802.y (2)



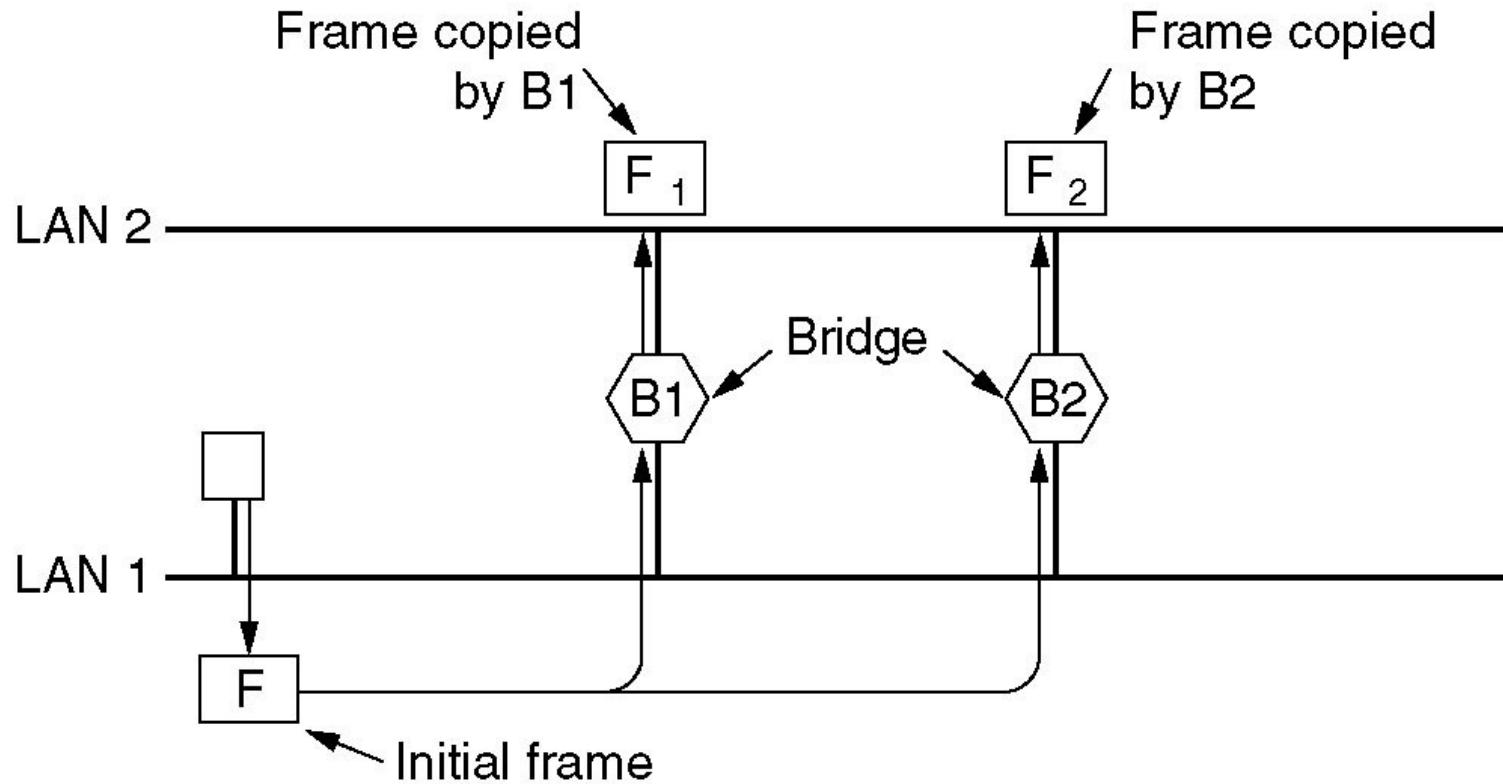
The IEEE 802 frame formats. The drawing is not to scale.

Local Internetworking



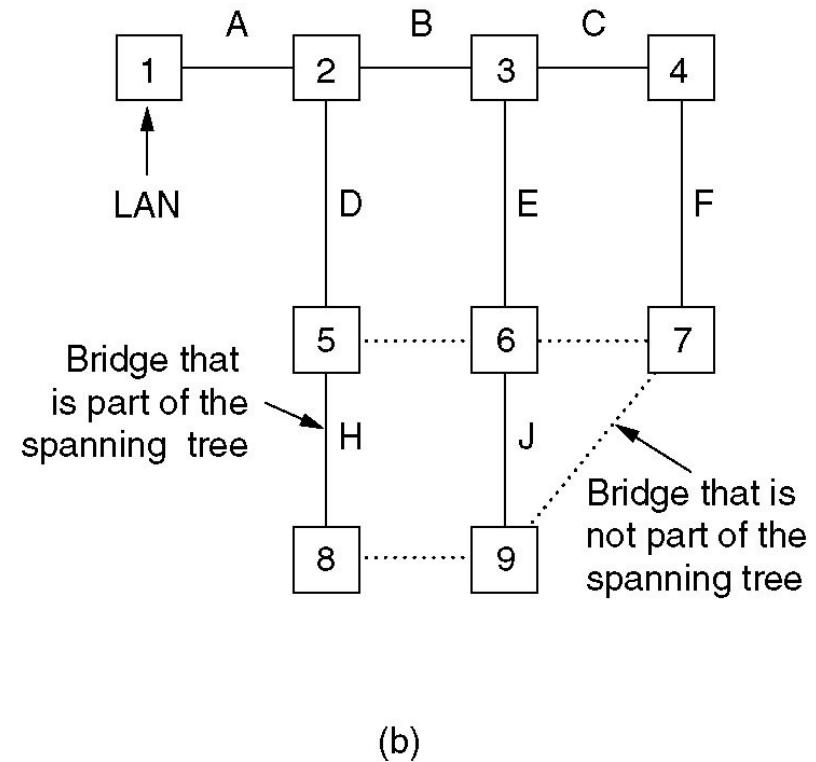
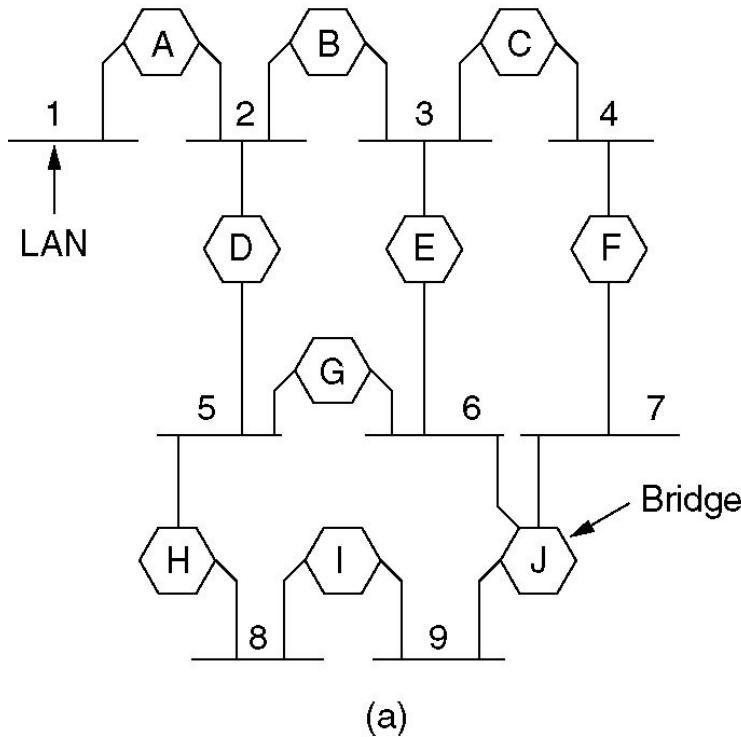
A configuration with four LANs and two bridges.

Spanning Tree Bridges



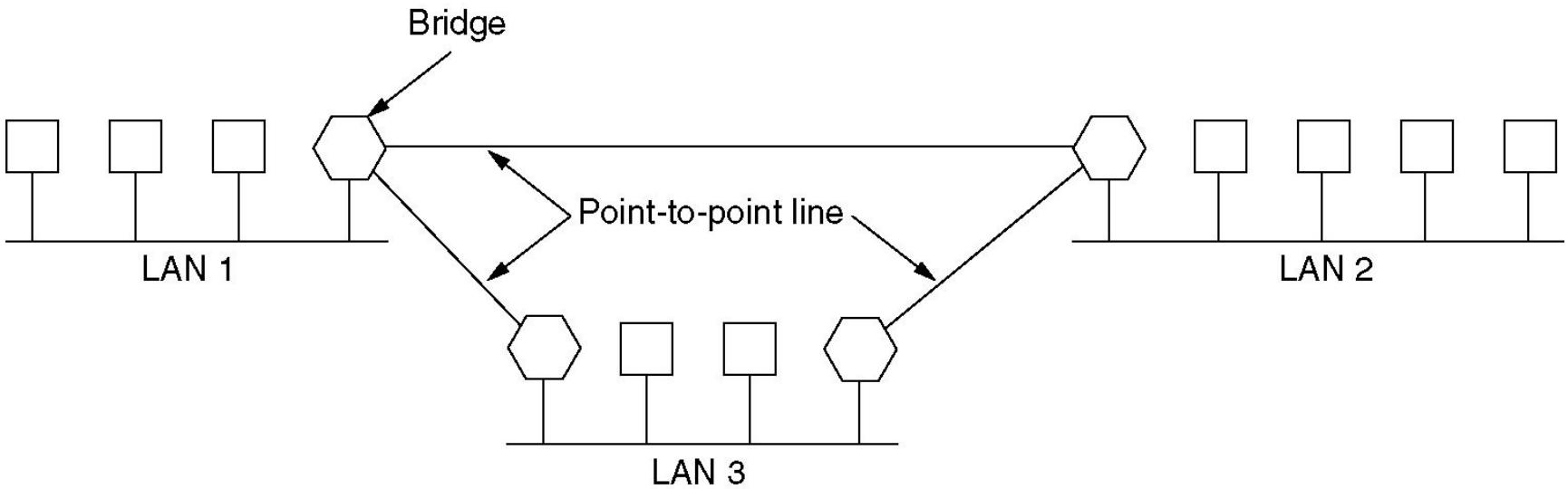
Two parallel transparent bridges.

Spanning Tree Bridges (2)



- (a) Interconnected LANs. (b) A spanning tree covering the LANs. The dotted lines are not part of the spanning tree.

Remote Bridges

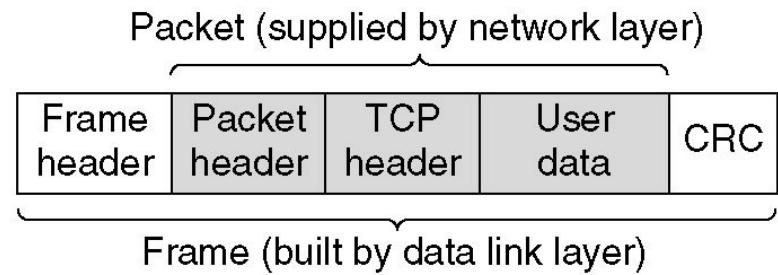


Remote bridges can be used to interconnect distant LANs.

Repeaters, Hubs, Bridges, Switches, Routers and Gateways

Application layer	Application gateway
Transport layer	Transport gateway
Network layer	Router
Data link layer	Bridge, switch
Physical layer	Repeater, hub

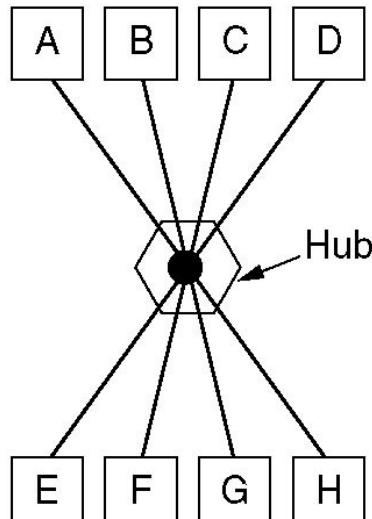
(a)



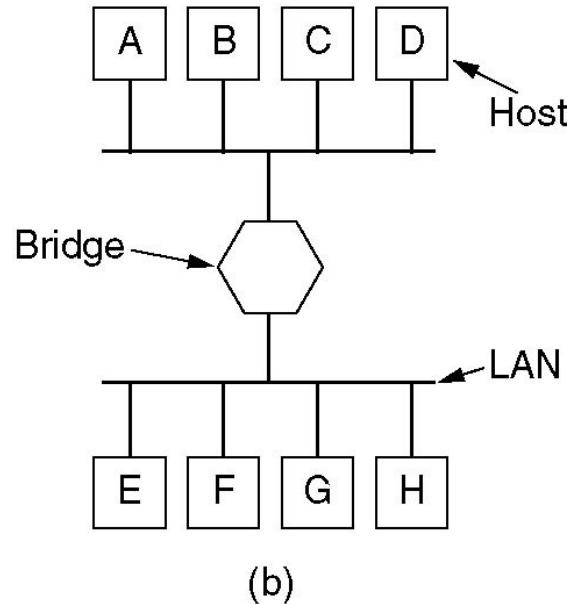
(b)

- (a) Which device is in which layer.
- (b) Frames, packets, and headers.

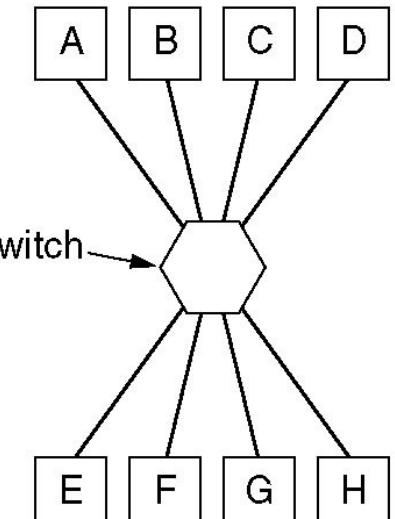
Repeaters, Hubs, Bridges, Switches, Routers and Gateways (2)



(a)



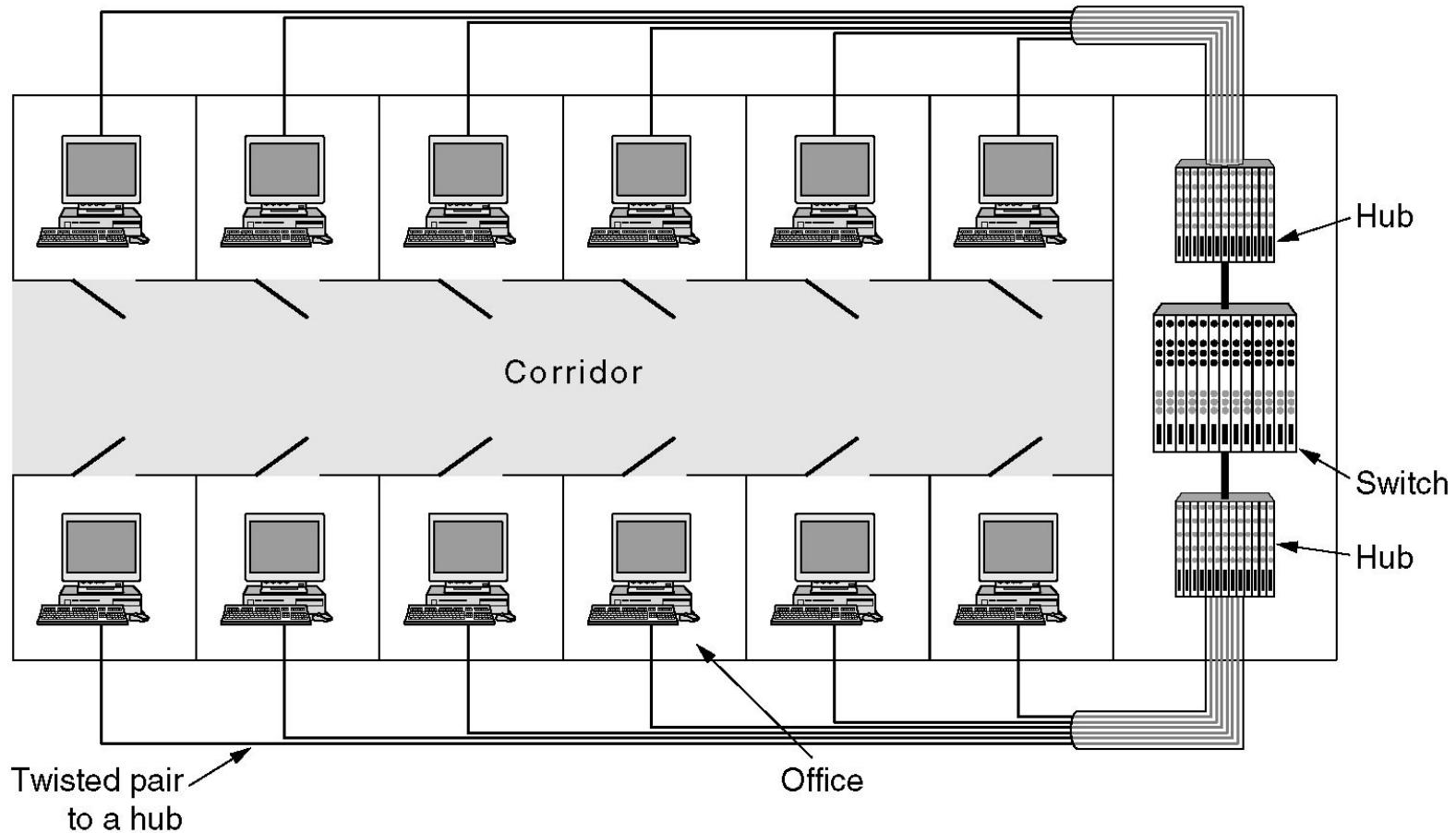
(b)



(c)

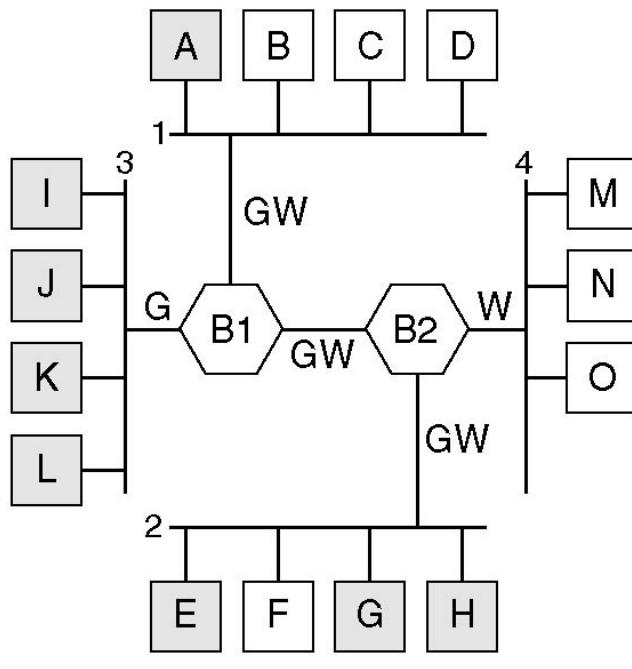
(a) A hub. (b) A bridge. (c) a switch.

Virtual LANs

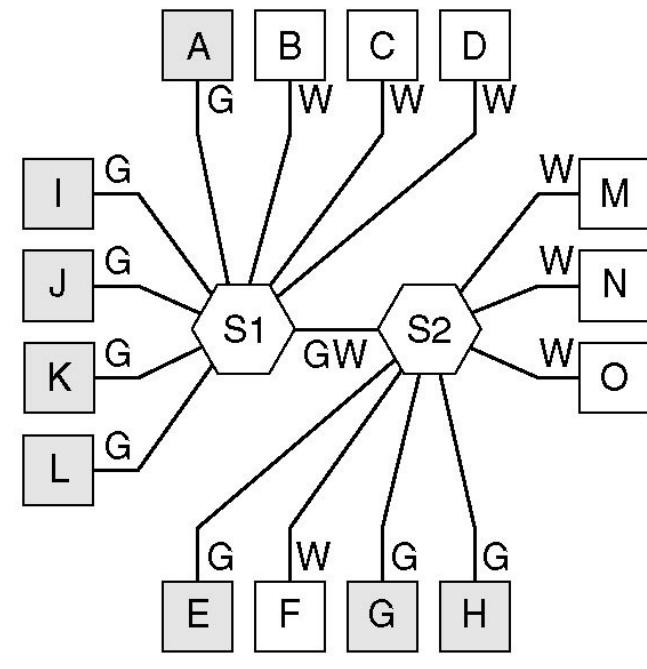


A building with centralized wiring using hubs and a switch.

Virtual LANs (2)



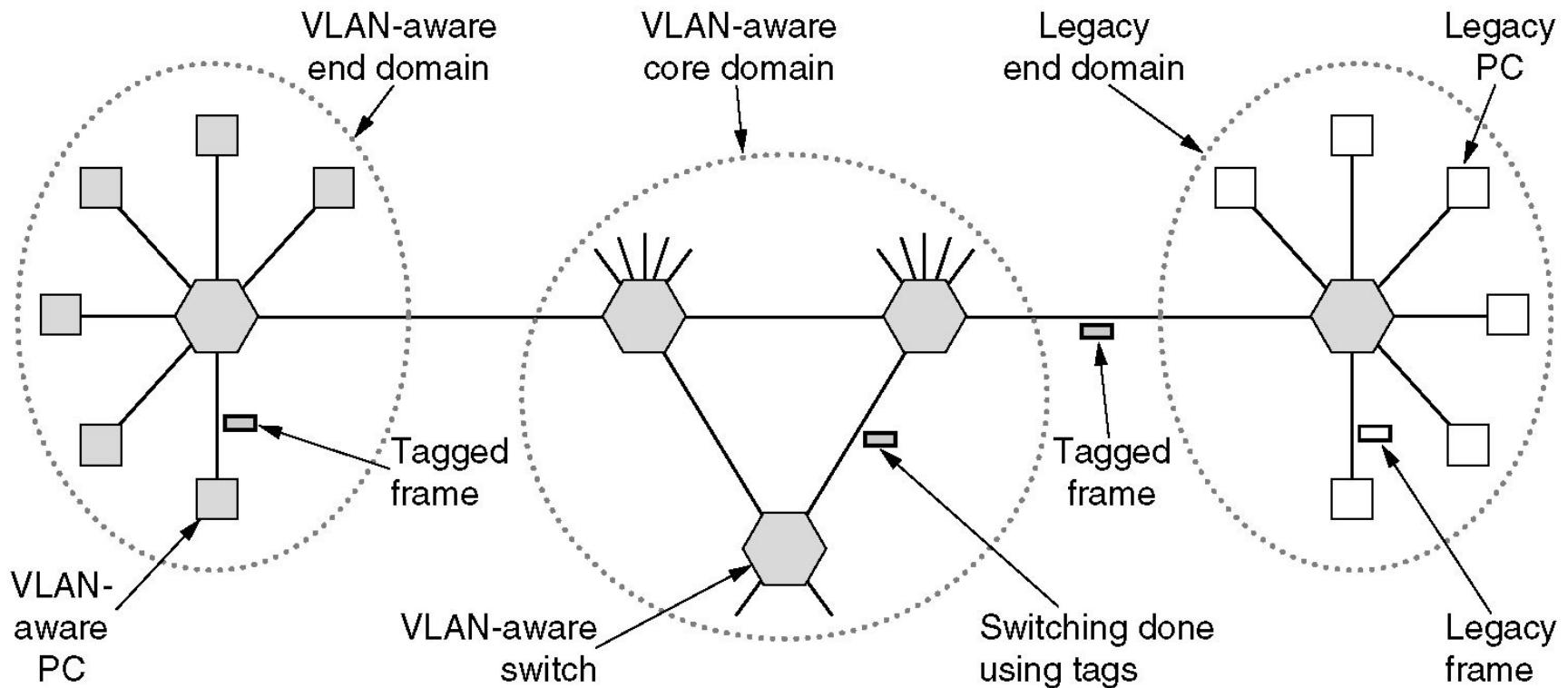
(a)



(b)

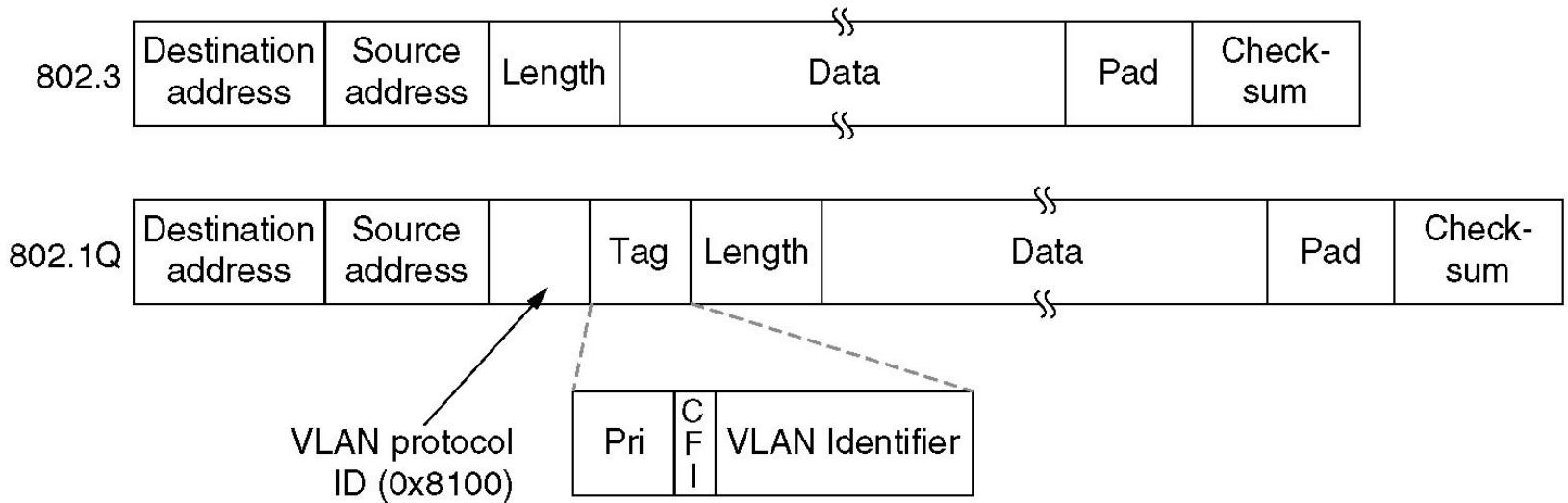
(a) Four physical LANs organized into two VLANs, gray and white, by two bridges. (b) The same 15 machines organized into two VLANs by switches.

The IEEE 802.1Q Standard



Transition from legacy Ethernet to VLAN-aware Ethernet. The shaded symbols are VLAN aware. The empty ones are not.

The IEEE 802.1Q Standard (2)



The 802.3 (legacy) and 802.1Q Ethernet frame formats.

Summary

Method	Description
FDM	Dedicate a frequency band to each station
WDM	A dynamic FDM scheme for fiber
TDM	Dedicate a time slot to each station
Pure ALOHA	Unsynchronized transmission at any instant
Slotted ALOHA	Random transmission in well-defined time slots
1-persistent CSMA	Standard carrier sense multiple access
Nonpersistent CSMA	Random delay when channel is sensed busy
P-persistent CSMA	CSMA, but with a probability of p of persisting
CSMA/CD	CSMA, but abort on detecting a collision
Bit map	Round robin scheduling using a bit map
Binary countdown	Highest numbered ready station goes next
Tree walk	Reduced contention by selective enabling
MACA, MACAW	Wireless LAN protocols
Ethernet	CSMA/CD with binary exponential backoff
FHSS	Frequency hopping spread spectrum
DSSS	Direct sequence spread spectrum
CSMA/CA	Carrier sense multiple access with collision avoidance

Channel allocation methods and systems for a common channel.



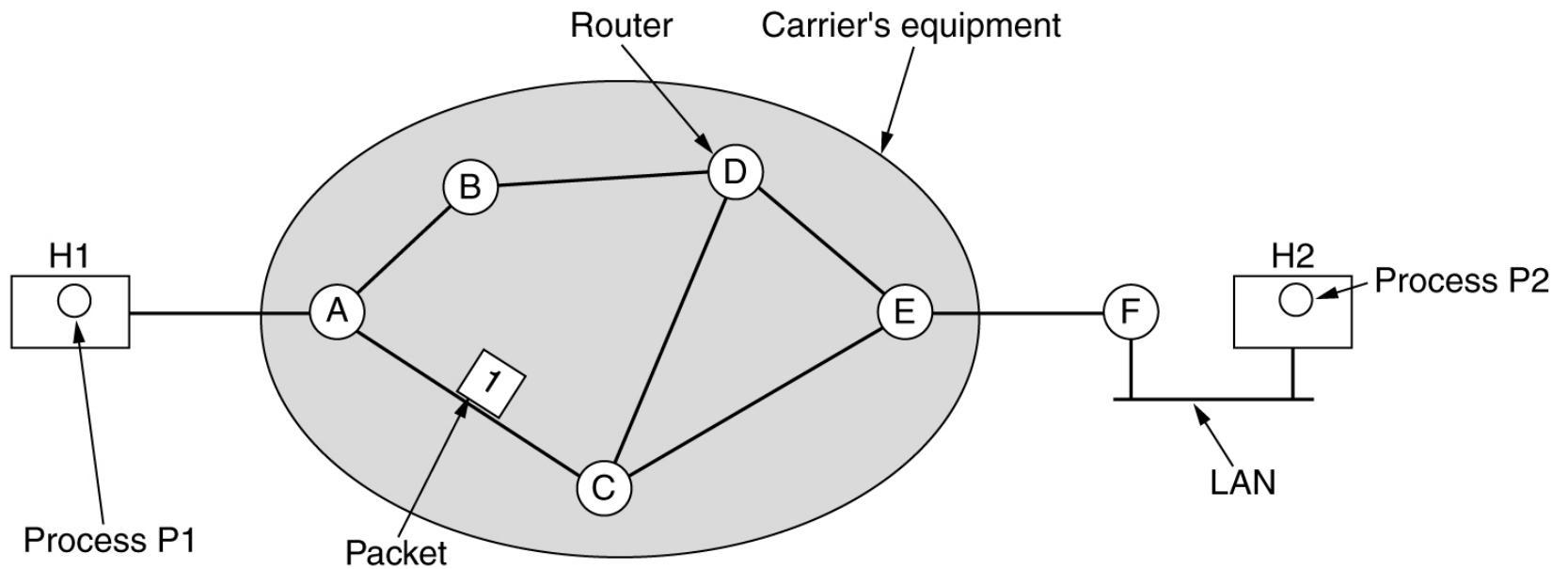
Chapter 5

The Network Layer

Network Layer Design Issues

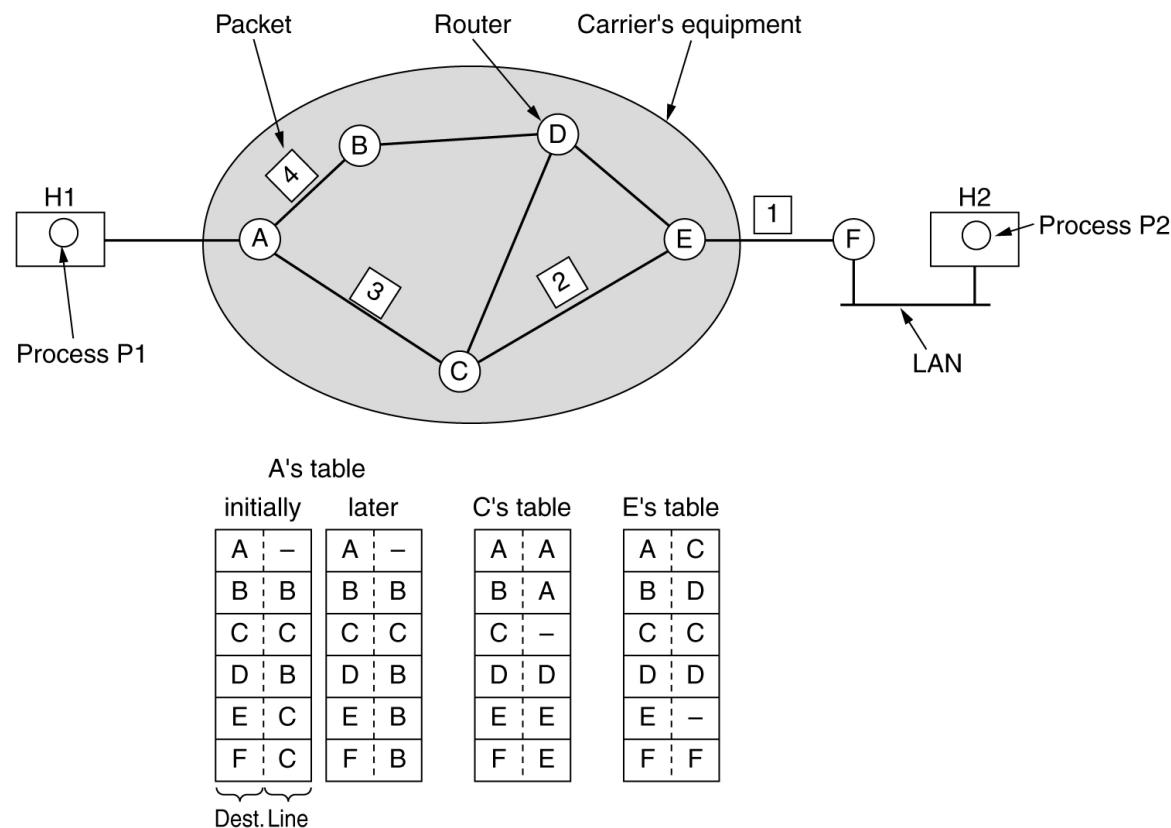
- Store-and-Forward Packet Switching
- Services Provided to the Transport Layer
- Implementation of Connectionless Service
- Implementation of Connection-Oriented Service
- Comparison of Virtual-Circuit and Datagram Subnets

Store-and-Forward Packet Switching



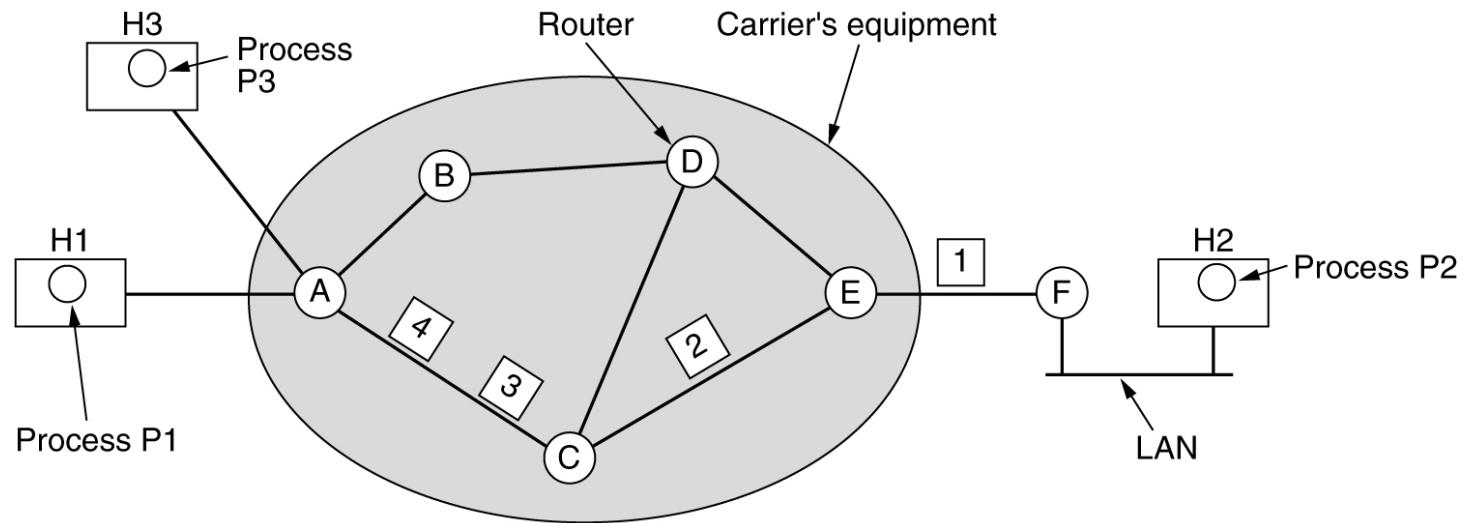
The environment of the network layer protocols.

Implementation of Connectionless Service



Routing within a diagram subnet.

Implementation of Connection-Oriented Service



A's table		C's table		E's table	
H1	1	C	1	E	1
H3	1	C	2	E	2
In		Out			

Routing within a virtual-circuit subnet.

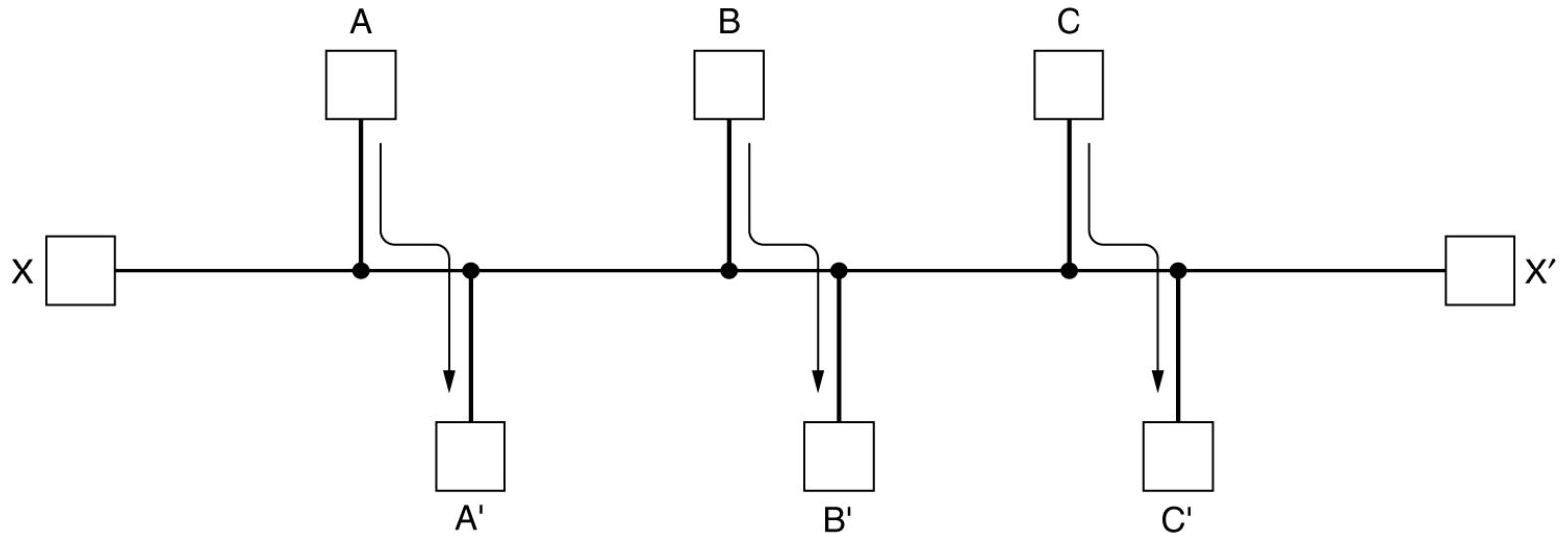
Comparison of Virtual-Circuit and Datagram Subnets

Issue	Datagram subnet	Virtual-circuit subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

Routing Algorithms

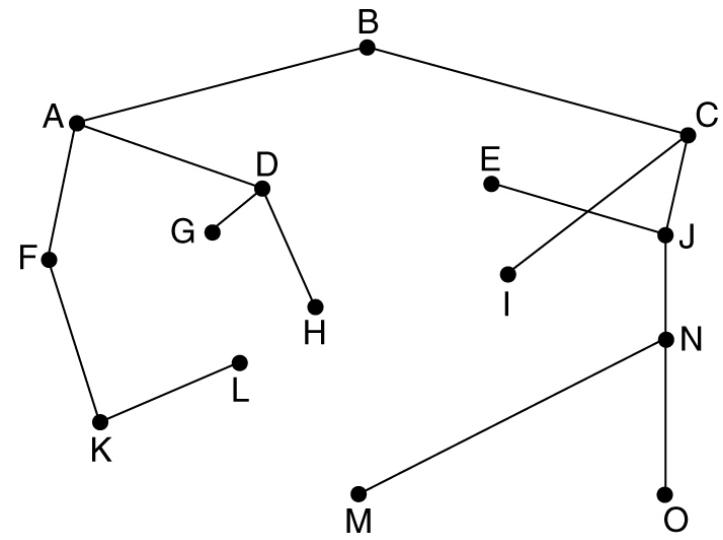
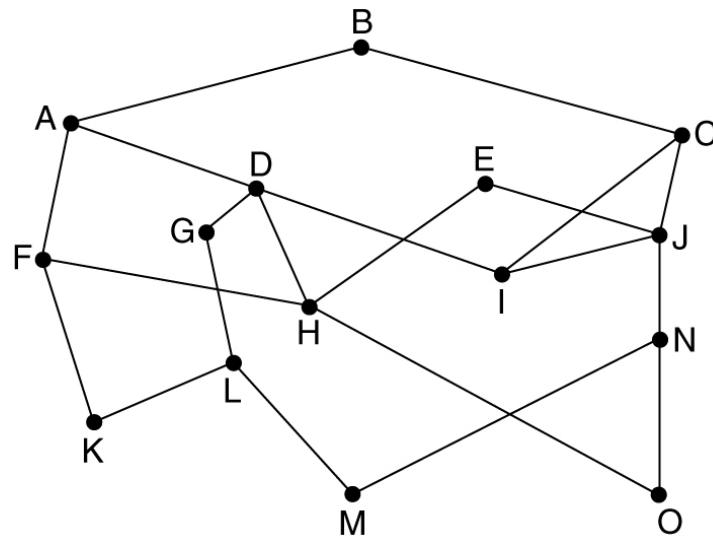
- The Optimality Principle
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- Routing for Mobile Hosts
- Routing in Ad Hoc Networks

Routing Algorithms (2)



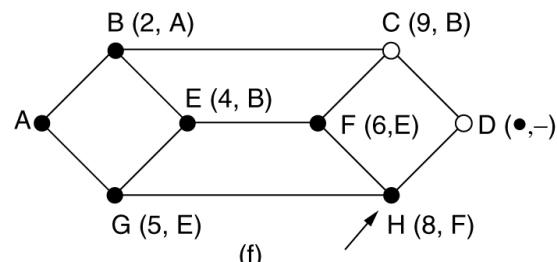
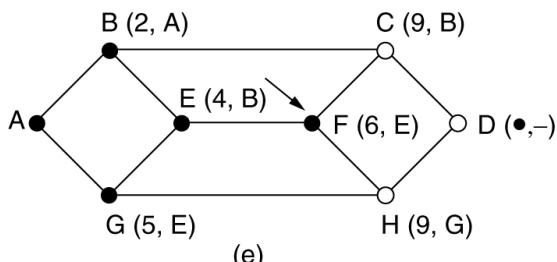
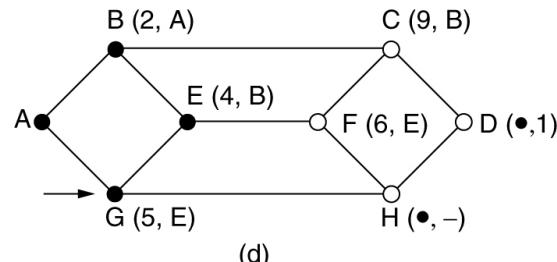
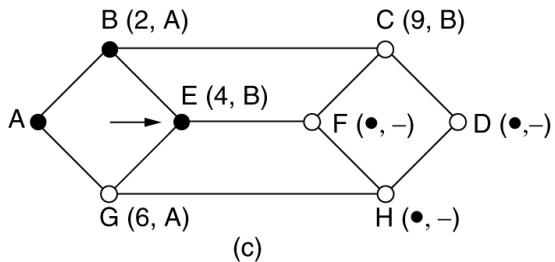
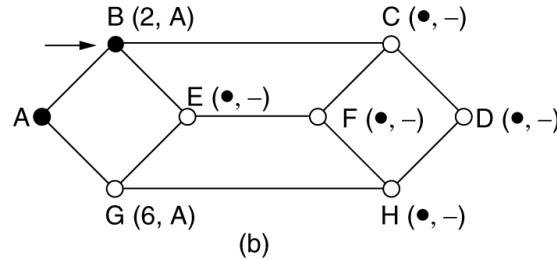
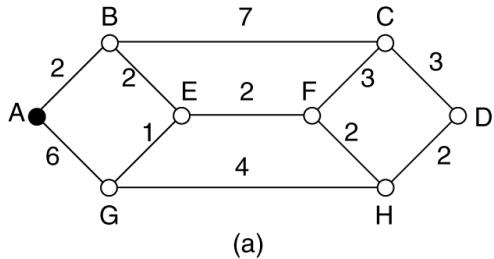
Conflict between fairness and optimality.

The Optimality Principle



(a) A subnet. (b) A sink tree for router B.

Shortest Path Routing



The first 5 steps used in computing the shortest path from A to D.
The arrows indicate the working node.

Flooding

```
#define MAX_NODES 1024           /* maximum number of nodes */
#define INFINITY 1000000000       /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                  /* the path being worked on */
    int predecessor;             /* previous node */
    int length;                  /* length from source to this node */
    enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];

int i, k, min;
struct state *p;

for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;                                /* k is the initial working node */
```

Dijkstra's algorithm to compute the shortest path through a graph.

Flooding (2)

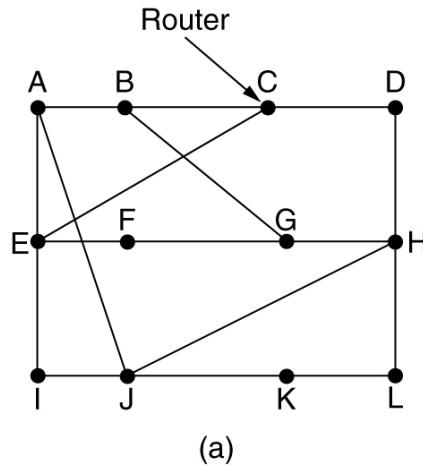
```
do {                                /* Is there a better path from k? */
    for (i = 0; i < n; i++)          /* this graph has n nodes */
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }
}

/* Find the tentatively labeled node with the smallest label. */
k = 0; min = INFINITY;
for (i = 0; i < n; i++)
    if (state[i].label == tentative && state[i].length < min) {
        min = state[i].length;
        k = i;
    }
state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor;} while (k >= 0);
}
```

Dijkstra's algorithm to compute the shortest path through a graph.

Distance Vector Routing



New estimated delay from J

To	A	I	H	K	Line
A	0	24	20	21	8 A
B	12	36	31	28	20 A
C	25	18	19	36	28 I
D	40	27	8	24	20 H
E	14	7	30	22	17 I
F	23	20	19	40	30 I
G	18	31	6	31	18 H
H	17	20	0	19	12 H
I	21	0	14	22	10 I
J	9	11	7	10	0 -
K	24	22	22	0	6 K
L	29	33	9	9	15 K

JA delay is 8 JI delay is 10 JH delay is 12 JK delay is 6

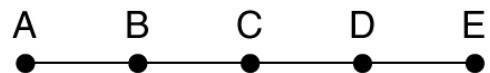
Vectors received from J's four neighbors

New routing table for J

(b)

(a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

Distance Vector Routing (2)



• • • •
Initially
1 • • • After 1 exchange
1 2 • • After 2 exchanges
1 2 3 • After 3 exchanges
1 2 3 4 After 4 exchanges

(a)

A	B	C	D	E	Initially
1	2	3	4	5	After 1 exchange
3	2	3	4	5	After 2 exchanges
3	4	3	4	5	After 3 exchanges
5	6	5	6	7	After 4 exchanges
7	6	7	6	7	After 5 exchanges
7	8	7	8	8	After 6 exchanges
⋮	⋮	⋮	⋮	⋮	⋮
•	•	•	•	•	⋮

(b)

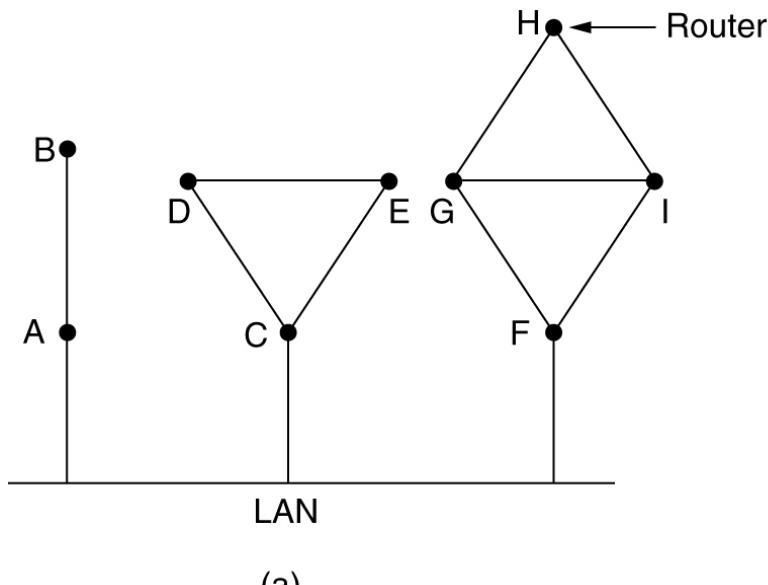
The count-to-infinity problem.

Link State Routing

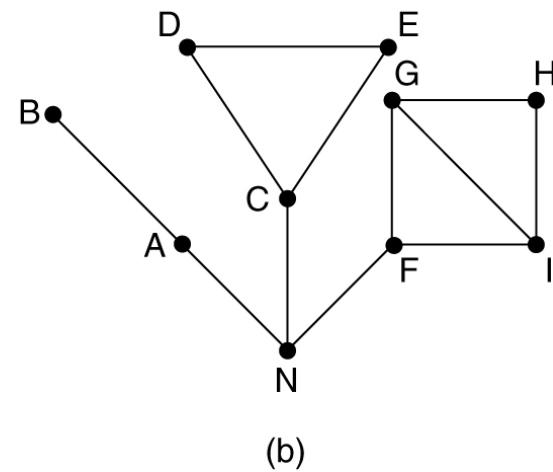
Each router must do the following:

- B. Discover its neighbors, learn their network address.
- C. Measure the delay or cost to each of its neighbors.
- D. Construct a packet telling all it has just learned.
- E. Send this packet to all other routers.
- F. Compute the shortest path to every other router.

Learning about the Neighbors



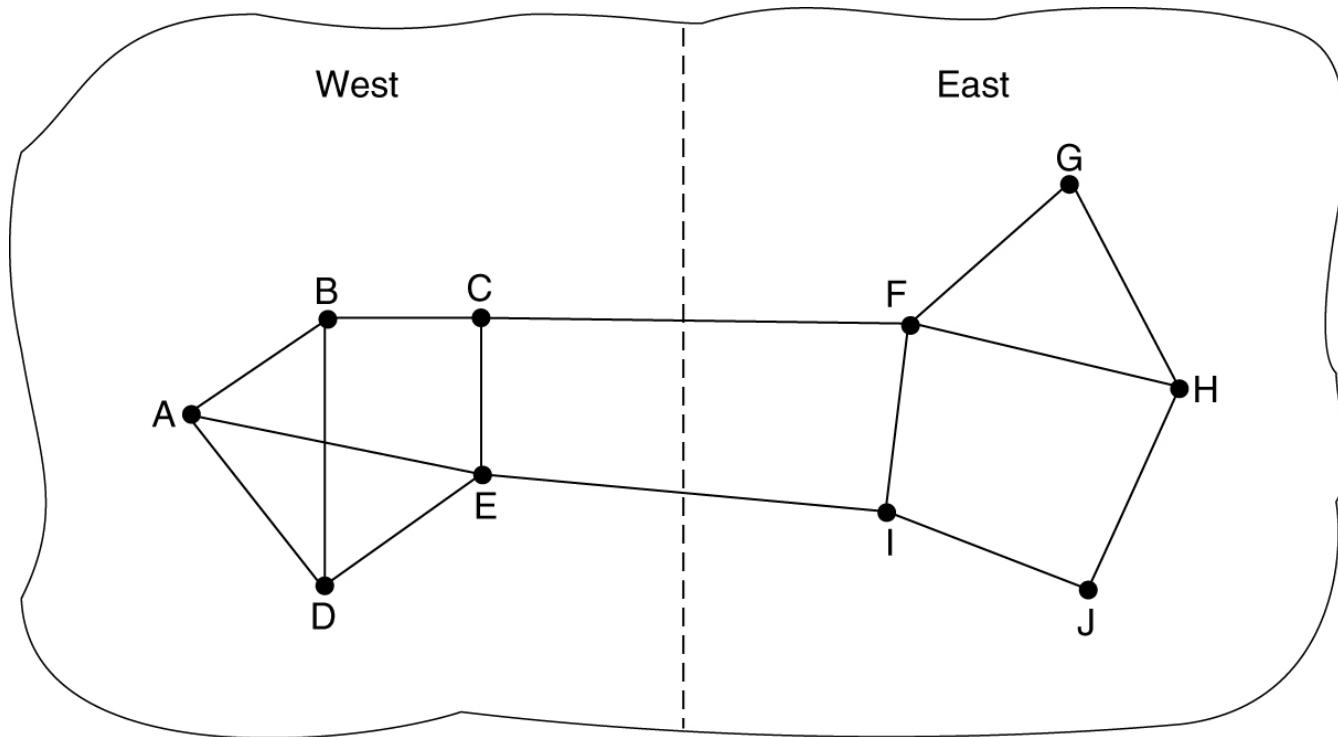
(a)



(b)

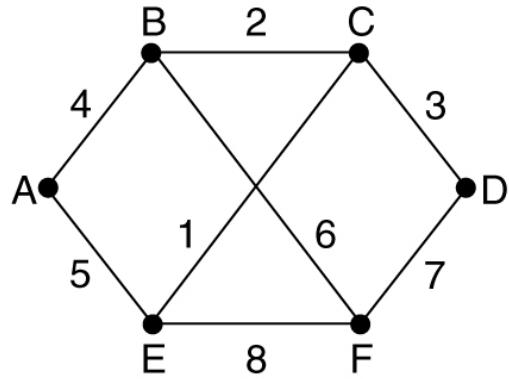
(a) Nine routers and a LAN. (b) A graph model of (a).

Measuring Line Cost



A subnet in which the East and West parts are connected by two lines.

Building Link State Packets



(a)

	Link	State	Packets	
A	B	C	D	E
	Seq.	Seq.	Seq.	Seq.
	Age	Age	Age	Age
	B 4	B 2	C 3	A 5
	A 4	B 2	D 3	B 6
	C 2	D 3	F 7	D 7
	F 6	E 1	F 8	E 8

(b)

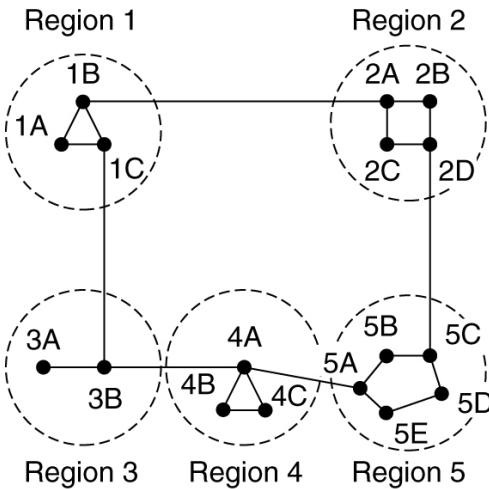
(a) A subnet. (b) The link state packets for this subnet.

Distributing the Link State Packets

Source	Seq.	Age	Send flags			ACK flags			Data
			A	C	F	A	C	F	
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

The packet buffer for router B in the previous slide (Fig. 5-13).

Hierarchical Routing



(a)

Full table for 1A

Dest.	Line	Hops
1A	-	-
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

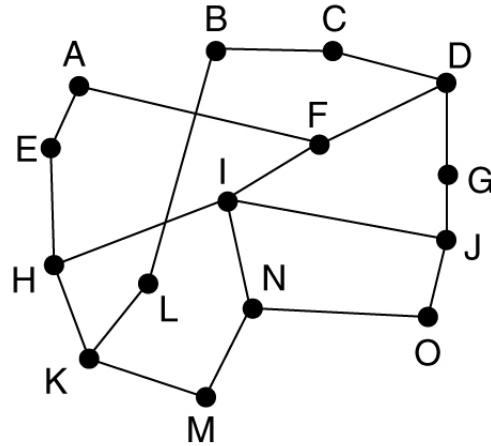
Hierarchical table for 1A

Dest.	Line	Hops
1A	-	-
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

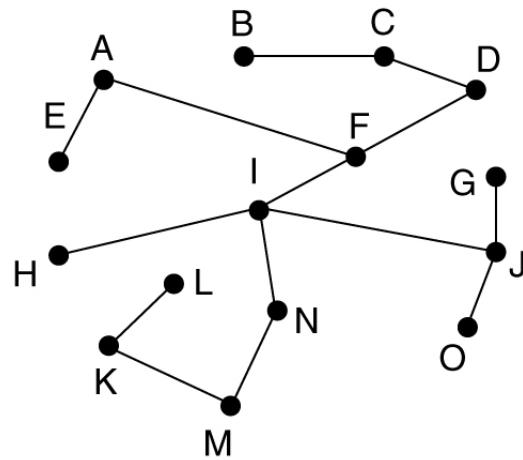
(c)

Hierarchical routing.

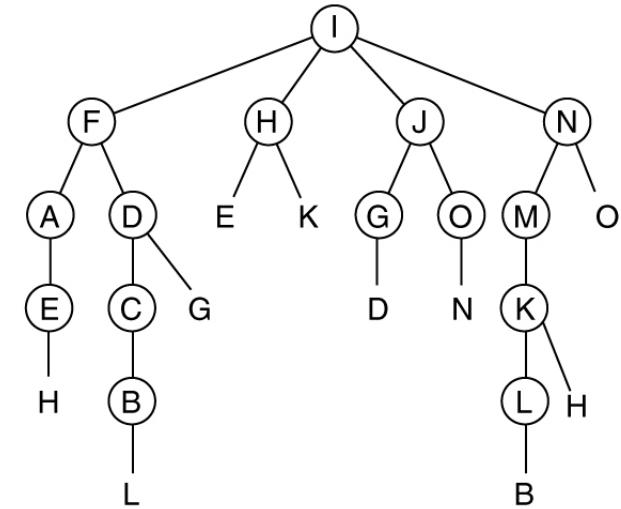
Broadcast Routing



(a)



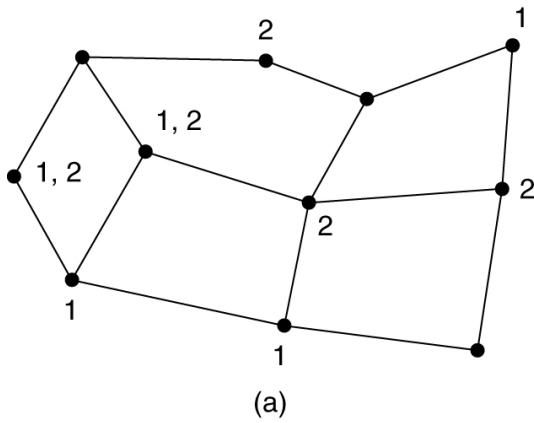
(b)



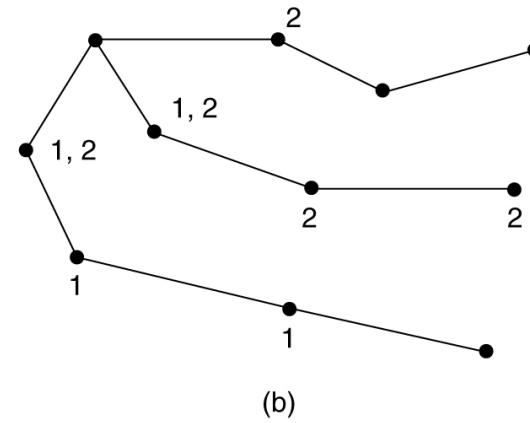
(c)

Reverse path forwarding. (a) A subnet. (b) a Sink tree. (c) The tree built by reverse path forwarding.

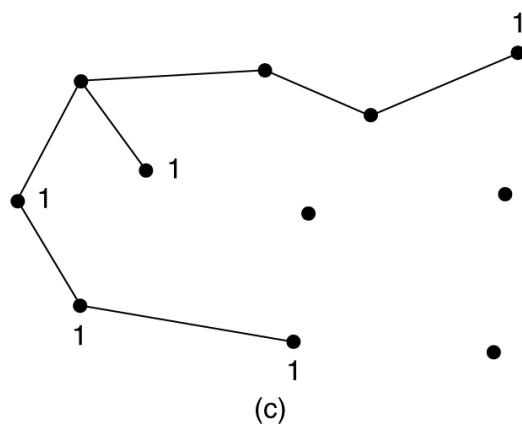
Multicast Routing



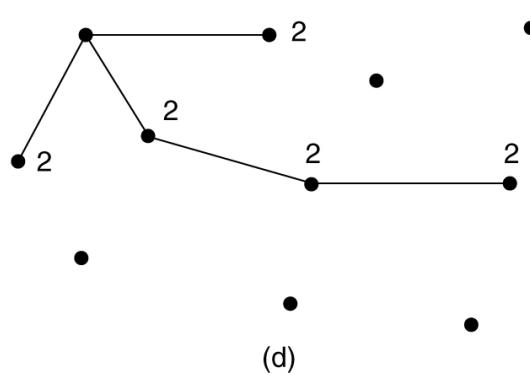
(a)



(b)



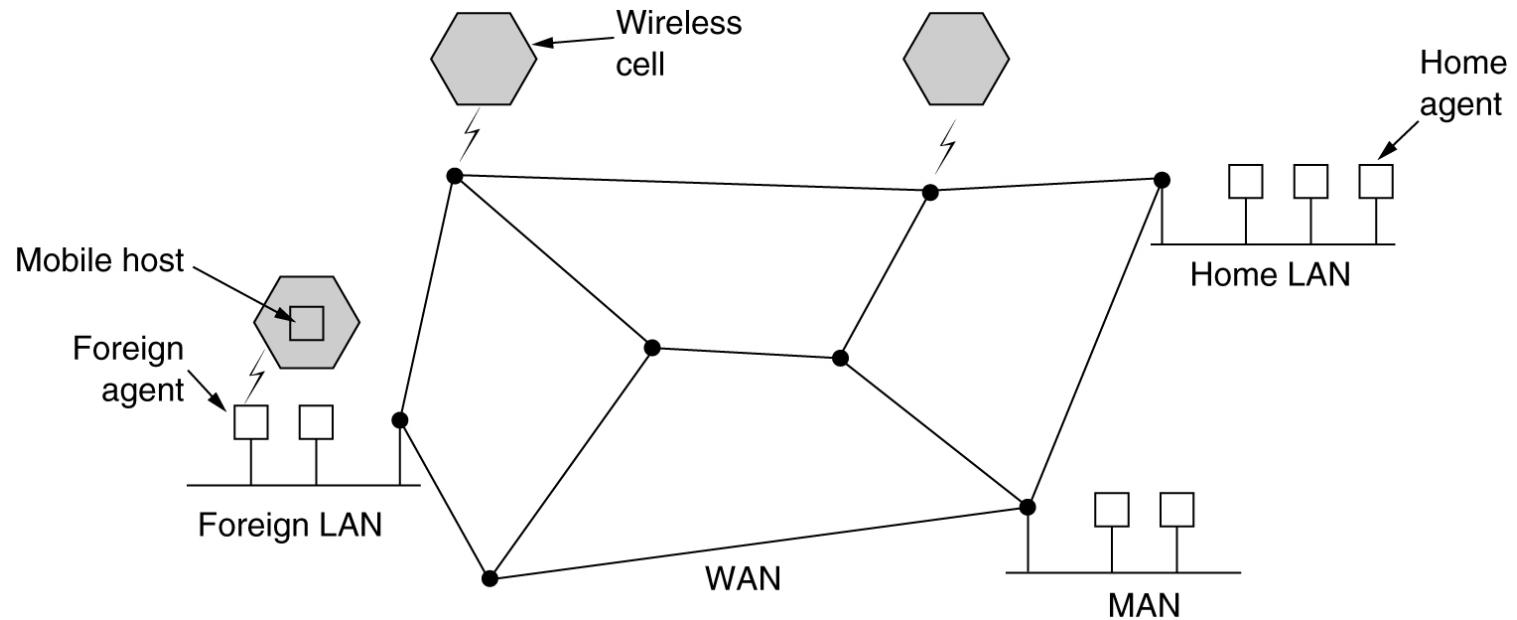
(c)



(d)

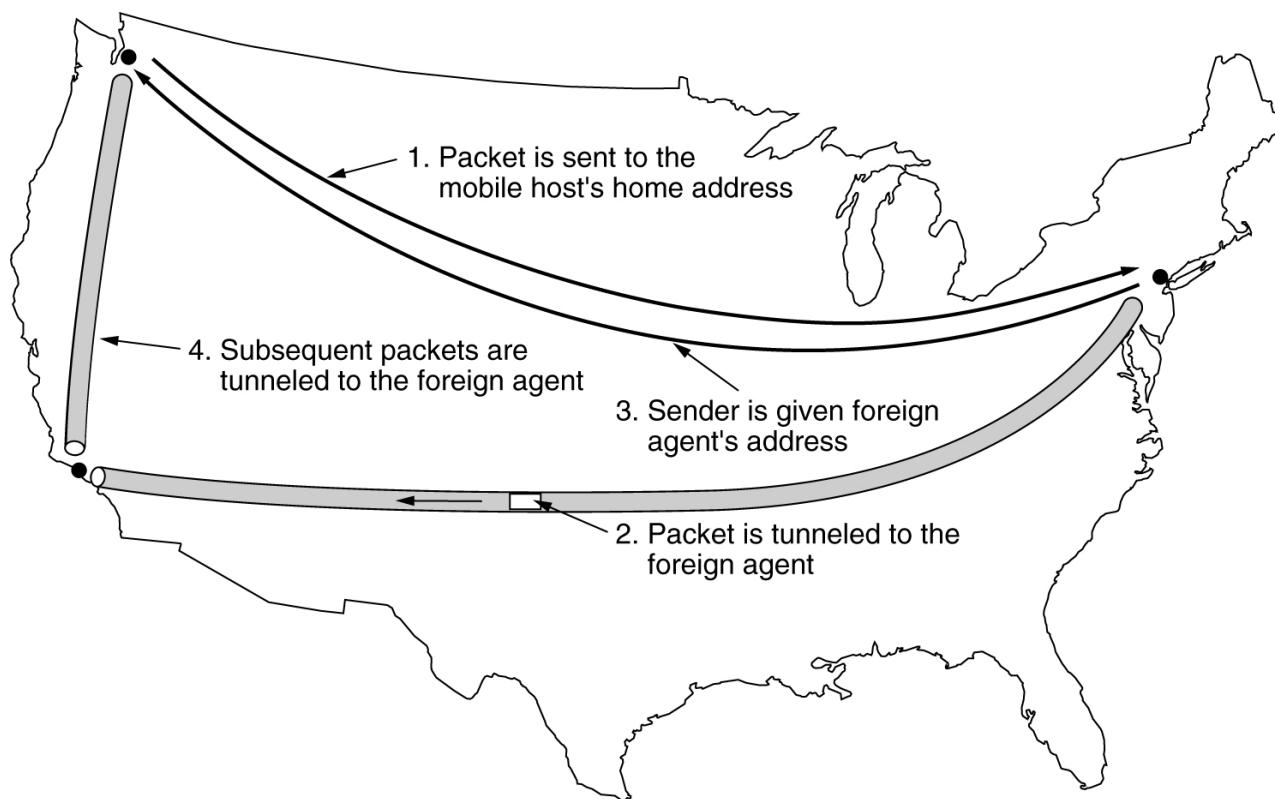
- (a) A network. (b) A spanning tree for the leftmost router.
 - (c) A multicast tree for group 1. (d) A multicast tree for group 2.

Routing for Mobile Hosts



A WAN to which LANs, MANs, and wireless cells are attached.

Routing for Mobile Hosts (2)



Packet routing for mobile users.

Routing in Ad Hoc Networks

Possibilities when the routers are mobile:

B. Military vehicles on battlefield.

- No infrastructure.

C. A fleet of ships at sea.

- All moving all the time

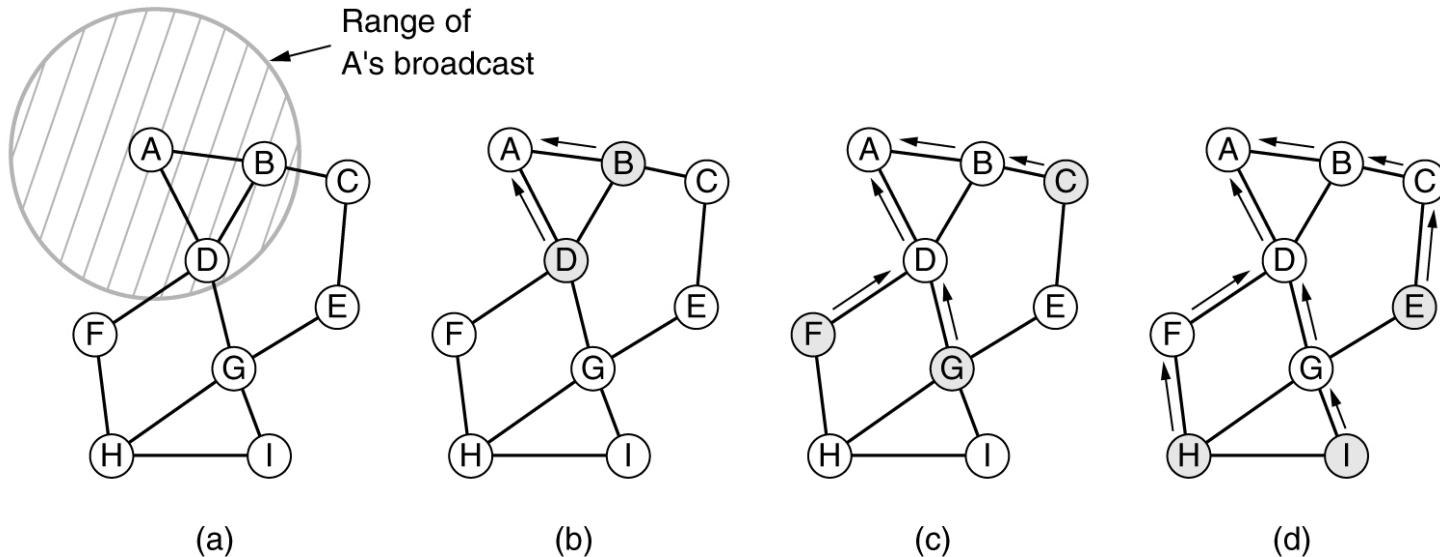
• Emergency works at earthquake .

- The infrastructure destroyed.

E. A gathering of people with notebook computers.

- In an area lacking 802.11.

Route Discovery



- (a) Range of A's broadcast.
- (b) After B and D have received A's broadcast.
- (c) After C, F, and G have received A's broadcast.
- (d) After E, H, and I have received A's broadcast.

Shaded nodes are new recipients. Arrows show possible reverse routes.

Route Discovery (2)

Source address	Request ID	Destination address	Source sequence #	Dest. sequence #	Hop count
----------------	------------	---------------------	-------------------	------------------	-----------

Format of a ROUTE REQUEST packet.

Route Discovery (3)

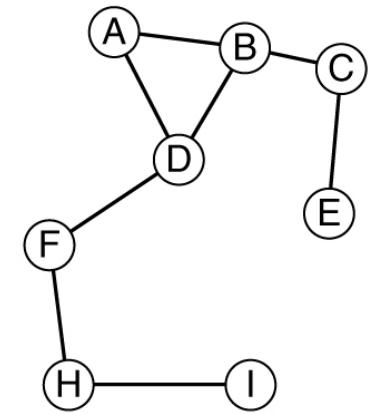
Source address	Destination address	Destination sequence #	Hop count	Lifetime
----------------	---------------------	------------------------	-----------	----------

Format of a ROUTE REPLY packet.

Route Maintenance

Dest.	Next hop	Distance	Active neighbors	Other fields
A	A	1	F, G	
B	B	1	F, G	
C	B	2	F	
E	G	2		
F	F	1	A, B	
G	G	1	A, B	
H	F	2	A, B	
I	G	2	A, B	

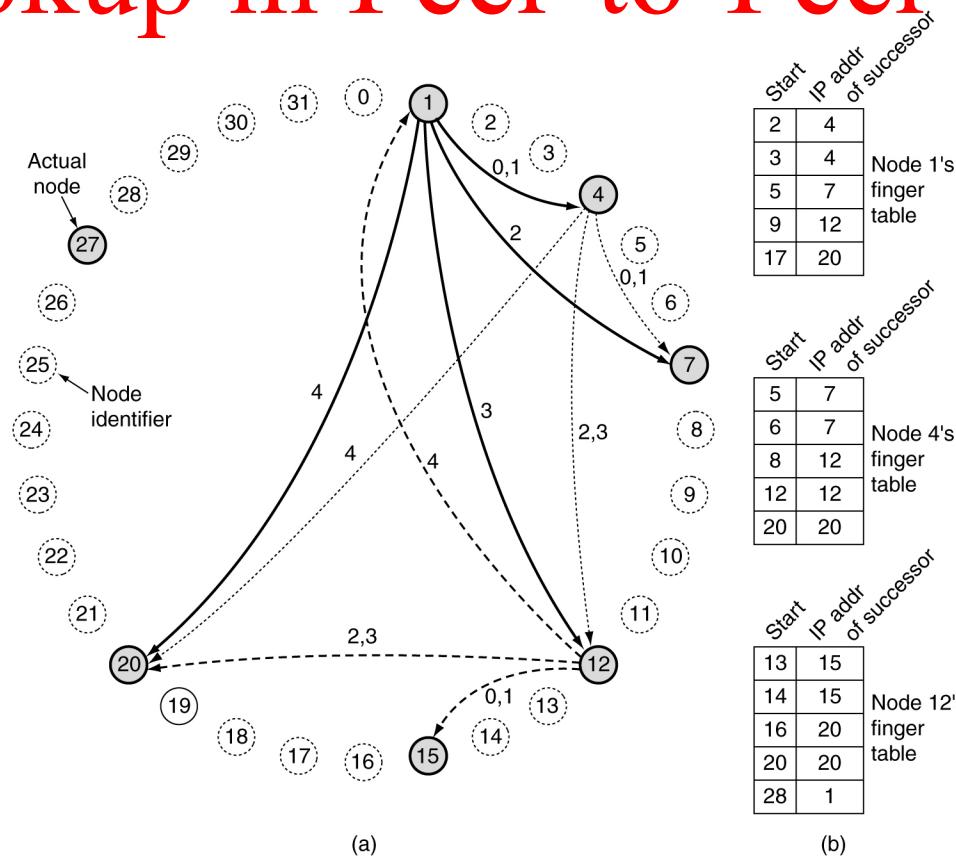
(a)



(b)

- (a) D's routing table before G goes down.
(b) The graph after G has gone down.

Node Lookup in Peer-to-Peer Networks

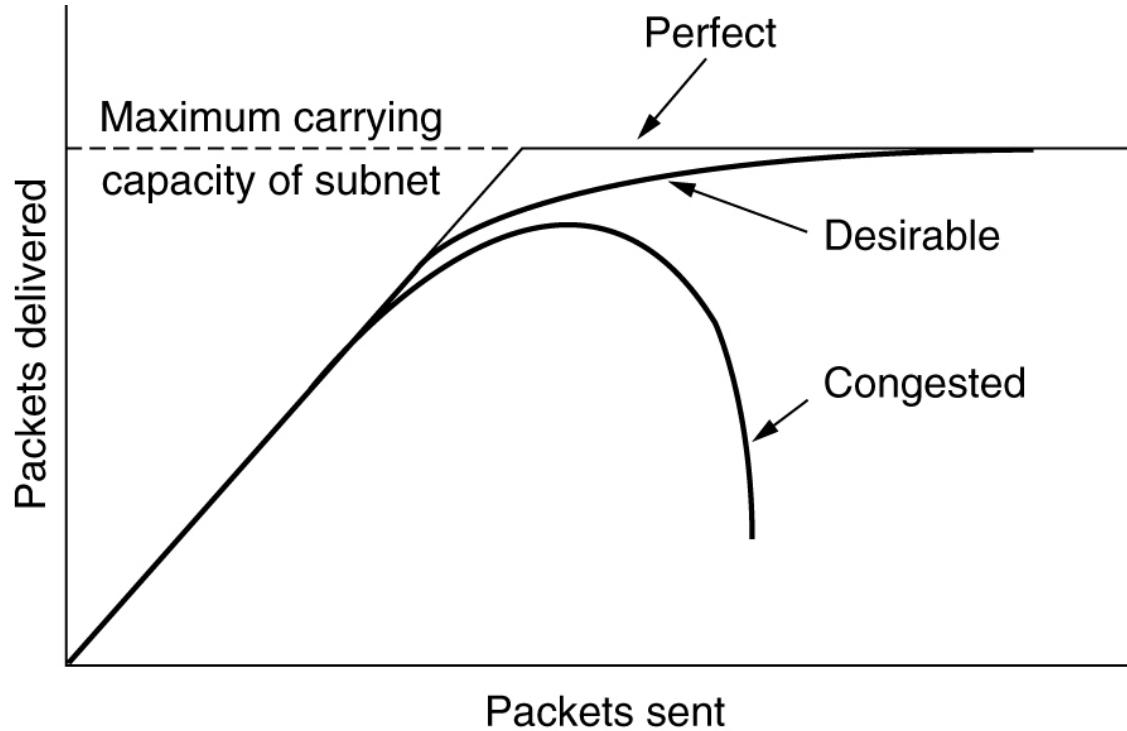


- (a) A set of 32 node identifiers arranged in a circle. The shaded ones correspond to actual machines. The arcs show the fingers from nodes 1, 4, and 12. The labels on the arcs are the table indices.
- (b) Examples of the finger tables.

Congestion Control Algorithms

- General Principles of Congestion Control
- Congestion Prevention Policies
- Congestion Control in Virtual-Circuit Subnets
- Congestion Control in Datagram Subnets
- Load Shedding
- Jitter Control

Congestion



When too much traffic is offered, congestion sets in and performance degrades sharply.

General Principles of Congestion Control

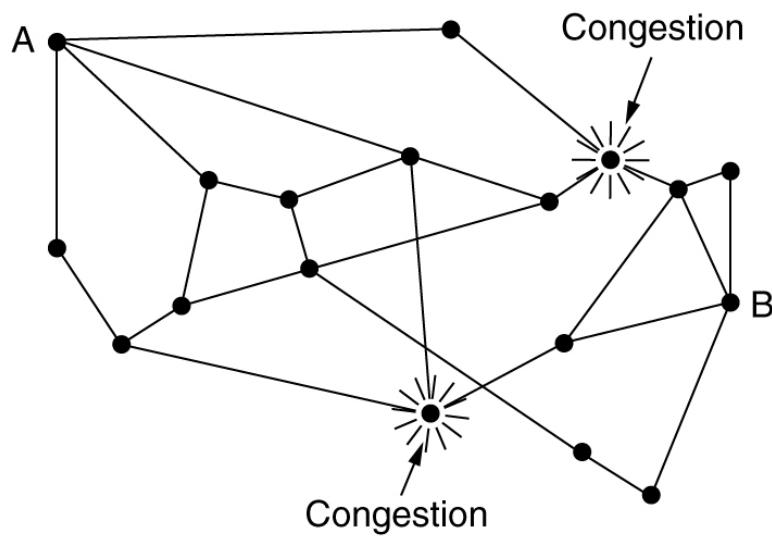
- A. Monitor the system .
 - detect when and where congestion occurs.
- B. Pass information to where action can be taken.
- C. Adjust system operation to correct the problem.

Congestion Prevention Policies

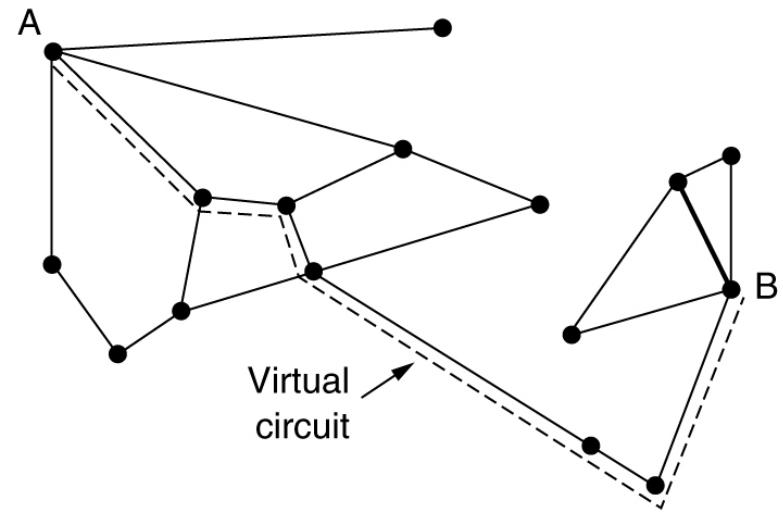
Layer	Policies
Transport	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy• Timeout determination
Network	<ul style="list-style-type: none">• Virtual circuits versus datagram inside the subnet• Packet queueing and service policy• Packet discard policy• Routing algorithm• Packet lifetime management
Data link	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy

Policies that affect congestion.

Congestion Control in Virtual-Circuit Subnets



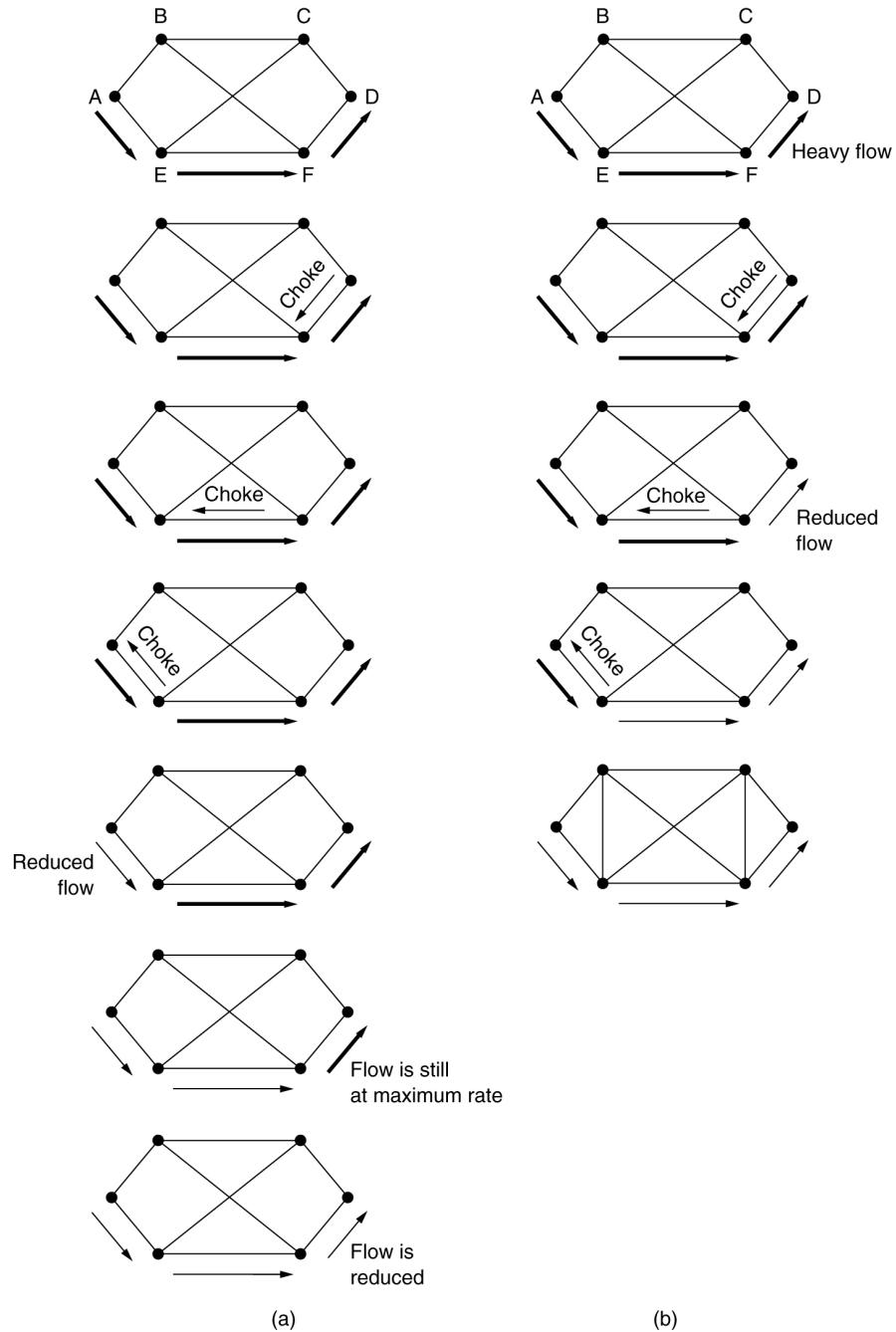
(a)



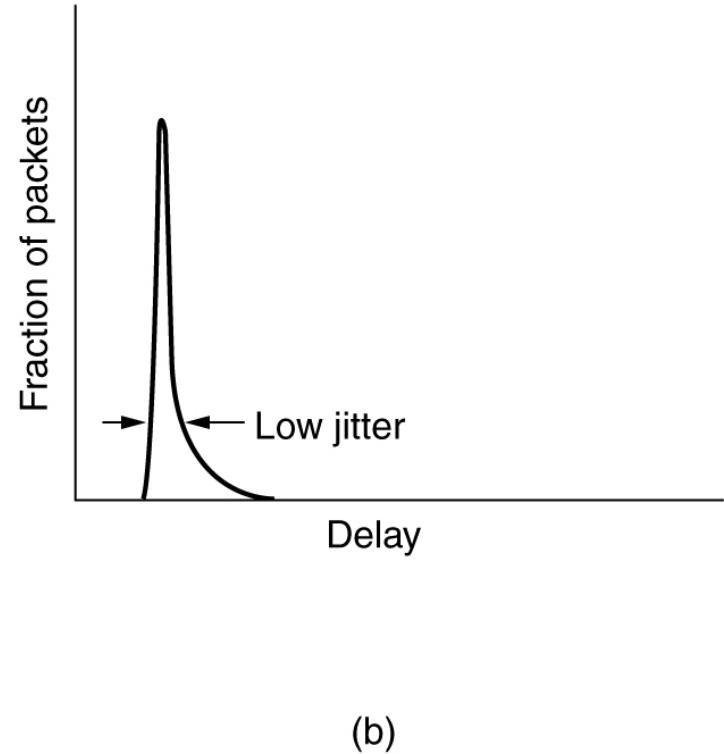
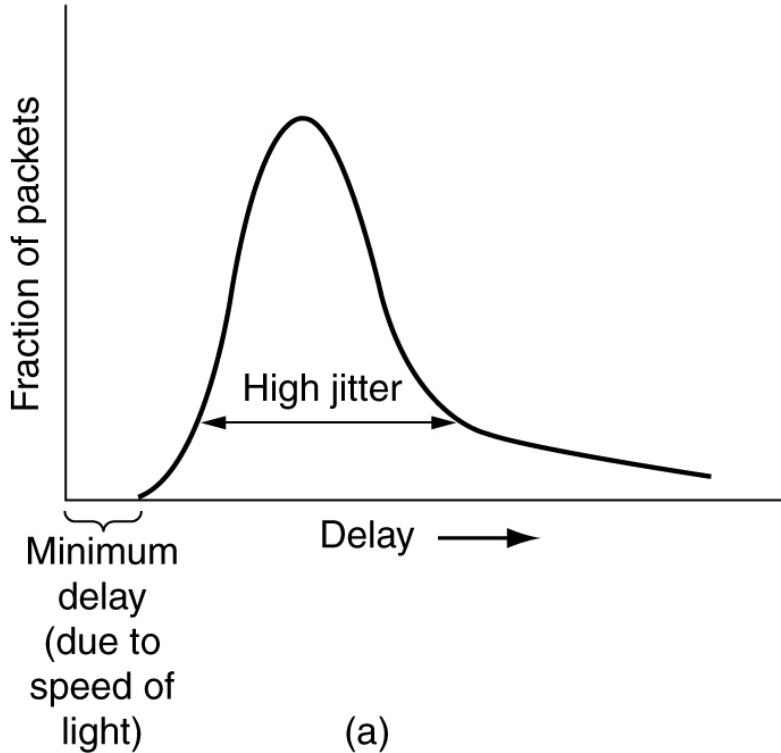
(b)

(a) A congested subnet. (b) A redrawn subnet, eliminates congestion and a virtual circuit from A to B.

Hop-by-Hop Choke Packets



Jitter Control



(a) High jitter.

(b) Low jitter.

Quality of Service

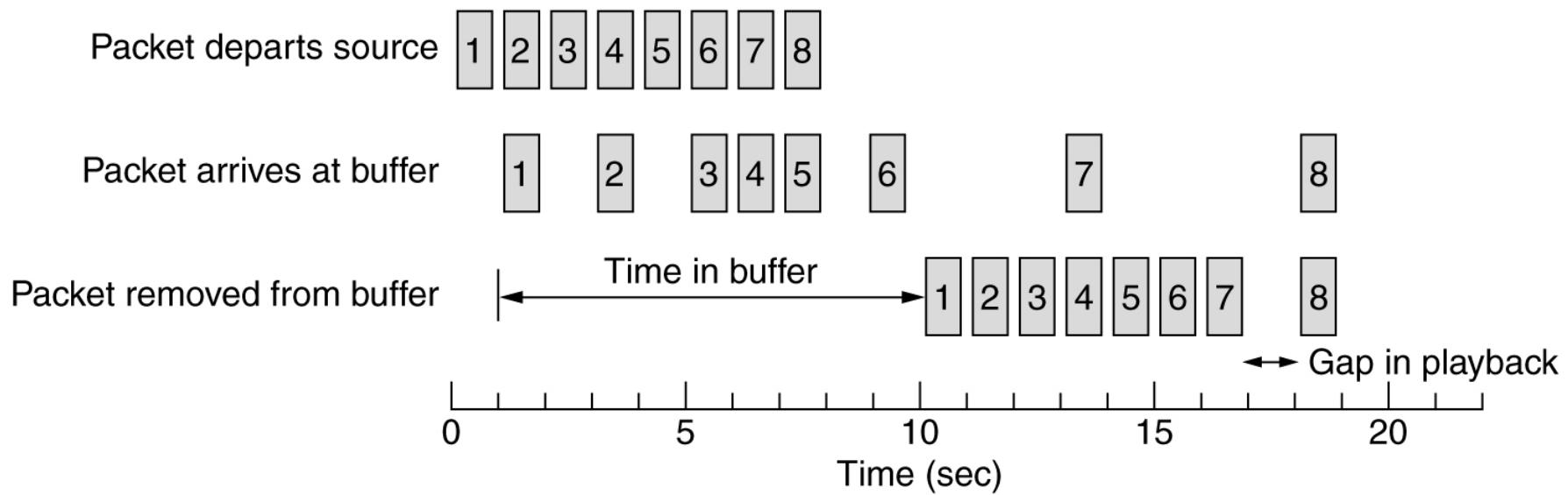
- Requirements
- Techniques for Achieving Good Quality of Service
- Integrated Services
- Differentiated Services
- Label Switching and MPLS

Requirements

Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

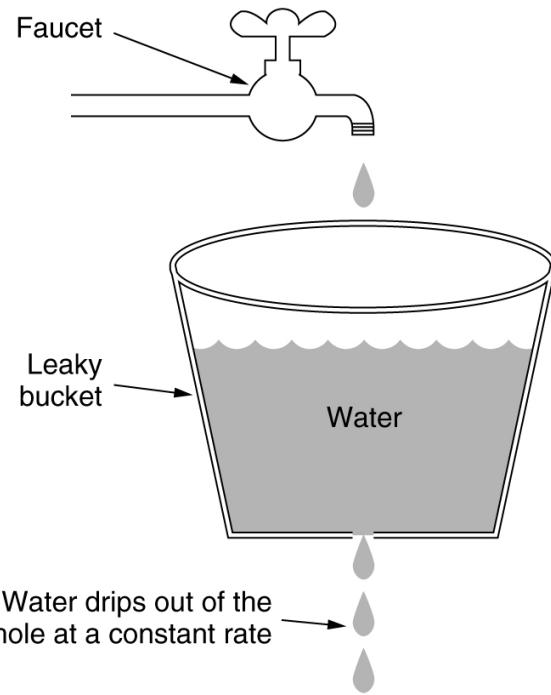
How stringent the quality-of-service requirements are.

Buffering

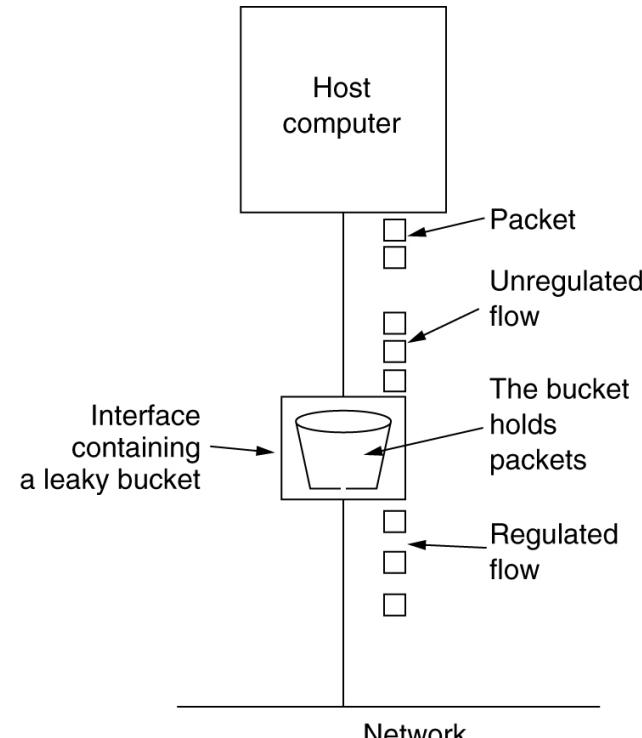


Smoothing the output stream by buffering packets.

The Leaky Bucket Algorithm



(a)

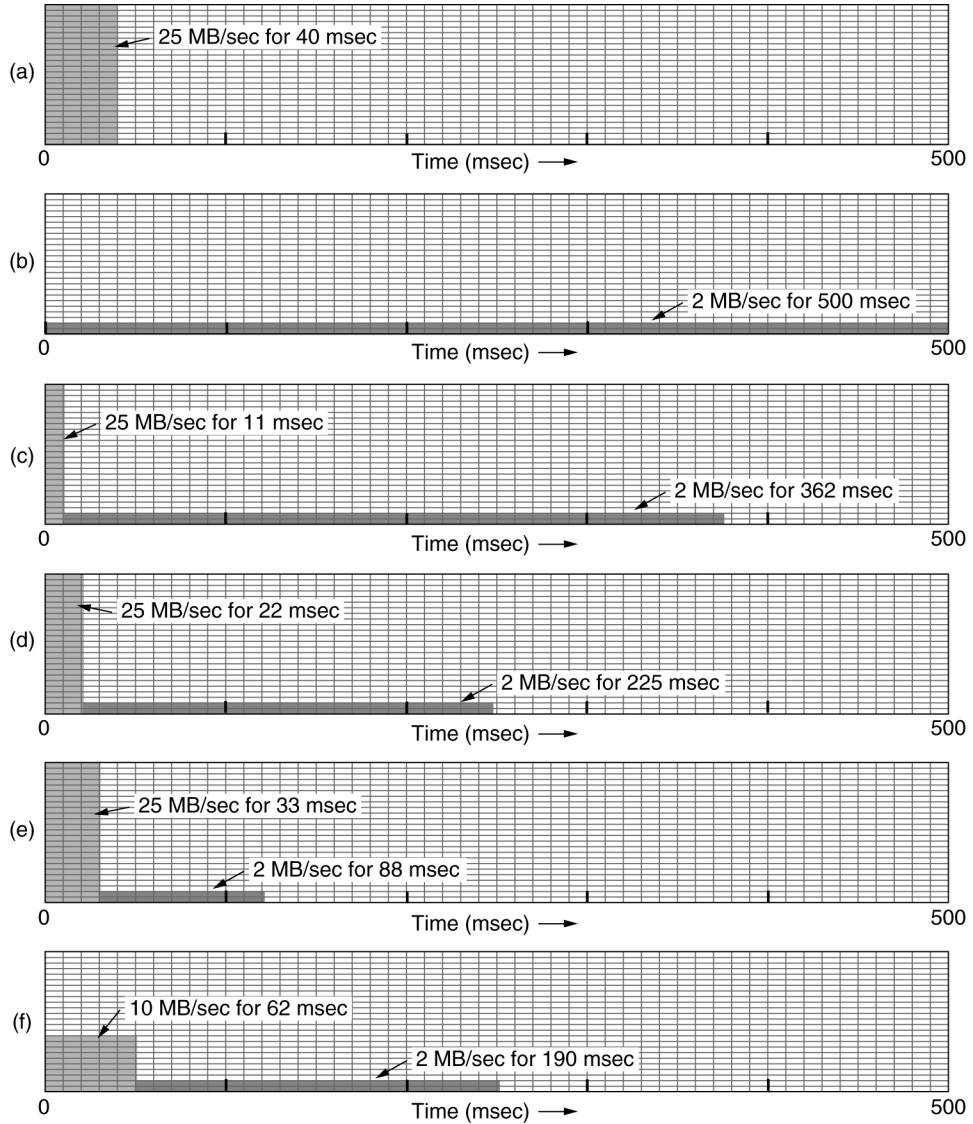


(b)

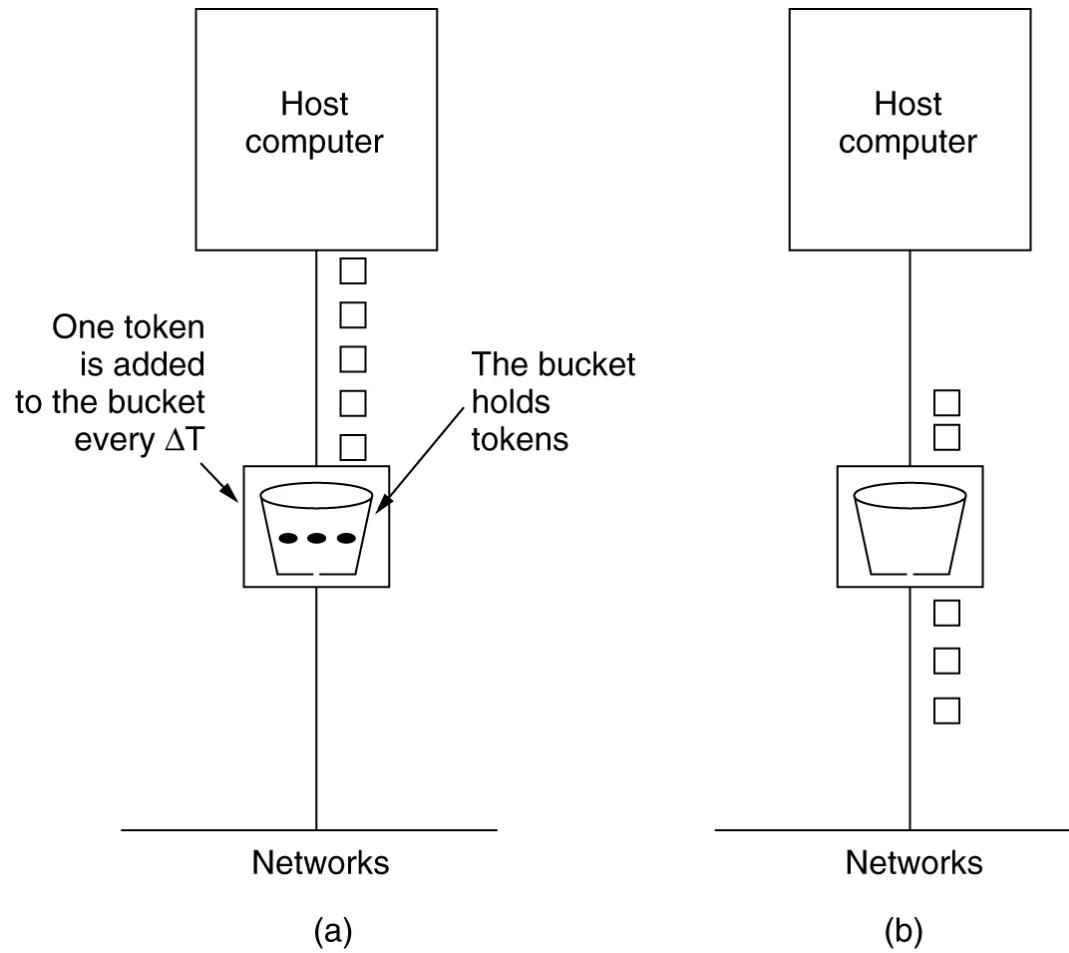
(a) A leaky bucket with water. (b) a leaky bucket with packets.

The Leaky Bucket Algorithm

- (a) Input to a leaky bucket.
- (b) Output from a leaky bucket. Output from a token bucket with capacities of (c) 250 KB, (d) 500 KB, (e) 750 KB, (f) Output from a 500KB token bucket feeding a 10-MB/sec leaky bucket.



The Token Bucket Algorithm



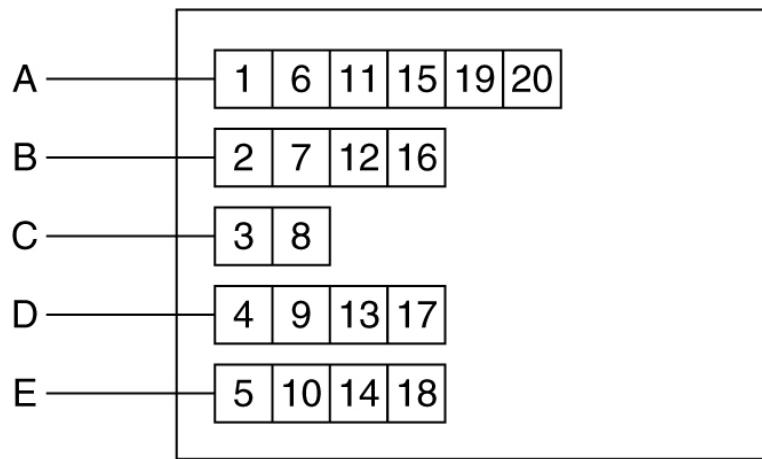
(a) Before. (b) After.

Admission Control

Parameter	Unit
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

An example of flow specification.

Packet Scheduling



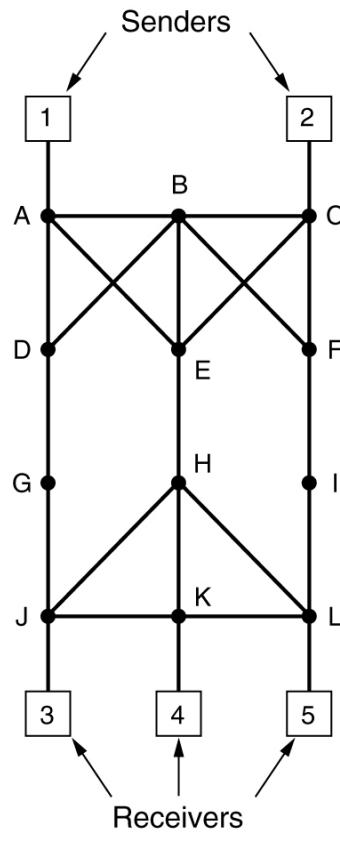
(a)

Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

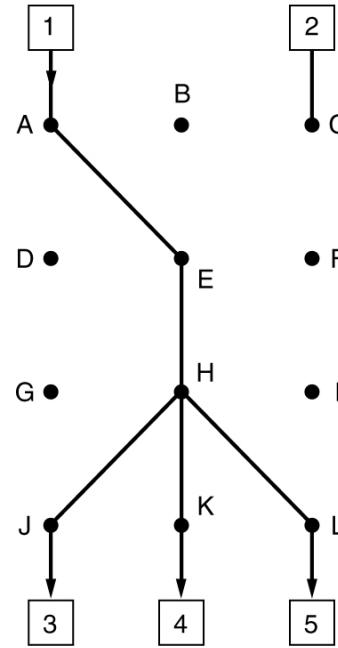
(b)

- (a) A router with five packets queued for line O.
(b) Finishing times for the five packets.

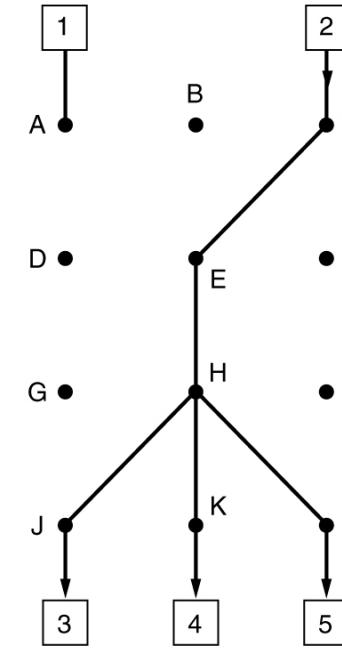
RSVP-The ReSerVation Protocol



(a)



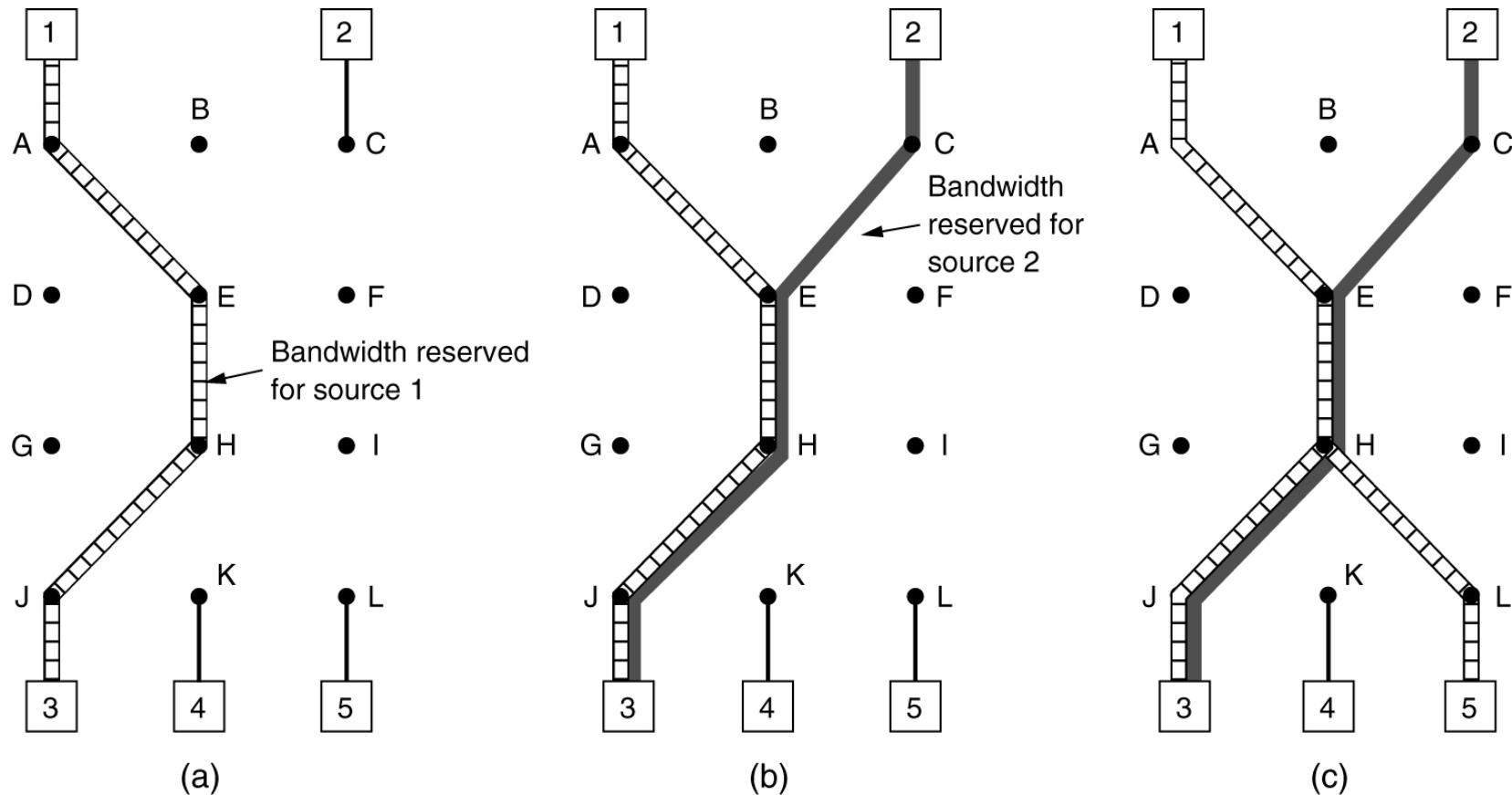
(b)



(c)

- (a) A network, (b) The multicast spanning tree for host 1.
 - (c) The multicast spanning tree for host 2.

RSVP-The ReSerVation Protocol (2)



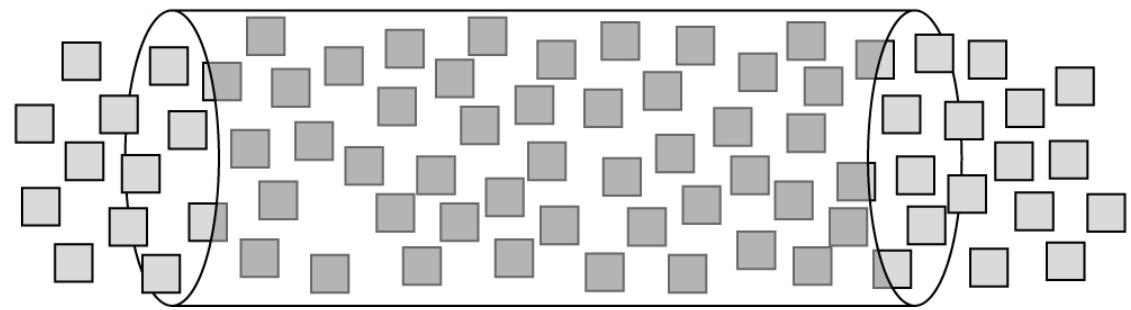
- (a)** Host 3 requests a channel to host 1. **(b)** Host 3 then requests a second channel, to host 2. **(c)** Host 5 requests a channel to host 1.

Expedited Forwarding

Expedited packets →

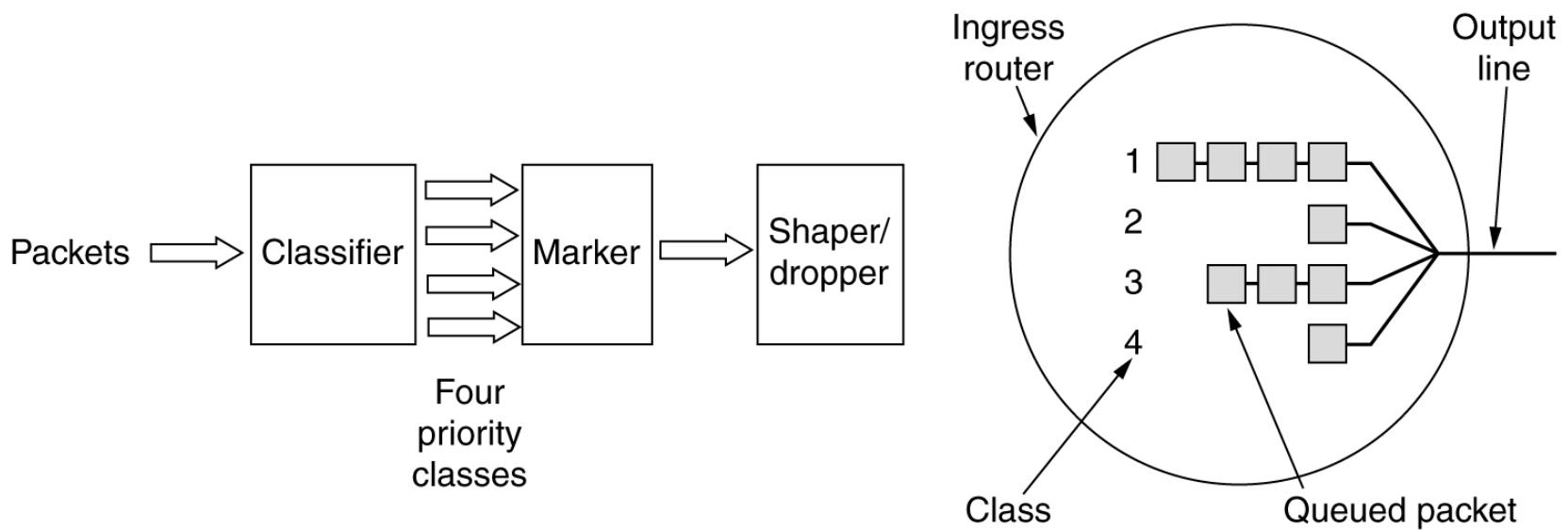


Regular packets →



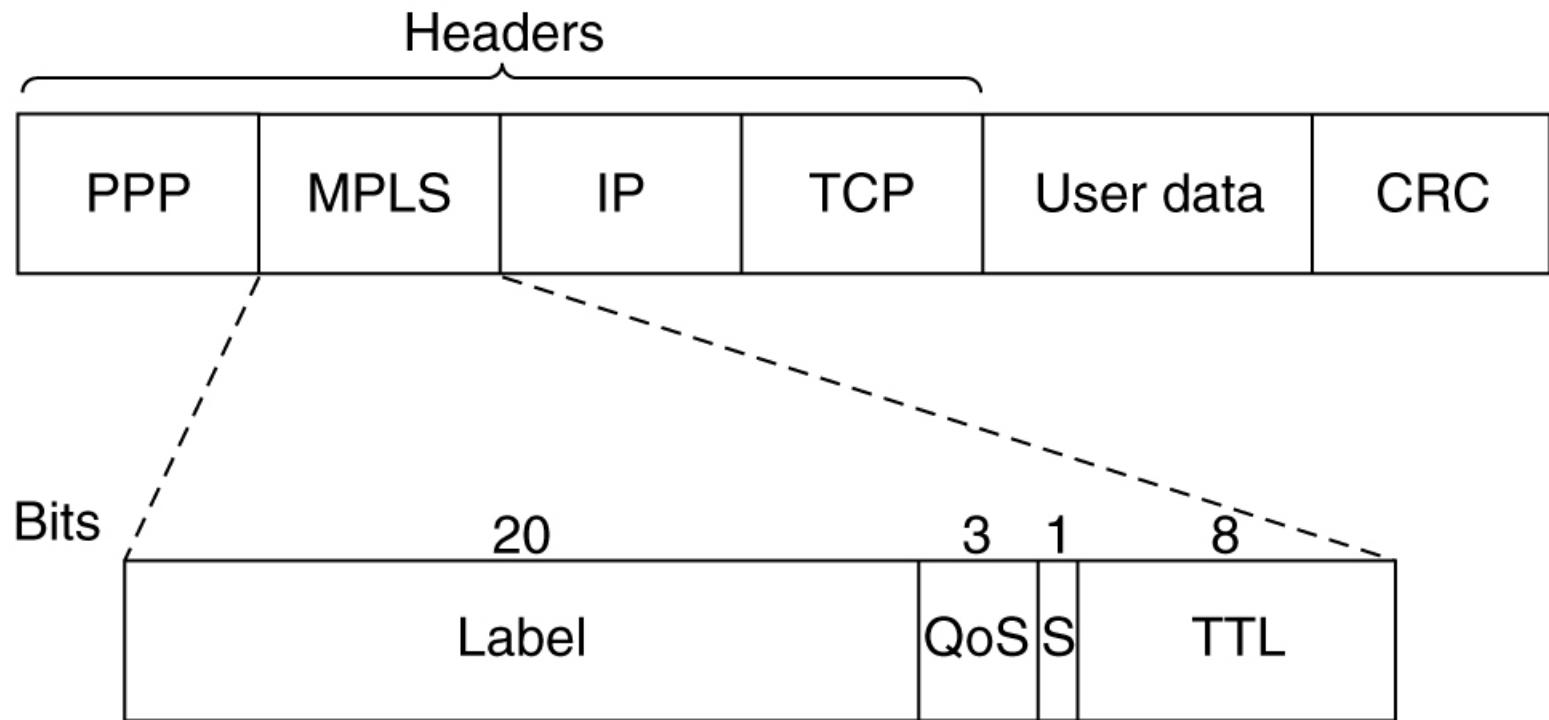
Expedited packets experience a traffic-free network.

Assured Forwarding



A possible implementation of the data flow for assured forwarding.

Label Switching and MPLS

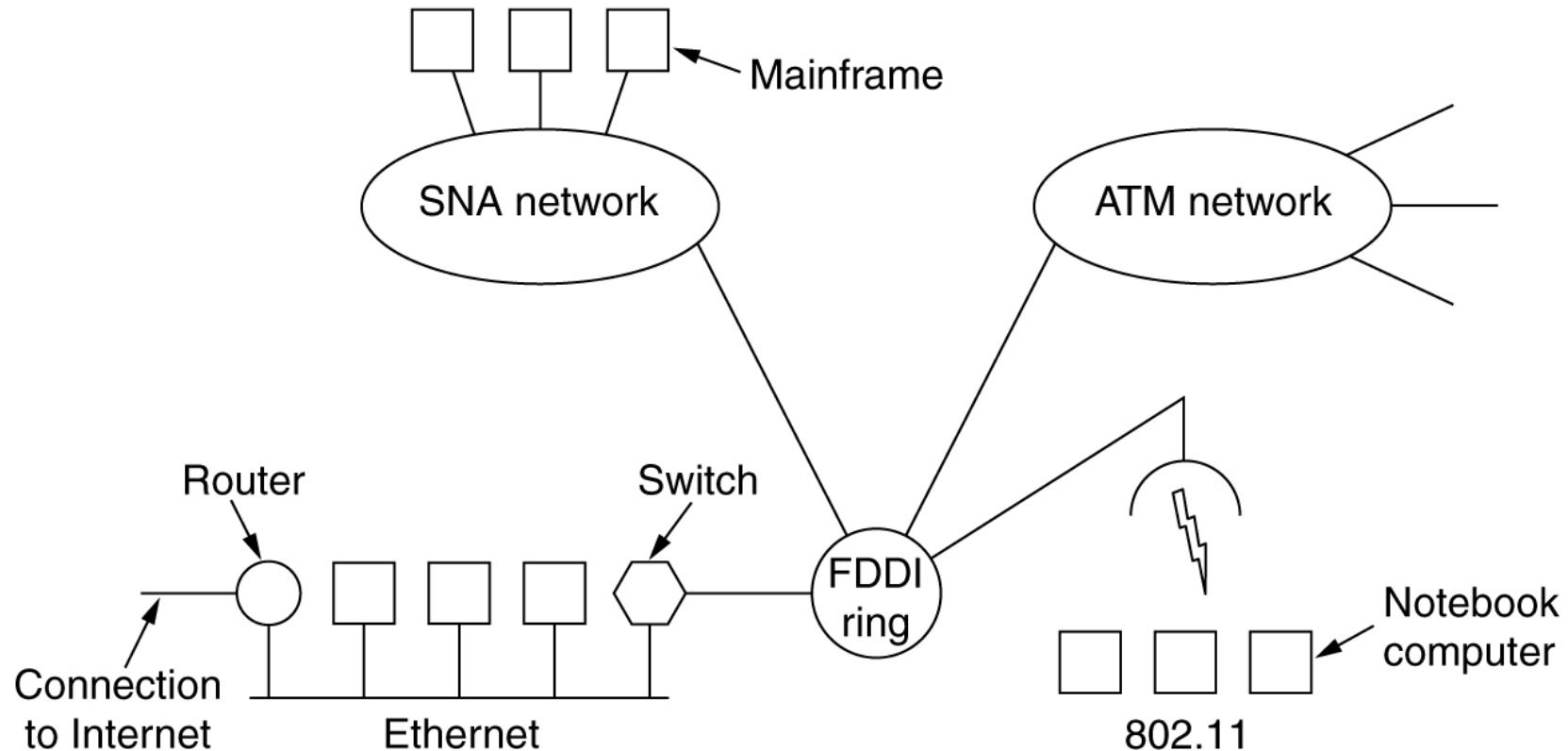


Transmitting a TCP segment using IP, MPLS, and PPP.

Internetworking

- How Networks Differ
- How Networks Can Be Connected
- Concatenated Virtual Circuits
- Connectionless Internetworking
- Tunneling
- Internetwork Routing
- Fragmentation

Connecting Networks



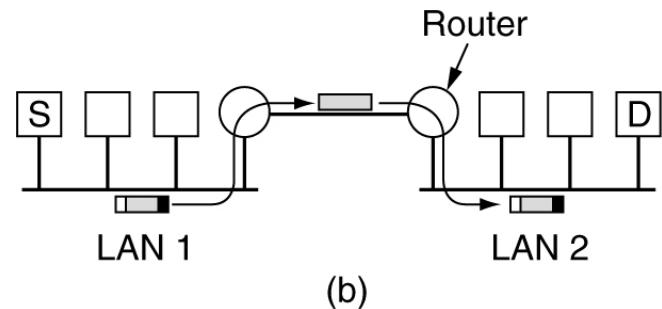
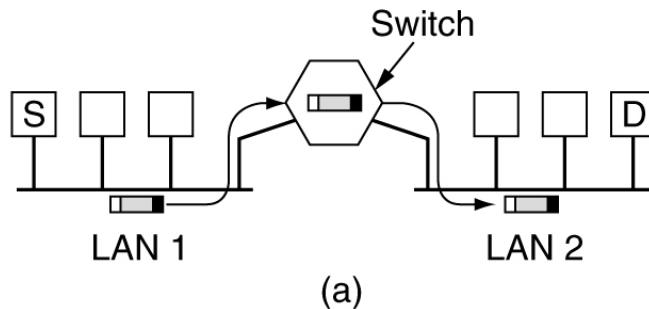
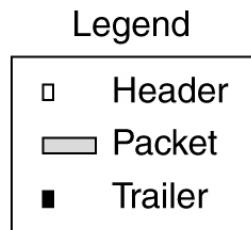
A collection of interconnected networks.

How Networks Differ

Item	Some Possibilities
Service offered	Connection oriented versus connectionless
Protocols	IP, IPX, SNA, ATM, MPLS, AppleTalk, etc.
Addressing	Flat (802) versus hierarchical (IP)
Multicasting	Present or absent (also broadcasting)
Packet size	Every network has its own maximum
Quality of service	Present or absent; many different kinds
Error handling	Reliable, ordered, and unordered delivery
Flow control	Sliding window, rate control, other, or none
Congestion control	Leaky bucket, token bucket, RED, choke packets, etc.
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, by packet, by byte, or not at all

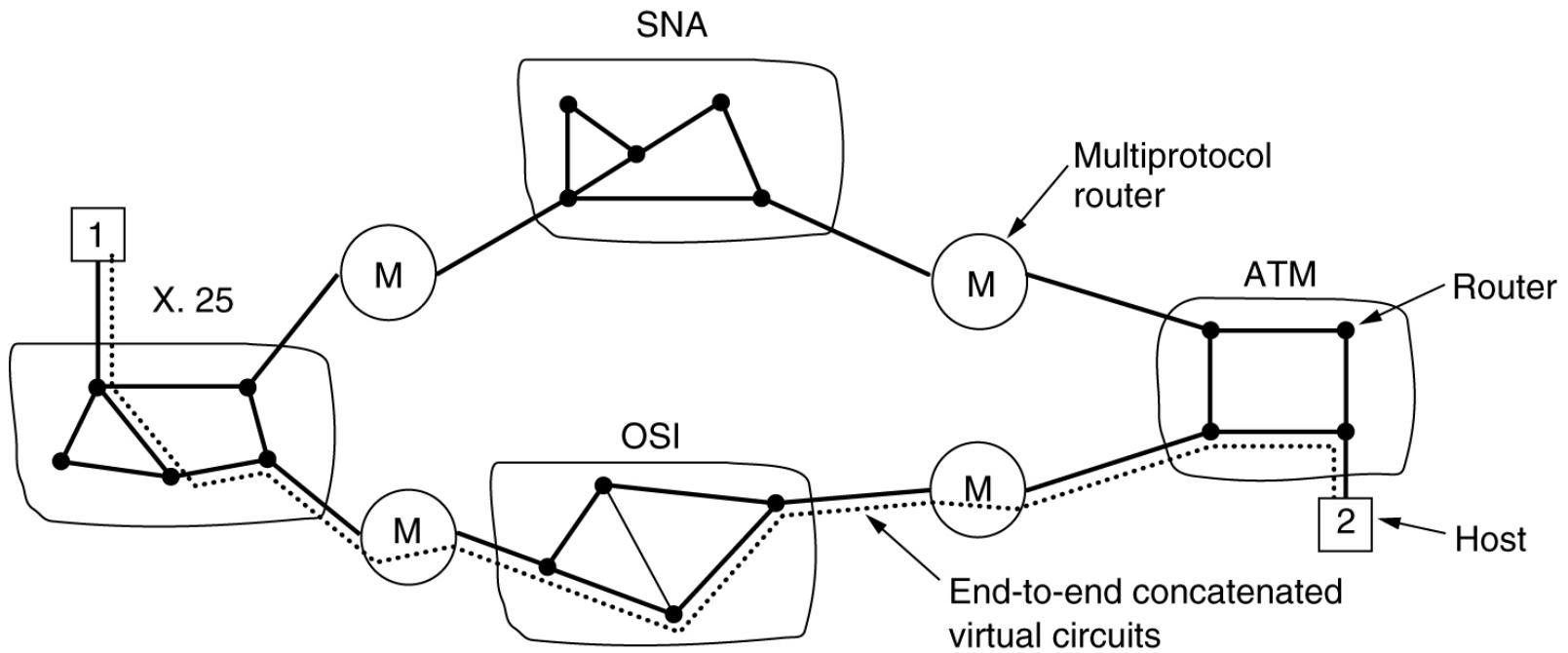
Some of the many ways networks can differ.

How Networks Can Be Connected



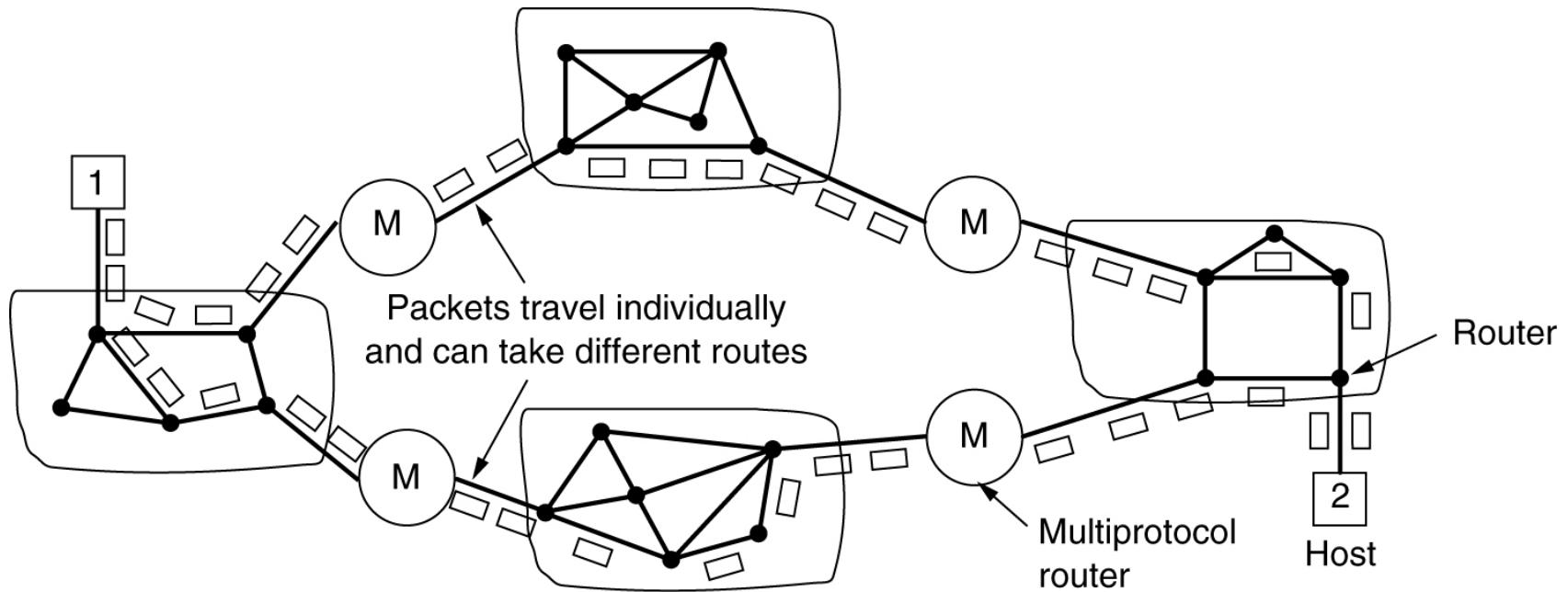
- (a) Two Ethernets connected by a switch.
- (b) Two Ethernets connected by routers.

Concatenated Virtual Circuits



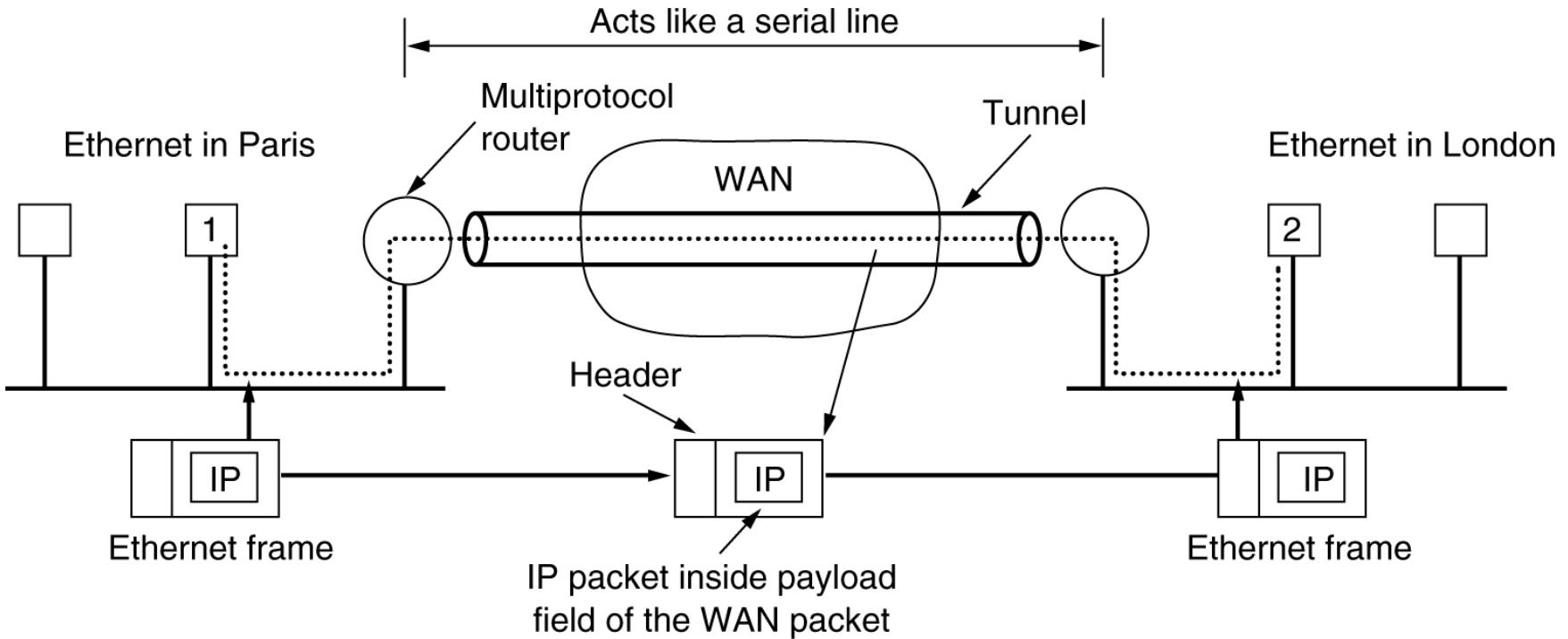
Internetworking using concatenated virtual circuits.

Connectionless Internetworking



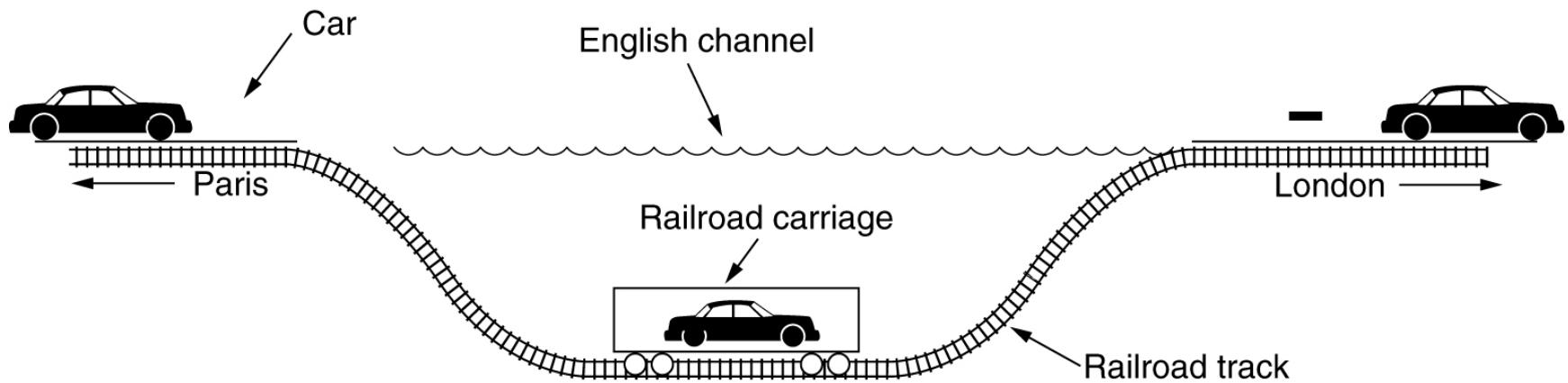
A connectionless internet.

Tunneling



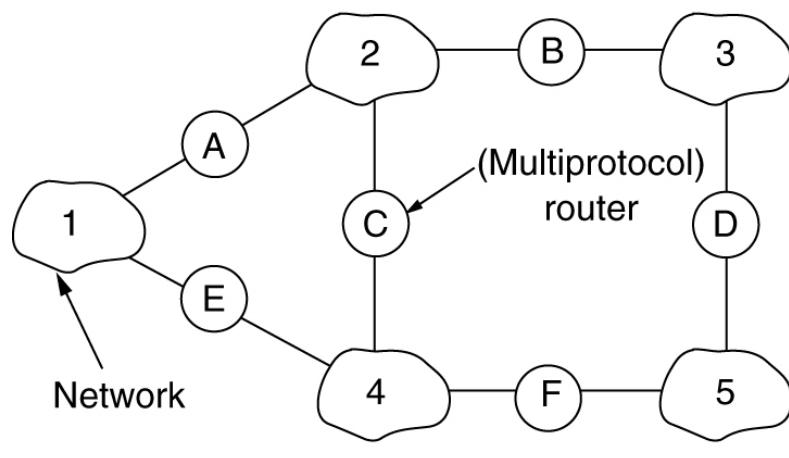
Tunneling a packet from Paris to London.

Tunneling (2)

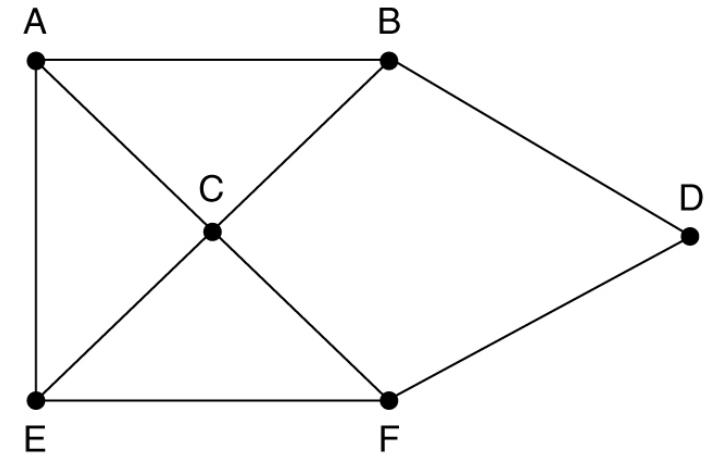


Tunneling a car from France to England.

Internet Routing



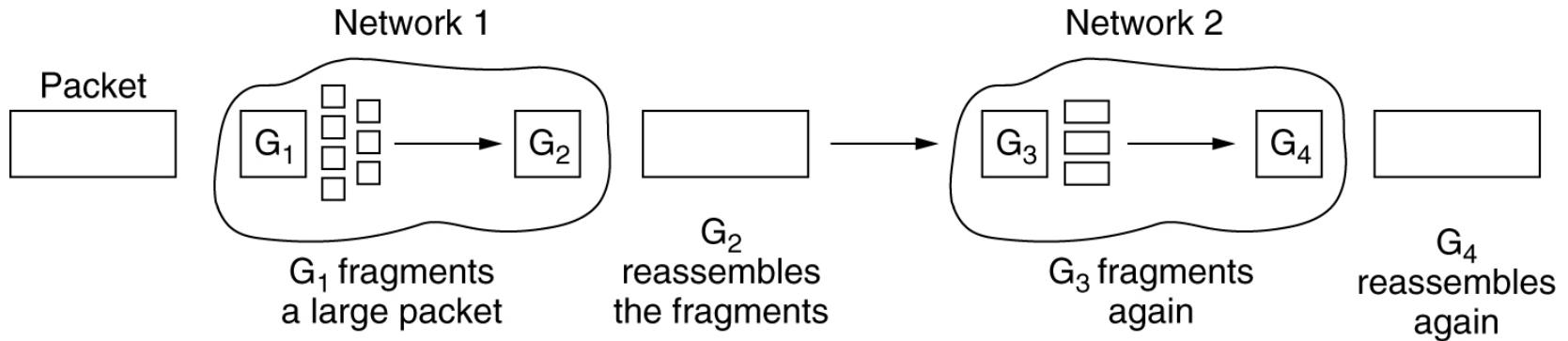
(a)



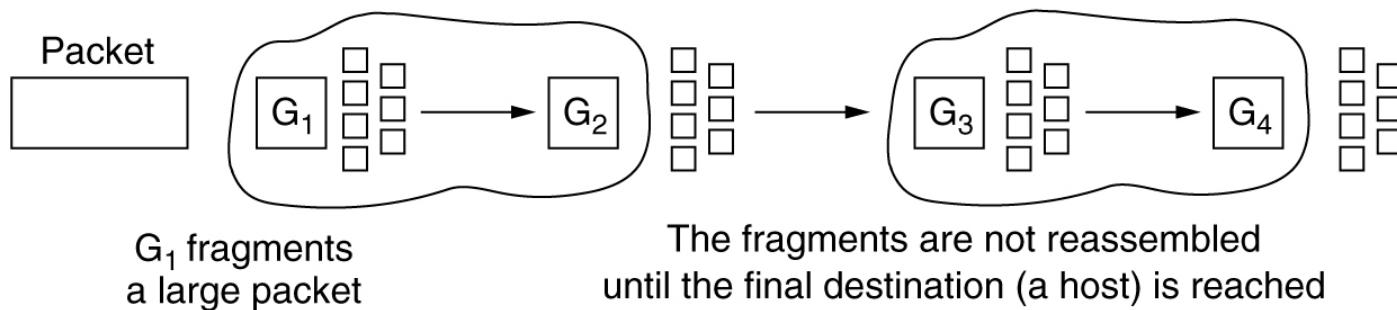
(b)

(a) An internetwork. (b) A graph of the internetwork.

Fragmentation



(a)

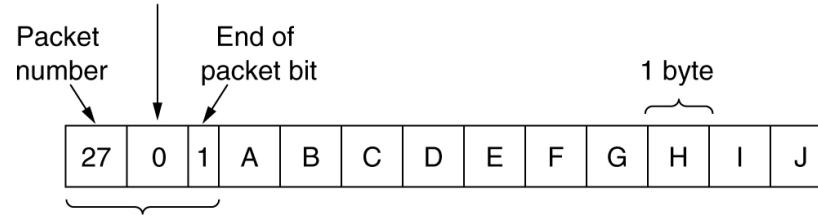


(b)

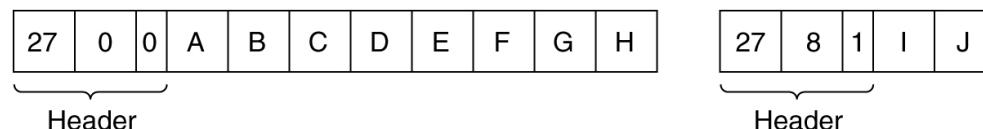
(a) Transparent fragmentation. (b) Nontransparent fragmentation.

Fragmentation (2)

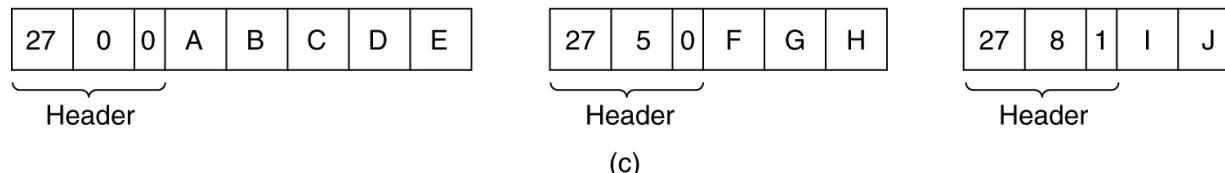
Number of the first elementary fragment in this packet



(a)



(b)



(c)

Fragmentation when the elementary data size is 1 byte.

- (a) Original packet, containing 10 data bytes.
- (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header.
- (c) Fragments after passing through a size 5 gateway.

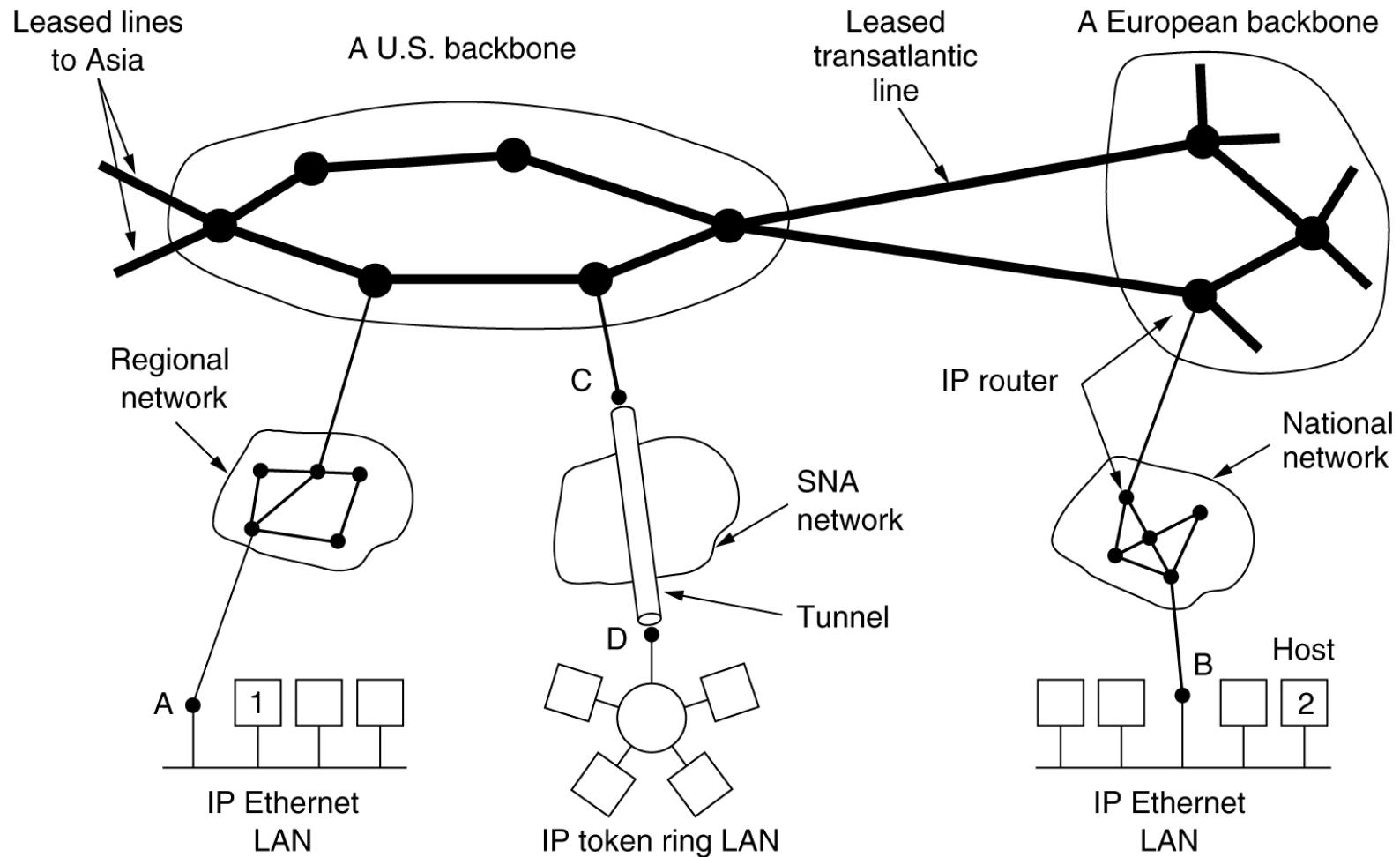
The Network Layer in the Internet

- The IP Protocol
- IP Addresses
- Internet Control Protocols
- OSPF – The Interior Gateway Routing Protocol
- BGP – The Exterior Gateway Routing Protocol
- Internet Multicasting
- Mobile IP
- IPv6

Design Principles for Internet

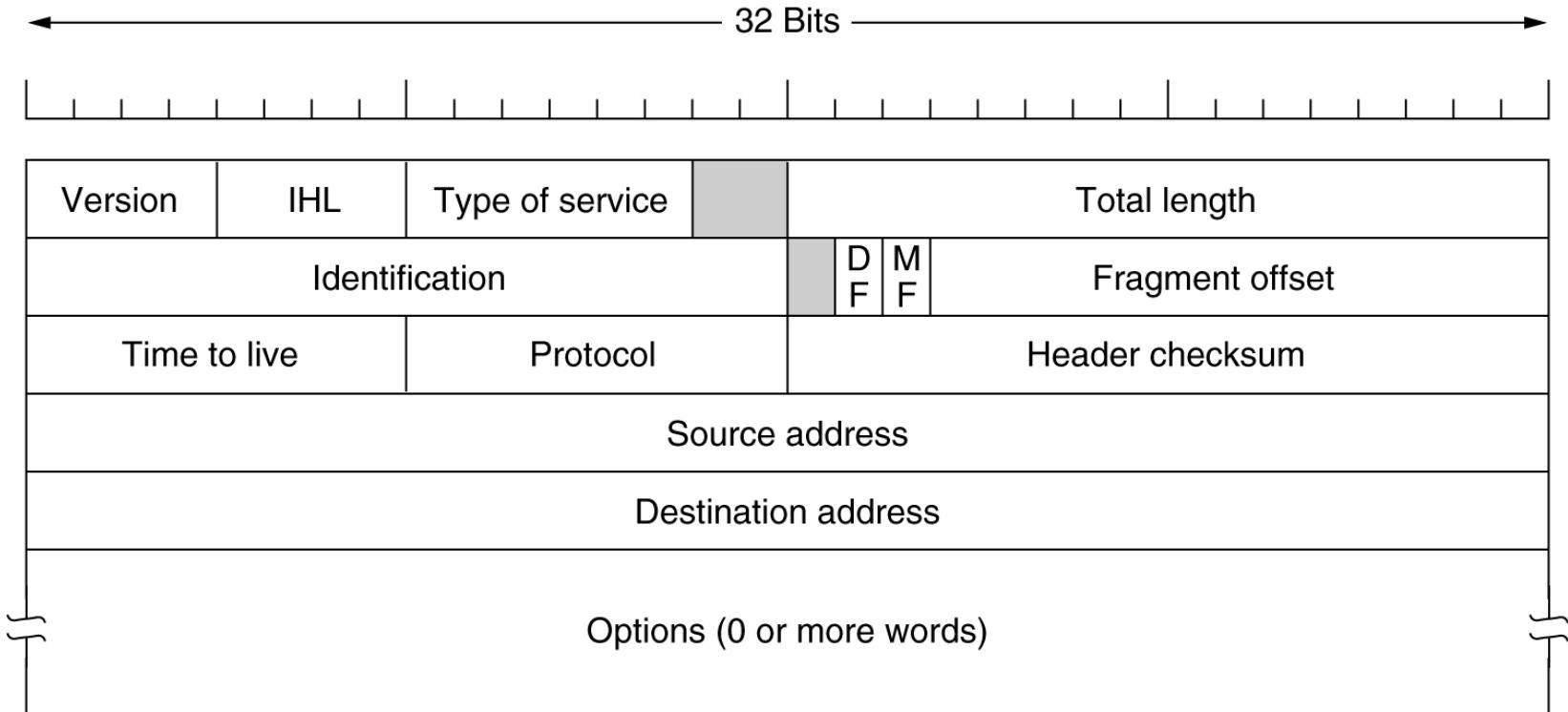
- A. Make sure it works.
- B. Keep it simple.
- C. Make clear choices.
- D. Exploit modularity.
- E. Expect heterogeneity.
- F. Avoid static options and parameters.
- G. Look for a good design; it need not be perfect.
- H. Be strict when sending and tolerant when receiving.
- I. Think about scalability.
- J. Consider performance and cost.

Collection of Subnetworks



The Internet is an interconnected collection of many networks.

The IP Protocol



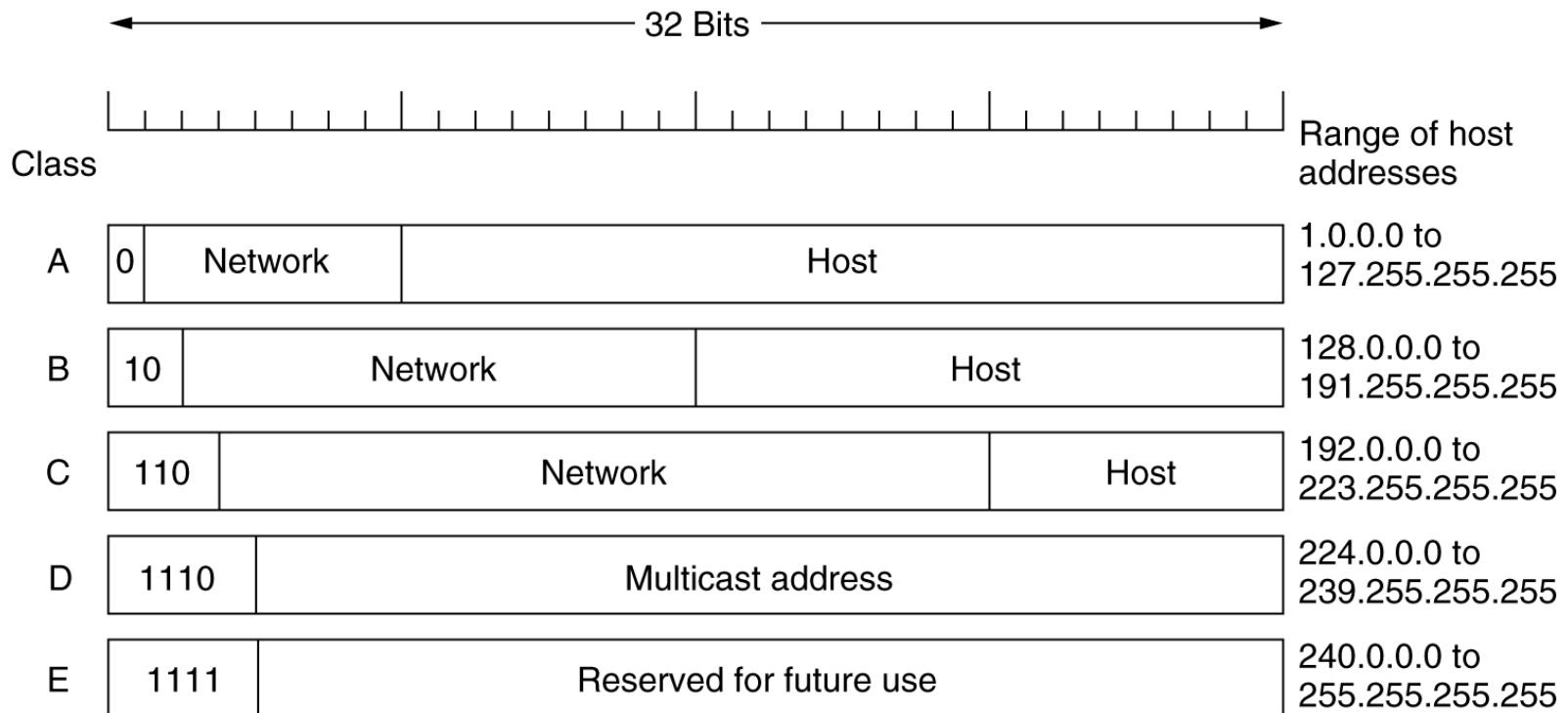
The IPv4 (Internet Protocol) header.

The IP Protocol (2)

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

Some of the IP options.

IP Addresses



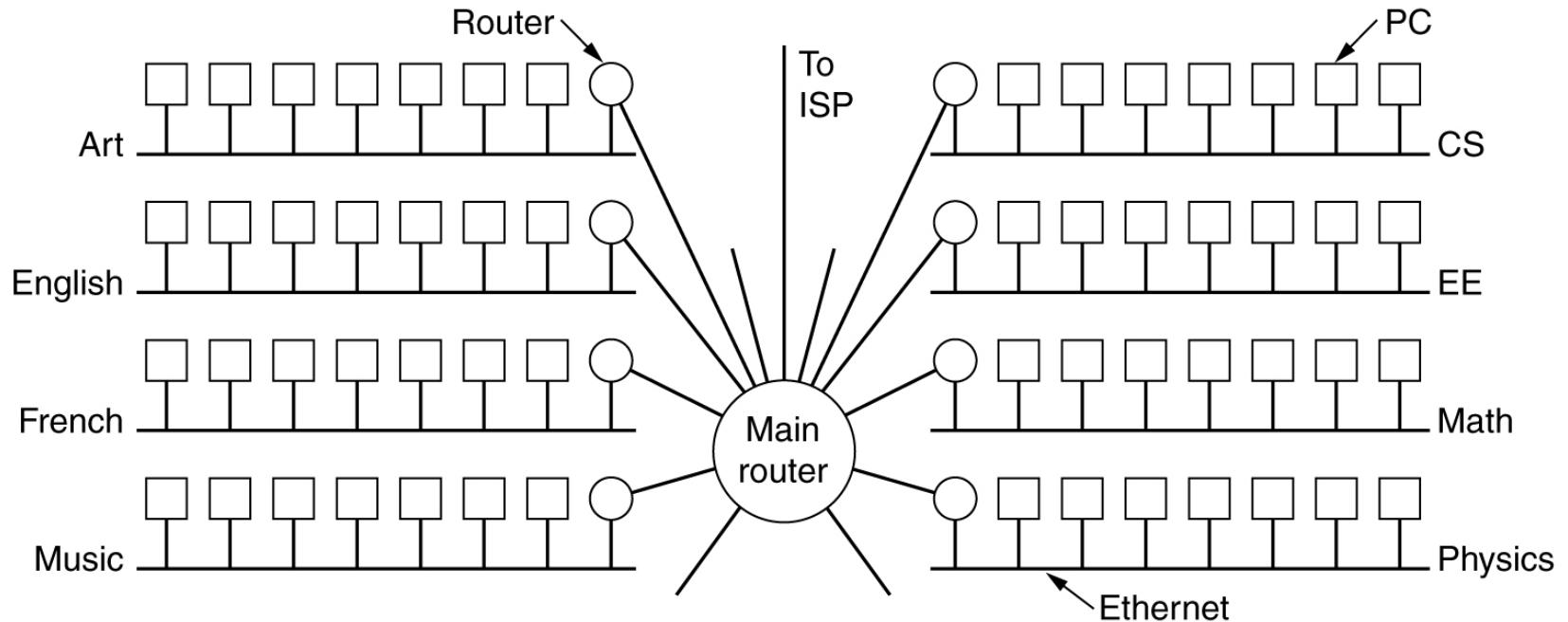
IP address formats.

IP Addresses (2)

0 0	This host
0 0 ... 0 0	Host
1 1	Broadcast on the local network
Network 1 1 1 1 ... 1 1 1 1	Broadcast on a distant network
127	(Anything)

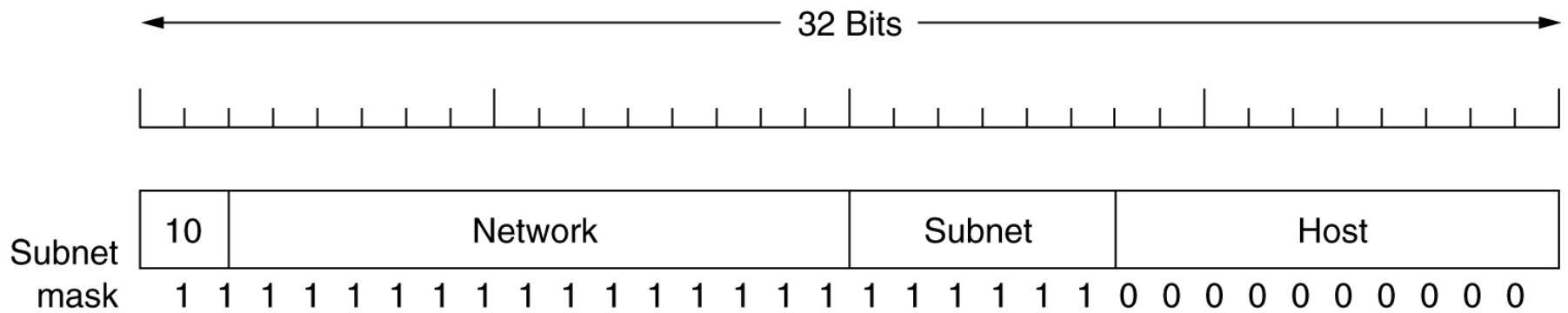
Special IP addresses.

Subnets



A campus network consisting of LANs for various departments.

Subnets (2)



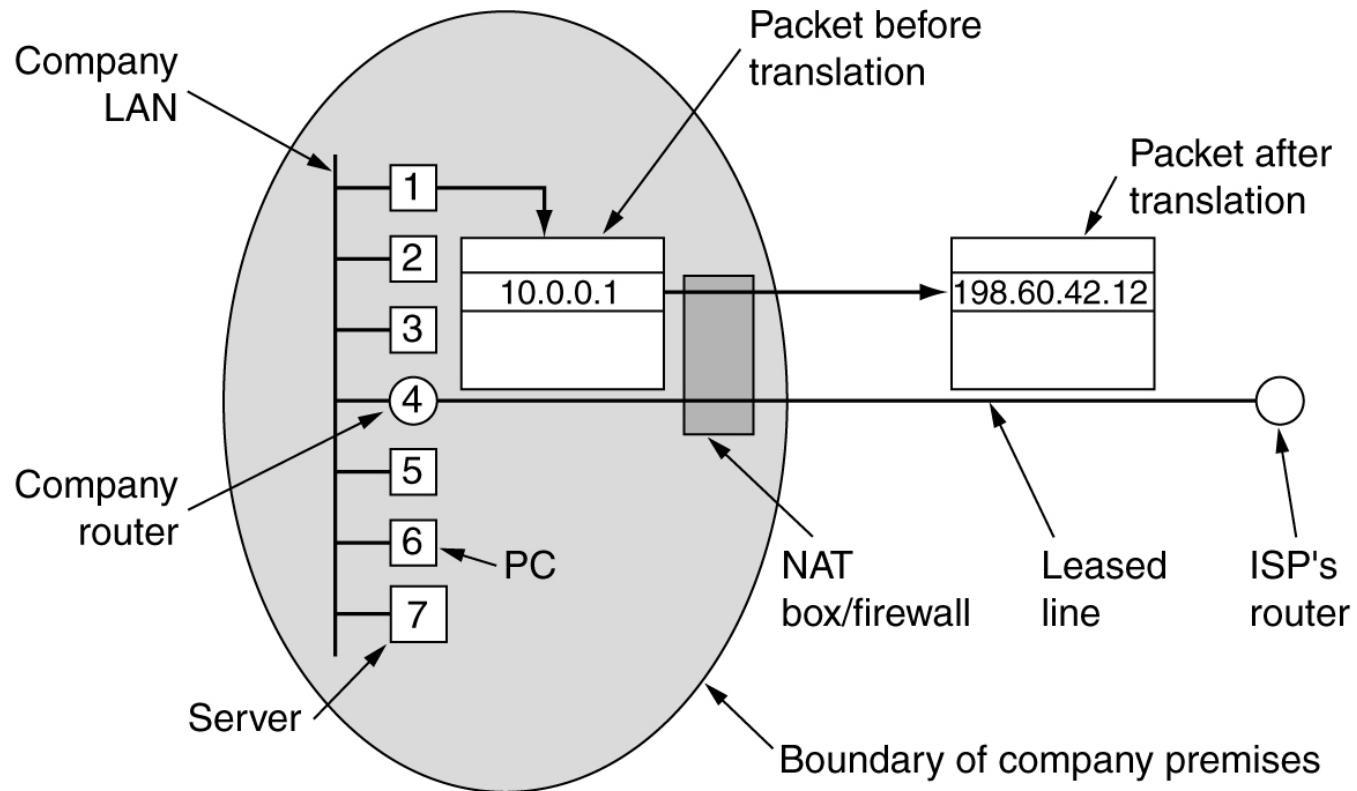
A class B network subnetted into 64 subnets.

CDR – Classless InterDomain Routing

University	First address	Last address	How many	Written as
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

A set of IP address assignments.

NAT – Network Address Translation



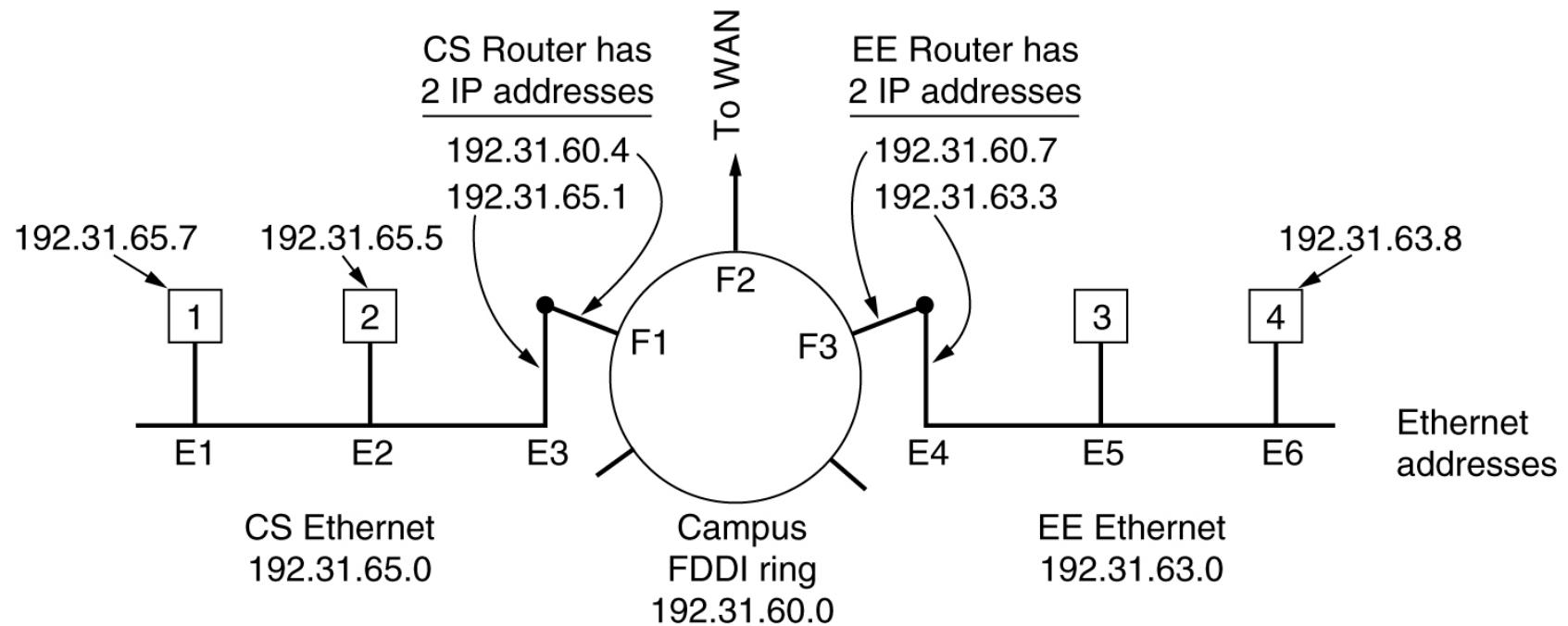
Placement and operation of a NAT box.

Internet Control Message Protocol

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo request	Ask a machine if it is alive
Echo reply	Yes, I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

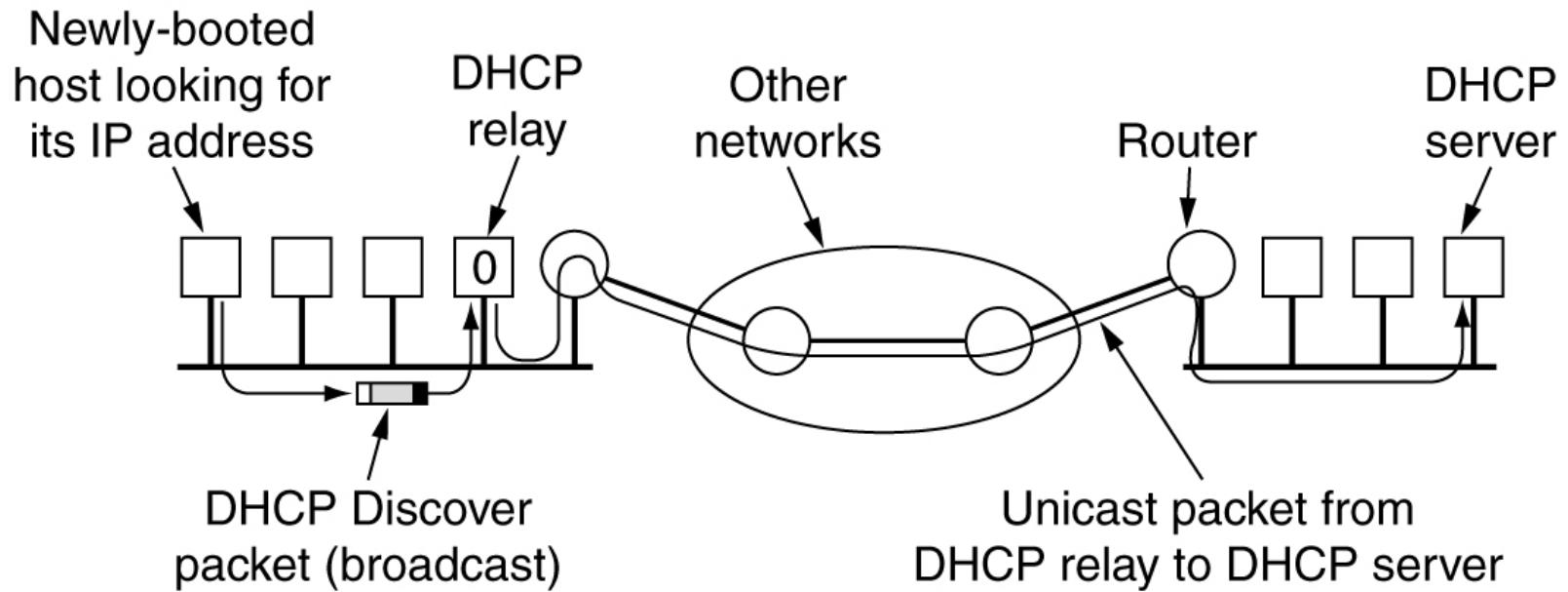
The principal ICMP message types.

ARP – The Address Resolution Protocol



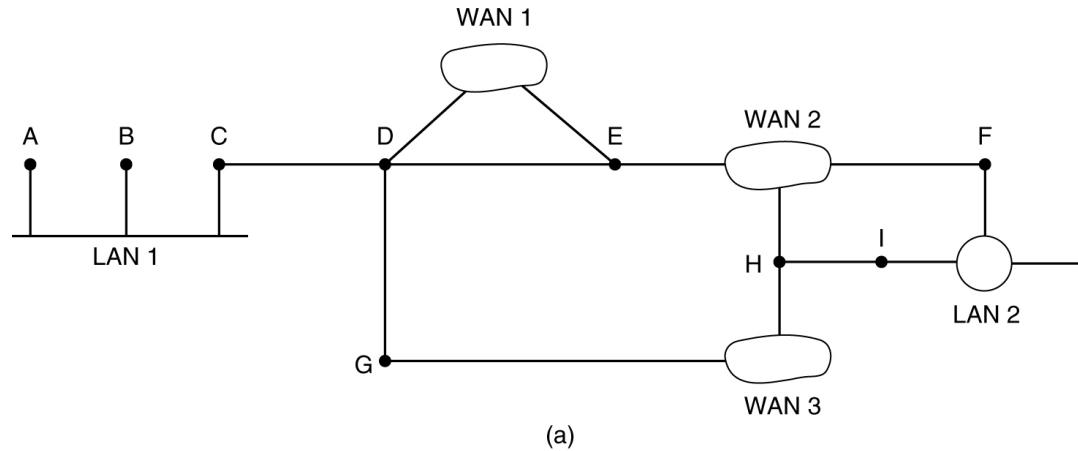
Three interconnected /24 networks: two Ethernets and an FDDI ring.

Dynamic Host Configuration Protocol

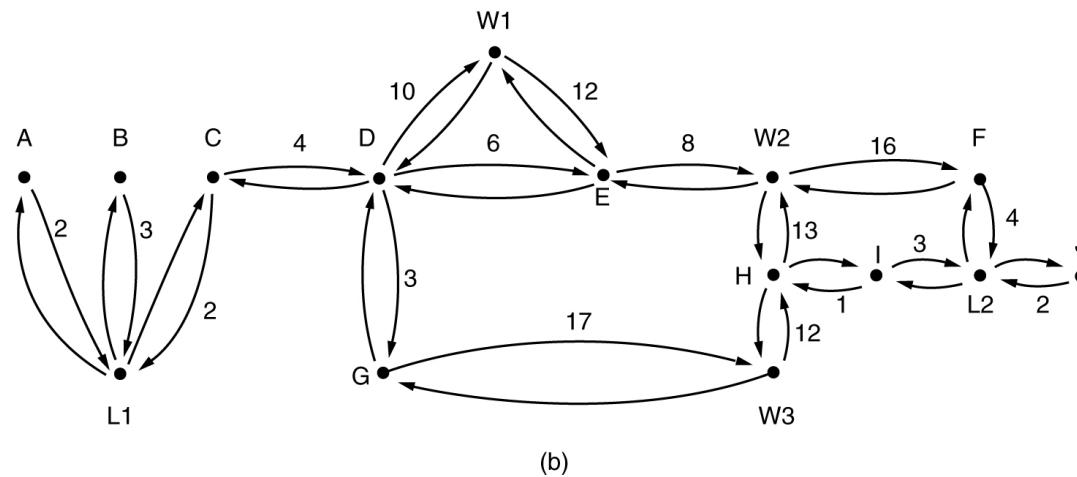


Operation of DHCP.

OSPF – The Interior Gateway Routing Protocol



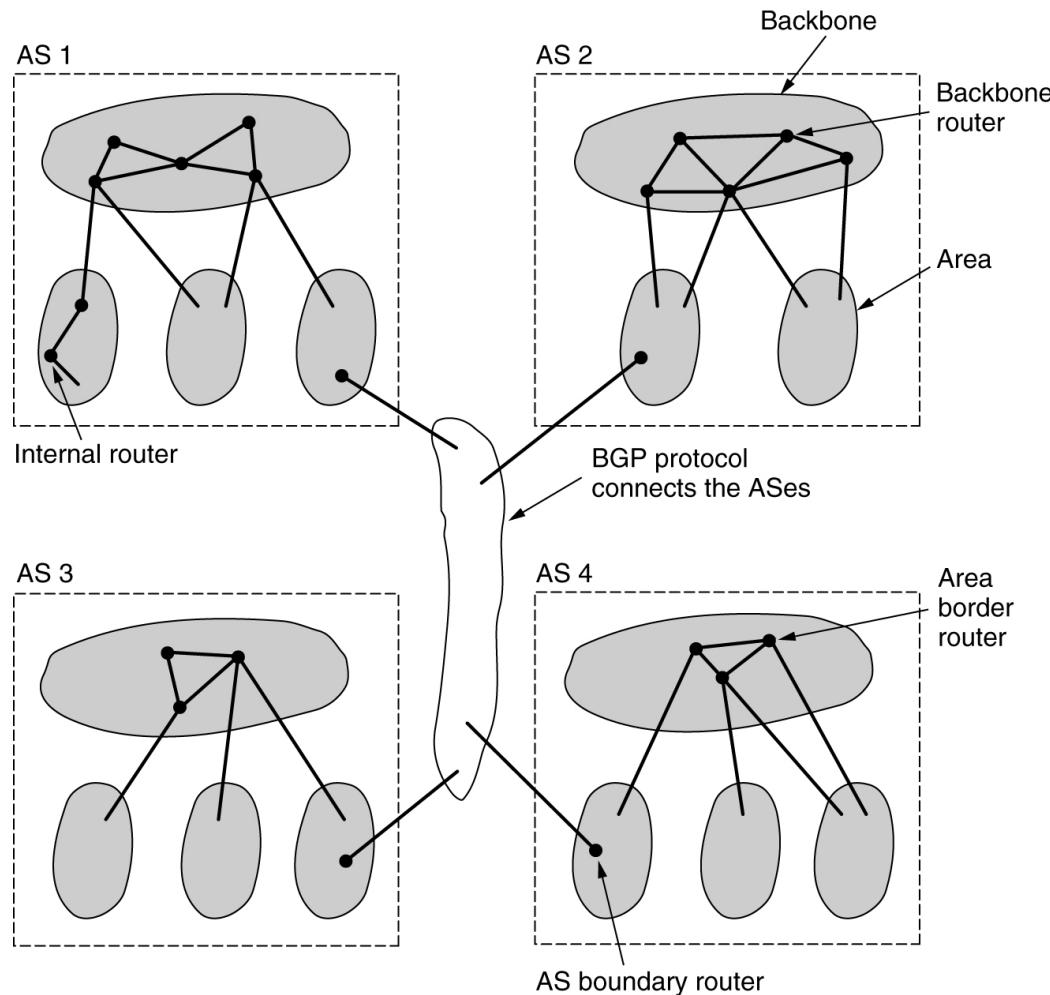
(a)



(b)

(a) An autonomous system. (b) A graph representation of (a).

OSPF (2)



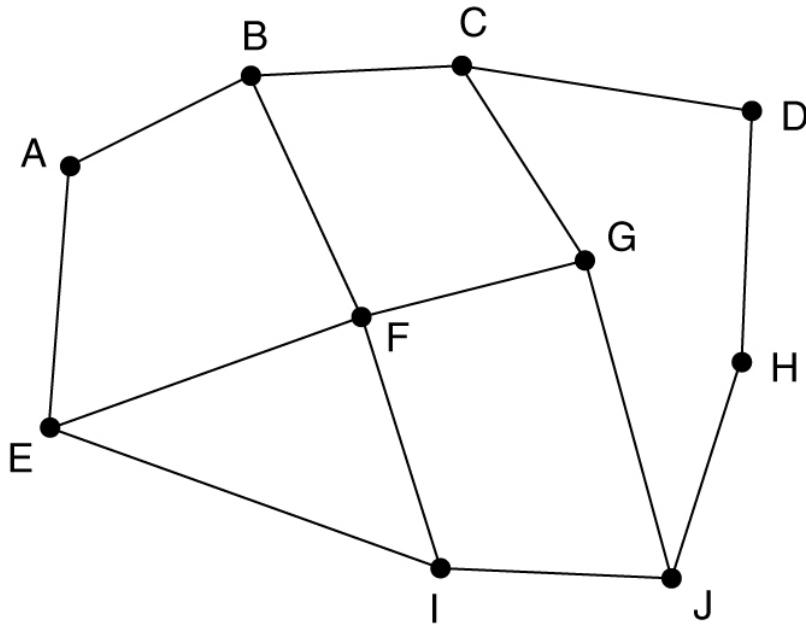
The relation between ASes, backbones, and areas in OSPF.

OSPF (3)

Message type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

The five types of OSPF messages.

BGP – The Exterior Gateway Routing Protocol



Information F receives from its neighbors about D

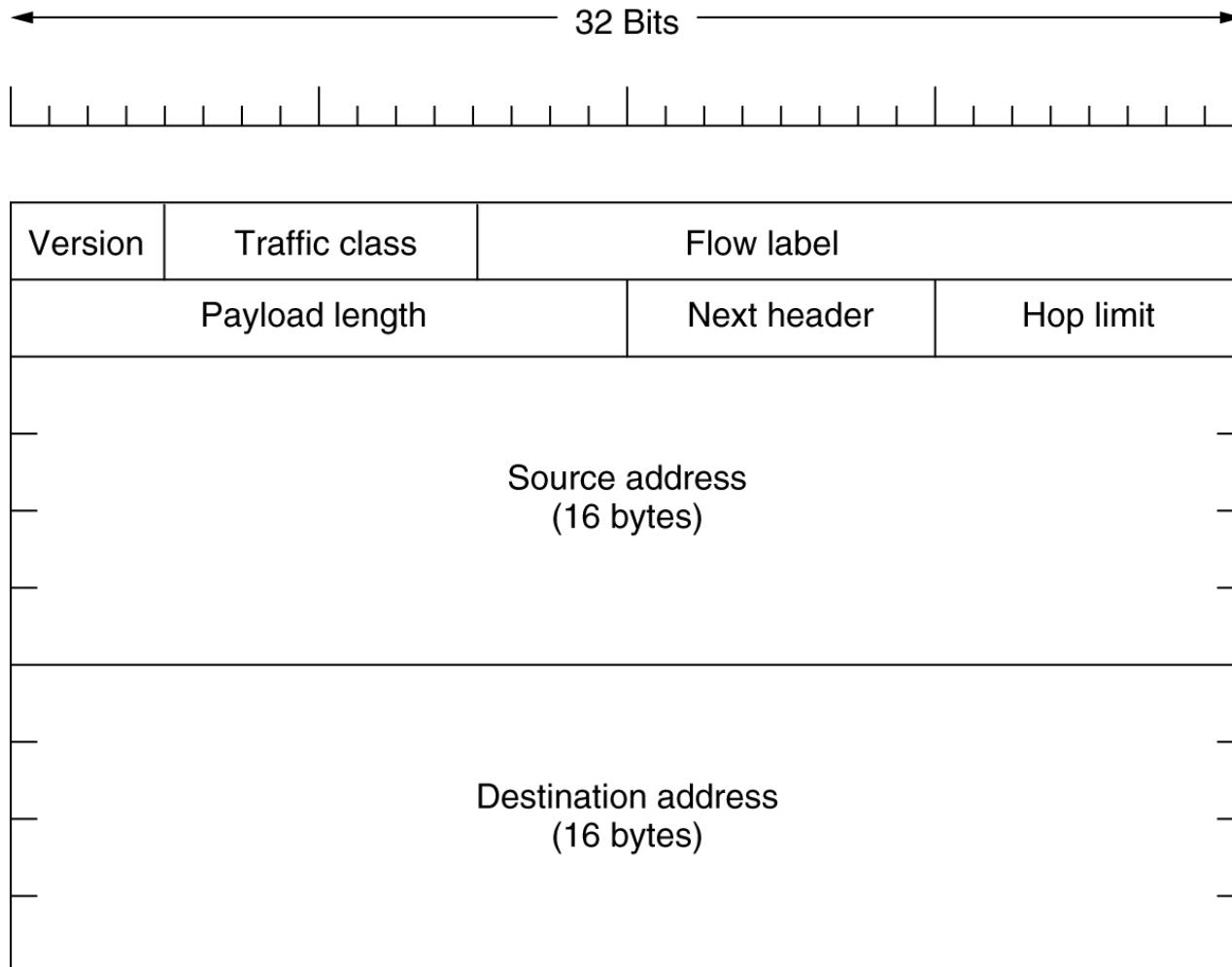
From B: "I use BCD"
From G: "I use GCD"
From I: "I use IFGCD"
From E: "I use EFGCD"

(b)

(a) A set of BGP routers.

(b) Information sent to F.

The Main IPv6 Header



The IPv6 fixed header (required).

Extension Headers

Extension header	Description
Hop-by-hop options	Miscellaneous information for routers
Destination options	Additional information for the destination
Routing	Loose list of routers to visit
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents

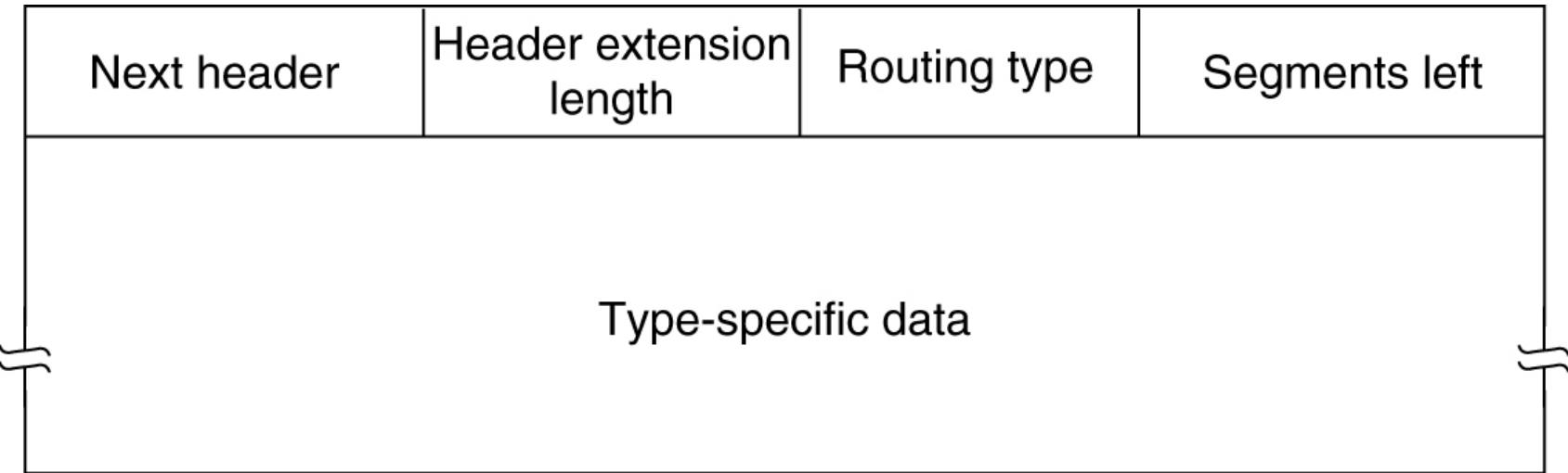
IPv6 extension headers.

Extension Headers (2)

Next header	0	194	4
Jumbo payload length			

The hop-by-hop extension header for large datagrams (jumbograms).

Extension Headers (3)



The extension header for routing.



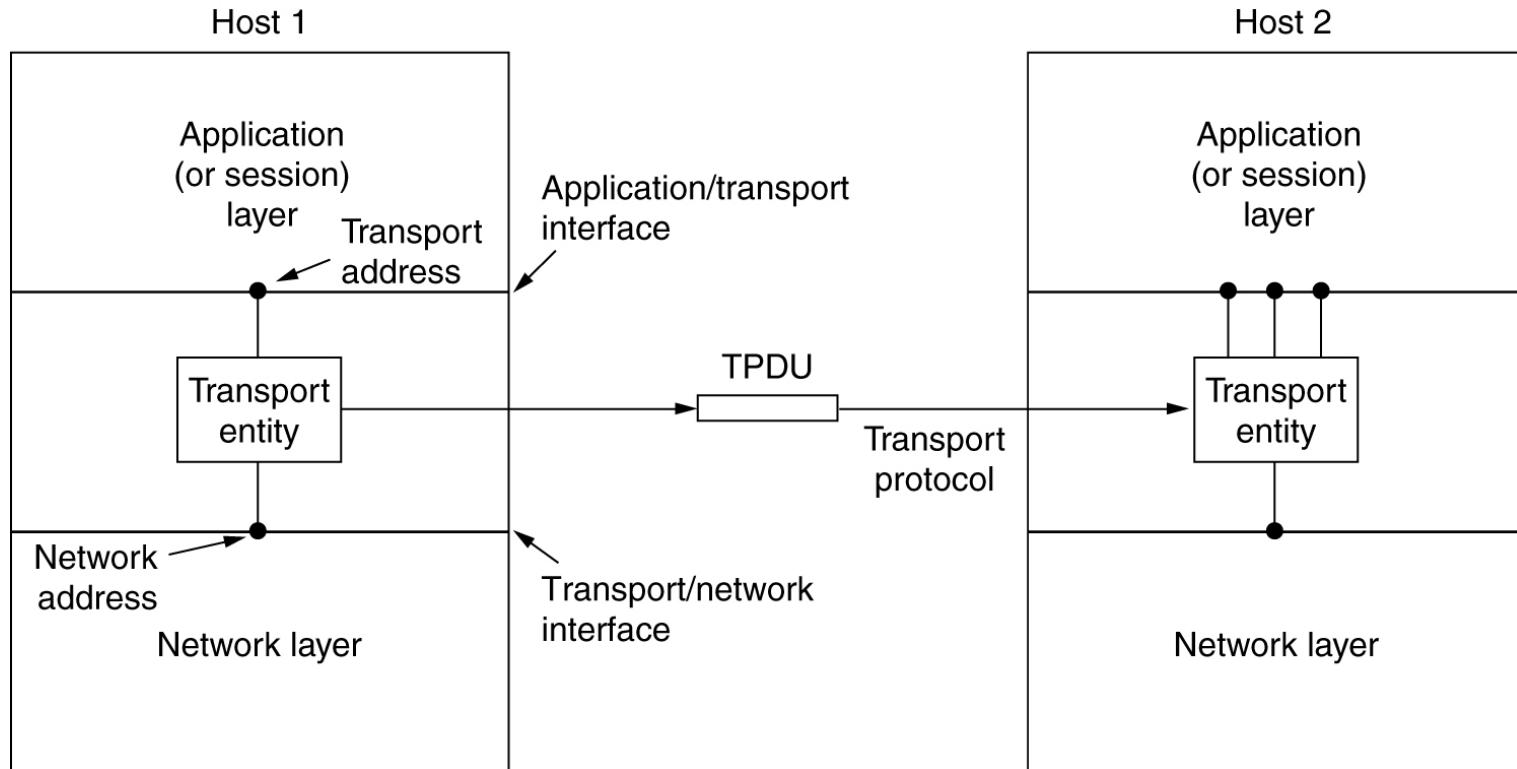
Chapter 6

The Transport Layer

The Transport Service

- Services Provided to the Upper Layers
- Transport Service Primitives
- Berkeley Sockets
- An Example of Socket Programming:
 - An Internet File Server

Services Provided to the Upper Layers



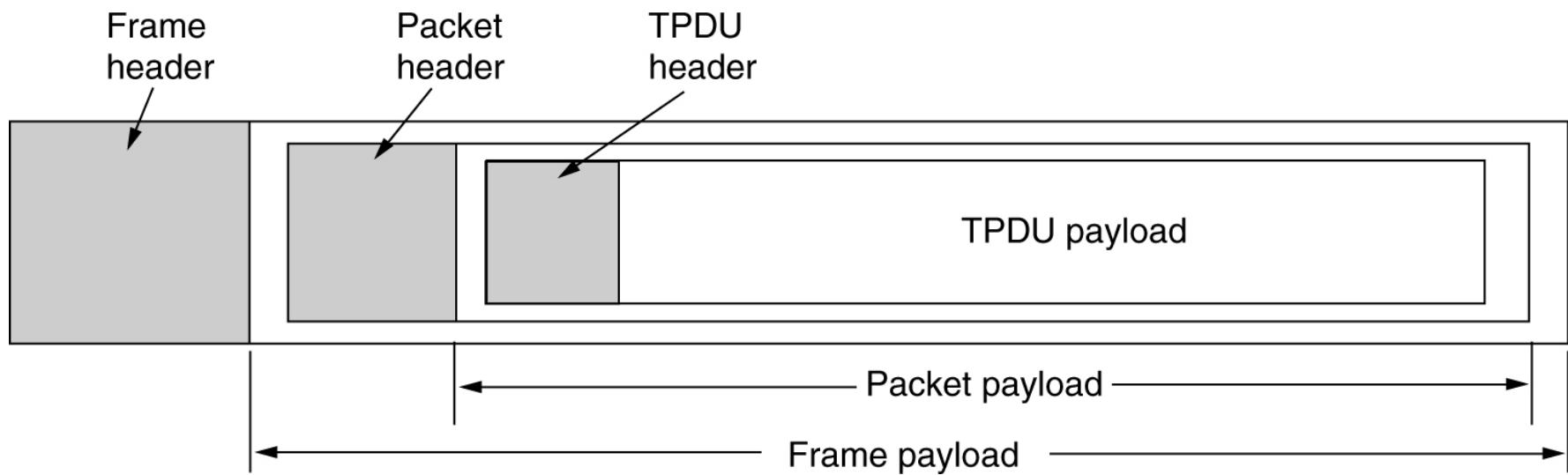
The network, transport, and application layers.

Transport Service Primitives

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

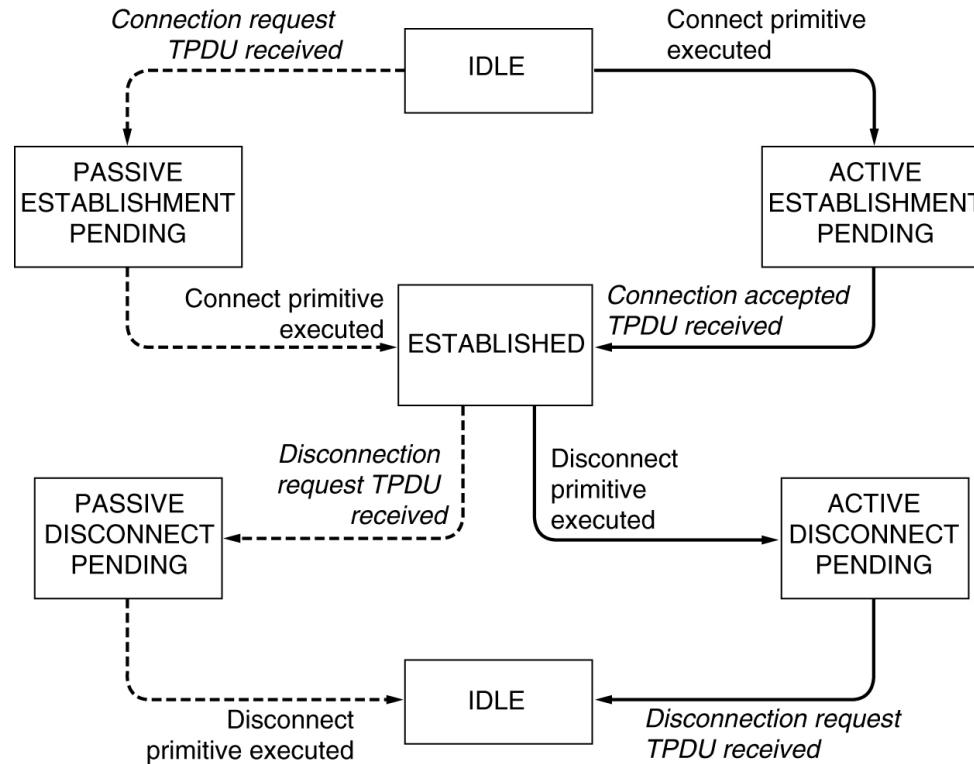
The primitives for a simple transport service.

Transport Service Primitives (2)



The nesting of TPDUs, packets, and frames.

Transport Service Primitives (3)



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Berkeley Sockets

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP.

Socket Programming Example: Internet File Server

Client code using sockets.

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345          /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096              /* block transfer size */

int main(int argc, char **argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE];           /* buffer for incoming file */
    struct hostent *h;             /* info about server */
    struct sockaddr_in channel;    /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file-name");
    h = gethostbyname(argv[1]);     /* look up host's IP address */
    if (!h) fatal("gethostbyname failed");

    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s < 0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family= AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port= htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2])+1);

    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF_SIZE);          /* read from socket */
        if (bytes <= 0) exit(0);                  /* check for end of file */
        write(1, buf, bytes);                    /* write to standard output */
    }
}

fatal(char *string)
{
    printf("%s\n", string);
    exit(1);
}
```

Socket Programming Example: Internet File Server (2)

Client code using sockets.

```
#include <sys/types.h>           /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345          /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096             /* block transfer size */

#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF_SIZE];           /* buffer for outgoing file */
    struct sockaddr_in channel;   /* hold's IP address */

    /* Build address structure to bind to socket. */
    memset(&channel, 0, sizeof(channel));      /* zero channel */
    channel.sin_family = AF_INET;
    channel.sin_addr.s_addr = htonl(INADDR_ANY);
    channel.sin_port = htons(SERVER_PORT);

    /* Passive open. Wait for connection. */
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* create socket */
    if (s < 0) fatal("socket failed");
    setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE_SIZE);           /* specify queue size */
    if (l < 0) fatal("listen failed");

    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
        sa = accept(s, 0, 0);           /* block for connection request */
        if (sa < 0) fatal("accept failed");
        read(sa, buf, BUF_SIZE);       /* read file name from socket */

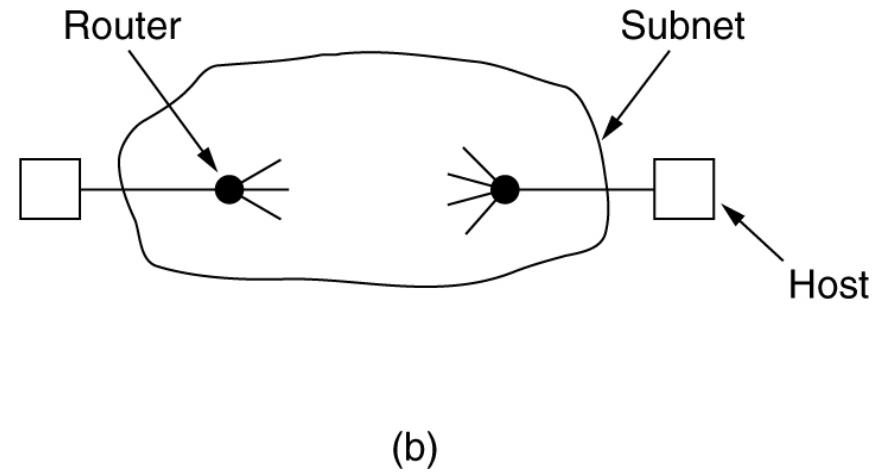
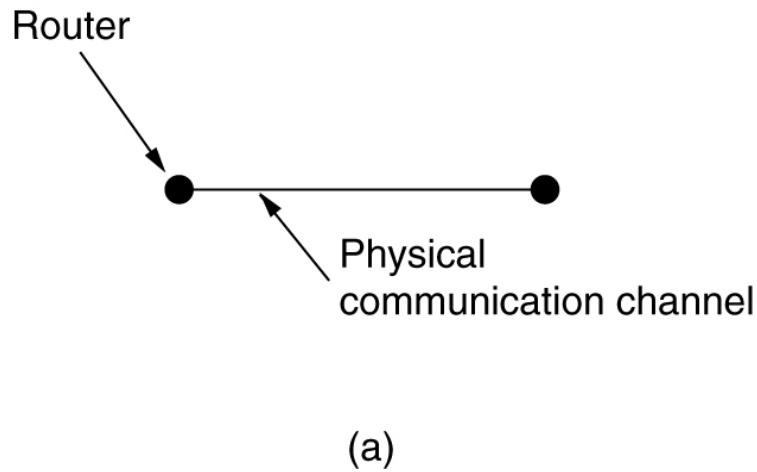
        /* Get and return the file. */
        fd = open(buf, O_RDONLY);      /* open the file to be sent back */
        if (fd < 0) fatal("open failed");

        while (1) {
            bytes = read(fd, buf, BUF_SIZE); /* read from file */
            if (bytes <= 0) break;         /* check for end of file */
            write(sa, buf, bytes);        /* write bytes to socket */
        }
        close(fd);                     /* close file */
        close(sa);                     /* close connection */
    }
}
```

Elements of Transport Protocols

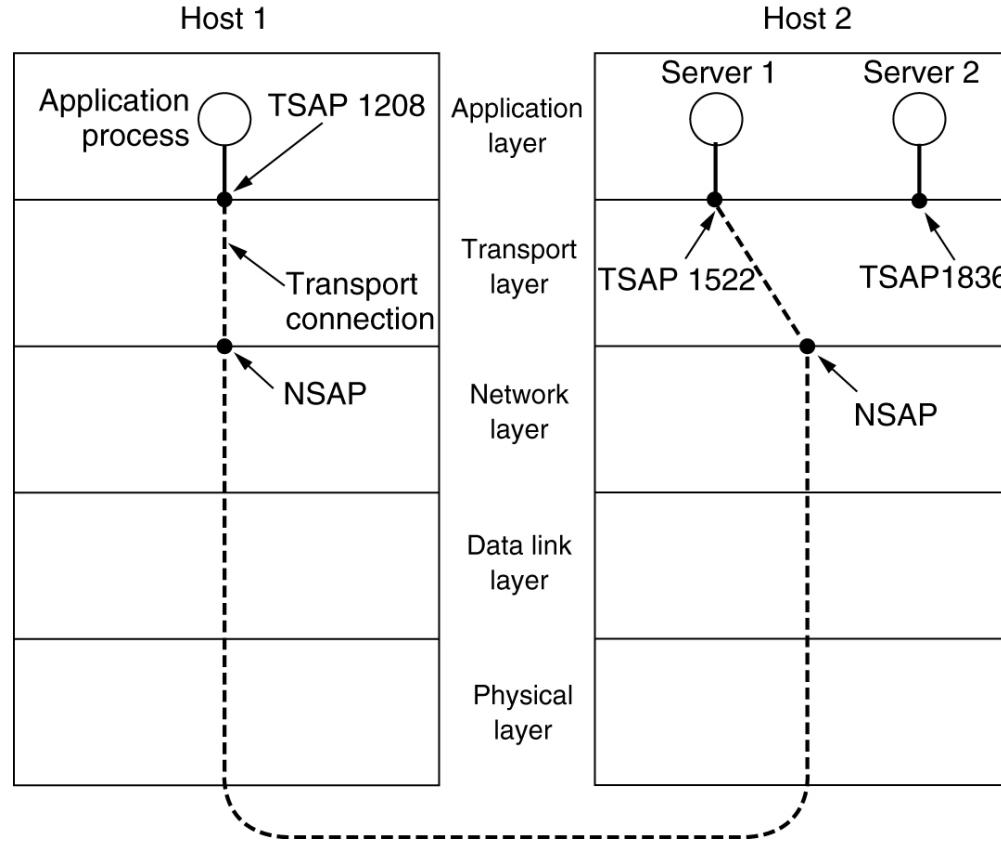
- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

Transport Protocol



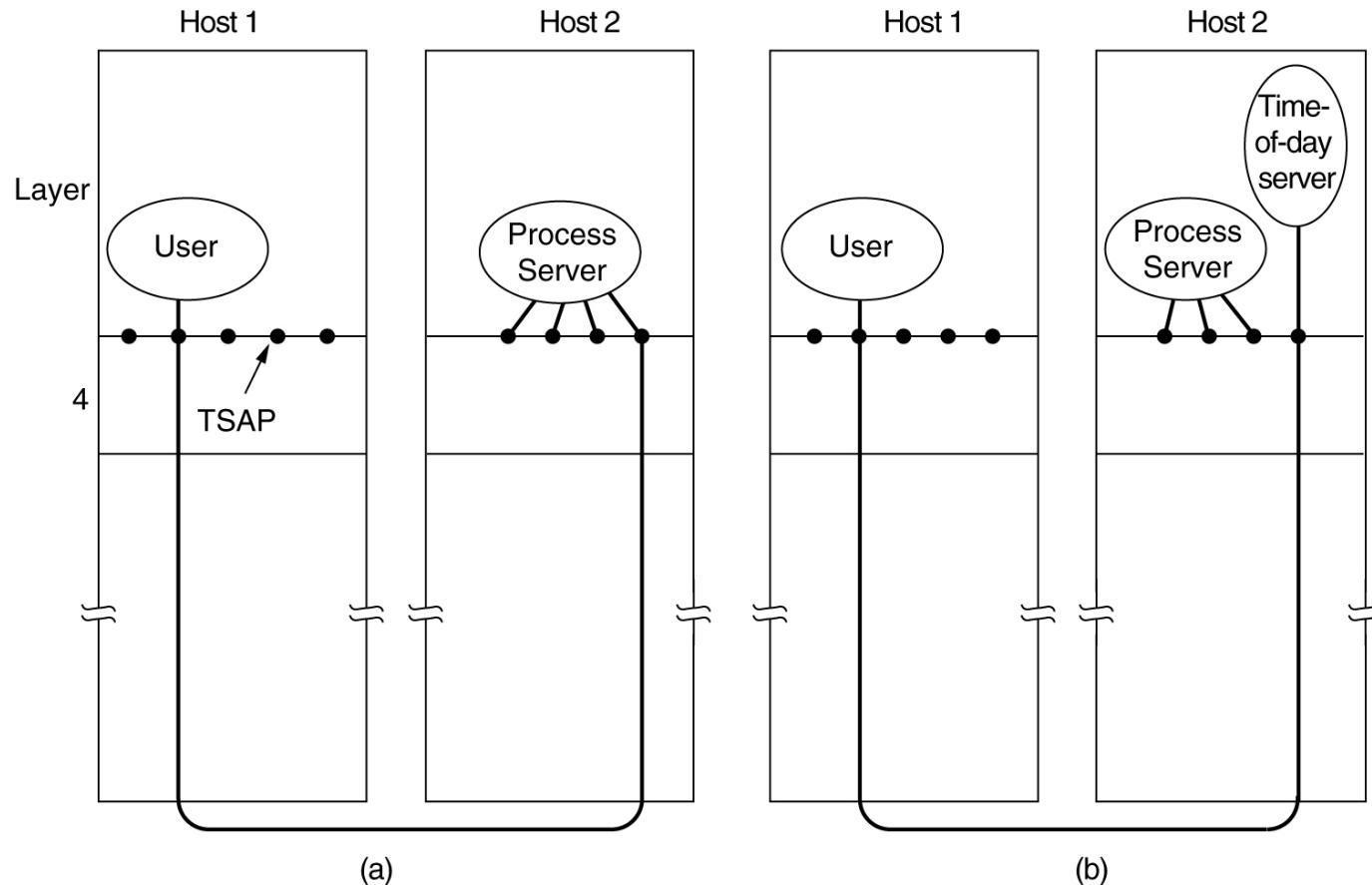
- (a) Environment of the data link layer.
- (b) Environment of the transport layer.

Addressing



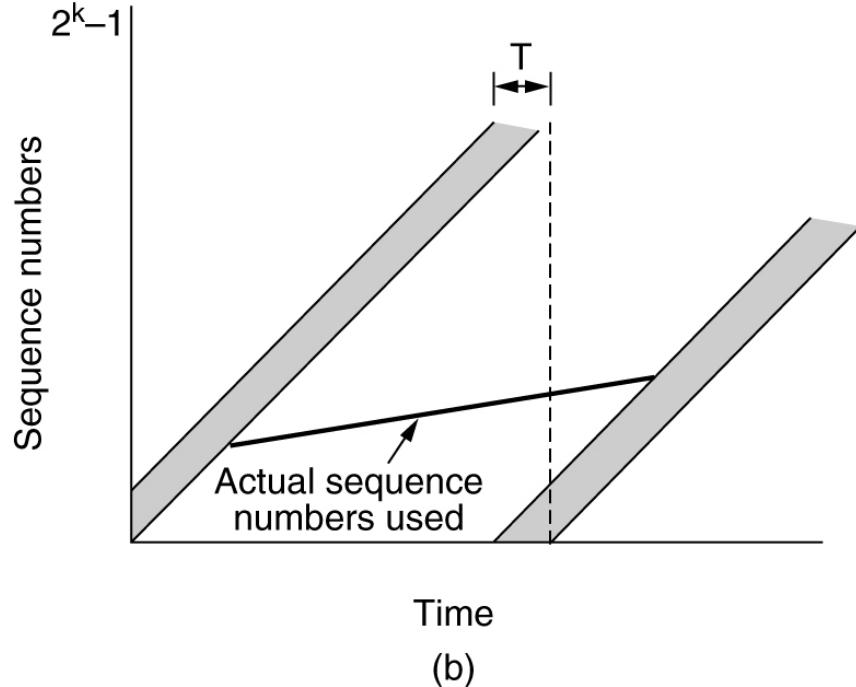
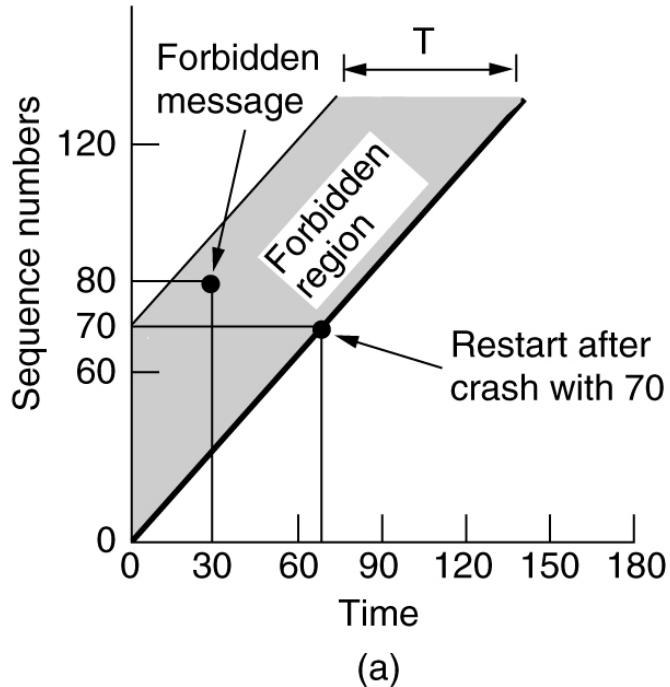
TSAPs, NSAPs and transport connections.

Connection Establishment



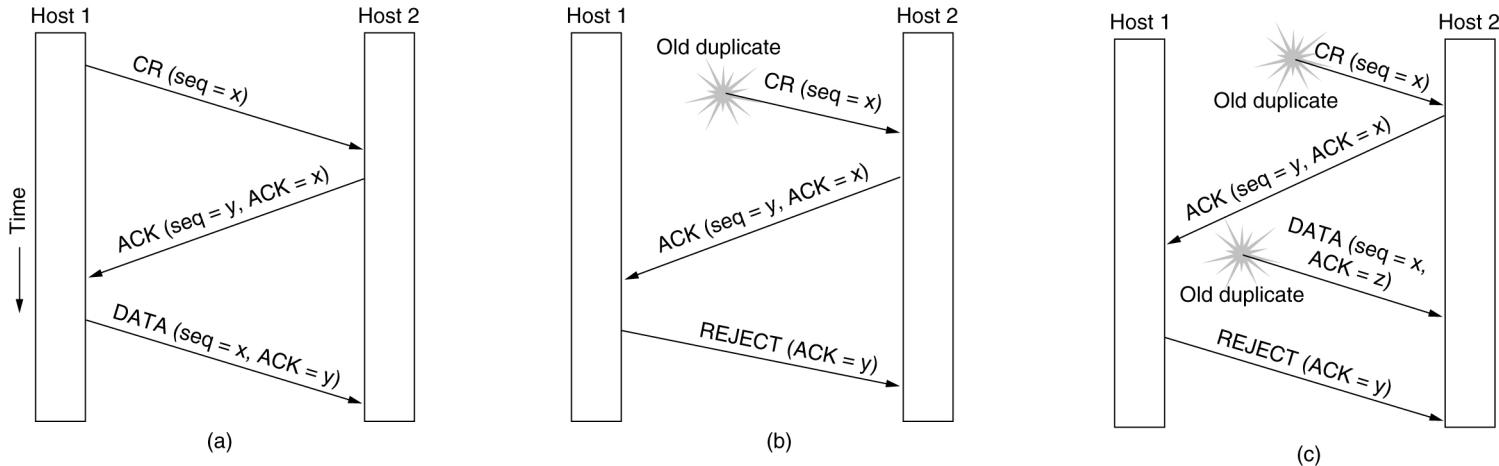
How a user process in host 1 establishes a connection with a time-of-day server in host 2.

Connection Establishment (2)



- (a) TPDUs may not enter the forbidden region.
- (b) The resynchronization problem.

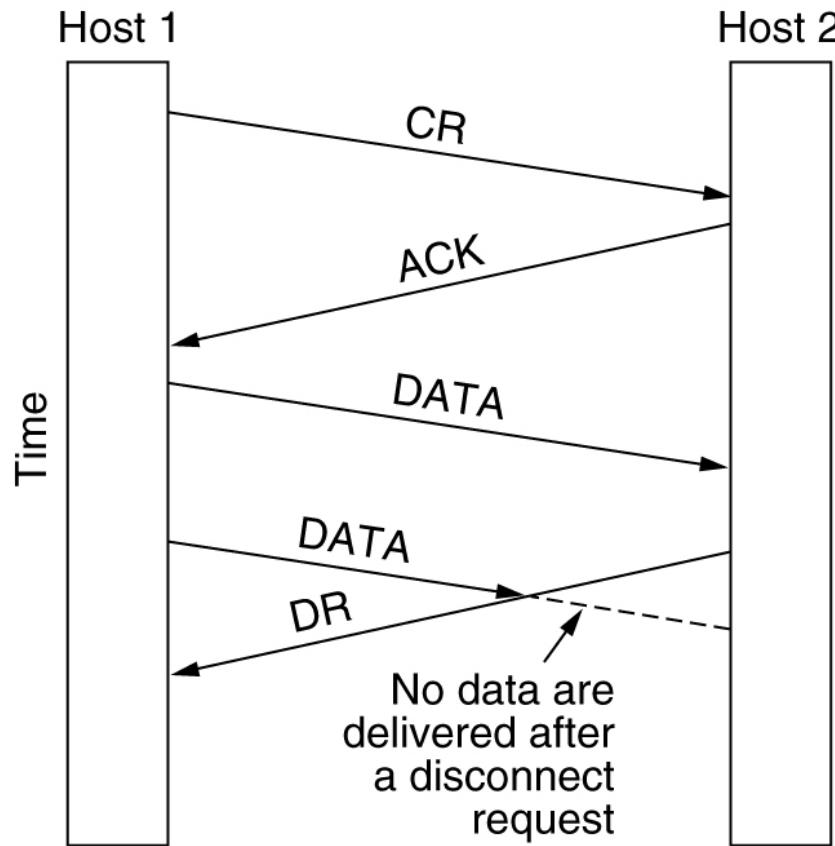
Connection Establishment (3)



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.

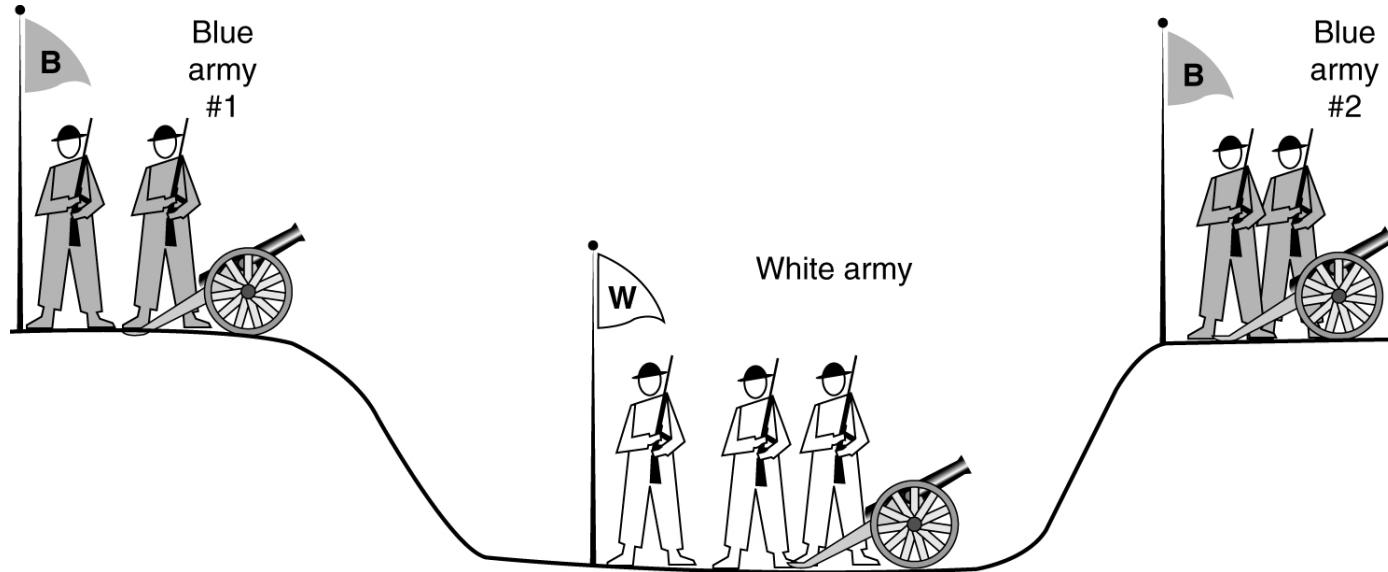
- (a) Normal operation,
- (b) Old CONNECTION REQUEST appearing out of nowhere.
- (c) Duplicate CONNECTION REQUEST and duplicate ACK.

Connection Release



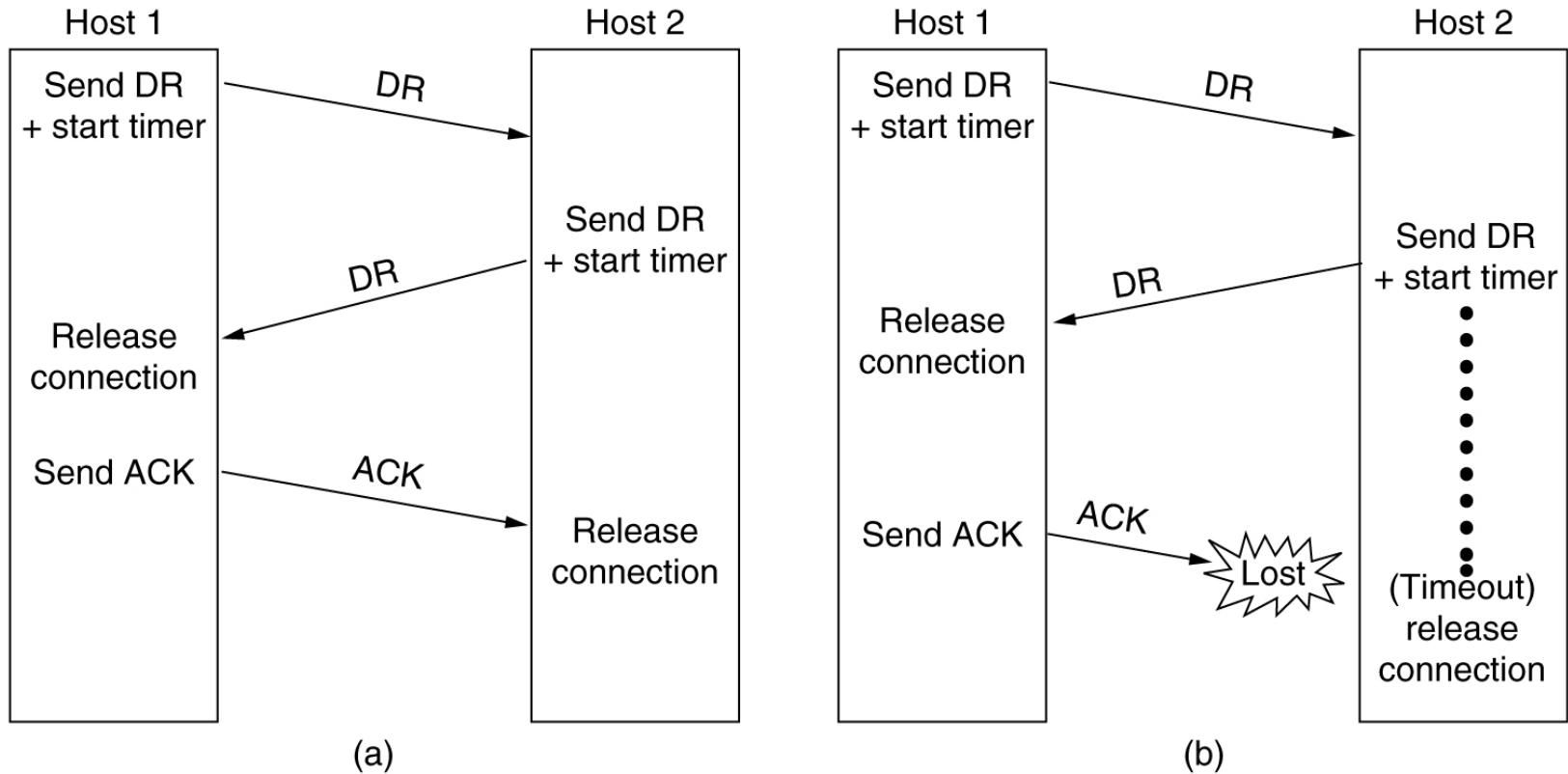
Abrupt disconnection with loss of data.

Connection Release (2)



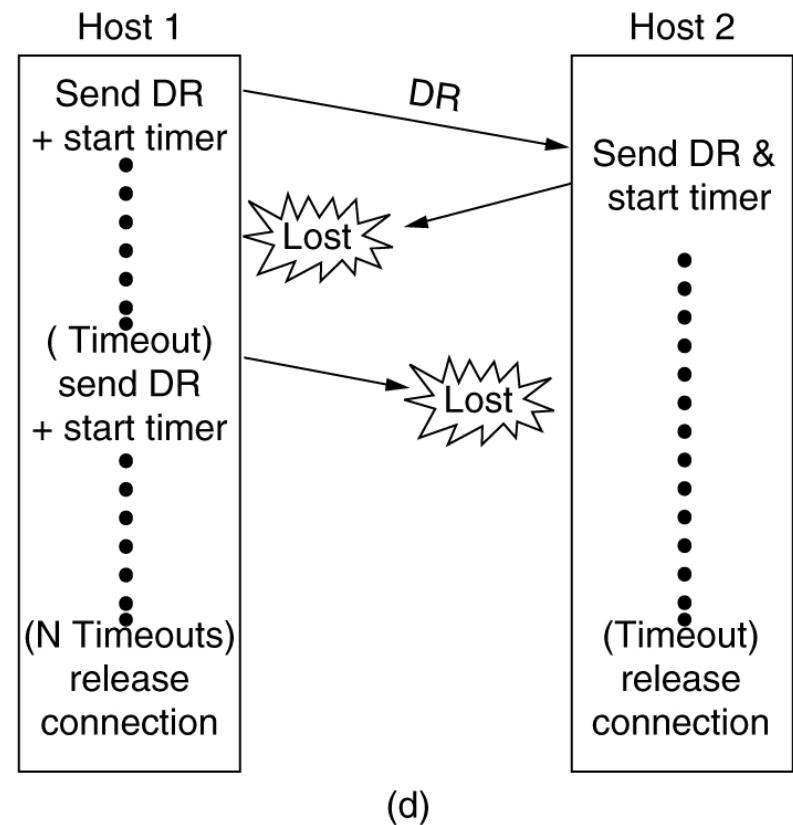
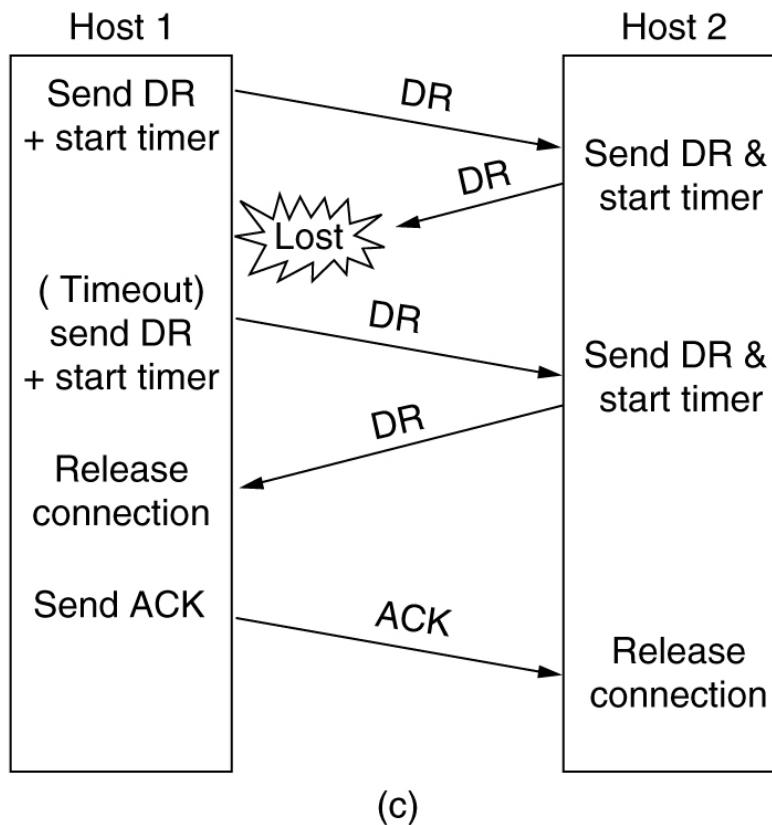
The two-army problem.

Connection Release (3)



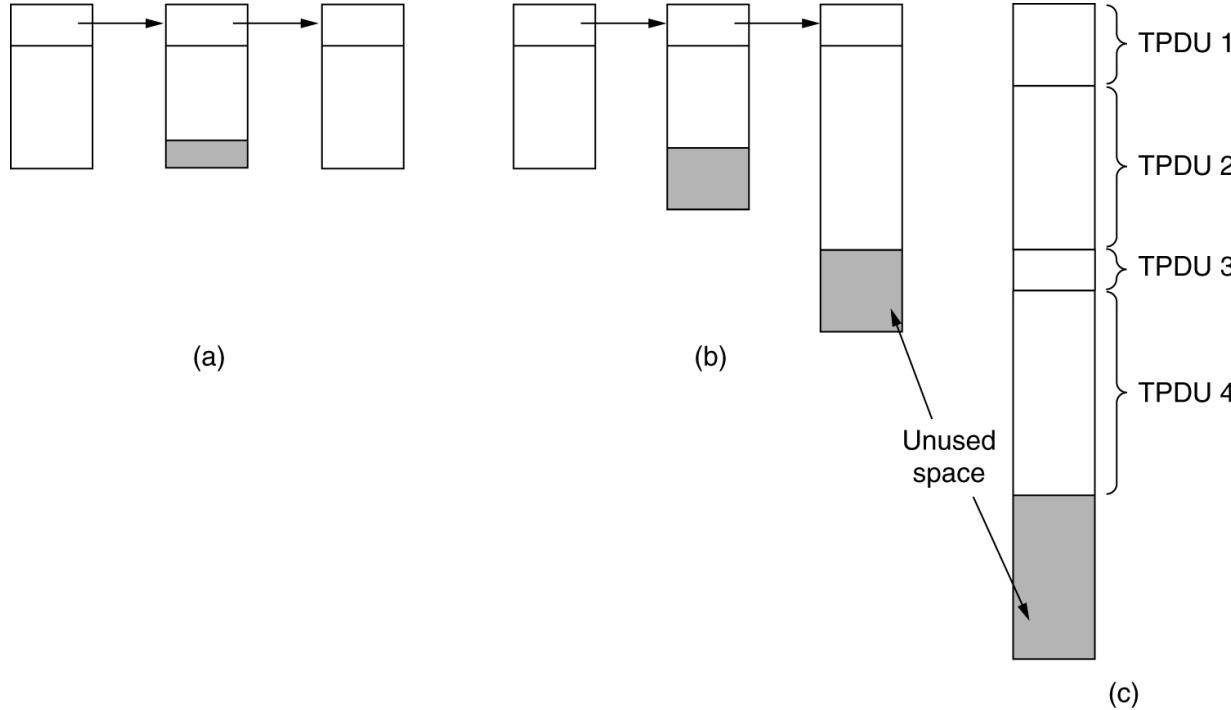
Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) final ACK lost.

Connection Release (4)



(c) Response lost. **(d)** Response lost and subsequent DRs lost.

Flow Control and Buffering



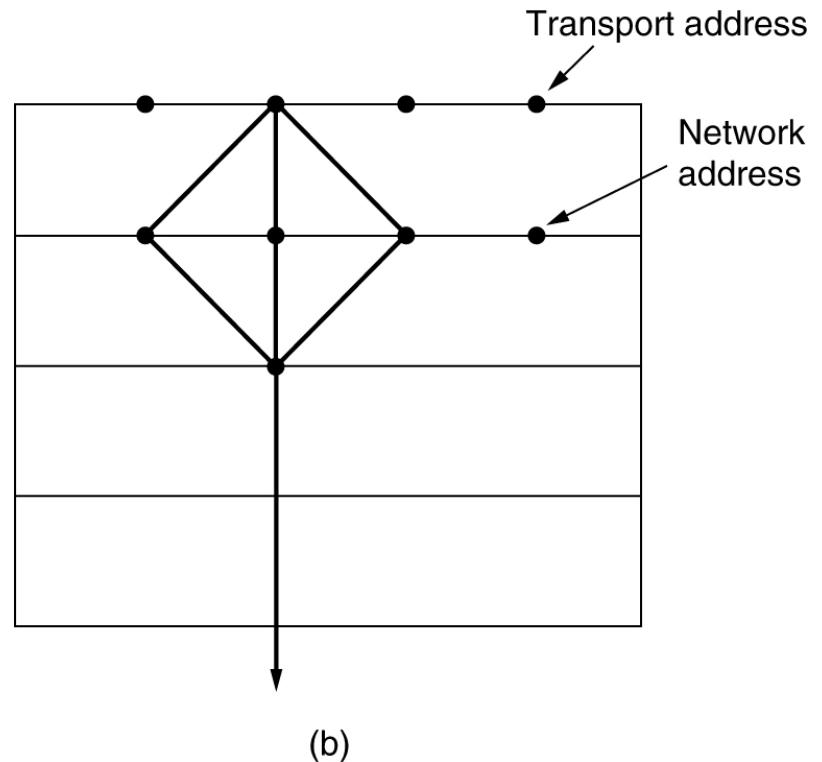
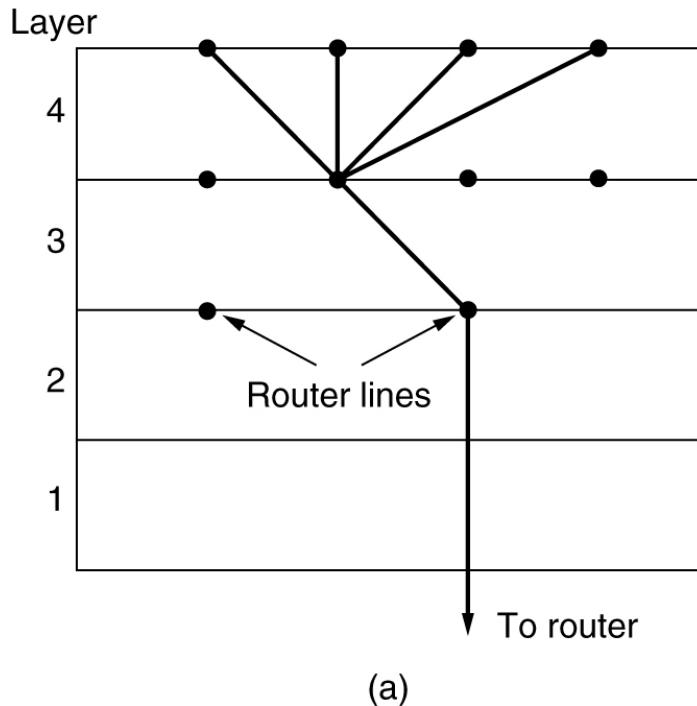
- (a) Chained fixed-size buffers.
- (b) Chained variable-sized buffers.
- (c) One large circular buffer per connection.

Flow Control and Buffering (2)

A	<u>Message</u>	B	<u>Comments</u>
1	→ < request 8 buffers>	→	A wants 8 buffers
2	← <ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→ <seq = 0, data = m0>	→	A has 3 buffers left now
4	→ <seq = 1, data = m1>	→	A has 2 buffers left now
5	→ <seq = 2, data = m2>	...	Message lost but A thinks it has 1 left
6	← <ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→ <seq = 3, data = m3>	→	A has 1 buffer left
8	→ <seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→ <seq = 2, data = m2>	→	A times out and retransmits
10	← <ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	← <ack = 4, buf = 1>	←	A may now send 5
12	← <ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→ <seq = 5, data = m5>	→	A has 1 buffer left
14	→ <seq = 6, data = m6>	→	A is now blocked again
15	← <ack = 6, buf = 0>	←	A is still blocked
16	... <ack = 6, buf = 4>	←	Potential deadlock

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU.

Multiplexing



(a) Upward multiplexing.

(b) Downward multiplexing.

Crash Recovery

		Strategy used by receiving host					
		First ACK, then write			First write, then ACK		
Strategy used by sending host		AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit		OK	DUP	OK	OK	DUP	DUP
Never retransmit		LOST	OK	LOST	LOST	OK	OK
Retransmit in S0		OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1		LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly

DUP = Protocol generates a duplicate message

LOST = Protocol loses a message

Different combinations of client and server strategy.

A Simple Transport Protocol

- The Example Service Primitives
- The Example Transport Entity
- The Example as a Finite State Machine

The Example Transport Entity

Network packet	Meaning
CALL REQUEST	Sent to establish a connection
CALL ACCEPTED	Response to CALL REQUEST
CLEAR REQUEST	Sent to release a connection
CLEAR CONFIRMATION	Response to CLEAR REQUEST
DATA	Used to transport data
CREDIT	Control packet for managing the window

The network layer packets used in our example.

The Example Transport Entity (2)

Each connection is in one of seven states:

- B. Idle – Connection not established yet.
- C. Waiting – CONNECT has been executed, CALL REQUEST sent.
- D. Queued – A CALL REQUEST has arrived; no LISTEN yet.
- E. Established – The connection has been established.
- F. Sending – The user is waiting for permission to send a packet.
- G. Receiving – A RECEIVE has been done.
- H. DISCONNECTING – a DISCONNECT has been done locally.

The Example Transport Entity (3)

```
#define MAX_CONN 32                                /* max number of simultaneous connections */
#define MAX_MSG_SIZE 8192                          /* largest message in bytes */
#define MAX_PKT_SIZE 512                           /* largest packet in bytes */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT} pkt_type;
typedef enum {IDLE,WAITING,QUEUED,ESTABLISHED,SENDING,RECEIVING,DISCONN} cstate;

/* Global variables. */
transport_address listen_address;                /* local address being listened to */
int listen_conn;                                  /* connection identifier for listen */
unsigned char data[MAX_PKT_SIZE];                /* scratch area for packet data */

struct conn {
    transport_address local_address, remote_address;
    cstate state;                                    /* state of this connection */
    unsigned char *user_buf_addr;                   /* pointer to receive buffer */
    int byte_count;                                /* send/receive count */
    int clr_req_received;                          /* set when CLEAR_REQ packet received */
    int timer;                                     /* used to time out CALL_REQ packets */
    int credits;                                   /* number of messages that may be sent */
} conn[MAX_CONN + 1];                            /* slot 0 is not used */
```

The Example Transport Entity (4)

```
void sleep(void);                                /* prototypes */
void wakeup(void);
void to_net(int cid, int q, int m, pkt_type pt, unsigned char *p, int bytes);
void from_net(int *cid, int *q, int *m, pkt_type *pt, unsigned char *p, int *bytes);

int listen(transport_address t)
{ /* User wants to listen for a connection. See if CALL_REQ has already arrived. */
    int i, found = 0;

    for (i = 1; i <= MAX_CONN; i++)           /* search the table for CALL_REQ */
        if (conn[i].state == QUEUED && conn[i].local_address == t) {
            found = i;
            break;
        }

    if (found == 0) {
        /* No CALL_REQ is waiting. Go to sleep until arrival or timeout. */
        listen_address = t; sleep(); i = listen_conn ;
    }
    conn[i].state = ESTABLISHED;             /* connection is ESTABLISHED */
    conn[i].timer = 0;                      /* timer is not used */
```

The Example Transport Entity (5)

```

listen_conn = 0;                                /* 0 is assumed to be an invalid address */
to_net(i, 0, 0, CALL_ACC, data, 0);             /* tell net to accept connection */
return(i);                                       /* return connection identifier */
}

int connect(transport_address l, transport_address r)
{ /* User wants to connect to a remote process; send CALL_REQ packet. */
    int i;
    struct conn *cptr;

    data[0] = r;   data[1] = l;                      /* CALL_REQ packet needs these */
    i = MAX_CONN;                                     /* search table backward */
    while (conn[i].state != IDLE && i > 1) i = i - 1;
    if (conn[i].state == IDLE) {
        /* Make a table entry that CALL_REQ has been sent. */
        cptr = &conn[i];
        cptr->local_address = l; cptr->remote_address = r;
        cptr->state = WAITING; cptr->clr_req_received = 0;
        cptr->credits = 0; cptr->timer = 0;
        to_net(i, 0, 0, CALL_REQ, data, 2);
        sleep();                                         /* wait for CALL_ACC or CLEAR_REQ */
        if (cptr->state == ESTABLISHED) return(i);
        if (cptr->clr_req_received) {
            /* Other side refused call. */
            cptr->state = IDLE;                      /* back to IDLE state */
            to_net(i, 0, 0, CLEAR_CONF, data, 0);
            return(ERR_REJECT);
        }
    } else return(ERR_FULL);                         /* reject CONNECT: no table space */
}

```

The Example Transport Entity (6)

```
int send(int cid, unsigned char bufptr[], int bytes)
{ /* User wants to send a message. */
    int i, count, m;
    struct conn *cptr = &conn[cid];

    /* Enter SENDING state. */
    cptr->state = SENDING;
    cptr->byte_count = 0;                      /* # bytes sent so far this message */
    if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
    if (cptr->clr_req_received == 0) {
        /* Credit available; split message into packets if need be. */
        do {
            if (bytes - cptr->byte_count > MAX_PKT_SIZE) /* multipacket message */
                count = MAX_PKT_SIZE; m = 1; /* more packets later */
            } else {                                /* single packet message */
                count = bytes - cptr->byte_count; m = 0; /* last pkt of this message */
            }
            for (i = 0; i < count; i++) data[i] = bufptr[cptr->byte_count + i];
            to_net(cid, 0, m, DATA_PKT, data, count); /* send 1 packet */
            cptr->byte_count = cptr->byte_count + count; /* increment bytes sent so far */
        } while (cptr->byte_count < bytes);      /* loop until whole message sent */
}
```

The Example Transport Entity (7)

```
    cptr->credits --;                                /* each message uses up one credit */
    cptr->state = ESTABLISHED;
    return(OK);
} else {
    cptr->state = ESTABLISHED;
    return(ERR_CLOSED);                            /* send failed: peer wants to disconnect */
}
}

int receive(int cid, unsigned char bufptr[], int *bytes)
{ /* User is prepared to receive a message. */
    struct conn *cptr = &conn[cid];

    if (cptr->clr_req_received == 0) {
        /* Connection still established; try to receive. */
        cptr->state = RECEIVING;
        cptr->user_buf_addr = bufptr;
        cptr->byte_count = 0;
        data[0] = CRED;
        data[1] = 1;
        to_net(cid, 1, 0, CREDIT, data, 2);      /* send credit */
        sleep();                                /* block awaiting data */
        *bytes = cptr->byte_count;
    }
    cptr->state = ESTABLISHED;
    return(cptr->clr_req_received ? ERR_CLOSED : OK);
}
```

The Example Transport Entity (8)

```
int disconnect(int cid)
{ /* User wants to release a connection. */
    struct conn *cptr = &conn[cid];

    if (cptr->clr_req_received) {           /* other side initiated termination */
        cptr->state = IDLE;                /* connection is now released */
        to_net(cid, 0, 0, CLEAR_CONF, data, 0);
    } else {                                /* we initiated termination */
        cptr->state = DISCONN;             /* not released until other side agrees */
        to_net(cid, 0, 0, CLEAR_REQ, data, 0);
    }
    return(OK);
}

void packet_arrival(void)
{ /* A packet has arrived, get and process it. */
    int cid;                               /* connection on which packet arrived */
    int count, i, q, m;
    pkt_type ptype; /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
    unsigned char data[MAX_PKT_SIZE];      /* data portion of the incoming packet */
    struct conn *cptr;

    from_net(&cid, &q, &m, &ptype, data, &count); /* go get it */
    cptr = &conn[cid];
```

The Example Transport Entity (9)

```
switch (ptype) {
    case CALL_REQ:                                /* remote user wants to establish connection */
        cptr->local_address = data[0]; cptr->remote_address = data[1];
        if (cptr->local_address == listen_address) {
            listen_conn = cid; cptr->state = ESTABLISHED; wakeup();
        } else {
            cptr->state = QUEUED; cptr->timer = TIMEOUT;
        }
        cptr->clr_req_received = 0; cptr->credits = 0;
        break;

    case CALL_ACC:                                 /* remote user has accepted our CALL_REQ */
        cptr->state = ESTABLISHED;
        wakeup();
        break;

    case CLEAR_REQ:                               /* remote user wants to disconnect or reject call */
        cptr->clr_req_received = 1;
        if (cptr->state == DISCONN) cptr->state = IDLE; /* clear collision */
        if (cptr->state == WAITING || cptr->state == RECEIVING || cptr->state == SENDING) wakeup();
        break;

    case CLEAR_CONF:                             /* remote user agrees to disconnect */
        cptr->state = IDLE;
        break;

    case CREDIT:                                  /* remote user is waiting for data */
        cptr->credits += data[1];
        if (cptr->state == SENDING) wakeup();
        break;

    case DATA_PKT:                               /* remote user has sent data */
        for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
        cptr->byte_count += count;
        if (m == 0) wakeup();
    }

}
```

The Example Transport Entity (10)

```
}
```

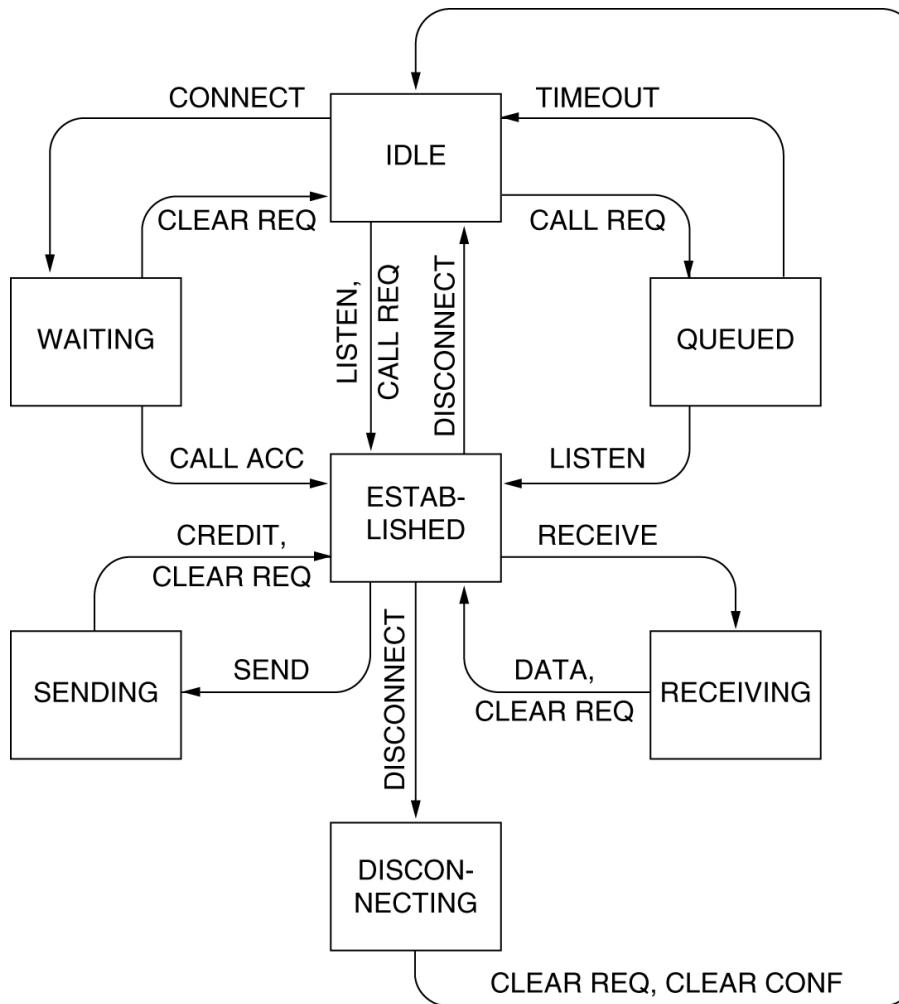
```
void clock(void)
{ /* The clock has ticked, check for timeouts of queued connect requests. */
    int i;
    struct conn *cptr;
```

```
for (i = 1; i <= MAX_CONN; i++) {
    cptr = &conn[i];
    if (cptr->timer > 0) {                      /* timer was running */
        cptr->timer--;
        if (cptr->timer == 0) {                    /* timer has now expired */
            cptr->state = IDLE;
            to_net(i, 0, 0, CLEAR_REQ, data, 0);
        }
    }
}
```

The Example as a Finite State Machine

The example protocol as a finite state machine. Each entry has an optional predicate, an optional action, and the new state. The tilde indicates that no major action is taken. An overbar above a predicate indicate the negation of the predicate. Blank entries correspond to impossible or invalid events.

The Example as a Finite State Machine (2)

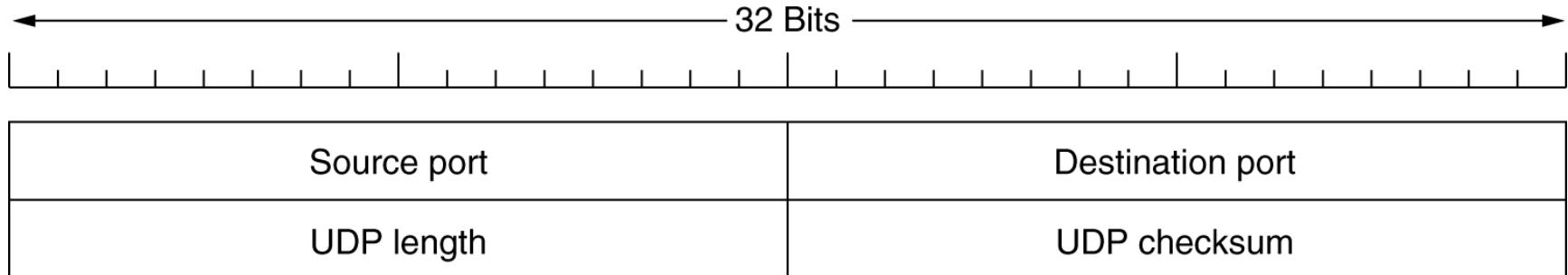


The example protocol in graphical form. Transitions that leave the connection state unchanged have been omitted for simplicity.

The Internet Transport Protocols: UDP

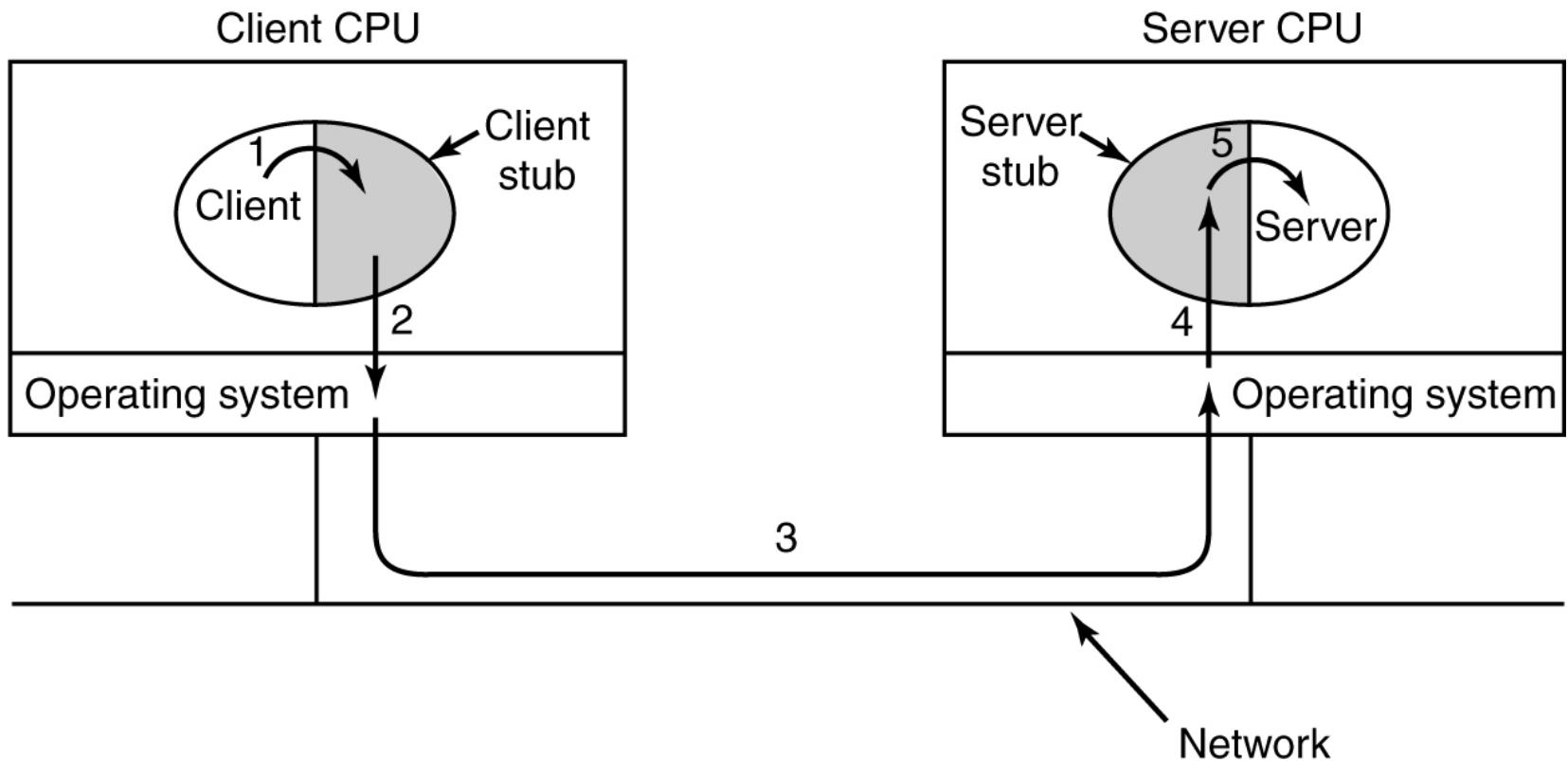
- Introduction to UDP
- Remote Procedure Call
- The Real-Time Transport Protocol

Introduction to UDP



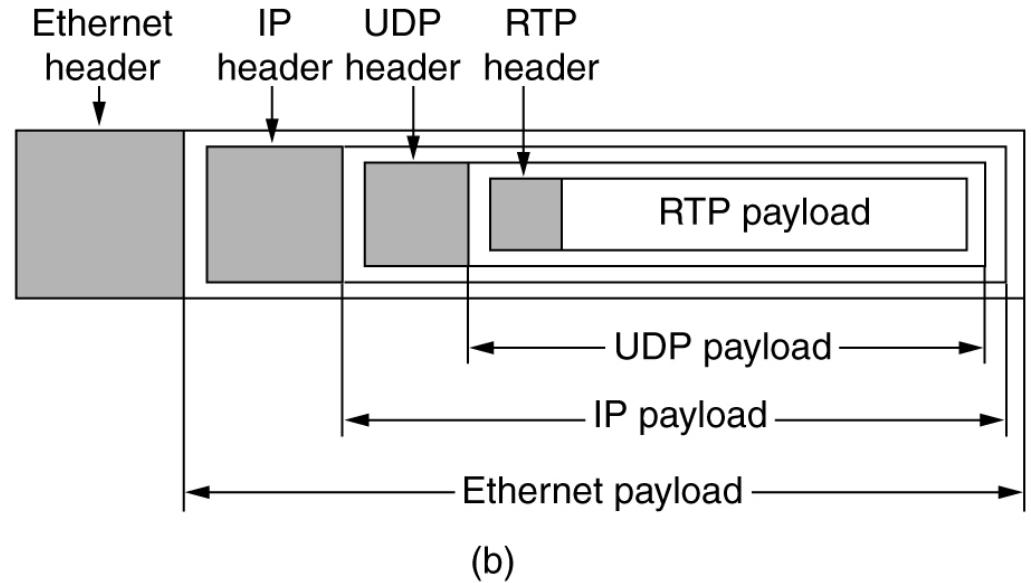
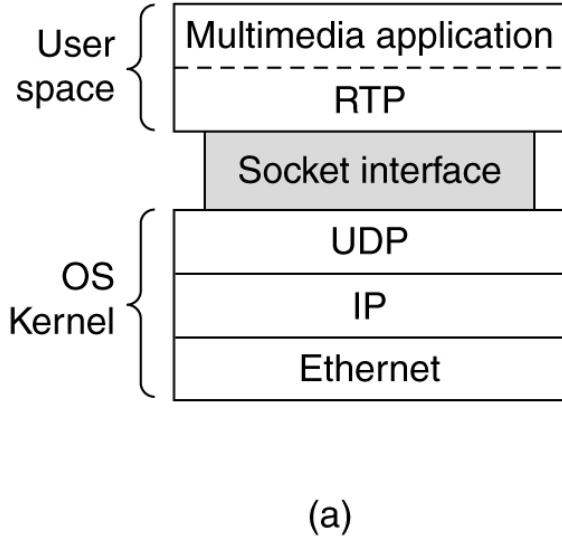
The UDP header.

Remote Procedure Call



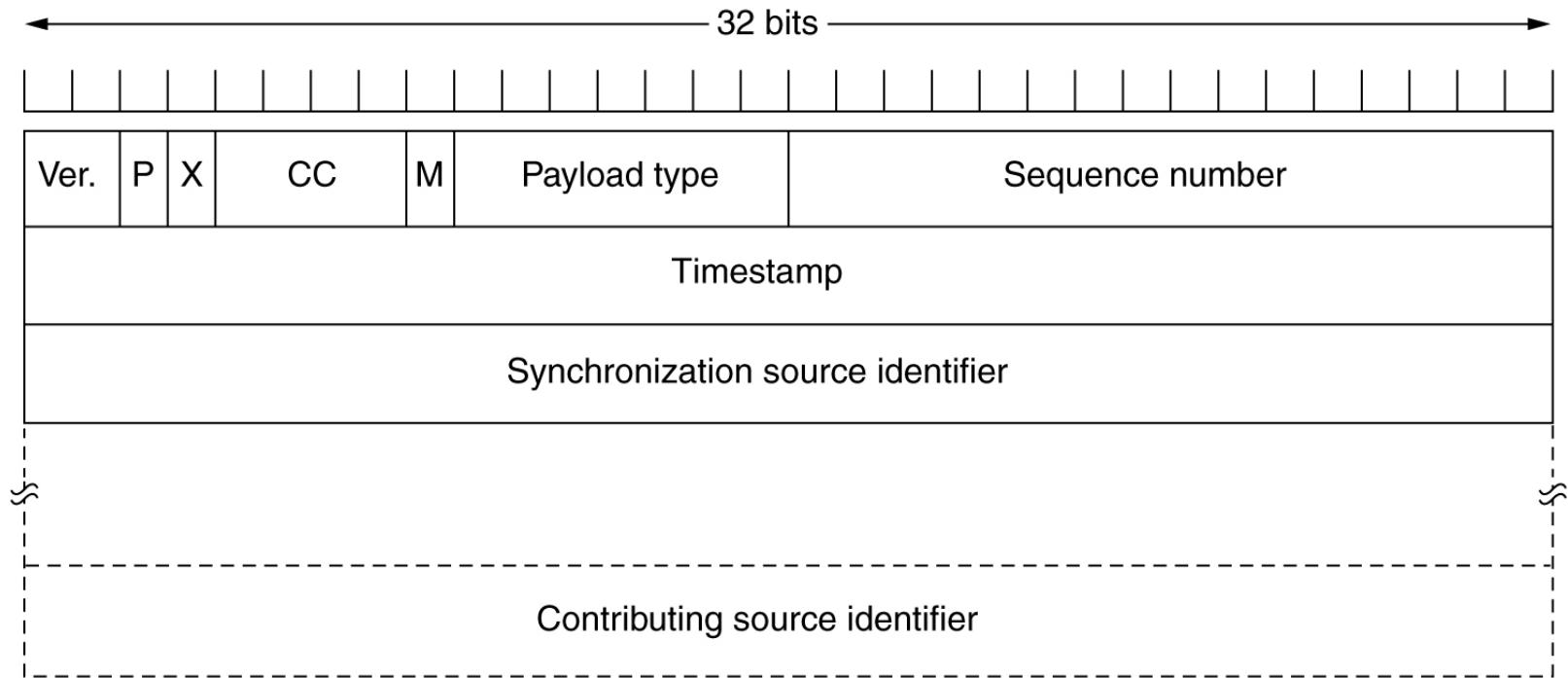
Steps in making a remote procedure call. The stubs are shaded.

The Real-Time Transport Protocol



(a) The position of RTP in the protocol stack. (b) Packet nesting.

The Real-Time Transport Protocol (2)



The RTP header.

The Internet Transport Protocols: TCP

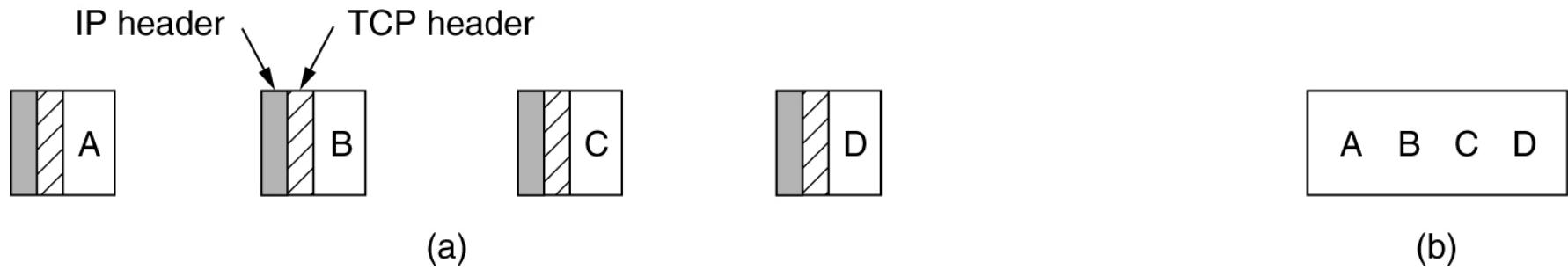
- Introduction to TCP
- The TCP Service Model
- The TCP Protocol
- The TCP Segment Header
- TCP Connection Establishment
- TCP Connection Release
- TCP Connection Management Modeling
- TCP Transmission Policy
- TCP Congestion Control
- TCP Timer Management
- Wireless TCP and UDP
- Transactional TCP

The TCP Service Model

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

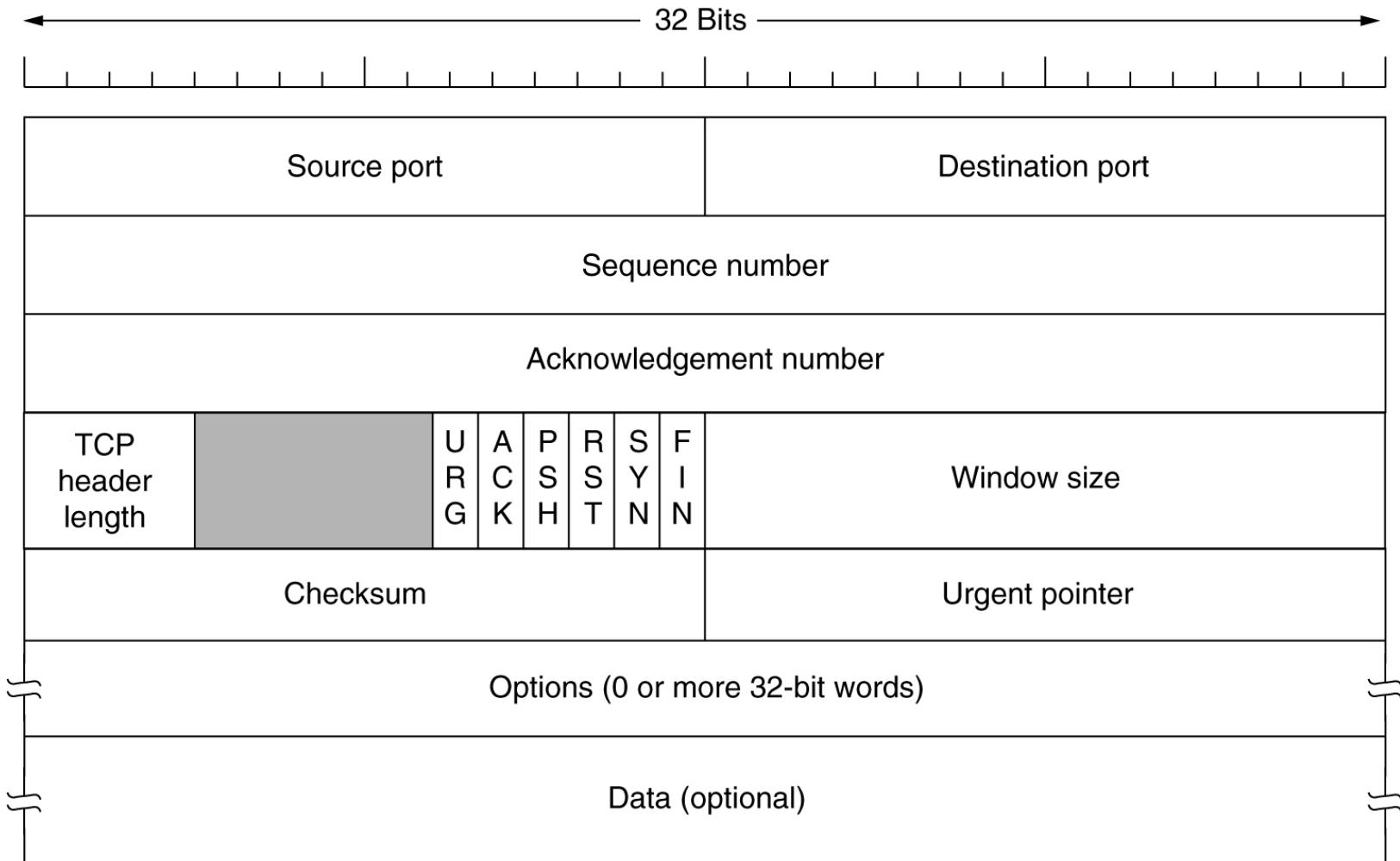
Some assigned ports.

The TCP Service Model (2)



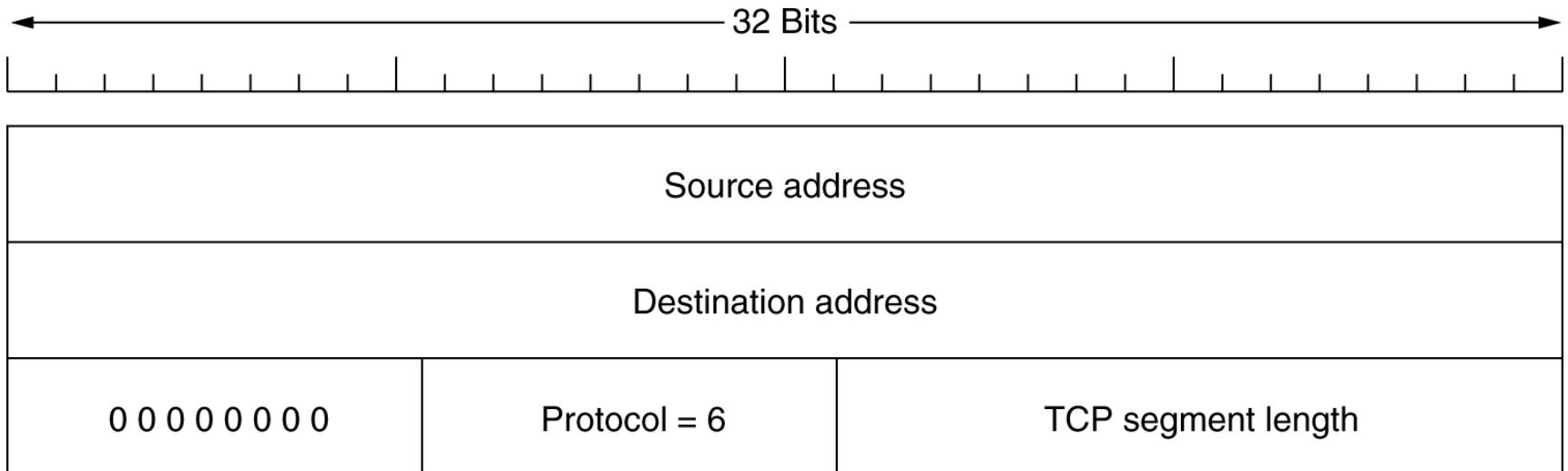
- (a) Four 512-byte segments sent as separate IP datagrams.
- (b) The 2048 bytes of data delivered to the application in a single READ CALL.

The TCP Segment Header



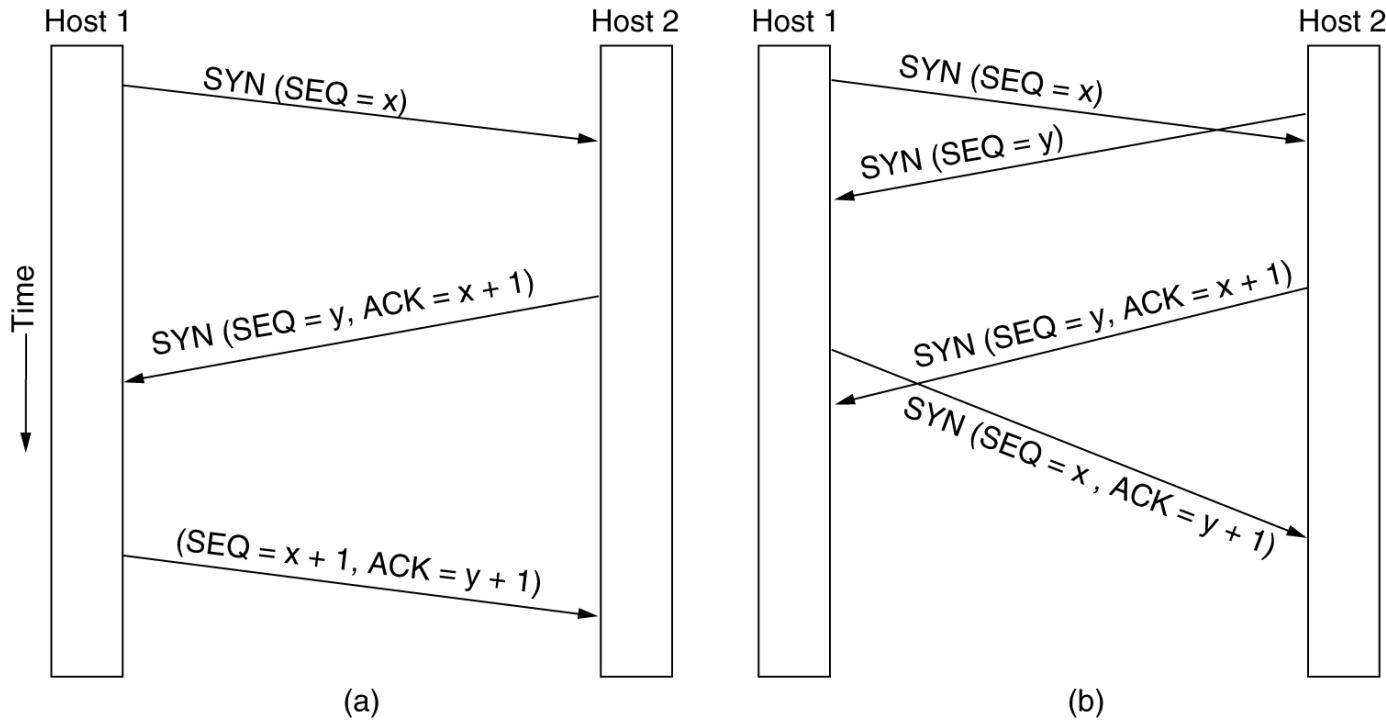
TCP Header.

The TCP Segment Header (2)



The pseudoheader included in the TCP checksum.

TCP Connection Establishment



- (a) TCP connection establishment in the normal case.
- (b) Call collision.

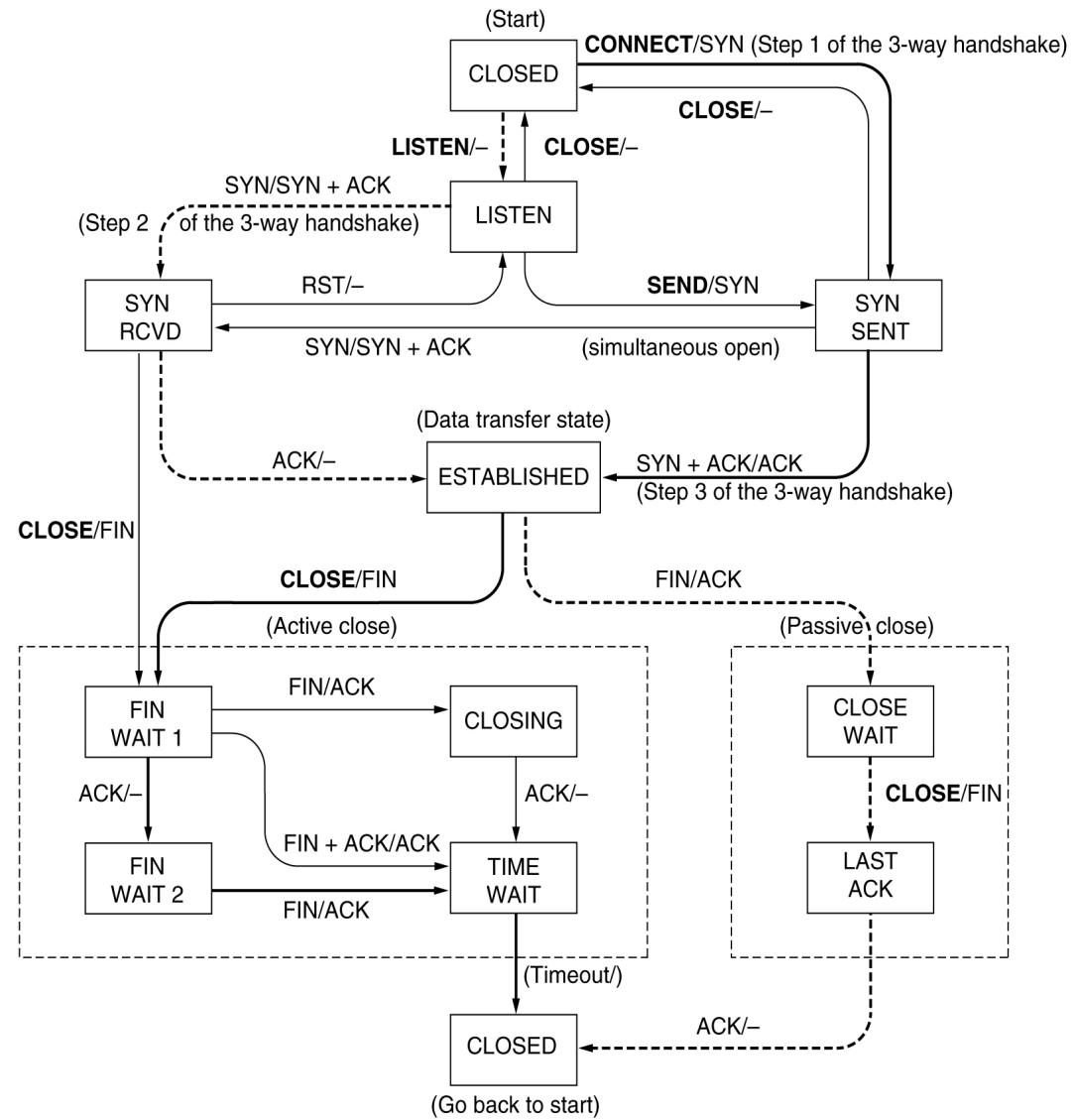
TCP Connection Management Modeling

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

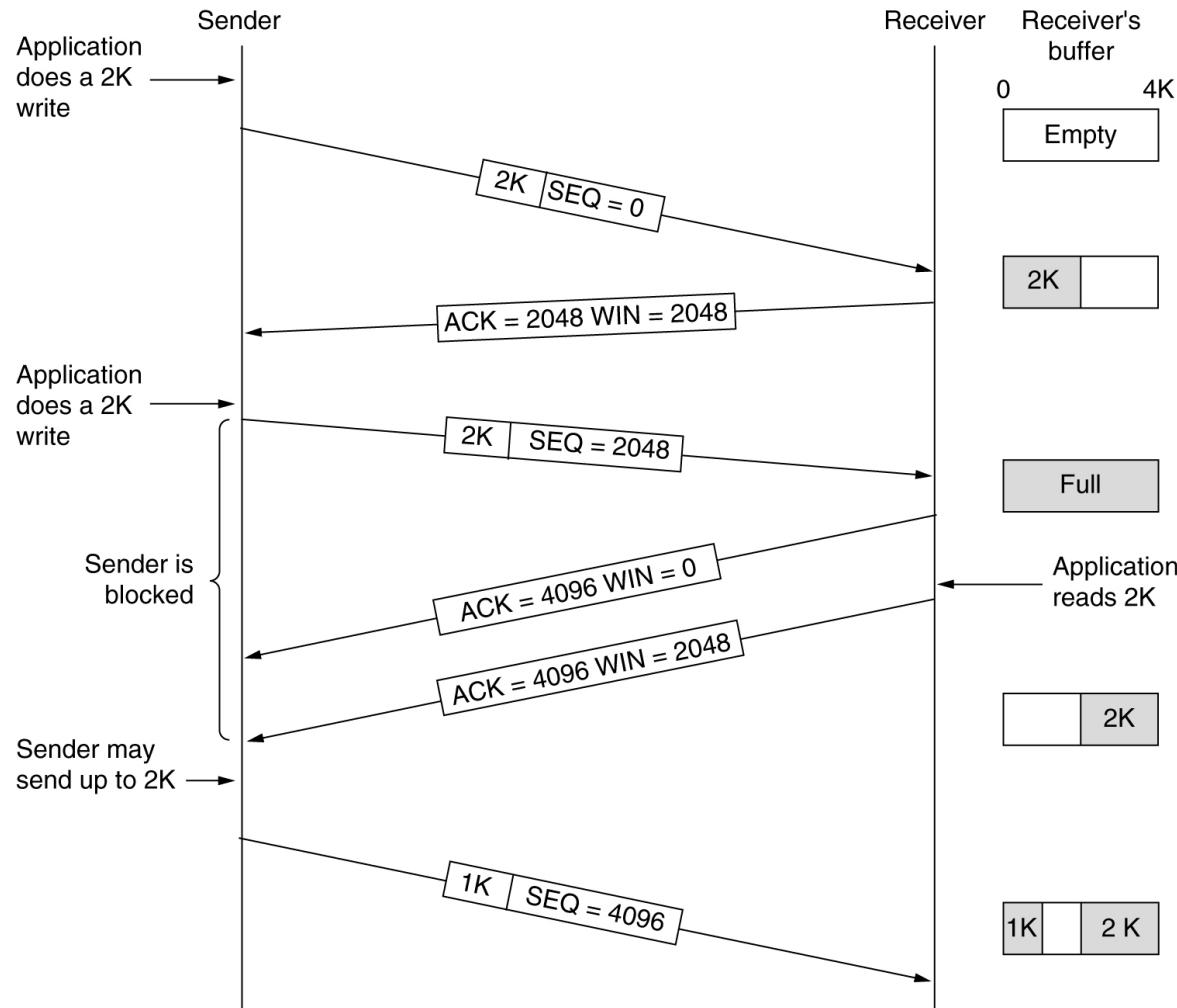
The states used in the TCP connection management finite state machine.

TCP Connection Management Modeling (2)

TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

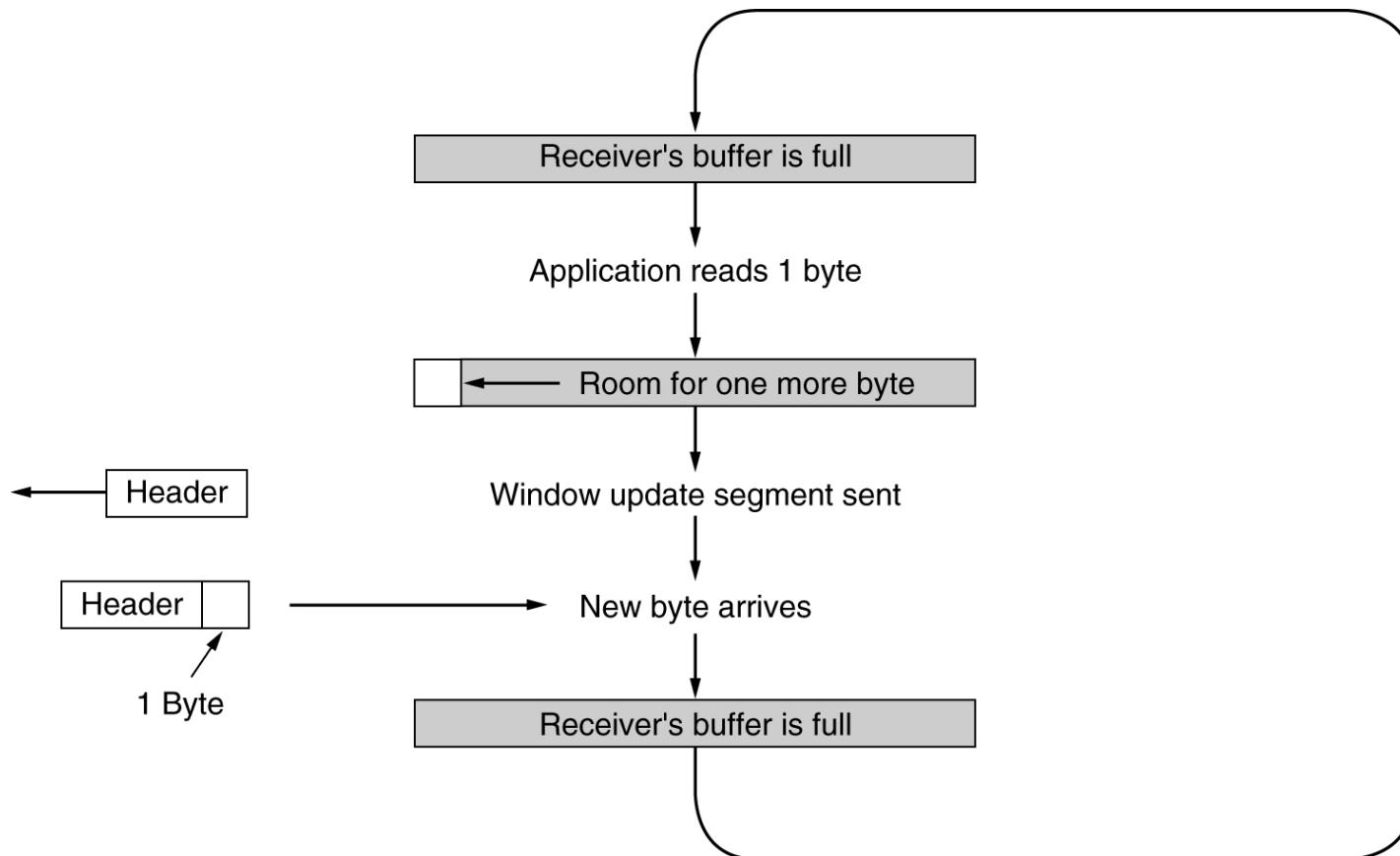


TCP Transmission Policy



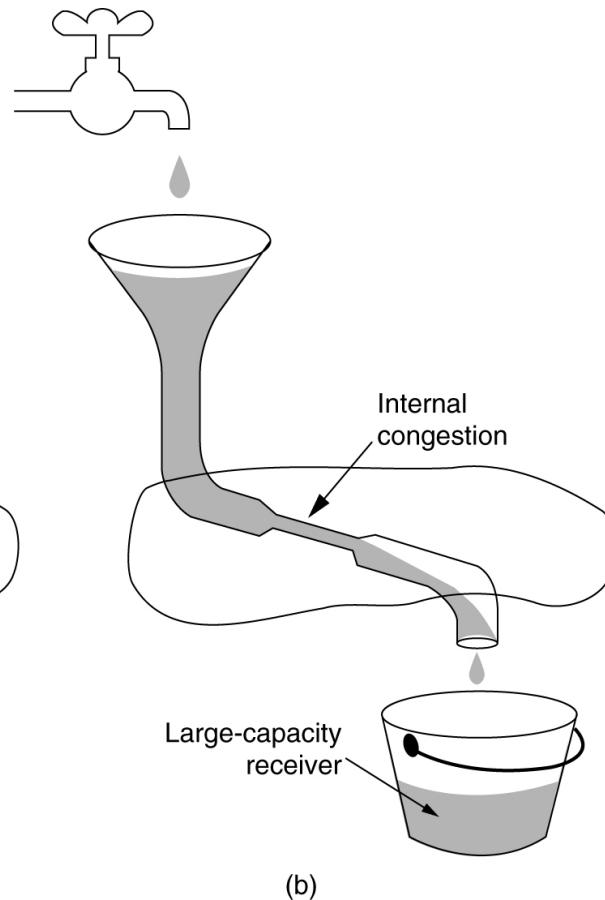
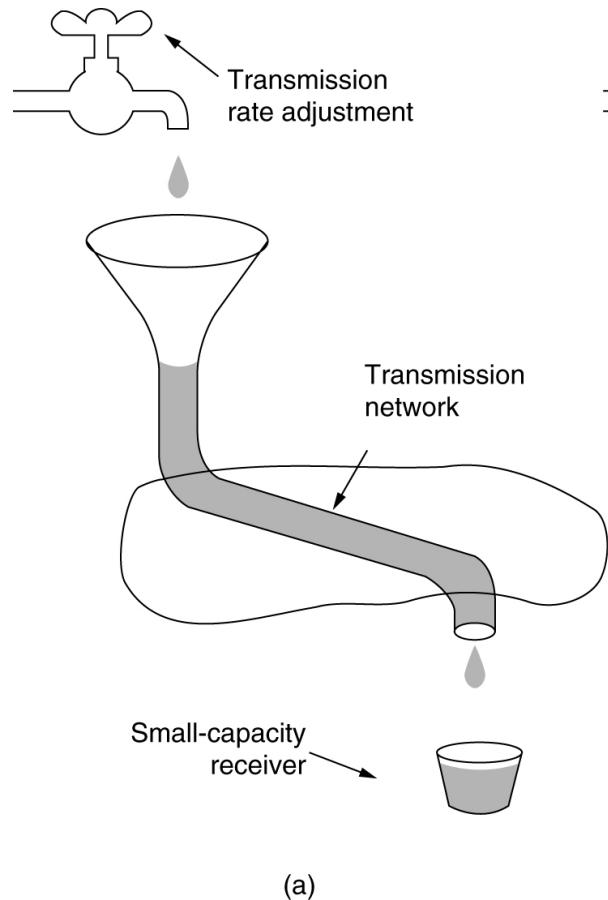
Window management in TCP.

TCP Transmission Policy (2)



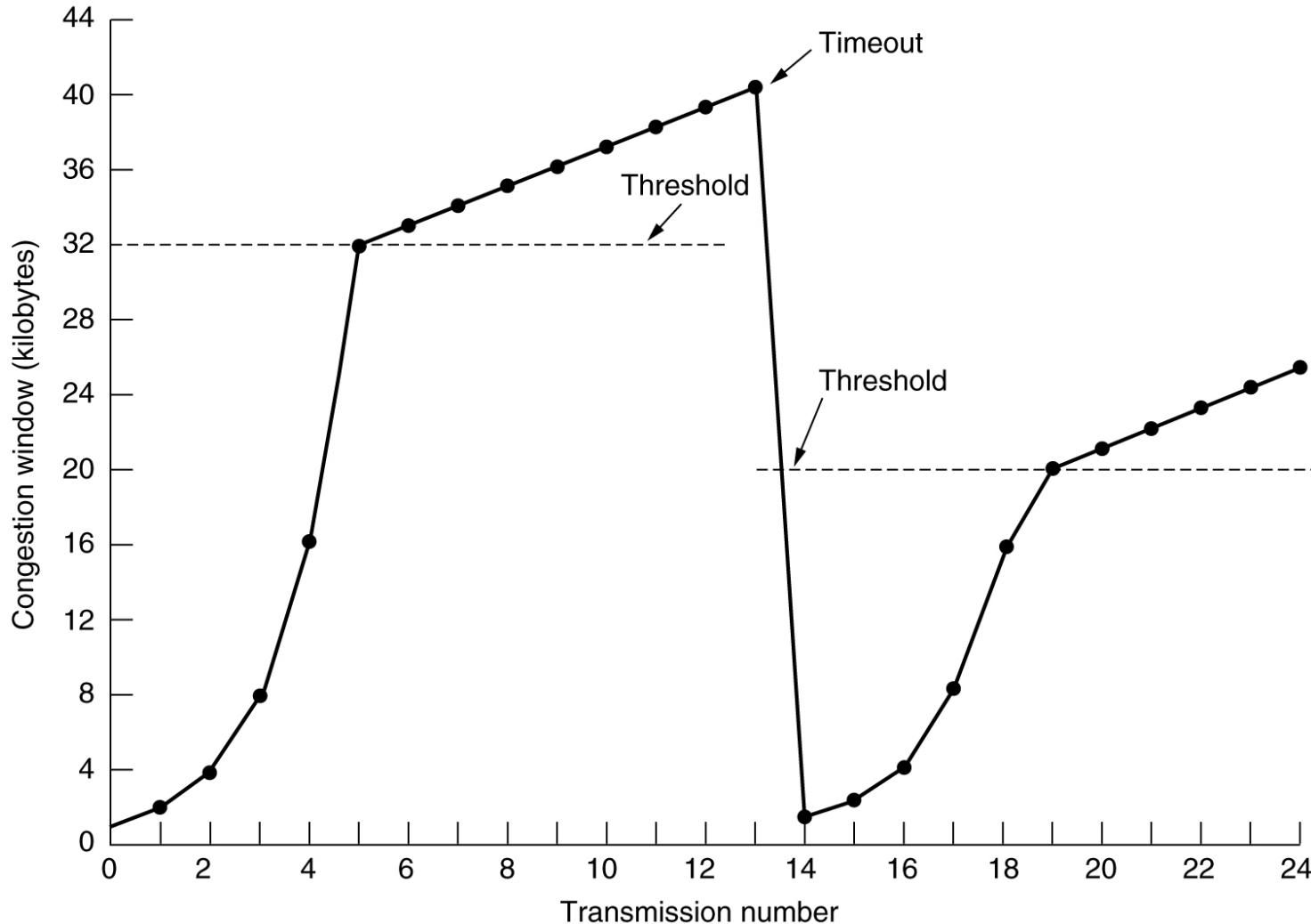
Silly window syndrome.

TCP Congestion Control



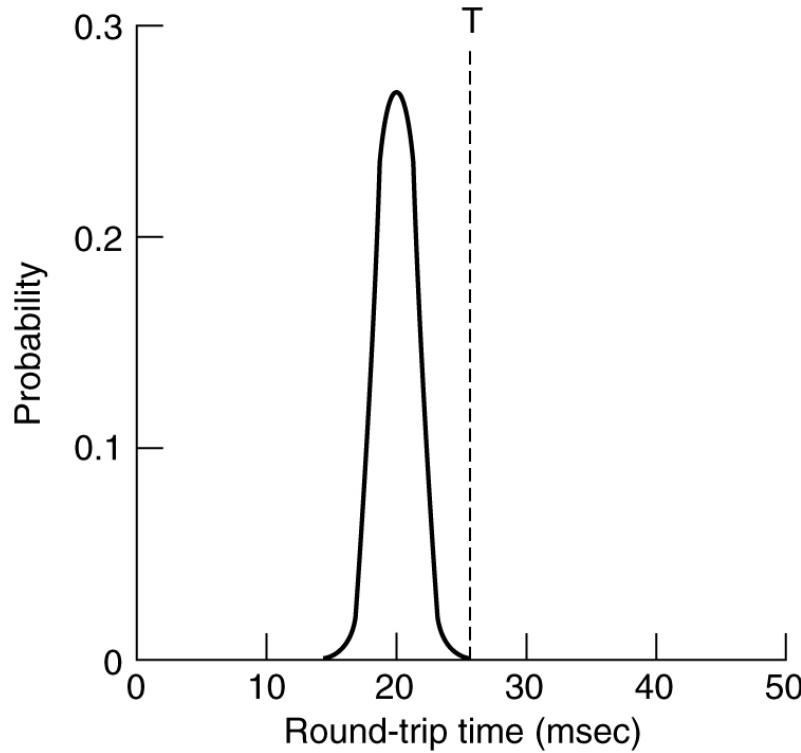
- (a) A fast network feeding a low capacity receiver.
- (b) A slow network feeding a high-capacity receiver.

TCP Congestion Control (2)

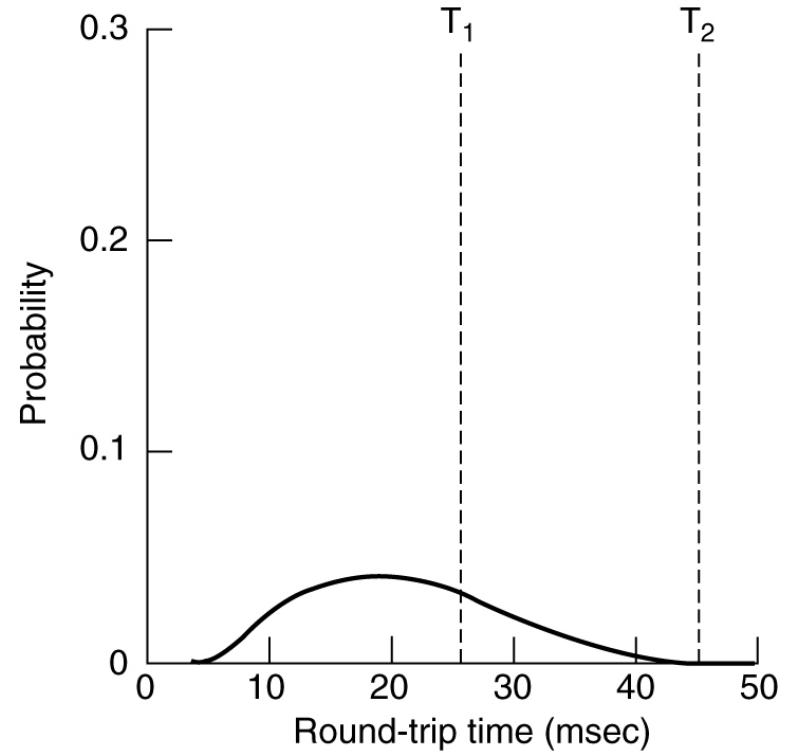


An example of the Internet congestion algorithm.

TCP Timer Management



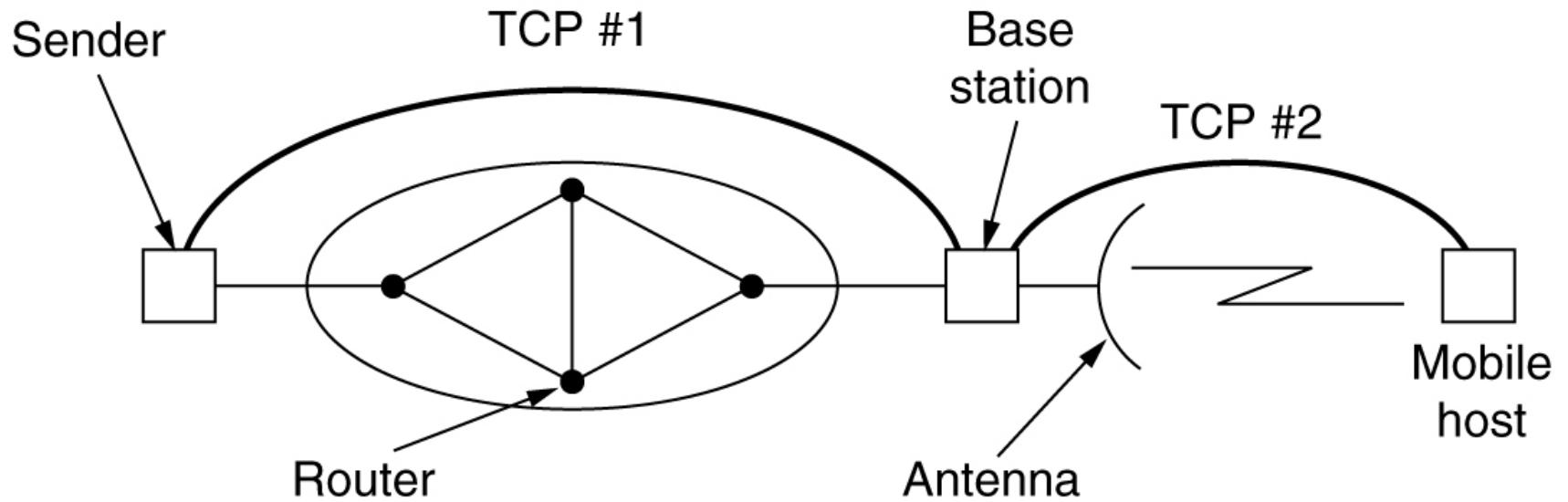
(a)



(b)

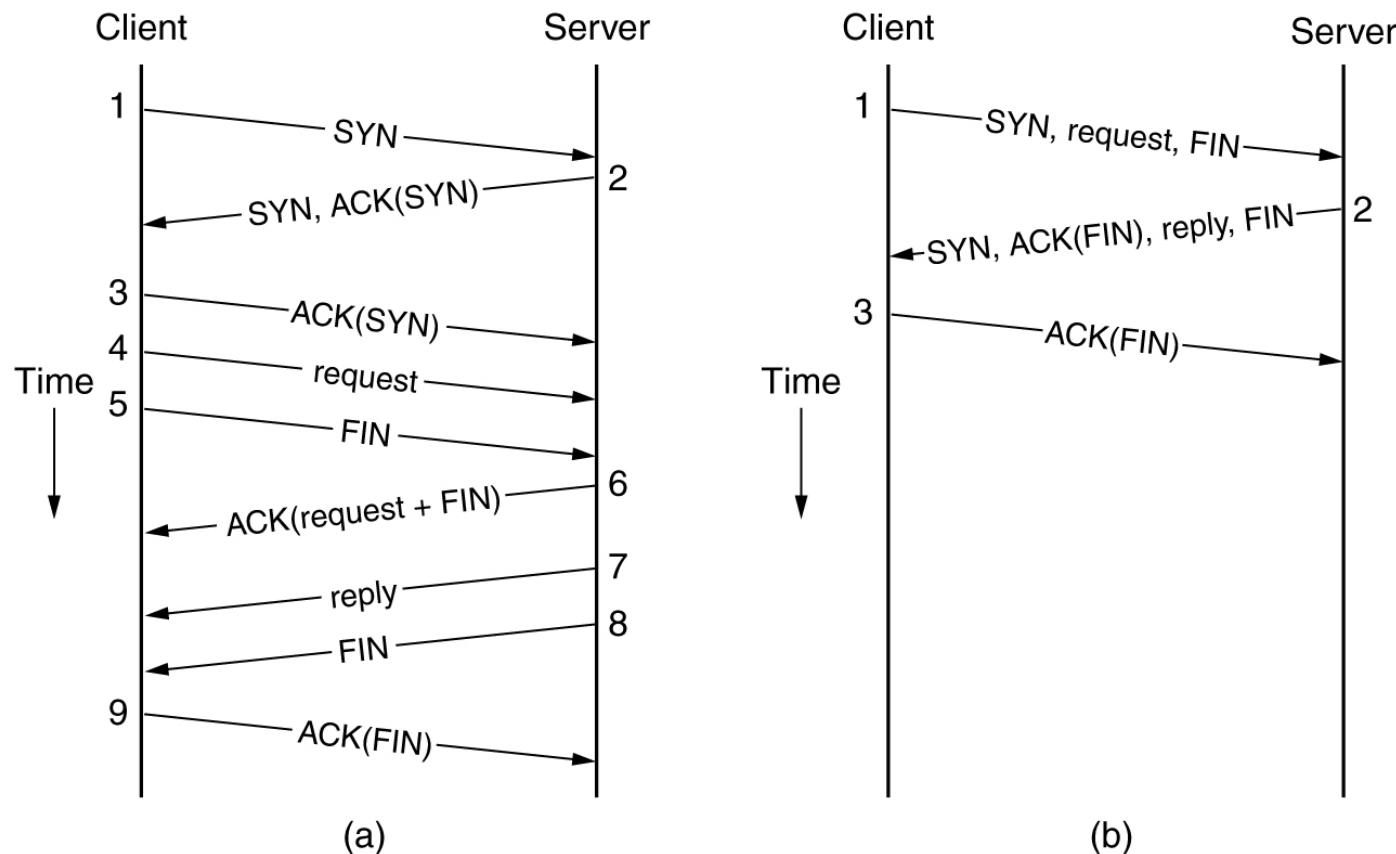
- (a) Probability density of ACK arrival times in the data link layer.
- (b) Probability density of ACK arrival times for TCP.

Wireless TCP and UDP



Splitting a TCP connection into two connections.

Transitional TCP

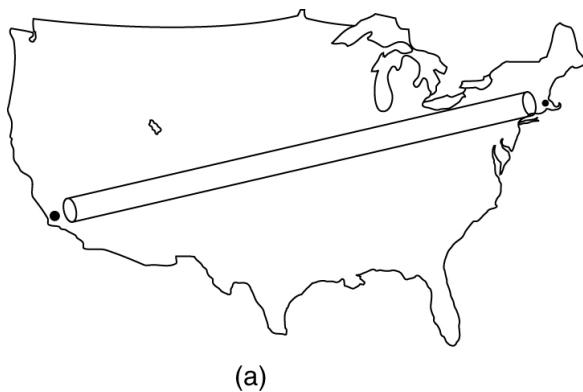


- (a) RPC using normal TPC.
(b) RPC using T/TCP.

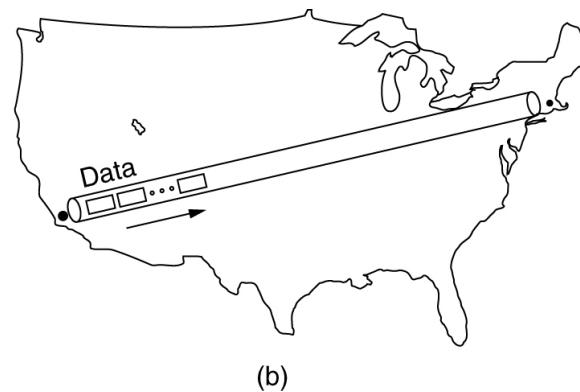
Performance Issues

- Performance Problems in Computer Networks
- Network Performance Measurement
- System Design for Better Performance
- Fast TPDU Processing
- Protocols for Gigabit Networks

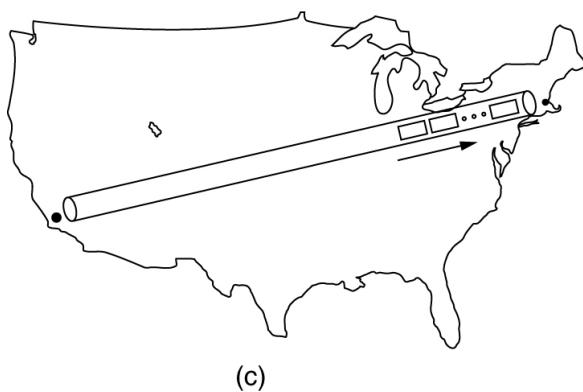
Performance Problems in Computer Networks



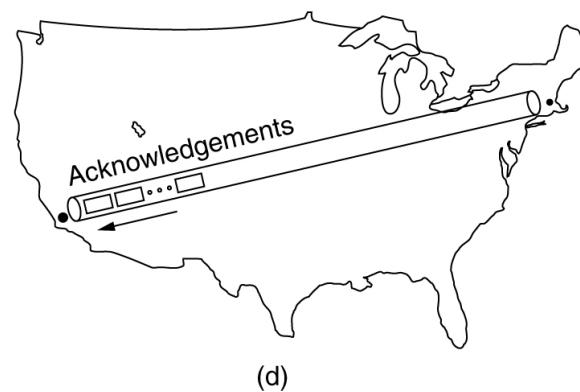
(a)



(b)



(c)



(d)

The state of transmitting one megabit from San Diego to Boston

(a) At $t = 0$, (b) After $500 \mu\text{sec}$, (c) After 20 msec , (d) after 40 msec .

Network Performance Measurement

The basic loop for improving network performance.

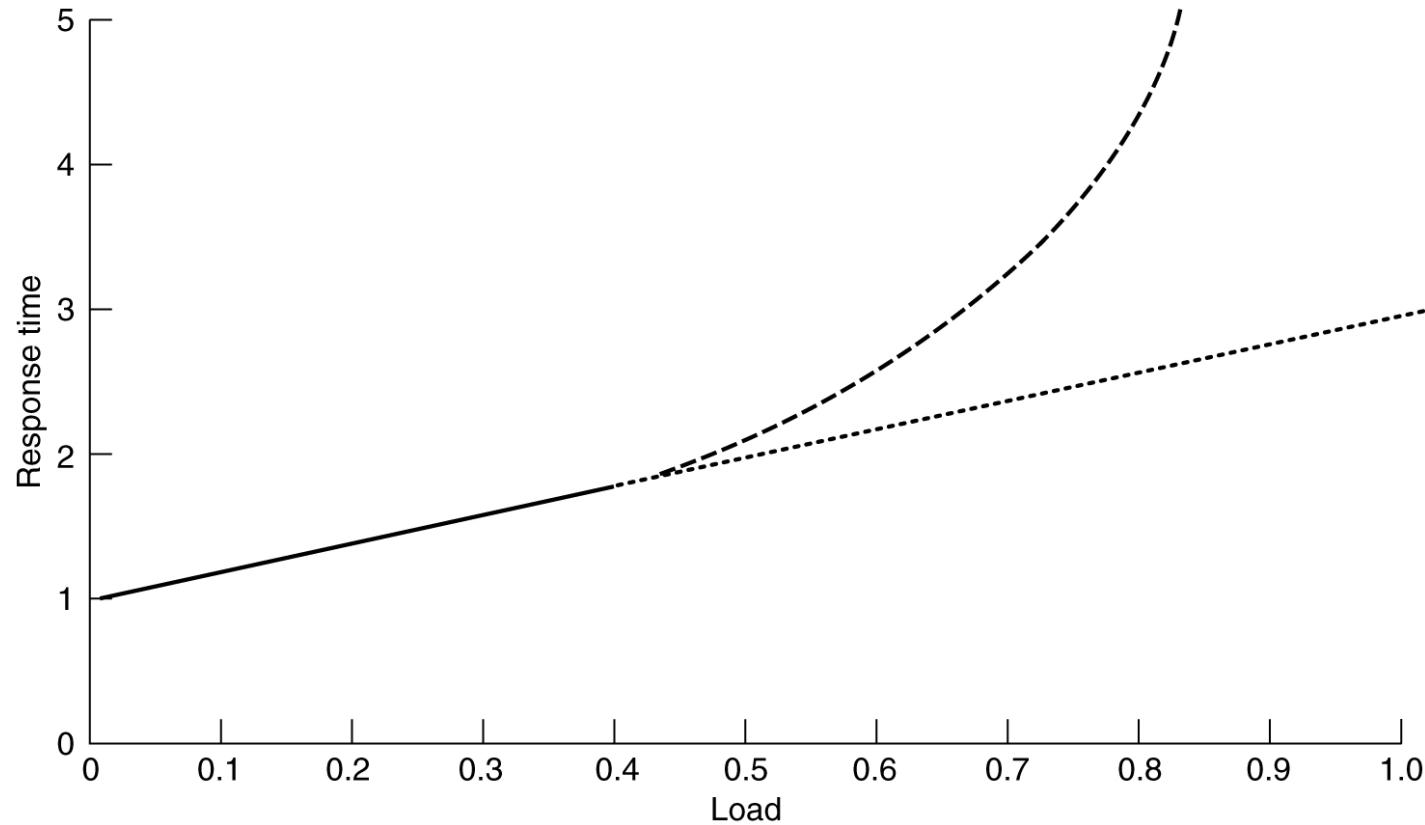
- B. Measure relevant network parameters, performance.
- C. Try to understand what is going on.
- D. Change one parameter.

System Design for Better Performance

Rules:

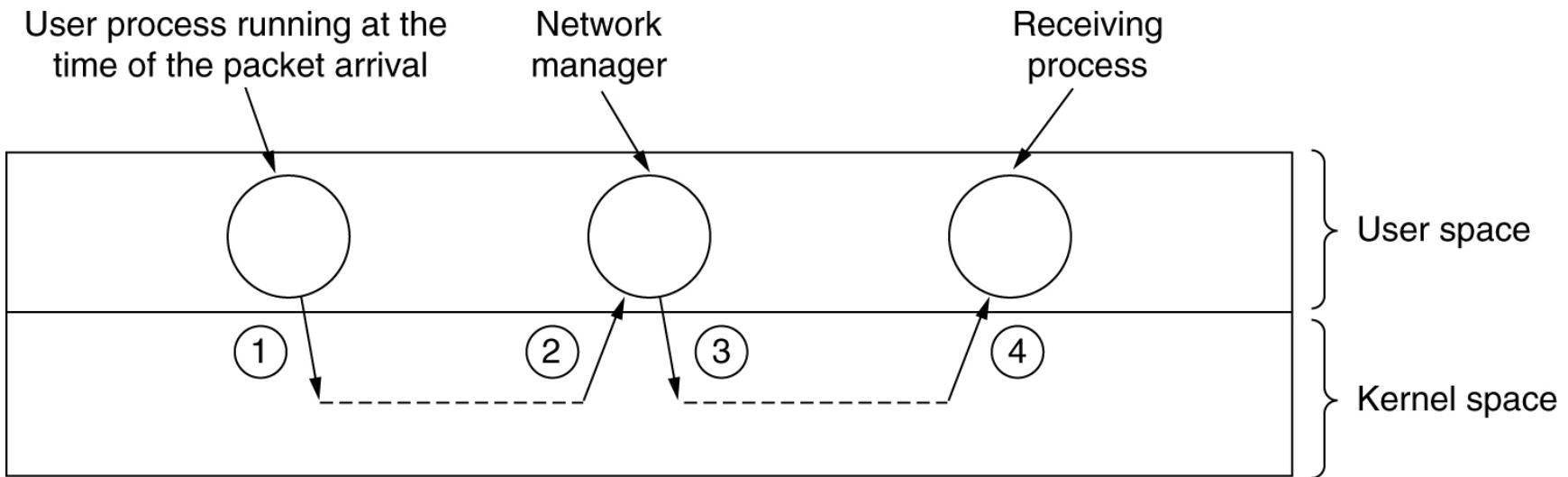
- B. CPU speed is more important than network speed.
- C. Reduce packet count to reduce software overhead.
- D. Minimize context switches.
- E. Minimize copying.
- F. You can buy more bandwidth but not lower delay.
- G. Avoiding congestion is better than recovering from it.
- H. Avoid timeouts.

System Design for Better Performance (2)



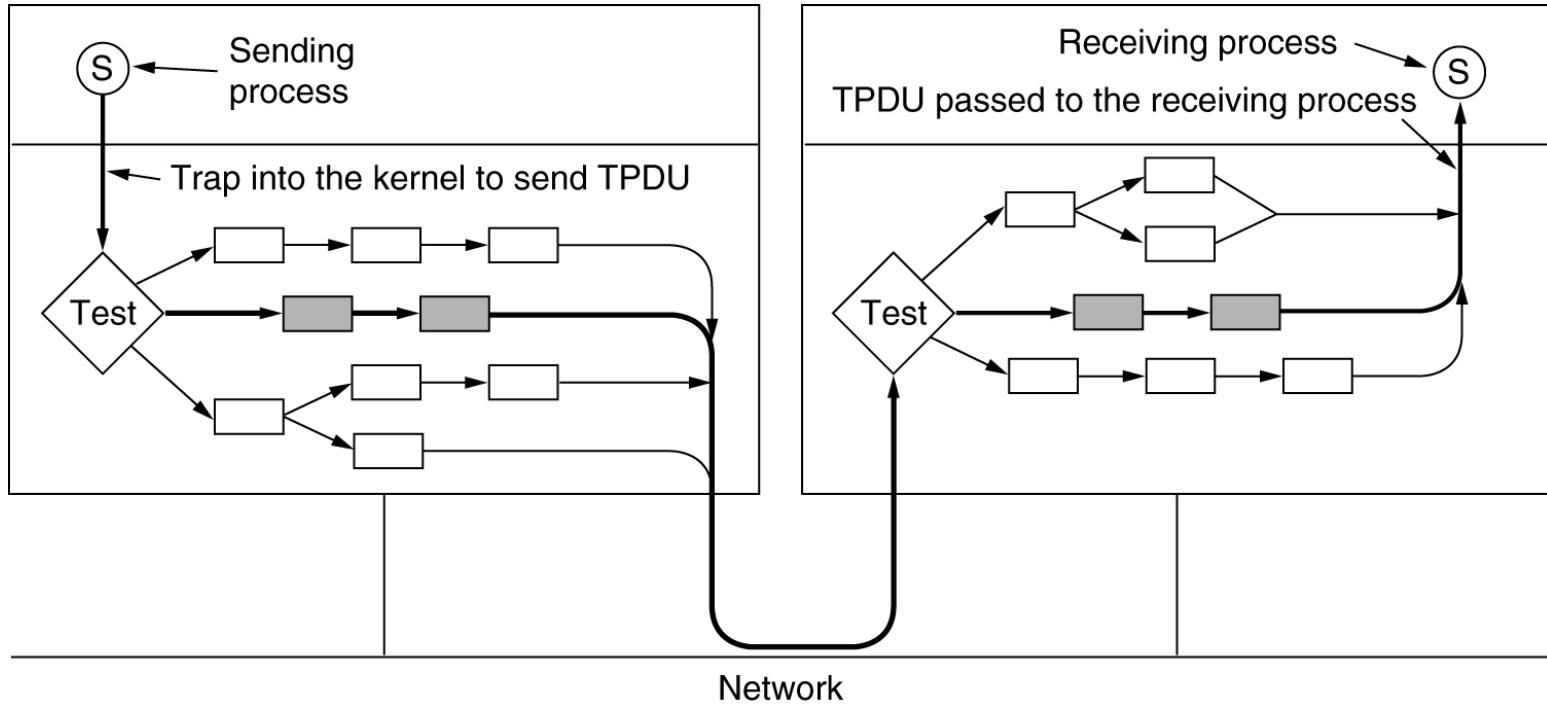
Response as a function of load.

System Design for Better Performance (3)



Four context switches to handle one packet
with a user-space network manager.

Fast TPDU Processing



The fast path from sender to receiver is shown with a heavy line.
The processing steps on this path are shaded.

Fast TPDU Processing (2)

Source port	Destination port
Sequence number	
Acknowledgement number	
Len	Unused
Checksum	Window size
Unused	Urgent pointer

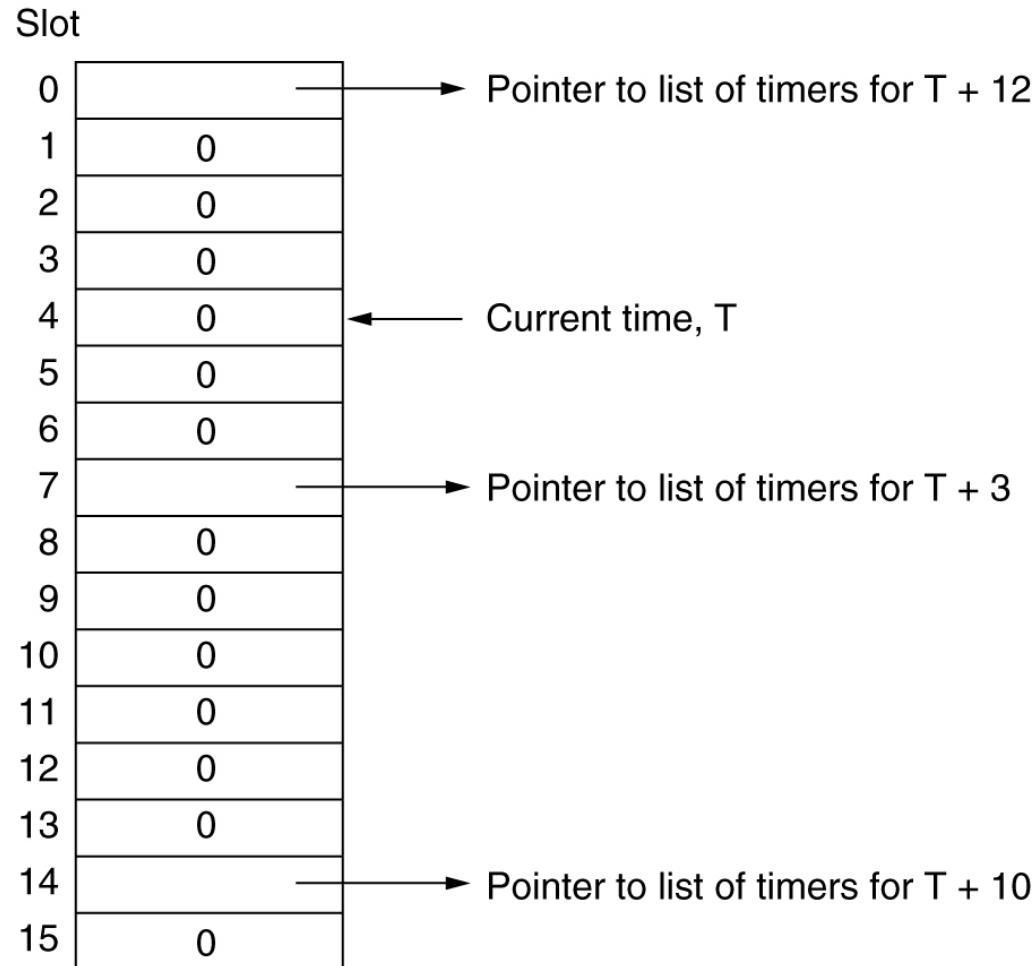
(a)

VER.	IHL	TOS	Total length
Identification			Fragment offset
TTL	Protocol	Header checksum	
Source address			Unused
Destination address			Unused

(b)

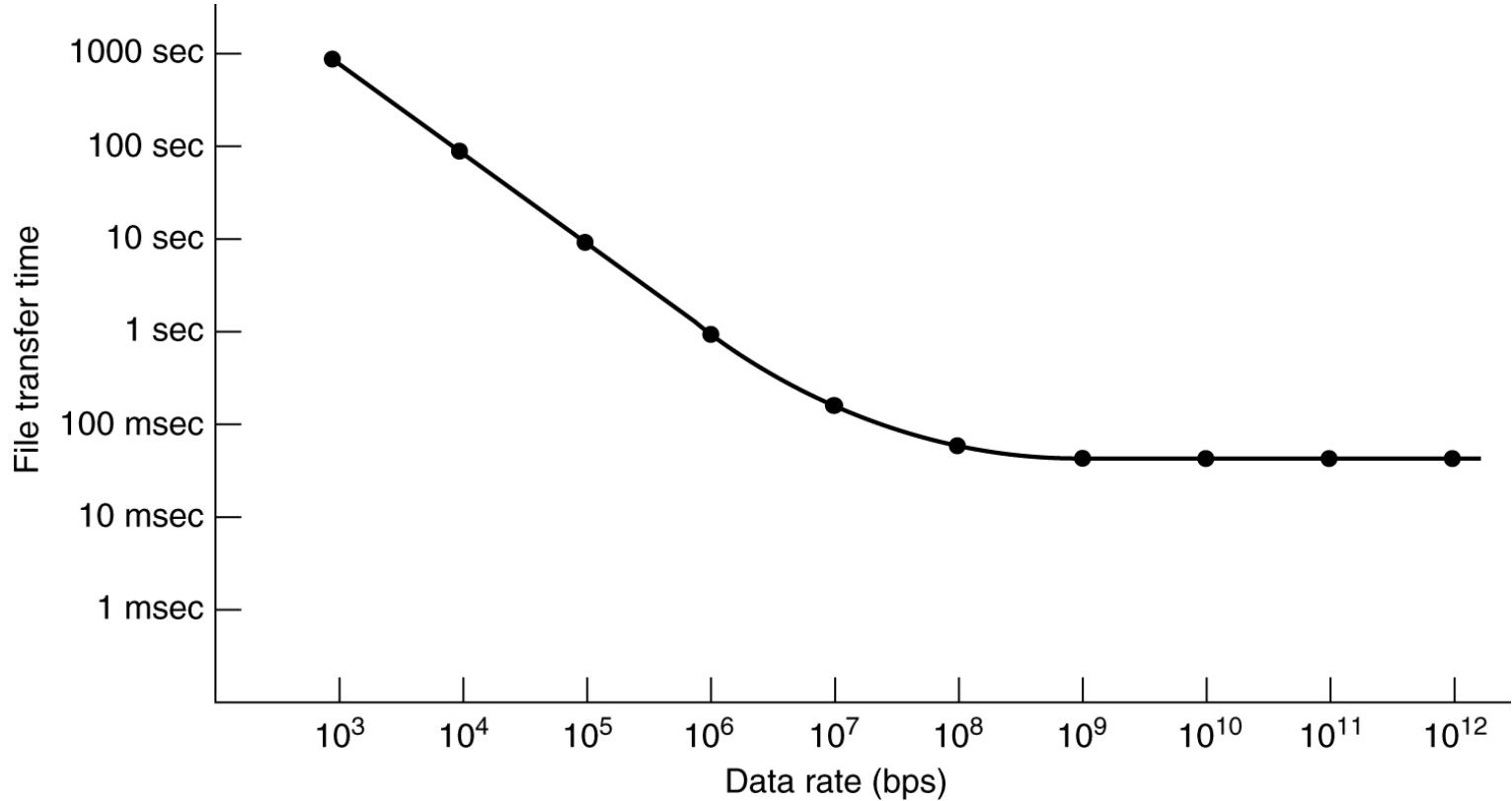
(a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Fast TPDU Processing (3)



A timing wheel.

Protocols for Gigabit Networks



Time to transfer and acknowledge a 1-megabit file over a 4000-km line.



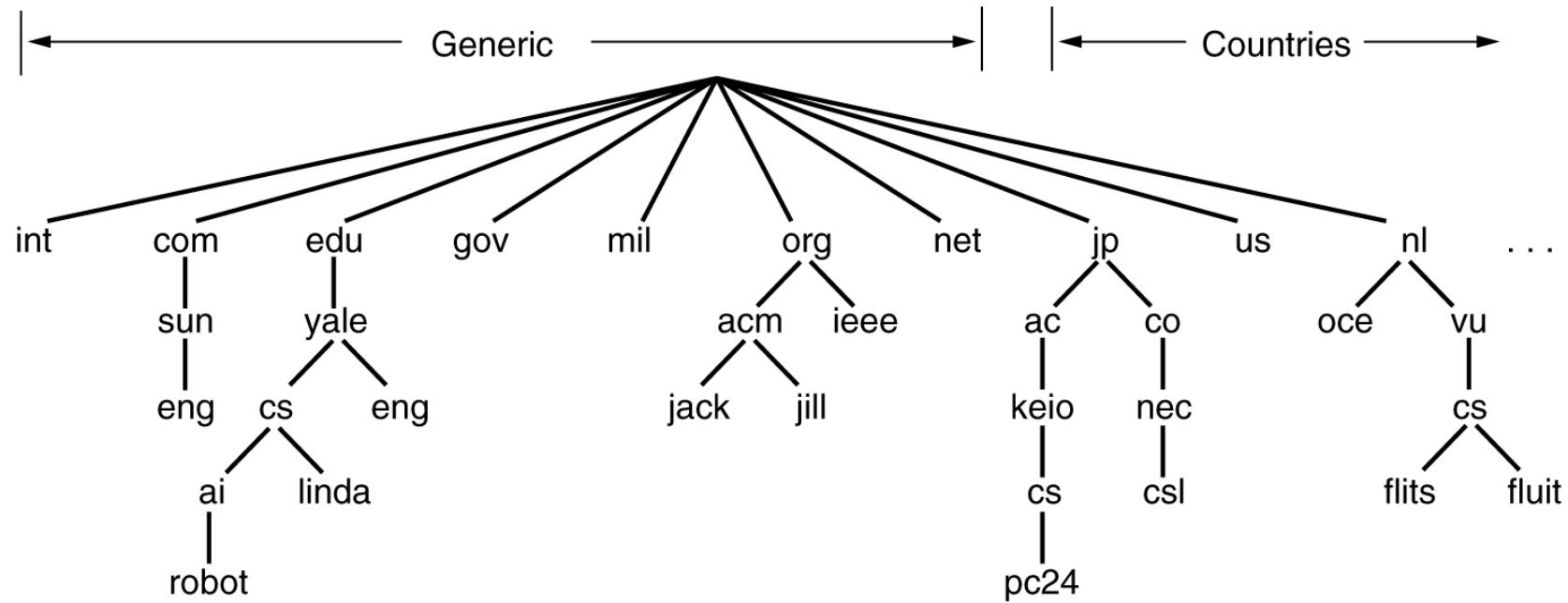
Chapter 7

The Application Layer

DNS – The Domain Name System

- The DNS Name Space
- Resource Records
- Name Servers

The DNS Name Space



A portion of the Internet domain name space.

Resource Records

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

The principal DNS resource records types.

Resource Records (2)

```
; Authoritative data for cs.vu.nl
cs.vu.nl.      86400  IN  SOA   star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.      86400  IN  TXT   "Divisie Wiskunde en Informatica."
cs.vu.nl.      86400  IN  TXT   "Vrije Universiteit Amsterdam."
cs.vu.nl.      86400  IN  MX    1 zephyr.cs.vu.nl.
cs.vu.nl.      86400  IN  MX    2 top.cs.vu.nl.

flits.cs.vu.nl. 86400  IN  HINFO Sun Unix
flits.cs.vu.nl. 86400  IN  A    130.37.16.112
flits.cs.vu.nl. 86400  IN  A    192.31.231.165
flits.cs.vu.nl. 86400  IN  MX   1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX   2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX   3 top.cs.vu.nl.
www.cs.vu.nl.   86400  IN  CNAME star.cs.vu.nl
ftp.cs.vu.nl.   86400  IN  CNAME zephyr.cs.vu.nl

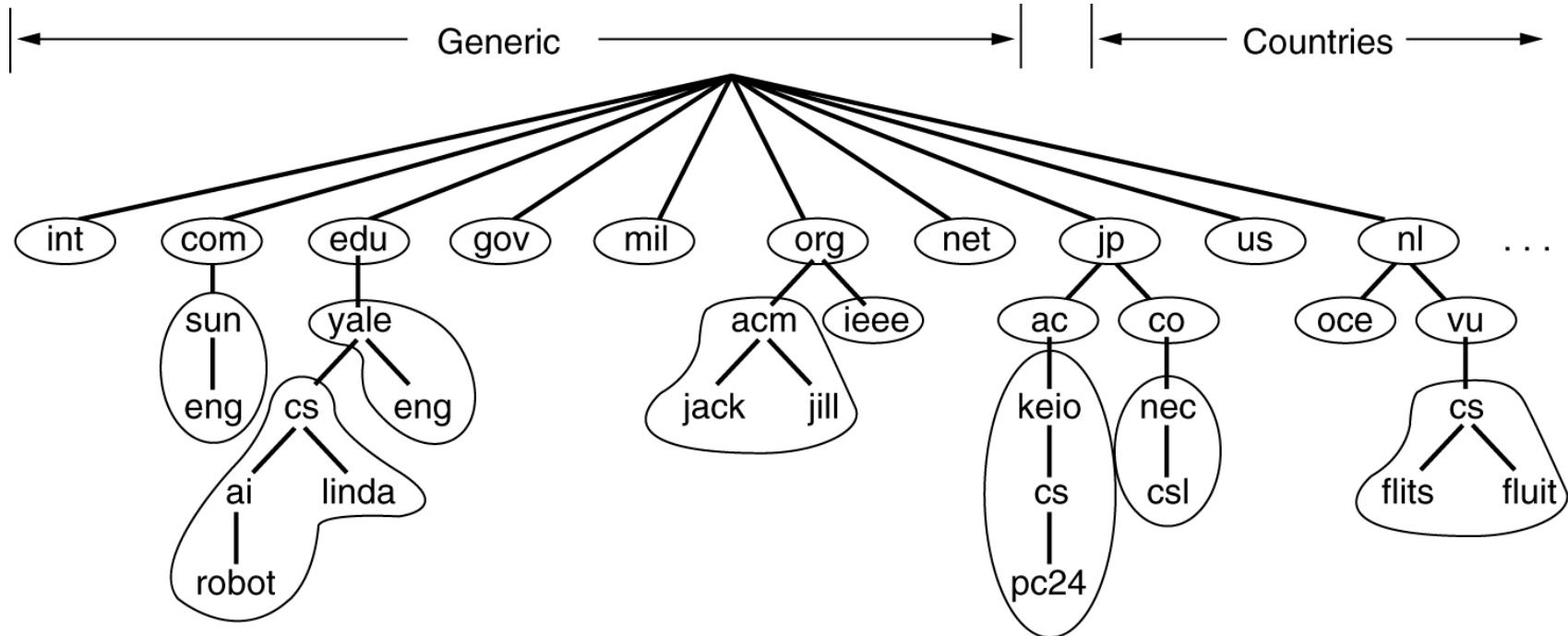
rowboat          IN  A    130.37.56.201
                  IN  MX   1 rowboat
                  IN  MX   2 zephyr
                  IN  HINFO Sun Unix

little-sister    IN  A    130.37.62.23
                  IN  HINFO Mac MacOS

laserjet         IN  A    192.31.231.216
                  IN  HINFO "HP Laserjet IISi" Proprietary
```

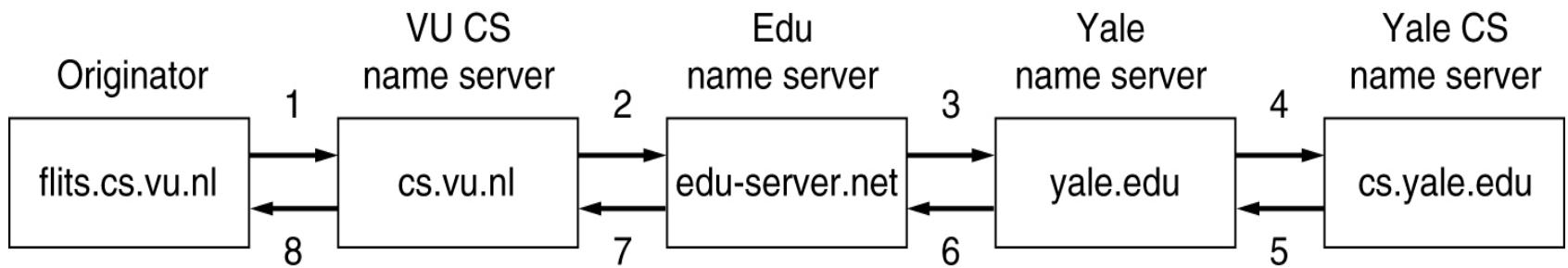
A portion of a possible DNS database for *cs.vu.nl*.

Name Servers



Part of the DNS name space showing the division into zones.

Name Servers (2)



How a resolver looks up a remote name in eight steps.

Electronic Mail

- Architecture and Services
- The User Agent
- Message Formats
- Message Transfer
- Final Delivery

Electronic Mail (2)

Smiley	Meaning	Smiley	Meaning	Smiley	Meaning
:-)	I'm happy	= :-)	Abe Lincoln	:+)	Big nose
:-('	I'm sad/angry	=):-)	Uncle Sam	:-))	Double chin
:-	I'm apathetic	*<:-)	Santa Claus	:-{})	Mustache
;:-)	I'm winking	<:-('	Dunce	#:-)	Matted hair
:-(O)	I'm yelling	(-:	Australian	8-)	Wears glasses
:-(*)	I'm vomiting	:-)X	Man with bowtie	C:-)	Large brain

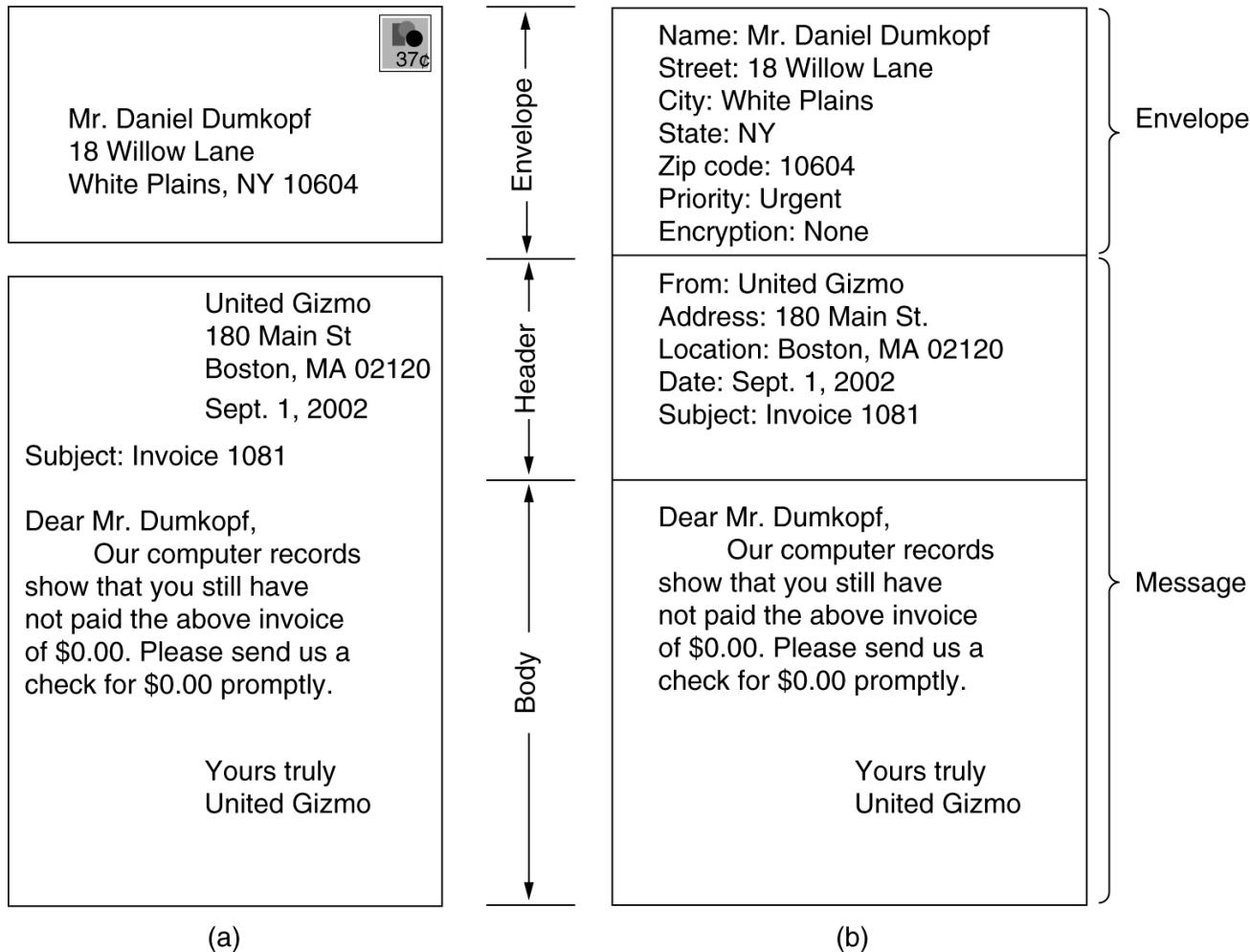
Some smileys. They will not be on the final exam :-).

Architecture and Services

Basic functions

- Composition
- Transfer
- Reporting
- Displaying
- Disposition

The User Agent



Envelopes and messages. (a) Paper mail. (b) Electronic mail.

Reading E-mail

#	Flags	Bytes	Sender	Subject
1	K	1030	asw	Changes to MINIX
2	KA	6348	trudy	Not all Trudys are nasty
3	K F	4519	Amy N. Wong	Request for information
4		1236	bal	Bioinformatics
5		104110	kaashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	guido	Our paper has been accepted
8		1204	dmr	Re: My student's visit

An example display of the contents of a mailbox.

Message Formats – RFC 822

Header	Meaning
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

RFC 822 header fields related to message transport.

Message Formats – RFC 822 (2)

Header	Meaning
Date:	The date and time the message was sent
Reply-To:	E-mail address to which replies should be sent
Message-Id:	Unique number for referencing this message later
In-Reply-To:	Message-Id of the message to which this is a reply
References:	Other relevant Message-Ids
Keywords:	User-chosen keywords
Subject:	Short summary of the message for the one-line display

Some fields used in the RFC 822 message header.

MIME – Multipurpose Internet Mail Extensions

Problems with international languages:

- Languages with accents
(French, German).
- Languages in non-Latin alphabets
(Hebrew, Russian).
- Languages without alphabets
(Chinese, Japanese).
- Messages not containing text at all
(audio or images).

MIME (2)

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

RFC 822 headers added by MIME.

MIME (3)

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

The MIME types and subtypes defined in RFC 2045.

MIME (4)

From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear Carolyn
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
access-type="anon-ftp";
site="bicycle.abcd.com";
directory="pub";
name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--

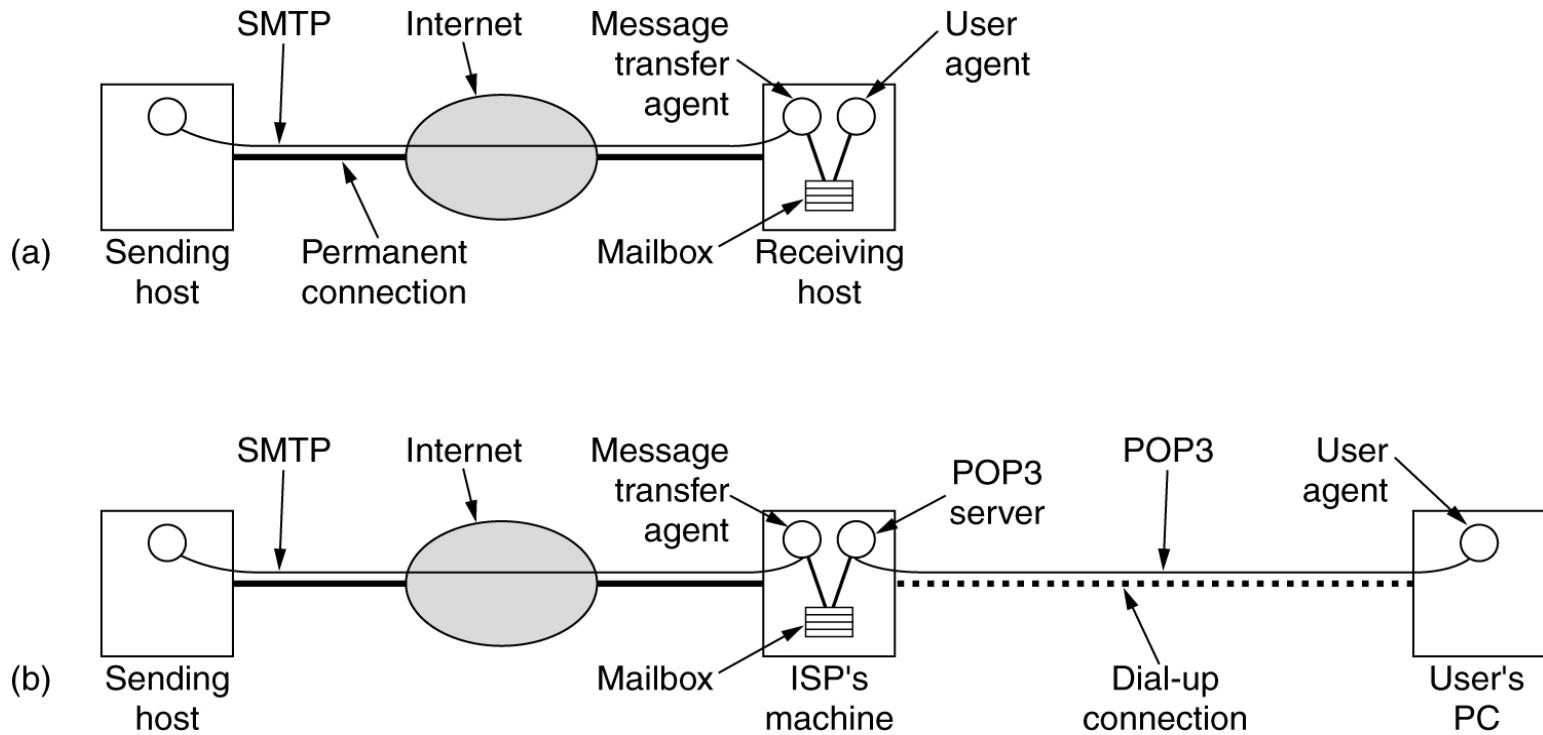
A multipart message containing enriched and audio alternatives.

Message Transfer

Transferring a message
from
elinore@abc.com to
carolyn@xyz.com.

```
S: 220 xyz.com SMTP service ready
C: HELO abcd.com
    S: 250 xyz.com says hello to abcd.com
C: MAIL FROM: <elinor@abcd.com>
    S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
    S: 250 recipient ok
C: DATA
    S: 354 Send mail; end with "." on a line by itself
C: From: elinor@abcd.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abcd.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/enriched
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:     access-type="anon-ftp";
C:     site="bicycle.abcd.com";
C:     directory="pub";
C:     name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
    S: 250 message accepted
C: QUIT
    S: 221 xyz.com closing connection
```

Final Delivery



(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. **(b)** Reading e-mail when the receiver has a dial-up connection to an ISP.

POP3

```
S: +OK POP3 server ready
C: USER carolyn
                S: +OK
C: PASS vegetables
                S: +OK login successful
C: LIST
                S: 1 2505
                S: 2 14302
                S: 3 8122
                S: .
C: RETR 1
                S: (sends message 1)
C: DELE 1
C: RETR 2
                S: (sends message 2)
C: DELE 2
C: RETR 3
                S: (sends message 3)
C: DELE 3
C: QUIT
                S: +OK POP3 server disconnecting
```

Using POP3 to fetch three messages.

IMAP

Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

A comparison of POP3 and IMAP.

The World Wide Web

- Architectural Overview
- Static Web Documents
- Dynamic Web Documents
- HTTP – The HyperText Transfer Protocol
- Performance Enhancements
- The Wireless Web

Architectural Overview

WELCOME TO THE UNIVERSITY OF EAST PODUNK'S WWW HOME PAGE

- Campus Information
 - [Admissions information](#)
 - [Campus map](#)
 - [Directions to campus](#)
 - [The UEP student body](#)
- Academic Departments
 - [Department of Animal Psychology](#)
 - [Department of Alternative Studies](#)
 - [Department of Microbiotic Cooking](#)
 - [Department of Nontraditional Studies](#)
 - [Department of Traditional Studies](#)

Webmaster@eastpodunk.edu

(a)

THE DEPARTMENT OF ANIMAL PSYCHOLOGY

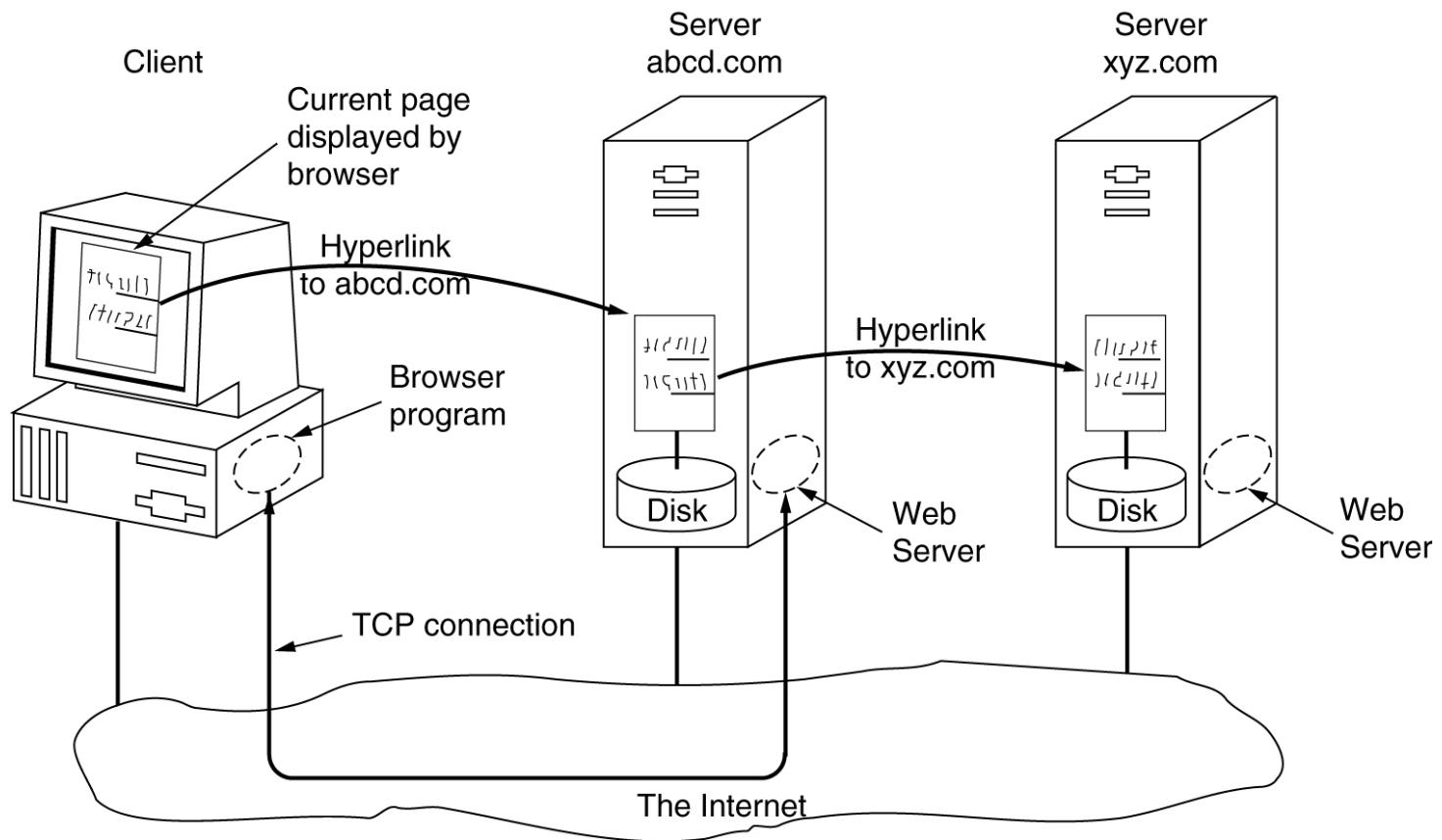
- [Information for prospective majors](#)
- Personnel
 - [Faculty members](#)
 - [Graduate students](#)
 - [Nonacademic staff](#)
- [Research Projects](#)
- [Positions available](#)
- Our most popular courses
 - [Dealing with herbivores](#)
 - [Horse management](#)
 - [Negotiating with your pet](#)
 - [User-friendly doghouse construction](#)
- [Full list of courses](#)

Webmaster@animalpsyc.eastpodunk.edu

(b)

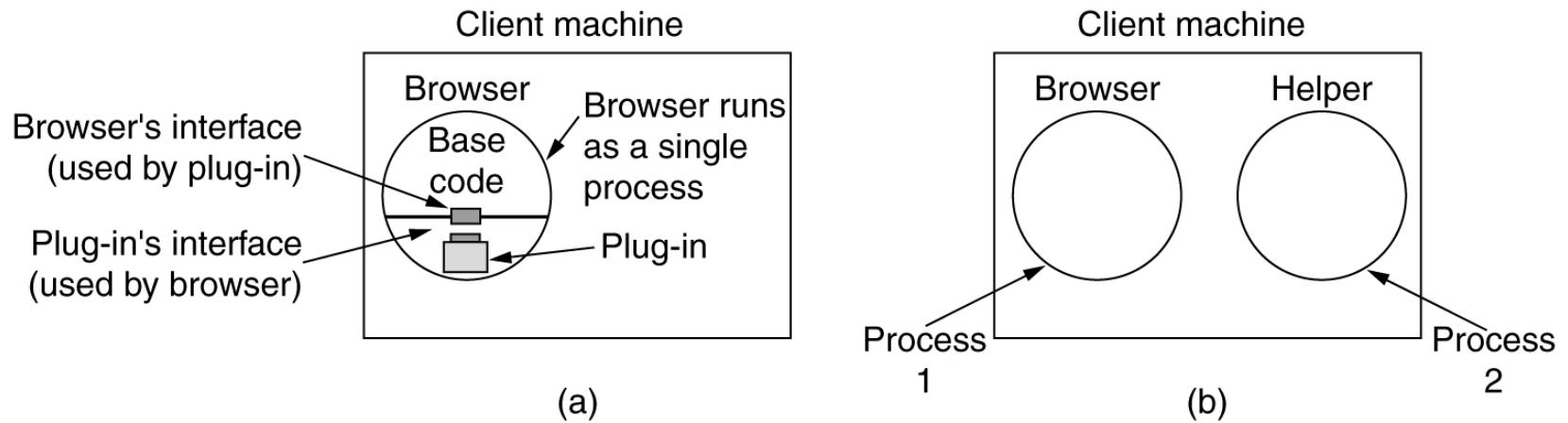
(a) A Web page (b) The page reached by clicking on [Department of Animal Psychology](#).

Architectural Overview (2)



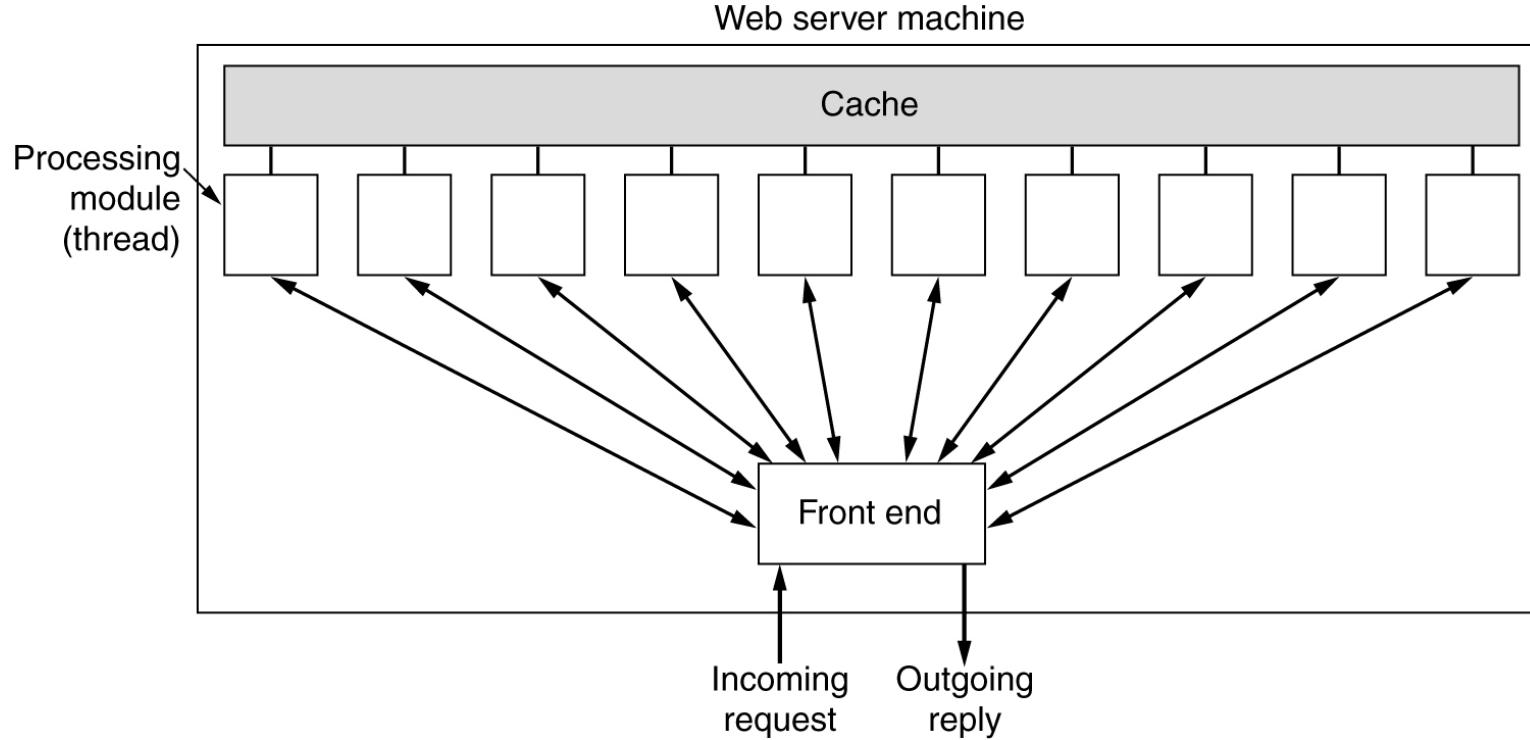
The parts of the Web model.

The Client Side



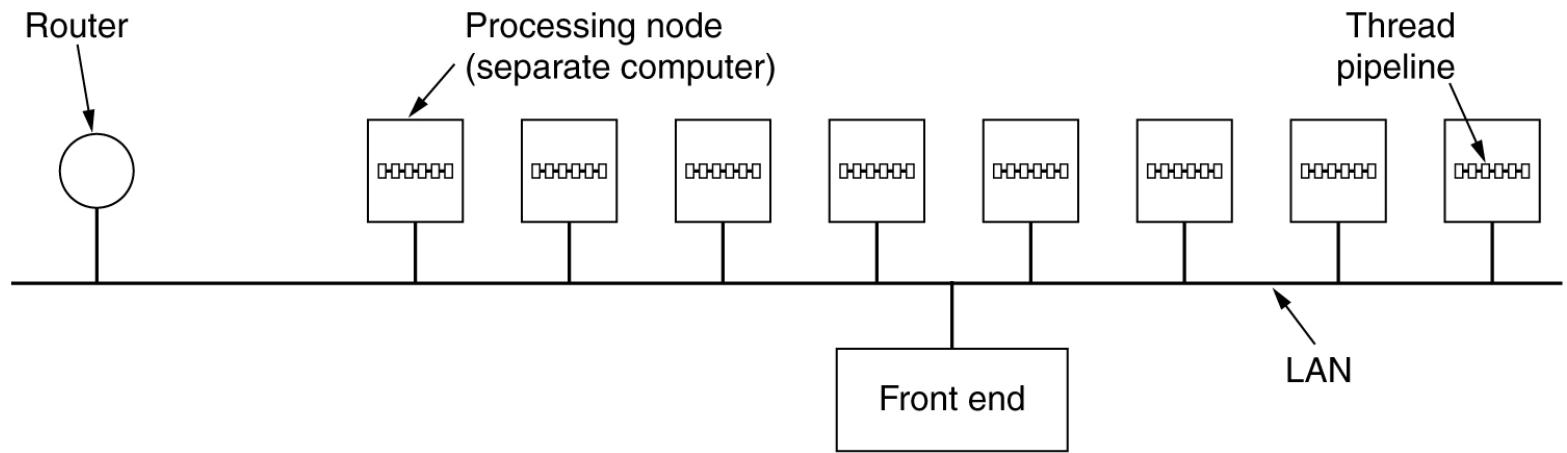
(a) A browser plug-in. (b) A helper application.

The Server Side



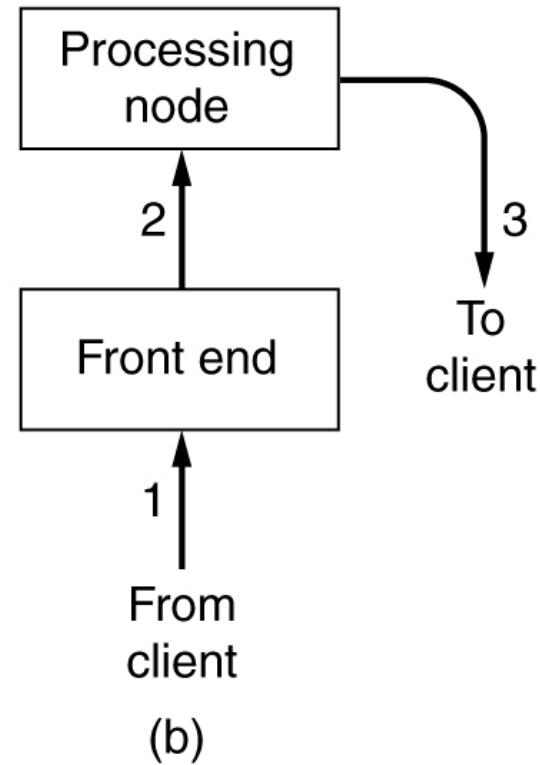
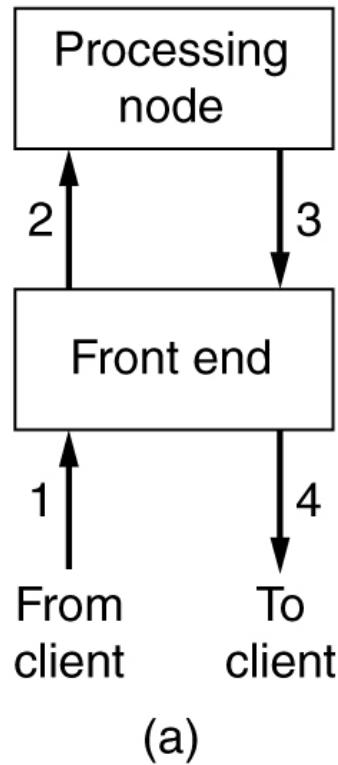
A multithreaded Web server with a front end and processing modules.

The Server Side (2)



A server farm.

The Server Side (3)



- (a) Normal request-reply message sequence.
- (b) Sequence when TCP handoff is used.

URLs – Uniform Resource Locators

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
news	Newsgroup	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending e-mail	mailto:JohnUser@acm.org
telnet	Remote login	telnet://www.w3.org:80

Some common URLs.

Statelessness and Cookies

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Yes
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	No
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	No
sneaky.com	/	UserID=3627239101	31-12-12 23:59	No

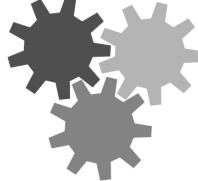
Some examples of cookies.

HTML – HyperText Markup Language

```
<html>
<head><title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p> Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Big widgets</a>
  <li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
  <li> By telephone: 1-800-WIDGETS
  <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)

Welcome to AWI's Home Page



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope you will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

Product Information

- [Big widgets](http://widget.com/products/big)
- [Little widgets](http://widget.com/products/little)

Telephone numbers

- 1-800-WIDGETS
- 1-415-765-4321

(b)

(a) The HTML for a sample Web page. (b) The formatted page.

HTML (2)

Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h _n > ... </h _n >	Delimits a level <i>n</i> heading
 ... 	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
 ... 	Brackets an unordered (bulleted) list
 ... 	Brackets a numbered list
	Starts a list item (there is no)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
	Displays an image here
 ... 	Defines a hyperlink

A selection of common HTML tags.
some can have additional parameters.

Forms

- (a) An HTML table.
- (b) A possible rendition of this table.

```
<html>
<head> <title> A sample page with a table </title> </head>
<body>
<table border=1 rules=all>
<caption> Some Differences between HTML Versions </caption>
<col align=left>
<col align=center>
<col align=center>
<col align=center>
<col align=center>
<tr> <th>Item <th>HTML 1.0 <th>HTML 2.0 <th>HTML 3.0 <th>HTML 4.0 </tr>
<tr> <th> Hyperlinks <td> x <td> x <td> x <td> x </tr>
<tr> <th> Images <td> x <td> x <td> x <td> x </tr>
<tr> <th> Lists <td> x <td> x <td> x <td> x </tr>
<tr> <th> Active Maps and Images <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Forms <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Equations <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Toolbars <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Tables <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Accessibility features <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Object embedding <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Scripting <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
</table>
</body>
</html>
```

(a)

Some Differences between HTML Versions

Item	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0
Hyperlinks	x	x	x	x
Images	x	x	x	x
Lists	x	x	x	x
Active Maps and Images		x	x	x
Forms		x	x	x
Equations			x	x
Toolbars			x	x
Tables			x	x
Accessibility features				x
Object embedding				x
Scripting				x

(b)

Forms (2)

- (a) The HTML for an order form.
- (b) The formatted page.

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

(a)

Widget Order Form

Name

Street address

City State Country

Credit card # Expires M/C Visa

Widget size Big Little Ship by express courier

Thank you for ordering an AWI widget, the best widget money can buy!

(b)

Forms (3)

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&  
state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&  
product=cheap&express=on
```

A possible response from the browser to the server with information filled in by the user.

XML and XSL

A simple Web page
in XML.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="b5.xsl"?>
<book_list>
  <book>
    <title> Computer Networks, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2003 </year>
  </book>
  <book>
    <title> Modern Operating Systems, 2/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2001 </year>
  </book>
  <book>
    <title> Structured Computer Organization, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 1999 </year>
  </book>
</book_list>
```

XML and XSL (2)

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">

<html>
<body>

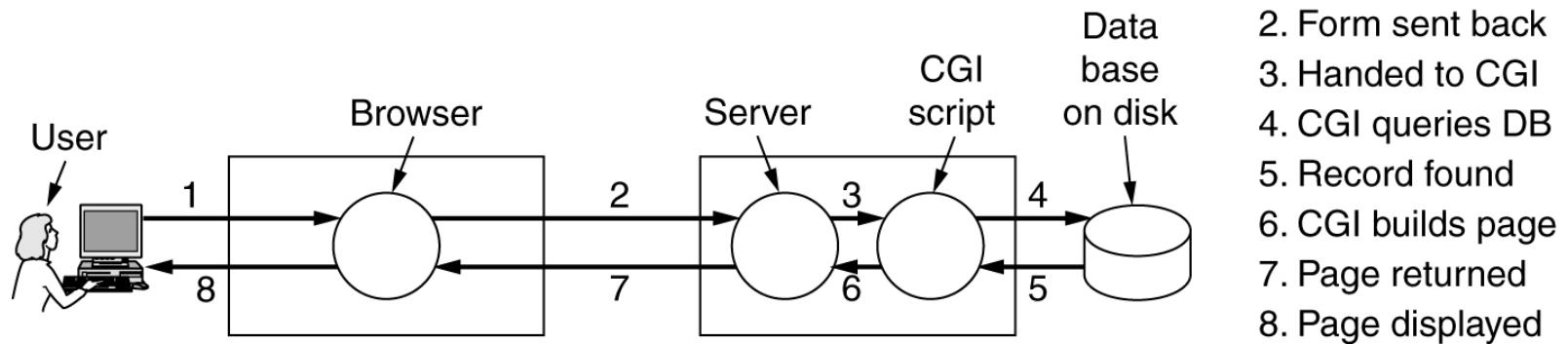
<table border="2">
<tr>
    <th> Title</th>
    <th> Author</th>
    <th> Year </th>
</tr>

<xsl:for-each select="book_list/book">
<tr>
    <td> <xsl:value-of select="title"/> </td>
    <td> <xsl:value-of select="author"/> </td>
    <td> <xsl:value-of select="year"/> </td>
</tr>
</xsl:for-each>
</table>

</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

A style sheet in
XSL.

Dynamic Web Documents



Steps in processing the information from an HTML form.

Dynamic Web Documents (2)

```
<html>
<body>
<h2> This is what I know about you </h2>
<?php echo $HTTP_USER_AGENT ?>
</body>
</html>
```

A sample HTML page with embedded PHP.

Dynamic Web Documents (3)

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

(b)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 25
</body>
</html>
```

(c)

- (a) A Web page containing a form. (b) A PHP script for handling the output of the form. (c) Output from the PHP script when the inputs are "Barbara" and 24 respectively.

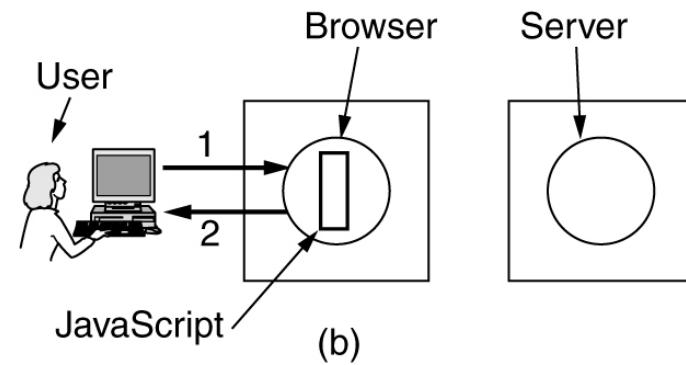
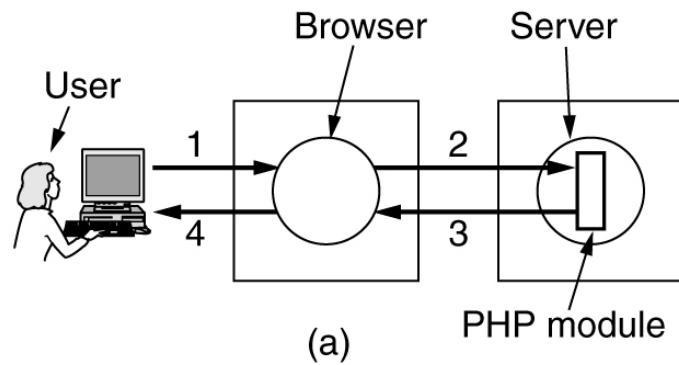
Client-Side Dynamic Web Page Generation

```
<head>
<script language="javascript" type="text/javascript">
function response(test form) {
    var person = test form.name.value;
    var years = eval(test form.age.value) + 1;
    document.open();
    document.writeln("<html> <body>");
    document.writeln("Hello " + person + ".<br>");
    document.writeln("Prediction: next year you will be " + years + ".");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>

<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>
```

Use of JavaScript
for processing a
form.

Client-Side Dynamic Web Page Generation (2)



(a) Server-side scripting with PHP.

(b) Client-side scripting with JavaScript.

Client-Side Dynamic Web Page Generation (3)

```
<html>
<head>
<script language="javascript" type="text/javascript">

function response(test_form) {
    function factorial(n) {if (n == 0) return 1; else return n * factorial(n - 1);}
    var r = eval(test_form.number.value);      // r = typed in argument
    document.myform.mytext.value = "Here are the results.\n";
    for (var i = 1; i <= r; i++)           // print one line from 1 to r
        document.myform.mytext.value += (i + "!" + factorial(i) + "\n");
}
</script>
</head>

<body>
<form name="myform">
Please enter a number: <input type="text" name="number">
<input type="button" value="compute table of factorials" onclick="response(this.form)">
<p>
<textarea name="mytext" rows=25 cols=50> </textarea>
</form>
</body>
</html>
```

A JavaScript program for computing and printing factorials.

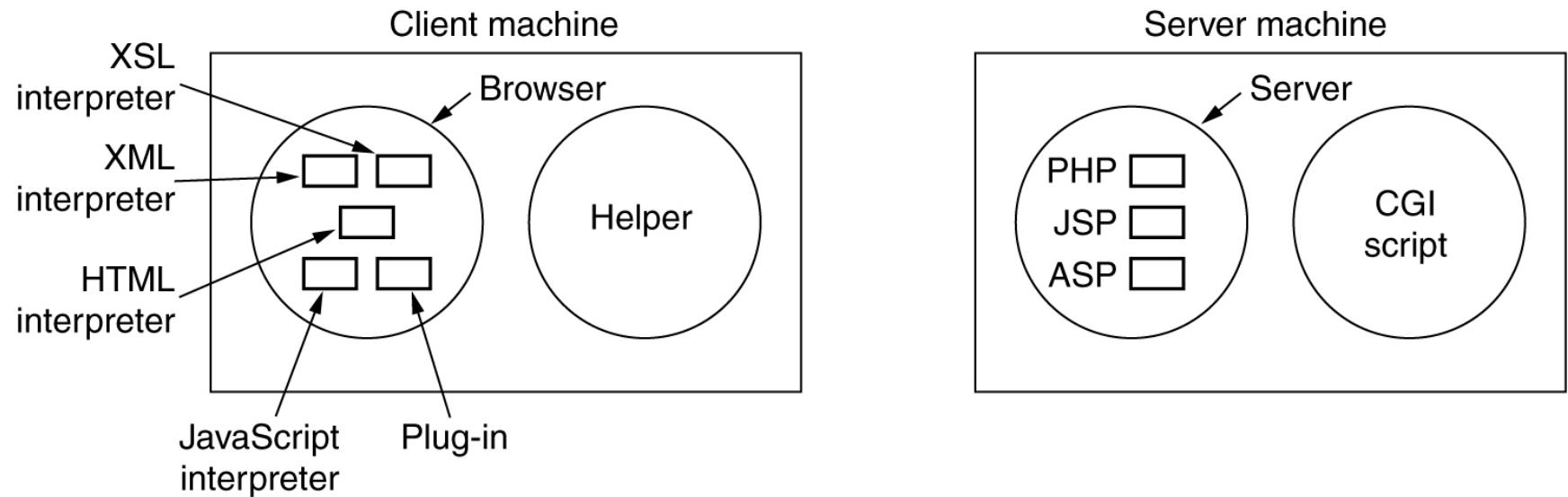
Client-Side Dynamic Web Page Generation (4)

```
<html>
<head>
<script language="javascript" type="text/javascript">
if (!document.myurl) document.myurl = new Array();
document.myurl[0] = "http://www.cs.vu.nl/~ast/im/kitten.jpg";
document.myurl[1] = "http://www.cs.vu.nl/~ast/im/puppy.jpg";
document.myurl[2] = "http://www.cs.vu.nl/~ast/im/bunny.jpg";
function pop(m) {
    var urx = "http://www.cs.vu.nl/~ast/im/cat.jpg";
    popupwin = window.open(document.myurl[m],"mywind","width=250,height=250");
}
</script>
</head>

<body>
<p> <a href="#" onMouseover="pop(0); return false;" > Kitten </a> </p>
<p> <a href="#" onMouseover="pop(1); return false;" > Puppy </a> </p>
<p> <a href="#" onMouseover="pop(2); return false;" > Bunny </a> </p>
</body>
</html>
```

An interactive Web page that responds to mouse movement.

Client-Side Dynamic Web Page Generation (5)



The various ways to generate and display content.

HTTP Methods

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

The built-in HTTP request methods.

HTTP Methods (2)

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

The status code response groups.

HTTP Message Headers

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Some HTTP message headers.

Example HTTP Usage

The start of the output of
www.ietf.org/rfc.html.

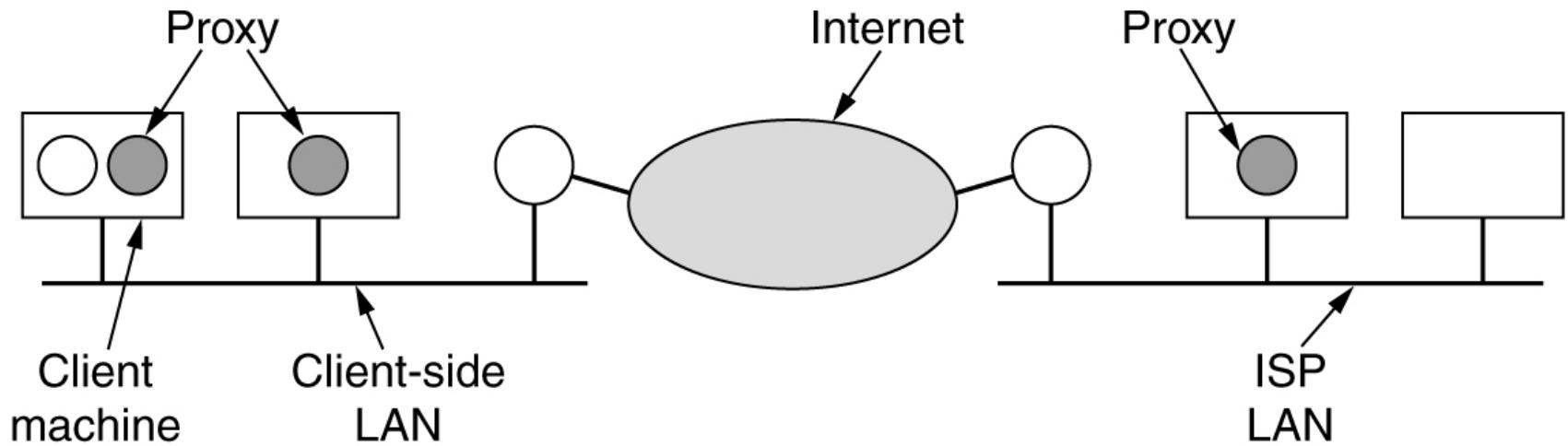
```
Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is '^].
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug
```

```
<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
    var x = document.form1.number.value
    if (x.length == 1) {x = "000" + x }
    if (x.length == 2) {x = "00" + x }
    if (x.length == 3) {x = "0" + x }
    document.form1.action = "/rfc/rfc" + x + ".txt"
    document.form1.submit
}
</script>

</head>
```

Caching



Hierarchical caching with three proxies.

Content Delivery Networks

```
<html>
<head> <title> Furry Video </title> </head>
<body>
<h1> Furry Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="bears.mpg"> Bears Today </a> <br>
<a href="bunnies.mpg"> Funny Bunnies </a> <br>
<a href="mice.mpg"> Nice Mice </a> <br>
</body>
</html>
```

(a)

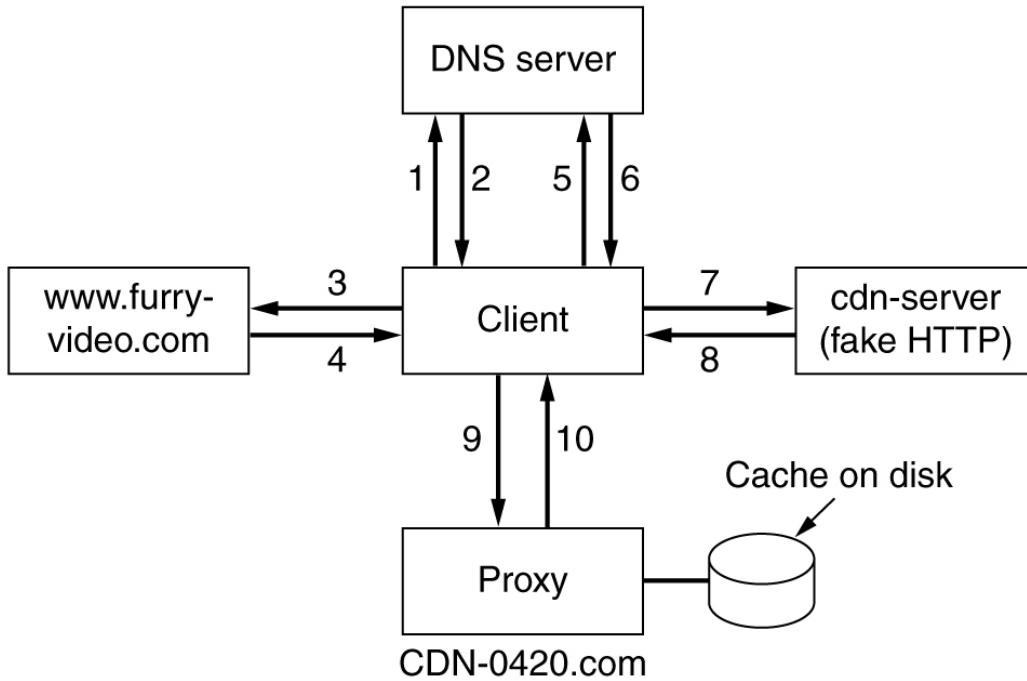
```
<html>
<head> <title> Furry Video </title> </head>
<body>
<h1> Furry Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="http://cdn-server.com/furryvideo/bears.mpg"> Bears Today </a> <br>
<a href="http://cdn-server.com/furryvideo/bunnies.mpg"> Funny Bunnies </a> <br>
<a href="http://cdn-server.com/furryvideo/mice.mpg"> Nice Mice </a> <br>
</body>
</html>
```

(b)

(a) Original Web page. (b) Same page after transformation.

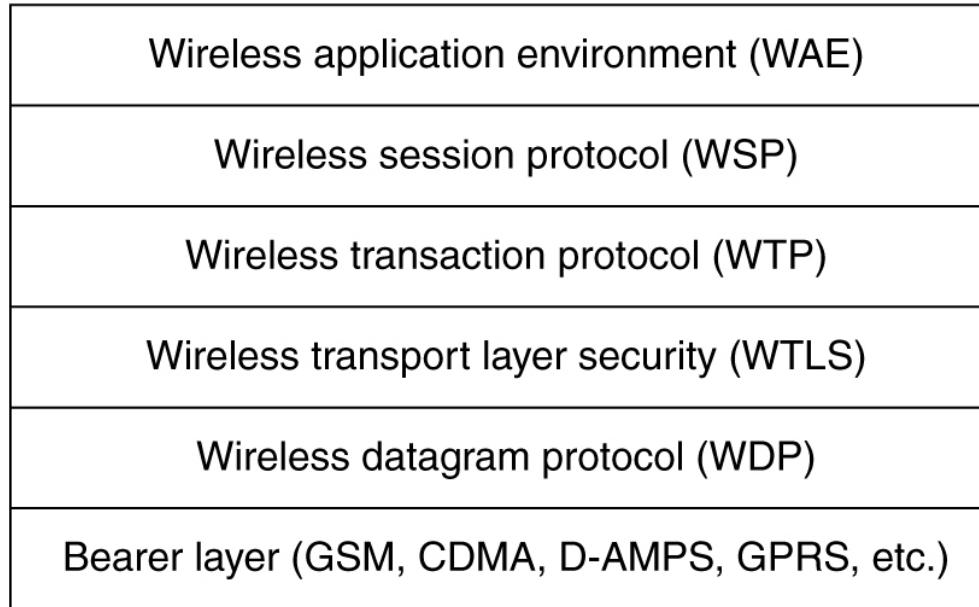
The Wireless Web



1. Look up www.furryvideo.com
2. Furry's IP address returned
3. Request HTML page from Furry
4. HTML page returned
5. After click, look up cdn-server.com
6. IP address of cdn-server returned
7. Ask cdn-server for bears.mpg
8. Client told to redirect to CDN-0420.com
9. Request bears.mpg
10. Cached file bears.mpg returned

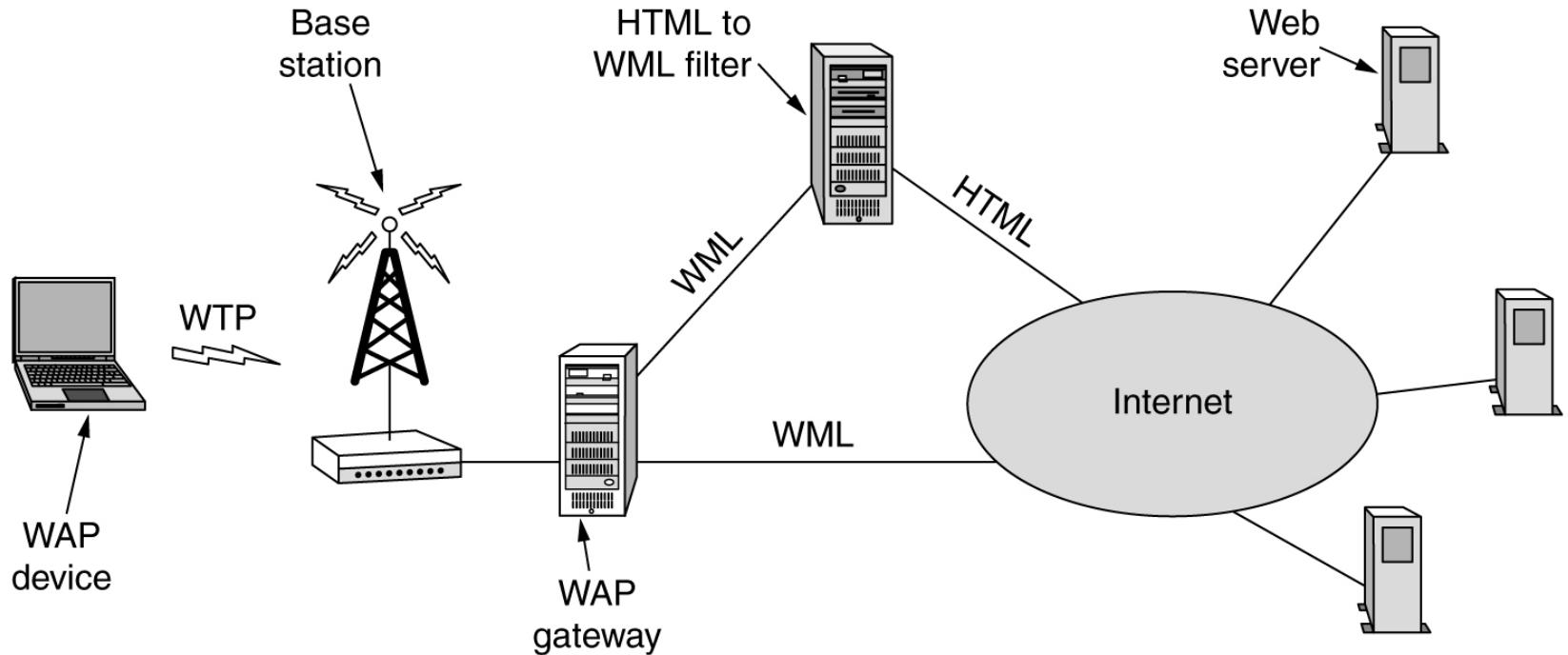
Steps in looking up a URL when a CDN is used.

WAP – The Wireless Application Protocol



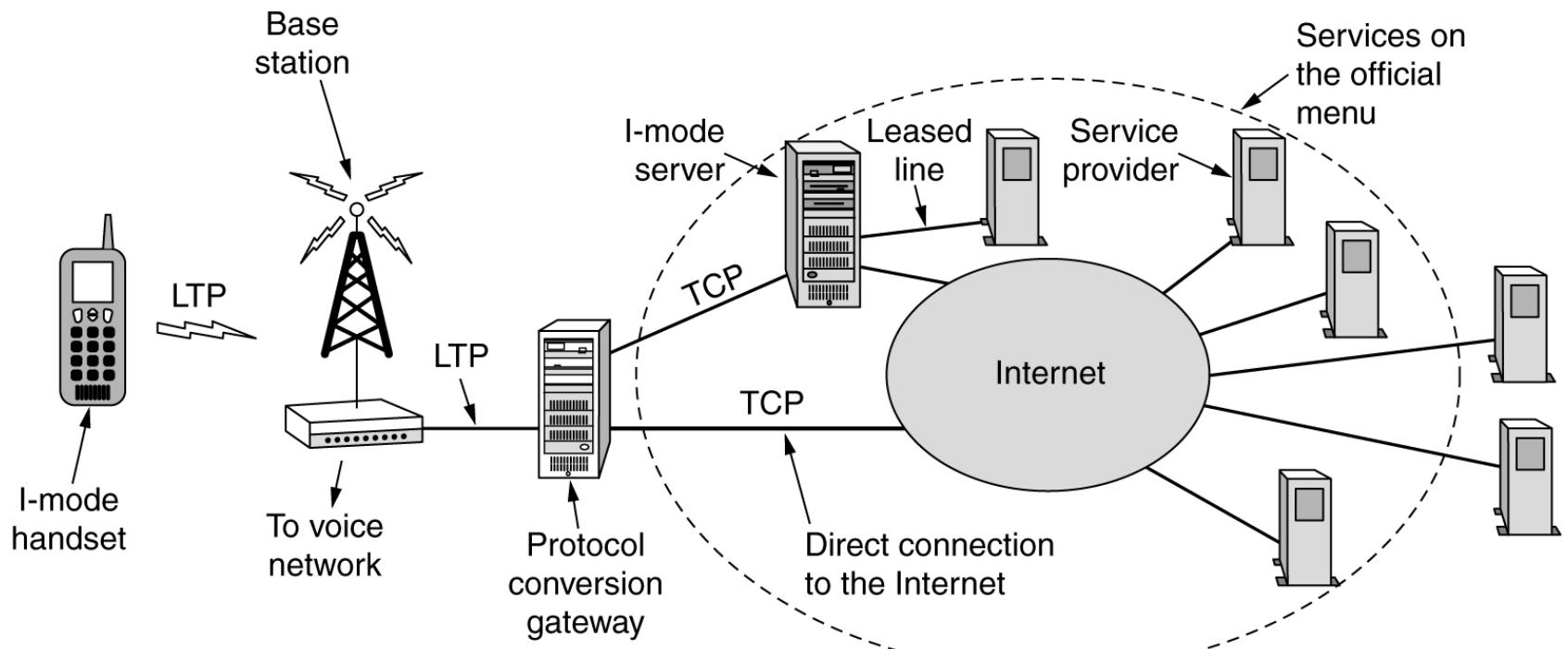
The WAP protocol stack.

WAP (2)



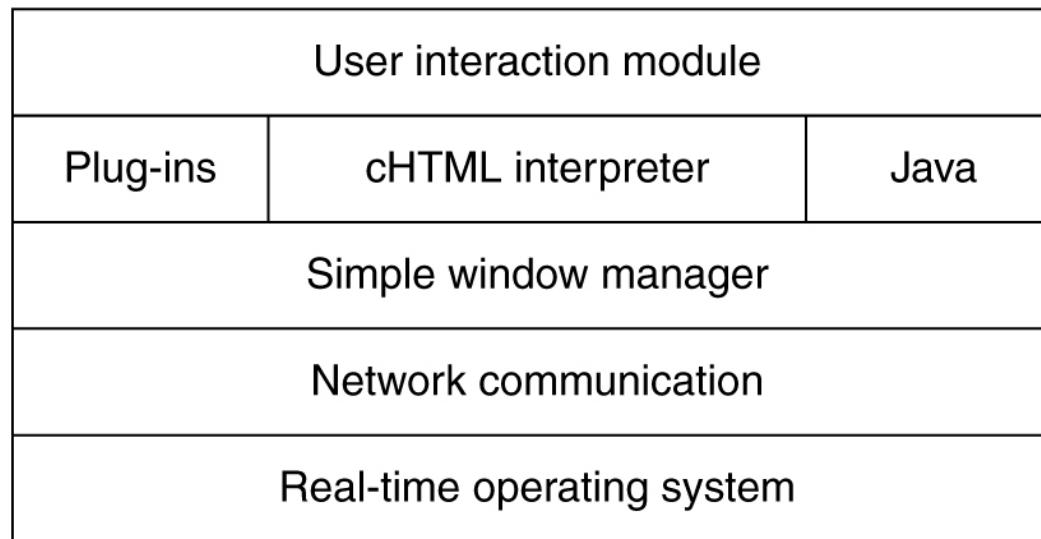
The WAP architecture.

I-Mode



Structure of the i-mode data network showing the transport protocols.

I-Mode (2)



Structure of the i-mode software.

I-Mode (3)

The time has come the walrus said to talk of many things. Of shoes and ships and sealing wax of c

(a)

The time has come the walrus said to talk of many things. Of shoes and ships and sealing wax

(b)

Lewis Carroll meets a 16 x 16 screen.

I-Mode (4)

```
<html>
<body>
<h1> Select an option </h1>
<a href="messages.chtml" accesskey="1"> Check voicemail </a> <br>
<a href="mail.chtml" accesskey="2"> Check e-mail </a> <br>
<a href="games.chtml" accesskey="3"> Play a game </a>
</body>
</html>
```

An example of cHTML file.

Second-Generation Wireless Web

Feature	WAP	I-mode
What it is	Protocol stack	Service
Device	Handset, PDA, notebook	Handset
Access	Dial up	Always on
Underlying network	Circuit-switched	Two: circuit + packet
Data rate	9600 bps	9600 bps
Screen	Monochrome	Color
Markup language	WML (XML application)	cHTML
Scripting language	WMLscript	None
Usage charges	Per minute	Per packet
Pay for shopping	Credit card	Phone bill
Pictograms	No	Yes
Standardization	WAP forum open standard	NTT DoCoMo proprietary
Where used	Europe, Japan	Japan
Typical user	Businessman	Young woman

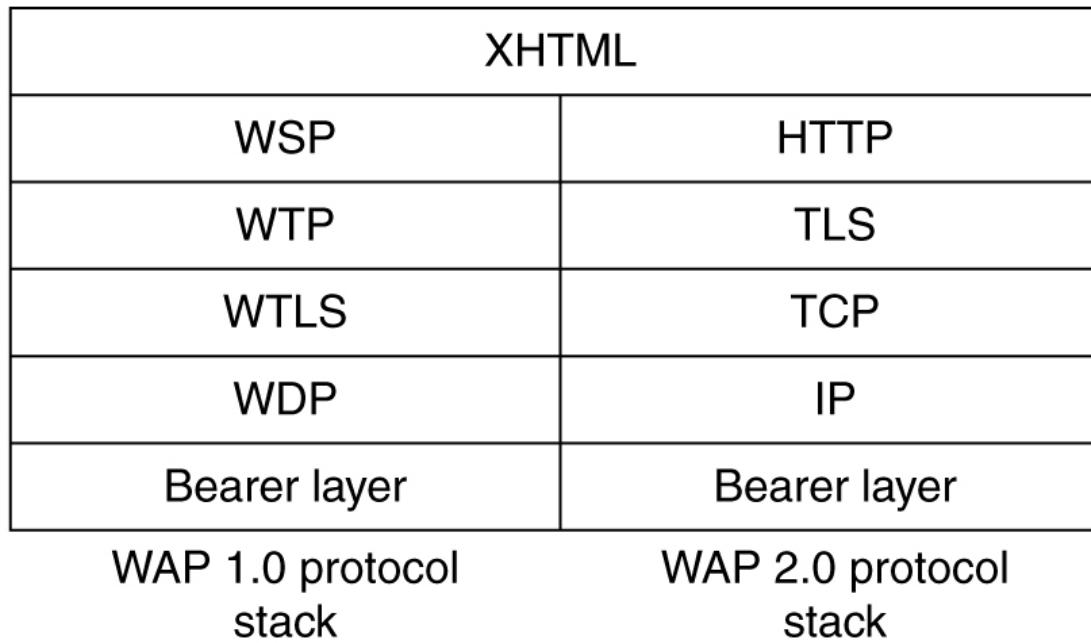
A comparison of first-generation WAP and i-mode.

Second-Generation Wireless Web (2)

New features of WAP 2.0.

- Push model as well as pull model.
- Support for integrating telephony into apps.
- Multimedia messaging.
- Inclusion of 264 pictograms.
- Interface to a storage device.
- Support for plug-ins in the browser.

Second-Generation Wireless Web (3)



WAP 2.0 supports two protocol stacks.

Second-Generation Wireless Web (4)

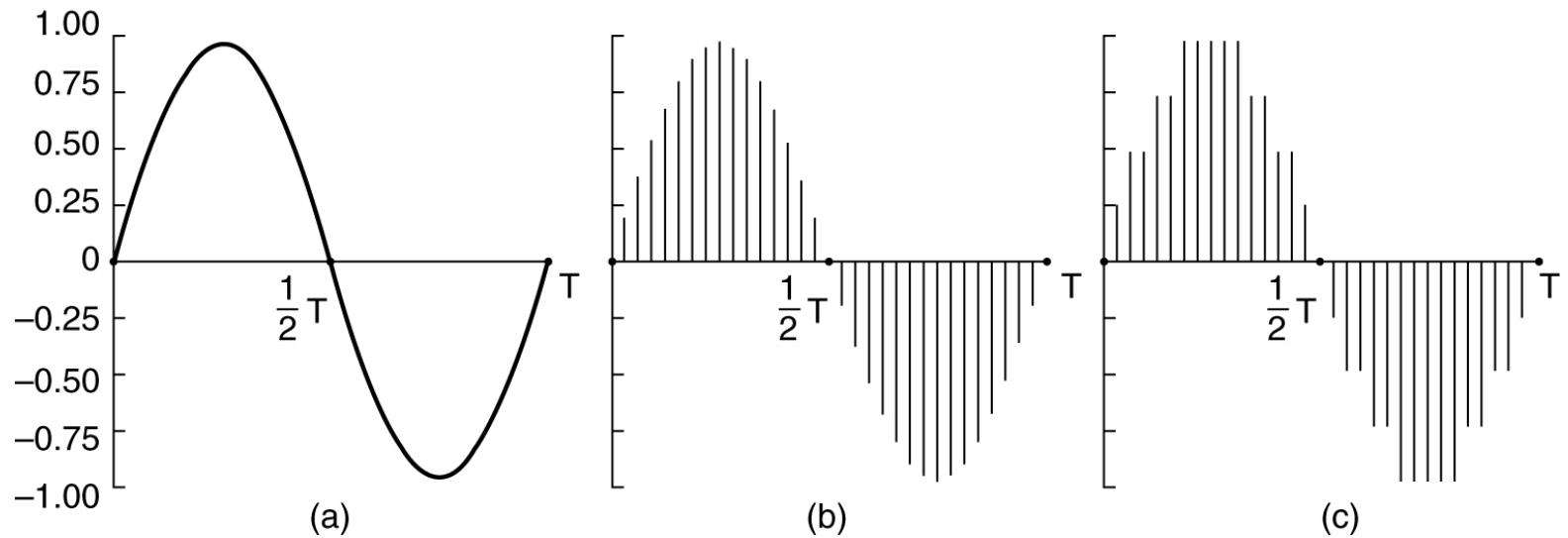
Module	Req.?	Function	Example tags
Structure	yes	Doc. structure	body, head, html, title
Text	yes	Information	br, code, dfn, em, hn, kbd, p, strong
Hypertext	yes	Hyperlinks	a
List	yes	Itemized lists	dl, dt, dd, ol, ul, li
Forms	No	Fill-in forms	form, input, label, option, textarea
Tables	No	Rectangular tables	caption, table, td, th, tr
Image	No	Pictures	img
Object	No	Applets, maps, etc.	object, param
Meta-information	No	Extra info	meta
Link	No	Similar to <a>	link
Base	No	URL starting point	base

The XHTML Basic modules and tags.

Multimedia

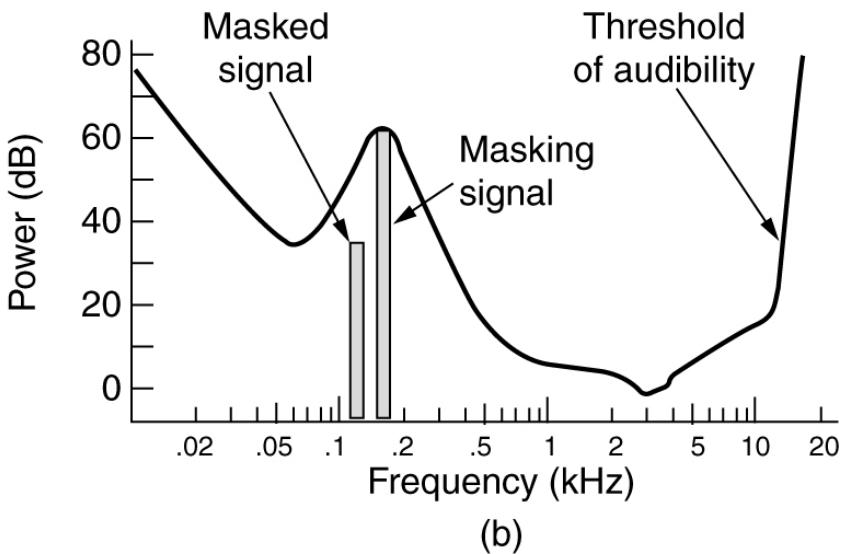
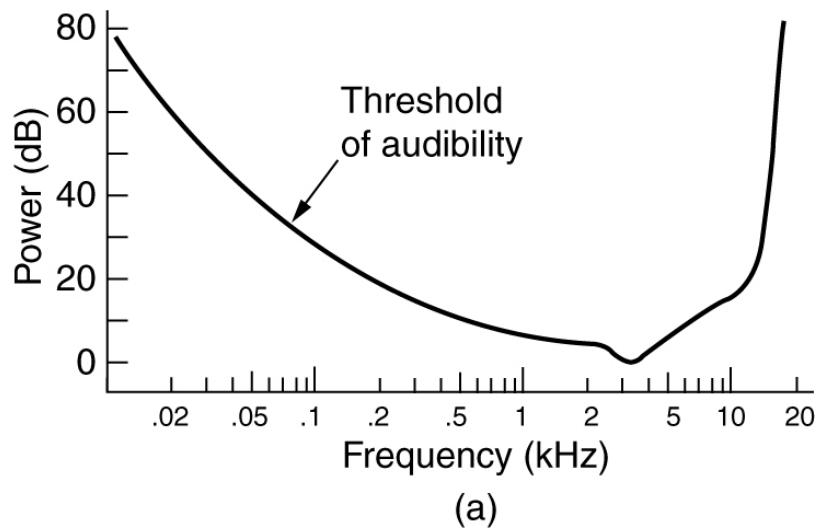
- Introduction to Audio
- Audio Compression
- Streaming Audio
- Internet Radio
- Voice over IP
- Introduction to Video
- Video Compression
- Video on Demand
- The MBone – The Multicast Backbone

Introduction to Audio



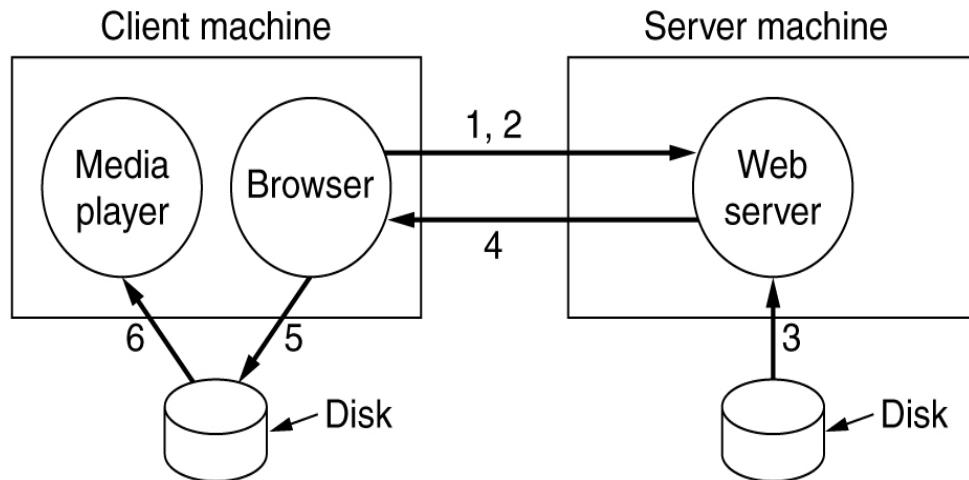
- (a) A sine wave. (b) Sampling the sine wave.
(c) Quantizing the samples to 4 bits.

Audio Compression



- (a) The threshold of audibility as a function of frequency.
- (b) The masking effect.

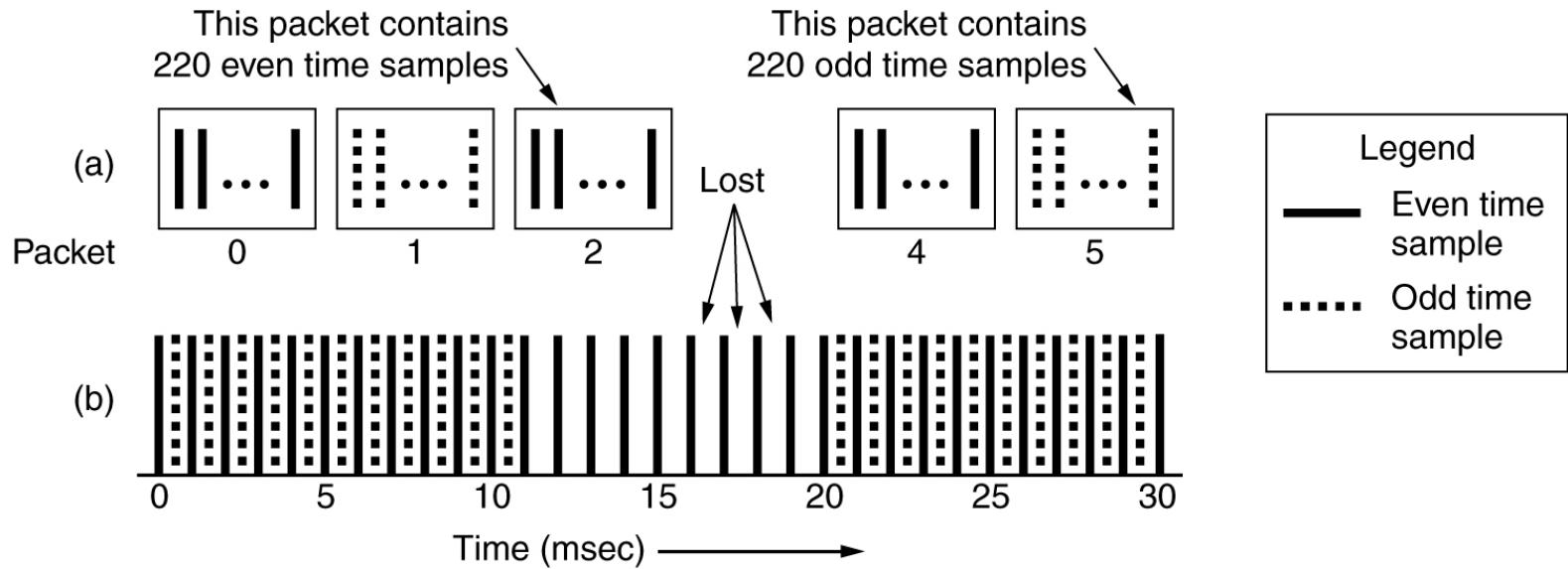
Streaming Audio



1. Establish TCP connection
2. Send HTTP GET request
3. Server gets file from disk
4. File sent back
5. Browser writes file to disk
6. Media player fetches file block by block and plays it

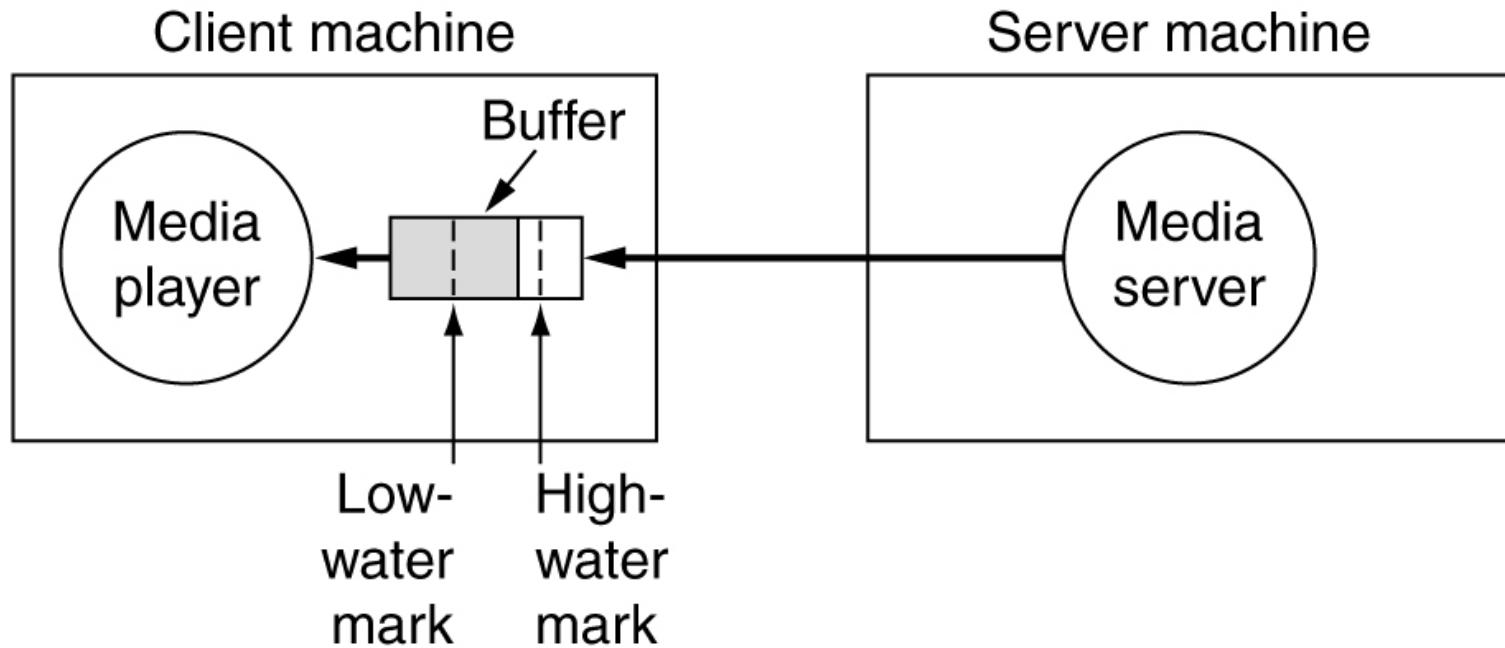
A straightforward way to implement clickable music on a Web page.

Streaming Audio (2)



When packets carry alternate samples, the loss of a packet reduces the temporal resolution rather than creating a gap in time.

Streaming Audio (3)



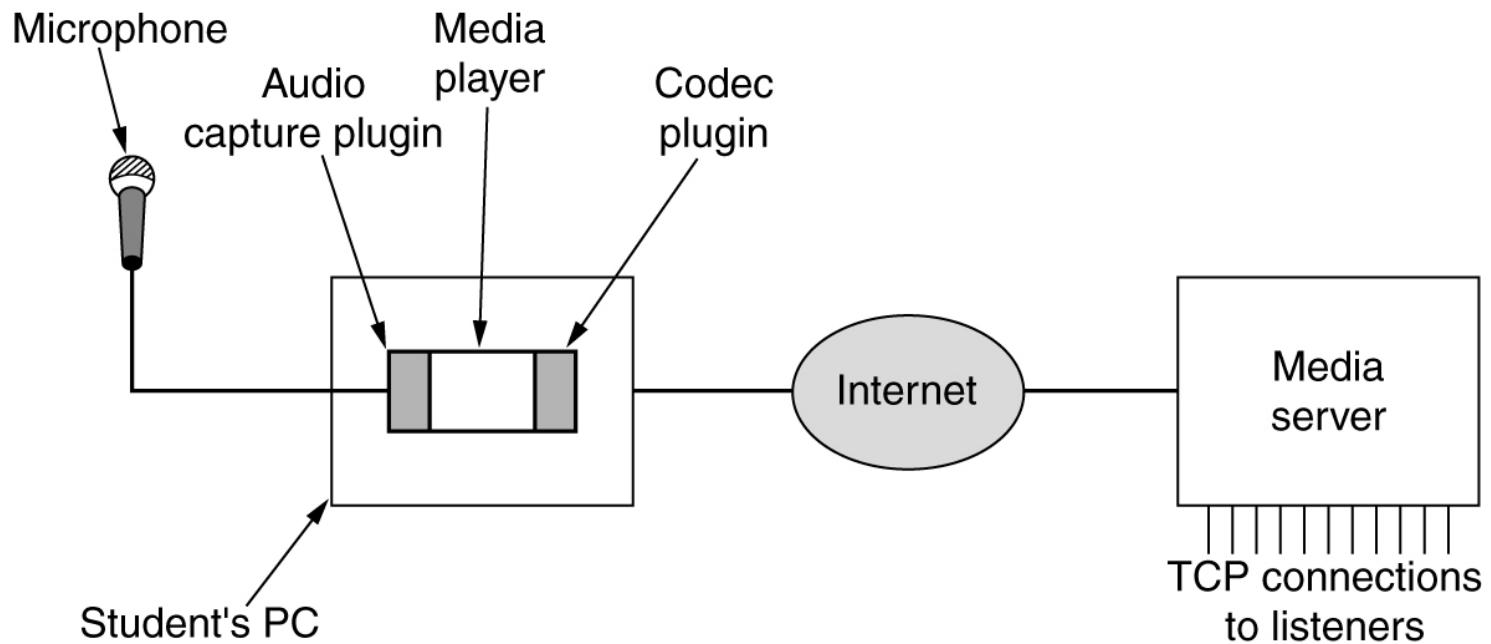
The media player buffers input from the media server and plays from the buffer rather than directly from the network.

Streaming Audio (4)

Command	Server action
DESCRIBE	List media parameters
SETUP	Establish a logical channel between the player and the server
PLAY	Start sending data to the client
RECORD	Start accepting data from the client
PAUSE	Temporarily stop sending data
TEARDOWN	Release the logical channel

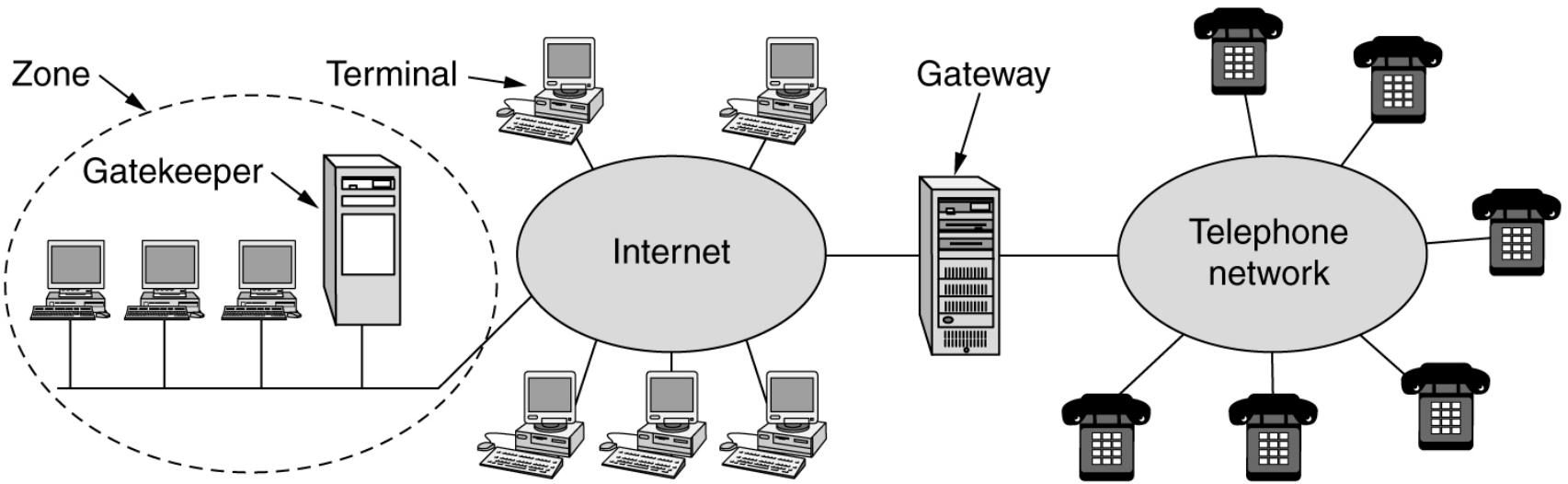
RTSP commands from the player to the server.

Internet Radio



A student radio station.

Voice over IP



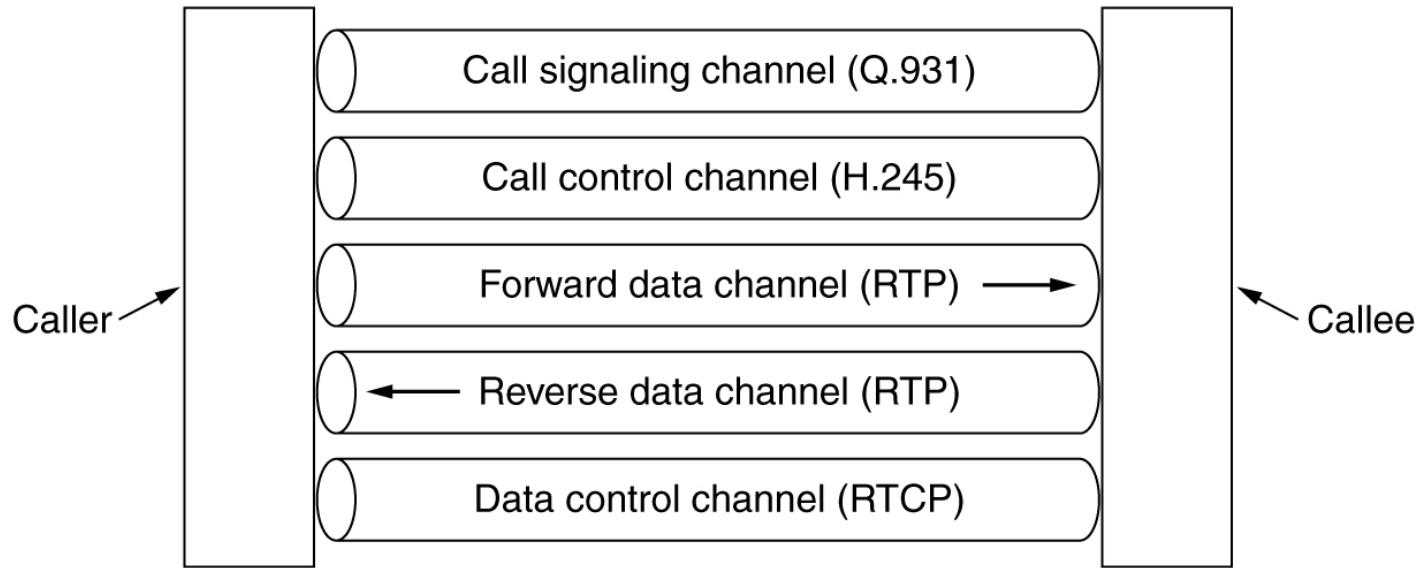
The H323 architectural model for Internet telephony.

Voice over IP (2)

Speech	Control							
G.7xx	RTCP	H.225 (RAS)	Q.931 (Call signaling)	H.245 (Call control)				
RTP								
UDP		TCP						
IP								
Data link protocol								
Physical layer protocol								

The H323 protocol stack.

Voice over IP (3)



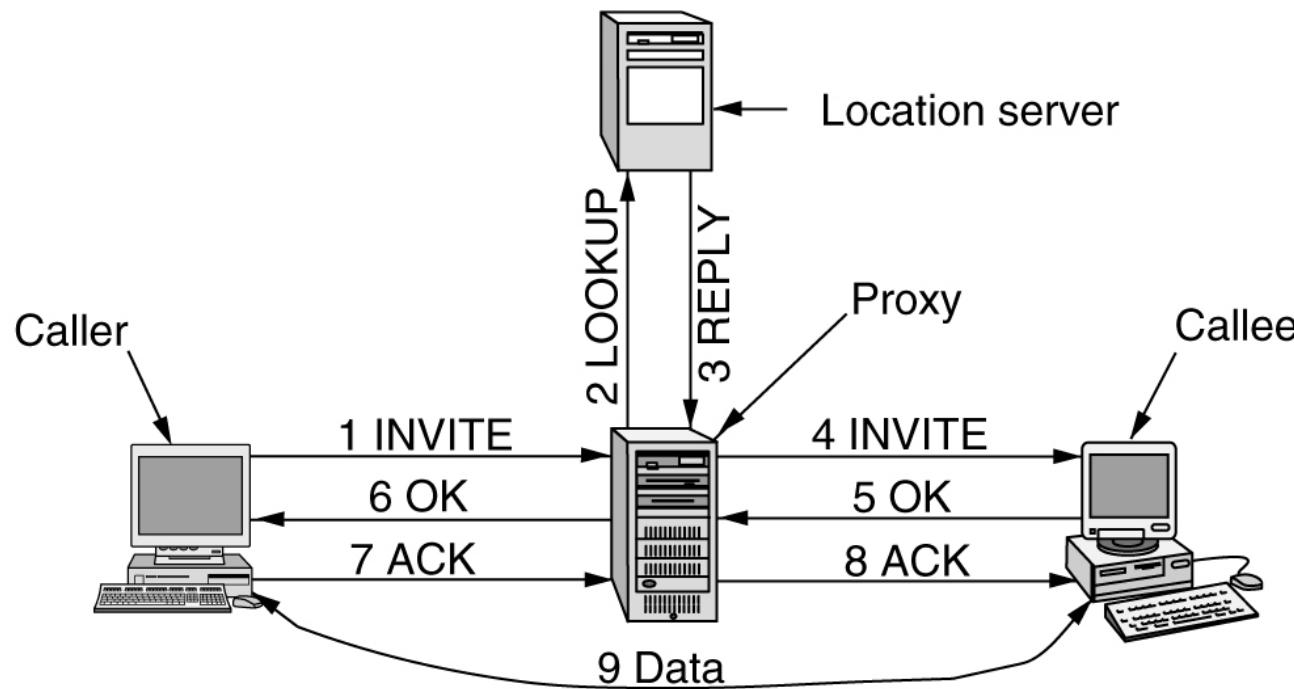
Logical channels between the caller and callee during a call.

SIP – The Session Initiation Protocol

Method	Description
INVITE	Request initiation of a session
ACK	Confirm that a session has been initiated
BYE	Request termination of a session
OPTIONS	Query a host about its capabilities
CANCEL	Cancel a pending request
REGISTER	Inform a redirection server about the user's current location

The SIP methods defined in the core specification.

SIP (2)

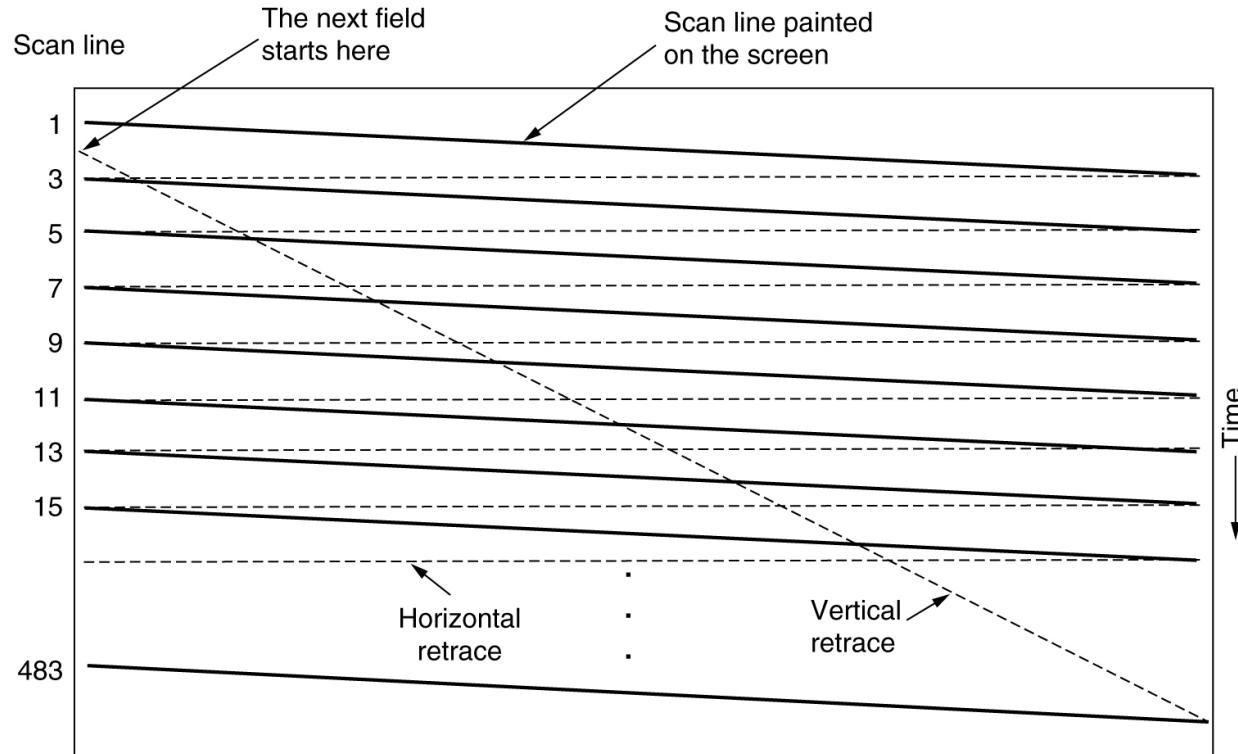


Use a proxy and redirection servers with SIP.

Comparison of H.323 and SIP

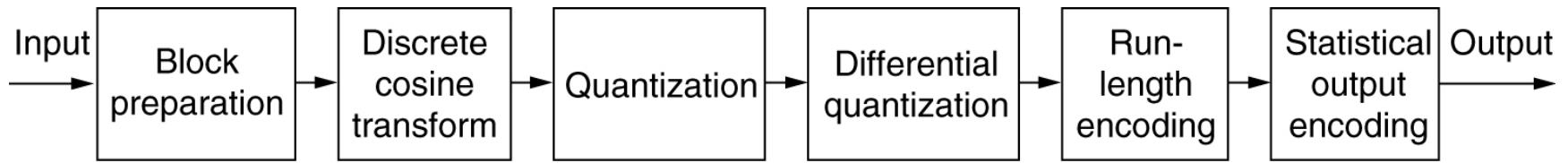
Item	H.323	SIP
Designed by	ITU	IETF
Compatibility with PSTN	Yes	Largely
Compatibility with Internet	No	Yes
Architecture	Monolithic	Modular
Completeness	Full protocol stack	SIP just handles setup
Parameter negotiation	Yes	Yes
Call signaling	Q.931 over TCP	SIP over TCP or UDP
Message format	Binary	ASCII
Media transport	RTP/RTCP	RTP/RTCP
Multiparty calls	Yes	Yes
Multimedia conferences	Yes	No
Addressing	Host or telephone number	URL
Call termination	Explicit or TCP release	Explicit or timeout
Instant messaging	No	Yes
Encryption	Yes	Yes
Size of standards	1400 pages	250 pages
Implementation	Large and complex	Moderate
Status	Widely deployed	Up and coming

Video Analog Systems



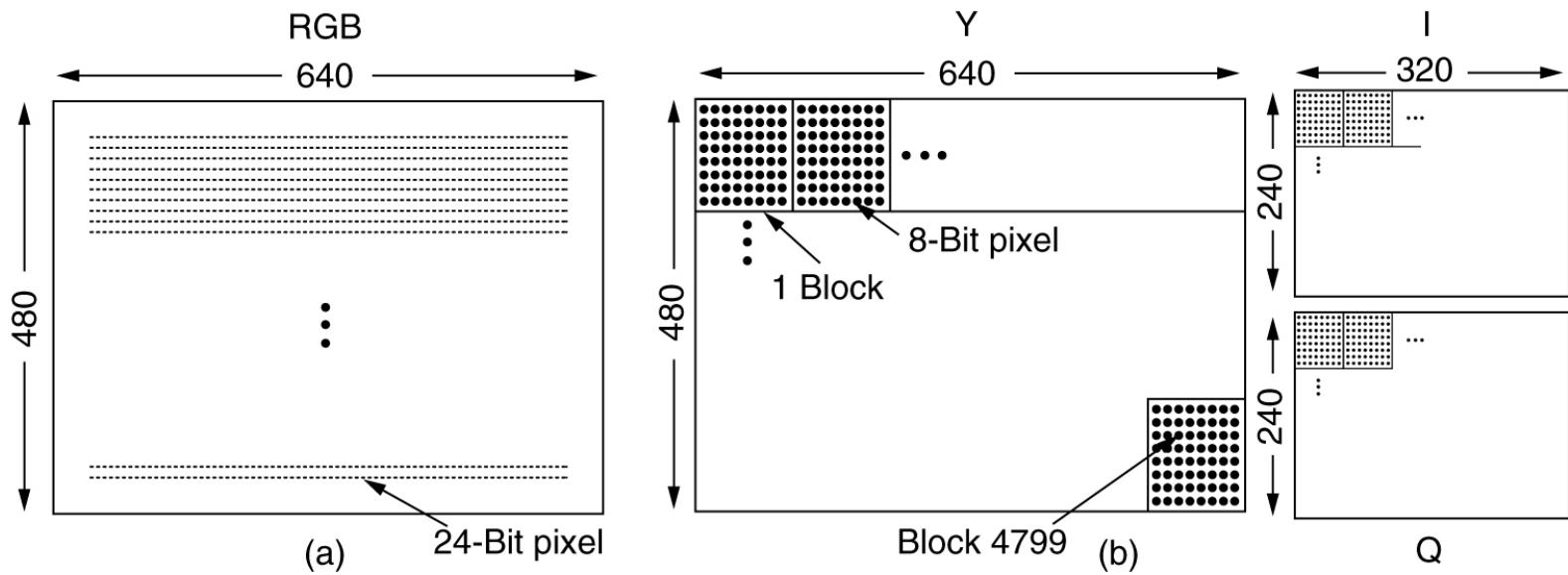
The scanning pattern used for NTSC video and television.

The JPEG Standard



The operation of JPEG in lossy sequential mode.

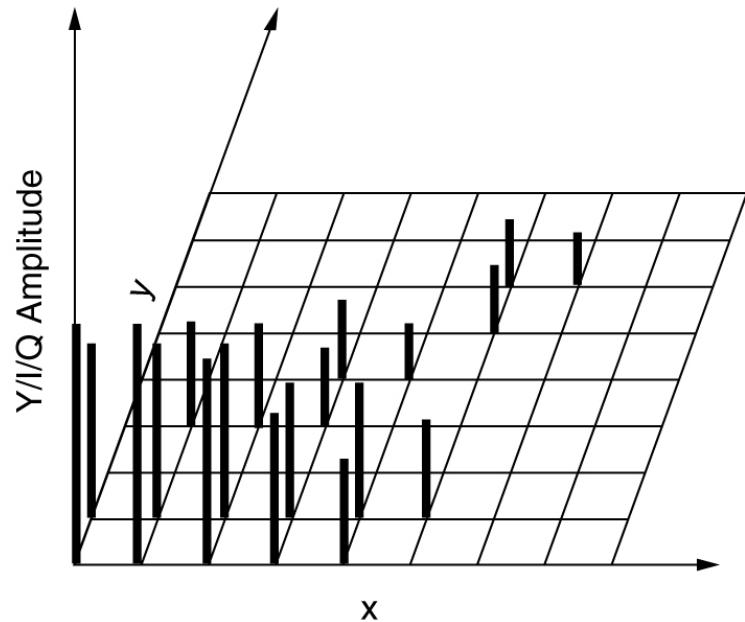
The JPEG Standard (2)



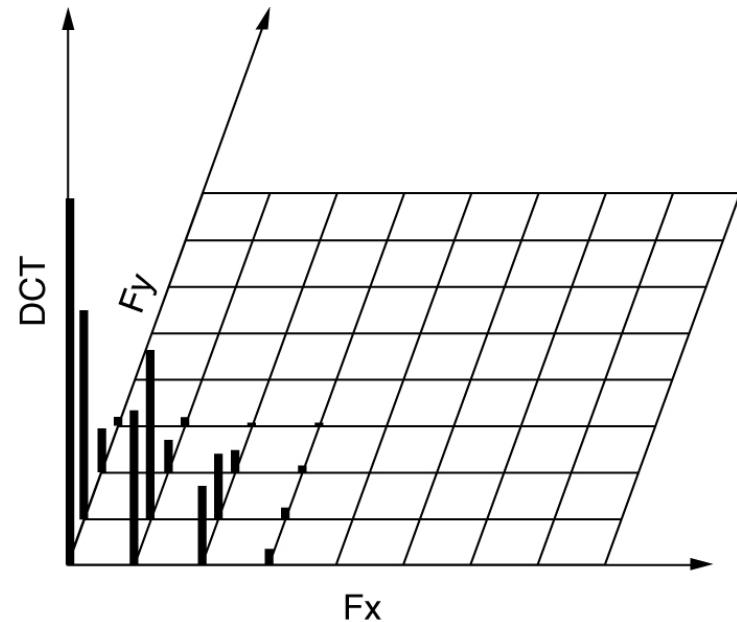
(a) RGB input data.

(b) After block preparation.

The JPEG Standard (3)



(a)



(b)

- (a) One block of the Y matrix.
- (b) The DCT coefficients.

The JPEG Standard (4)

DCT Coefficients

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization table

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Quantized coefficients

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

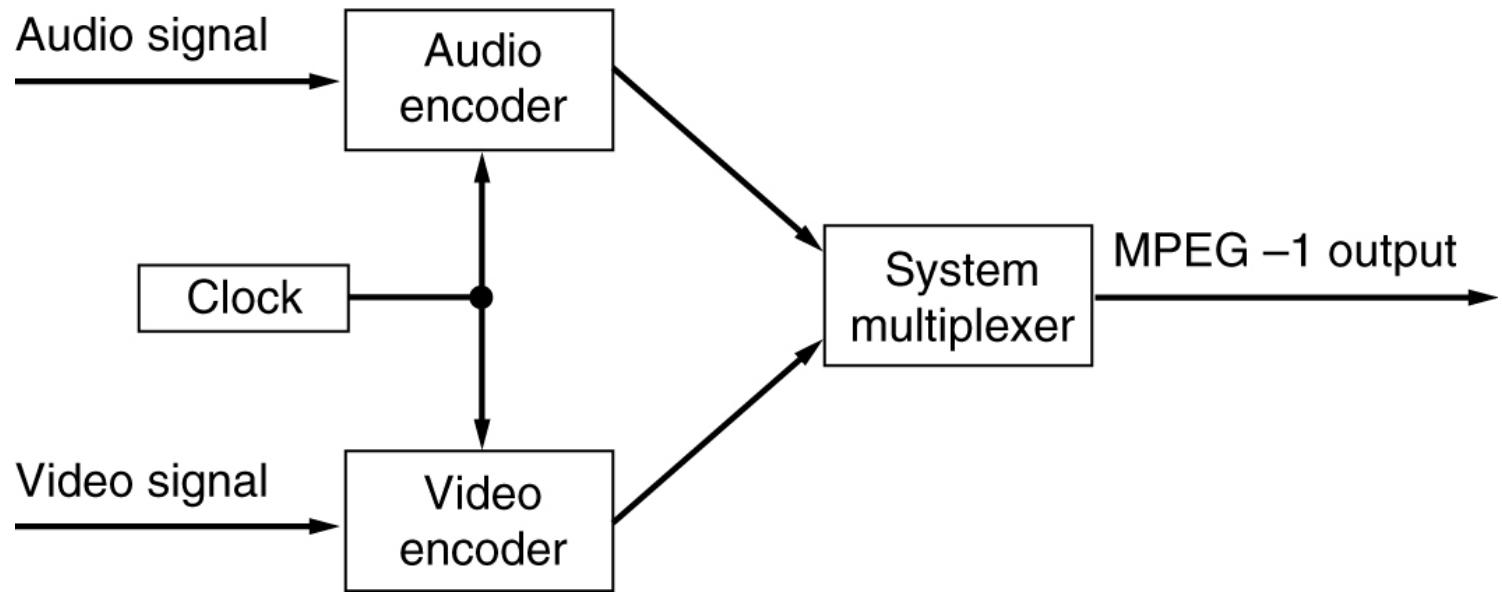
Computation of the quantized DTC coefficients.

The JPEG Standard (5)

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

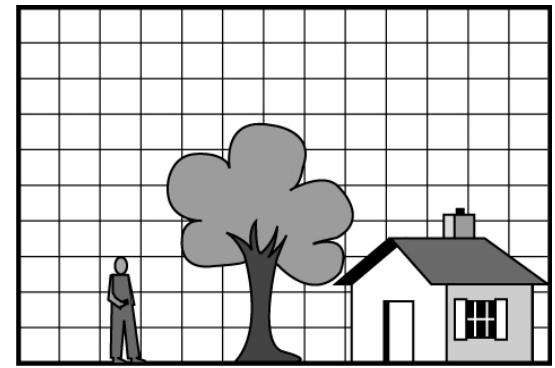
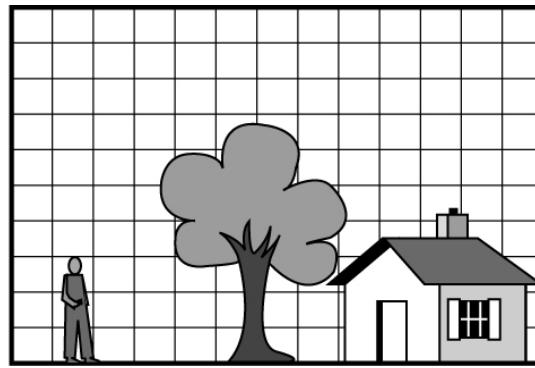
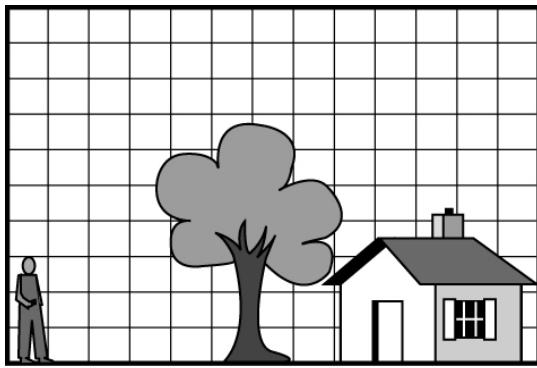
The order in which the quantized values are transmitted.

The MPEG Standard



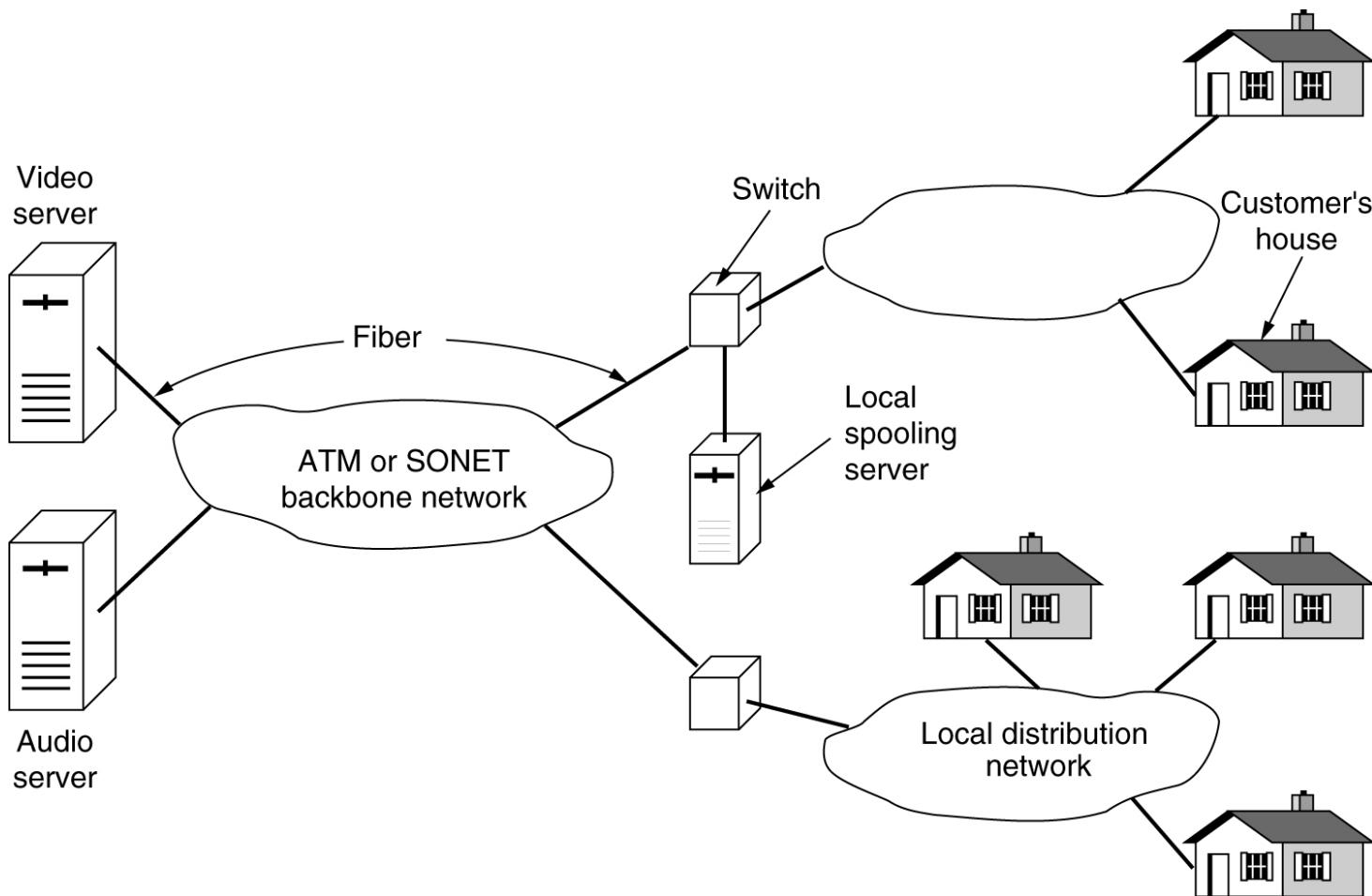
Synchronization of the audio and video streams in MPEG-1.

The MPEG Standard (2)



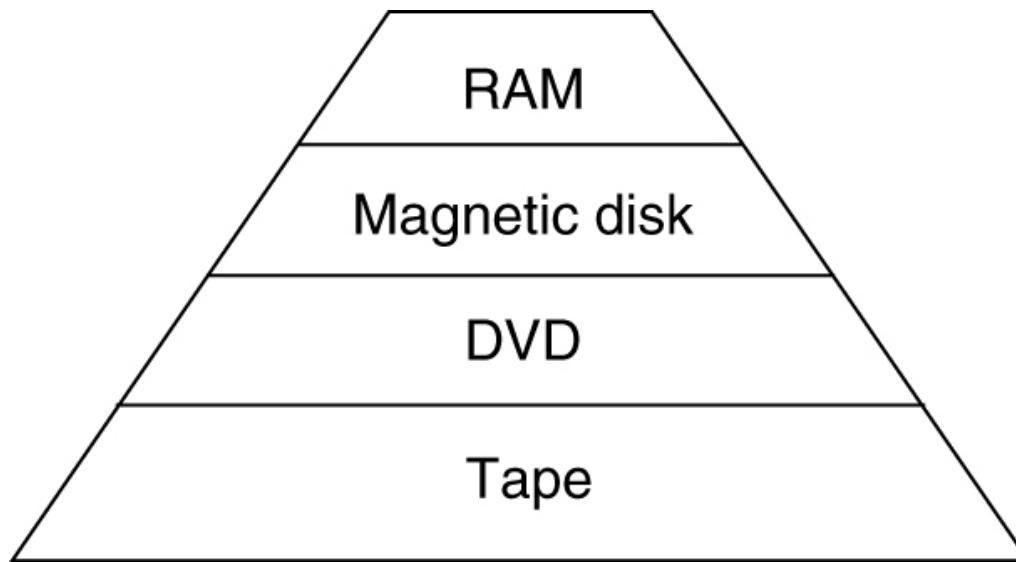
Three consecutive frames.

Video on Demand



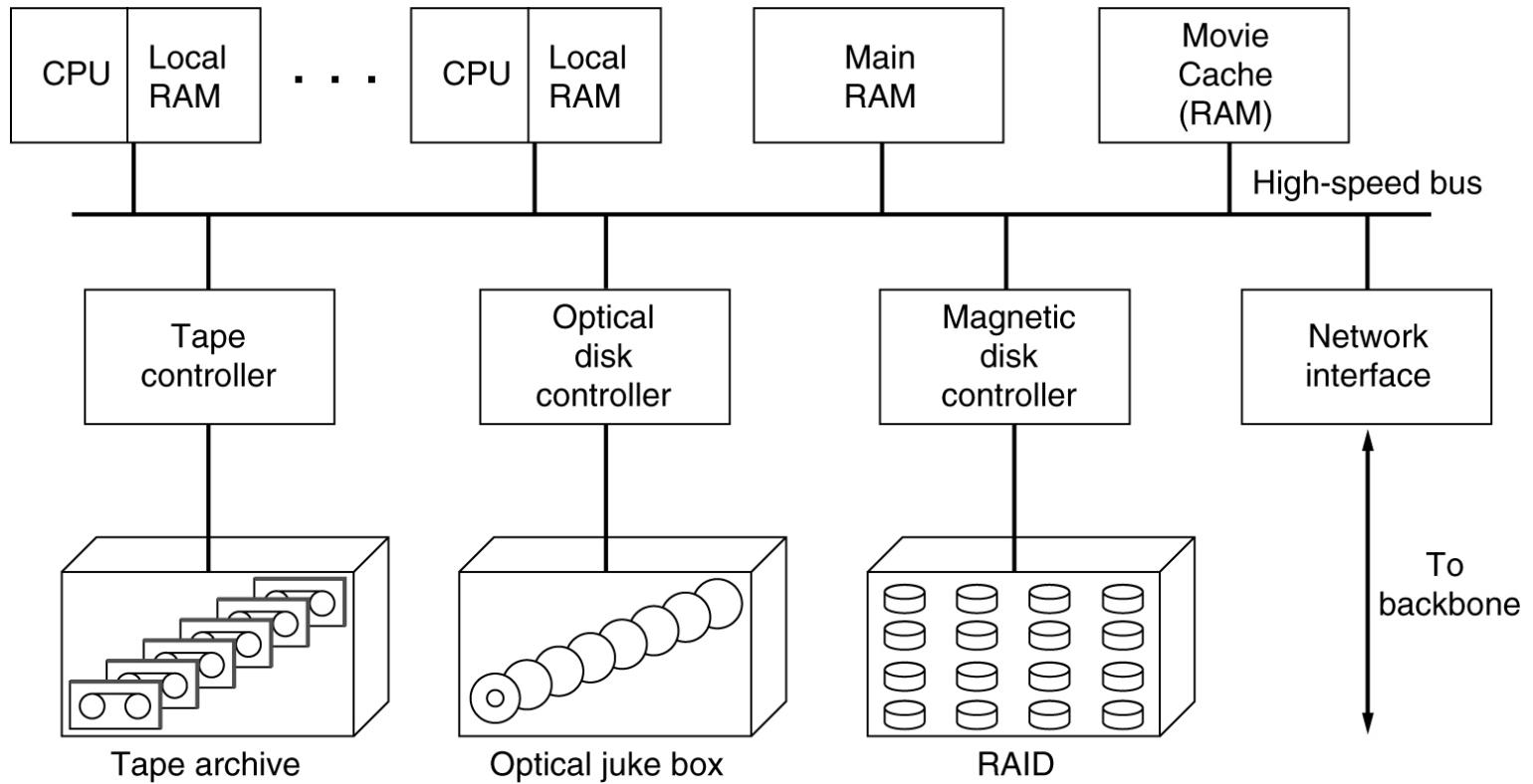
Overview of a video-on-demand system.

Video Servers



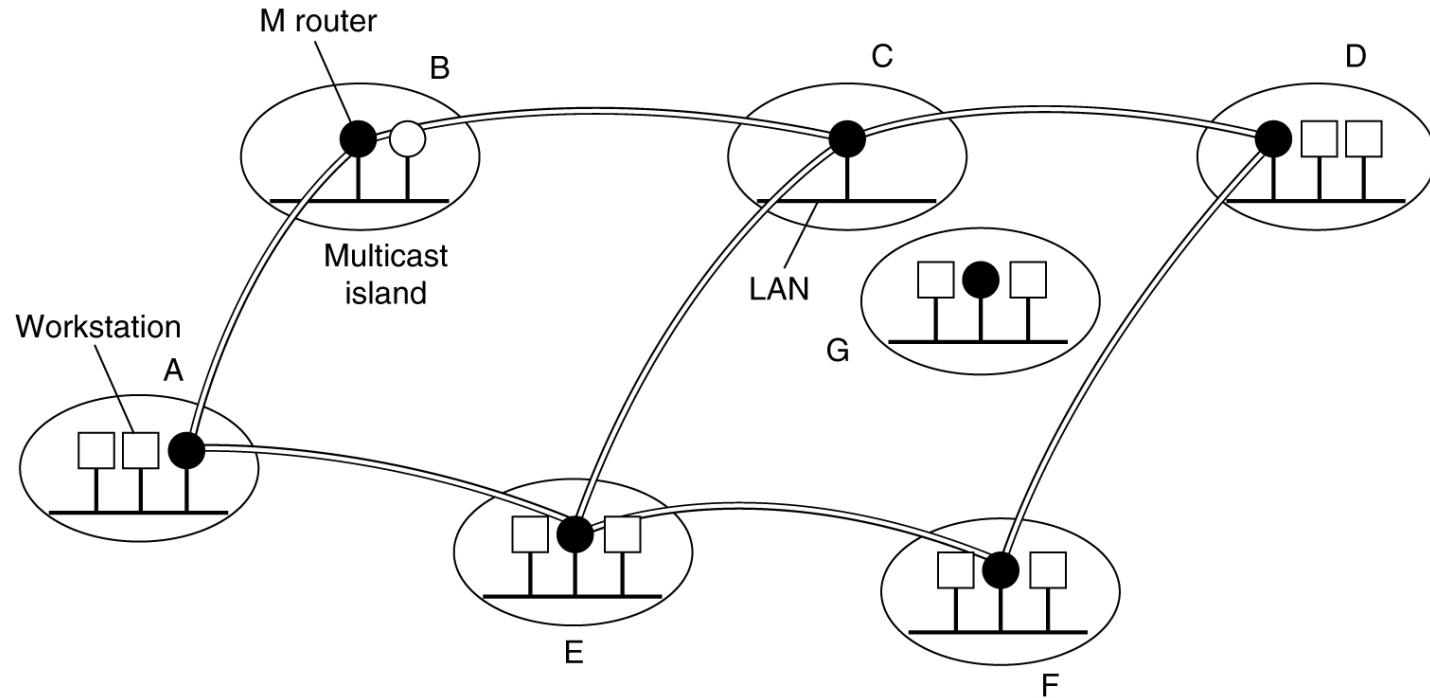
A video server storage hierarchy.

Video Servers (2)



The hardware architecture of a typical video server.

The MBone – The Multicast Backbone



MBone consists of multicast islands connected by tunnels.



Chapter 8

Network Security

Cryptography

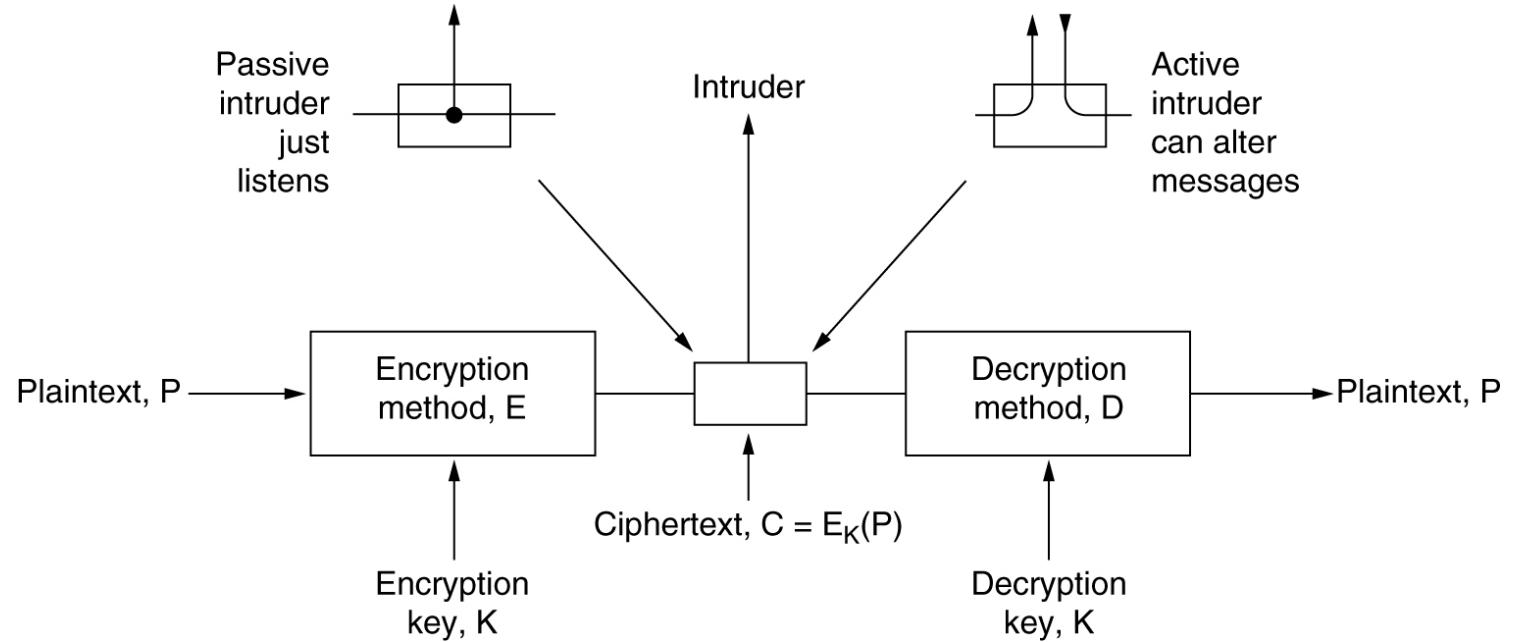
- Introduction to Cryptography
- Substitution Ciphers
- Transposition Ciphers
- One-Time Pads
- Two Fundamental Cryptographic Principles

Need for Security

Adversary	Goal
Student	To have fun snooping on people's e-mail
Cracker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Businessman	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by e-mail
Con man	To steal credit card numbers for sale
Spy	To learn an enemy's military or industrial secrets
Terrorist	To steal germ warfare secrets

Some people who cause security problems and why.

An Introduction to Cryptography



The encryption model (for a symmetric-key cipher).

Transposition Ciphers

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

please transfer one million dollars to
my swiss bank account six two two

Ciphertext

AFLLSKSOSELAWAIATO OSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUEIRICXB

A transposition cipher.

One-Time Pads

Message 1: 1001001 0100000 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110

Pad 1: 1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011

Ciphertext: 0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101

Pad 2: 1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110

Plaintext 2: 1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011

The use of a one-time pad for encryption and the possibility of getting any possible plaintext from the ciphertext by the use of some other pad.

Quantum Cryptography

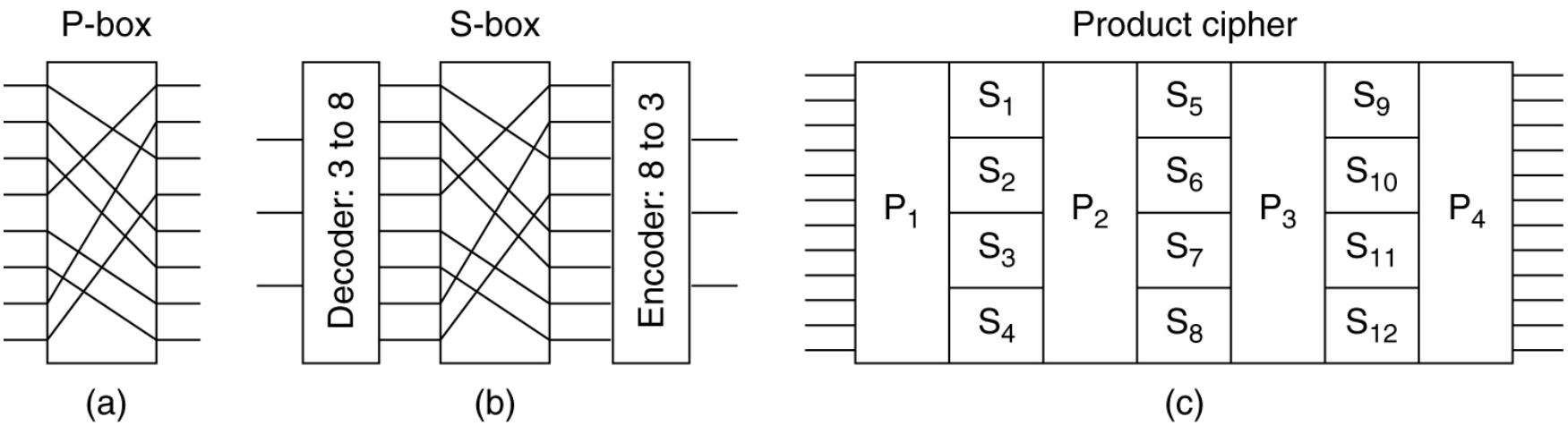
Bit number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Data	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	0	
(a)	↖	↑↓	↑↓	↖	↔	↖	↗	↔	↔	↑↓	↔	↗	↖	↔	↔	↑↓	What Alice sends
(b)	↔	↔	✗	✗	✗	✗	↔	✗	↔	✗	✗	✗	✗	✗	✗	✗	Bob's bases
(c)	↓	↓	↗	↖	↖	↑↓	↔	↗	↔	↗	↔	↖	↖	↔	↔	↗	What Bob gets
(d)	No	Yes	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Correct basis?
(e)		0		1				0	1		1	0	0		1		One-time pad
(f)	✗	↔	↔	✗	↔	↔	✗	↔	↔	✗	✗	↔	✗	✗	✗	✗	Trudy's bases
(g)	x	0	x	1	x	x	x	?	1	x	?	?	0	x	?	x	Trudy's pad

An example of quantum cryptography.

Symmetric-Key Algorithms

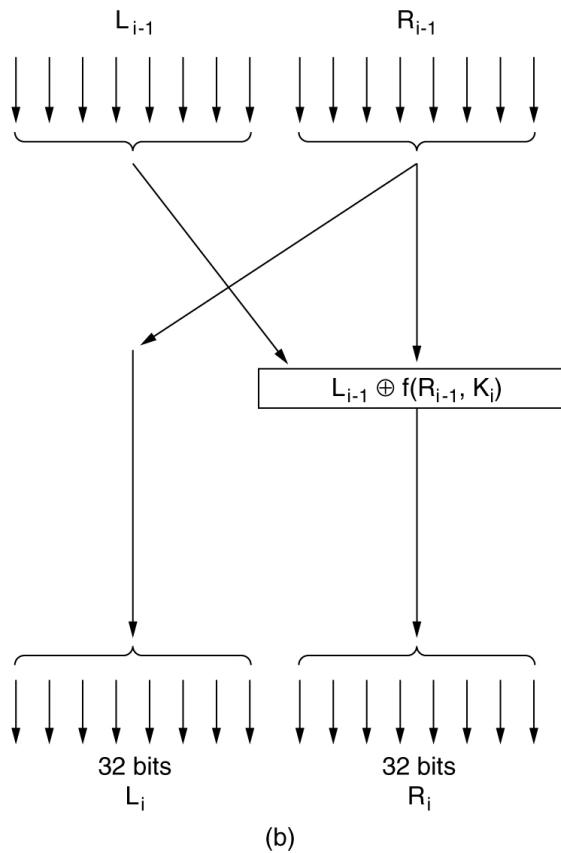
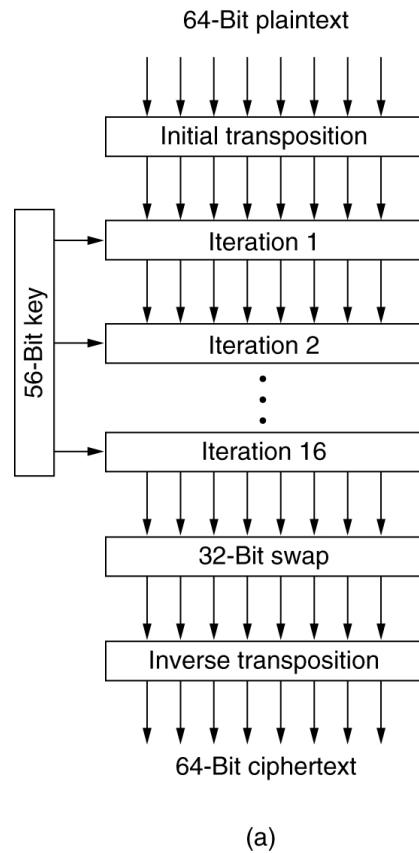
- DES – The Data Encryption Standard
- AES – The Advanced Encryption Standard
- Cipher Modes
- Other Ciphers
- Cryptanalysis

Product Ciphers



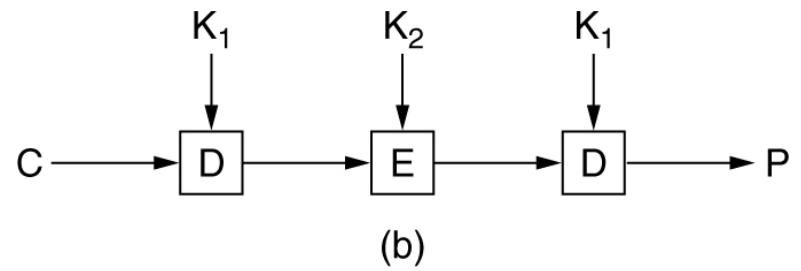
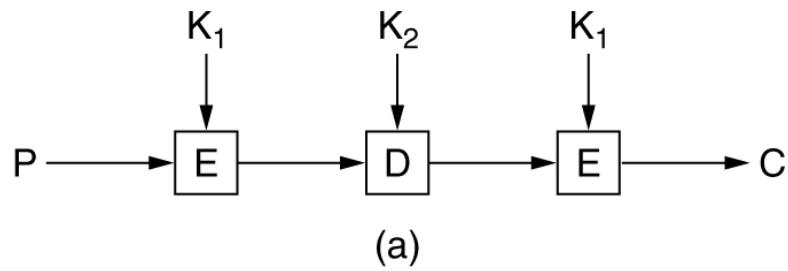
Basic elements of product ciphers. (a) P-box. (b) S-box. (c) Product.

Data Encryption Standard



The data encryption standard. (a) General outline.
(b) Detail of one iteration. The circled + means exclusive OR.

Triple DES



(a) Triple encryption using DES. (b) Decryption.

AES – The Advanced Encryption Standard

Rules for AES proposals

2. The algorithm must be a symmetric block cipher.
3. The full design must be public.
4. Key lengths of 128, 192, and 256 bits supported.
5. Both software and hardware implementations required
6. The algorithm must be public or licensed on nondiscriminatory terms.

AES (2)

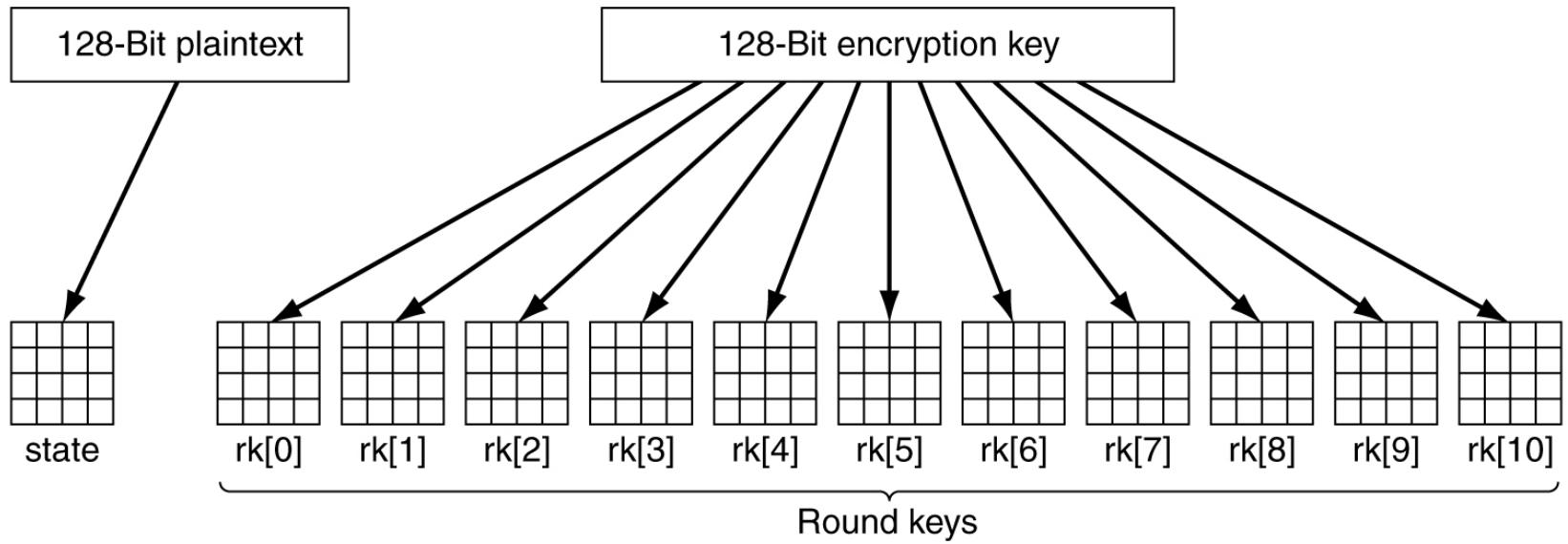
An outline of
Rijndael.

```
#define LENGTH 16                                /* # bytes in data block or key */
#define NROWS 4                                    /* number of rows in state */
#define NCOLS 4                                    /* number of columns in state */
#define ROUNDS 10                                   /* number of iterations */
typedef unsigned char byte;                      /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
    int r;                                         /* loop index */
    byte state[NROWS][NCOLS];                     /* current state */
    struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1]; /* round keys */
    expand_key(key, rk);                          /* construct the round keys */
    copy_plaintext_to_state(state, plaintext);    /* init current state */
    xor_roundkey_into_state(state, rk[0]);         /* XOR key into state */

    for (r = 1; r <= ROUNDS; r++) {
        substitute(state);                         /* apply S-box to each byte */
        rotate_rows(state);                      /* rotate row i by i bytes */
        if (r < ROUNDS) mix_columns(state);      /* mix function */
        xor_roundkey_into_state(state, rk[r]);    /* XOR key into state */
    }
    copy_state_to_ciphertext(ciphertext, state);   /* return result */
}
```

AES (3)

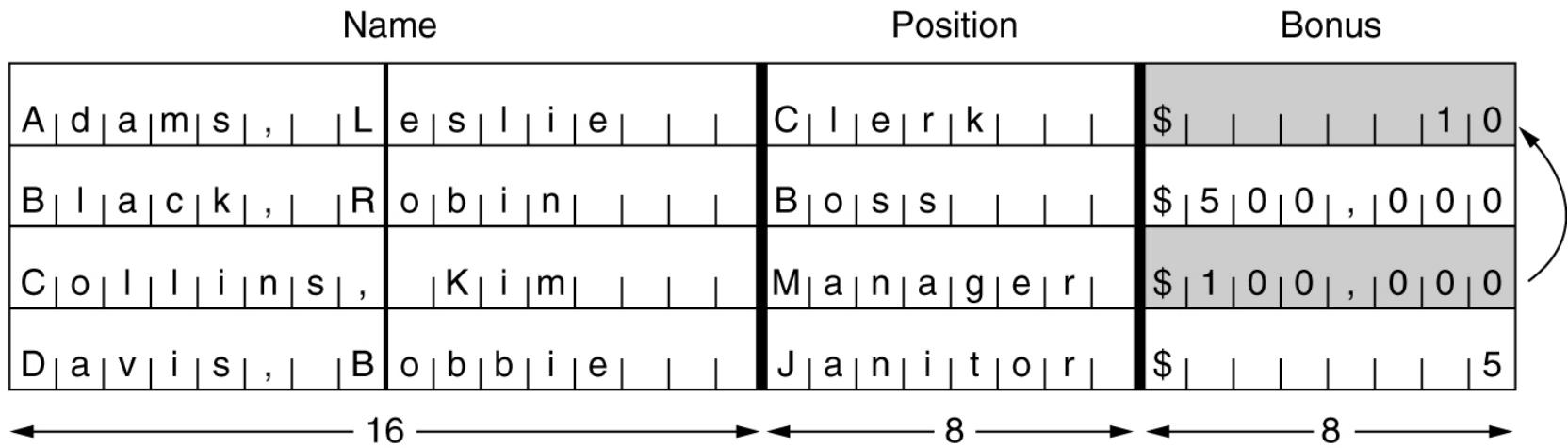


Creating of the *state* and *rk* arrays.

Electronic Code Book Mode

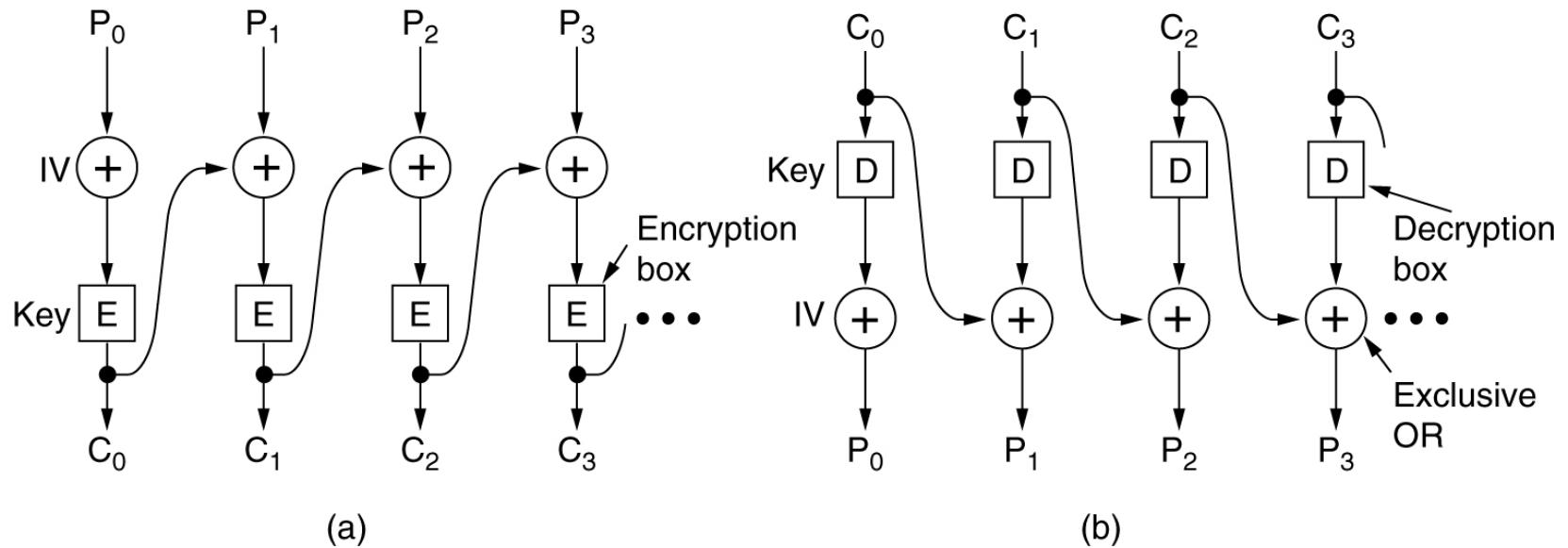
Name	Position	Bonus
A d a m s , L e s l i e	C l e r k	\$ 1 0
B l a c k , R o b i n	B o s s	\$ 5 0 0 , 0 0 0
C o l l i n s , K i m	M a n a g e r	\$ 1 0 0 , 0 0 0
D a v i s , B o b b i e	J a n i t o r	\$ 5

Bytes ← 16 → ← 8 → ← 8 →



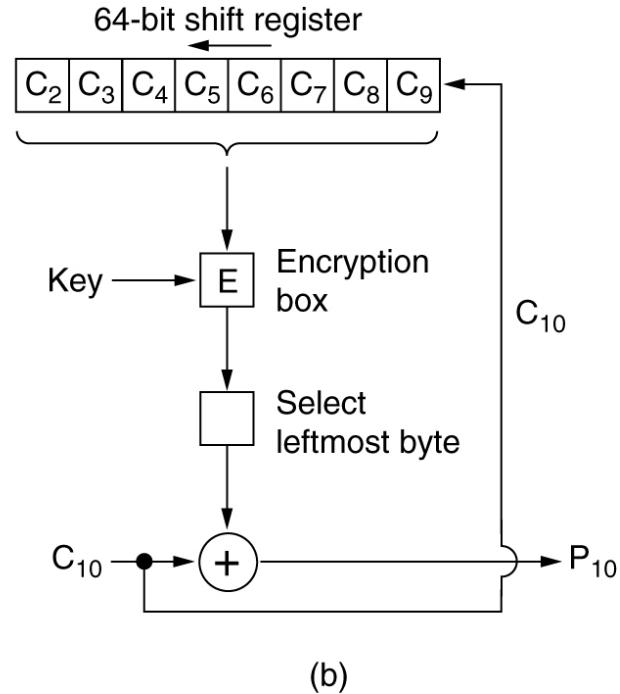
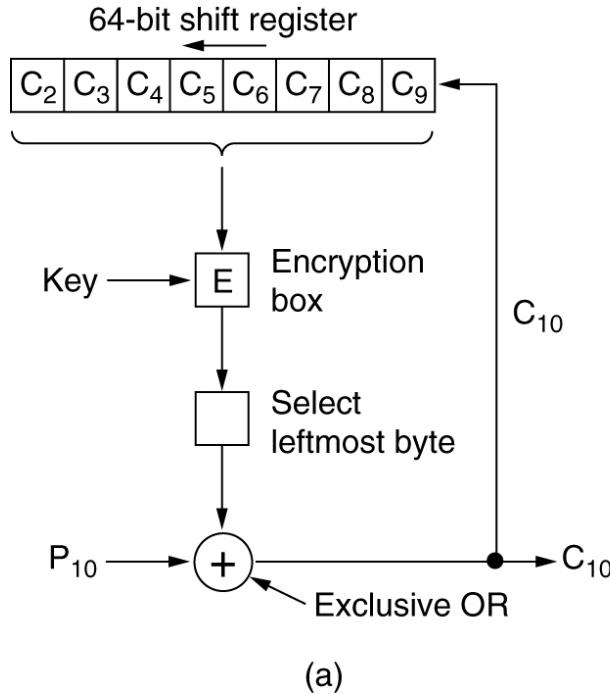
The plaintext of a file encrypted as 16 DES blocks.

Cipher Block Chaining Mode



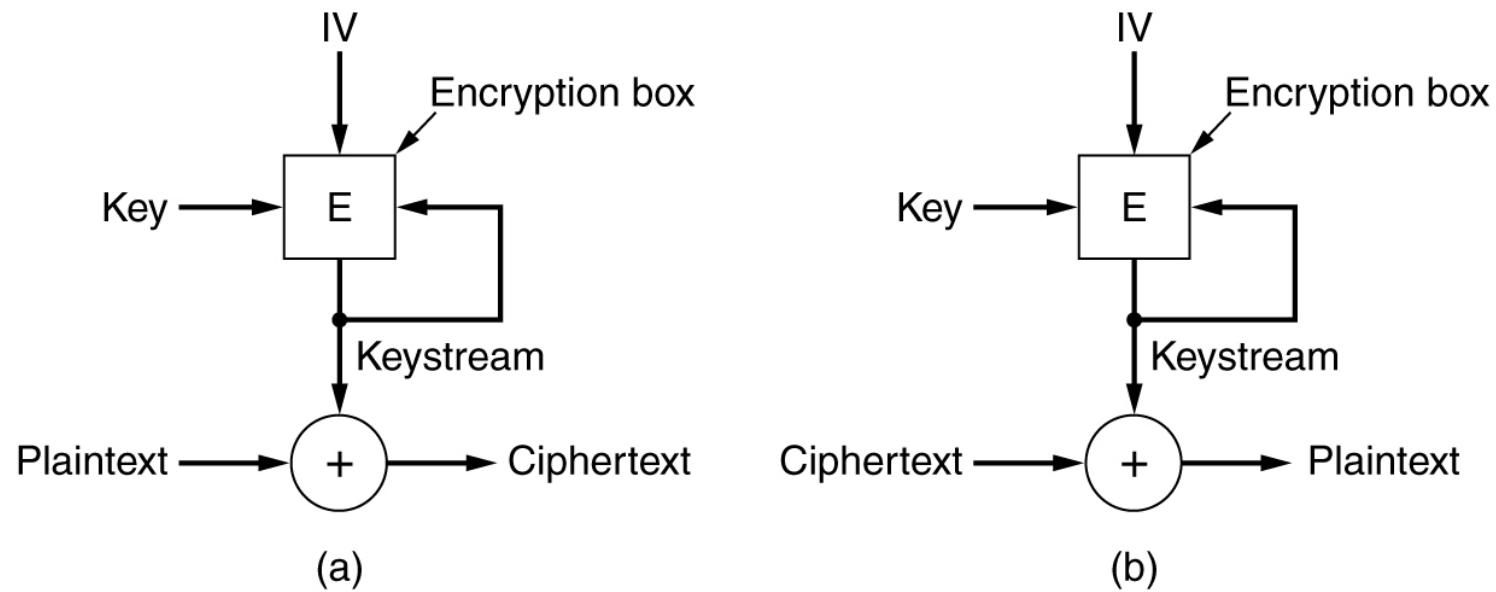
Cipher block chaining. (a) Encryption. (b) Decryption.

Cipher Feedback Mode



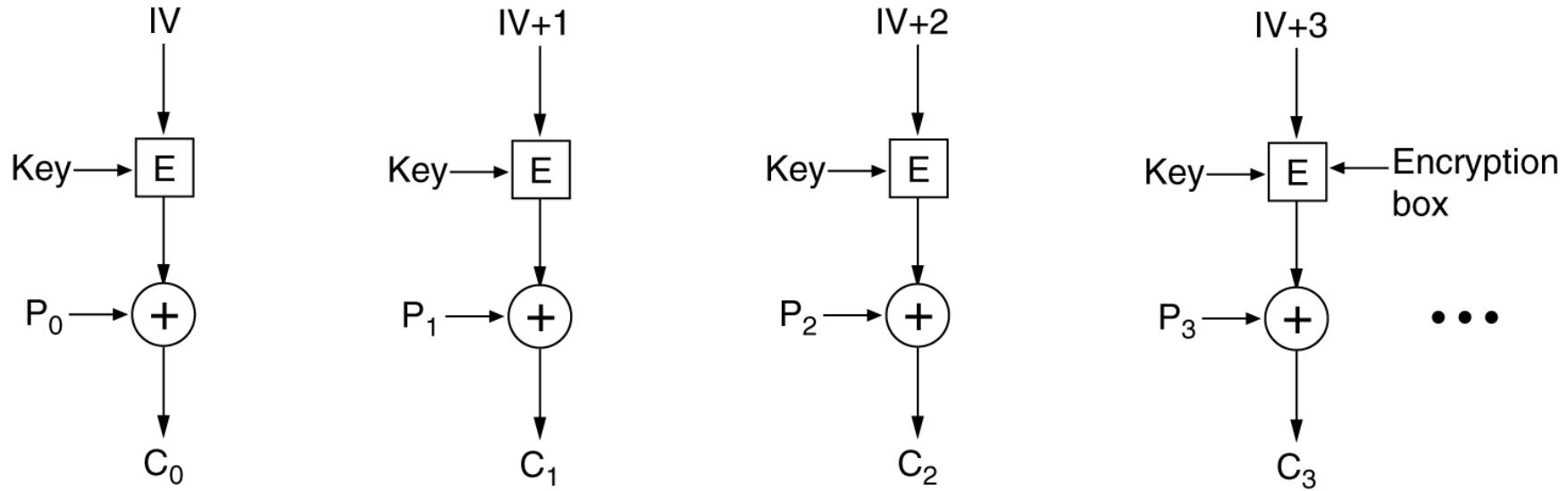
(a) Encryption. (c) Decryption.

Stream Cipher Mode



A stream cipher. (a) Encryption. (b) Decryption.

Counter Mode



Encryption using counter mode.

Cryptanalysis

Cipher	Author	Key length	Comments
Blowfish	Bruce Schneier	1–448 bits	Old and slow
DES	IBM	56 bits	Too weak to use now
IDEA	Massey and Xuejia	128 bits	Good, but patented
RC4	Ronald Rivest	1–2048 bits	Caution: some keys are weak
RC5	Ronald Rivest	128–256 bits	Good, but patented
Rijndael	Daemen and Rijmen	128–256 bits	Best choice
Serpent	Anderson, Biham, Knudsen	128–256 bits	Very strong
Triple DES	IBM	168 bits	Second best choice
Twofish	Bruce Schneier	128–256 bits	Very strong; widely used

Some common symmetric-key cryptographic algorithms.

Public-Key Algorithms

- RSA
- Other Public-Key Algorithms

RSA

Plaintext (P)		Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$
S	19	6859	28	13492928512	19
U	21	9261	21	1801088541	21
Z	26	17576	20	1280000000	26
A	01	1	1	1	01
N	14	2744	5	78125	14
N	14	2744	5	78125	14
E	05	125	26	8031810176	05

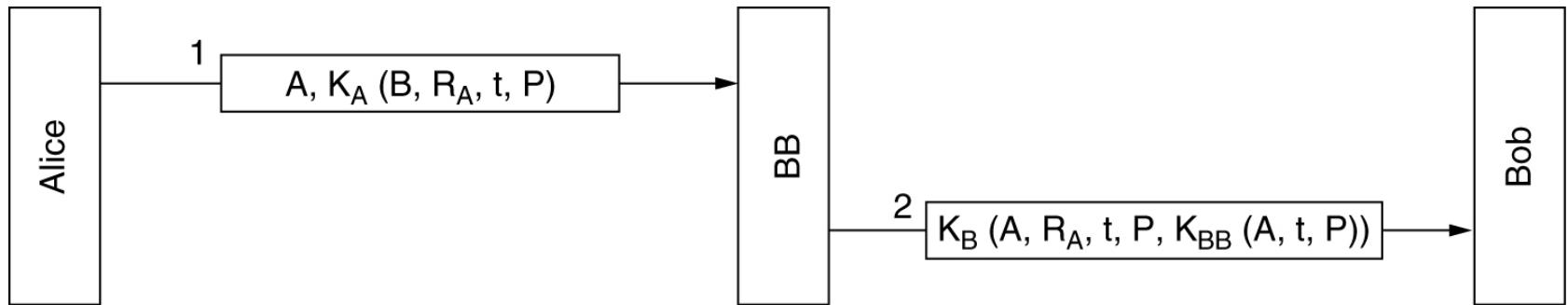
$\brace{ \text{Sender's computation} }$ $\brace{ \text{Receiver's computation} }$

An example of the RSA algorithm.

Digital Signatures

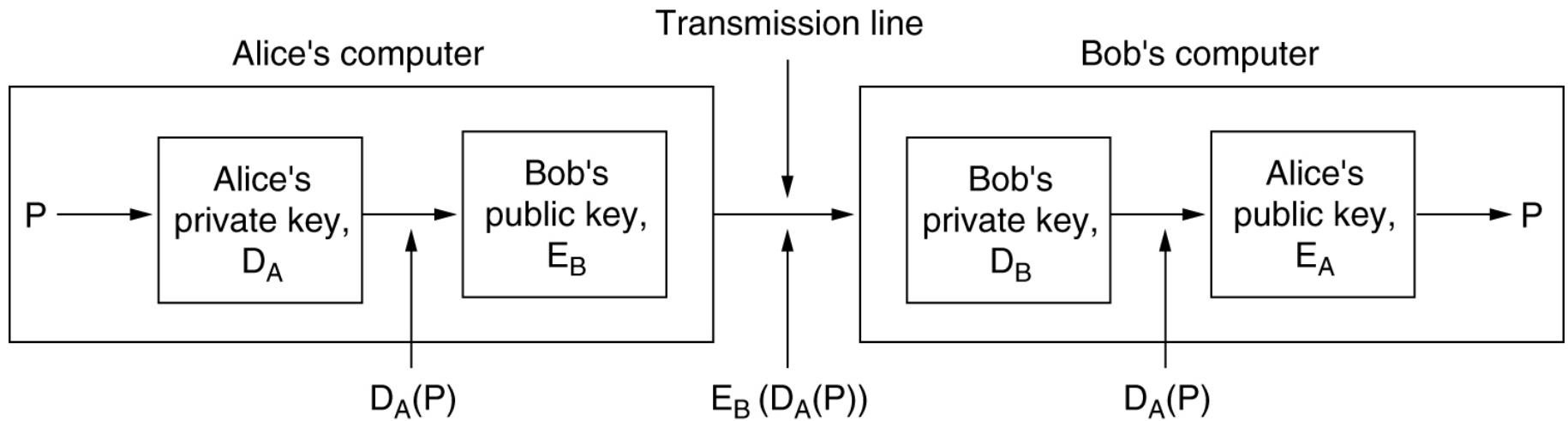
- Symmetric-Key Signatures
- Public-Key Signatures
- Message Digests
- The Birthday Attack

Symmetric-Key Signatures



Digital signatures with Big Brother.

Public-Key Signatures



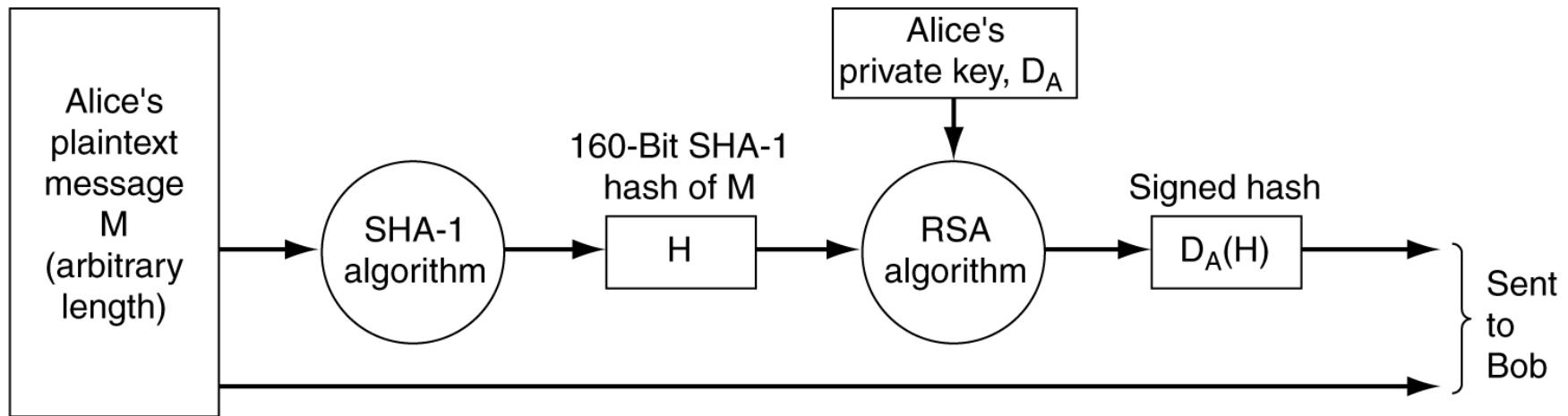
Digital signatures using public-key cryptography.

Message Digests



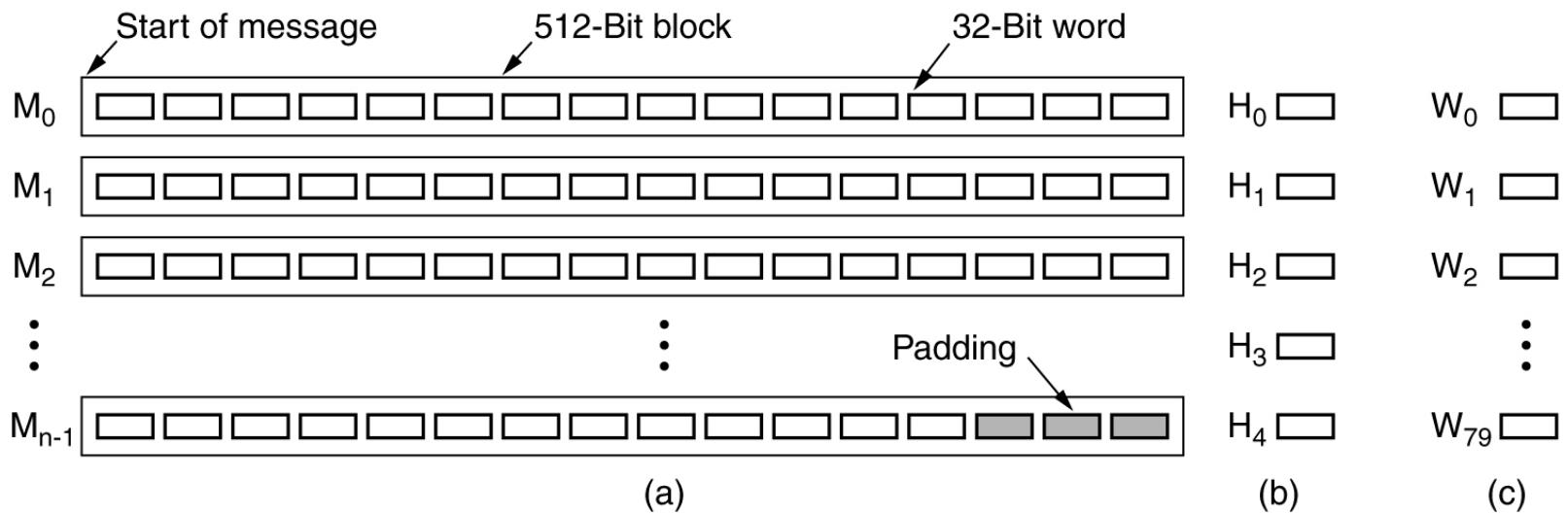
Digital signatures using message digests.

SHA-1



Use of SHA-1 and RSA for signing nonsecret messages.

SHA-1 (2)

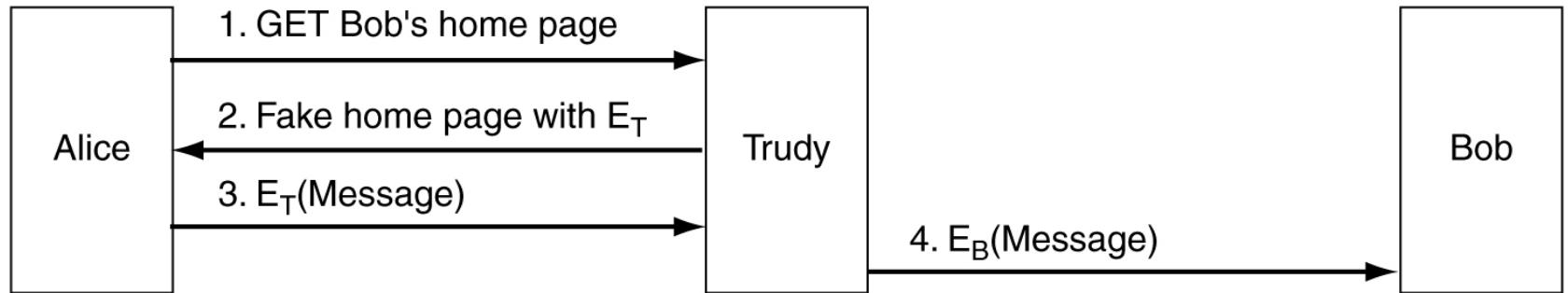


- (a) A message padded out to a multiple of 512 bits.
(b) The output variables. (c) The word array.

Management of Public Keys

- Certificates
- X.509
- Public Key Infrastructures

Problems with Public-Key Encryption



A way for Trudy to subvert public-key encryption.

Certificates

I hereby certify that the public key

19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A

belongs to

Robert John Smith

12345 University Avenue

Berkeley, CA 94702

Birthday: July 4, 1958

Email: bob@superduper.net.com

SHA-1 hash of the above certificate signed with the CA's private key

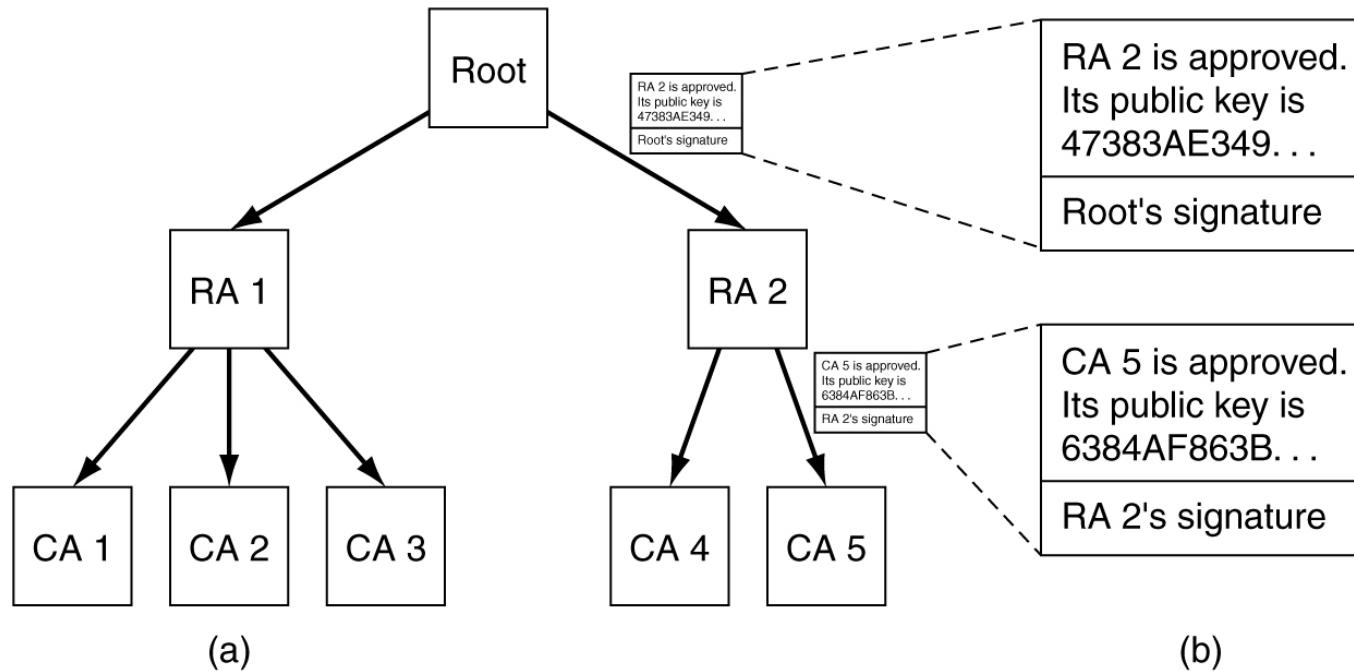
A possible certificate and its signed hash.

X.509

Field	Meaning
Version	Which version of X.509
Serial number	This number plus the CA's name uniquely identifies the certificate
Signature algorithm	The algorithm used to sign the certificate
Issuer	X.500 name of the CA
Validity period	The starting and ending times of the validity period
Subject name	The entity whose key is being certified
Public key	The subject's public key and the ID of the algorithm using it
Issuer ID	An optional ID uniquely identifying the certificate's issuer
Subject ID	An optional ID uniquely identifying the certificate's subject
Extensions	Many extensions have been defined
Signature	The certificate's signature (signed by the CA's private key)

The basic fields of an X.509 certificate.

Public-Key Infrastructures

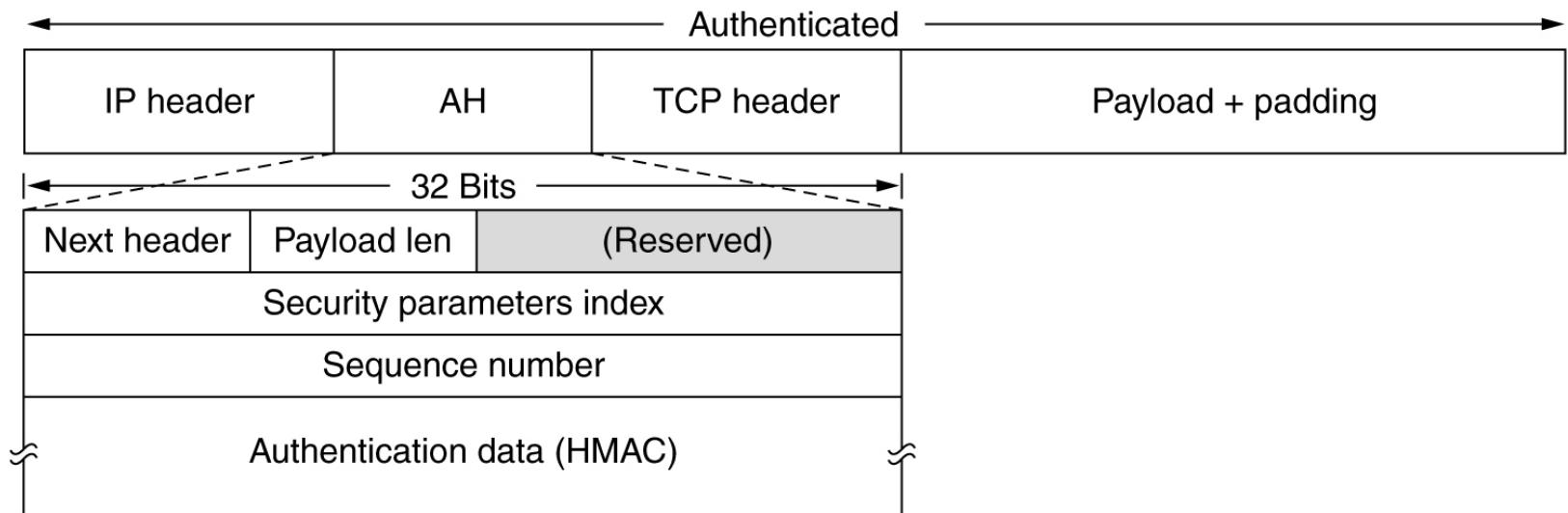


(a) A hierarchical PKI. (b) A chain of certificates.

Communication Security

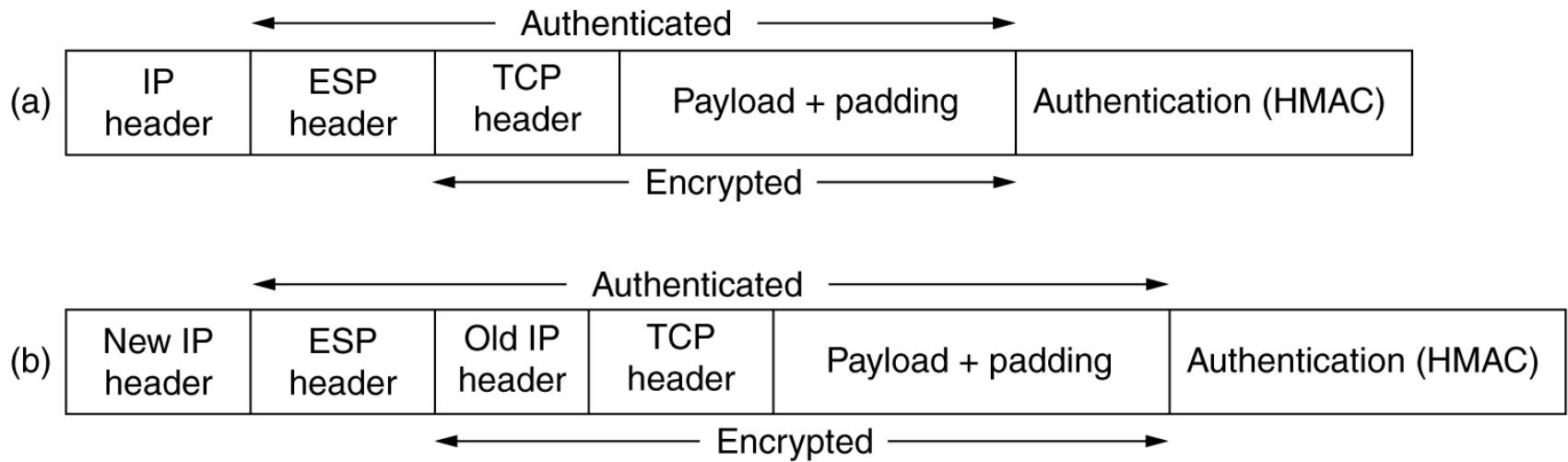
- IPsec
- Firewalls
- Virtual Private Networks
- Wireless Security

IPsec



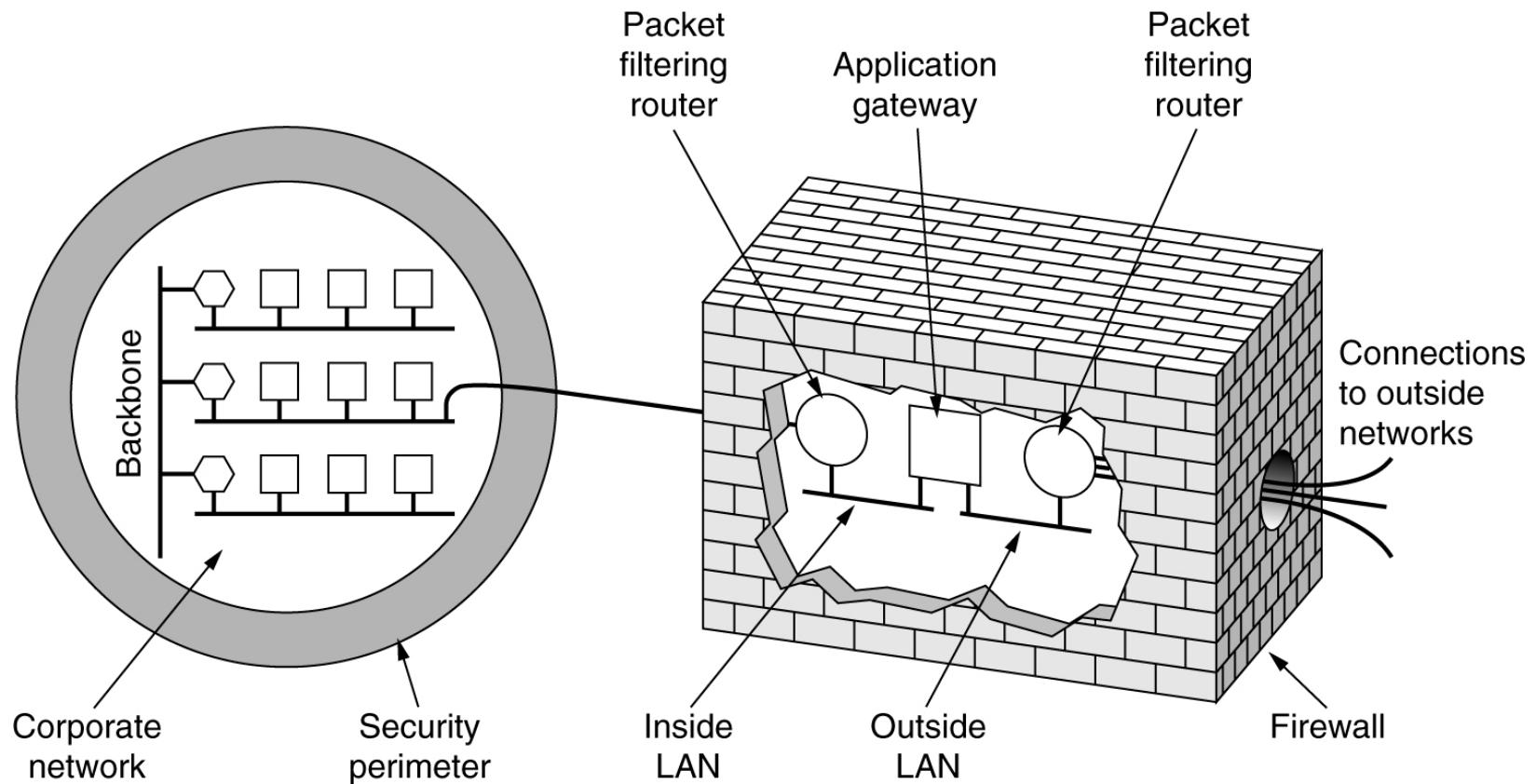
The IPsec authentication header in transport mode for IPv4.

IPsec (2)



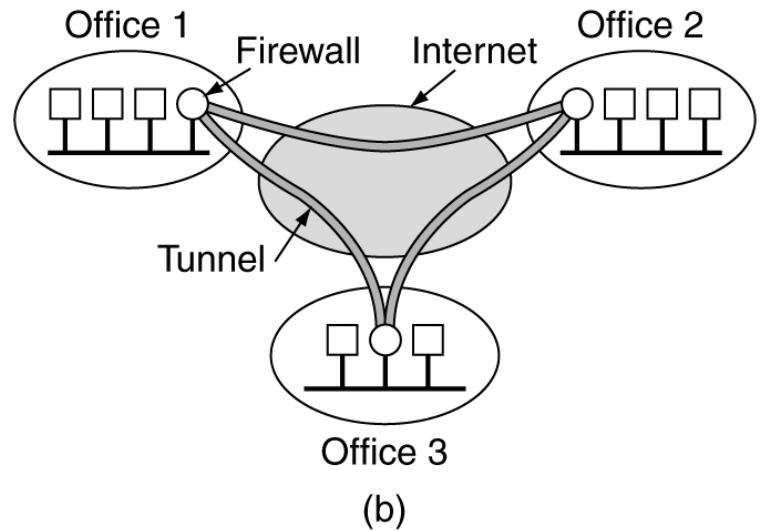
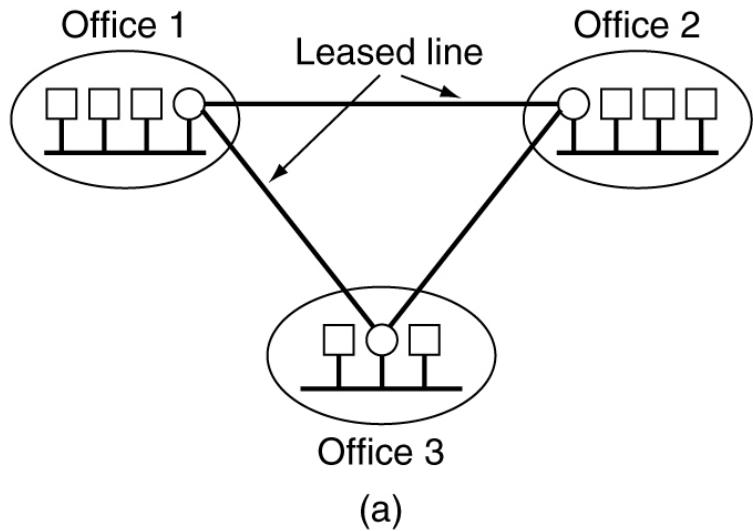
(a) ESP in transport mode. (b) ESP in tunnel mode.

Firewalls



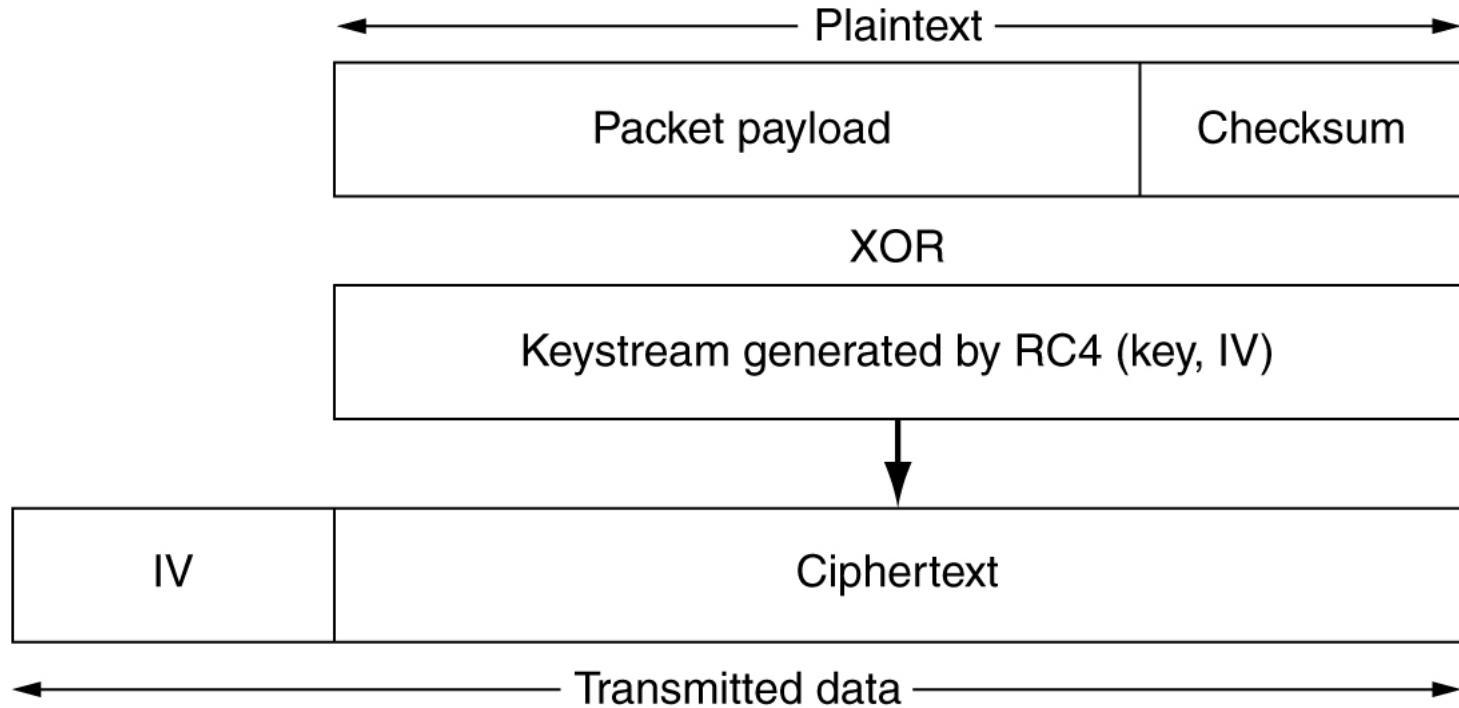
A firewall consisting of two packet filters and an application gateway.

Virtual Private Networks



(a) A leased-line private network. (b) A virtual private network.

802.11 Security

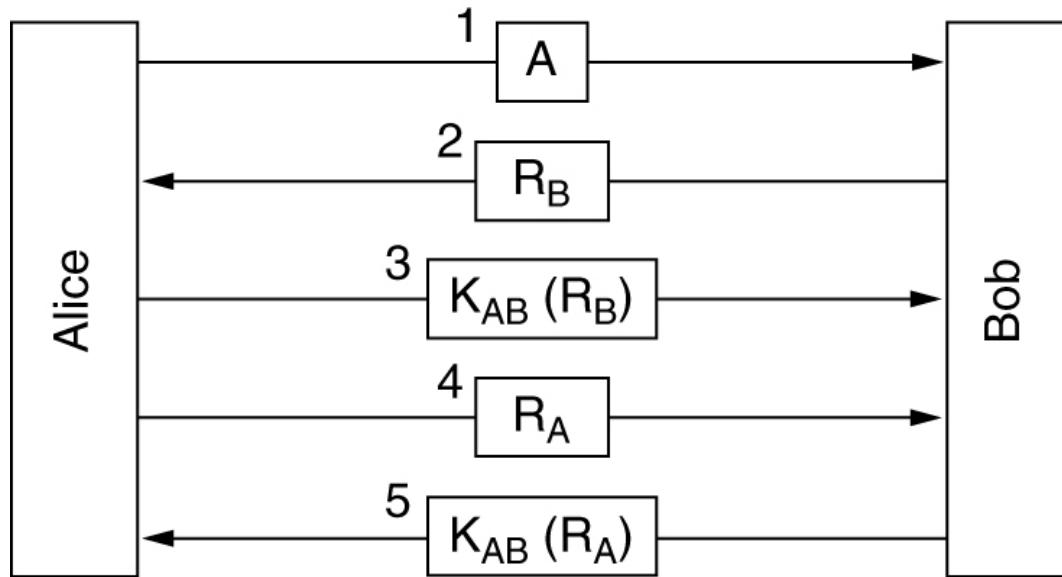


Packet encryption using WEP.

Authentication Protocols

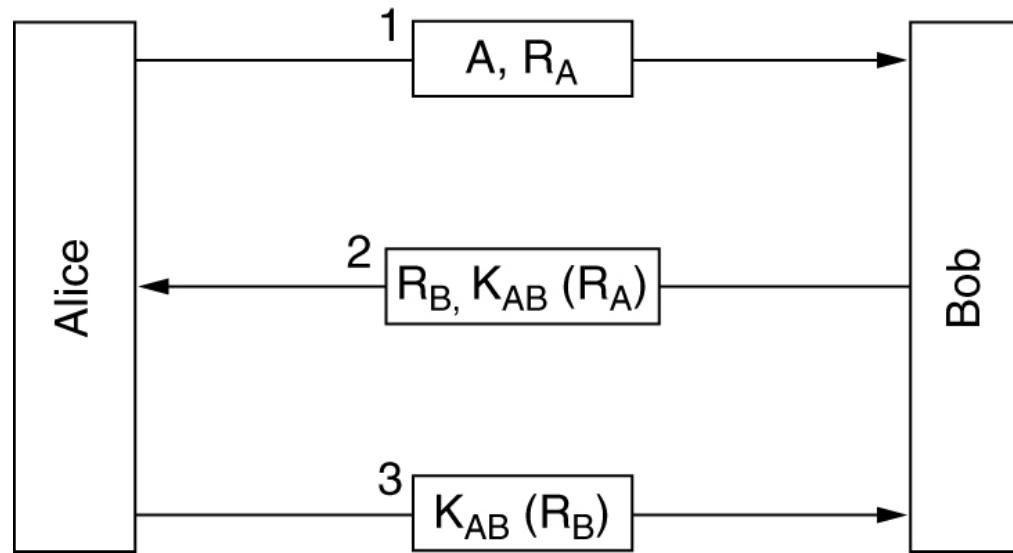
- Authentication Based on a Shared Secret Key
- Establishing a Shared Key: Diffie-Hellman
- Authentication Using a Key Distribution Center
- Authentication Using Kerberos
- Authentication Using Public-Key Cryptography

Authentication Based on a Shared Secret Key



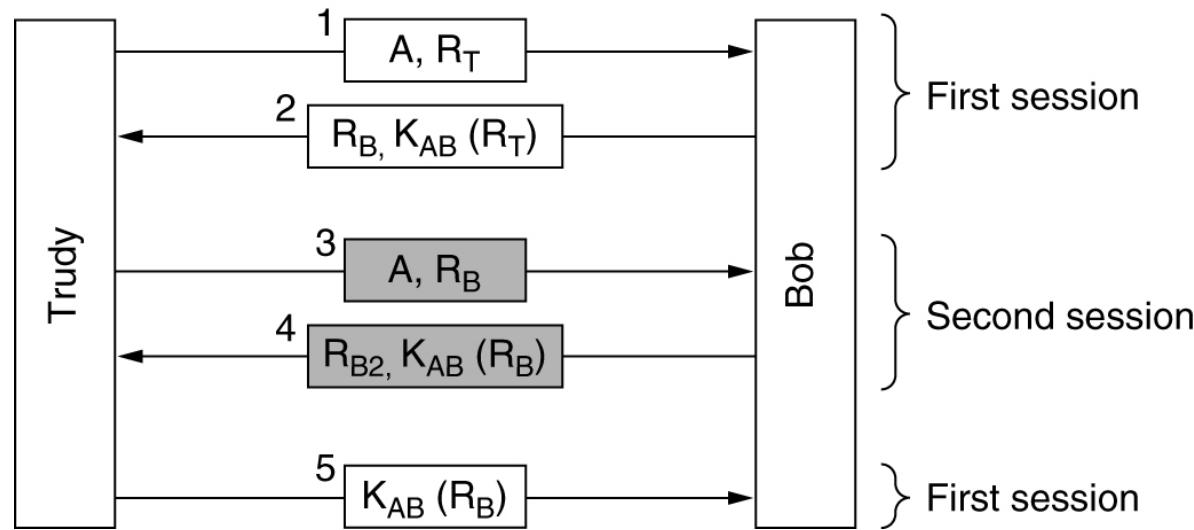
Two-way authentication using a challenge-response protocol.

Authentication Based on a Shared Secret Key (2)



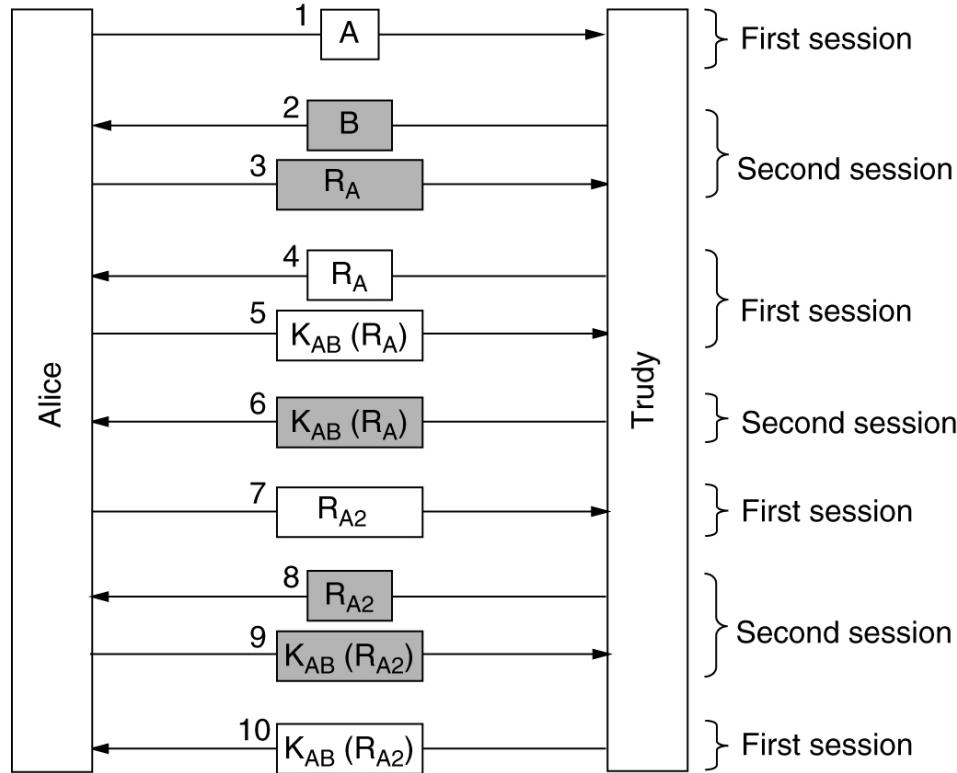
A shortened two-way authentication protocol.

Authentication Based on a Shared Secret Key (3)



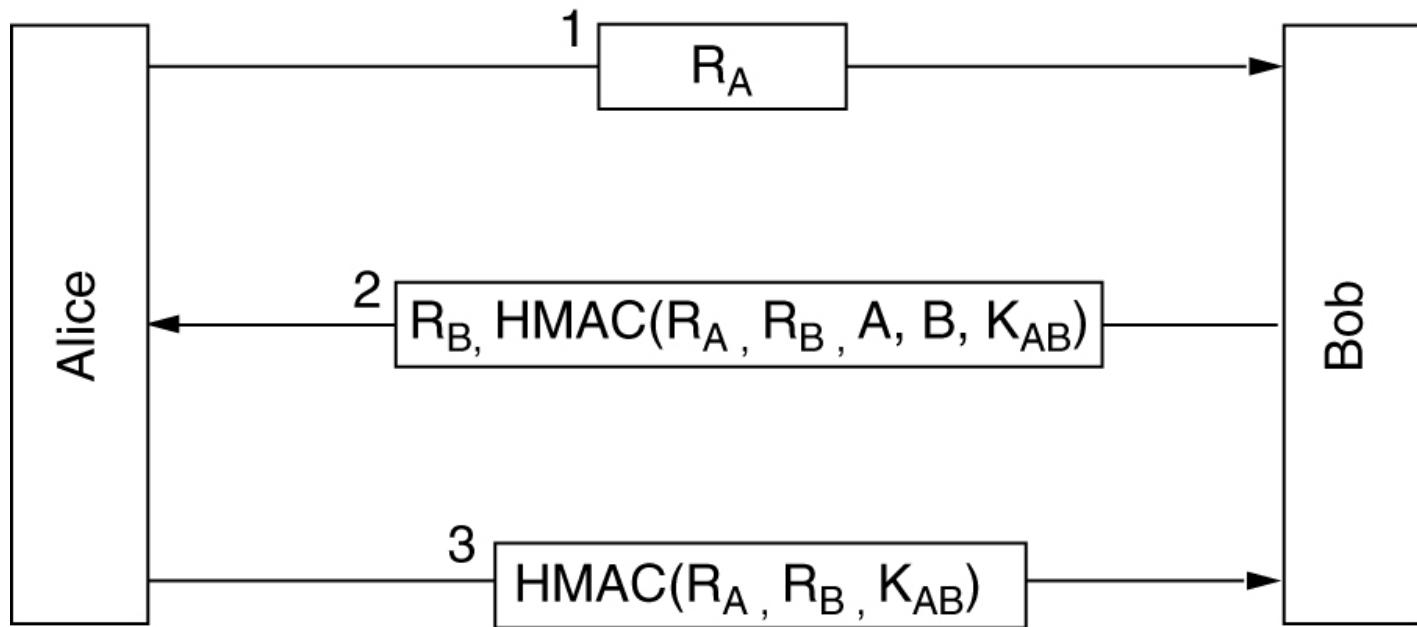
The reflection attack.

Authentication Based on a Shared Secret Key (4)



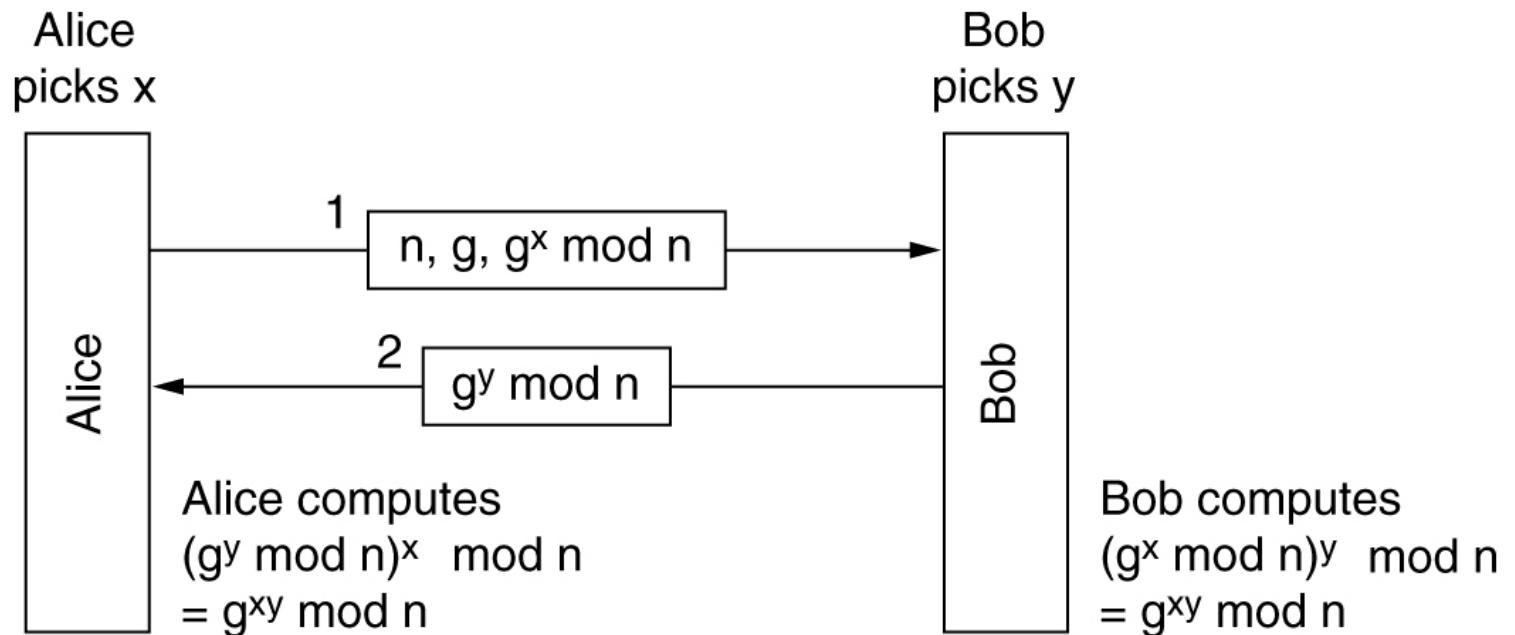
A reflection attack on the protocol of Fig. 8-32.

Authentication Based on a Shared Secret Key (5)



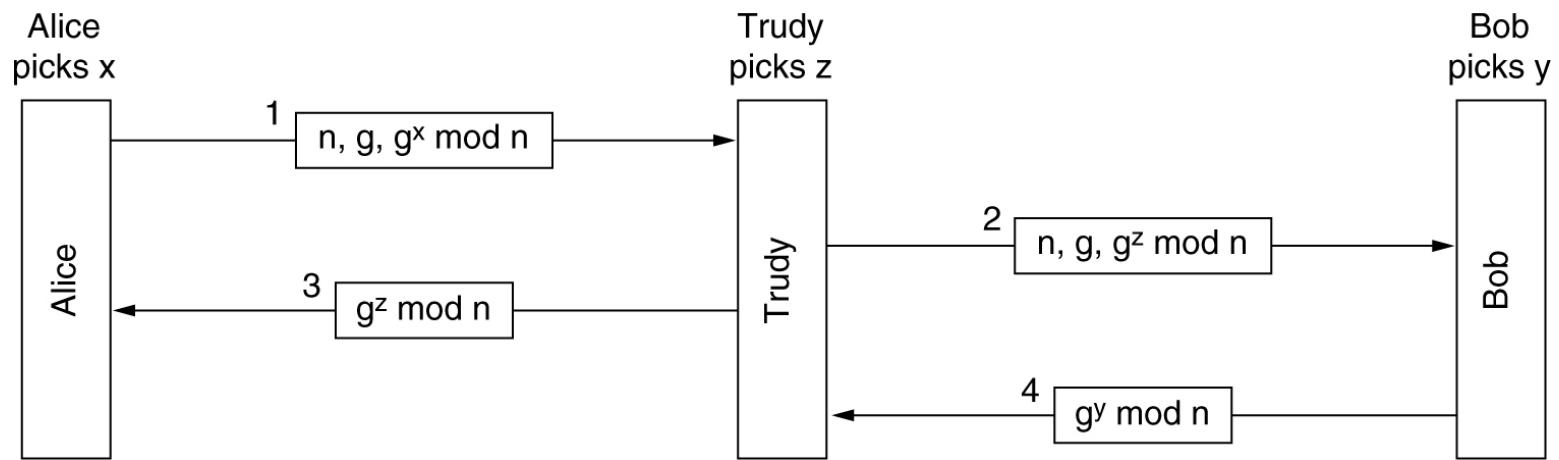
Authentication using HMACs.

Establishing a Shared Key: The Diffie-Hellman Key Exchange



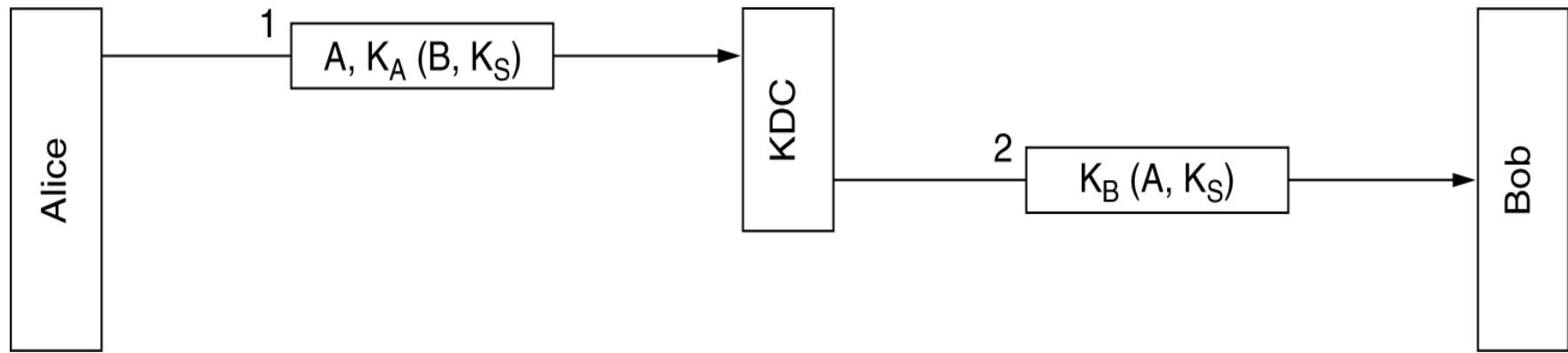
The Diffie-Hellman key exchange.

Establishing a Shared Key: The Diffie-Hellman Key Exchange



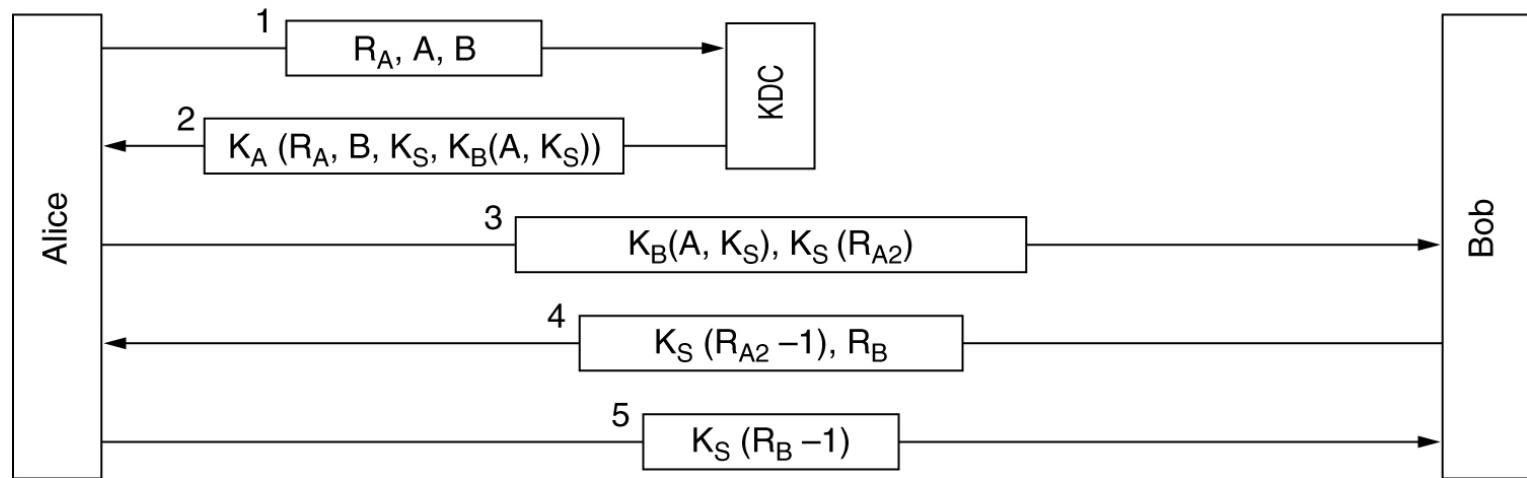
The bucket brigade or man-in-the-middle attack.

Authentication Using a Key Distribution Center



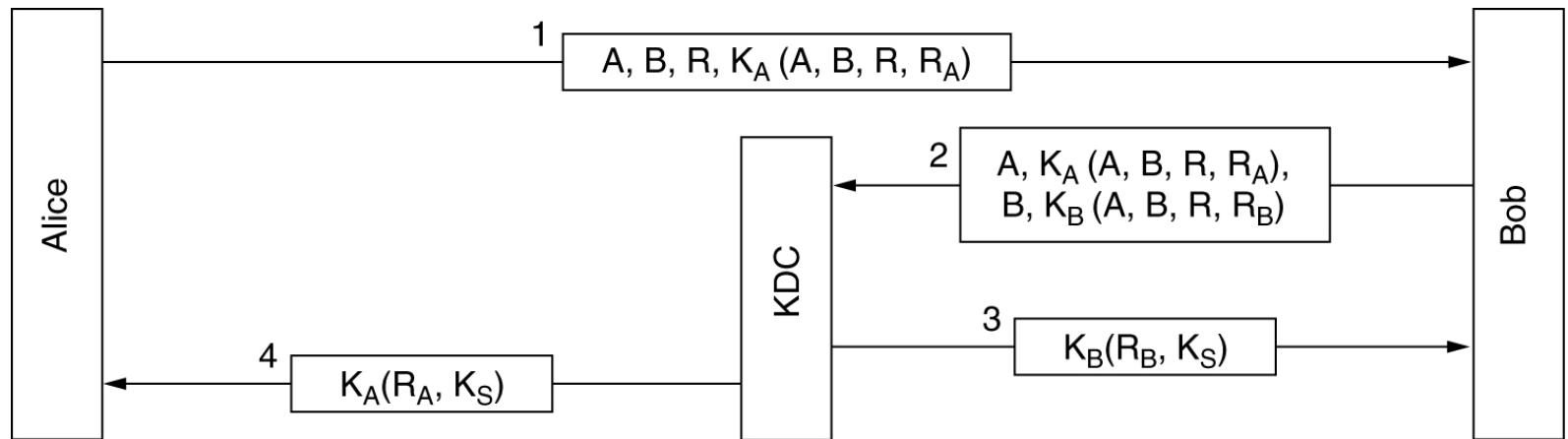
A first attempt at an authentication protocol using a KDC.

Authentication Using a Key Distribution Center (2)



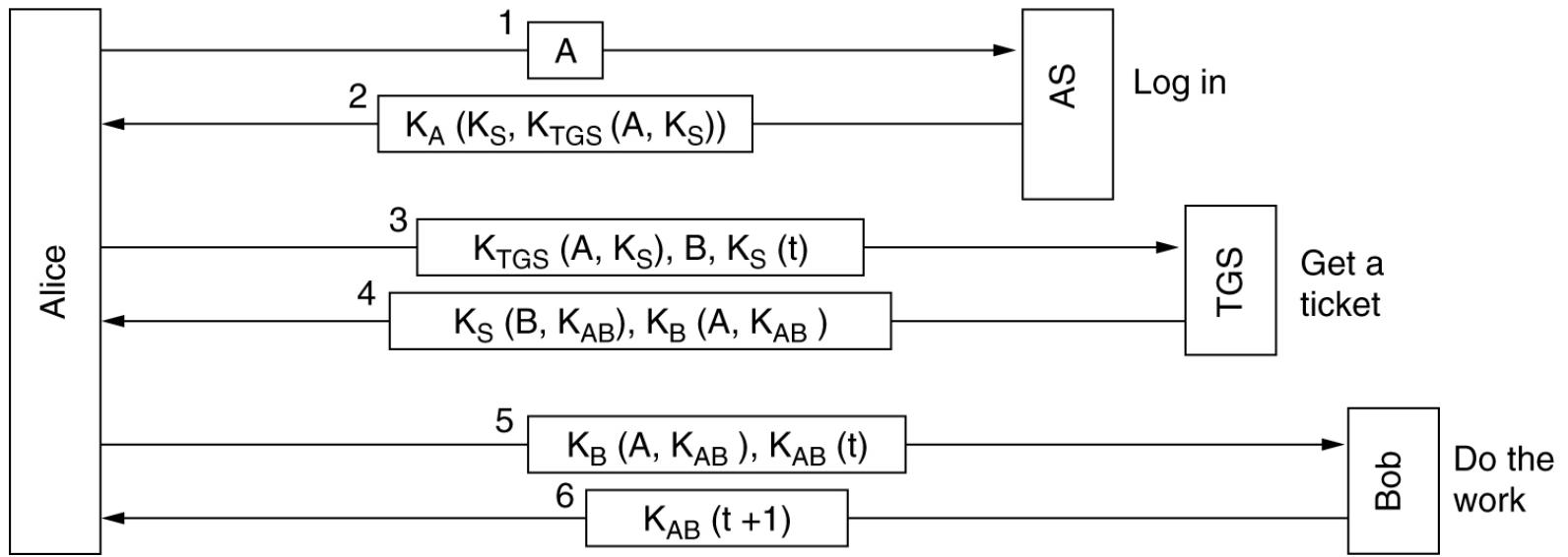
The Needham-Schroeder authentication protocol.

Authentication Using a Key Distribution Center (3)



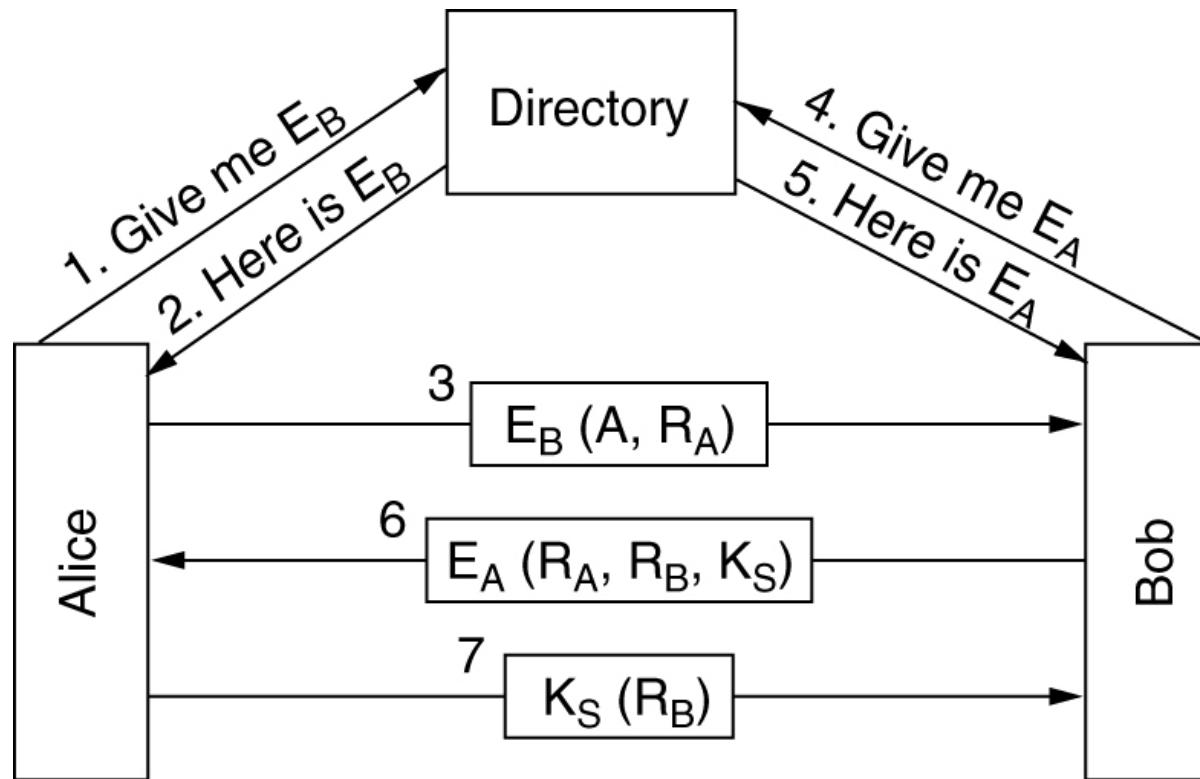
The Otway-Rees authentication protocol (slightly simplified).

Authentication Using Kerberos



The operation of Kerberos V4.

Authentication Using Public-Key Cryptography



Mutual authentication using public-key cryptography.

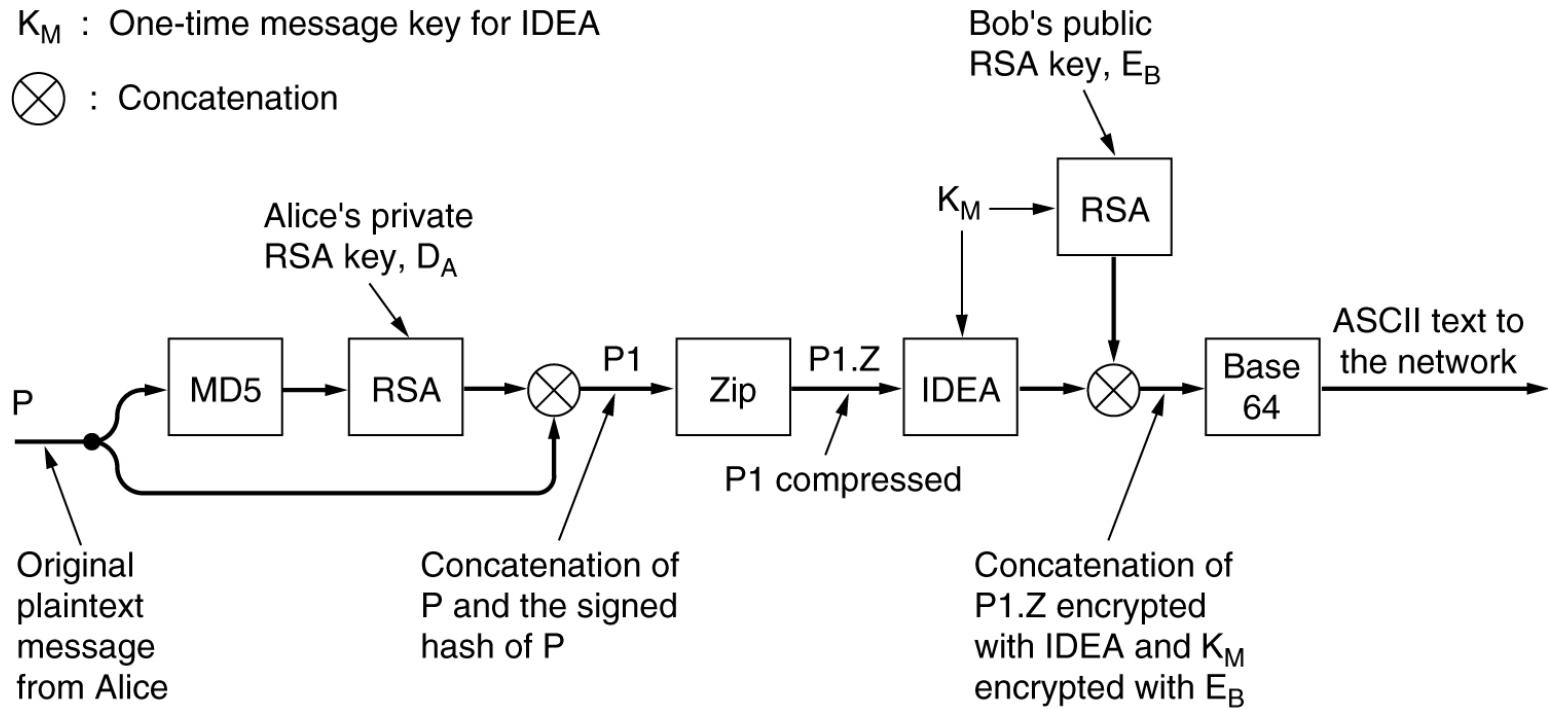
E-Mail Security

- PGP – Pretty Good Privacy
- PEM – Privacy Enhanced Mail
- S/MIME

PGP – Pretty Good Privacy

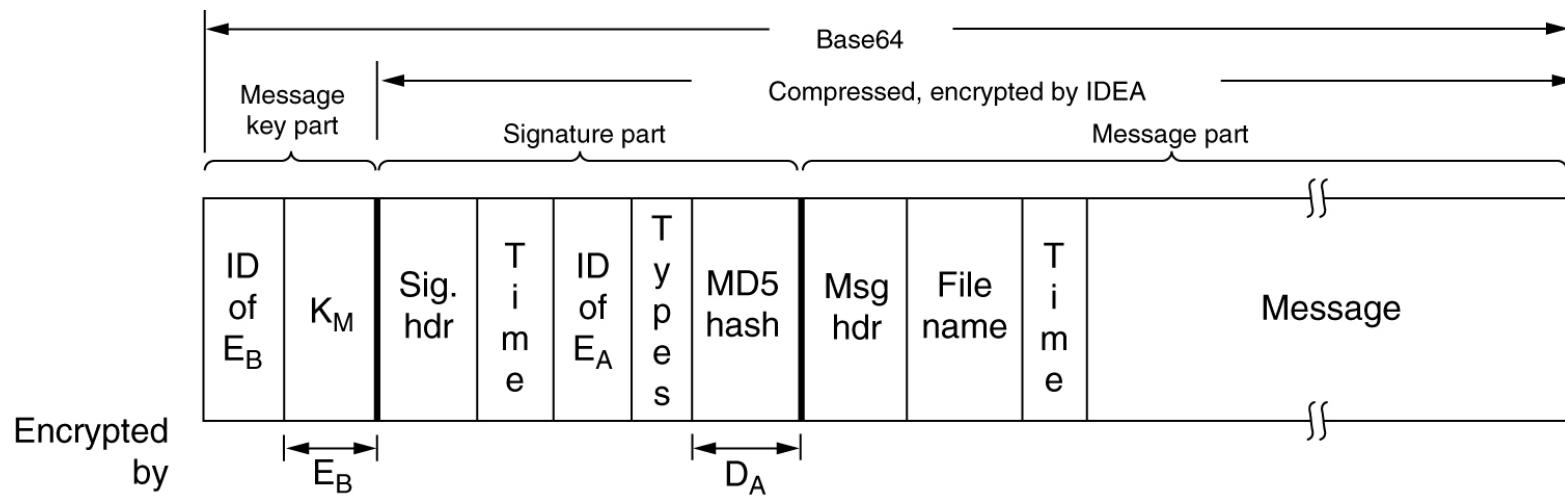
K_M : One-time message key for IDEA

\otimes : Concatenation



PGP in operation for sending a message.

PGP – Pretty Good Privacy (2)

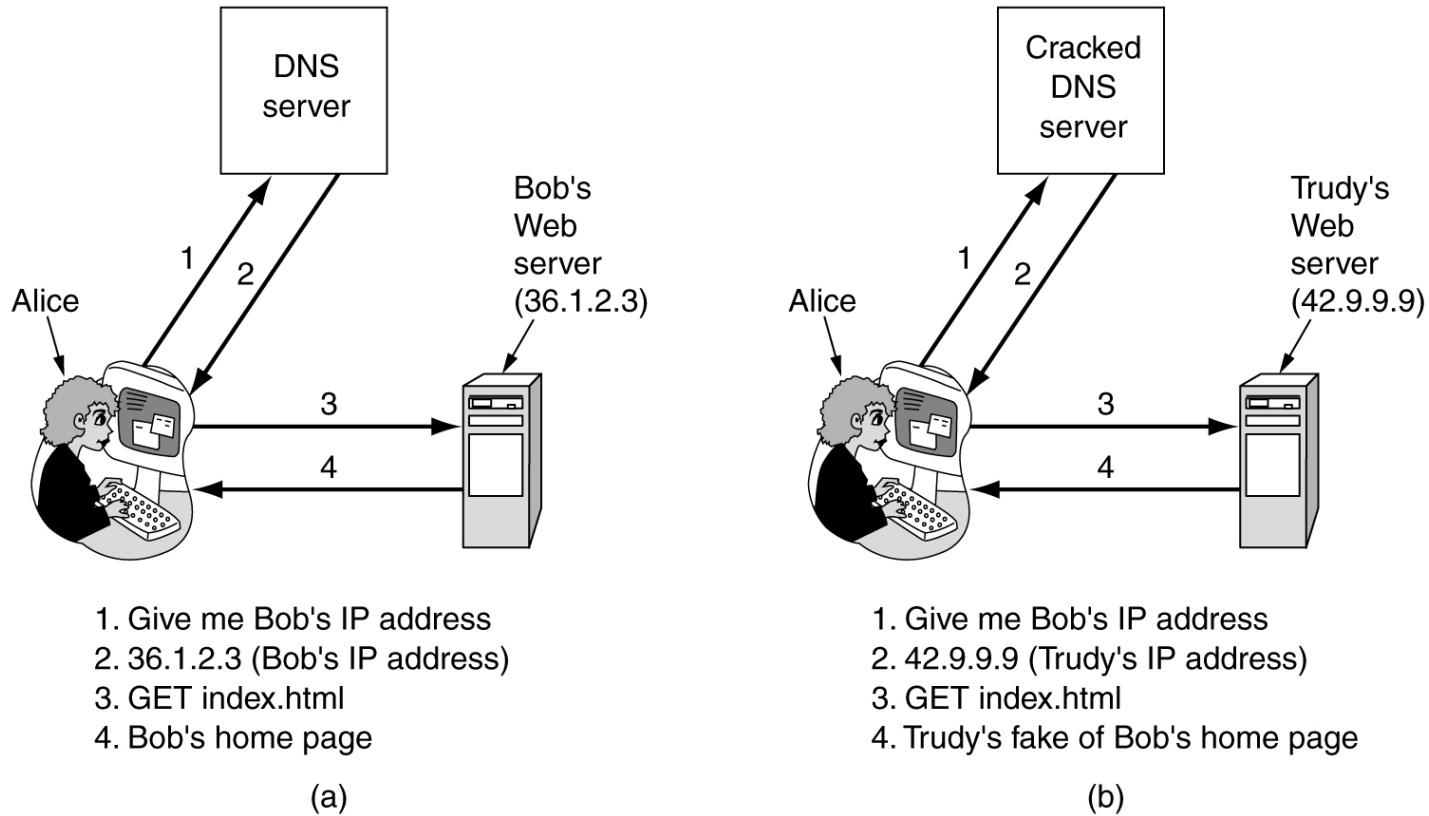


A PGP message.

Web Security

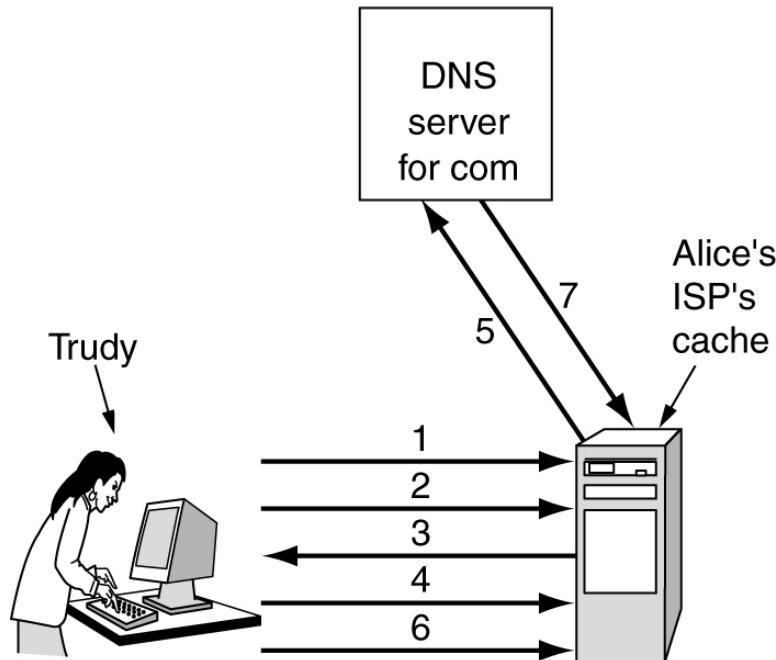
- Threats
- Secure Naming
- SSL – The Secure Sockets Layer
- Mobile Code Security

Secure Naming



(a) Normal situation. **(b)** An attack based on breaking into DNS and modifying Bob's record.

Secure Naming (2)



1. Look up foobar.trudy-the-intruder.com
(to force it into the ISP's cache)
2. Look up www.trudy-the-intruder.com
(to get the ISP's next sequence number)
3. Request for www.trudy-the-intruder.com
(Carrying the ISP's next sequence number, n)
4. Quick like a bunny, look up bob.com
(to force the ISP to query the com server in step 5)
5. Legitimate query for bob.com with seq = n+1
6. Trudy's forged answer: Bob is 42.9.9.9, seq = n+1
7. Real answer (rejected, too late)

How Trudy spoofs Alice's ISP.

Secure DNS

Domain name	Time to live	Class	Type	Value
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C...

An example RRSet for *bob.com*. The *KEY* record is Bob's public key. The *SIG* record is the top-level *com* server's signed has of the *A* and *KEY* records to verify their authenticity.

Self-Certifying Names

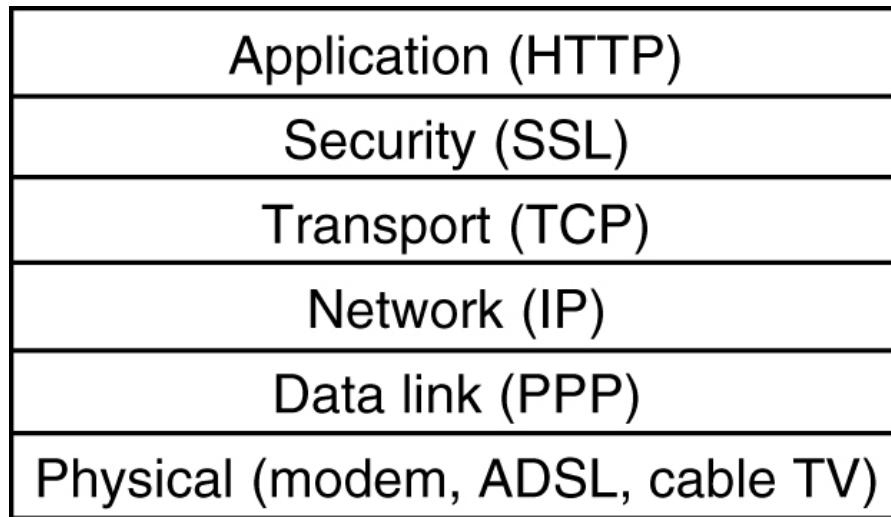
Server SHA-1 (Server, Server's Public key) File name

The URL http://www.bob.com:2g5hd8bfjkc7mf6hg8dgany23xds4pe6/photos/bob.jpg is divided into three segments by curly braces above the URL. The first segment, 'http://www.bob.com:', is labeled 'Server'. The second segment, ':2g5hd8bfjkc7mf6hg8dgany23xds4pe6', is labeled 'SHA-1 (Server, Server's Public key)'. The third segment, '/photos/bob.jpg', is labeled 'File name'.

http://www.bob.com:2g5hd8bfjkc7mf6hg8dgany23xds4pe6/photos/bob.jpg

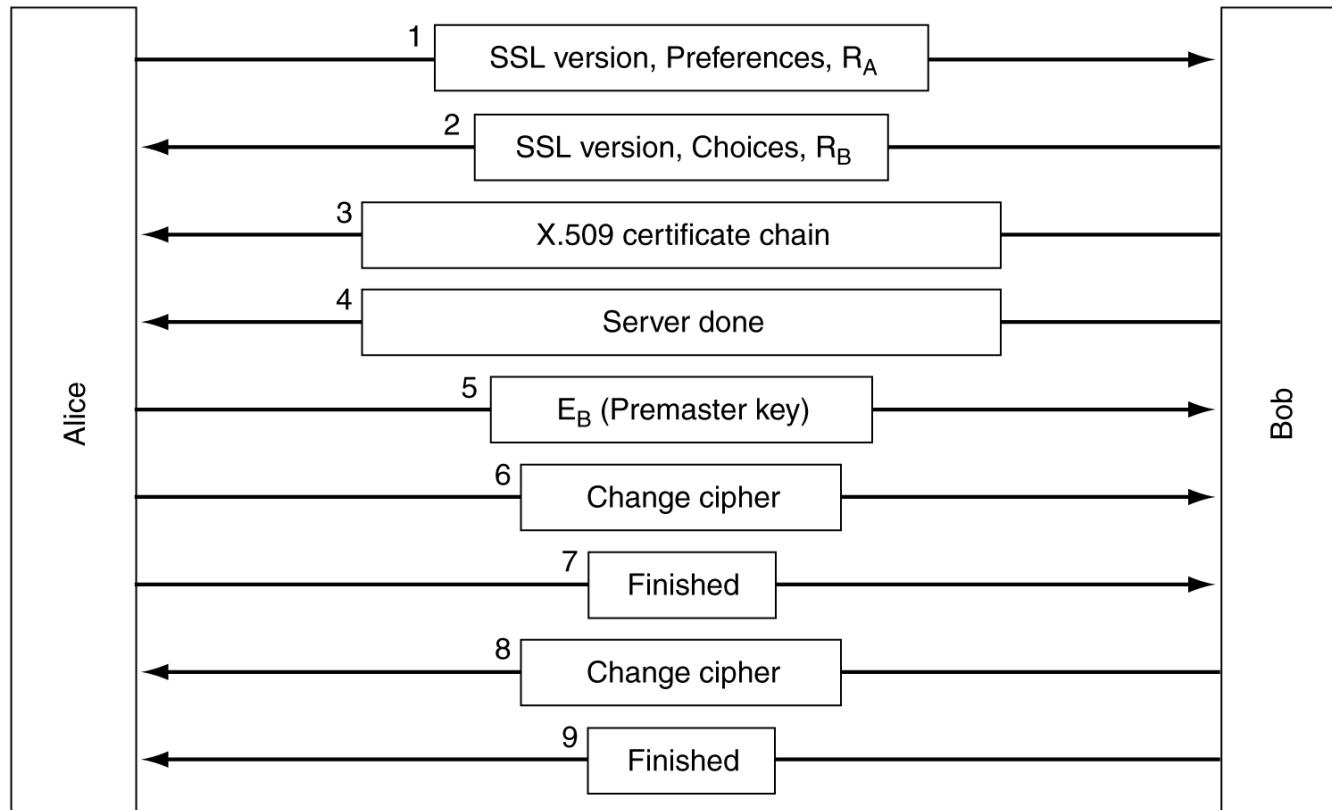
A self-certifying URL containing a hash of server's name and public key.

SSL—The Secure Sockets Layer



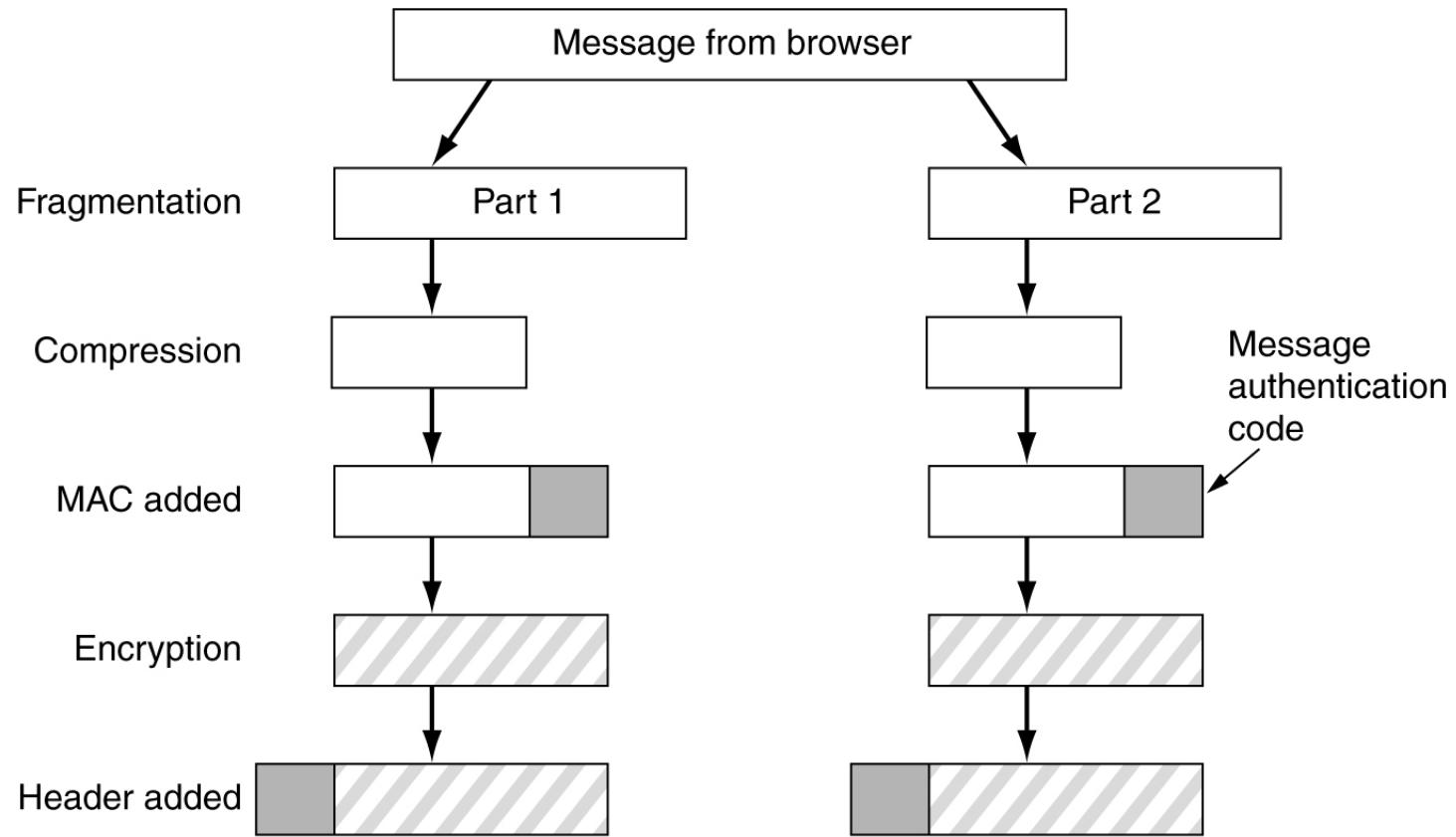
Layers (and protocols) for a home user browsing with SSL.

SSL (2)



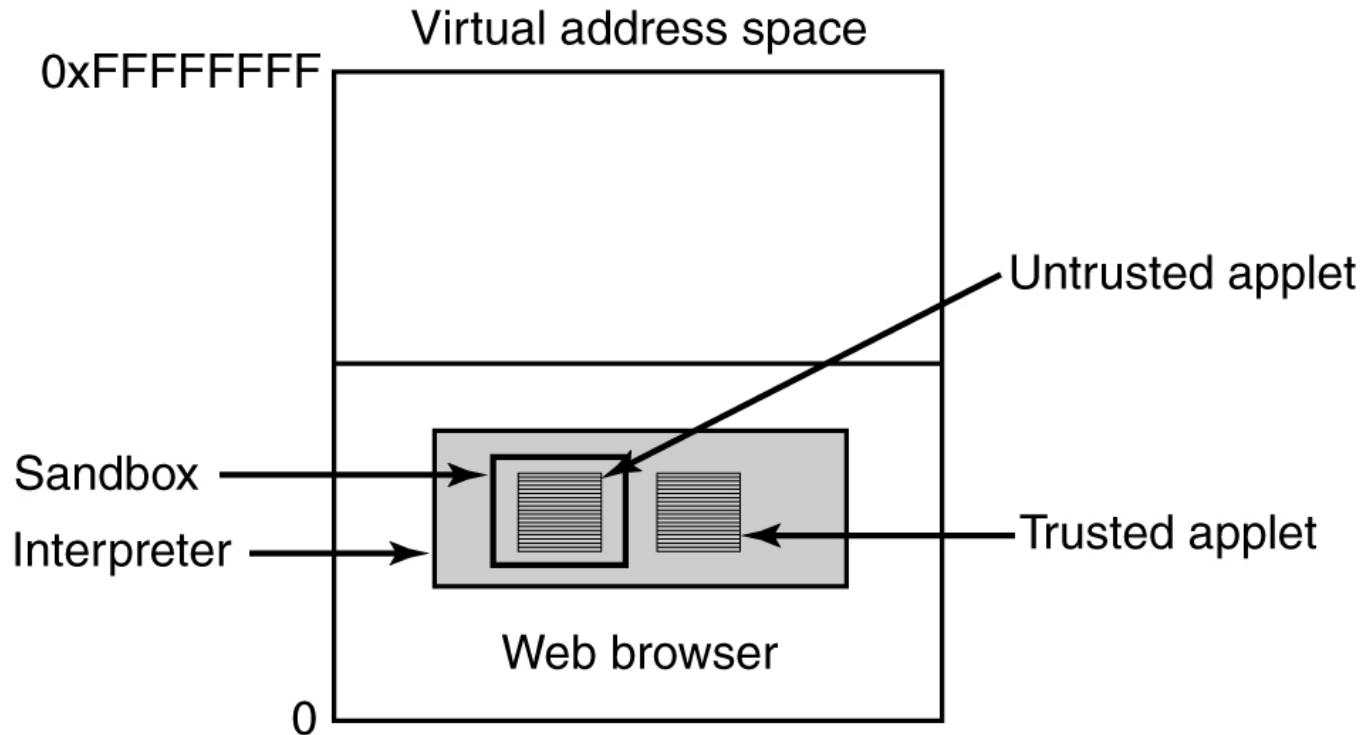
A simplified version of the SSL connection establishment subprotocol.

SSL (3)



Data transmission using SSL.

Java Applet Security

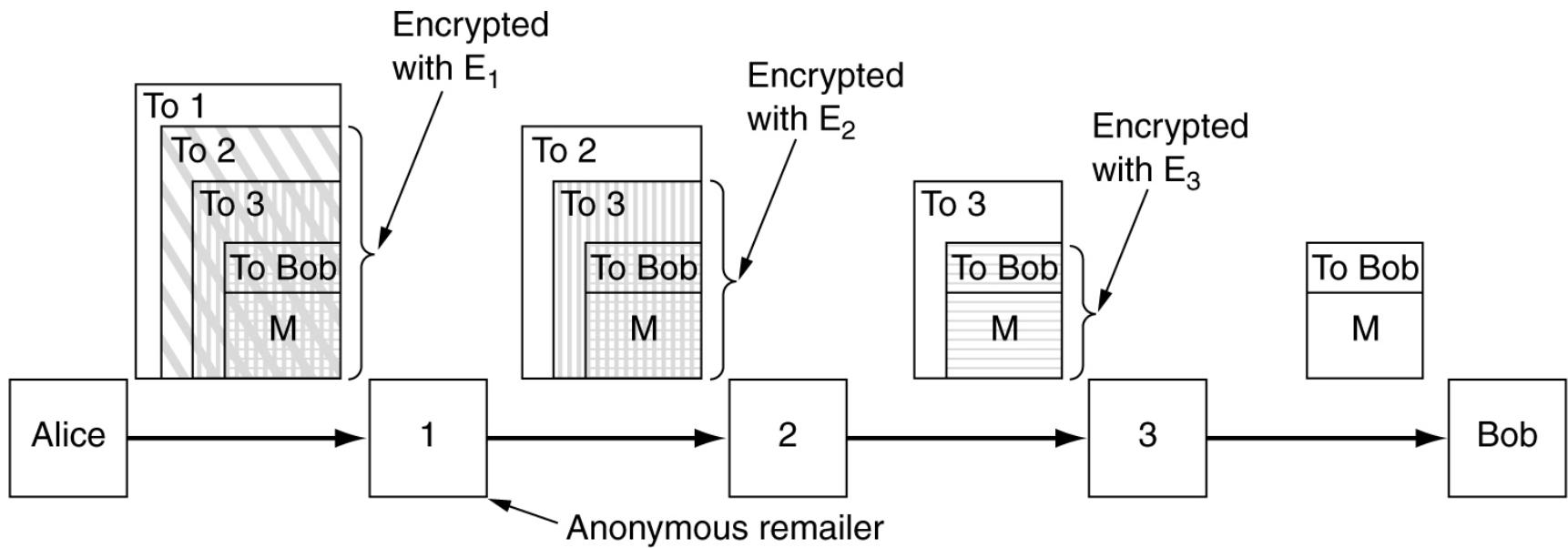


Applets inserted into a Java Virtual Machine
interpreter inside the browser.

Social Issues

- Privacy
- Freedom of Speech
- Copyright

Anonymous Remailers



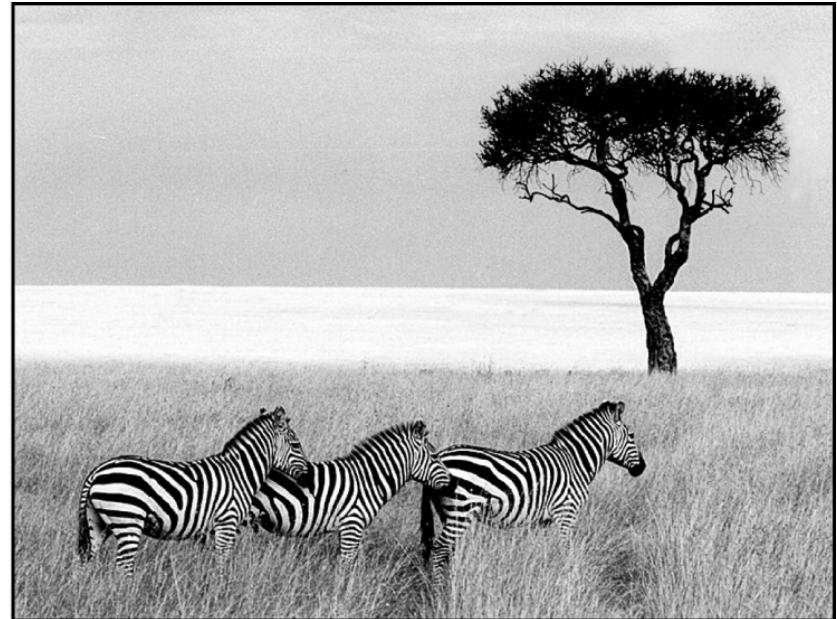
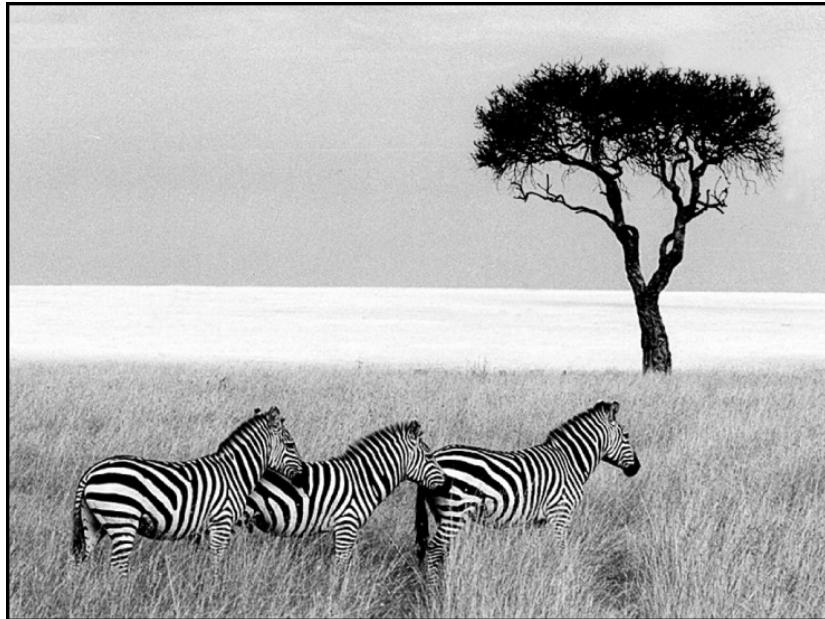
Users who wish anonymity chain requests through multiple anonymous remailers.

Freedom of Speech

Possibly banned material:

2. Material inappropriate for children or teenagers.
3. Hate aimed at various ethnic, religious, sexual, or other groups.
4. Information about democracy and democratic values.
5. Accounts of historical events contradicting the government's version.
6. Manuals for picking locks, building weapons, encrypting messages, etc.

Steganography



- (a) Three zebras and a tree. (b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

