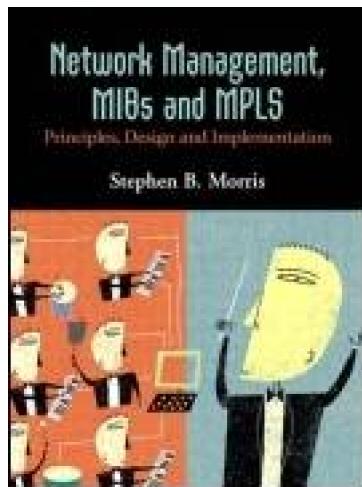


[Team LiB]

NEXT ►



- [Table of Contents](#)
- [Index](#)

Network Management, MIBs and MPLS: Principles, Design and Implementation

By [Stephen B. Morris](#)

Start Reading ►

Publisher: Addison Wesley

Pub Date: June 20, 2003

ISBN: 0-13-101113-8

Pages: 416

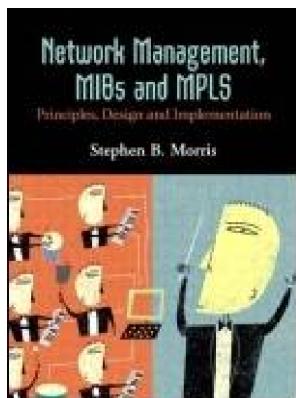
Network Management, MIBs and MPLS: Principles, Design and Implementation is the definitive guide to managing and troubleshooting enterprise and service provider networks. This in-depth tutorial from networking expert Stephen Morris delivers clear and concise instruction on networking with MIBs, SNMP, MPLS, and much more. Coverage includes SNMPv3, network management software components, IP routing, HP Openview Network Node Manager, NMS software components, among other key techniques and tools for managing large network systems.

[Team LiB]

NEXT ►

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)



- [Table of Contents](#)
- [Index](#)

Network Management, MIBs and MPLS: Principles, Design and Implementation

By [Stephen B. Morris](#)

[Start Reading ▶](#)

Publisher: Addison Wesley

Pub Date: June 20, 2003

ISBN: 0-13-101113-8

Pages: 416

[Copyright](#)

[Acronyms](#)

[Foreword](#)

[Preface](#)

[Intended Audience](#)

[Purpose of This Book](#)

[Using This Book](#)

[Linked Overviews](#)

[Source Code Location](#)

[The Four Ms](#)

[Outline of the Book](#)

[A Note About Abbreviations](#)

[Additional Resources](#)

[Acknowledgments](#)

[Chapter 1. Large Enterprise Networks](#)

[Managing Enterprise Networks](#)

[Why Use Network Management?](#)

[SNMP: The De Facto Network Management Standard](#)

[Summary](#)

[Chapter 2. SNMPv3 and Network Management](#)

[SNMPv3 Structure](#)

[SNMPv3 Applications](#)

[SNMPv3 Message Formats](#)

[Network Elements](#)

[Introducing MPLS: First Chunk](#)

[The Trend Towards IP](#)

[MPLS Concepts](#)

[Summary](#)

[Chapter 3. The Network Management Problem](#)

[Bringing the Managed Data to the Code](#)

[Scalability: Today's Network Is Tomorrow's NE](#)

[MIB Note: Scalability](#)

[Light Reading Trials](#)

[Large NEs](#)

[Expensive \(and Scarce\) Development Skill Sets](#)

[Linked Overviews](#)

[Elements of NMS Development](#)

[Expensive \(and Scarce\) Operational Skill Sets](#)

[MPLS: Second Chunk](#)

[MPLS and Scalability](#)

[Summary](#)

[Chapter 4. Solving the Network Management Problem](#)

[Filling the Development Skills Gap](#)

[Smarter NMS](#)

[Smarter MIBs](#)

[One Data Model](#)

[Smarter NEs](#)

[Policy-Based Network Management \(PBNM\)](#)

[Network Management Policies](#)

[Directory-Enabled Networking \(DEN\)](#)

[IP QoS and the Enterprise](#)

[Summary](#)

[Chapter 5. A Real NMS](#)

[HP OpenView Network Node Manager \(NNM\)](#)

[Network Discovery and Mapping](#)

[Notification Processing](#)

[Reporting](#)

[Data Warehousing](#)

[Backup and Restore of Firmware and Configuration](#)

[Java Interface for Remote Access](#)

[MIB Support Features](#)

[MPLS Support](#)

[Policy Support](#)

[Reliability Features](#)

[Integration with Other Software](#)

[Programmability](#)

[Workflows and Business Processes](#)

[Applications of NMS](#)

[The Network Is the True Database](#)

[The Network Boundary](#)

[Summary](#)

[Chapter 6. Network Management Software Components](#)

[Fault Server](#)

[Configuration Server](#)

[Accounting Server](#)

[Performance Server](#)

[Security Server](#)

[Other Servers](#)

[Summary](#)

[Chapter 7. Rudimentary NMS Software Components](#)

[Building a Rudimentary Management System](#)

[Configuring SNMP on Windows NT/2000](#)

[Setup Required for the Visual C++ Program](#)

[Building the Sample Visual C++ SNMP Manager](#)

[The Structure of the Sample Visual C++ Program](#)

[Using the Rudimentary Management System](#)

[A Note On Security](#)

[The Sample JDMK Java SNMP Manager](#)

[Building the Sample Java Manager](#)

[Extending the Sample SNMP Programs](#)

[Summary](#)

[Chapter 8. Case Study: MPLS Network Management](#)

[The \(Internet Draft\) Standard MPLS MIBs](#)

[Configuring LSPs Through an LSR](#)

[Creating an LSP Using the LSR MIB](#)

[Creating a Tunnel Using the TE MIB](#)

[Creating LSPs and Tunnels Using an NMS](#)

[NextObjectIndex and Synchronization](#)

[A Note About Standards](#)

[Summary](#)

[Chapter 9. Network Management Theory and Practice](#)

[MIBs Again](#)

[Intelligence in the Network: Manufacturing](#)

[Pushing FCAPS into the Network](#)

[Service-level Network Components](#)

[Generic Objects Realized Using Software Abstraction](#)

[The Need for End-to-End Security](#)

[Shrink-Wrapped Solutions or Consultancy Buy-In](#)

[Integration with OSS Layers: Northbound Interface \(NBI\)](#)

[Developer Note: The Roles of QA, IT, and Developers](#)

[Solutions Revisited: Thin Software Layers](#)

[Facilitating a Solution Mindset](#)

[A Final Word](#)

[Appendix A. Terminal Server Serial Ports](#)

[Appendix B. Some Simple IP Routing Experiments](#)

[Section B.1. The IP Routing Table](#)

[Section B.2. Ping](#)

[Section B.3. Traceroute](#)

[Appendix C. The IP MIB Table](#)

[MIB Walk on the IP Table](#)

[Appendix D. Visual C++ Sample Program Source Code](#)

[Section D.1. snmpdefs.h](#)

[Section D.2. snmpmgr.c](#)

[Appendix E. JDMK Sample Program Source Code](#)

[Section E.1. synchronousManager.java](#)

[References](#)

[Glossary](#)

[Index](#)

[\[Team LiB \]](#)

[!\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\) PREVIOUS](#) [!\[\]\(5f42d2cd7ad901bc24e5d35a38c777fd_img.jpg\) NEXT !\[\]\(628bc0b1ef2b63d1fc4442fb794e3e78_img.jpg\)](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Copyright

Library of Congress Cataloging-in-Publication Data available

Editorial/production supervision: *Kathleen M. Caren*

Acquisition Editor: *Mary Franz*

Editorial Assistant: *Noreen Regina*

Marketing Manager: *Dan DePasquale*

Manufacturing Manager: *Maura Zaldivar*

Cover Design Director: *Jerry Votta*

Interior Design: *Gail Cocker-Bogusz*

© 2003 Published by Pearson Education, Inc.

Publishing as Prentice Hall Professional Technical Reference

Upper Saddle River, NJ 07458

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.

For information regarding corporate and government bulk discounts please contact: Corporate and Government Sales (800) 382-3419 or corpsales@pearsontechgroup.com

Other company and product names mentioned herein are the trademarks or registered trademarks of their respective owners.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10987654321

Pearson Education LTD.

Pearson Education Australia PTY, Limited

Pearson Education Singapore, Pte. Ltd.

Pearson Education North Asia Ltd.

Pearson Education Canada, Ltd.

Pearson Educación de Mexico, S.A. de C.V.

Pearson Education—Japan

Pearson Education Malaysia, Pte. Ltd.

Pearson Education, Upper Saddle River, New Jersey

Dedication

To Siobhán

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Acronyms

API	Application Programming Interface
ASN.1	Abstract Syntax Notation Number One
AS	Autonomous System
BGP4	Border Gateway Protocol
CDR	Call Detail Record
CIM	Common Information Model
CLI	Command-Line Interface
CNC	Computer Numerical Control Machine
CO	Central Office
COM	Component Object Model
COS	Class of Service
COTS	Commercial Off-The-Shelf
CPE	Customer Premises Equipment
CTI	Computer Telephony Integration
DECT	Digital Enhanced Cordless Telecommunications
DEN	Directory-Enabled Networking
DES	Data Encryption Standard
DLCI	Data Link Connection Identifier
DMI	Desktop Management Interface
DNS	Domain Name System
DS	Differentiated Services
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
DTL	Designated Transit List
DWDM	Dense Wavelength Division Multiplexing
ECN	Explicit Congestion Notification
ELAN	Emulated LAN

EMS	Element Management System
FCAPS	Fault, Configuration, Accounting, Performance, and Security
FEC	Forwarding Equivalence Class
FR	Frame Relay
FTN	FEC-to-NHLFE
GMPLS	Generalized Multiprotocol Label Switching
IANA	Internet-Assigned Numbers Authority
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IN	Intelligent Networking
INTSERV	Integrated Services
IP	Internet Protocol
IS-IS	Intermediate System-Intermediate System
ISDN	Integrated Service Digital Network
ISP	Internet Service Provider
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JDK	Java Development Kit
JDMK	Java Development Management Kit
JIT	Just-in-Time
JVM	Java Virtual Machine
L2TP	Layer 2 Tunneling Protocol
LDAP	Lightweight Directory Access Protocol
LER	Label Edge Router
LSP	Label Switched Path
LSR	Label Switching Router
MAP	Mobile Application Part
MD5	Message Digest 5
MIB	Management Information Base
MPLS	Multiprotocol Label Switching
MTP	Message Transfer Part
MVNO	Mobile Virtual Network Operator
NAP	Network Access Point

NE	Network Element
NGN	Next-Generation Network
NHLFE	Next-Hop Label Forwarding Entry
NIC	Network Interface Card
NMS	Network Management System
NOC	Network Operations Center
OID	Object Identifier
OOD	Object-Oriented Design
ORB	Object Request Broker
OSPF	Open Shortest Path First
OSI	Open Systems Interconnection
OSS	Operations Support System
PBNM	Policy-Based Network Management
PDP	Policy Decision Point
PDR	Performance Data Record
PDU	Protocol Data Unit
PEP	Policy Enforcement Point
PHB	Per-Hop-Behavior
PIB	Policy Information Base
PLC	Programmable Logic Controller
PNNI	Private Network-To-Network Interface
POP	Point-of-Presence
POTS	Plain Old Telephone Service
PSTN	Public Switched Telephone Network
PVC	Permanent Virtual Connection
PVX	Permanent Virtual Connection (Generic term)
QA	Quality Assurance
QoS	Quality of Service
RADIUS	Remote Access Dial-In User Service
RAS	Remote Access Server
RFC	Request for Comment
RMI	Remote Method Invocation
RPC	Remote Procedure Call

SAN	Storage Area Network
SCCP	Signaling Connection Control Part
SDL	Specification and Description Language
SHA1	US Secure Hash Algorithm 1
SIN	Ships-in-the-Night
SLA	Service Level Agreement
SMS	Short Message Service
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
SPPI	Structure of Policy Provisioning Information
SPVCC	Switched Permanent Virtual Channel Connection
SPVX	Generic Switched Permanent Virtual Circuit
SS7	Signaling System Number 7
TCAP	Transaction Capability Application Part
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TL1	Transaction Language One
TLS	Transparent LAN Service
TMN	Telecommunications Management Network
TOM	Telecommunications Operations Map
TOS	Type of Service
UML	Unified Modeling Language
USM	User-based Security Model
VCI	Virtual Channel Identifier
VLAN	Virtual Local Area Network
VoIP	Voice-over-IP
VPI	Virtual Path Identifier
WAN	Wide Area Network
XML	Extensible Markup Language

[Team LiB]

◀ PREVIOUS NEXT ▶

Foreword

This book provides a good introduction to and overview of the challenges involved in managing large enterprise networks and in developing effective software solutions for managing such networks. It focuses on the use of SNMP and MIBs in developing such solutions and uses the example of managing MPLS to illustrate the ideas presented.

Managing large networks is a difficult and expensive endeavor due to the wide variety of deployed technologies and products. While there are many tools available to help ease the burden of network management, there is still a huge amount of work to be done in improving these tools. In addition, new tools must constantly be developed to manage new technologies and products as they are introduced into networks.

Network administrators must be able to effectively evaluate and select tools and solutions to aid in managing their networks. As new technologies and products are introduced into their networks, administrators must understand how they are to be managed so that they can communicate the requirements for managing these new technologies and products to developers. To accomplish all this effectively, they must have a broad understanding of both the technologies and products being managed and of how management tools are designed and built.

Similarly, developers must have a good understanding of the challenges facing network administrators in managing their networks and how these challenges drive up the cost of network management. This understanding is vital to the development of better, more cost-effective tools.

Network Management, MIBs, and MPLS: Principles, Design, and Implementation begins with an overview of many of the technologies in common use in networks today and then discusses what it means to manage a network. It also gives an overview of SNMP and MIBs. SNMP is currently the most widely used protocol for network management. Having a general understanding of SNMP and MIBs is very useful in understanding network management in general. The book provides enough information about SNMP to give the reader a basic understanding of the protocol. It then discusses some of the challenges involved in network management along with some guidelines and strategies for dealing with these challenges.

The book offers a detailed discussion of how network management tools are designed, using examples of commercial and homegrown tools and the specific example of how the MPLS MIBs are used to manage MPLS. It concludes with some comments about current directions in the field of network management.

In order to address the challenges of network management, it is important for network administrators and developers of the tools they use to cooperate closely. In order to accomplish this, both must have a common understanding of the difficulties faced by each other. This book should help develop this common understanding and cooperation, and will provide the reader with a good overall understanding of the field of network management.

David Levi
Senior Staff Software Engineer, Nortel Networks
February 2003

[Team LiB]

◀ PREVIOUS NEXT ▶

Preface

The last two decades have been dominated by distinct patterns of computer use. The 1980s saw wide-scale deployment of PC- and microprocessor-based technology. The 1990s saw this infrastructure becoming internetworked, connected to the Internet, and increasingly embracing client/server technology. Initially, clients were heavy duty (or fat) and communicated with local servers, but by the end of the 1990s clients had become thin and servers were increasingly distributed. The first decade of the 21st century may well be one of global system (as well as network) integration and management during which we will see previously disparate networks and systems interconnected for new purposes. Nowhere is this more apparent than in the area of telecommunications and data networking.

The relentless growth and extended reach of both enterprise and service provider (SP) networks have been accompanied by an increased demand for advanced vendor-independent network management software tools. This is particularly the case as enterprises leverage their network investments by deploying evermore advanced, mission-critical systems like voice-over-IP and desktop video conferencing applications [[ATM&IP2001](#)]. At the same time, service providers are consolidating and deploying Multiprotocol Label Switching (MPLS) cores and IP services, such as IP VPNs, as part of their migration path to an end-to-end packet-based infrastructure [[Alcatel2001](#)].

In many ways the managers of enterprise networks face a daunting task because of the sheer diversity of network elements (NE) and systems—multi-vendor routers, switches, leased lines, WANs, VLANs, Storage Area Networks (SANs), mobile and desktop telephony, PABXs, soft switches [[Sweeney2001](#)], databases, a wide range of software applications, NT/Windows 2000/Unix servers, minicomputers, mainframes, and so on. These systems are the data lifeblood of modern corporations, and their continuous availability is crucial. Unfortunately, most of these enterprise NEs have their own proprietary management tools, which have to be learned and maintained over time, adding to the cost of ownership. Proprietary systems (a bad thing) and centralized, automated management (a good thing) are, in general, mutually exclusive.

Enterprises have a lot of legacy systems and equipment, which must be depreciated over many years. Forklift upgrades (getting rid of all the old stuff and putting in the latest) are generally too expensive, so system and network management skill sets must be present throughout the lifecycle. This means that enterprises will continue to be a highly complex network management proposition for a long time to come. Complex management equates to high operational cost. This book proposes that all network-based systems produced for the enterprise market should:

- Provide MIB module files that describe the principal managed objects.
- Conform to or extend standard MIBs.
- Provide any proprietary MIBs in text file or downloadable form at the time of purchase.
- Guarantee as far as possible that the management facilities will be available during periods of high traffic or even congestion.
- Include high-performance agents that can issue useful notifications and execute both read and write operations against their MIBs.
- Deploy SNMPv3 agents (entities).
- Provide simple scripts for reading from and writing to their MIBs.
- Provide snap-in modules (Java/C++) to allow easy integration with existing network management systems.
- Support automation via SNMP of routine administrative tasks such as adding users to a VLAN or disks to a SAN.

Network managers should insist on these minimum requirements before making purchasing decisions. On the supply side of the industry, vendors should from day one build standards-based (SNMPv3) management facilities into their networking products. Many vendors leave the management infrastructure (agents, MIBs, etc.) development until quite late in the development cycle. This can result in poor-quality agents and MIBs, ultimately reducing the manageability of the NEs and the wider network.

Several equipment vendors have a large number of different element management systems (EMS) to manage just their own equipment. Since the EMS often forms the basis for the NMS, this multiplicity of different EMS can make NMS software harder to develop, particularly in multivendor networks. A single EMS across all NEs is a better proposition.

Another major theme of this book is the migration of networks toward a layer 3 model based on the IP protocol. This is a mega-trend affecting pretty much any industry involved in moving data from one networked location to another. Management systems for layer 3-converged (voice, video, and data) networks are an increasingly important issue. We use MPLS as a running example of how networks are evolving in terms of providing quality of service, traffic engineering, and so on.

[Team LiB]

◀ PREVIOUS

NEXT ▶

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

Intended Audience

This book provides a practitioner's approach to understanding the area of network management. The only prerequisites are a reasonable understanding of network technology and a passing familiarity with SNMP. The book is suitable for:

- Network management software developers.
- Software developers considering a move into the area of network management system development.
- Network managers seeking a deeper insight into the area of network management.
- Network equipment vendors.
- Enterprise and SP networking professionals.
- Standards bodies producing MIBs for new technologies, such as the IETF, and industry-wide technology advancement groups such as the MPLS Forum (among others).
- Students taking courses in telecommunications, computer science, or network management.

Network Management, MIBs, & MPLS: Principles, Design, & Implementation provides much discussion of networks, MIBs, management software, and managed objects. Important points that are relevant to MIB authors are indicated by special sections entitled "MIB Notes."

We also include "Developer Note" sections that are of primary concern to software developers. Readers seeking an introductory overview can safely skip these few marked sections.

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Purpose of This Book

This book is not a detailed description of the major versions of SNMP (1, 2c, and 3). Many other books do a good job of this. Our focus is on the use of SNMP technology for managing networks. It also attempts to tackle the complexities faced by the developers of NMS software products. MIBs are a crucial element of this for modeling the operation of large networks.

The field of network management is extremely broad with a vast range of products from many companies. This book is intended primarily as a learning aid for hard-pressed engineers tasked with software development or development and maintenance of complex networks and management systems. The book is also a guide to adopting enhanced approaches to both NMS/MIB requirements definition and NMS/MIB development. It has no affiliation with any vendor or technical organization. Any mention of technologies—IP, MPLS, ATM, Frame Relay, VLANs, Ethernet, and so on—is intended purely as a teaching tool to illustrate network management principles and to place the latter in a modern and interesting context.

We use a big-picture approach and try to give a reasonable overall description of managed networks. In this context, network management should be seen as a type of abstraction; that is, it seeks to manage networks, not provide a comprehensive understanding of all the constituent technology (e.g., IP, ATM, and MPLS).

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Using This Book

We hope our readers will gain a solid foundation for understanding the principles and practice of NMS use and development. The networking industry is highly dynamic, so referring to specific products tends to quite quickly date a book. Vendor devices and software management products come and go, but concepts tend to have a longer shelf life. For this reason, we focus mostly on principles and concepts with reference to important RFCs [[IETFWeb](#)]. The major exception to this is in[Chapter 5](#), "A Real NMS," where the highly successful HP OpenView Network Node Manager is described. Even in this case, however, we describe generic network management areas and then see how HP supports them. Also, some mention is made of SNMP software development tools. The book can be used to:

- Assist network managers in defining management requirements for their equipment and system suppliers.
- Gain a better understanding of the relationship between network management and cost of ownership.
- Bring network management needs to the top of the priority list for NE software and hardware development engineers.
- Encourage a sound approach to development on the part of management system vendors.
- Locate Internet resources on MIBs and network management.
- Learn how to build and extend a rudimentary SNMP-based management system.
- Get a good look inside big networks from a management perspective rather than to merely present the constituent technologies.

We hope that the book gives an integrated overview of network management issues. This includes an understanding of both management system technologies and trends in NEs. Much of the technology mentioned (MPLS, VLANs, etc.) is described in Andrew S. Tanenbaum's *Computer Networks* [[Tanenbaum2003](#)], which also has much useful detail concerning the various standards organizations.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Linked Overviews

We introduce a four-step, fast-track technique in [Chapter 3](#), "The Network Management Problem," for gaining an overview of a given NE. Briefly, this technique is as follows:

1. Review the NE technology.
2. Use the device EMS.
3. Learn the MIBs.
4. Write some source code to manipulate the MIBs.

Clearly, software developers need all three steps, but steps 1 and 2 could also be useful for project managers, marketing executives, and others.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Source Code Location

In order to help solidify the concepts discussed, [Chapter 7](#), "Rudimentary NMS Software Components," includes the source code of two programs, one written in Visual C++ and the other in Java. The example source code is freely available from the Prentice Hall Web site, <http://authors.phptr.com/morris>.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

The Four Ms

Many existing books do an excellent job of describing SNMP, MIBs, proxies, and other technologies. While this book describes SNMP (including version 3), its primary focus is more on what might be called the four Ms:

- Manageability of NEs
- MIBs
- Multiservice devices
- MPLS

The manageability of NEs directly affects the cost attached to introducing them into large networks. The quality of the associated MIBs has an important bearing on the cost of introducing the NEs into existing and new management systems. Multitechnology NEs are increasingly the norm, supporting combinations of TCP/IP, MPLS, ATM, Frame Relay, Ethernet, SONET, DWDM, and so on. MPLS and GMPLS are now part of both the strategy and production environment of many service providers. MPLS is also finding its way into the WANs of some very large enterprises. MPLS is such an important technology that it forms a common thread throughout this book.

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

Outline of the Book

[Chapter 1](#) presents a general overview of modern networks and introduces the area of management. Reference is made to sample enterprise and SP networks. VLAN technology and layer 3 are introduced, and the difference between ports and interfaces is described. The importance of network management is discussed and includes a brief explanation of the areas of network management. The network management pyramid is introduced to show the way different management system software layers can be deployed. Alternative techniques to SNMP are described. Aggregate network-resident objects are described, followed by the overall goal of an NMS. A closer look is taken at the elements of SNMP.

[Chapter 2](#) describes some of the details of SNMPv3 message content. Some SNMPv3 message interactions are described with reference to an example network. Some of the problems associated with SNMP are described. The different versions of SNMP in common use are enumerated, followed by an introduction to the area of SNMP applications. A closer examination of a MIB is made to reveal the general structure of all MIBs. This is followed by a brief comparison between NMS software and more familiar applications. The generic structure of a network device is described, and an introductory section on MPLS is included.

[Chapter 3](#) describes the fundamental operational problem in network management: scalability. The other major problem is a severe skills shortage among management system developers. The traditional approach to development of being able to specialize in application (high-level) development as opposed to system (low-level) development doesn't seem to fit the network management model. Developing management software requires a rarified mixture of skills, and some of these are enumerated and described. Likewise, operating and maintaining modern networks require a hard-to-find mixture of knowledge of layers 1, 2, and 3.

[Chapter 4](#) presents some strategies for solving the network management problems identified in [Chapter 2](#). This includes augmenting development skill sets, smarter management systems, smarter MIBs, smarter devices, good data models, distribution, policy, and directories. The distribution of management system servers is one possible approach to solving compute-intensive bottlenecks. The emerging area of policy-based techniques for network management is introduced, followed by a discussion of directory-enabled networking. IP Differentiated Services architecture model is introduced in the context of the ongoing MPLS discussion.

[Chapter 5](#) presents a description of a widely used network management system. HP OpenView Network Node Manager is introduced in terms of its ability to discover networks and process notifications. Issues such as the bringing up and down of large networks are described, and the FCAPS areas are revisited. The important issue of visually depicting a network state is then described, followed by a discussion of client-side software.

[Chapter 6](#) presents the internal software that makes up a network management system. This includes servers that talk to the network devices, clients that talk to the servers, MIBs, backend software, and database schema versus MIB content. Each of the FCAPS is described in the context of a software block description. The various databases employed in a managed network are described along with typical operations performed against them. Middleware is briefly described, and the trend toward using Java for developing network management systems is discussed.

[Chapter 7](#) describes how to build some basic network management system components using Visual C++ and Java. Typical SNMP operations are illustrated with screenshots. A scheme for extending these rudimentary components is enumerated.

In [Chapter 8](#), the MPLS management case study is introduced with a description of the principal components of this forwarding technology. The MPLS MIB content of interest is then introduced, and detailed directions on how to create an LSP and a tunnel are provided. The merits of using signaling for the creation of MPLS tunnels are briefly described followed by a discussion of adding new entries into MIB tables. The role of the standards process is then described.

[Chapter 9](#) brings together all of the threads running through the book and reiterates the overriding importance of MIB structure and design for successful network management system development. There is a strong need for thin, well-separated software layers in network management systems. Scalability is also a major issue in network management, the more so given the emerging generation of dense, multitechnology NEs. The latter provide a compelling argument for pushing more decision-making into the network. We illustrate a trend in this direction with the MPLS FTN MIB. As far as possible, technology-specific code in the network management system should be minimized. Security is high on the agenda of network operators, as is the need for solutions (rather than just technology). Economic downturns may diminish operator appetite for purchasing consultancy services—this can put more pressure on vendors to produce generic overall solutions. The need for solution mindsets has an important bearing on the roles of QA, IT, and software developers.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

A Note About Abbreviations

The field of network management features an enormous and ever-growing array of abbreviations and acronyms. Since this book is aimed at practitioners, we chose not to expand all abbreviations inline (though many are). Instead, there is a detailed glossary at the end of the book containing descriptions and short definitions of many of the abbreviations. Readers less familiar with the abbreviated terms will hopefully find the glossary useful. There is also a list of abbreviations at the start of the book.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

Additional Resources

One of the big problems attached to working in the network management area lies in knowing where to locate good sources of information. Below are some additional resources (in alphabetical order) for further reading.

www.etsi.org/ European Telecommunications Standards Institute—details on standards for mobile telephony, signaling, and so on.

www.ietf.org/ Internet Engineering Task Force—RFCs, MIBs, and many other useful documents. This site provides lots of interesting reading written by industry experts. Some IETF documents are a little impenetrable at first but are usually well worth the effort of reading in full.

www.itu.int/home/index.html International Telecommunications Union—general telecommunications.

www.metroethernetforum.org/about.htm Metro Ethernet Forum—dedicated to accelerating the adoption of Optical Ethernet in metropolitan networks worldwide.

www.mplsforum.org/ The MPLS Forum serves two important functions. First, it produces Implementation Agreements (IAs) in the areas that are not covered elsewhere and/or are related to a combination of technologies. Examples are IAs on voice-over-MPLS and MPLS PVC UNI. Second, it works with major interoperability labs, such as the ones in the University of New Hampshire and at the technical university of Berlin (EANTC), on defining interoperability requirements for various MPLS protocols and then organizing testing events. The MPLS Forum also has an MPLS educational function, and in this role it develops and presents tutorials related to advanced and emerging areas of MPLS and provides speakers to major conferences. The MPLS Forum actively works with other organizations, such as the ATM Forum and the Frame Relay Forum, and many members of the MPLS Forum are active participants in the IETF, particularly in the PWE3 group.

www.mplsrc.com/ The MPLS Resource Center—lots of information about MPLS.

www.protocols.com/pbook/ Protocol directory—details on various protocols.

www.simple-times.org/ Historical introduction to SNMP.

standards.ieee.org/catalog/olis/ IEEE—standards and many other technical topics.

www.simpleweb.org/ietf/mibs/ IETF Web site dedicated to MIBs.

www.telecoms-mag.com/ Telecommunications Online Magazine—good topical coverage.

www.telcordia.com/ Telcordia—excellent site on general telecommunications issues, current research and ideas on algorithms, management, and so on.

www.telelogic.com/ Telelogic—producers of development tools for telecommunications. Its impressive Tau product has SDL and UML interfaces and provides code-generation technology based on formal methods.

Any of the Internet search engines can provide further information if required.

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Acknowledgments

My deepest thanks go to Siobhán for always believing in me and providing my inspiration.

I'd also like to warmly thank the staff at Prentice Hall, particularly Mary Franz who guided this book from start to finish. Thanks also to Dan DePasquale, Jennifer Blackwell, Noreen Regina, Maiko Isobe, Joan Caruso, Lisa Iarkowski, Gail Cocker, Kathleen M. Caren, and Carol Lallier.

I'd like to express my sincere gratitude for the excellent comments we received from the following reviewers, who patiently read the entire manuscript: Victoria Fineberg, David Levi, Rob Rosenberry, David Green, and Andrew Walding. A special word of thanks goes to David Levi for kindly writing the foreword. It has been an education and a privilege to work with such gifted people.

All the opinions expressed in this book are my own and in no way reflect those of any organization. Any errors or omissions are my own.

I'm interested to hear reader comments: how the book might be improved, areas that need more coverage, and other suggestions or opinions. Please feel free to contact me at netmgmt_smorris@phptr.com.

[Team LiB]

◀ PREVIOUS NEXT ▶

Chapter 1. Large Enterprise Networks

Modern networks are divided, in terms of their operations, into essentially two main categories: enterprise and service provider (SP). In this book we focus on the principles and concepts of managing large enterprise networks. Examples of such networks are government departments, global corporations, and large financial/healthcare organizations. Most such enterprises employ the products and services of SP networks, so we try to balance the discussion by including some general comments about managing SP networks as well. It is in the latter network type that we tend to describe Multiprotocol Label Switching (MPLS), a widely deployed technology. In passing, we mention that MPLS is also finding its way into large enterprise WANs.

An important point to note is that network management is a distinct and separate discipline from both enterprise and SP networking. For this reason, our study of enterprise, SP, and MPLS network management should be seen merely as applications of network management technology. As we'll see, many elements of network management are common across all such application areas. We have six main aims:

1. To illustrate some important aspects of network management, especially enterprise networks but also SP networks.
2. To describe some increasingly important problems facing Simple Network Management Protocol (SNMP)-based network management systems (NMS^[1]).
3. To describe some Management Information Base (MIB) improvements that would assist manageability.
4. To illustrate the construction of a rudimentary NMS using Visual C++ and Java.
5. To describe MPLS and the advantages that it provides to enterprise and SP networks.
6. To illustrate the need for increased (policy-based) intelligence in managed devices.

[1] The term [NMS](#) is used throughout this book. Depending on the context, it may be either singular or plural: "an NMS is..." or "many NMS provide this feature...." Rather than using the unwieldy NMSs, we opted for just NMS and let the context indicate either singular or plural.

We set the scene by describing in general terms some of the components of large enterprise networks. These networks are big and geographically dispersed (often spanning many countries), have lots of legacy equipment, and are hard to manage—scalability is an issue affecting both their manageability and usability. After introducing the general area, we begin our discussion of network management.

Generally, enterprise networks are owned by a single organization, such as IBM, federal government bodies, and financial institutions. These networks exist to provide data and telecommunications services to employees, customers, and suppliers. Services can include:

- File and data storage
- Print
- Email
- Access to shared applications
- Internet access
- Intranet
- Extranet
- E-commerce
- Dial tone

- International desk-to-desk dialing (using voice-over-TDM or voice-over-IP)
- Video
- LAN and virtual LAN (VLAN)— often heavily overengineered (more bandwidth than necessary) to avoid congestion
- Corporate WAN— can be used for data and also voice-over-IP
- Virtual private network (VPN)— can be used for securely joining multiple sites and remote workers and replacing expensive leased lines
- Disaster recovery— maintaining network service after some cataclysmic event

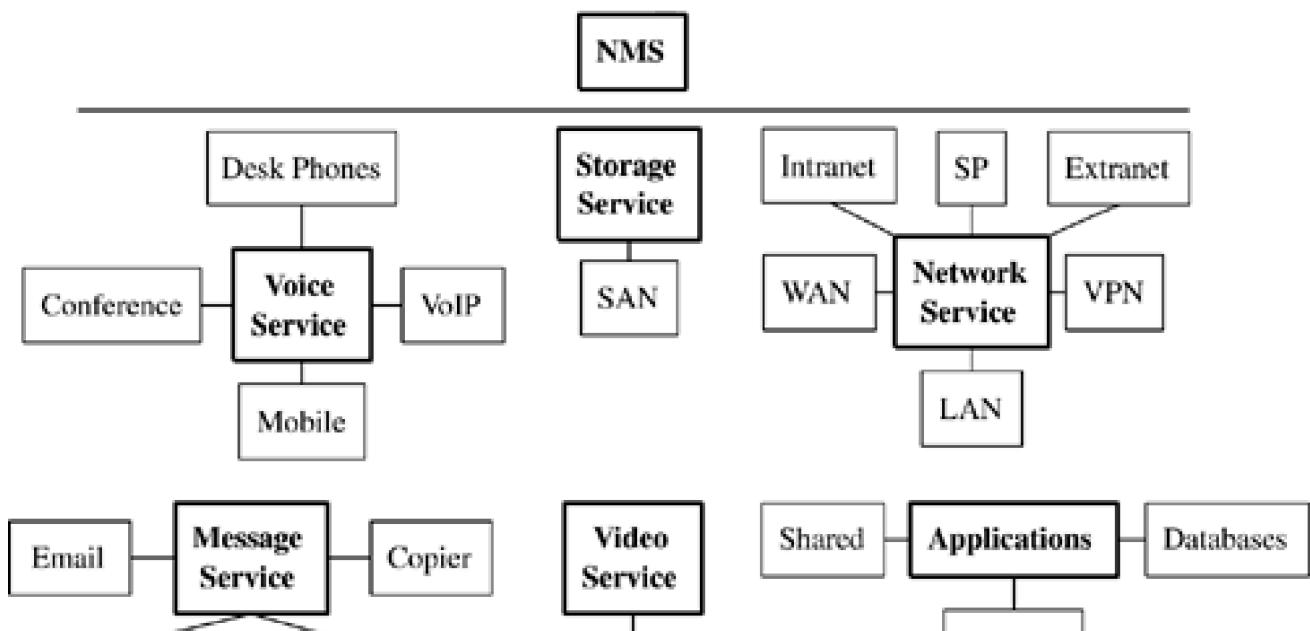
Enterprise networks achieve these and other services by deploying a wide variety of different technologies and systems. Some services encompass several technologies, such as voice-over-IP (VoIP) [Tanenbaum2003], which can be transported over a WAN link to achieve toll bypass or migration away from overlay networks (for voice, video, and data).

An enterprise^[2] uses its network as a means of providing or improving business processes and saving money rather than as a vehicle for profit. This mindset influences enterprise decisions to deploy solutions like VoIP telephony. The guiding principle is service enhancement and business advantage rather than reductions in spending (though the latter is also extremely important). In fact, some global organizations are so big that they can often negotiate reduced tariffs with their local telecommunications carrier—in many cases a quicker and easier way to save money than rolling out expensive, complex, new technology. It may be more important for an organizational department, such as a provider of frontline PC support, to direct a minimum of incoming phone calls to voicemail. This can influence the decision to deploy in-building mobile telephony (e.g., IEEE 802.11 a/b, DECT in Europe) so that call handling is not restricted to the desk phone. In other words, service levels are enhanced because calls are less frequently routed to voicemail.

^[2] An enterprise may decide to outsource or even sell its entire network infrastructure to a third party. The enterprise can then lease back the network from the new owner. The net effect is that the enterprise outsources much of its network management workload and exchanges a depreciating asset for cash. The same type of sale and leaseback can be done on pretty much any type of asset, including buildings. The buyer leases the asset back to the enterprise for a fixed annual outlay. The burden of ownership then resides in the leasing company. The merit of doing this with network hardware is that the enterprise usually gets good terms for upgrades.

[Figure 1-1](#) illustrates a typical simplified enterprise network. [Figure 1-1](#) is highly simplified in order to give us a flavor of enterprise networking issues. Real enterprise networks tend to feature additional technologies, such as Asynchronous Transfer Mode (ATM), VLANs, broadband connections, and redundant configurations. Later (in [Figure 1-4](#)) we will see a portion of an enterprise network realized using VLANs.

Figure 1-1. Enterprise network functional components.



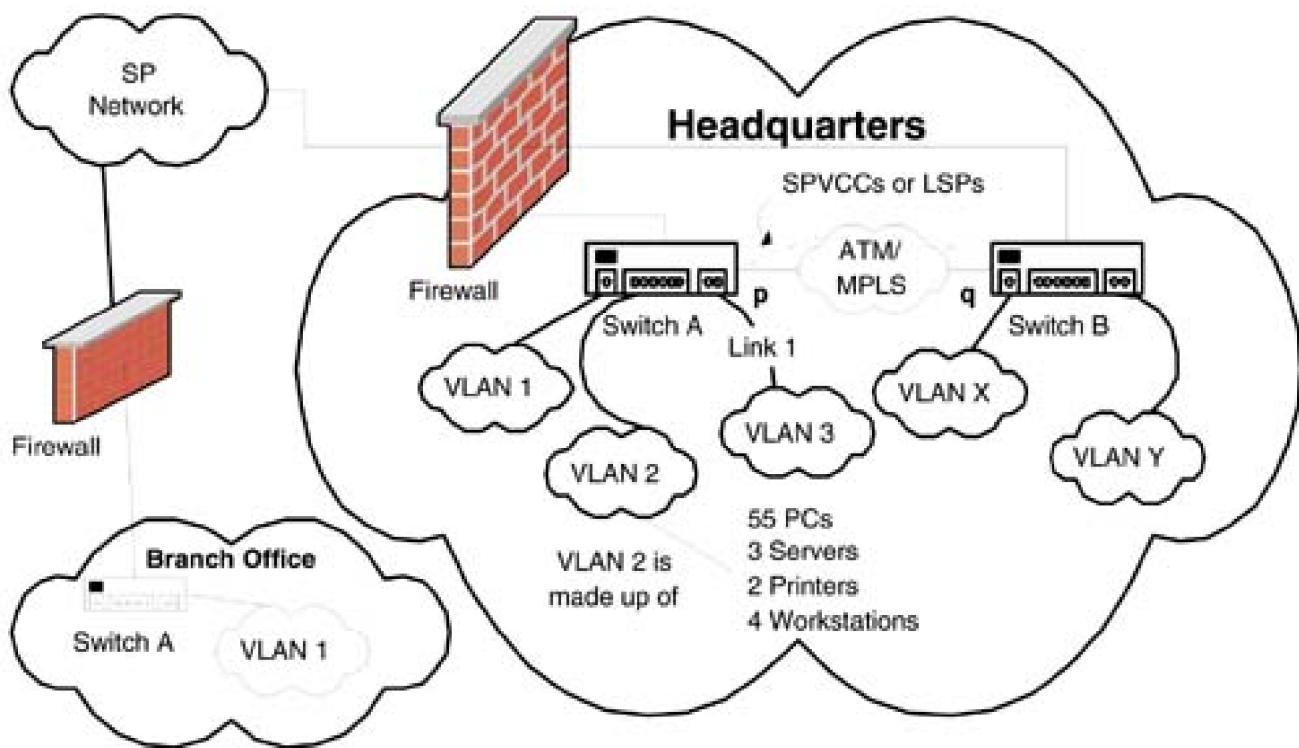
Fax

Voicemail

Conference

Desktop

Figure 1-4. VLANs in an enterprise network.



All the boxes with bold text and borders in [Figure 1-1](#) provide some type of service—for example, Voice Service. The connected boxes provide access to the service—for example, VoIP phones (in the VoIP box). In fact, the network in [Figure 1-1](#) can serve a large, geographically distributed corporate user population. Alternatively, [Figure 1-1](#) might be a corporate headquarters with hundreds of remote branch offices. It's easy to see why the provision and management of enterprise networks are so critical to modern organizations.

The networks and systems in [Figure 1-1](#) add value to the organization, and later we'll see how the enterprise network managers (in many cases, IT groups) can play an important role in assisting the developers of network management software. In this way, IT initiatives are closely aligned with broader business objectives [[EnterpriseIT](#)].

Also noteworthy (as mentioned above) is the use of IP phones in a LAN environment, reducing the need for legacy PABX equipment and prompting migration to a packet-based infrastructure. The migration to layer 3 mentioned here is discussed in [Chapter 2](#), "SNMPv3 and Network Management," and is a recurring theme throughout the book.

One point about [Figure 1-1](#) is that many or all of its components may be repeated on other sites linked to this one via a WAN. These other sites include normal branches of the organization as well as unmanned backup sites. This means that essentially the same corporate services are offered to all employees regardless of their location, whether it is in New York City or the West of Ireland. Many organizations fund this type of arrangement by charging a straight percentage from the revenues of each local site. Also, different sites can offer services, such as audio conference bridges, to other sites. In this case, the site hosting the bridge bills the users dialing into it from remote sites. There are many reasons for using this geographically distributed approach to enterprise network deployment:

- Expensive systems, software applications and licenses can be shared across time zones.
- Valuable data, such as subscriptions to ETSI and ITU, can be shared.
- Remote sites can help the company gain access to specific local markets.
- Access can be gained to specialized labor skills, such as software development or manufacturing.
- Organizations can take advantage of different tax regimes to improve revenues.
- Some configuration can be handled from a central location—for example, PABX maintenance can be carried out by a centrally

located specialist team.

Notable features of [Figure 1-1](#) are the incorporation of separate networks for storage (i.e., storage area network, or SAN), WAN, SP networks, and telephony. SANs provide access to data storage facilities. WANs provide access to remote network facilities. SP networks provide Internet access (among other services), and the Public Switched Telephone Network (PSTN) provides access to the global telephony networks (fixed and mobile). Typically, an enterprise will use several service providers, each providing one or more of the above services.

The enterprise network enables access to a wide variety of devices and services. The important point about the structure depicted in [Figure 1-1](#) is its flexibility: Large numbers of users can share the corporate, productivity-enhancing services using a wide range of access methods. By this means, an employee working from home can be at least as effective as one based in the office without the need for commuting. Similarly, sales staff can access (e.g., via a VPN) the enterprise network during business trips.

Another trend is unified messaging for integrated access to email, voicemail and fax mail messages using an email client. PCs can also be used for access to videoconference broadcasts and even videophone calls. Audio conference calls can also be accessed via unified messaging or by using a desk phone. Some organizations even use broadcast voicemail to make important announcements. Another aspect of enterprise networks is linkages between desktop calendars and the reservation of meeting rooms. Rooms are booked and invitees are reminded via their email client.

Intranets provide official enterprise information channels for employees. Many organizations use intranets for posting important information such as product announcements and corporate media coverage. Another intranet facility is integration of productivity tools such as document management systems. In this sense, the intranet becomes just another desktop tool accessible using a Web browser. As we'll see later, the Web browser is often an indispensable part of an NMS.

Enterprise data flows can become very complex once extranets and e-commerce are employed. Extranets are parts of intranets that are extended to organizations external to the enterprise, such as software contractors. E-commerce allows for secure financial transactions between external customers and a given organization. The data flows in the latter case feed into various systems, such as finance, stock control, and manufacturing.

Other important aspects of maintaining secure enterprise networks include:

- Automated software distribution (e.g., of anti-virus software)
- Policy setup (e.g., auto-logout after a specified interval of nonactivity)
- Software application license checking

Many organizations distribute enterprise software in a centralized fashion, for example, using Microsoft Systems Management Server. This can include defensive procedures such as anti-virus software updates. Likewise, productivity software such as word processors and spreadsheets can generally be updated in the same way. Many end users of enterprise systems tend not to log out, so policies can be applied to host machines that will log the user out after, say, 15 minutes of inactivity. This can be done for security reasons and also in order to update anti-virus software once the user logs back in again. A full virus scan can then occur at night. The important area of software license checking can also be handled remotely to verify that the number of end users who have installed software packages does not exceed the license limit.

These various uses of enterprise facilities clearly illustrate the power of the underlying network. Following are some general features of enterprise networks:

- They incorporate a wide range of multivendor devices, such as routers, switches, exchanges, PCs, servers, printers, terminal servers, digital cross-connects, multiplexers, storage devices, VoIP telephones, servers, and firewalls.
- Network elements (NEs) can incorporate other intelligent devices, such as PCs with network interface cards (NICs) and possibly modems. Likewise, desk phones can contain computer-telephony integration (CTI) hardware for applications like call centers and e-commerce bureaus.
- Individual NEs provide a variety of different shared services; for example, a legacy PABX or a soft switch provides basic telephony and can form the foundation of a call center. In this way, a base system is leveraged to provide another system or service.
- Backup and restore of NE firmware are important for rolling out new network services.
- Specialized servers are deployed to provide advanced services such as SANs.

- Many users are supported simultaneously.
- The overall network services, such as email and video/audio conferencing, are used by employees of the organization as essential business process components.

Enterprise systems and networks all have individual lifecycles comprised of:

- Planning
- Deployment
- Operation and management
- Retirement, replacement, or upgrade

In this book we focus mostly on network operation and management, but the other lifecycle stages are equally important. An example of this is a SAN in which the following steps typically occur:

- Planning the required storage capacity, server links, and network connection
- Deploying the SAN in a production environment
- Operation and management of the SAN: discovering SAN components in a vendor-independent fashion, monitoring faults, checking performance, and backup and restore
- Extending the SAN as storage requirements grow

Growing storage requirements in enterprises can have the effect of reducing backup time windows. This and other storage issues may cause loss of service and require that administrators deal with problems such as:

- Devices going offline
- Capacity being exceeded
- Performance degradation
- Application software with rapidly increasing storage requirements

All of these require some type of reactive (after the problem has occurred) manual intervention. Clearly, there is a relationship between storage planning and the incidence of storage capacity being exceeded. The same is true for the ever-increasing storage demands of application software. Network administrators need tools to help them balance these dynamic requirements. Where possible, the NEs should be engineered to facilitate this type of advanced management. In conjunction with NE-resident self-management capabilities, there is a need for high-quality management systems. The latter should then provide features that match the organizational workflows (broadly speaking, these are plan, construct, and operate).

Another very common enterprise technology is the VLAN. Many organizations employ VLANs in order to provide a switched layer 2 infrastructure with designated broadcast domains. A broadcast domain is a set of layer 2 devices with a defined boundary (typically an IP router) beyond which broadcast traffic will not flow. For example, an organization could group the NEs on each floor of a building into a different VLAN (i.e., broadcast domain). All of these floor-level VLANs could then be connected to a single high-speed switch that is in turn connected to another set of VLANs. One of the merits of VLANs is scalability—to add more devices, you can just create another VLAN. This helps to avoid the problem of running out of broadcast domain capacity on a single medium (such as a large Ethernet network).

Building and operating VLANs can be carried out using either an element management system (EMS) or an NMS. A typical workflow for adding a new PC to VLAN X is as follows:

- Physically connect the host PC to a port on the switch containing VLAN X.
- Using the switch element manager, add the port to VLAN X.

- Specify no tagging (the legacy case), that is, the PC NIC adds no IEEE 802.1p/Q fields to its Ethernet headers (these are two fields contained in the Ethernet frame header: 3 bits for priority and 12 bits for a VLAN ID value).
- Verify host PC connectivity (by logging into the network, pinging servers, etc.).

As far as possible, the NMS—or EMS in this case—should facilitate this type of workflow. For example, when adding a port to a VLAN, only options appropriate to that hardware should be presented. So, if a port does not support 802.1Q, then the EMS/NMS should not present an option to set a VLAN ID. This information can be acquired by the EMS/NMS (via automatic dialog with the NE) and greatly assists in managing such devices.

There is a downside to the rich environment provided by enterprise networks. They are expensive to build and run, and they require skilled maintenance and support personnel. Traditionally, the network support effort (excluding PC support) has been divided into two camps, data networking and telecommunications, but these two areas are rapidly converging. PABX technology is gradually being phased out and replaced by server-based solutions [[CiscoVoIP](#)]. Multiple incompatible networks for voice, video, and data are gradually being migrated onto a packet-based infrastructure.

Many organizations seek to centralize servers in secure locations and then lease WAN lines from there to branch offices and divisions. This reduces remote site support but increases dependency on communications lines, an increasingly cheap commodity [[GlobalCross2002](#)]. Services are resolving down to the process of transporting bits from location X to location Y over a single physical network.

[[Team LiB](#)]

 PREVIOUS  NEXT 

Managing Enterprise Networks

Why is enterprise network management important? First, it helps keep the overall network running—end users are kept happy and the business processes are not blocked by downtime. Second, good network management facilities assist in all the lifecycle stages. Third, such facilities should help to reduce the cost of running the network. This last point is particularly important during periods when IT budgets and staff numbers are cut.

An important issue concerning enterprise networks is the presence of multiple incompatible management systems. While expensive resources are shared using the underlying network, these resources are generally not centrally managed in a technology-independent fashion. An example of this is the SAN facility shown in [Figure 1-1](#). The individual components of SANs (disk subsystems, network switches, and SAN servers) typically each have a dedicated management system. This substantially adds to the cost of ownership. Gartner Group Research claims that the cost of managing storage is five to seven times the price of purchasing storage [\[NovellSAN\]](#). Generic enterprise management systems, such as HP OpenView, already exist, but not all of the networked systems (such as in [Figure 1-1](#)) have the necessary infrastructure that would allow them to be managed in an integrated fashion. An illustration of this [\[Figure 1-1\]](#) occurs if one of the digital phone cards in the PABX (the Voice Service in [Figure 1-1](#)) fails. If the PABX does not emit some type of message to this effect, then the desk phones connected to the card in question will lose service until the problem is fixed. Likewise, if a WAN access switch fails, then the WAN connection may be lost. If there is no integrated NMS in place to detect and signal these types of problems, then service loss will occur until the problems are reported and fixed.

It is a central theme of this book that the vendors of as many systems as possible should include SNMP (preferably version 3) management capability as a priority. This would allow for all managed elements to emit traps (or messages) as soon as a problem occurs. The necessary minimal components required for making a system manageable are:

- MIBs
- Agents/entities—hosted on network devices to provide management facilities
- Scripts for manipulating MIB objects
- Java/C/C++ software modules for manipulating MIB objects

MIBs provide a detailed description of the managed data objects. Typically, the description of each MIB object consists of:

- Accessibility (read-only, read-write, not-accessible)
- Status (mandatory, deprecated)
- Description

Agents (or entities in SNMPv3) are software components that implement the MIB and map the objects to real data on the NE. It is the agent's job to maintain, retrieve, and modify MIB object instance values. The network manager delegates this important task to the SNMP agent. The agent also emits special messages called notifications to signal the occurrence of important events, such as a device restarting or a network interface going down. Finally, the agent must implement all of this using some preconfigured security scheme ranging from simple passwords to stronger techniques involving authentication and encryption.

On the manager side, it is important to be able to manipulate the various agent MIBs. This can be done using scripts or via binary software modules built using various programming languages such as Java/C/C++. In either of these two cases it is often necessary to load the associated agent MIB module files into a management application. An example of this is a MIB browser: an application that allows for MIB objects to be viewed (some browsers allow for MIB object instances to be modified). Most MIB browsers merely require MIB module files to be loaded; that is, they are preconfigured with the necessary SNMP protocol software.

Another very important topic is the management of both newly commissioned and legacy NEs. It is rare (particularly during periods of economic recession) for large networks to have forklift upgrades in which the very latest NEs are deployed in place of legacy devices. Normally, new NEs are added and old ones are replaced. For this reason, one can expect a rich mixture of deployed devices, both old and

new. This generally also means a complex set of MIBs deployed across the network. As we'll see, this can result in problems related to the support of backwards compatibility (a little like saving a word-processed document using version 4 and then experiencing problems opening the document with version 3 on your laptop).

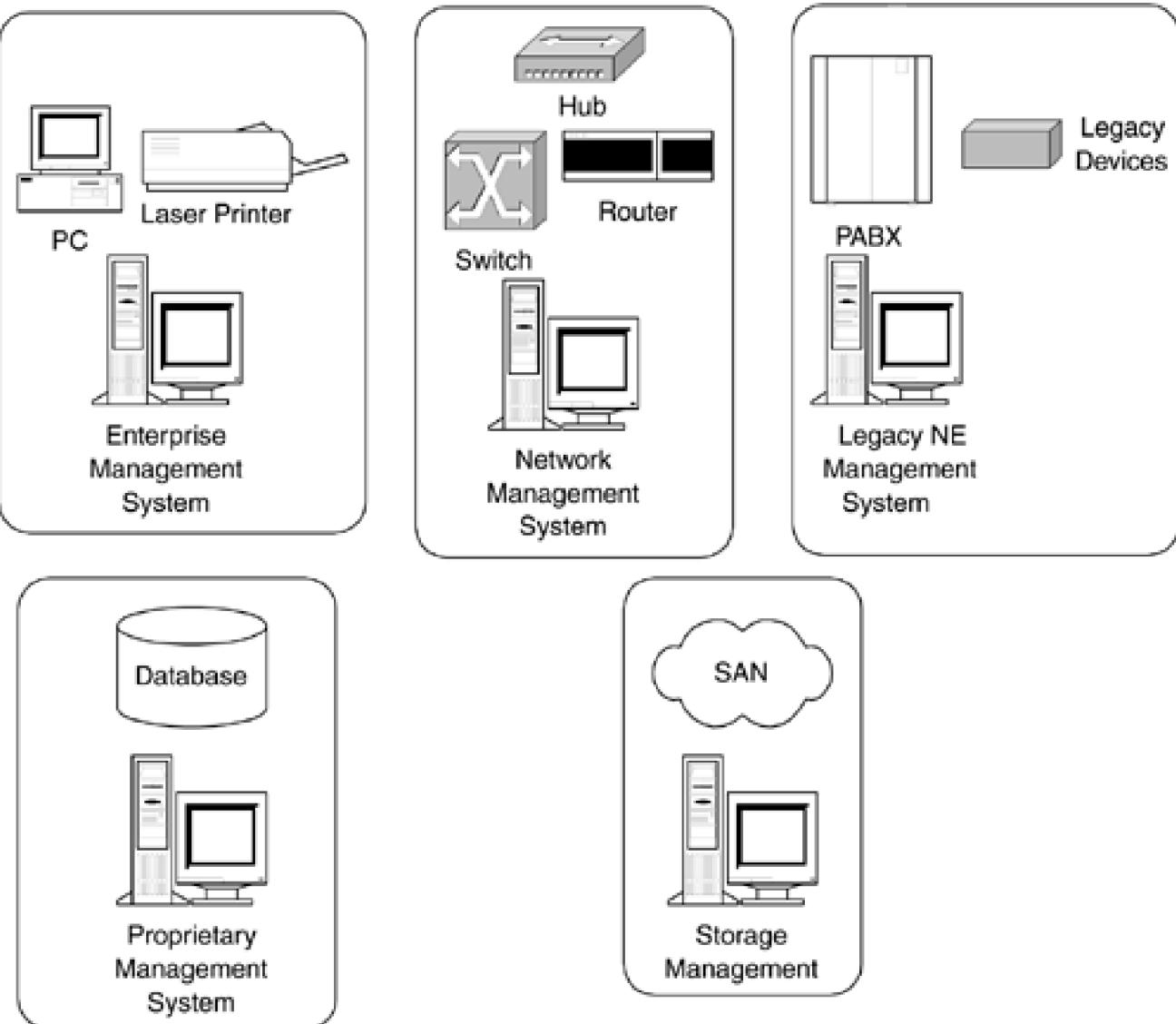
MIBs provide the managed object definitions (type, structure, syntax, etc.) for the underlying system; for example, a terminal server may implement the following principal managed objects:

- Serial interfaces
- Serial interface attributes such as bit rate, word size, and parity
- IP address

To provide baseline SNMP management for a terminal server, the relevant MIB must be consulted for the requisite managed-object definitions. The instance values of these objects can then be looked up using a MIB browser. The SNMP software modules (along with the MIBs) can be integrated into a management system and used to monitor and configure the associated agent. This approach (using SNMP) obviates the need for a proprietary management system. More details on the topic of terminal-server serial-interface MIB objects can be found in [Appendix A](#), "Terminal Server Serial Ports." Later, we'll see that the quality of the MIBs has an important bearing on the manageability of a given NE.

[Figure 1-2](#) illustrates a different view of an enterprise network.

Figure 1-2. Enterprise management systems.



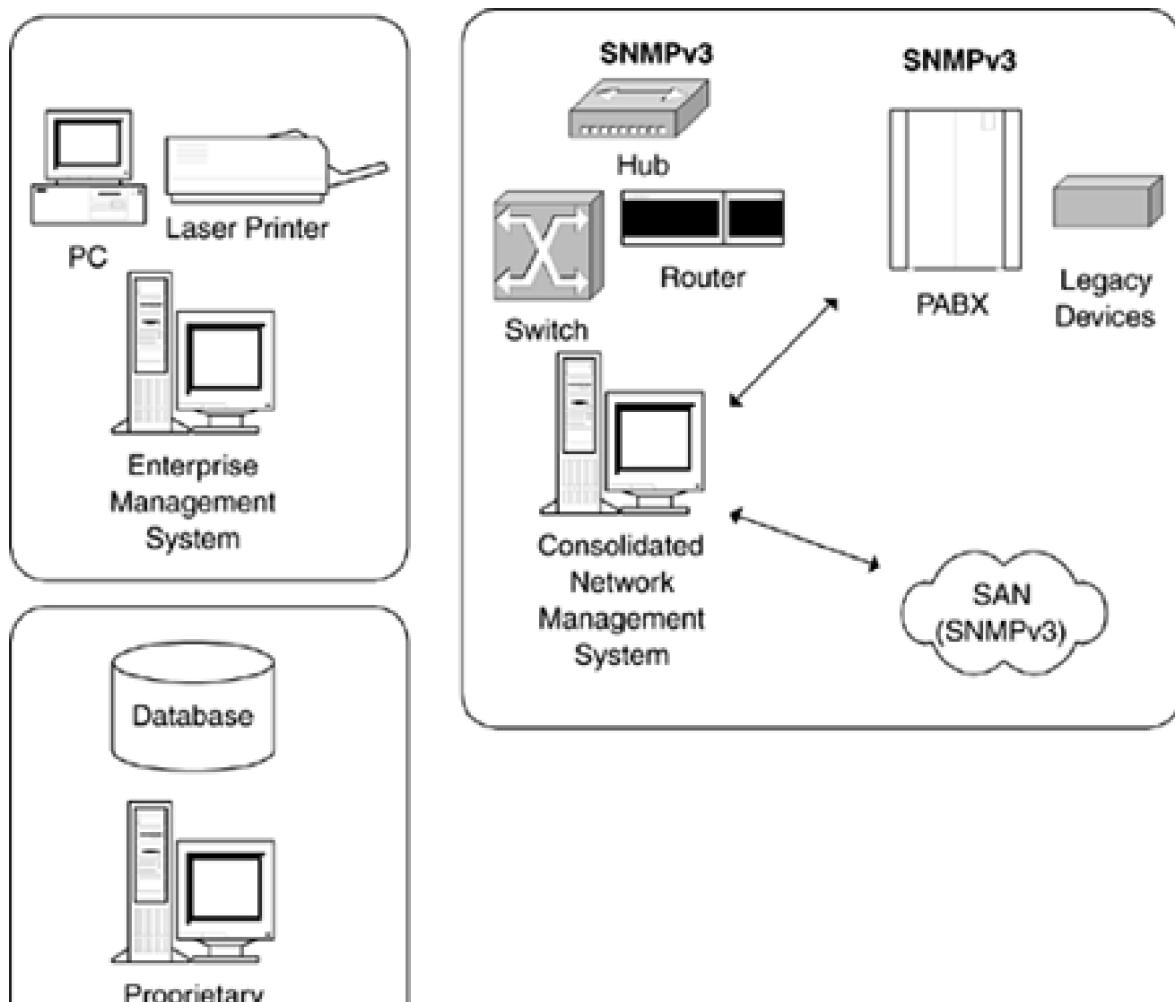
In this diagram, the NEs are grouped alongside their associated management systems. The multiplicity of management systems is one of the reasons why enterprise network management is so difficult. This is what we mean by multiple incompatible management systems: Problems in a device attached to the PABX are not reflected back to the enterprise network manager. Instead, they register by some proprietary means in the legacy NE management system (if one is deployed), and it is up to IT to discover and resolve the problem. Many smaller devices (such as terminal servers) support only a simple text-menu-based EMS or command-line interface (CLI). The absence of SNMP agents (or the deployment of only SNMPv1) on these devices contributes to making them difficult to manage in an integrated, vendor-independent, and centralized fashion.

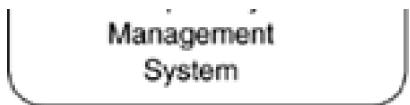
In order to manage enterprise networks as seen in [Figure 1-2](#), it is necessary to learn all of the deployed technologies as well as their proprietary management systems. This is an increasingly tall order. In many organizations, the management facilities consist of simple scripts to configure and monitor devices. While many enterprise network managers may implement ingenious script-based facilities, all such solutions suffer from being proprietary. An added problem is seen when the author leaves the organization—the requisite knowledge often leaves at the same time. Adoption of standards-based network management technology helps in avoiding this. Standards-based consolidation of management systems can help enterprises to achieve the following:

- Fewer and simpler user interfaces for managing networked systems
- Reduction in the time required for IT staff training
- Faster resolution of NE problems, such as switch interface congestion

A single (or a reduction in proprietary) management technology in the network contributes to making that network easier (and cheaper) to operate and maintain. It is for this reason that we say as many as possible of the components of enterprise networks should implement SNMPv3 agents (or entities, as they are called). [Figure 1-3](#) illustrates a modified enterprise network with SNMPv3 entities deployed in the SAN, legacy NEs, and in the switch/router/hub NEs.

Figure 1-3. Example of consolidated enterprise NMS.





If all of the NEs deploy SNMPv3 entities, then it is possible that one or more of the proprietary management systems (in [Figure 1-2](#)) can be removed and consolidated into one NMS. Of course, it's not so easy to just add SNMPv3 capability to all of these NEs (particularly the legacy NEs). The point is that it has a substantial benefit.

The other enterprise systems in [Figures 1-2](#) and [1-3](#) (the networked PCs, print servers, and database management system) generally tend not to deploy SNMP for their management and operation. This is largely for historical reasons. Since this book is about network management rather than system management, we do not consider this area any further. However, before moving on, we should say that there are no major reasons why SNMP technology should not be used for managing such systems.

Manageability

For a number of reasons, not all NEs lend themselves to flexible, integrated, centralized management. This tends to add to the cost of ownership and arises for a range of reasons:

- The NE is a legacy device with proprietary management infrastructure.
- The NE implements only SNMPv1 *with* support for set operations (a set operation is an update to a network-resident managed object).
- The NE implements only SNMPv1 *without* support for set operations.
- The NE supports SNMPv3, but it has been poorly implemented.
- The NE supports SNMPv3 but has a number of low quality MIB modules.

Proprietary management infrastructure may consist of just a simple CLI with no SNMP deployment. It is difficult and costly to incorporate these NEs into an NMS because customized software must be written to send and receive messages to them. NEs that support just SNMPv1 and set operations are generally felt to be a security risk (because the relevant password is transmitted across the network as clear text). As a result, no set operations may be allowed. Configuring such NEs is usually achieved via CLI scripts. While this is a fairly standard approach, it negates some of the benefits of using an NMS, such as security, audit trails, and GUI-based help facilities. Much the same applies for those NEs with SNMPv1 and no set operation support. Configuration must be achieved using non-SNMP methods.

Poor implementation of SNMPv3 might consist of low resource allocation (process priority, message buffers, caching, etc.) with the result that the management system regularly gets choked off. This may be seen during periods of high device or network loading (often the time when network management is most needed).

Badly written MIBs are the bane of the NMS developer's life. We'll see examples of good MIB design later on, but for now we illustrate this with a simple example of adding a new row to a table indexed by an integer value. To add a new row to this table, a new index value is required. Often, MIB tables do not implement a simple integer object to store the value of the next free index. This may require a full (expensive) walk of the table in order to calculate the next free index. This is inconvenient when the table is small (less than 100 entries), but when the table is big (many thousands of entries), a MIB walk becomes an expensive option because of the number of agent operations and the associated network traffic. The inclusion of a specific index object to facilitate new row addition can greatly assist the management system. We will see all of these considerations in action later on.

In summary, an NE is considered to have good manageability if it supports a well-implemented SNMPv3 agent and a high-quality MIB.

Operating and Managing Large Networks

Running networks such as the ones described above is difficult. The growing range of services offered to end users means that traffic

levels are always increasing. Deploying more bandwidth can offset rising traffic levels but, unfortunately, the nature of this traffic is also changing as the associated applications become more resource-intensive and mission-critical. This is seen in [Figure 1-1](#) with LAN-based voice, video, and data applications, which (except for data applications) impose stringent timing requirements on the network. Some way of guaranteeing network transport (and NE) availability is needed, and best-effort IP service in the long run is probably insufficient for large, distributed enterprises. This is one of the biggest challenges facing all network operators—how to provision bandwidth-intensive, time-constrained applications on layer 3 networks. Many enterprises and SPs have used overengineering of the network core bandwidth to cater to increased traffic levels. This is ultimately not scalable, and later on we examine the solution MPLS offers to hard-pressed network operators. It is increasingly important for the network to provide defined quality of service levels for traffic.

Some important aspects of enterprise network management are:

- Availability of NEs, interfaces, links, and services
- Discovery and inventory management
- Monitoring the status of NEs, interfaces, links, virtual circuits, VLANs, and so on
- Measuring traffic levels and checking for network congestion
- Configuration—VLAN setup, SAN volume setup, storage allocation, remote-control software (Microsoft Systems Management Server), and database redundancy (e.g., Informix)
- Service level agreement (SLA) reporting, SLA verification between an enterprise and SP
- Security control—resistance to attacks from both sides of the firewall
- Scalability—handling increased numbers of users, traffic, NEs, and so on
- Disaster recovery

We will cover many of these topics. In the next sections we look at those OSI network layers of greatest relevance for the forthcoming discussions.

Layers 2, 3, and 2.5

Reference is made throughout this book to layer 2 and 3 devices [\[Puzmanova2001\]](#). Some confusion seems to surround the use of these terms both in the industry and in the literature. Issues affecting layers 2 and 3 on enterprise networks are a recurring theme throughout this book. Our use of the terms layer 2 and layer 3 follows the guidelines of the OSI model. A layer 2 device is one that operates no higher than the data-link layer—for example, ATM, Frame Relay (FR), and Ethernet switches. The basic unit of transmission at layer 2 is the frame (or cell for ATM). A layer 3 device operates at the network layer and deals only in packets. An example of a layer 3 device is an IP router. Layer 2.5 is a special mode of operation where some of the advantages of layer 2 are leveraged at layer 3. The different layers are described in the following sections.

Layer 2 and VLANs

[Figure 1-4](#) illustrates the core of a fictitious enterprise network operated exclusively using ATM/MPLS multiservice switches. This is a layer 2 network that is logically divided into VLANs (well described in [\[Tanenbaum2003\]](#)). VLANs, as we noted earlier, are broadcast domains that allow communication between member devices as if they were all on the same physical LAN segment.

The switches in [Figure 1-4](#) serve to partition the VLANs by forwarding only appropriately addressed frames. In an effort to improve convergence time, some switches support, on a per-VLAN basis, the spanning tree algorithm (the means by which loops are avoided [\[Tanenbaum2003\]](#)). Spanning Tree Protocol is usually implemented across all LANs, not just VLANs. If it is implemented on a per-VLAN basis, it improves convergence.

The constituents of any of the VLANs in [Figure 1-4](#) can include a number of machines; for example, VLAN 2 consists of 55 PCs, three servers, two printers, and four workstations. Layer 2 broadcasts originating inside any of the VLANs do not cross the boundary of that VLAN. One possible configuration is to allocate a specific VLAN for each layer 3 protocol—for example, IPX in VLAN 1 and IP in the other VLANs. Since VLAN 1 has nodes that understand only IPX, there is no reason for pushing IP traffic into it. Likewise, the nodes in the other VLANs might not understand IPX, so there is no reason for pushing IPX traffic into them. Only layer 3 traffic that needs to exit a VLAN crosses the boundary (via routing) of its container VLAN.

The merit of a VLAN arrangement is that traffic between the constituent devices does not pass needlessly into the other VLANs. Also, if one of the VLANs fails (or if a node inside that VLAN becomes faulty), then the other VLANs can continue to operate. This allows for a more scalable and flexible network design than using IP routers in conjunction with Ethernet segments.

Typically, the hosts in each of the VLANs support layer 3 routing capabilities (e.g., IP, IPX). This is required for communication outside the VLAN boundary. Each such host supports layer 3 routing tables with at least one entry pointing to an external router. The latter may be implemented on the local switch (A or B in [Figure 1-4](#)) and serves to direct outgoing and incoming IP traffic across the VLAN boundary. To illustrate this, [Table 1-1](#) depicts an excerpt from a routing table from one of the 55 PCs in VLAN 2. The data in [Table 1-1](#) is obtained by using the `netstat -r` command from within a DOS console.

Table 1-1. IP Routing Table for a Host PC in VLAN 2

NETWORK DESTINATION	NETMASK	GATEWAY	INTERFACE	METRIC
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
Default Gateway	142.159.65.17	N/A	N/A	N/A

[Table 1-1](#) illustrates two routing table entries: one for the loopback address and the other for the default gateway. Any packets addressed to the loopback address are sent back to the sender. So, if you ping 127.0.0.1, your host machine (i.e., the sender) will reply. The second entry in [Table 1-1](#) is for the default gateway. This is the IP address of last resort (Internet core routers do not have default gateway entries), that is, the address to which packets are sent for which no other destination can be found. In [Figure 1-4](#) this address (142.159.65.17) would be located on Switch A. It is by this means that hosts in VLAN 2 can exchange messages with entities outside their VLAN boundary. [Appendix B](#) includes examples of using some of the Windows NT/2000 networking utilities.

Another important point about VLANs is that the backbone network (between switches A and B) may be implemented using ATM. If this is the case, then the backbone may implement ATM LAN Emulation (LANE). This serves to make the ATM network behave like a LAN. The backbone can also run MPLS.

Greater flexibility again is afforded by the use of IEEE 802.1Q VLANs. In this technology, the 802.1 Ethernet frame headers have a special 12-bit tag for storing a VLAN ID number. This allows for traffic to flow between different VLANs. It is also possible to use another tag in the 802.1 header for storing priority values; this is the IEEE 802.1p tag—a 3-bit field. This allows different types of traffic to be marked (with a specific priority number) for special treatment.

Traffic that must pass across the ATM/MPLS backbone is destined for another VLAN (e.g., VLAN X in [Figure 1-4](#)). This traffic can be transported using either ATM or MPLS. ATM cells are presented at interface p of ATM Switch A. An ATM Switched (Soft or Smart) Permanent Virtual Channel Connection (SPVCC) has been created between switches A and B. This virtual circuit traverses the ATM/MPLS cloud between switches A and B. An SPVCC is a signaled virtual circuit, which forms a connection between interfaces on a number of switches. An SPVCC is conceptually similar to a time-division multiplexing (TDM) phone call: An end-to-end path is found, bandwidth is reserved, and the circuit can then be used. The SPVCC in [Figure 1-4](#) starts at interface p on Switch A, travels across the intermediate link, and terminates at interface q on Switch B. This bidirectional virtual circuit transports traffic across the backbone between switches A and B. An important point about circuits that traverse the backbone is that some switches allow the mapping of IEEE 802.1p values to specific circuits. This allows for quite fine-grained quality of service across the backbone.

The SPVCC is a layer 2 connection because the constituent switches have only layer 2 knowledge of the traffic presented on their ingress interfaces. The layer 2 addressing scheme uses a label made up of two components: the Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI) pair. Each switch does a fast lookup of the label and pushes the traffic to the associated egress interface. The switches have no idea about the underlying structure or content of the traffic, which can be anything from telephony to IP packets. As indicated in [Figure 1-4](#), the virtual circuit can also be realized using MPLS label switched paths (LSPs). Such LSPs carry layer 2 traffic encapsulated using MPLS labels (more on this later).

The layer 2 technology that we describe has the following general characteristics:

- Paths through the network can be reserved either manually (by using ATM PVCs or MPLS LSPs) or using signaling (such as ATM PNNI,^[3] MPLS LDP/RSPV-TE).

^[3] Strictly speaking, PNNI (Private Network-to-Network Interface) is both a routing and a signaling protocol.

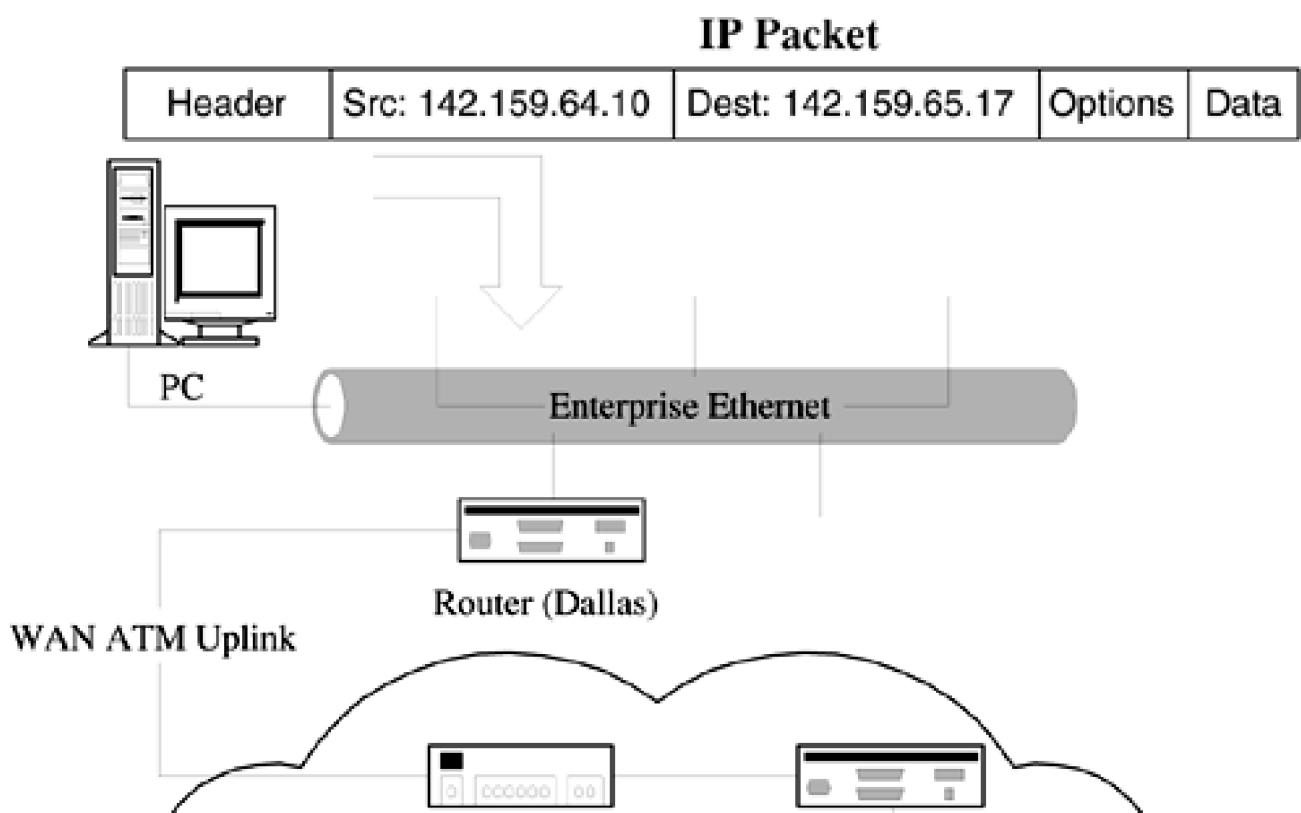
- Paths can be assigned different classes of service, a crucial component for SLAs.
- Layer 2 forwarding is fast because addresses can be looked up with hardware assistance. This is no longer an advantage of layer 2 devices because line-rate forwarding is now also possible with layer 3 devices (i.e., routers).
- ATM layer 2 forwarding allows for traffic policing where contract noncompliant cells can be tagged or dropped. It is also possible to shape traffic so that its arrival rate is controlled. As we'll see when we look at DiffServ, policing and shaping are also available at layer 3.

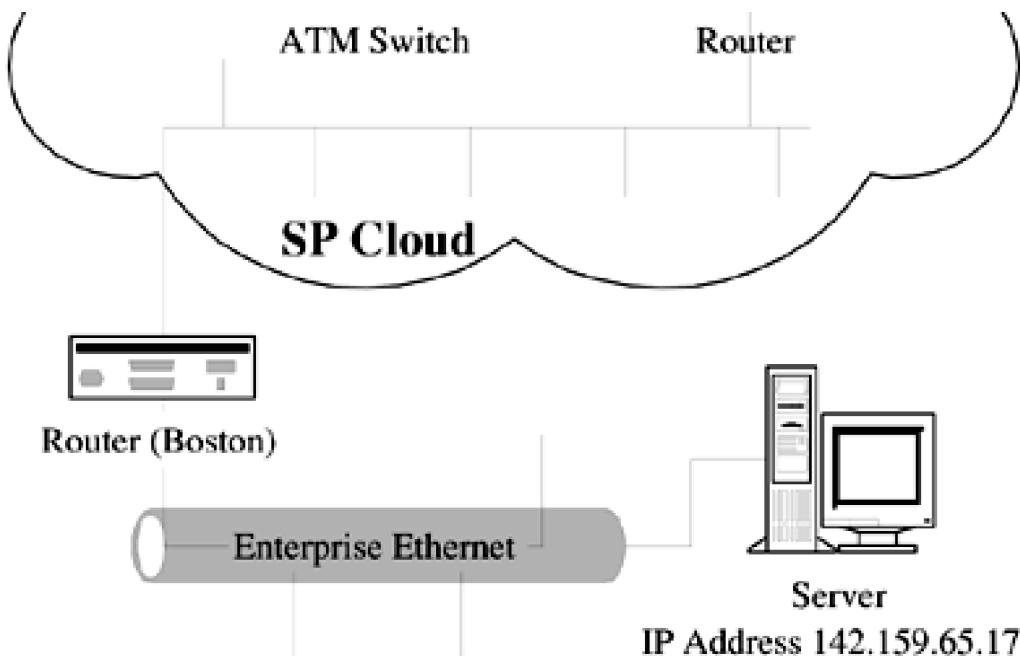
The SPVCC/LSPs in [Figure 1-4](#) represent our first example of virtual circuits. The different categories of traffic (TDM, IP, etc.) presented at interface p can be transported across appropriate virtual circuits. These circuits can be provisioned with different quality of service (more on this later) characteristics to ensure that the traffic receives specific forwarding treatment. So, far, we've only hinted at some of the elements of MPLS but it will be seen that many of the advantages of layer 2 technologies can be obtained at layer via MPLS.

Layer 3

[Figure 1-5](#) illustrates an IP network with an intermediate WAN that crosses an SP network. A client PC in Dallas has some IP data to send to a server in Boston, and the traffic is carried to the destination via the SP network. Each router along the path performs a lookup of the destination IP address (142.159.65.17) and forwards the packet to an appropriate output interface.

Figure 1-5. An IP network.





One of the other major differences between layer 2 and IP is that the latter cannot reserve either resources (such as bandwidth) or paths ahead of time. Even with static routes installed, a full IP address lookup is required at each router, and the direction that the packet takes can change at each hop (for example, if a static route goes down). So, IP packets from a given source can travel over different routes at different times, and ordering is not guaranteed. The TCP protocol gets over some of these problems, but TCP can't reserve bandwidth and full address lookups are still required at each hop.

Layer 2.5 (or Sub-IP)

A further possibility exists for transporting layer 3 traffic: MPLS. MPLS operates at what is often called layer 2.5, that is, not quite layer 3 but also higher than layer 2. MPLS operates by adding a fixed-length (4-byte shim header) label to the payload, which includes an unstructured 20-bit label. This label is then used in forwarding the encapsulated packet. The label is structured for compatibility with ATM [4] VPI/VCI addressing and allows for ATM switches to be upgraded to MPLS. MPLS can also be deployed on routers and brings numerous benefits to IP networks:

[4] Where ATM switches are upgraded to function as MPLS nodes, there is no shim header. Instead, the ATM VPI/VCI fields are used for conveying the label.

- Paths can be reserved before traffic arrives at the network for transport. These can be created either manually or via a signaling protocol.
- Different classes of service can be applied to the reserved paths; for example, VoIP traffic would need a higher class of service than email traffic. This facilitates differentiated services that can be applied to fulfill customer SLAs.
- Traditional IP routing protocols, such as OSPF, IS-IS, and BGP4, can be used. This reduces the cost of developing and deploying MPLS because it leverages proven routing protocols (when they are appropriately extended).
- Traffic engineering becomes possible, allowing every packet to be individually and dynamically processed, resulting in different routes being taken. This helps avoid congested routes.

One disadvantage of MPLS is that all nodes in the path must run the MPLS protocols—an additional burden on network operators. Traffic engineering is often called the MPLS killer app because it permits connection-oriented operation of IP networks. Incoming IP traffic can be redirected to a higher or lower bandwidth path.

Apart from traffic engineering, an emerging function of MPLS is the generic transport of legacy layer 2 services, such as ATM, FR, TDM, and Ethernet. This is an effort to provide a standards-based migration path for network operators who do not want to fully deploy MPLS

throughout their networks. In other words, the legacy services continue to be deployed, but they are transported across a fully or partially deployed MPLS core.

Ports and Interfaces

The terms *port* and *interface* are often used interchangeably. In this book they have a specific meaning. Ports are taken to be underlying hardware entities, such as ATM or Ethernet ports. Interfaces exist at a higher level of abstraction and are configured on top of ports. This is similar to the way an Ethernet port on a PC is configured to run IP. Interfaces are sometimes referred to as logical ports. Examples of interfaces are:

- Routing, such as OSPF, IS-IS, BGP-4
- Signaling, such as RSVP-TE and LDP
- MPLS
- IP

In many cases, the user must manually configure interfaces. The key difference is that ports work out of the box, whereas interfaces generally do not. A lot of action takes place at interfaces—for example, quality of service (QoS) imposition in a DiffServ domain. QoS is a scheme by which traffic is marked prior to or at the entry point to a network. Each node traversed by the traffic then examines (and possibly updates) the marked values. The function of the traffic markings is a signal to the network nodes to try to provide the indicated level of service. Required service levels differ depending on the traffic type; for example, VoIP traffic has specific timing requirements that are more stringent than those for email. The point is that network node interfaces are an integral part of the provision of the QoS scheme. We will see more on this later.

Many SPs provide customer premises equipment (CPE) as part of an enterprise service. CPE is a term that describes some type of switch or router owned by the service provider but located on the customer premises. Examples of CPE devices are seen in [Figure 1-5](#), such as "Router (Boston)". The CPE provides access to the SP network from within the enterprise network. Typically, the CPE provides access to services such as Metro Ethernet, VPN, ATM, FR, and TDM. All of these tend to take the form of one or more ports on a CPE device. Depending on the service purchased, CPE management may be executed either by the service provider, the enterprise, or some combination of the two.

In [Chapter 6](#), "Network Management Software Components," [Figure 6-8](#) illustrates some issues concerning the automatic configuration of IP interfaces. In [Chapter 8](#), "Case Study: MPLS Network Management," [Figure 8-3](#) illustrates a MIB table that provides details of MPLS interfaces on a given NE. One use for the MPLS interface table is selecting MPLS-specific interfaces in an NMS. Selected interfaces can then be used for inclusion in LSPs.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Why Use Network Management?

Devices deployed in networks are increasingly intelligent, so it is interesting to ponder the need for network management. If devices are so smart, then why bother with network management? Can't NEs just self-heal in the event of problems like interfaces going down? Many enterprise networks do not employ NMS—this may be just a matter of policy or even history. There are a number of reasons why network management is a crucial enterprise and SP component:

- NEs don't tend to have an overview of an entire network; management systems do, and this helps in creating objects like connections such as the ones shown in [Figure 1-4](#). The NMS overview is particularly useful for aggregate objects, as we'll see later in this chapter.
- An NMS maintains useful records and audit trails of past configuration actions.
- If NEs do not support SNMP, then an NMS can facilitate a superior CLI because security can be imposed, actions are recorded, and scripts can be managed (stored, updated, etc.).
- NMS can facilitate useful networkwide services like traffic engineering, QoS, planning, modeling, and backup/restore (of firmware or configuration data).
- NMS enables fast access to faults. Some network faults can be meaningfully processed only by an NMS. For example, if a network contains many ATM permanent virtual circuits (PVCs) and an unprotected link fails, then the switches cannot automatically recover, because PVCs do not use signaling. In this situation, management intervention is required to restore the broken link and then the connection. As enterprise networks become increasingly mission-critical with IT offering stringent service contracts, downtime is a luxury few enterprises can afford. So, if a connection fails and has no backup, then the NMS needs to detect it as soon as possible and assist in recovery.
- NMS assists in rebalancing networks after new hardware is added. As networks expand and new switches and routers are added, it is often necessary to bring the new devices into service quickly. Often, such reconfigurations are done during periods of low traffic.^[5] A management system can assist this process by allowing automated bulk operations, such as simultaneously creating or moving hundreds (or even thousands) of virtual circuits such as ATM PVCs or MPLS LSPs.

[5] Such reconfigurations can result in both signaling and routing storms as the network attempts to converge. MPLS networks tend to carry routing and signaling in-band, whereas optical networks carry routing out-of-band. This makes MPLS networks less resilient in the face of massive reconfigurations; hence the need for management systems.

- Management systems can provide networkwide object support for service profiles. Subscriber management on large mobile phone networks is a good example of this. The management system can be used to create thousands of subscriber records and write them into a service database. Individual subscribers can then be updated as they connect to the network.

A good quality NMS broadens the operator's view of the network. This can help to leverage the increasing intelligence of modern NEs.

What Is Network Management?

The preceding sections have described some typical large networks now in common use. They have hinted at issues concerning network management, which is now described more fully. Network management provides the means to keep networks up and running in as orderly a fashion as possible. It includes planning, modeling, and general operation. It also provides command and control facilities. Broadly speaking, the functional areas required for effective network management are:

- Fault: All devices at some point can become faulty, and virtual connections, links, and interfaces can go up or down. These can all cause the generation of network fault data. Events are similar to faults except that they do not necessarily signify

anything is wrong with the network. They exist to inform the management system of important occurrences, such as an LSP becoming operational (i.e., ready to forward incoming traffic).

- Configuration: All devices tend to require some type of configuration or tuning. Configuration settings may be both written to and read from devices. In [Chapter 8](#) we illustrate the configuration of MPLS MIB objects.
- Accounting: Billing for service is an important component of enterprise network management (e.g., for departmental service billing). This function can be used for charging back the use of resources, such as dial-up facilities, to individual departments as well as for verifying the bills submitted by a service provider.
- Performance: As user populations and bandwidth needs grow, it is essential to be able to measure performance, particularly for SLA fulfillment. Performance checks can assist in predicting the onset of congestion.
- Security: Attacks against networks can include unauthorized access, data modification or theft, and so on. Security is needed to ensure that both data and the underlying network are protected.

The above points describe what are known as the OSI functional areas of network management, **FCAPS**, described at length in [\[Stallings1999\]](#). A good management system should fulfill all the FCAPS areas (many products provide only fault and performance management, leaving the other areas to proprietary means or ancillary products). An important point is that the FCAPS areas are inter-dependent. Fault management has to know about the network configuration in order to provide meaningful reports. The same is true of performance monitoring, particularly for complying with SLAs. Likewise, billing (or accounting) has to have some knowledge of the underlying configuration. Providing all this management capability is a big challenge, especially for large distributed networks containing lots of legacy equipment.

Organizations implement their FCAPS functions in interesting ways: Some do all the management inhouse while others outsource it to third parties. Large enterprises can operate a number of different types of networks, such as broadband, IP, and telephony. Each network has to be managed, and in many cases each has its own management system, making overall management complex, error-prone, and potentially very time-consuming. In cases where network management is felt to be too difficult or no longer a core activity, an organization can turn to outsourcing. Outsourcing to a third party can help to alleviate some of the duplication of multiple management systems by connecting the network to a Network Operations Center (NOC). The owner of the NOC then provides billable services in any or all of the FCAPS areas. One merit of using a NOC is that network management costs can become more predictable. However, there are no hard and fast rules about this: An enterprise can also have its own NOC. The use of CPE is another example of outsourcing.

Who Produces Network Management Software?

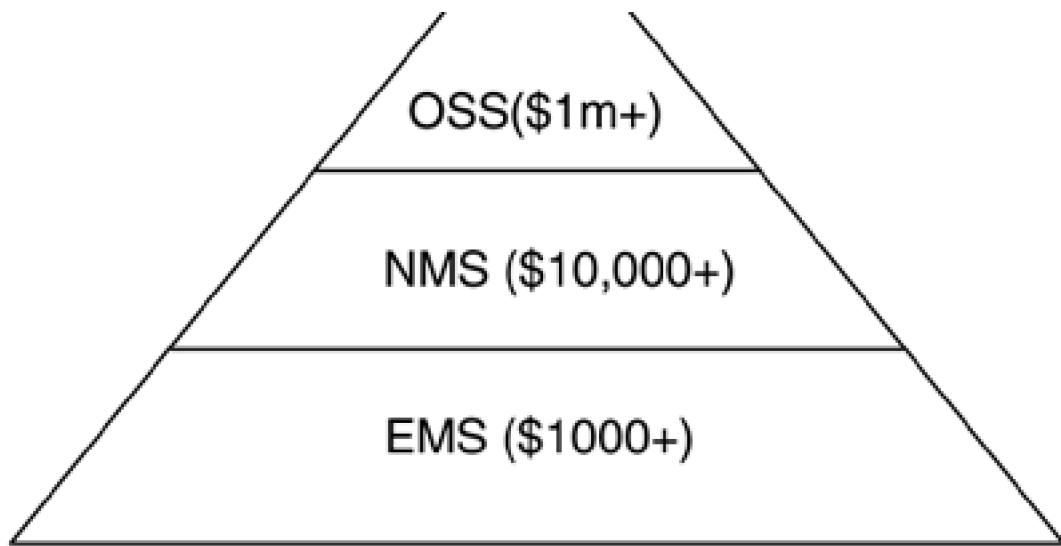
Equipment vendors such as Cisco, Nortel, Hewlett-Packard, and Alcatel generally provide SNMP agents on their devices. Separately purchased, integrated management systems are also available from these and many other organizations. These management systems typically run on UNIX or Windows NT/2000 platforms and feature GUIs, object palettes for topology definition, and fairly extensive FCAPS facilities.

The Management System Pyramid

The owners of large networks tend to functionally separate their software tools, and FCAPS provides a conceptual framework for this. Management systems for large service providers tend to follow a distinct layered structure, illustrated in [Figure 1-6](#) as a pyramid.

Figure 1-6. The management system value pyramid.





There is good reason—both technical and financial—for this functional separation. At the top of the pyramid is the Operational Support System (OSS), which is used for SP business support (feeding into other corporate systems) and overall network support. In passing, we should note that there is some interest in enterprise operational support systems for converged enterprise networks (e.g., AceComm is one vendor in this area). The OSS layer is expensive to develop and deploy. Below the OSS is the NMS, which tends to be used for network-facing operations such as creating, monitoring, and deleting virtual connections—ATM PVCs and MPLS LSPs, for example. The NMS is generally focused on multiple devices at any given time; in other words, it has a networkwide usage. This is reflected in the value of NMS, which can be priced per network node. Very often, NMS are sold in conjunction with NEs in order to help in quickly bringing up and subsequently maintaining the network. Below the NMS is the EMS, which—at the bottom of the NMS food chain—is generally focused on a single device at any given time.

Sometimes the EMS is a separate application hosted on an external platform in just the same way as the NMS. This can be the case where devices are:

- Primitive (not enough onboard capacity for adding management)
- Old or near the end of their lifecycle

Vendors of such devices can add an external EMS to extend the device utility. In other cases, the devices themselves host the EMS, allowing device-centered management facilities such as:

- Software upload/download
- Configuration database backup/restore
- Alarm processing and storage

The guiding principle is always the same for management systems deployment: It should make the object of attention (NE, network, virtual connections, etc.) easier to use.

As we mentioned, the NMS has a networkwide perspective and provides facilities to:

- Create, delete, and modify multiple NE objects, such as VLANs and VPNs.
- Create virtual connections between network devices.
- Create, monitor, and delete various soft objects on network devices such as connections, profiles, and paths.^[6]

^[6] A path in this context is a specific set of nodes and interfaces (on the nodes in question) between two points in the network. Both ATM and MPLS support path objects for use in creating virtual connections. Typically, preconfigured paths are provisioned in the network via signaling. For this reason, paths have an independent significance in an NMS.

- Correlate alarms with connections when a failure occurs.
- Apply actions, such as software (or configuration data) uploads and downloads, on a networkwide basis.

It is important to note that the gap between the EMS and NMS is not always clear cut. Sometimes, an NMS is called upon to process raw device alarms or even to distribute new firmware across a range of devices. The dividing line between EMS and NMS is that most NMS operations tend to simultaneously involve more than one NE. EMS operations tend to center on a single NE. In some cases, an EMS can be developed that runs on a system (such as a PC or UNIX system) external to the NE. The EMS then handles the NE interactions. EMS development for devices without SNMP agents can be quite cumbersome, often involving some type of automated interaction with an NE menu system or CLI. In this, the NE presents its menu options and the EMS emulates a human user and selects the required option, moving to the next menu level.

Even with an EMS that interacts with an NE CLI, it is still possible for an NMS to then interact with the EMS. So, even though the EMS is not onboard the NE, this does not concern the NMS. However, the whole scheme tends to be proprietary in nature. In effect, the approach taken is message-based: The EMS directly exchanges messages with the NEs. This approach is not without difficulties: Different versions of firmware may support slight variations in CLI message format or content, making it difficult to formulate a generic approach to such EMS-NE exchanges. Having SNMP on the devices makes EMS development much easier because then the operations are based on standard SNMP message exchanges.

As we mentioned earlier, many network operators use a script-based approach to setting up and monitoring devices. This consists of writing large and complex vendor-specific scripts that are then sent to the NEs for batch execution. Clearly, this requires scripts that adhere to a given manufacturer's CLI. This is fine if all network devices support the same CLI (i.e., same vendor and CLI version), but this cannot be guaranteed. SNMP, as part of an NMS, provides a better (standard) mechanism for such operations. In [Chapter 6](#) we will see an example of using an NMS to configure IP interfaces. An NMS offers facilities such as security, script management, and audit trails.

Up from the NMS is the OSS layer [[Tele2001](#)]. As mentioned earlier, these are very large bodies of software typically deployed in big SP (and some enterprise) networks. OSS provide a variety of business- and network-support functions such as:

- Subscriber setup and management
- Switching new services on and off
- Workflow ordering for device configuration, connection creation, etc.
- Trouble ticketing (forwarding notifications about faults)
- Asset management

Many OSS can use the services of the underlying NMS to do some or all of the following:

- Retrieve faults and network configuration
- Retrieve performance and billing data
- Execute provisioning

In this way, an OSS uses the NMS services in the manner of an API.^[7] The TeleManagement Forum (TMF) [[TeleMgmtForum](#)] has made great progress in modeling SP operations processes and defining vendor-independent interfaces between OSS and NMS. An example of such a model is the Telecommunications Operations Map (TOM). The TMF has defined Interface Definition Language (IDL) specifications for this purpose. The OSS and NMS both use the IDL for communication.

[7] An API in this context means that the NMS exposes a software interface (e.g., a CORBA interface) to the OSS. This interface might provide services like `retrieveAll-Alarms()` for a given NE. When the OSS uses the interface, the underlying NMS executes the request, retrieves the data, and presents it to the OSS. The use of a standard OSS-NMS API frees the OSS from the need to understand any details of the NMS structures.

One consequence of the connection between OSS and NMS is that it is often hard to decide where and how specific management software should be written. For example, an operator might require the ability to create an ATM SPVCC (or an MPLS LSP) connection between two nodes on its network. Should the NMS vendor provide this capability through its NMS GUI or as an IDL API function?

Probably both, but it's possible that the GUI version might never be used in an SP environment, so should the vendor provide two solutions when only one is needed? These are heady design questions that have a profound effect on the way in which NMS are both built and used.

Other Management Technologies

SNMP is not the only management technology. Other proprietary systems approaches include:

- Microsoft Systems Management Server (SMS)
- Telnet-based menu systems
- Serial link-based menu systems
- Desktop Management Interface (DMI)

Microsoft SMS allows system administrators very flexible control of networks of Windows machines. Software applications deployed on host machines can be determined by remotely viewing the local Windows registry (a type of configuration database) on each machine. This can be very useful for verifying on large sites that software licenses have not been exceeded—too many users installing a given package. SMS also allows software to be distributed to destination machines. This is very useful for updating applications like virus detectors (indispensable nowadays). Remote operations like this greatly facilitate IT support call centers. A shortcoming of SMS is that it works only on Windows machines. SNMP differs from SMS in one crucial way: SNMP is technology-independent. The only local facilities needed for SNMP management are distributed agents with encapsulated MIBs. Management applications then interact with the agents to monitor and control operation.

Telnet refers to a menu-based EMS/CLI style of management. This approach requires the management user to connect to the IP address of a given device using telnet. The device then provides a text menu-based application with which the user interacts. This is useful and is a widely adopted approach for device management. It is generally possible to use telnet to configure devices such as laser printers, routers, switches, and terminal servers. The problem with it is that menu-based management systems are proprietary by their nature and don't easily lend themselves to centralized, standards-based management (as does SNMP).

Serial link-based menu systems are very similar to NEs that support telnet. Just the access technology is different. Normally, a serial link-based system includes simple text menus (accessed via a serial interface) that are used for initial configuration. Typical devices for these facilities include small terminal servers. Often, these devices do not have an IP address, and the user configures one via the menu system. Connecting the device to an appropriately configured PC serial port facilitates this. Again, by its nature this is proprietary.

DMI was developed by the Desktop Management Task Force and is completely independent of SNMP. Its purpose is the management of desktop environments, and it includes components similar to those of SNMP, such as DMI clients (similar to SNMP managers), DMI service providers (similar to SNMP agents), the DMI management information format (similar to the MIB), and DMI events (similar to SNMP notifications).

Network Convergence and Aggregate Objects

The provision of services such as Metro Ethernet and layer 3 VPNs is presenting an interesting network management challenge. Not only are new services being deployed, SP (and to some extent, enterprise) network cores are migrating to layer 3. Managing these converged networks in a scalable, end-to-end fashion is a necessity, especially when competitive SLAs are sold to end users. The service sold to the user may consist of Ethernet (with different priorities supported for specific traffic types), cross-connected into an MPLS core. Modeling this for network management support requires the use of what we call aggregate objects. Aggregate objects are comprised of a number of related managed objects. Examples are VLANs, VPNs, and cross-connect technologies (e.g., Ethernet over MPLS). As the range of technologies and services deployed on networks continues to grow, aggregate objects are becoming increasingly important. We'll see this in [Chapter 8](#) when we discuss LSP creation.

[Figure 1-4](#) introduced us to VLANs. From a network management perspective, VLANs are aggregate objects made up of:

- Switches
- Ports, MAC addresses, IEEE 802.1Q VLAN IDs
- Links between separate VLANs

Generally, there are two ways for an NMS to build up a picture of a VLAN:

- Manual creation by an IT manager
- Automatic discovery from the network

Manual creation requires a combination of human input and network-side provisioning. The user selects the switches required for the VLAN and adds the VLAN members. Provisioning software in the NMS then updates the appropriate MIBs. This is the textbook way of operating networks, but in reality networks may tend to change quite often.

NMS Discovery

In many cases, changes are made to individual switches via the EMS (usually via the onboard CLI) and unless the user manually updates the NMS, then the EMS-NMS pictures may differ. This is where an NMS feature called *network discovery* is important. Network discovery is the process by which an NMS uses SNMP (and also ICMP) to read, process, and store the contents of designated MIB tables. In this way, the NMS picks up any changes made via the EMS. This process of ongoing discovery and update is an important aspect of managing large networks. Network discovery also picks up the details of both simple objects and complex aggregate objects.

Not all NMS products provide automatic network discovery, because it introduces traffic into the managed network. Also, the workflows of the operator may (manually) provide the same service with no need for an automated solution. We will tend to assume that an automated network discovery function exists.

So (using either manual or automatic network discovery), we now have our picture of the network and its higher level constructs (including aggregate objects such as VLANs and VPNs). Having a clear picture of the network objects leaves the operator free to effectively manage the network.

What kinds of things can the operator expect to happen to the network? Links and interfaces can go down; for example, if Link 1 in [Figure 1-4](#) goes down, then VLAN 3 will become isolated from the enterprise network. The NMS (not shown [Figure 1-4](#)) should receive a notification from the network that the link has gone down. The NMS then has to cross-reference the notification with the associated aggregate object (in this case VLAN 3) and infer that VLAN 3 is no longer connected to the network. The NMS should indicate the problem (usually in a visual fashion, such as via a GUI color change) to the operator and possibly even suggest a fix. Similarly, if an NE in one VLAN becomes faulty—for example, if a NIC starts to continually broadcast frames—then the NMS should figure this out (by looking at interface congestion indicators) and reflect it back to the user. The user can then resolve the problem.

The Goal of an NMS

This mechanism of important events occurring in the network and the NMS (and operator) racing to figure out what happened is crucial to understanding NMS technology. **The difference between the NMS picture of the network and the real situation in the network must be kept as small as possible. The degree of success attributed to an NMS is directly related to this key difference.** This is an important NMS concept that we will refer to frequently.

Notifications

We use the term *notification* to mean any one of three different things:

- Events
- Faults
- Alarms

An event is an indication from the network of some item of interest to the NMS, for example, a user logging into an NE CLI. A fault is an indication of a service-affecting network problem, such as a link failure. The NMS must respond as quickly as possible to a fault, even suggesting some remedial action(s) to the operator. An alarm is an indication that a potentially service-affecting problem is about to occur, perhaps an interface congestion-counter threshold that has been exceeded. Clearly, in most cases, faults should be processed by the NMS ahead of events and alarms.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

[PREVIOUS] [NEXT]

SNMP: The De Facto Network Management Standard

Two efforts were made in the 1980s to standardize the area of network management: OSI and SNMP. The intention was that the OSI approach would eventually replace SNMP, but this never happened. In the end, the OSI approach was found to be too complex for widespread adoption and was overtaken by its simpler counterpart. Some OSI inventions, such as ASN.1, did find their way into SNMP. SNMP is a much lighter variant that was globally adopted and has now become the de facto standard for network management. SNMP-based management system components are distributed throughout a network in the form of agents and managers. The appeal of the lightweight SNMP entities is that they consume minimal resources: This can be an aid to scalability (discussed in [Chapter 3](#)). The latter is an important management requirement even on modern networks with highly powerful hosts. The principal components of SNMP are:

- Agents
- Managers
- MIBs
- A communications protocol

The SNMPv3 standard replaces the terms agent and manager with *entity*. While entity is the correct term for SNMPv3, we need to distinguish between the manager (server) side and device (agent) side. So, for clarity, we will continue to use the terms agent and manager. Unless otherwise stated, these refer to SNMPv3 entities. Also, any reference to SNMP from here on should be interpreted as SNMPv3 unless otherwise stated. We now describe these principal components.

The SNMP Agent

SNMP agents are the entities that reside on managed devices. They listen on UDP port 161 for incoming SNMP messages; they use UDP port 162 for sending notification messages. Agents are the workhorses of management and provide the following functionality:

- Implementing and maintaining MIB objects
- Responding to management operations such as requests
- Generating notifications, both traps (unacknowledged) and informs (acknowledged)
- Implementing security—SNMPv1 and SNMPv2c support community-based security with clear-text passwords; stronger security (authentication and encryption) is available with SNMPv3
- Setting the access policy for external managers

SNMPv3 also provides an access control framework, which consists of:

- MIB view—the set of managed objects in an agent MIB accessible to an SNMP manager. This is the manager's client view with respect to the agent.
- Access mode to managed objects—either READ-ONLY or READ-WRITE. A READ-ONLY access mode means that no agent MIB objects can be written by a manager. MIB views are associated with specific access modes.

SNMP agents can be hosted on almost any computing device, including:

- Windows NT/2000 machines

- UNIX hosts
- Novell NetWare workstations and servers
- Many network devices, including hubs, routers, switches, terminal servers, PABXs, and so on

The agent listens on UDP port 161 for the following SNMP message types:

- **Get** requests the values of the specified object instances.
- **Get-next** requests the values of the lexical successors of the specified object instances.
- **Get-bulk** requests the values of portions of a table.
- **Set** modifies a specified set of object instance values.

The above messages either retrieve (**get**) or modify (**set**) NE data as defined in the MIB. The agent uses UDP port 162 for sending notification messages to a preconfigured IP address. Agents reside in the managed network and communicate with managers (described in the next section).

The SNMP Manager

SNMP managers are the entities that interact with agents. They provide the following functionality:

- Getting and setting the values of MIB object instances on agents
- Receiving notifications from agents
- Exchanging messages with other managers

It is unusual nowadays to have to write either SNMP agent or manager programs. Many system software vendors include them as standard software components. For example, all of the following products include an SNMP agent and manager:

- The pSOS [[pSOS](#)] real-time, embedded operating system
- The VxWorks [[VxWorks](#)] real-time, embedded operating system
- The Java JDMK toolkit

In the cases of pSOS and VxWorks, the SNMP agent can be ported to an embedded system, such as a switch or router. This device then constitutes an NE and can be managed by an NMS. The SNMP agent on the NE can be considered part of another component called the EMS (which we met earlier). This is software dedicated to managing the NE. Various mechanisms for accessing the EMS are allowed, including:

- Serial
- Telnet
- SNMP

The NMS generally interacts with the EMS on its managed NEs using one of the above access methods. When an NE is first deployed in its factory-default state, it is often necessary to configure it via a serial interface. The other services and protocols available on the NE can then be enabled so that it can subsequently be accessed over a network. The major focus of this book is the NMS.

In [Chapter 7](#), "Rudimentary NMS Software Components," we build basic Visual C++ and JDMK Java SNMP manager programs. Normally, an SNMP manager is a low-level software entity embedded in a larger body of software called the management application. The

combination of the user, management application, SNMP manager, and multiple distributed SNMP agents constitutes the management system. Facilities offered by a management system are:

- FCAPS
- A centralized database
- Reporting
- Support for many simultaneous client users
- Topology discovery (not all NMS provide this)
- A full-featured, multilevel GUI representing the managed network

Both agents and managers support MIBs. Agents implement their MIB objects and (where appropriate) map them to real NE data. An example of such a mapping is between the `ipInReceives` object (from the IP MIB table) and the underlying NE IP protocol implementation. Strictly speaking, this mapping holds true for a host. For a router, the `ipInReceives` object is maintained by the interface statistics. However, in either case, the `ipInReceives` object maps to a piece of data maintained by a designated section of the NE.

The MIB

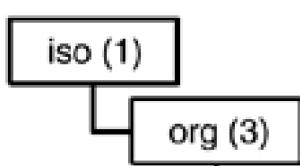
The importance of MIBs cannot be overstated. This is a recurring theme throughout this book. MIBs are a crucial component—perhaps *the* crucial component—of an NMS because they contain the data definitions for the managed objects. In [Chapter 8](#) we use the MPLS MIBs to create LSPs. A MIB is simply a managed-object data description. The MIB defines the syntax (type and structure) and semantics of the managed objects. SNMP managers and agents exchange managed object instances using the SNMP protocol.

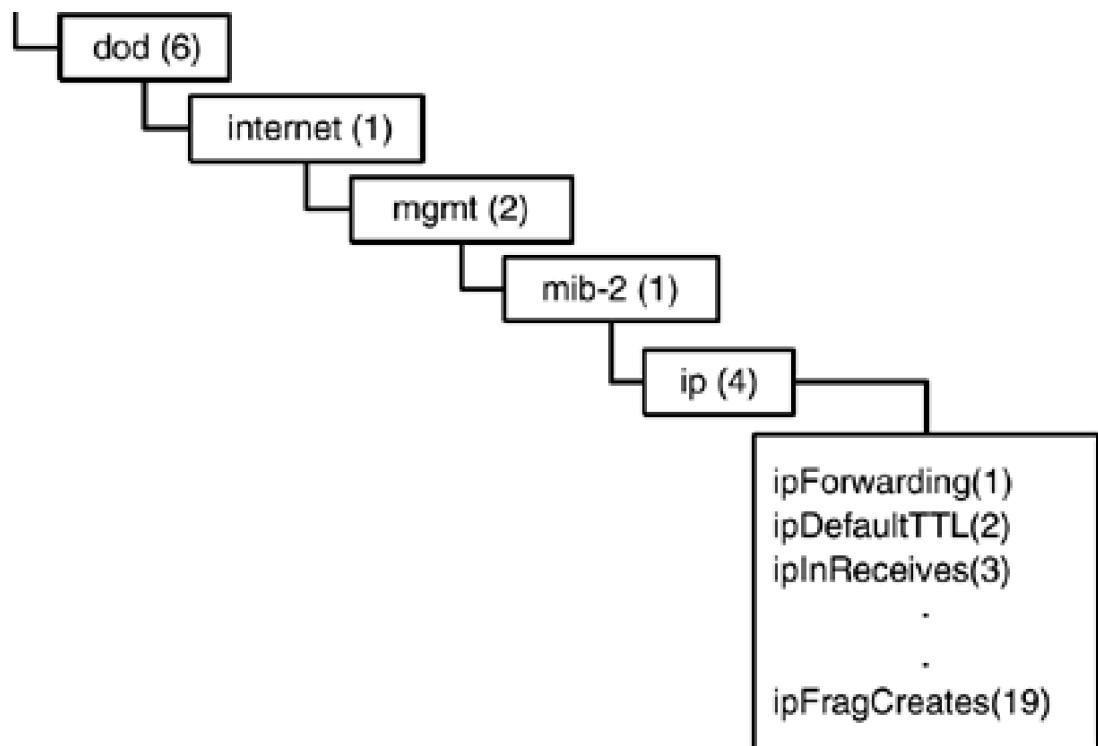
Managed objects may be defined using what are called *textual conventions*. These are essentially refinements of basic types (that are very loosely analogous to programming language data types or even Java/C++ classes), and some of those included in SMIv2 (Structure of Management Information) are:

- `MacAddress` is an IEEE 802 MAC address.
- `TruthValue` is a boolean value representing true (1) or false (2).
- `TestAndIncr` prevents two managers from simultaneously modifying the same object. Setting an object of type `TestAndIncr` to a value other than its current value fails. We will see a similar mechanism used in the MPLS tables.
- `RowStatus` is a standard way for adding and removing entries from a table (we will see this object used many times in the MPLS configuration examples).
- `StorageType` specifies how a row should be stored.

As discussed in the previous section, an example of a MIB object is the number of IP packets received by a host TCP/IP protocol stack from its interfaces. The MIB object called `ipInReceives`, in the IP group, fulfills this function (see [Figure 1-8](#)). Each IP packet received from a registered interface (including those received in error) results in the host agent incrementing the MIB object instance value for `ipInReceives`.

Figure 1-8. The MIB-II IP group.





In addition to using textual conventions, MIB objects have additional attributes that are now described.

MIB Object Attributes

All SMIv2 MIB objects have a number of common attributes, including:

- **SYNTAX:** This is the object format—for example,**Unsigned32** (an integer), **TruthValue** (a Boolean true or false), and **SEQUENCE** (a container of other objects).
- **MAX-ACCESS:** This specifies the accessibility of the object—for example,**read-only** means that the object can only be read (but not written) by managers.
- **STATUS:** This is the state of support for the object in the MIB—for example**current** means that the object is relevant and can or should be supported.
- **DESCRIPTION:** This is a text description of the object.
- **DEFVAL:** This is a default value that the agent can use when the object instance is first created.
- **OBJECT IDENTIFIER:** This is the unique name for a MIB object, described in the next section.

Managers use the object attributes in order to manipulate and understand MIB objects. [Figure 1-7](#) illustrates an object called **mplsFTNAddrType** from the MPLS Forwarding Equivalency Class-To-Next Hop Label Forwarding Entry (FTN) MIB. This important MPLS MIB is described in more detail in [Chapter 9](#), "Network Management Theory and Practice," to illustrate the way in which policy-based management is finding its way into the operation of MPLS NEs. For now, we examine the elements of a single object from this MIB in order to describe the above attributes.

Figure 1-7. MIB object example.

<u>Line Number</u>	<u>Object Attributes</u>
--------------------	--------------------------

1	mplsFTNAddrType OBJECT-TYPE
2	SYNTAX InetAddressType
3	MAX-ACCESS read-create
4	STATUS current
5	DESCRIPTION

"The type of IP packet against which this entry will be matched. If this object has the value `ipv4(1)`, then the objects in this entry of type `InetAddressIpv6` MUST be ignored by management applications."

6	DEFVAL { ipv4 }
7	::= { mplsFTNEntry 6 }

It's very important to be able to read MIBs, so we briefly describe the object in [Figure 1-7](#). The first line is added for information only. It describes the columns in the figure. In the left-hand column is the line number, and the right-hand column shows the attributes (or characteristics) of the object. Real MIBs *do not* contain line numbers or headings like this. So, the real version of this MIB (in an agent or NMS) would not contain either the top line or the line numbers in the left-hand column.

On line 1, we see a MIB object called `mplsFTNAddrType`. This identifies the MIB object with a symbolic name. An NMS (or MIB browser) can do **gets** and **sets** using this name. We know this is a MIB object because of the keyword **OBJECT-TYPE**.

Line 2 indicates the syntax of the object (`mplsFTNAddrType`). It shares the syntax of another object called `InetAddressType` (defined in a MIB called `INET-ADDRESS-MIB`). This illustrates the way SNMP reuses legacy components to build new ones. The **SYNTAX** `InetAddressType` is imported from the latter MIB and represents an IP address string.

Line 3 indicates that the **MAX-ACCESS** (or operational permissions) allowed on object instances of type `mplsFTNAddrType` is `read-create`. This means that a manager can either read an existing object instance or create a new one.

Line 4 indicates that the **STATUS** of `mplsFTNAddrType` is `current`, meaning that this object should be supported.

Line 5 gives a **DESCRIPTION** of `mplsFTNAddrType` and provides a useful textual reason for the use of this object.

Line 6 provides an acceptable default value for instances of `mplsFTNAddrType`. This is indicated by the **DEFVAL** clause and in this case has the symbol value `ipv4` (this has the value `ipV4`, or `1`, as seen in the **DESCRIPTION**). Later we will see the importance of default values in the NMS.

Line 7 indicates the name used to access this object via SNMP—in this case it is column number 6 in the table row called `mplsFTNEntry` (defined earlier in this MIB).

Understanding the contents of [Figure 1-7](#) takes us a long way on the road to understanding MIB objects. We now delve a little more deeply into the overall structure of MIBs.

OIDs and Lexicographic Ordering

All MIB objects have unique names called object identifiers (OIDs). An OID is a sequence of 32-bit unsigned integers that represents a node within a tree-based structure (with a single root). Only an instance of a MIB object can be retrieved from an agent. An instance of a MIB object is identified by an OID concatenated with the instance value. The instance value is a sequence of one or more 32-bit unsigned integers.

The order of the OIDs is an important aspect of SNMP. All objects can be traced from the root in a process called *walking the MIB*. During a walk, each branch of the MIB tree is traversed from left to right starting at the root. For example, the standard IP group or table has the OID `1.3.6.1.2.1.4`, as illustrated in [Figure 1-8](#). The IP group and some of its constituent objects are shown in this diagram.

MIBs are plain-text files. They are compiled into the agent source code and become part of the executable file. If a manager wants to access some agent MIB objects, then either the associated MIB module file is needed or a MIB walk can be attempted.

All MIB objects follow the model depicted in [Figure 1-8](#). The IP object is a table that contains scalar (single-value) objects. It is also possible to have non-tabular scalar objects in the MIB, as we'll see in [Chapter 2](#), [Figure 2-5](#). [Appendix C](#) contains a list for part of the IP table retrieved from a real device.

Another important aspect of lexicographic ordering is that a manager can use it to "discover" an agent MIB. This is for that case in which the manager does not have a copy of the agent MIB and needs to determine what objects the agent supports. The discovery process consists of walking the MIB. It should be noted that this is not a very good way of retrieving agent data. It is far better to have the MIB details at the manager side because the structure and meaning of the NE data will then be apparent.

SNMP Protocol Data Units (PDU)

SNMP managers and agents communicate using a very simple messaging protocol. This is a straightforward fetch ([get](#)), store ([set](#)), and notification model [[ComerStevens](#)]. Managers retrieve agent data using [get](#) operations, and they modify agent data using [set](#) operations. When agents want to communicate some important event, they do so by sending a notification message to a preconfigured IP address. If the agent wants to receive an acknowledgment from the manager, then it sends an inform message.

[Table 1-2](#) illustrates the protocol messages provided by the different versions of SNMP.

Table 1-2. Protocol Data Units in the Different Versions of SNMP

SNMPV1	SNMPV2C	SNMPV3	RESPONSE PDU
GetRequest	GetRequest	GetRequest	GetResponse
GetNextRequest	GetNextRequest	GetNextRequest	GetResponse
SetRequest	SetRequest	SetRequest	GetResponse
Trap	Trap	Trap	None
	GetBulkRequest	GetBulkRequest	GetResponse
	InformRequest	InformRequest	GetResponse

In [Chapter 2](#) we illustrate details of the SNMPv3 message types and their interactions between agents and managers.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Summary

Enterprise and SP networks are complex, interdependent entities. Enterprise network managers seek to improve business processes and workflow efficiencies by leveraging their technology. Service providers can help them achieve this by offering advanced managed or unmanaged (billable) services, such as VPNs. Both types of network have to be managed effectively using dedicated technology. We focus on SNMP-based network management, but it is important to note that this is not the only approach. The trend in networking is towards what we refer to as *aggregate objects*. These can be seen in the many variants of interconnection technologies, such as VLANs. VLANs allow for LANs to be scaled upwards in a controlled fashion because the broadcast domain can be partitioned. This means that individual VLAN members (e.g., the software engineering department) can communicate within the one broadcast domain without its traffic crossing into a neighboring VLAN (e.g., the sales and marketing VLAN). Traffic crosses VLAN boundaries only as required, and this occurs using layer 3 routing. The mix of technologies involved in VLAN-based environments gives rise to aggregate objects. These objects in turn present scalability challenges to network management.

A successful NMS is one that maintains an accurate and up-to-date picture of the managed network. This is a lot harder than it sounds, particularly with the complex mix of technology and traffic types (many now have stringent real-time requirements) found in networks.

NMS constituent technology tends to follow a client/server architecture with many products based on Java technology. A typical NMS product offers a range of applications that fulfills the basic FCAPS areas as well as others, such as reporting and mult-client control.

SNMP provides a distributed model that uses managed-object schema definitions (MIBs) on remote devices. Instances of managed objects can be retrieved from agents on remote NEs. This can be done by a manager in conjunction with a local copy of the agent MIB; that is, there are two copies of the MIB. MIB structures often must be reflected in the data model (more on this later, but for now the data model is the way the NMS looks at the information relating to the managed objects). For this reason, the NMS quality can suffer if the MIBs are badly written. The mapping of MIBs to real NEs is reasonably easy to understand, particularly after using a MIB browser application (some are freely available on the Web).

A security scheme protects the agent data as well as the data in transit from the agent. A notification mechanism allows agents to asynchronously send messages to a manager when important events (such as faults) occur.

SNMPv3 offers a small number of protocol messages designed to allow effective management of NEs.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Chapter 2. SNMPv3 and Network Management

Having seen the reasons why network management is needed, we now delve a little more deeply into SNMPv3. We look at the SNMPv3 message formats and see how this version of SNMP provides some useful new message types as well as strong security. We then review some of the problems affecting all versions of SNMP.

We described in [Chapter 1](#), "Large Enterprise Networks," that SNMP is essentially a network management data access technology. As such, SNMP is an enabler for a variety of applications including NMS products. However, other applications, such as MIB browsers, also use SNMP. We take a look at some sample output from one of our two handcrafted SNMP applications, described later in [Chapter 7](#), "Rudimentary NMS Software Components."

MIBs are one of the cornerstones of this book, so we take a closer look at the main elements of one of the IETF MPLS MIBs. All of our discussion so far has aimed toward providing a clear picture of the purpose and structure of a typical NMS. An analogy for an NMS is briefly explored to solidify the concepts.

The purpose of an NMS is to manage networks of devices or NEs. We briefly review some typical software components of an NE before presenting our first chunk of MPLS technology.

[Team LiB]

◀ PREVIOUS NEXT ▶

SNMPv3 Structure

SNMPv3 provides a modular structure that allows for specific subsystems to be used for certain tasks. This approach is in line with the increasing trend toward component technology (we discuss this later, but for now just think of components as real-world mini-objects that are embodied in software. Components are joined together to form more complex entities, such as VLANs, virtual connections made up of Ethernet cross-connections joined across an ATM/MPLS core network, etc.). Broadly speaking, an SNMPv3 entity consists of two main components:

- An SNMP engine
- A collection of SNMP applications

Our discussion of SNMPv3 is more of an overview than a detailed description. The latter can be found in [Zeltserman1999](#).

SNMPv3 Engine

The SNMPv3 engine is made up of four subcomponents:

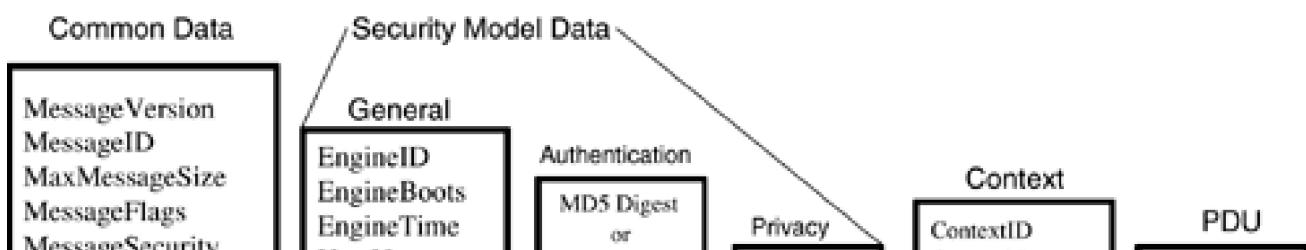
- *Dispatcher* handles message sending and receiving.
- *Message subsystem* handles message processing for SNMPv3, SNMPv2c, SNMPv1, and any other models.
- *Security subsystem* handles security processing for SNMPv3 user-based security model (USM), SNMPv1/v2c community-based security model, and any additional (newly defined) models.
- *Access control subsystem* handles the granting/rejecting of access to specific managed objects.

Two important points to note about the engine subcomponents are that they:

1. Can hand off the message processing to each other as required.
2. Are themselves extensible entities.

The SNMPv3 architecture is flexible and modular. It remains to be seen whether this facility will be used over time, but one area where change is quite likely is that of security. Another security model could be added to the architecture by extending the security subsystem and adding an extra value in the security model number field (as illustrated in [Figure 2-1](#) with the **MessageSecurity** parameter). Such a change would require a potentially costly software upgrade, but the benefits of extra security may become a necessity.

Figure 2-1. SNMPv3 message format.



[[UserName](#)] [[SHA Digest](#)] [[DES Key](#)] [[ContextName](#)] [[PDU Types](#)]

[[Team LiB](#)]

[[PREVIOUS](#)] [[NEXT](#)]

[Team LiB]

◀ PREVIOUS NEXT ▶

SNMPv3 Applications

There are currently five SNMPv3 applications defined:

1. *Command generators* create SNMP messages.
2. *Command responders* respond to SNMP messages.
3. *Notification originators* send trap or inform messages.
4. *Notification receivers* receive and process trap or inform messages.
5. *Proxy forwarders* forward messages between SNMP entity components.

The SNMPv3 framework allows other applications to be defined over time and added to the above list.

[Team LiB]

◀ PREVIOUS NEXT ▶

SNMPv3 Message Formats

RFC 2572 contains the SNMPv3 message format specification; an extract is illustrated in [Figure 2-1](#).

The message format is broken down into four overall sections made up of the following:

- *Common data*: These fields occur in all SNMPv3 messages.
- *Security model data*: This area has three subsections—one general, one for authentication, and one for privacy data.
- *Context*: These two fields are used to provide the correct context in which the protocol data unit (PDU) should be processed.
- *PDU*: This area contains an SNMPv2c PDU.

Both the context and PDU areas are either encrypted or in plain text. The format used for SNMPv3 messages follows a left-to-right and top-to-bottom pattern. So, from [Figure 2-1](#), the first field in the message is **MessageVersion**, the next field is **MessageID**, and so on. We describe the **SNMPv3** message fields in the following sections.

MessageVersion

The first field in the message is the SNMP version. This is located in the same position for all versions of SNMP to allow differentiation during message processing. This provides for backwards and forwards compatibility. A value of 3 in this field indicates an SNMPv3 message. A value of 2 indicates SNMPv2c and a value of 1 indicates SNMPv1.

MessageID

The **MessageID** is a number used between two entities for message correlation. So, if a manager sends **GetRequest** with **MessageID x**, then it is important that the manager does not re-use **x** until the outstanding message is answered or timed out. The PDU contains a request ID field, which was used for the same purpose in SNMPv1 and SNMPv2c, but since SNMPv3 allows for encrypted PDUs, the **MessageID** is in the (unencrypted) header. The **MessageID** also provides a means by which multiple copies of a response (e.g., if the underlying datagram service duplicates a message) can be distinguished. Messages re-transmitted by a manager should use a new **MessageID**.

MaxMessageSize

The **MaxMessageSize** is the maximum message size supported by the sender of the message. This is the largest size packet that the transport protocol can carry without having to use fragmentation. The receiver uses the value of **MaxMessageSize** to ensure that its reply is within the allowed size range.

MessageFlags

The **MessageFlags** object is 1-byte long and determines the authentication and privacy settings for the message. It also indicates if this message requires a (report) response from the receiver. The three right-most bit positions are used when encoding this object, and the following are the allowed combinations:

- No authentication and no privacy (bit values 000)
- Authentication and no privacy (bit values 001)
- Authentication and privacy (bit values 011)

All three of the above may have the report option set. This indicates that a response is required.

MessageSecurity

The **MessageSecurity** is an integer object that indicates the security setting associated with the message. The range of values supported is as follows:

- 0 is reserved for "any."
- 1 is reserved for SNMPv1.
- 2 is reserved for SNMPv2c.
- 3 is reserved for USM.
- 4–255 is reserved for standards-track security models.

Values above 255 can be used to specify enterprise-specific security models. The receiver must use the same security model when executing its security processing. The security subsystem handles the processing of this section of the SNMPv3 message.

Security Model Data: General

The general section of the security model data contains the following fields:

- **EngineID:** unique identification of an SNMPv3 engine
- **EngineBoots:** the number of times an SNMP engine has either been started up or reset since the value of **EngineID** was last modified
- **EngineTime:** the number of seconds that have passed since the value of **EngineBoots** was last modified
- **UserName:** the name of a user

The above fields precede the authentication and privacy data areas. **EngineID** and **UserName** are used to form an index into a table called **usmUserTable**. This table stores the security model data for a given engine ID and user pair.

Security Model Data: Authentication Protocol

Two authentication protocols are supported in SNMPv3, namely MD5 and SHA. Both protocols serve the same purpose: that of authenticating the SNMPv3 message. The MD5 algorithm calculates a 16-byte (128-bit) digest and the first 12 bytes (96 bits) are included as part of the message in the Authentication field in [Figure 2-1](#). The user must select a 16-octet secret key for use in the MD5 algorithm. If the user opts for the SHA authentication algorithm, then the (SHA) algorithm calculates a 20-byte (160-bit) digest and again the first 12 bytes (96 bits) are included as part of the message in the Authentication field in [Figure 2-1](#). The user must select a 20-octet secret key for use in the SHA algorithm.

Whichever algorithm is used, the authentication protocol field is a 12-byte octet string used as an electronic fingerprint (or message authentication code) to authenticate the message. It is similar to the cyclic redundancy check (CRC) codes used in many applications (ATM, disk drives, etc.) to verify that data has not been modified in transit. When an SNMP entity (i.e., a manager) wants to send an SNMP request to another entity (i.e., an agent), it must use a secret authentication key (described in the previous paragraph) known to both parties. This key is used to generate the fingerprint. When the authenticated message is received, the fingerprint is recalculated, and if the two match, then the message is deemed to be authentic.

Security Model Data: Privacy Protocol

The privacy protocol field is an 8-byte octet string used for the Data Encryption Standard (DES) algorithm. The encryption uses a 16-byte key. The first 8 octets of the 16-octet secret key are used as a DES key. The second 8 octets are used as an initialization vector; this is a unique 8-octet value that is manipulated to ensure the same value is not used for encrypting different packets. Again, both parties use a secret private key to encrypt and decrypt messages.

Context

The historical background for SNMPv3 context is interesting. It arose from discussions about how to deal with cases in which a given MIB table already exists with a specific indexing scheme, but the indexing scheme must be extended. Some tables in the Bridge MIB are indexed by port number, and in a rack-based system or a stacked system, there may be multiple cards or units with the same port numbering. Contexts were invented to allow multiple instances of the same MIB table within the same SNMP agent in order to handle cases like this.

ContextName is an octet string, and **ContextID** uniquely identifies an entity that may recognize an instance of a context with a particular context name. The context details are considered part of the PDU field.

PDU

This object represents either an unencrypted (plaintext) PDU or an encrypted PDU. The value of the **MessageFlags** object dictates which one is the case.

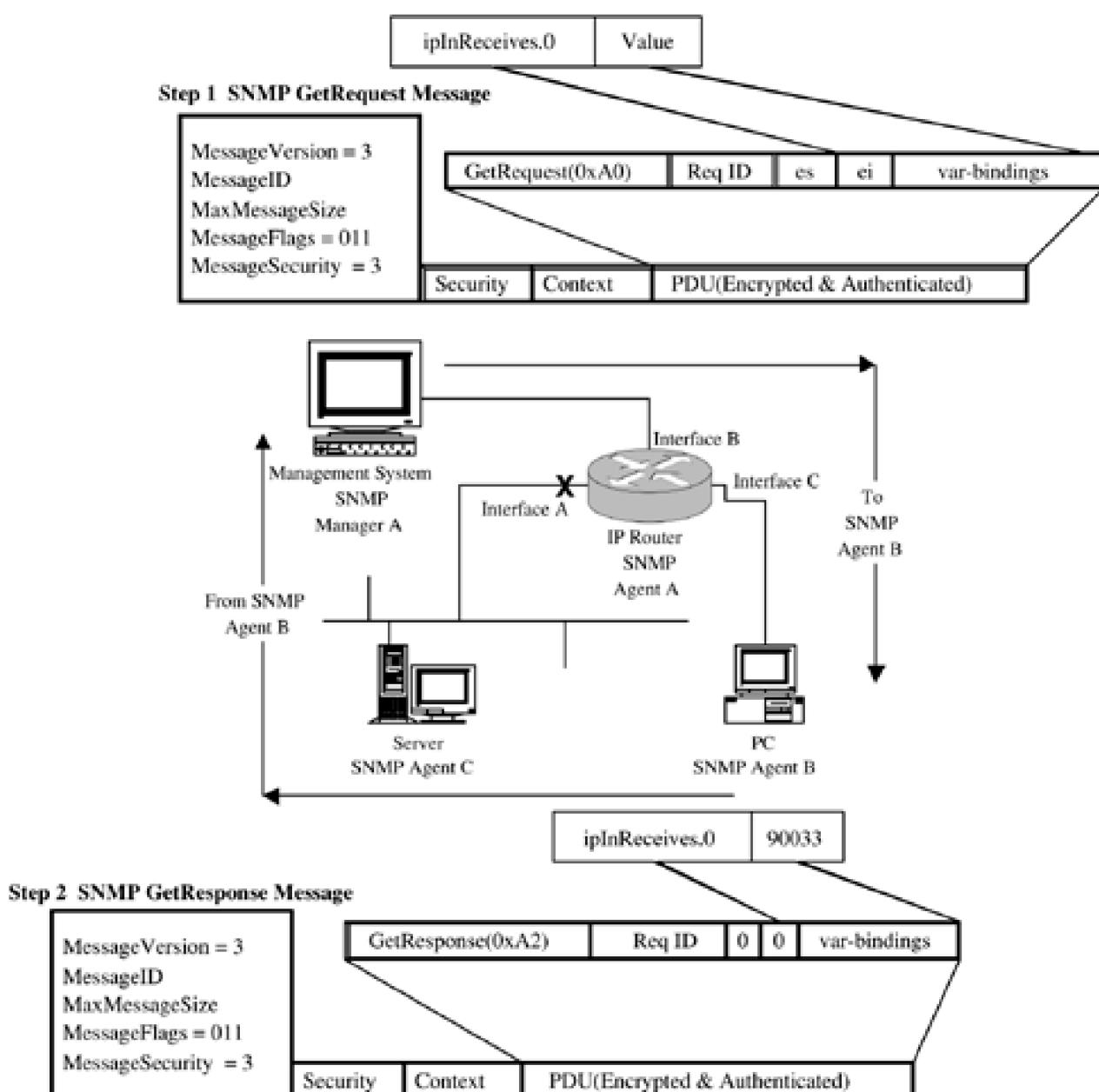
SNMPv3 Security and View-Based Access Control

An important point to note about the SNMPv3 USM is that it provides authentication and privacy at the message level. The view-based access control mechanism operates at the PDU level and determines if access to a given MIB object should be granted to a principal (or user). These issues are comprehensively covered in [\[Zeltserman1999\]](#).

SNMPv3 Message Exchanges

We now look at some SNMPv3 message exchanges, using [Figure 2-2](#). In an effort to solidify the above concepts, [Figure 2-2](#) has a good deal of detail—hopefully not too much.

Figure 2-2. SNMP GetRequest and GetResponse messages.



[Figure 2-2](#) illustrates a network with a management system (containing SNMP Manager A) connected to an IP router. The router has three network interfaces: A, B, and C. It hosts an SNMP agent (Agent A). The management system host is connected to Interface B on the IP router. Router interface A is connected to another network segment on which a server is located. The server hosts SNMP Agent C. Router

Interface C is connected to a PC that hosts SNMP Agent B.

An important point to note is that the PDU fields in [Figure 2-2](#) are SNMPv2c PDUs; that is, SNMPv3 introduced no new PDUs. So, let's take a look at [Figure 2-2](#) starting with a **GetRequest-GetResponse** message exchange.

SNMPv3 GetRequest

Manager A in [Figure 2-2](#) wants to retrieve the value of the **ipInReceives.0** object instance from Agent B. So, Manager A builds a GetRequest message. The network operator is a little worried about hackers, so the message is authenticated and encrypted before being sent across the network. Step 1 is now complete. Agent B receives the message, processes it (applying the required security processing), and retrieves the required MIB object instance. Next, Agent B builds a response message, applies the required security, and sends the message back to Manager A. Step 2 is now complete. After verifying the message security, Manager A will now extract the required data and store it in some application-specific fashion (usually in a database). A few points can be made about [Figure 2-2](#):

- SNMPv3 is used.
- The first field in the PDU has the value **0xA0 (get)**.
- The value of **MessageFlags** is binary 011; that is, the message is authenticated and encrypted.
- The value of **MessageSecurity** is 3; that is, the SNMPv3 USM is employed.
- The **es** (error-status, the overall result of the operation) and **ei** (error-index, the position of the first object in the variable bindings with which an error occurred) fields are always zero for a **GetRequest**.
- In the **GetResponse** message, the first field in the PDU has the value **0xA2 (getResponse)** and the values of **es** (error-status) and **ei** (error-index) are both zero; that is, no errors occurred in retrieving the MIB object instance.
- A response message is created (by Agent B) with the variable bindings object instance value set to 90033. The agent pushes this value into the same space provided in the received PDU.

Manager A now has the required data. Usually, an NMS makes many such requests simultaneously, often requesting entire tables at a time.

SNMPv3 Get-NextRequest

If Manager A wants to perform a **getNextRequest** on the **ipInReceives.0** object, then the only differences required in [Figure 2-2](#) are as follows:

- The first field in the PDU has the value **0xA1 (getNext)**.
- The response includes the lexical successor to **ipInReceives.0**, for example, **ip.ipDefaultTTL.0**.

After this message exchange, Manager A has the required data.

SNMPv3 GetBulkRequest

GetBulkRequest is a clever way of retrieving a range of objects from a table. The required objects are provided in a variable-bindings list. The objects are retrieved based on the values of two numbers:

- Non-repeaters: Objects for which one get-next is required
- Max-repetitions: Objects for which more than one get-next is required

So, let's say we want to retrieve the number of interfaces on a given NE and then use that number to retrieve the speed of those interfaces. This can be done with one or more **getRequests**, but we can do it in one step using **getBulkRequest**. Our non-repeater is the object **interfaces.ifNumber**. This value will also be used to specify the max-repetitions for the object **interfaces.ifTable.ifEntry.ifSpeed**. So, the call to a conceptual **GetBulkRequest** API might look like the following:

```
GetBulkRequest(non-repeaters = 1, max-repetitions = interfaces.ifNumber,  
varBindList = {interfaces.ifNumber, interfaces.ifTable.ifEntry.ifSpeed } )
```

GetBulkRequestNonRepeater = interfaces.ifNumber.0
Type and Value = Integer32 5 =====> So, the number of interfaces is 5

GetBulkRequestMaxRepetitions of 5 =====> We now get the 5 interface speeds

```
Variable = interfaces.ifTable.ifEntry.ifSpeed.1  
Value = Gauge32 155000000  
Variable = interfaces.ifTable.ifEntry.ifSpeed.2  
Value = Gauge32 155000000  
Variable = interfaces.ifTable.ifEntry.ifSpeed.3  
Value = Gauge32 100000000  
Variable = interfaces.ifTable.ifEntry.ifSpeed.4  
Value = Gauge32 4294967295  
Variable = interfaces.ifTable.ifEntry.ifSpeed.5  
Value = Gauge32 4294967295
```

From this, we can see that the host (in this case, an MPLS label edge router) to which the **GetBulkRequest** was sent has five high-speed interfaces supporting bit rates of 155Mbps (155000000), 100Mbps (100000000), and 4Gbps (4294967295) respectively.

If Manager A wants to execute a **getBulkRequest** on the IP table, then the only differences required in [Figure 2-2](#) are the following:

- The first field in the PDU has the value **0xA5 (getBulk)**.
- The **es** field stores the non-repeaters value.
- The **ei** field stores the max-repetitions value.

After this message exchange, Manager A has the required data. Typically, this type of operation might occur during a discovery procedure; that is, NE x has been found, so we discover its attributes (number and type of interfaces, speeds, etc.).

SNMPv3 SetRequest

A **SetRequest** message follows a very similar set of steps. The only differences required in [Figure 2-2](#) are the following:

- The first field in the PDU has the value **0xA3 (set)**.
- The required value of the object is encoded in the variable-bindings field.

After this message exchange, Manager A has modified the required data. Typically, this type of operation might occur during a provisioning procedure; that is, we wish to alter some data in NE x, so we execute a set (e.g., add a new row to a MIB table or reset a

counter to zero). We will see examples of this in the MPLS MIB tables in [Chapters 8 and 9](#).

SNMPv3 Notifications

We now describe the notification mechanism. A notification message can be either a trap or an inform. Let's now look at an example of a notification. Agent A on the IP router in [Figure 2-2](#) now detects that one of its three network interfaces has gone into the down state (link failure is a commonly occurring hardware fault). This is illustrated in [Figure 2-2](#) with an X on Interface A. It can no longer send or receive network traffic on that interface. The IP router agent has to notify its registered manager of this event, so it sends a notification message to Manager A. Manager A receives the notification, processes it, and realizes that the host for Agent A now has only two working network interfaces. Unfortunately, Manager A can no longer contact SNMP Agent C. Typically, this event would be propagated upwards to a GUI topology, where the associated network link icon (for the link attached to Interface A) would change color to red. Or, the subnet containing the router could change color. Manager A could then poll the router MIB to verify the interface state. The notification has fulfilled its purpose, because the problem can now be resolved. This is the power of notifications: Intelligence is distributed in the SNMP agents, and they emit notifications if and when problems occur. It is then up to the management system to try to resolve the problem if one exists. Notifications do present scalability concerns, particularly as network sizes increase. Many notifications occurring simultaneously can have unforeseen consequences for both the network and the management system.

We now briefly describe the notification PDU and start with a look at an SNMPv1 Trap PDU, illustrated in [Figure 2-3](#).

Figure 2-3. SNMPv1 trap.

Type	Ent	Addr	Gen	Spec	Time	Varbind
------	-----	------	-----	------	------	---------

The trap PDU fields in [Figure 2-3](#) have the following meanings:

- **Type** has the value **0xA4** for traps.
- **Ent** is the enterprise agent software that generated the trap. This is encoded as an OID in the enterprise subtree. A Cisco **sysObjectID** for a 7200 router has the value 1.3.6.1.4.1.9.1.223, where 1.3.6.1.4.1 is the MIB-II enterprise branch, 9 is the Cisco-assigned enterprise number, and 1.223 represents a Cisco product (the value 1 indicates this) with 223 as the product ID.
- **Addr** is the agent IP address.
- **Gen** is the generic-trap field for which there are seven definitions **coldStart(0)** or agent reset; **warmStart(1)** or agent reinitialization, **linkDown(2)** or a link has gone down (the interface is the first object in the **varbind**); **linkUp(3)** or a link has gone up (the interface is the first object in the **varbind**); **authenticationFailure(4)** or an SNMP message has failed authentication (we see an example of authentication failure in [Chapter 7](#)); **egpNeighborLoss(5)** or an EGP neighbor has gone down; and **enterprise-specific(6)**.
- **Spec** is the enterprise-specific trap. A problem with this is that of reuse of the same value by different vendors. This necessitates extra work in having to figure out which vendor generated the trap by looking at the enterprise value. The SNMPv2 trap definition helps to solve this problem.
- **Time** is the time stamp for the trap represented by the value of **sysUpTime** when the trap was generated.
- **Varbind** is the variable bindings object that allows for the encoding of different traps.

[Chapter 7](#) has an example of some SNMPv1 traps that occur during a security violation [Figure 7-11](#). We now briefly look at an SNMPv2 trap PDU in [Figure 2-4](#).

Figure 2-4. SNMPv2 trap.

Type	ReqID	0	0	Varbind
------	-------	---	---	---------

The fields in [Figure 2-4](#) are identical to those of `get`, `get-next`, or `set PDU`. The only difference is the `type` value of `0xA7`. The main difference between this message and an SNMPv1 trap is that the variable-bindings field (often called the `varbind`) is made up of:

- `SysUpTime.0`
- `SysTrapOID`
- An OID representing the SNMPv2 trap

Each SNMPv2 trap is defined in the MIB using the `NOTIFICATION-TYPE` macro [\[Zeltserman1999\]](#). Typically, an NE emits a notification when it wants to inform the manager of some important event or fault, such as a link going into the down state.

The last PDU we will look at is the SNMPv2 inform. The only difference between an inform and an SNMPv2 trap is that the `type` value for an inform is `0xA6`. Informs use a timeout/retry mechanism in an effort to ensure delivery to the manager. By their nature, notifications occur at undefined moments in time. Once a notification message is received, the NMS must decode it and then try to figure out the origin of the problem. This is sometimes called root-cause analysis, which when successful, allows the network operator to understand the exact nature of the problem that caused the notification. Root-cause analysis should also help the user in fixing the problem (if one exists).

Access Rights

An important point to note is that for `get` and `set` operations to succeed, the manager must have the appropriate access rights. This means that the access policy (mentioned earlier) must be configured to allow the manager appropriate read and write access. If a manager attempts an operation for which it does not have access privileges, then the operation will fail.

Message Size

Another important point is that SNMP management messages can refer to many objects, not just to one, as in the preceding examples. In other words, the SNMP `GetRequest` message in [Figure 2-2](#) can include more objects than just the `ipInReceives` object (up to the maximum size allowed by the transport service). However, agents will generally have a maximum packet size that they can handle. A manager must be prepared to handle the case in which an agent packet-size limit is too small for it to return instances of all objects which the manager requested. In this case, the manager will probably need to separate the requests into multiple packets.

SNMPv3 Security

As we saw in [Figure 2-2](#), SNMPv3 provides both authentication and encryption (privacy). Authentication is provided by the industry-standard MD5 hashing scheme or by Secure Hash Algorithm (SHA), and privacy is provided by DES. The configuration settings required on the agent side are generally as follows:

- SNMPv3 user name
- Security level, which can be one of `noAuthNoPriv`, `authNoPriv`, or `authPriv`
- Privacy protocol

- Privacy password
- Authentication password
- Authentication protocol, which can be one of MD5 or SHA-1

The settings for [Figure 2-2](#) consist of **authPriv** (i.e., both authentication and privacy). The two passwords are used during message encryption and authentication. For enhanced security, it is important that network operators change these passwords regularly.

Problems with SNMP

SNMP is a far-from-perfect technology. Some of the more serious problems with it include the following:

- SNMP is not transaction-oriented but instead offers an all-or-nothing style of execution. This can give rise to inconsistent MIB states when an exception occurs during the execution of a number of interrelated set operations.
- It is difficult to manipulate very large data sets.
- Scalability issues where tables grow to include thousands of rows.
- Notifications are not guaranteed to arrive at their destination. Inform requests, which are acknowledged notifications, make use of a timeout/retry mechanism, but even this does not guarantee delivery.
- Management operations (such as **get** or **set**) can time out if the network is congested or the agent host is heavily loaded.
- SNMP messages use the UDP protocol (best-effort datagram service).

Despite these shortcomings, the widespread deployment and simplicity of SNMP are among its greatest strengths.

The Different Versions of SNMP

The versions of SNMP in widespread commercial use are:

- SNMPv1
- SNMPv2c
- SNMPv3

SNMPv1 has community name-based security and includes fairly coarse-grained error handling. For example, when a **GetRequest PDU** includes more than one variable, then either all or none of the values are returned. A failed **SNMP set** operation will generally result in the manager receiving a **GetResponse PDU** containing "Bad Value" and indicating the problem variable. This is of limited use for debugging in operational environments. The issue of "holes" in SNMPv1 tables is particularly troublesome. If a **GetRequest** is sent to an agent for a given MIB object instance and the object has no value, then the agent replies with a "No such name" error. This is not very useful information and makes tabular retrieval a very fragile proposition.

SNMPv2c provides the same security as SNMPv1. It also adds a new message called **getBulkRequest** (that we saw earlier) that allows multiple rows of tabular data to be retrieved in one operation. It allows the sender to specify that **getNext** be used for a range of managed objects. SNMPv2c also provides better error reporting than SNMPv1.

SNMPv3 also supports the **getBulkRequest** message and supports three security settings (again, as we saw earlier):

- No authentication or privacy (equivalent security to SNMPv1/v2c community strings)
- Authentication with no privacy—the manager is authenticated but data is not encrypted
- Authentication and privacy—the manager is authenticated and data is encrypted

As we have seen, the strong security of SNMPv3 is a compelling reason for its adoption. The configuration of SNMPv1 and SNMPv2c agents consists of community strings (and trap/notification destinations). Usually, two community strings are used, one for gets and one for sets. The "get" password is usually "public" and the "set" password is usually "private." We will see this in action in [Chapter 7](#). SNMPv3 configuration consists of (at a minimum) selecting authentication/encryption protocols and specifying (if applicable) authentication and encryption passwords. These settings are written to the agents (or SNMPv3 entities) and must then be used by the NMS in its message exchanges with the agents.

SNMP Applications: MIB Browsers

MIB browsers are specialized tools used to examine the values of MIB object instances on a given agent. A MIB browser can be a fully integrated GUI-based application or a simple text-based one. Regardless of the packaging, they are indispensable for NMS developers and are also very useful for learning about SNMP. Typically, a MIB browser allows a user to "load up" (or compile) a set of MIB files and then view the values of the associated object instances. If a given object instance value is changed (i.e., set) by an NMS, then the MIB browser allows the user to see (i.e., get) the modified value—a simple but very powerful facility. [Table 2-1](#) lists the IP Group leaf objects, one of which was seen earlier in [Figure 2-2](#). These object instances are part of the output of a MIB walk on the IP Group from an NT workstation. The tool used to generate this data was the Microsoft Visual C++ SNMPv1 sample program, which is described in [Chapter 7](#).

Table 2-1. Sample MIB Walk on the IP Group of a Host

MIB OBJECT NAME	OBJECT TYPE	OBJECT INSTANCE VALUE
Ip.ipForwarding.0	INTEGER	2
Ip.ipDefaultTTL.0	INTEGER	128
Ip.ipInReceives.0	Counter	90033
Ip.ipFragCreates.0	Counter	0

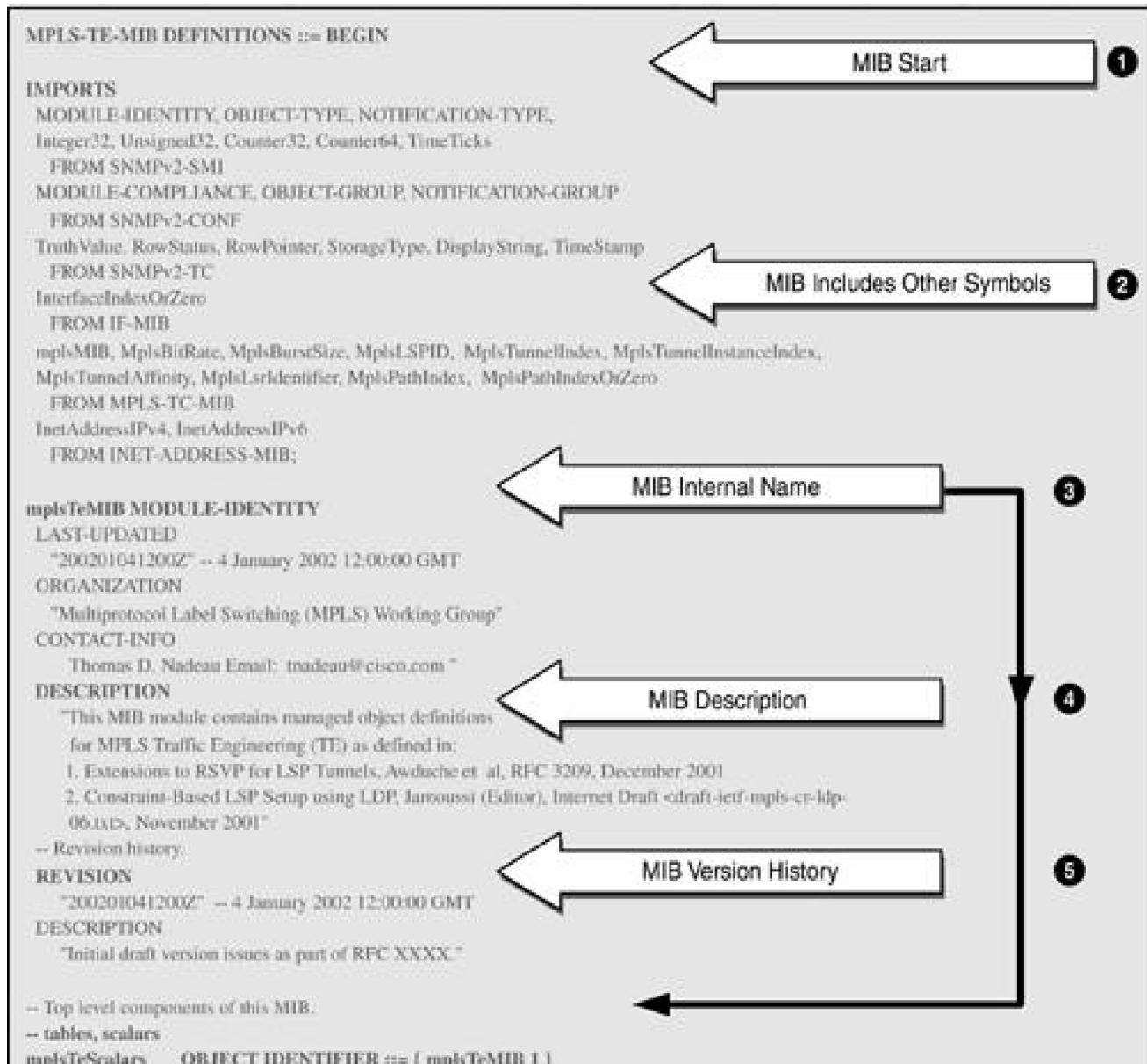
In the MIB object name column, each object has a zero appended. This illustrates the difference between a MIB object definition and its instantiation in a real NE. An instantiated object has a value appended to it. Scalar (nontabular) objects always have zero appended. Tabular objects have an index appended. In the case of the IP Group illustrated in [Table 2-1](#), the objects are all scalar and so have 0 appended.

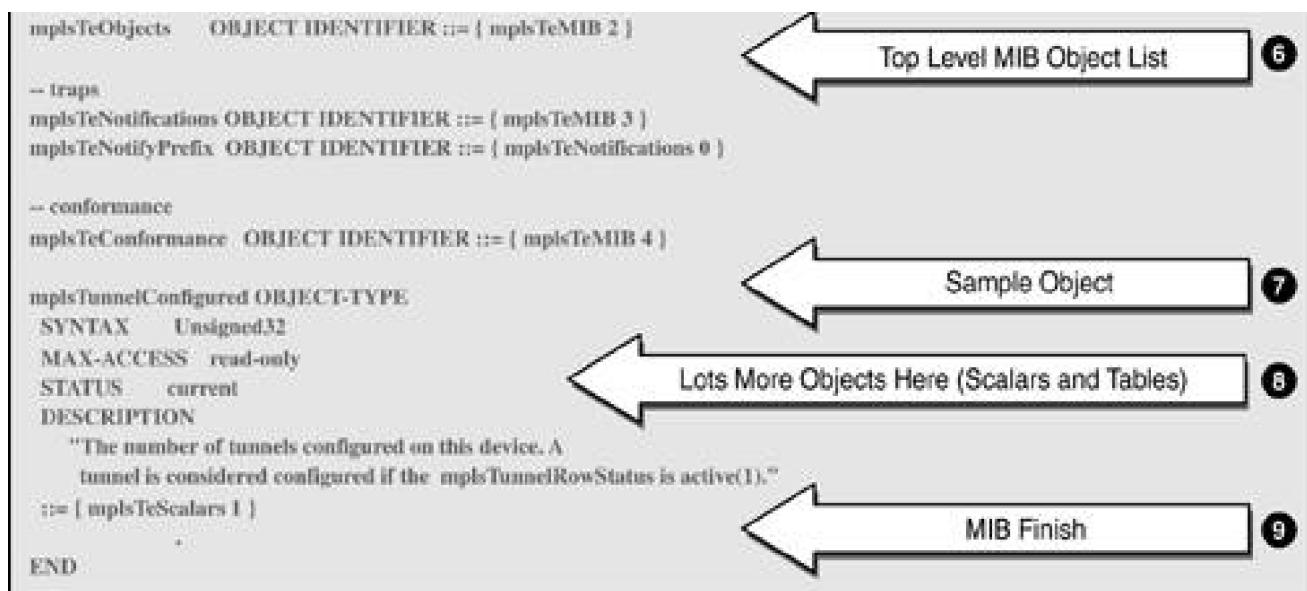
A Closer Look at a MIB

Like all great ideas, MIBs are fairly simple to understand. They provide a detailed description of the managed objects supported by a given device. As mentioned earlier, the MIB defines managed objects in a conceptual way, including the syntax and semantic information about each object. These managed objects can then be instantiated as real objects in an agent host device. [Figure 2-5](#) is an extract from one of the draft MPLS MIBs [[IETF-TE-MPLS](#)] taken from the IETF Web site. As usual, we try to present the overall picture of a complete MIB. We will examine this MIB more closely in [Chapter 8](#), "Case Study: MPLS Network Management." It illustrates most of the general detail needed to understand MIBs. From [Figure 2-5](#), we can see that MIBs are made up of just a few sections clearly identified by keywords. The main points of [Figure 2-5](#) (identified by numbers and corresponding arrowheads) are the following:

1. The **BEGIN** keyword indicates the start of the MIB (arrow 1).
2. The **IMPORTS** keyword introduces descriptors from external MIBs in a similar way to `#include` in C and `import` in Java. The **IMPORTS** statement identifies the descriptor and the module in which it is defined (arrow 2).
3. The **MODULE-IDENTITY** keyword describes an entry point name for objects defined later in the MIB. The objects defined further down "hang" off this name (arrow 3), as shown by the black arrowed line.
4. The **DESCRIPTION** keyword provides details about the MIB content (arrow 4).
5. The **REVISION** keyword indicates the change history of the MIB (arrow 5).
6. The **OBJECT IDENTIFIER** keyword defines either new managed objects or placeholders for them in the MIB (arrow 6).
7. A sample, scalar, read-only integer object, **mplsTunnelConfigured**, is shown (arrow 7).
8. The remainder of the MIB (more scalars and tables) is skipped over (arrow 8).
9. The MIB finishes with the **END** keyword (arrow 9).

Figure 2-5. An extract from one of the draft-standard MPLS MIBs.





[Figure 2-5](#) therefore includes most the elements of a MIB structure that will be encountered in practice.

MIB objects can be scalar (such as integers) or tabular (rows of other objects). In [Chapter 8](#) we look closely at tables, particularly the MPLS MIB tables. The objects defined in the MIB are instantiated in the agent host and can be retrieved using a **get** operation via a MIB browser. Similarly (if they are read-write), they can be modified using a **set** operation. The SNMP agent asynchronously dispatches device notifications. Notifications are sent to a preconfigured IP address, usually that of the NMS.

Managed Objects

Managed objects are the basic unit of exchange between an NMS and NEs. The managed objects are defined in the MIB and deployed in the network. The NMS provides software that, combined with the managed objects, gives the user the means of operating and maintaining the network. The importance of MIBs and managed objects cannot be overstated. The managed objects defined in the MIB must match the user's needs: not too detailed and also not too coarse-grained.

There Is only One MIB

One merit of a standard MIB is ease of extension. As new technologies are invented and deployed, the associated managed objects must be defined in new MIB modules. The latter can then be added to the standard MIB in an orderly fashion, e.g., by using enterprise-specific numbers. New objects can be defined and included in MIB module files, such as the MPLS MIB files we will see in [Chapter 8](#). The objects in such files are implemented in the NEs that support the associated technology (e.g., MPLS). The important point to note is that these are extensions to the standard MIB, i.e., there is only one MIB.

Analogy for an NMS

It may be helpful to draw some comparisons between a standard operating system (such as UNIX or Windows 2000) and an NMS. Both provide a set of abstractions to assist in the end use and management of the system. In the case of operating systems, some of the abstract objects are:

- Files

- Applications
- Processes
- Devices, such as hard disks and network interfaces
- Soft objects, such as print jobs and semaphores

These abstract entities map onto real objects that users and applications employ for getting work done. NMS also employ the above objects in addition to other objects specific to network management. These NMS abstract objects are:

- MIB modules
- Applications—agents and managers
- Devices—remote NEs
- Soft objects—connections, paths, interfaces, and so on

These objects are used for managing networks. The NMS employs these objects and provides additional abstractions (GUI, software wizards, etc.) to assist the network operator.

[\[Team LiB \]](#)

 PREVIOUS  NEXT 

Network Elements

The previous sections have introduced networks with a brief overview of some of their components and management infrastructure. This section examines NEs a little more closely. [Figure 2-6](#) illustrates (in no particular order) some of the typical software components that combine to make up an NE.

Figure 2-6. Typical NE software components.

Application Software		
Boot Configuration		
Software Image Download		
Telnet	SNMP Agent	Security
Operating System		
Network Protocols	Proprietary Protocols	

An example of an NE is an intelligent line card, which is hosted inside another system, such as a PABX, ATM/MPLS switch, or IP router. An intelligent line card is essentially a computer inside another computer and may contain millions of lines of source code hosted on an embedded real-time operating system, such as pSOS or VxWorks. Some characteristics of intelligent line cards include the following:

- They can extend the lifespan of the host by adding advanced functions such as SNMP and VoIP for a PABX.
- They can take a long time to develop.
- Operators like to extract the maximum performance from them—for example, port bandwidth.
- They increasingly incorporate numerous layer 1, 2, and 3 protocols.

An NMS interacts with the SNMP agent in [Figure 2-6](#), getting and setting MIB object instances and also receiving notifications. Clearly, the SNMP agent in the NE competes for compute and I/O resources with all the other onboard software entities. During times of high device loading, the SNMP agent may become starved of resources. This is a bad thing because the management facility can become essentially disabled. High loading can occur when:

- Many voice calls are in transit through a PABX.
- Large numbers of ATM virtual circuits are transporting many ATM cells.
- Large numbers of IP packets are in transit across a router.
- Network topology changes result in routing protocol convergence.

Compute resource depletion is one type of NE congestion. It can sometimes be cured by some combination of changing process priority, adding extra memory, or adding extra processing power. A more subtle problem is one in which the number of managed objects becomes

so great that the NMS finds it hard to keep up with changes. This is the general area of scalability and is discussed in [Chapter 3](#), "The Network Management Problem."

[[Team LiB](#)]

[[PREVIOUS](#)] [[NEXT](#)]

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Introducing MPLS: First Chunk

We use MPLS as a running example throughout the book. Deployment of MPLS in the enterprise core is likely to occur only in extremely large organizations. Typically, such organizations have global reach and may already use a lot of ATM. Service providers are already deploying MPLS. Enterprises may initially deploy MPLS on WAN backbones and later on, they may move MPLS into the network core. In fact, the expansion of MPLS to the enterprise premises (via the MPLS UNI) is part of the work of the MPLS Forum.

On a more general note, a good understanding of MPLS is important for appreciating issues such as traffic engineering, network-QoS, and connection-oriented IP networks.

This section introduces the first chunk of MPLS [[DavieRehker2000](#)] technology presented in this book. MPLS is essentially quite simple, but the building blocks are a little difficult to learn because they span both layers 2 and 3 and require some understanding of signaling and IP routing protocols. For this reason, the discussion is split into easy-to-understand, bite-sized chunks,^[1] starting here and finishing up with the case study on some aspects of MPLS network management in [chapters 8 and 9](#).

[1] This is in keeping with our concepts of learning about network management and creating NMS solutions. These ideas are introduced in later chapters, but for now we say that a given technology must be understood before an attempt is made to manage it. This is obvious enough, but the depth of understanding is the key. In many cases, a good overview is all that is needed to get started on producing an NMS solution. This is the model we use with MPLS.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

The Trend Towards IP

As mentioned earlier, a major trend in networking is the migration towards IP. Enterprise (and SP) networks are generally expected to support native IP cores in the next five to ten years. This is often referred to as the next-generation network. There are many reasons for this migration:

- IP has become the *lingua franca* of networking—other protocols, such as Novell's IPX, will continue to exist, but the global protocol will almost certainly be IP version 4, possibly moving to IP version 6.
- End-user devices, such as mobile phones, PDAs, and TV set-top boxes, have become IP-capable, so end-to-end IP (from the user to the core) will become more important.
- Existing layer 2 devices do not easily support massive (scalable) deployment of layer 3 protocols such as is increasingly needed for services like IP VPNs.
- The need for specialized layer 2 maintenance skills is reduced.
- A single layer 3 control plane is easier to manage.
- Aggregation of IP traffic becomes possible, improving scalability.
- Different (guaranteed) levels of network service can be sold to customers.
- Management system object models can become more generic.

Users will continue to access enterprise networks using a variety of technologies, such as ATM, FR, xDSL, ISDN, and POTS (dial-up), but their layer 2-encapsulated IP traffic will increasingly be extracted and repackaged as pure IP/MPLS at the network edge. In effect, layer 2 traffic is being pushed out of the core and into the access edge of the network. This is another reason for end-to-end IP: The need for terminating multiple layer 2 technologies begins to disappear. MPLS is a good starting point for this migration because:

- MPLS allows traffic engineering (putting the traffic where the bandwidth is).
- MPLS integrates IP QoS with layer 2 QoS.
- Many vendors are providing MPLS capability in their devices—for example, Cisco, Juniper, Nortel Networks, and Marconi.

Many of the issues relating to traffic engineering, QoS, and handling legacy layer 2 services are highly relevant to enterprises and SP networks. Enterprise networks feature an increasingly rich mixture of traffic types: email, Web, audio/video, VoIP, and so on. Such a range of traffic types may well necessitate techniques such as traffic engineering and bandwidth management rather than just adding more capacity (i.e., overengineering the core).

[Team LiB]

◀ PREVIOUS NEXT ▶

[\[Team LiB \]](#)

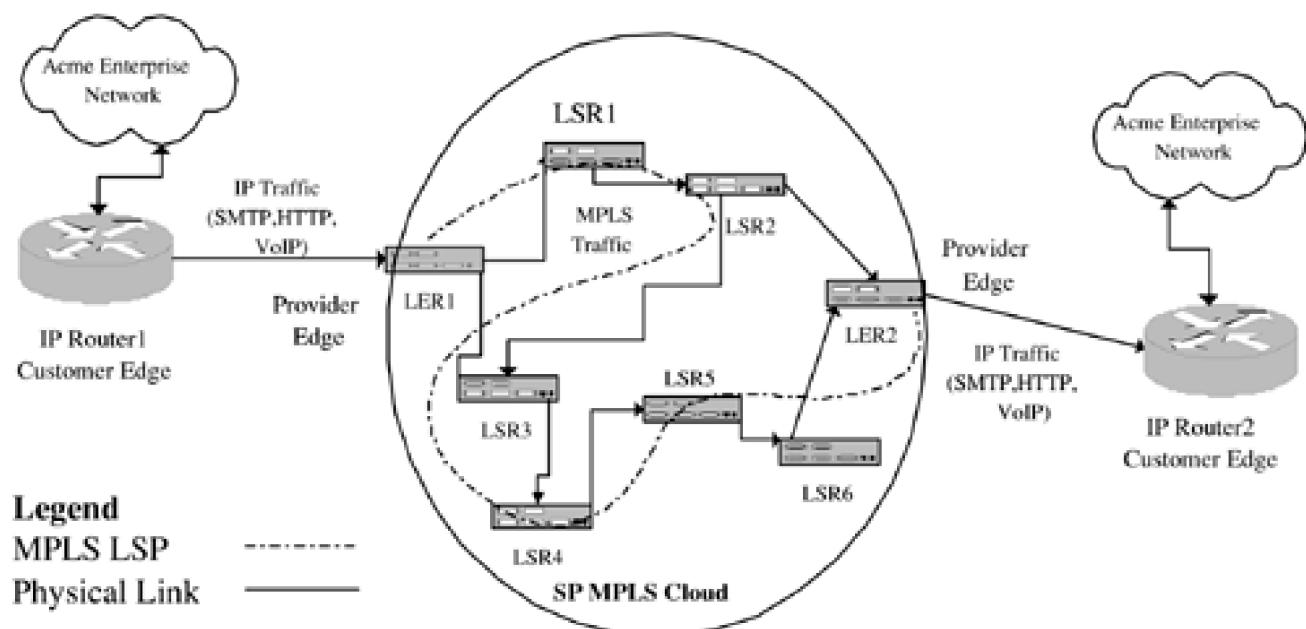
[!\[\]\(00a889b91bf7d5fb42b190f53401604e_img.jpg\) PREVIOUS](#) [!\[\]\(742cd384b579c258cad4334b5a7942b1_img.jpg\) NEXT !\[\]\(fe271906d6395c5f704fb35fd6753862_img.jpg\)](#)

MPLS Concepts

MPLS is a forwarding technology. Its purpose is to receive an incoming traffic type (layer 2 or 3) at the network edge, encapsulate it, and then transmit it through an MPLS core (or cloud). At the exit from the cloud, another edge device removes the MPLS header and forwards the traffic towards its destination.

An example is illustrated in [Figure 2-7](#), where the incoming IP traffic from the Acme enterprise network consists of a mixture of SMTP (email), HTTP (Web), and VoIP. This traffic is routed from IP Router 1 on the customer premises. The traffic then lands at the provider edge (LER1), where an IP header lookup is carried out on each packet prior to pushing an MPLS-encapsulated packet into the LSP (label switched path; this process is described in a little more detail below). The MPLS cloud in [Figure 2-7](#) consists of many routers; a big network might have hundreds (or even thousands) of such routers distributed over a wide geographic area. The MPLS cloud routers in a real network would have many more edge connections than just the two IP routers shown. In other words, the SP network could have many thousands of such devices connected to it.

Figure 2-7. An MPLS network joining enterprise branches.



MPLS nodes are either edge or core devices. Edge routers are called label edge routers (LERs) and core routers are called label switching routers (LSRs). Edge routers (such as LER1 in [Figure 2-7](#)) sit at the boundary (or provider edge) of the network, facing the IP traffic stream on one side and the interior of the MPLS cloud on the other. Core routers, such as LSR1 in [Figure 2-7](#), sit inside the MPLS cloud. Ingress LERs encapsulate IP traffic as MPLS packets and push these onto LSPs in the core of the MPLS cloud. We define LSPs more fully in the next section—for the moment, just think of them as layer 3 virtual connections or pipes that carry traffic from edge to edge through the network.

In [Figure 2-7](#), IP Router 1 presents an IP traffic stream (SMTP, HTTP, VoIP) at an ingress interface of LER1. LER1 performs normal lookups on the IP headers of the incoming packets. From the destination IP address (or some other part of the IP header), LER1 can decide how best to forward the packets, and it has a number of choices. Taking the first IP packet that arrives, LER1 can:

- Forward the packet unlabeled; the packet is then routed to the next hop. In this mode, the MPLS nodes act as pure IP routers.
- Drop the packet.
- Encapsulate the packet with an MPLS label and push it onto an LSP.

In [Figure 2-7](#), LER1 decides to take the last option in the above list, and the MPLS packet is transported via an LSP. The MPLS traffic is then pushed onto the LSP comprised of the ingress interfaces on the following nodes: LSR1-LSR2-LSR3-LSR4-LSR5-LSR6-LER2. This

path is shown as a dashed line in [Figure 2-7](#). An LSP has the following characteristics:

- The LSP is created manually or via a signaling protocol.
- The path taken by the LSP may be either user-specified or computed by LER1.
- The LSP may have reserved resources, such as bandwidth, along the path.
- There is a link between LER1 and LSR3, but the incoming traffic does not take this route. Instead, traffic at LER1 is pushed onto the LSP and follows the route LER1-LSR1-LSR2-LSR3. This route overrides any default shortest path IP routing (between LER1 and LER2), giving the operator a greater degree of control in the paths taken by traffic (i.e., traffic engineering). In this sense, LSPs make the network connection-oriented, just like a telephone network.
- IP traffic from IP Router 1 landing on LER1 is MPLS-encapsulated and forwarded across the LSP all the way to LER2. LER2 removes the MPLS encapsulation, carries out an IP lookup, and forwards the IP packet to IP Router 2. In other words, the traffic on the provider edge links is IP.
- Only two IP lookups are required in getting from IP Router 1 through the MPLS cloud to IP Router 2. While line-rate IP lookups are now available in routers, MPLS provides this as well as the ability to create traffic-engineered connections (i.e., LSPs) that may or may not reserve bandwidth.
- Once the IP traffic is MPLS-encapsulated, all subsequent routing is done using a label rather than any IP packet header-based addressing (the label structure is described in [Figure 4-10](#) in [Chapter 4](#)).
- As well as traffic engineering, MPLS provides a QoS function. This means that the LSP allocates network resources that enable it to ensure the traffic experiences a specified service level. We will see QoS in later chapters.

Some MPLS nodes can simultaneously function as ATM switches and MPLS nodes. ATM-based MPLS nodes have an important feature called ships-in-the-night (SIN). This allows both ATM and MPLS protocols to operate independently of one another on the same port (that is, MPLS is configured on the port, creating an MPLS interface). Not all MPLS nodes can simultaneously act as MPLS and ATM switches; for example, Juniper routers and Cisco 7000/12000 routers cannot. Some models of switches from Nortel, Lucent, and Marconi can. The provision of SIN is an effort to facilitate a gradual migration of networks from ATM to MPLS. Service providers can continue to deploy revenue producing, legacy services based on ATM while slowly introducing MPLS-based services (such as RFC 2547 VPNs). So, the nodes in [Figure 2-7](#) can also create ATM virtual circuits alongside MPLS LSPs. These ATM circuits can then natively transport ATM cells. SIN conceptually splits a switch into a combination of an ATM and an MPLS device, like two mini-switches. It can result in a fragmented label space, and also there may be an effect on performance if a great deal of unlabeled IP traffic is in transit across the switch.

The MPLS nodes can run traffic engineering-enabled routing protocols such as Open Shortest Path First (OSPF) and Intermediate System-Intermediate System (IS-IS). This allows the exchange of traffic engineering data, such as available (and used) link bandwidth.

Definition of an LSP

As we've seen, an LSP is an automatically or manually configured (optionally traffic-engineered with optional QoS) path through an MPLS network. An LSP originates on an LER, passes through zero or more LSRs, and terminates on another LER. The path taken by the LSP can be set by the operator or computed by the LER. Network resources, such as bandwidth, can be reserved along the path, or the LSP can offer a best-effort service.

With reference to the MIBs examined in [Chapter 8](#), an LSP is comprised of the following components on the originating LER:

- A tunnel
- A cross-connect
- An out-segment

Each LSR in the core then supports the LSP by providing the following components:

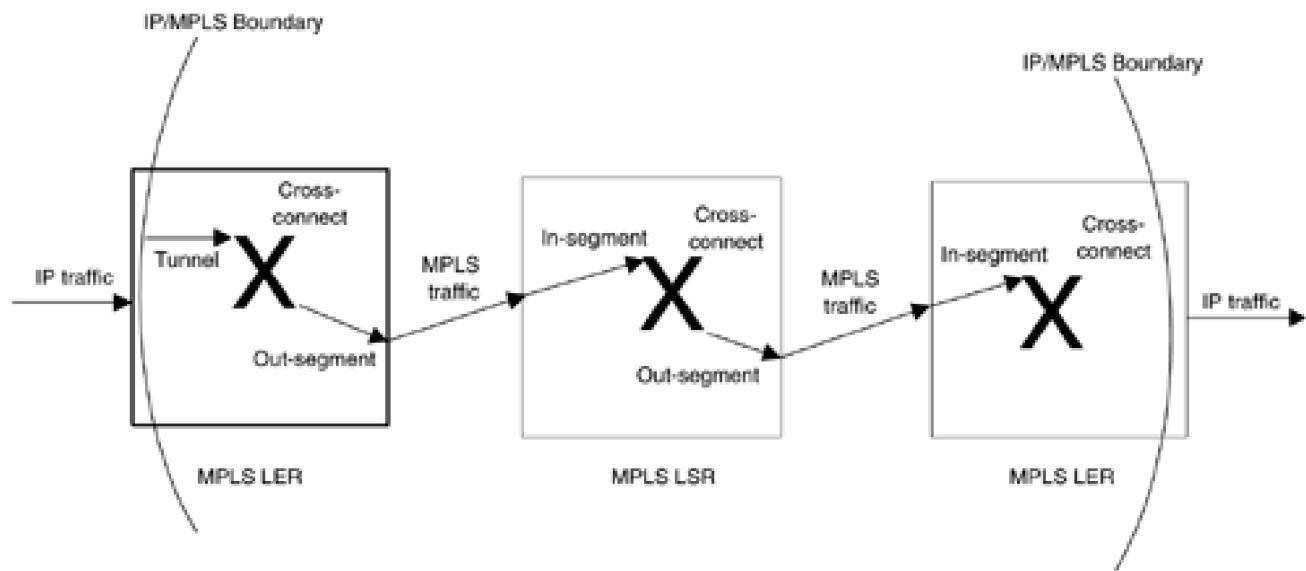
- An in-segment
- A cross-connect
- An out-segment

Finally, the terminating LER provides the endpoint for the LSP using the following components:

- An in-segment
- A cross-connect

So, a notional network made up of two LERs and one LSR with an LSP spanning all three nodes might look like [Figure 2-8](#).

Figure 2-8. LSP components.



[Figure 2-8](#) illustrates the above components as well as another important concept: the IP/MPLS boundary. This boundary is the edge of the MPLS cloud; that is, on the IP side of the boundary there is only IP—all MPLS labels have been stripped off. On the MPLS side of the boundary there can be both IP and MPLS. In other words, there is no reason why pure IP traffic cannot traverse an MPLS core. Why might this arise? One possibility is that a packet arrives with a destination IP address that is not served by any existing LSP. In this case, the packet can be routed hop by hop through the MPLS core.

Packets arriving on the edge of the IP/MPLS boundary are subjected to a normal IP lookup, but if their destination address is served by an LSP, then they are MPLS-encapsulated and pushed into the appropriate tunnel (as illustrated in [Figure 2-8](#)). The encapsulated packets then pass through the segments and cross-connects of the LSP path, consuming any of the resources reserved on that path. One important part of LSP setup is the programming of the hardware. One of the major drivers of MPLS is that the MPLS management plane hides the platform-specific details. In this way, the platform-specific protocols impose the required labels, traffic engineering requirements, and QoS settings on the relevant interfaces across the network. The network operator employs the management interface to initiate this process. SNMP, in this context, is just another management interface, as is a CLI.

LSPs are set up from edge to edge traversing the LSRs in the core. LSPs serve as a conduit for transporting MPLS traffic from the point of ingress at the edge to the point of egress on another edge. We will see the structure illustrated in [Chapter 8](#). One final important point about MPLS is that it is not restricted to carrying just IP traffic. The traffic landing at LER1 can also be Ethernet, ATM, and TDM. Carrying legacy traffic is an important part of MPLS.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Summary

SNMPv3 provides several compelling advantages over previous versions. Authentication and strong security; improved error, exception, and notification reporting; and bulk operations are among the most important. SNMPv3 features like these can help make life a lot easier for network managers. Setting up security on a networkwide basis is still a difficult task. The different versions of SNMP require support in a given NMS. This consists of community strings and trap/notification destination settings. The latter is also required in SNMPv3. For SNMPv3, community strings are not used. Instead, the user must set up the required level of authentication and encryption.

MIB browsers represent an indispensable tool for both NMS software developers and network managers. They provide a detailed view of MIB objects. While MIBs can be defined with arbitrary complexity and size, they all have a relatively simple structure. Understanding this structure is a key element of studying network management technology. An essential point to note about managed objects is that there is just a single MIB. The MIB can be extended to support new objects, such as, those for MPLS.

An NMS can also be arbitrarily complex with a great many components. However, a typical NMS is at least conceptually similar to other more familiar applications, such as operating systems. Just as for MIBs, it is important to gain a good conceptual understanding of the NMS structure.

NEs are those components that combine together to make up a managed network. There are some broad similarities between many NEs, and an appreciation of these helps in deciding how to manage them.

MPLS is an extremely important technology that is being widely deployed in SP networks and also in some enterprise WANs. The major components of this technology were described in order to lay the foundations for [Chapter 8](#), where we examine the mechanisms for creating MPLS-managed objects.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Chapter 3. The Network Management Problem

Having looked at some of the nuts and bolts of network management technology, we now consider some of the problems of managing large networks. In many respects the large enterprise networks of today are reminiscent of the islands of automation that were common in manufacturing during the 1980s and 1990s. The challenge facing manufacturers was in linking together the islands of microprocessor-based controllers, PCs, minicomputers, and other components to allow end-to-end actions such as aggregated order entries leading to automated production runs. The hope was that the islands of automation could be joined so that the previously isolated intelligence could be leveraged to manufacture better products. Similar problems beset network operators at the beginning of the 21st century as traffic types and volumes continue to grow. In parallel with this, the range of deployed NMS is also growing. Multiple NMS adds to operational expense.

There is a strong need to reduce the cost of ownership and improve the return on investment (ROI) for network equipment. This is true not just during periods of economic downturn, but has become the norm as SLAs are applied to both enterprise and SP networks. NMS technology provides the network operator with some increasingly useful capabilities. One of these is a move away from tedious, error-prone, manually intensive operations to software-assisted, automated end-to-end operations.

Network operators must be able to execute automated end-to-end management operations on their networks [[Telcordia](#)]. An example of this is VLAN management in which an NMS GUI provides a visual picture—such as a cloud—of VLAN members (ports, MAC addresses, VLAN IDs). The NMS can also provide the ability to easily add, delete, and modify VLAN members as well as indicate any faults (e.g., link failures, warm starts) as and when they occur. Another example is enterprise WAN management in which ATM or FR virtual circuits are used to carry the traffic from branch offices into central sites. In this case, the enterprise network manager wants to be able to easily create, delete, modify, and view any faults on the virtual circuits (and the underlying nodes, links, and interfaces) to the remote sites. Other examples include storage (including SANs) management and video/audio conferencing equipment management. As we saw in [Chapter 1](#), "Large Enterprise Networks," the range of enterprise network services is growing all the time and so also is the associated management overhead.

The benefit of this type of end-to-end capability is a large reduction in the cost of managing enterprise networks by SLA fulfillment, less need for arcane NE know-how, smooth enterprise business processes, and happy end users. Open, vendor-independent NMS are needed for this, and later we look at ways in which software layering helps in designing and building such systems. Simple ideas such as always using default MIB values (seen in [Chapter 1](#)), pragmatic database design (matching default database and MIB values) and technology-sensitive menus also play an important part in providing NMS vendor-independence. The issue of presenting menu options appropriate to a given selected NE provides abstraction; for example, if the user wants to add a given NE interface to an IEEE 802.1Q VLAN, then (in order for the operation to be meaningful) that device must support this frame-tagging technology. The NMS should be able to figure this out and present the option *only* if the underlying hardware supports it. By presenting only appropriate options (rather than all possible options), the NMS reduces the amount of data the user must sift through to actually execute network management actions.

Automated, flow-through actions are required for as many network management operations as possible, including the following FCAPS areas:

- Provisioning
- Detecting faults
- Checking (and verifying) performance
- Billing/accounting
- Initiating repairs or network upgrades
- Maintaining the network inventory

Provisioning is a general term that relates to configuring network-resident objects, such as VLANs, VPNs, and virtual connections. It

resolves down to the act of modifying agent MIB object instances, that is, SNMP **setRequests**. Provisioning usually involves both **sets** and **gets**. Later in this chapter we see this when we want to add a new entry to the MPLS tunnel table. We must read the instance value of the object **mplsTunnelIndexNext** before sending a **setRequest** to actually create the tunnel. Many NMS do not permit provisioning for a variety of reasons:

- Provisioning code is hard to implement because of the issue of timeouts (i.e., when many **set** messages are sent, one or more may time out).
- NE security settings are required to prevent unauthorized actions.
- There is a lack of support for transactions that span multiple SNMP sets (i.e., SNMP does not provide rollback, a mechanism for use when failure occurs in one of a related sequence of SNMP sets. The burden of providing lengthy transactions and/or rollback is on the NMS).
- Provisioning actions can alter network dynamics (i.e., pushing a lot of sets into the network adds traffic and may also affect the performance of the local agents).

If the NMS does not allow provisioning, then some other means must be found; usually, this is the EMS/CLI. SNMPv3 provides adequate security for NMS provisioning operations.

Fault detection is a crucial element of network management. NMS fault detection is most effective when it provides an end-to-end view; for example, if a VLAN link to the backbone network is broken (as in VLAN 2 in [Chapter 1, Figure 1-4](#)), then that VLAN GUI element (e.g., a network cloud) should change color instantly. The NMS user should then be able to drill down via the GUI to determine the exact nature of the problem. The NMS should give an indication of the problem as well as a possible resolution (as we've seen, this is often called root-cause analysis). The NMS should also cater to the case where the user is not looking at the NMS topology and should provide some other means of announcing the problem, for instance, by email, mobile phone short text message, or pager.

Performance management is increasingly important to enterprises that use service level agreements (SLAs). These are contractual specifications between IT and the enterprise users for service uptime, downtime, bandwidth/system/network availability, and so on.

Billing is important for those services that directly cost the enterprise money, such as the PSTN. It is important for appropriate billing to be generated for such services. Billing may even be applied to incoming calls because they consume enterprise network resources. Other elements of billing include departmental charges for remote logins to the network (external SP connections may be needed, for example, for remote-access VPN service) and other uses of the network, such as conference bridges. An important element of billing is verifying that network resources, such as congested PSTN/WAN trunks, are dimensioned correctly. In [Chapter 1](#), we mentioned that branch offices are sometimes charged a flat rate for centralized corporate services (e.g., voice, LAN/WAN support). This is accounting rather than billing. In billing, money tends to be paid to some external organization, whereas in accounting, money may be merely transferred from one part of an organization to another. Many service providers offer services that are billed using a flat-rate model—for example, x dollars per month for an ATM link with bandwidth of y Mbps. Usage-based billing is increasingly attractive to customers because it allows for a pay-for-use or pay-as-you-grow model. It is likely that usage-based billing/accounting will increasingly be needed in enterprise NMS applications. This is particularly true as SLAs are adopted in enterprises.

Networks are dynamic entities, and repairs and upgrades are a constant concern for most enterprises. Any NE can become faulty, and switch/router interfaces can become congested. Repairs and upgrades need to be carried out and recorded, and the NMS is an effective means of achieving this.

All of the FCAPS applications combine to preserve and maintain the network inventory. An important aspect of any NMS is that the FCAPS applications are often inextricably interwoven; for example, a fault may be due to a specific link becoming congested, and this in turn may affect the performance of part of the network. We look at the important area of mediation in [Chapter 6](#), "Network Management Software Components."

It is usually difficult to efficiently create NMS FCAPS applications without a base of high-quality EMS facilities. This base takes the form of a well-implemented SNMP agent software with the standard MIB and (if necessary) well-designed private MIB extensions. Private MIB extensions are needed for cases where vendors have added additional features that differentiate their NEs from the competition.

All these sophisticated NMS features come at a price: NMS software is expensive and is often priced on a per-node basis, increasing the network cost base. Clearly, the bigger the network, the bigger the NMS price tag (however, the ratio of cost/bit may go down).

This chapter focuses on the following major issues and their proposed solutions:

- Bringing the managed data to the code

- Scalability
- The shortage of development skills for creating management systems
- The shortage of operational skills for running networks

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

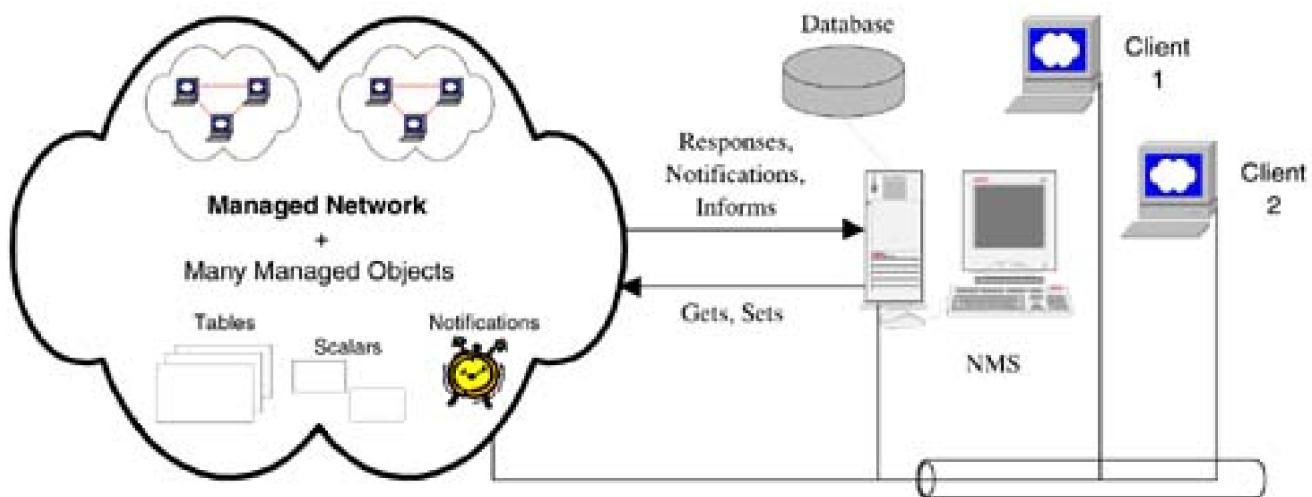
Bringing the Managed Data to the Code

Bringing data and code together is a fundamental computing concept. It is central to the area of network management, and current trends in NE development bring it to center stage. Loading a locally hosted text file into an editor like Microsoft Notepad is a simple example: The editor is the code and the text file is the data. In this case, the code and data reside on the same machine, and bringing them together is a trivial task. Getting SNMP agent data to the manager code is not a trivial task in the distributed data model of network management because:

- Managed objects reside on many SNMP agent hosts.
- Copies of managed objects reside on SNMP management systems.
- Changes in agent data may have to be regularly reconciled with the management system copy.

Agent-hosted managed objects change in tandem with the dynamics of the host machine and the underlying network—for example, the `ipInReceives` object from [Chapter 1](#), which changes value every time an IP packet is received. This and many other managed objects change value constantly, providing a means for modeling the underlying system and its place in the network. The same is true of all managed NEs. MIBs provide a foundation for the management data model. The management system must keep track of relevant object value changes and apply new changes as and when they are required. As mentioned in [Chapter 1](#), the management system keeps track of the NEs by a combination of polling, issuing `set` messages, and listening for notifications. This is a classic problem of storing the same data in two different places and is illustrated in [Figure 3-1](#), where a management system tracks the objects in a managed network using the SNMP messages we saw in [Chapter 2](#), "SNMPv3 and Network Management."

Figure 3-1. Components of an NMS.



[Figure 3-1](#) illustrates a managed network, a central NMS server, a relational database, and several client users. The clients access the FCAPS services exported by the NMS, for example, viewing faults, provisioning, and security configuration. The NMS strives to keep up with changes in the NEs and to reflect these in the clients.

Even though SNMP agents form a major part of the management system infrastructure, they are physically remote from the management system. Agent data is created and maintained in a computational execution space removed from that of the management system. For example, the `ipInReceives` object is mapped into the tables maintained by the host TCP/IP protocol suite, and from there it gets its value.^[1] Therefore, `get` or `set` messages sent from a manager to an agent result in computation on the agent host. The manager merely collects the results of the agent response. The manager-agent interaction can be seen as a loose type of message-based remote procedure call

(RPC). The merit of not using a true RPC mechanism is the lack of associated overhead.

[1] This is true for hosts; for routers, the ipInReceives object is part of the interface statistics and not part of the IP stack.

This is at once the strength and the weakness of SNMP. The important point is that the problem of getting the agent data to the manager is always present, particularly as networks grow in size and complexity. (This problem is not restricted to SNMP. Web site authors have a similar problem when they want to embed Java or JavaScript in their pages. The Java code must be downloaded along with the HTML in an effort to marry the browser with the Web site code and data. Interestingly, in network management the process is reversed: The data is brought to the code.) So, should the management system simply request all of the agent data? This is possibly acceptable on small networks but not on heavily loaded, mission-critical enterprise and SP networks. For this reason, the management system struggles to maintain an accurate picture of the ever-changing network. This is a key network management concept.

If an ATM network operator prefers not to use signaled virtual circuits, then an extra monitoring burden is placed on the NMS. This is so because unsignaled connections do not recover from intermediate link or node failures. Such failures give rise to a race between the operator fixing the problem and the user noticing a service loss. These considerations lead us to an important principle concerning NMS technology: **The quality of an NMS is inversely proportional to the gap between its picture of the network and the actual state of the underlying network—the smaller the gap, the better the NMS.** An ideal NMS responds to network changes instantaneously. Real systems will always experience delays in updating themselves, and it is the goal of the designers and developers to minimize them.

As managed NEs become more complex, an extra burden is placed on the management system. The scale of this burden is explored in the next section.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

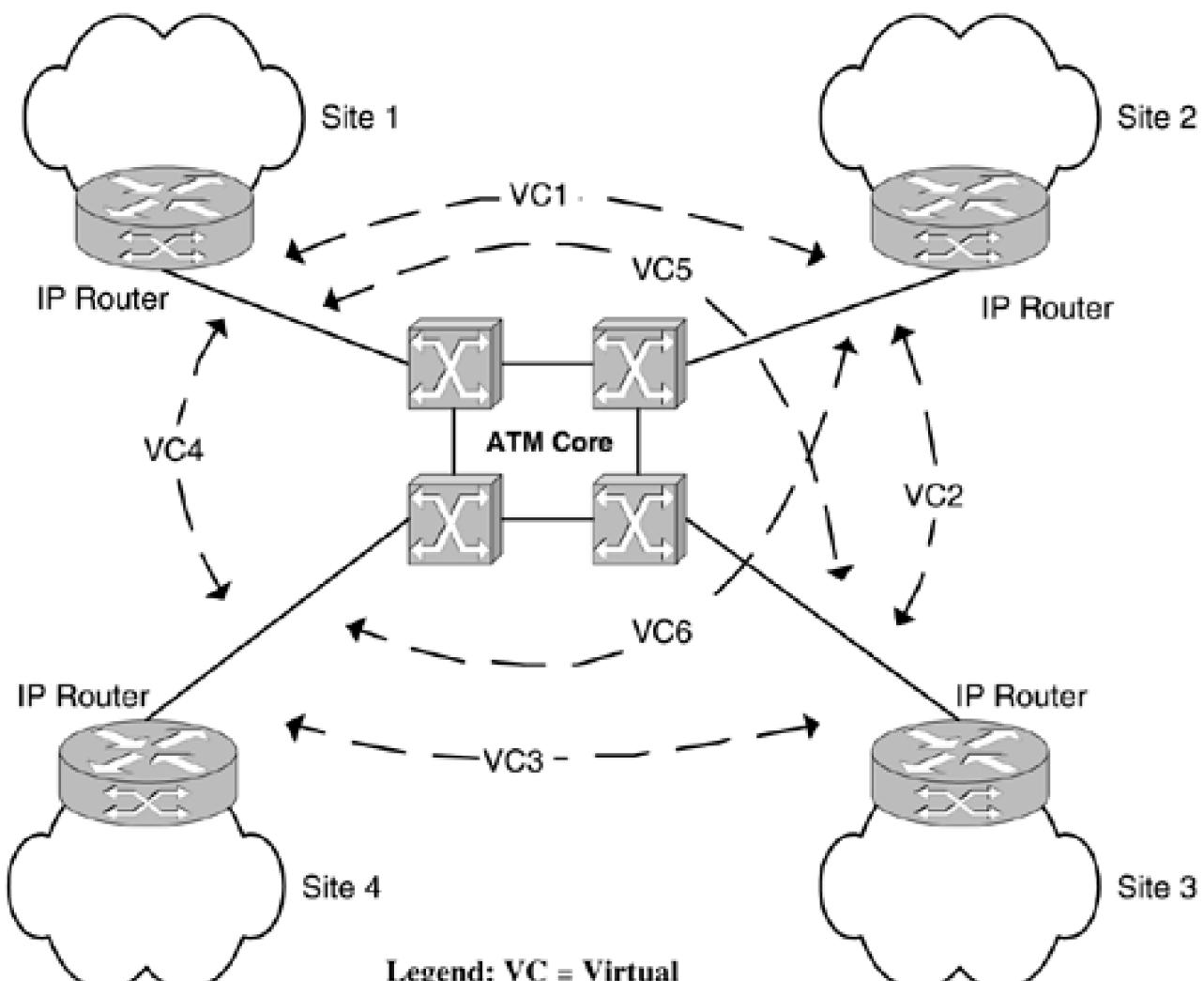
Scalability: Today's Network Is Tomorrow's NE

Scalability is one of the biggest problems facing modern networking (security is another). It affects devices [RouterScale2002], interfaces, links, internal soft objects (such as virtual circuits), and management systems. A scalability problem occurs when an increase in the number of instances of a given managed object in the network necessitates a compensating, proportional resource increase inside the management system.

Layer 2 VPN Scalability

Scalability problems tend to arise in situations of proportional growth. [Figure 3-2](#) illustrates a layer 2 VPN, which provides a good example of scalability. This scheme is often referred to as an overlay network because the IP network is overlaid on the underlying ATM infrastructure.

Figure 3-2. Layer 2 VPN scalability: the N^2 problem.



Circuit

Sites 1 to 4 are all part of the one enterprise. This makes [Figure 3-2](#) what is often called an intranet VPN. If one or more of the other sites is part of another organization, such as a customer or supplier, then we have an extranet VPN. Yet another VPN variant is the access VPN, which allows remote users to connect to it over some type of access technology, such as dialup.

In [Figure 3-2](#), four sites are contained in the VPN, with one IP router in each site cloud. In order to achieve full layer 3 connectivity, each site must have a virtual circuit connection to every other site. These connections are created through the ATM core. So, the number of ATM virtual circuits required is six; that is, $N * (N - 1)/2$, where N is the number of sites. The full mesh of six bidirectional virtual circuits is shown in [Figure 3-2](#) as VC1-VC6. A full mesh provides the necessary connectivity for the VPN [PrinRussell](#). This is generally referred to as the N^2 problem because the number of layer 2 virtual circuits required is proportional to the square of the number of sites. Anything in networking that grows at the rate of N^2 tends to give rise to a problem of scale. The reason for calling this the N^2 problem is because as the number of sites gets bigger, the N^2 term is more significant than the other terms.

The problem gets worse if the ATM virtual circuits in the core are unidirectional (some vendors support only unidirectional permanent virtual circuits, or PVCs) in nature because then the number must be doubled in order for IP traffic to flow in both directions. Adding a new site to the VPN requires the creation of new virtual circuits from that site to all other sites. When the number of sites and subscribers is very large, the number of virtual circuits required tends to become unmanageable. Another less obvious problem with this is that each virtual circuit consumes switch capacity in terms of memory and control processor resources. Added to this is link bandwidth and fabric switching capacity if the virtual circuits reserve QoS resources.

As if that wasn't enough, a further problem with layer 2 VPNs is that topology changes in the core can result in routing information exchanges of the order of N^4 [DavieRehker2000](#).

In contrast, layer 3 VPNs provide a much more scalable solution because the number of connections required is proportional to number of sites, not the square of the number of sites. Layer 3 VPNs (such as RFC 2547) avoid the need for a full mesh between all of the customer edge routers by providing features such as:

- A layer 3 core
- Overlapping IP address ranges across the connected sites (if separate organizations use the same VPN service)
- Multiple routing table instances in the provider edge routers

Not surprisingly, layer 3 VPN technology is an area of great interest to both enterprise network managers and service providers. For enterprises, layer 3 VPNs provide advanced, potentially low-cost networking features while allowing the service to be provided and managed by a service provider. For SP networks, layer 3 VPNs provide a scalable solution as well as an opportunity to extend services all the way to the customer premises.

Virtual Circuit Status Monitoring

Another example of scalability concerns the way in which an NMS manages status changes in ATM virtual circuits. If it is required for the NMS to read and interpret a MIB object table row for every ATM circuit, and this is done in an unrestricted fashion (where the NMS attempts to read all table entries at the same time), then there is scope for scalability problems. They arise when the number of MIB table entries becomes very large. In attempting to keep up with large tables on possibly hundreds of agents throughout a big network, the NMS can run low on resources such as processing power, memory, and disk, and can effectively grind to a halt (unrestricted SNMP **gets** also use up network bandwidth). The large number of managed objects gives rise to NMS resource issues—that is, scalability. Really large SNMP tables can possibly also tax the resources of the host agent, resulting in timeouts and lost messages. The discussion here assumes this not to be the case.

Some method has to be devised for reading only those table entries that have changed rather than all entries. Processing large (ever-increasing) tables is not scalable. Agents may be able to assist in this by using some form of compressed data in **GetResponse** and **Trap** messages. This would require:

- A new type of MIB object.

- Compression software facilities in the agents and managers. To a degree, this could be considered to run counter to the philosophy of simplicity associated with SNMP.

A similar problem can occur in reverse when a manager wants to add entries to a large MIB table. In many cases (for example, when creating MPLS LSPs), it is necessary for the SNMP manager to specify an integer index for the new row. The index is a unique (in the sense of a relational database key) column in the table, and the next free value is used for the new LSP. So, if there are 10,000 LSPs already stored sequentially starting at 1, then the next available index is 10,001. The agent knows what the next free value is because it maintains the table. However, the manager may not necessarily know the extent of the table and often has to resort to an expensive MIB walk to determine the next free index. This is because agent data can be changed without the management system knowing about it; for example, using a command-line (or craft) interface, a user can independently add, delete, or modify NE data. This is mapped into the agent MIB, and if these operations do not generate traps, then the management system is oblivious to the data changes (unless it has an automatic discovery capability that reads the affected tables). There is a better way of navigating tables to cater to this and other situations. This brings us to our first MIB note.

[\[Team LiB \]](#)

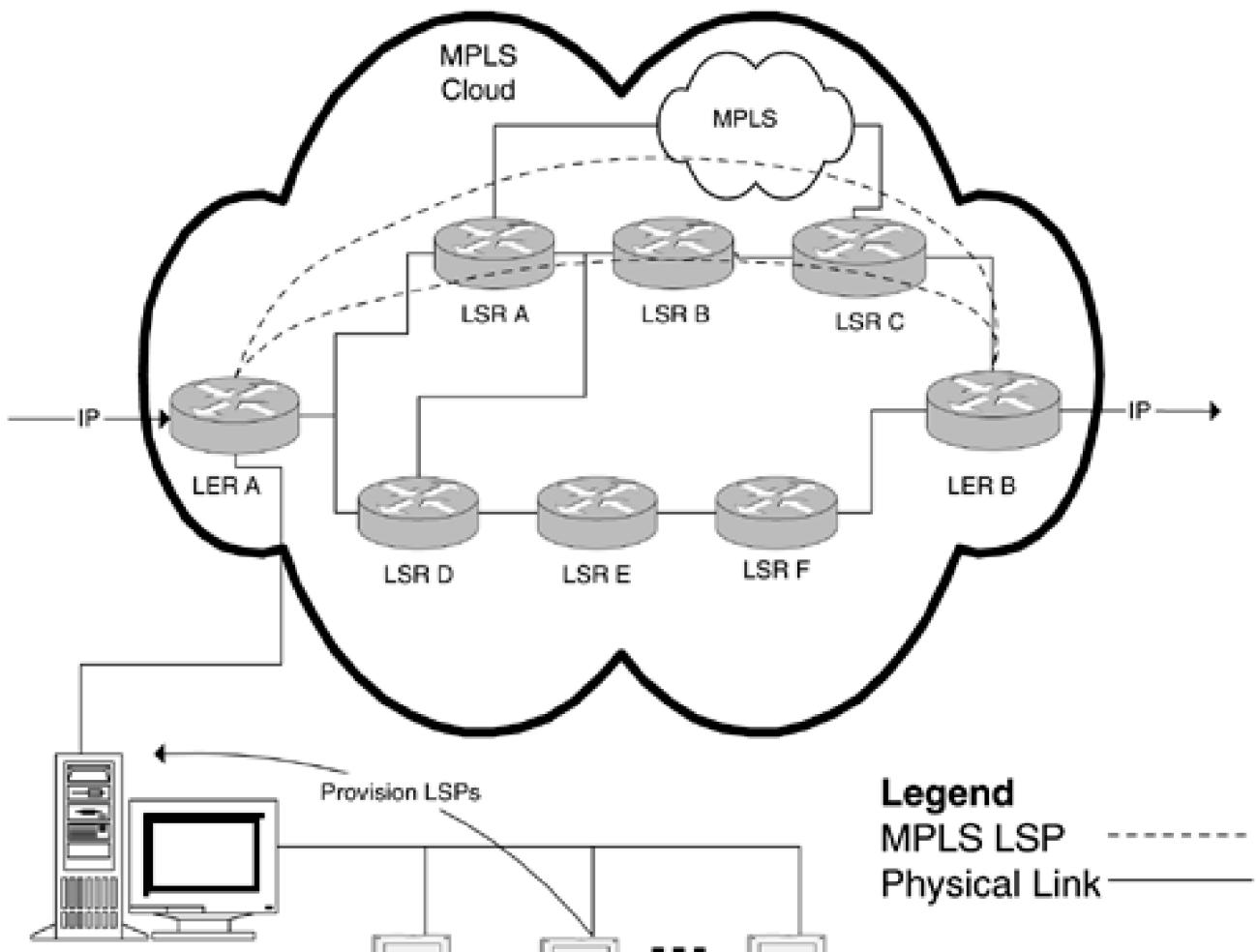
[◀ PREVIOUS](#) [NEXT ▶](#)

MIB Note: Scalability

A very useful object type for large table management (as described above) is a counter conceptually called **nextObjectIndex**. This object provides the index number of the next available slot in a table. The agent maintains the value of this object so that when the manager has to add a new row to the MPLS tunnel table, it need only get the value of the associated **nextObjectIndex**. This avoids the overhead of MIB walks to actually count the entries and figure out the next free value. Once a new entry is added to the table, the agent increments the value of its **nextObjectIndex**. It is encouraging to see this type of MIB object being used in the IETF draft-standard MPLS MIBS, (e.g., [mplsTunnelIndexNext \[IETF-TE-MPLS\]](#)). This takes cognizance of scalability issues in the standard MIB document and avoids the need for proprietary solutions. It also provides a good example for implementers. Scalability issues like this can be difficult (or impossible) to resolve without the support of special MIB objects. This type of scalability issue will become more pressing as networks and the complexity of the constituent managed objects continue to grow.

Network operators and their users increasingly demand more bandwidth, faster networks, and bigger devices. Scalability concerns are growing because routers and switches are routinely expected to support the creation of millions of virtual circuits [\[ATM&IP2001\]](#). Not only can devices support this number of objects, they can also create these circuits at an increasingly fast pace—tens of thousands per second. To illustrate the scale of this, let's assume in [Figure 3-3](#) that there are hundreds of thousands of nodes (we show just a few).

Figure 3-3. Creating LSPs in an MPLS network.





Client 2 now executes a bulk provisioning operation. This results in the NMS server requesting that MPLS router LER A is to create two blocks of 10,000 signaled LSPs originating at A. The first 10,000 LSPs follow the path LER A-LSR A-LSR B-LSR C-LER B, while the second set follows the path LER A-LSR A-Cloud-LSR C-LER B. (The cloud in the latter case could be another network.) Further, let's assume that LER A can create LSPs at a rate of 10,000 per second. This means that once the intermediate node MIBs have been populated and the LSPs become operational, the network will emit a tunnel-up trap for every LSP. So, the management system has to be able to handle 20,000 traps coming in very fast from the network. There could be scope here for aggregating traps in compressed form, as mentioned earlier.

Since the LSPs are now operational, this must be reflected in the management system database and the active client/user interfaces (Clients 1 to n in [Figure 3-3](#)). The clients could be viewing (or provisioning, like Client 2) LSPs in the network, and any required changes to their views should be made as quickly as possible.

The problems don't stop there, because the LSPs must then be managed for further changes, such as:

- Status (e.g., becoming congested or going out of service)
- Faults such as an intermediate node/link failure or receipt of an invalid MPLS label
- Deletion by a user via a CLI (i.e., outside the management system)
- Modification by a user (changing the administrative status from up to down)

The result of any or all of these is some change in the LSP managed object attributes. The NMS picture of the network state is then at variance with the actual picture. All such changes must be reflected in the NMS as quickly as possible. The detailed functions of a typical NMS are discussed in [Chapter 5](#), "A Real NMS."

The above discussion is a little simplistic; that is, it is likely that many of the above LSPs might be aggregated into one LSP with more reserved bandwidth. However, we illustrate the case merely to point out that if the emerging NEs are capable of generating large numbers of virtual circuits quickly, then the NMS must be able to support that in all of the affected FCAPS areas.

A noteworthy point in [Figure 3-3](#) is the direction of the IP service; this is indicated as being from left to right. This reflects the fact that MPLS is a forwarding technology. If it is required to move IP traffic from LER B towards LER A, then LSPs have to be created specifically for this purpose, originating at LER B and terminating at LER A.

Other Enterprise Network Scalability Issues

The discussion in the previous section applies mostly to SP networks. Scalability concerns are also profoundly affecting enterprise networks in the following areas:

- Storage solutions, such as adding, deleting, modifying, and monitoring SANs
- Administration of firewalls, such as rules for permitting or blocking packet transit
- Routers, such as access control lists and static routes
- Security management, such as encryption keys, biometrics facilities, and password control
- Application management

SANs are becoming a vital storage service. Storage needs are steadily increasing as the number and complexity of applications in use grows. The administration burden associated with firewalls, routers, security, and applications deployment is growing all the time as user populations expand and work practices become more automated.

[Team LiB]

[PREVIOUS] [NEXT]

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Light Reading Trials

Internet core routers from Cisco, Juniper, Charlotte's Networks, and Foundry Networks were stress-tested during 2001 [[LightReading](#)]. A range of tests were carried out, including:

- MPLS throughput
- Latency
- IP throughput at OC-48
- IP throughput at OC-192

During the MPLS throughput testing, the Juniper router supported the creation and use of up to 10,000 LSPs, while the Cisco router topped out at 5,000. Service providers expect to be able to build networks using devices capable of creating millions of LSPs. Clearly, at the time these tests were run, the equipment vendors involved had a long way to go.

In late November 2002, Light Reading executed trials against multiservice switches. One of the tests concerned a sequential (i.e., one after the other) LSP (specifically, tunnel) setup in which the vendor switch (from Alcatel) acted as an LSR and later as an egress LER. The switch was subjected to a steady stream of LSP-creation request messages using RSVP-TE. The test stopped as soon as the switch rejected a setup message. The test imposed 8,000 simultaneous RSVP-TE LSPs for core switches (i.e., LSRs) and 15,000 simultaneous RSVP-TE tunnels for edge switches (i.e., LERs). The Alcatel switch passed the test.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Large NEs

The trend is towards the deployment of much bigger devices, and as with any engineering proposition, this has advantages and disadvantages. The advantages of bigger (denser) routers and switches are:

- They reduce the number of devices required, saving central office (CO) space and reducing cooling and power requirements.
- They may help to reduce cabling by aggregating links.
- They offer a richer feature set.

Reduced inventory is important because less maintenance is needed, and a richer feature set provides for greater control and functional diversity. (Minimizing inventory also reduces operational and capital expenditure and helps retain cash. It also reduces the risk of holding obsolescent stock.) The disadvantages of such devices are:

- They are harder to manage.
- They potentially generate vast amounts of management data.
- They are a possible single point of failure if not backed up.

Compressing so much functionality into devices makes them harder to manage. Correlating faults with services and their users becomes problematic because of the sheer weight of connections. Also, the management system must support more interaction in all of the FCAPS functional areas. This adds up to more I/O and computation.

A related problem is that of SNMP agents timing out during periods of heavy traffic. The SNMP software process on a given NE may have been given a lower priority than the other service implementation modules, with the result that management operations may not be processed quickly enough. In other words, heavily loaded devices may temporarily starve the onboard management software. It is precisely at the time of heavy loading that the management function is most needed in order to control the network, and this may not be possible because of:

- Process priority clashes
- SNMP message queue sizes that are too small
- Excessive I/O interrupts

Management system availability is increasingly important on large networks, and it is also more at risk precisely because of the growing dimensions of the network. Management system capabilities have to be extended beyond what is currently available. The skills required to create these systems seem to be in short supply in the industry. This shortage of skills is examined in the next section.

[Team LiB]

◀ PREVIOUS NEXT ▶

Expensive (and Scarce) Development Skill Sets

Building management systems for the devices of today and tomorrow is increasingly difficult. (The same is true for device development where new technologies such as MPLS and Gigabit Ethernet are being added to new and legacy layer 2 NEs.) Some vendor organizations have completely separate groups devoted to NE and management system development. This introduces a need for clear communication between the groups. Aside from this, the skill set required of NMS software developers is growing and includes, in some cases, what have traditionally been separate disciplines:

- Object-oriented development and modeling using Unified Modeling Language (UML) for capturing requirements, defining actors (system users) and use cases (the principal transactions and features), and mapping them into software classes
- Java/C++
- GUI, often packaged as part of a browser and providing access to network diagrams, provisioning facilities, faults, accounting, and so on
- Server software for long-running, mult-client FCAPS processes
- Specific support for mature/developing features, such as ATM/MPLS
- CORBA for multiple programming languages and remote object support across heterogeneous environments
- Database design/upgrade—matching MIB to database schema across numerous NMS/NE software releases^[2]

^[2] A schema is the set of data definitions for a given repository; for example, table 1 contains an integer key field, table 2 contains text fields, and so on. In other words, the schema describes the database structure similarly to the way a MIB describes management data.

- Deployment and installation issues—performance is always an important deployment issue, as is ease of installation
- IP routing
- MPLS
- Layer 2 technologies such as ATM, FR, and Gigabit Ethernet
- Legacy technologies such as voice-over-TDM and X.25
- Ability to develop generic software components and models—the management system can hide much of the complex underlying detail of running the network
- Client/server design
- Managed object design, part of the modeling phase for the management system
- MIB design—often there is a need for new objects in the managed devices to support the management system

This is an impressive set of skills for even the most experienced engineer. An excellent overall knowledge of these areas is needed along with an ability to focus on any one of them. The general migration to a layer 3 infrastructure is another reason for the widening gap between available development skills and required product features. Natural attrition, promotions, and new entrants to the industry ensure that there is a steady supply of engineers who are fairly *unlikely* to have all the required skills. Added to this is the need for customers to see rapid ROI for all infrastructural purchases. It seems a different type of approach is needed for developing management systems, one that involves adoption of:

- A solution mindset
- Distributed, creative problem solving
- Taking ownership
- Acquiring domain expertise
- Embracing short development cycles
- Minimizing code changes
- Strong testing capability

Acquiring skills like these would positively enhance the development process. We examine strategies for developing these capabilities in the next chapter. For now, the elements of them and their advantages are described.

Developer Note: A Solution Mindset

Adopting a solution mindset is an important first step in effective NMS development. It reflects the move away from the purely technological aspect of products to embrace the way enterprises and service providers now look at overall solutions to business problems. The days of box shifting (selling NEs with little or no management facilities) are probably gone forever. This is no bad thing because the real cost of adding NEs to networks is the time and money required to debug and integrate them into the fabric of the network. This is why network operators require ease of management for new NEs. An extra problem for vendors is that hardware is increasingly sold at heavily discounted prices. The revenue from hardware sales may no longer match the effort required for development, though upgrades and enhancements may help offset this. Customers will pay for overall solutions that make it easier to manage and operate their networks. An example is consolidation of incompatible NMS into a single NMS, as we saw in [Chapter 1](#).

Solutions have a number of characteristics:

- Clear economic value
- Fulfillment of important requirements
- Resolution of one or more end-user problems

An issue facing many network operators is what to do with legacy layer 2 equipment. Should the operator simply throw away its existing hardware? This is a difficult question, and providing a migration path for such users is a good example of a solution. Existing deployed device software should also be maintained by the vendor for as long as possible in order to protect the network operator investment. (This is often easier said than done, as devices such as PABXs are increasingly discontinued because they have reached the end of their lifecycle.) Large networks don't change overnight, so management systems should be written to accommodate both legacy and new equipment. The MPLS ships-in-the-night (SIN) option that we discussed in [Chapter 2](#) is an example of such an approach. SIN is a special mode of operation on MPLS nodes. It allows ATM users to upgrade their firmware (some devices may also need hardware upgrades) to MPLS and then simultaneously use both ATM and MPLS control planes on the same switch. The two technologies do not interact, but pass each other like ships in the night (hence the name). The logical progression of this is to try to allow any layer 2 service to cross an MPLS cloud. This is a good example of solutions thinking because it saves money, protects existing investments, and addresses important user problems.

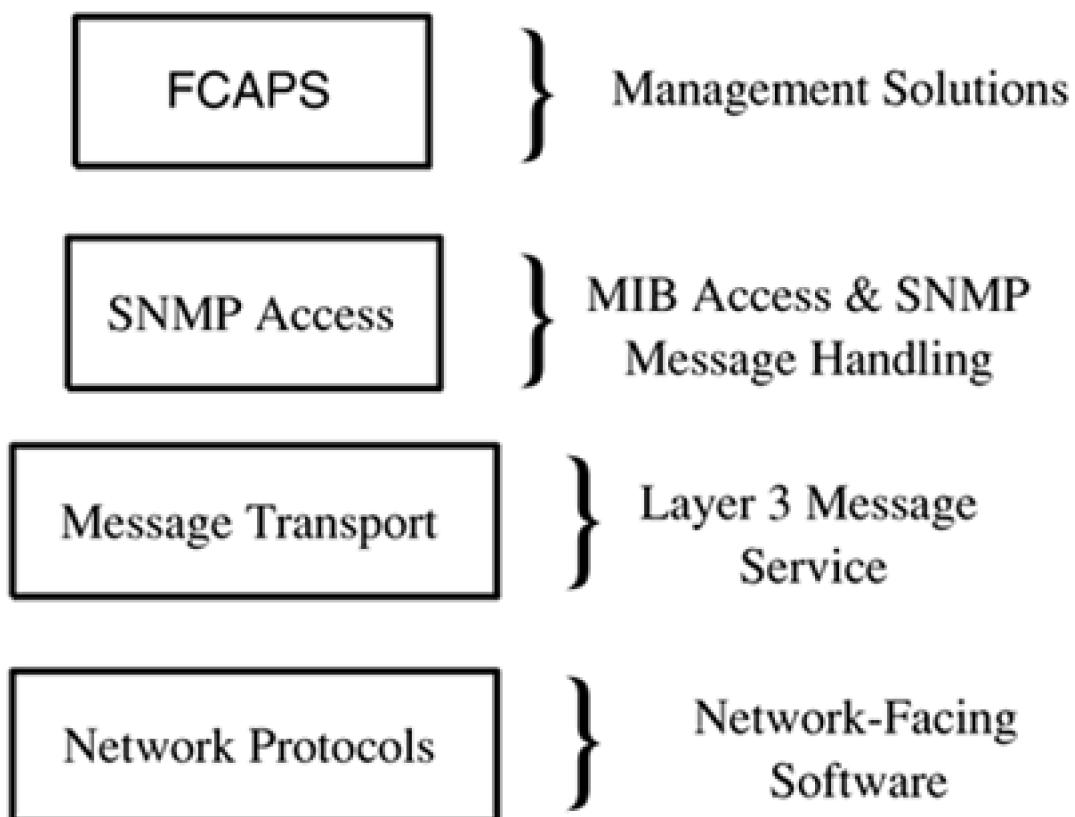
Well-engineered management solutions are also of benefit to vendors when they are built from components. The elements of such solutions can be re-used in other areas and products. The vendor can leverage the solution for many purposes. Examples of management systems solutions include the following:

- Providing minimal management support for third-party devices. Many NMS are proprietary, supporting only the equipment vendor's hardware. Networks may contain multivendor NEs, so separate NMS are often required to support what are often very similar devices from different vendors. It is better for end users if the incumbent NMS provides limited (rather than no) support for third-party NEs. NMS vendors should be prepared to offer this support even if it means just device discovery and notification/trap handling.

- Creating generic management system components that can be used across numerous different products and technologies, such as ATM and MPLS connections. An ATM virtual circuit is not the same thing as an MPLS LSP, but the management software can still provide a technology-independent GUI abstraction for both. The user is then freed from the complexity of the underlying technologies and can perform similar management operations for both. This also reduces training time.
- Aiming for technology-independent software infrastructure using standard middleware, such as CORBA-based products, rather than custom-built facilities.

As far as possible, the management system should also provide code encapsulation for functions such as SNMP access, network message transport, and network protocols. This is illustrated in [Figure 3-4](#), where the FCAPS areas are shielded from the complexities of the underlying SNMP, messaging services, and network technologies.

Figure 3-4. FCAPS software layers.



While this seems an obvious point relating to good software development practice, it's surprising how often low-level code (such as SNMP API calls) is called directly from the FCAPS layer. In many cases, this is just poor coding practice caused by inexperience or excessive haste. Then, the smallest change in the low-level code requires a full FCAPS rebuild. It is important that changes to MIBs or underlying protocols should *not* necessitate a full rebuild of the management system. This loose coupling (via APIs and layering) between components makes it easier for developers to take ownership of substantial product areas. In turn, this can help in avoiding situations in which a change to one component breaks the code in another component.

Developer Note: Distributed, Creative Problem Solving

Once management systems have been built and integrated into a vendor test network, they often present complex problems. The distributed nature of managed networks (NEs with local agents reporting to a centralized NMS) and the broad functional composition of NMS present difficult logistical development problems. Solving such problems is where the expanded skill set comes into its own. Typical problems are:

- Software bugs
- NE bugs (can be very hard to identify)
- Performance bottlenecks in any of the FCAPS applications due to congestion in the network, DBMS, agent, manager, and so on
- Database problems such as deadlocks, client disconnections, log files filling up, and so on
- Client applications crashing intermittently
- MIB table corruption, such as a number of `set` operations that only partially succeed—for example, three `setRequests` (against a MIB table) are sent but one message results in an agent timeout and the other two are successful, which could leave the table in an inconsistent state
- SNMP agent exceptions

Solving these and other problems requires a wide-ranging view of the system components and an excellent understanding of technologies like SNMP, databases, and networking. It also requires a creative approach to the use of debuggers, MIB browsers, trace files, and so on. Clearly, part of creative problem solving is a requirement for developers to have a high aptitude for testing. Such developers leverage their product and software knowledge to comprehensively test the system *prior* to delivery to QA. This helps in providing solid software builds to QA. Organizations play a role in facilitating this by the provision of many of the excellent tools available, including:

- UML support packages
- Java/C++/SDL products
- Version control
- Debuggers

The ultimate goal is zero-defect software. The complexity of NMS code often means that bug fixes can take a long time, often a day or more. Taking the time to do this is nearly always a good investment, provided any changes are properly tested.

Developer Note: Taking Ownership

Taking ownership is another important part of a solution mindset. In this, engineers strive to produce a complete feature without the need for handing off part of the development to others. A broad task can be ring-fenced by a small group of developers who take responsibility for design, development, and delivery. Given the skill set required for management system development, this is a difficult undertaking. It means that traditional development boundaries are removed: no more pure GUI, backend, or database developers. All NMS software developers should strive to extend their portfolio of skills to achieve this.

Another aspect of taking ownership is being prepared to fix bugs in old code produced in earlier projects. This can be achieved in conjunction with maintenance and support developers. The important point is that ownership is maintained even as new projects are undertaken. This has the additional merit of extending institutional memory and minimizing the incidence of coding errors during support bug fixes. Institutional memory relates to individual developers with key knowledge of product infrastructure. It equips the organization to smoothly migrate products over numerous release cycles and is an essential skill for long-term development. The end result is more robust management software in customer networks.

Developer Note: Acquiring Domain Expertise and Linked Overviews

Many service providers employ domain experts for producing documents such as bid requests and requests for proposal. These are highly detailed documents that are sent to vendors. Service provider domain experts may be permanent staff or external consultants. Vendors tend to employ sales and marketing executives as inhouse domain experts. The interplay between these two groups ultimately drives much of the vendor's engineering effort. Both groups of domain experts tend to have impressive expertise. It is important that these skills are also available in engineering, because domain experts [JavaDev] tend to be in great demand. In other words, engineers need to become domain experts as well.

Domain expertise represents a range of detailed knowledge, such as IP/MPLS, that can be readily applied to the needs of an organization. For service providers, the knowledge of their domain experts is leveraged for structuring bid and proposal documents and generally formulating short-, medium-, and long-term strategies. Such knowledge might include areas such as:

- Layer 2 and layer 3 traffic engineering
- Layer 2 and layer 3 QoS
- Network management
- Convergence of legacy technologies into IP. Many service providers have built large IP networks in anticipation of forecasted massive demand. These IP networks are, in many cases, not profitable, so service providers are keen to push existing revenue-generating services (such as layer 2) over them.
- Backward and forward compatibility of new technologies, such as MPLS. An example is that of a service provider with existing, revenue-generating services such as ATM, FR, TDM, and Ethernet. The service provider wants to retain customers but migrate the numerous incoming services into a common MPLS core.

The choice of technology, systems, and devices in each of the above areas is critical and is an opportunity for one or more domain experts.

Domain expertise is needed by engineers, for example, when adding technologies such as Gigabit Ethernet or MPLS to a suite of IP routers or ATM switches. The acquisition of domain expertise is an essential component of solutions engineering. This is easier said than done, because the number of technologies is increasing at the same time as layers 2 and 3 converge. Interestingly, the boundaries of modern networks are also shifting: Devices that were in the core a few years ago are now moving out to the edge. Also, devices that were in the access layer can be enhanced and moved into the distribution layer. In many cases, the different network layers may have individual management systems. The movement of devices across the layers means that support for a given NE may have to be removed from one management system and added to another. This adds to the knowledge burden of developers. Acquiring domain expertise is necessary for hard-pressed developers.

A key to becoming a domain expert lies in what we call *linked overviews*, described in the next section.

[[Team LiB](#)]

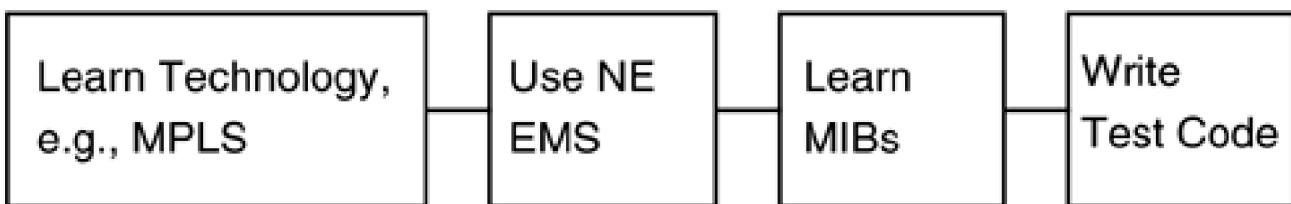
 PREVIOUS  NEXT 

Linked Overviews

An increasingly common problem faced by NMS software developers is implementing network management support for new NE features, such as MPLS. The following four steps provide a linked overview of the management aspects of an NE feature:

1. Acquire a basic understanding of the managed technology—what it does, how it operates, and so on. For MPLS, the basic purpose is to carry traffic across an MPLS core, so connections (LSPs and tunnels) are important. These tunnels can be traffic-engineered and can also support QoS, so path and resource blocks (more on these later in the chapter) are needed.
2. Use the EMS to get an understanding of how the NE provides the feature; for example, for MPLS, the user can separately create objects like paths and resource blocks. Then, these can be combined to create MPLS LSPs or tunnels. User manuals are often very useful during this process.
3. View the relevant MIBs using a MIB browser.
4. Write simple test code (e.g., in Java or C++) to view and modify the MIB objects.

Step 4 essentially automates the actions in steps 2 and 3. The software produced in step 4 can then form the basis of Java classes that can eventually be imported into the finished NMS. The final stage in development is then adding code to the NMS to implement the overall MPLS management feature (i.e., FCAPS). An important observation about the above is that it depicts NMS development as a type of reverse engineering. If network management is provided at the end of NE development, then it has a reverse engineering flavor. If the two occur in parallel, then no real reverse engineering effort is required. We therefore view a linked overview as the resulting knowledge emanating from following the above four steps.



Some vendors provide simultaneous releases of both NE firmware and NMS software. In other words, NE and NMS development are inextricably interwoven.

Step 1 can be assisted using the RFCs on the IETF Web site [[IETFWeb](#)]. The other steps are carried out in conjunction with the NEs themselves. Some examples of linked overviews now follow.

Developer Note: An ATM Linked Overview

Many technologies can seem extremely complex at first, and then, as the learning curve progresses, the essential patterns begin to emerge. ATM [[Tanenbaum1996](#)], [[ATMOBJets](#)] is a good example. Stripped down to its bare (linked overview) essentials:

- ATM is a layer 2 protocol suitable for deployment in a range of operational environments (in VLANs and ELANs, in the WAN, and also in SP networks).
- ATM offers a number of different categories and classes of service. The required service level is enforced by switches using policing (traffic cop function), shaping (modifying the traffic interarrival time), marking (for subsequent processing), and

dropping.

- Traffic is presented to an ATM switch and converted into a stream of 53-byte ATM cells.
- The stream of cells is transmitted through an ATM cloud.
- A preconfigured virtual circuit dictates the route taken by the cell stream. Virtual circuits can be created either manually or using a signaling protocol. If no virtual circuit is present then PNNI can signal switched virtual circuits (SVCs).
- The virtual circuit route passes through intermediate node interfaces and uses a label-based addressing scheme.
- Bandwidth can be reserved along the path of this virtual circuit in what is called a contract.
- Various traffic engineering capabilities are available, such as dictating the route for a virtual circuit.

This list provides an overview of ATM technology. It joins (or links) the principal components needed for managing ATM. From this list, the essential ATM managed objects can be derived:

- ATM nodes
- A virtual circuit (switched, permanent, or soft) spanning one or more nodes
- A set of interfaces and links
- A set of locally significant labels used for addressing
- An optional route or designated transit list
- A bandwidth contract
- Traffic engineering settings
- QoS details

This is a good start on the road to defining managed objects for the support of ATM. It points to the merit of studying documents from the ATM Forum and ITU-T Broadband ISDN. The next stage (step 2) would be to experiment with the EMS of an ATM switch and see the above objects in practice, e.g., creating PVCs and SPVCs. Next, we would examine the MIB objects (step 3) [[ATMObjects](#)] involved in step 2, and then produce software (step 4) to read and write instances of those objects.

Developer Note: An IP Linked Overview

The convergence of layers 2 and 3 (e.g., connection-oriented operation of layer 3 networks) has forced the need for knowledge about IP onto practically everyone's agenda. IP is often used as an abbreviated form of TCP/IP, and this is the way it is used in this book. Like UNIX and Ethernet, IP is one of the great software engineering feats of the 20th century. Both are in widespread use (although UNIX, unlike DOS or Windows, has no single standard implementation) and have withstood the test of time. IP has a steep learning curve, but it can be summarized into a linked overview as follows:

- IP is packet-based—IP nodes make forwarding decisions with every packet.
- IP is *not* connection-oriented.
- IP provides a single class of service: best effort.
- IP does not provide traffic engineering capabilities.
- IP packets have two main sections: header and data.
- IP header lookups are required at each hop (with the present line-rate technology, lookups are no longer such a big issue. Routing protocol convergence may cause more problems).

- IP devices are either hosts or routers (often called gateways).
- Hosts do not forward IP packets—routers do.
- IP devices have routing tables.
- IP operates in conjunction with other protocols, such as OSPF, IS-IS, Border Gateway Protocol 4 (BGP4), and Internet Control Message Protocol (ICMP).
- Large IP networks can be structured as autonomous systems made up of smaller interior areas or levels.

So, the essential managed objects of IP are:

- IP nodes (routers, hosts, clients, servers)
- IP interfaces
- IP subnets
- IP protocols (routed protocols such as TCP/IP and routing protocols such as OSPF and IS-IS)
- Interior Gateway Protocol (IGP) areas (OSPF) or levels (IS-IS)
- Exterior Gateway Protocol (EGP) autonomous systems

The next stage (step 2) would be to experiment with the EMS of an IP router (or an IP/MPLS switch) and see the above objects in practice, for example, creating IP interfaces and subnets, and configuring routing protocols. Next, we would examine the MIB objects (step 3) involved in step 2 and then produce software (step 4) to read and write instances of those objects.

Embracing Short Development Cycles

The need for immediate ROI is prompting a demand for innovative management system solutions. Enterprises must be able to leverage their investments for both productivity (easier operation of networks) and financial gains (smoother business processes). This can result in shorter NMS software development cycles, which in turn requires a modified approach to producing NMS:

- Reduced feature sets in more frequent releases
- Foundation releases
- Good upgrade paths
- Getting good operational feedback from end users

If a management system release occurs every four or five months, then it is no longer necessary for every single requirement to be fulfilled. Instead, requirements can and probably should be prioritized over a range of releases. When a new technology is adopted and implemented on a range of NEs (such as VoIP or FR interworking), then only the mandatory management facilities should be implemented first. The rest can follow later. The first release becomes a foundation for the later ones. In time, the initial reduced feature set grows steadily to become part of a sophisticated end-user solution. Not all users would necessarily upgrade—just the ones who need the new foundation release features. The developers would carry out the bulk of the testing. As the releases occur, the user's data has to be carefully protected and upgraded as necessary. So, upgrade issues (such as scalability-related database schema changes) increasingly have to become part of the bread-and-butter of development.

Implicit in all this is the end user becoming a development partner providing valuable operational feedback. The user receives regular, reliable releases, and the vendor sees fast return on development investment. Another benefit is that developers gain experience and expertise with each of the minor releases.

Minimizing Code Changes

Perhaps one of the most difficult software development skills to acquire is the ability to resist changing code. This applies to good and bad code, old and new. A crucial skill for developing NMS software is the ability to make small, focused fixes for specific problems without introducing new bugs. It can be extremely difficult to resist making simultaneous changes to neighboring code, but it is a vital discipline. Unnecessary code changes introduce bugs and increase the need for testing. Every code change should be fully tested as part of a mandatory change control process.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Elements of NMS Development

Development of NMS software is interesting and challenging. This section introduces some of the typical development areas and some important issues that often arise.

NMS Development

A typical scenario for a management system developer is the following:

- Using a browser-based GUI, the developer has provisioned onto the network a managed object such as an ATM virtual circuit or an MPLS LSP.
- The developer wants to check that the software executed the correct actions.
- During provisioning, the developer verifies that the correct Java code executed using a Java console and trace files (similar actions can be done for C/C++ systems).
- The database is updated by the management system code, and this can be checked by running an appropriate SQL script.
- The next step is verifying that the correct set of managed objects was written to the NE. To do this, the developer uses a MIB browser to check that the row object has been written to the associated agent MIB.

Clearly, this type of development requires a broad range of skills. Other skills not mentioned include:

- Data analysis—matching NE data to the NMS database schema
- Data analysis—defining NMS-resident objects that exist in complex component form in the network (an example is a VPN, as discussed earlier in this chapter)
- Upgrade considerations for when MIBs change (as they regularly do)
- UML, Java, and object-oriented development
- Class design for major NMS features, like MPLS provisioning
- GUI development
- Middleware using CORBA-based products
- Insulating applications from low-level code

These are now described in more detail.

Data Analysis

MIBs are crucial components in a management system. Many vendors provide the MIB modules for their NEs in the form of a number of text files. These files can then be incorporated into an NMS and also used in conjunction with MIB browsers. MIBs contain the managed object definitions of interest and are used to derive part of the database schema for the NMS. This latter is the structure and definition of the data used as the basis of the NMS. We assume the use of relational database technology in this book, and the model for this consists of tables. Typically, the NMS database schema contains a great many tables; for instance, there might be a table for storing the details of LSPs, another for PVCs, and so on. The schema represents an overall data definition for the NMS, and the managed object data is also defined here. This latter point means that there is a degree of duplication in that the NMS has a schema and the MIB objects of interest are contained in it as well as in the agent MIBs. This is what was meant earlier in the chapter when we mentioned storing the same data in two different places. The NMS tracks and modifies the values of NE-managed object instances and stores these in its own database.

While the MIBs can be used to form the basis of the management system information model, there are additional elements needed in the NMS database schema. These are default values, or the values used when the database is first built. Examples are –1 for integers and NULL for booleans (i.e., neither true nor false). Later, we will see the need for sensible default values, particularly for MIB objects that can be modified by the NMS.

The database product can be any of the excellent, general-purpose engines available, such as Informix and Oracle.

When MIBs Change: Upgrade Considerations

Supporting different MIB versions is a recurring network management problem. Let's assume that a given network has 20 of the same ATM/MPLS switch model all running firmware revision 3.2. Now, the network operator decides to upgrade five of the switches to firmware revision 4.0 (in order to gain access to some new feature). Upgrading software like this can be expensive if it results in any downtime due to software bugs. The cost can also be increased if extra hardware is needed, such as a processor upgrade, more RAM or flash memory, or extra line cards.

The NMS can provide assistance during upgrades by downloading the new image to the selected switches (and backing up the old image). However, the new features added to the switches means that they now support extended and possibly also new MIB modules. The NMS must continue to be able to interact with the devices not upgraded, so it must be able to recognize both the new and the old MIBs. Following are guidelines for providing an upgrade path following MIB changes:

- Deprecate old objects no longer in use—don't delete them from the MIB if at all possible.
- Keep the MIB object identifiers sequential; add new OIDs as necessary. It is not uncommon for new columns to be added to MIB tables as NEs are enhanced. The old objects should not be modified during any such enhancements in order to maintain backward compatibility.
- Don't change any existing OIDs in MIBs that are currently in use by the NMS. RFC 2578 provides guidelines for this.
- Ensure that MIB files do not have to be changed in order to work with management systems. Sometimes MIBs that successfully compile into agents cannot be parsed into management systems. This can be caused by limitations on the part of the management system or the agent parser. Whatever the reason, it is important that no manual changes are needed in order to incorporate MIBs into an NMS. This underlines the crucial role played by MIBs.

Following these guidelines helps provide a seamless upgrade path for the addition of new MIB objects while at the same time maintaining support for existing ones. MIB objects should only ever be removed with the utmost caution because there may be management system software that relies on their presence. Non-existent MIB objects that are accessed by an NMS will result in an SNMP exception propagating back to the NMS.

Adding new technologies to NEs is a major cause of significant MIB changes. This causes additional problems for the management system because (as we've seen) it derives its managed object model from the MIBs. New MIB objects that are needed in the database require corresponding schema changes. These can be effected using either SQL scripts or special-purpose code. Changing the management system schema is not without risk. Existing application code is affected, and it can easily introduce bugs. The skills required to match MIB changes to schema updates are very important.

UML, Java, and Object-Oriented Development

The use of standards is a recurring theme throughout the networking industry. It is essential for management system developers to adopt a standards-based approach in their work. To this end, we can use linked overviews in conjunction with documents like IETF RFCs and ETSI/ANSI standards. UML [[UMLRumbaugh](#)] provides a standard technique for system development. It provides some very useful tools for both specification and development.

UML allows for the development process to be opened up to a degree that is difficult to match with older methods such as the waterfall model. It allows for the separation of requirements from specification and design decisions by the provision of different views including:

- Structured classification (use cases, classes, components, and nodes)
- Dynamic behavior (describes system changes over time)
- Model management (organization of the model itself)

In effect, UML provides a model-based scheme of development. The model and its associated views then become part of the finished software.

Domain experts and other stakeholders can assist the development by defining the principal actors (either internal system users or end users) in the management system. Use cases provide a means for defining the ways in which the actors interact with the system. Sequence diagrams describe message exchanges that go together to make up transactions. The different views fit well into a solution-engineering context because they naturally allow for several different perspectives. These and the other features of UML can greatly assist the construction of robust management systems. Products like Rational Rose are excellent for this purpose [[RationalRose](#)].

Class Design for Major NMS Features

Class design can be initiated from UML use cases. Using tools like Rational Rose helps to facilitate automation of this process. Normally, generated classes are merely skeletons with no code included. However, even this is useful, because once the main classes are defined, the programming task is bounded. The great merit of UML is that the stakeholders can influence the class structure. This departs from the older approach in which stakeholders just specify requirements.

GUI Development

An important aspect of management system development is the GUI. This is particularly so when the client is thin. A well-designed GUI reduces the need for training and provides an effective tool for managing networks. It provides the user interface and should be as generic as possible. To this end, visual controls should be as technology-independent as possible, for example, using the words like *connection* instead of PVC or LSP/tunnel, or *routes* instead of ATM designated transit list or MPLS Explicit Route Object (we describe these last two objects later in the chapter—for the moment, let's take them as simple paths through a network). The visual controls should also hide as much complexity as possible and provide sensible default values that the user can override if required.

There are many excellent tools, such as Borland JBuilder, available for crafting GUIs. Often, the GUI is the last item of a management system to be fully tested. So, the individual GUI components should be fully tested using tools like JBuilder well in advance of full integration. Developers' problem-solving skills should be focused on fully exercising as much GUI code as possible to reduce delays in delivery.

Middleware Using CORBA-Based Products

CORBA is a suite of specifications issued by the Object Management Group [[OMGWeb](#)]. Using the Internet Inter-Orb Protocol (IIOP), a CORBA-based program from any vendor can interoperate with another vendor's CORBA-based program. This works for almost any combination of platforms, programming languages, and networks. CORBA applications are made up of objects that present an interface. This interface is part of the contract that a server object offers to the clients that invoke it. The interface is defined in the OMG Interface Definition Language (IDL) and is completely separate from the underlying implementation. So, application A written on PC X can call the exported methods of classes in application B on UNIX workstation Y located in a remote network. CORBA provides the glue needed to join many different code components together into what looks like a single system.

The distributed nature of network management provides a good setting for the use of CORBA-based software. One example is SNMP notification/trap management. When the management system receives a trap from the network, it can store the details in a database and then notify another application such as a GUI client. This notification can take the form of invoking an object in a CORBA application implemented on the client machine. This can also be achieved using technologies such as Java Remote Method Invocation (RMI), RPC, or COM, but CORBA provides what is almost complete independence from the underlying systems and networks. This is an extremely powerful capability.

Insulating Applications from Low-Level Code

Insulating applications from low-level code was briefly described earlier and illustrated in [Figure 3-4](#). It is very important that the various layers of management system software be as technology-independent as possible. This is similar in concept to the way in which network architectures are layered. Each layer is implemented independently. Only the services offered to upper layers are exposed [[Tanenbaum1996](#)]. The implementation is encapsulated inside the layer. In a similar fashion, low-level code that provides access to technology such as SNMP, MIBs, IP, and User Datagram Protocol (UDP) should be partitioned as much as possible. Only a simple interface should be exposed to the layer above. This also aids comprehension by other developers. We will see this technique in the MPLS case study in [Chapter 8](#).

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

Expensive (and Scarce) Operational Skill Sets

Skills shortages are not restricted to the supply side of the industry. The growing complexity of networks is pointing to increasingly scarce operational skills. After all the investment has been made and the training courses run, it is up to the network operator to deploy skilled personnel to actually run the network. The skill set required for running networks is increasingly broad and includes knowledge of a wide range of different technologies. These technologies are not restricted to a single layer of the OSI model; they include devices that support the following:

- ATM
- FR
- Gigabit Ethernet
- Optical technologies such as SONET/SDH, DWDM, cross-connects, and multiplexers
- Access
- Transport
- MPLS
- IP
- VPN
- VoIP
- SAN and NAS
- Firewalls, load balancing, servers

Added to this is possibly more than one management system, particularly for complex networks. So, the operator needs to have a reasonable understanding of the various management systems. The types of skills needed by network operators are:

- Deploying and configuring devices
- Setting up and enabling routing and signaling protocols
- Partitioning layer 2 and layer 3 networks
- Billing and accounting
- Planning, including capacity planning
- Performance
- Provisioning of LSPs, PVCs, and so on
- Traffic management
- Backup and restore of NE firmware and configuration databases
- MIB browsing
- Trap management

- Security
- Modeling what-if scenarios
- Understanding application trace files and debug facilities

Just as for developers, this is a complex skill set. Vendors can greatly assist network operators by providing high-quality solutions in both the NEs and NMS. Network operators should also try to keep up with new technologies by studying the relevant standards documents.

Multiservice Switches

We have made much mention of the migration towards a packet-based infrastructure and its relevance to enterprise network operators. We have also noted that enterprise networks usually contain much legacy equipment. Enterprise network operators typically want to:

- Reduce the payback period for new purchases
- Maintain and expand existing network services
- Reduce operational costs associated with multiple networks, such as telephony and LAN

MPLS provides a way of filling these needs in conjunction with multiservice switches. These switches allow specified levels of QoS and traffic engineering for the following technologies:

- ATM
- FR
- TDM
- IP

It is anticipated [[MultiserviceSwitch](#)] that these technologies will be deployed on multiservice switches once the relevant standards have been finalized.

[[Team LiB](#)]

[PREVIOUS](#) [NEXT](#) ▶

MPLS: Second Chunk

[Chapter 2](#) gave a brief introduction to MPLS and described how it enables IP networks to be operated in a fashion similar to layer 2 networks. Currently, ATM is the only layer 2 technology that provides traffic engineering and guaranteed QoS for a range of network service classes—voice, video, and data. A specified class of service can be allocated to new services without affecting the existing ones (provided the network has the appropriate bandwidth resources). MPLS promises to bring similar flexibility to layer 3 networks:^[3]

^[3] We should point out that a lot of work is also underway to use MPLS for the transport of legacy traffic below layer 3, such as Ethernet, ATM AAL5, and voice. MPLS offers similar capabilities to the operators of such networks in the form of LSPs and so on. So, for our discussions we assume that the traffic type being transported is not so important from the management perspective.

This second chunk of MPLS digs a little more deeply into MPLS and more fully describes the principal components of the technology. This is another example of a linked overview, which strips the technology down to its essential components. These can be considered to be the managed objects of MPLS:

- Explicit Route Objects (ERO), strict and loose
- Resource blocks
- Tunnels and LSPs
- In-segments
- Out-segments
- Cross-connects
- Routing protocols
- Signaling protocols
- Label operations: lookup, push, swap, and pop
- Traffic engineering
- QoS

As we'll see, the hardest way to manage MPLS networks is to not use signaling to set up LSPs. Why would a user not want to use signaling? The NEs might not support signaling. Another reason is control: The operator alone might want to dictate all the objects that get provisioned in an unsignaled network. Some large service providers actually operate their ATM networks in this way and might decide to do the same with MPLS. With no signaling support, it is up to the operator to create all of the above objects—we will do just this in [Chapter 8](#) when we manually create an LSP. Another reason for not wanting signaling protocols running in the network is the extra resources they consume. All networks are only as strong as their weakest link, and with complex signaling protocols running in the background, bottlenecks may suddenly appear unexpectedly. We don't necessarily agree with running or not running any particular signaling protocols—we just mention the above as possible explanations. One important point is that signaling is probably required as networks become very large (tens to hundreds of thousands of nodes).

We describe LSP setup using the MPLS MIBs in some detail in [Chapter 8](#).

Explicit Route Objects

An ERO is a list of layer 3 address hops inside an MPLS cloud. Similar to an ATM designated transit list (DTL), it describes a list of MPLS nodes through which a tunnel passes. The path taken by the tunnels in [Figure 3-3](#) is an example of an ERO. The purpose of an ERO is to allow the user to specify the route that a tunnel will take. In other words, it allows the user to constrain the route. EROs can be either strict or loose. A strict ERO specifies *all* the hops in the path. A loose ERO allows for intermediate networks in the path, such as another cloud (e.g., an SP network). EROs are stored in a MIB table on the originating node and can be used by more than one tunnel originating on that MPLS node. EROs are *not* used in the manual creation of LSPs.

EROS are used by signaling protocols (such as RSVP-TE) to create tunnels. The path specified in the ERO must be realizable (i.e., links must exist between the designated nodes) and any required bandwidth resources (described in the next section) must be available.

Resource Blocks

MPLS permits the reservation of resources in the network. This provides a means for network operators to deterministically carve up their bandwidth and allocate it to specific LSPs. Resource blocks provide a means for recording the bandwidth settings, and they can then be assigned to specific LSPs. The components of a resource block include:

- Maximum reserved bandwidth
- Maximum traffic burst size
- Packet length

A given LSP, such as one of those in [Figure 3-3](#), could have an associated end-to-end bandwidth reservation of 2Mbps. This means that the LSP is engineered to carry 2Mbps of traffic along the specified route. Best-effort LSP operation is also possible, in which case the resource block is null.

Tunnels and LSPs

MPLS tunnels (as we saw in [Chapter 2](#), [Figure 2-8](#)) represent a type of container for paths made up of nodes with configured in-segments, cross-connects, and out-segments. The tunnel is the on-ramp into the associated label switched path. What we didn't say in [Figure 2-8](#) is that tunnel instances are also supported. A tunnel instance is a separate tunnel, which is in some sense owned by the first tunnel. In other words, the tunnel and its tunnel instances have a relationship with each other. This relationship consists of sharing the same endpoints and might be needed for backup (a disjoint path) or for load sharing. We will see this more clearly in [Chapter 8](#).

MPLS-encapsulated packets enter the tunnel, pass across the appropriate path, and exhibit three important characteristics:

- Forwarding is based on MPLS label (rather than IP header) lookups (this is no longer an advantage enjoyed by MPLS nodes, as IP routers can now forward at line rate).
- Resource usage is fixed, based on those reserved at the time of connection creation.
- The path taken by the traffic is constrained by the path chosen in advance by the user.

Tunnels and LSPs provide reachability for traffic with specific destination IP addresses. Routing protocols direct packets onto specific tunnels and LSPs in order to reach the appropriate IP destination.

In-Segments and Out-Segments

In-segments on an MPLS node represent the point of ingress for traffic. Out-segments represent the point of egress for traffic. The two types of segment objects are logically combined using a cross-connect.

Cross-Connects

Cross-connects are objects that associate in- and out-segments together. The MPLS node uses the cross-connect settings to decide how to switch traffic between the segments. The cross-connect table supports the following connection types:

- Point-to-point
- Point-to-multipoint
- Multipoint-to-point

A cross-connect entry has both an administrative status and an operational status. The administrative status indicates the state required by the operator, whereas the operational status indicates the actual state of the cross-connect in the node. Operationally down cross-connects will not forward packets.

Routing Protocols

MPLS incorporates standard IP routing protocols such as OSPF, IS-IS and BGP4. This is done because these protocols have been used and proven over the course of many years. Incorporating them into the MPLS standards improved the chances of widespread deployment of MPLS. Traffic engineering extensions added to the routing protocols means that they can advertise and distribute both routing and resource (e.g., link bandwidth) details. This is crucial for facilitating the creation of route-constrained LSPs (i.e., tunnels). This ultimately allows the user requirements to influence the path taken by IP traffic through an MPLS cloud.

Signaling Protocols

As we've seen, the creation of LSPs and tunnels can be achieved either manually (similar to the way ATM PVCs are created) or via signaling. Signaled connections have resources reserved, labels distributed, and paths selected by protocols such as RSVP-TE or LDP.

Label Operations

As MPLS-labeled traffic traverses an LSP or tunnel, it is forwarded based on its encapsulated label value. The IP header is no longer consulted while the packet is inside the MPLS domain. MPLS labels can be stacked in a last-in-first-out fashion; that is, more than one label can be applied to a packet. Labels can be stacked (if the hardware supports it) up to the limit allowed by the layer 2 protocol. The label counts as part of the layer 2 message size. The outermost label (i.e., the last one pushed on the stack) is the one used for forwarding the packet. If more than one label is on the stack, then stripping off (or popping) the topmost label exposes the next label down.

Forwarding is then carried out based on this label. This is illustrated below with an MPLS-encapsulated packet that has a stack containing two labels.



The current MPLS node uses Label2 when forwarding this packet. The operations that can be executed against labels are:

- Lookup: The node examines the value of the topmost label. This operation occurs at every node in an MPLS cloud. In our example, lookup would occur using Label2. Typically, a label lookup results in the packet being relabeled and forwarded through a node interface indicated by the incoming label.
- Swap: This occurs when an MPLS node replaces the label with a new one.
- Pop: This occurs when the topmost label is removed from the stack. If the label stack has a depth of one, then the packet is no longer MPLS-encapsulated. In this case, an IP lookup can be performed using the IP header.
- Push: This occurs when a label is either pushed onto the label stack or attached to an unlabeled packet.

In [Chapter 4](#), "Solving the Network Management Problem," we will see that the MPLS shim header contains a bit field called Stack. A value of 1 in the stack field indicates that this is the last remaining label in the stack; the value zero indicates that other labels are pushed beneath the current label. The value of the Stack field changes appropriately as labels are pushed and popped. An important point to note is that the MPLS labels have local significance only. The contents of MPLS labels can also assist in the QoS scheme (we will see this use of labels in more detail in [Chapter 4](#)).

MPLS Encapsulation

FR and ATM can accommodate MPLS labels in their layer 2 headers. Other technologies use a shim header for the label. This is a mini-header (more than one is allowed because stacking is supported) that sits beside the IP header. FR uses the data link connection identifier (DLCI) field, and ATM uses the VPI/VCI fields. ATM has another field called cell loss priority (CLP) that is used for QoS support. This field is used to mark cells for two service levels: Cells with a CLP of 1 are discarded (if incoming traffic levels are too high) prior to cells with a CLP of 0. Cells with a CLP of 0 are not guaranteed to be forwarded, but they will have precedence over cells with CLP of 1.

The MPLS encapsulation specifies four reserved label values:

- 0 – IPv4 explicit null that signals the receiving node to pop the label and execute an IP lookup
- 1 – Router alert that indicates to the receiving node to examine the packet more closely rather than simply forwarding it
- 2 – IPv6 explicit null
- 3 – Implicit null that signals the receiving node to pop the label and execute an IP lookup

When an MPLS node operates in SIN mode (ATM and MPLS running simultaneously), there may be additional constraints on the label range, but this is platform-specific.

QoS and Traffic Engineering

The Internet currently offers a single default service level: best effort. Many enterprises also offer just best-effort QoS for IP traffic but on an overengineered underlying network. Bandwidth in the LAN is relatively cheap and can be augmented as needed using switches. This means that the excess bandwidth helps avoid congestion, but this is a blunt instrument that doesn't scale well and is prone to congestion

during very high bursts of traffic. In effect, overengineering passively avoids congestion but doesn't proactively prevent it. It's not unknown for the developers of NMS software to inadvertently flood their local LAN with SNMP traffic. Without proactive QoS and traffic engineering in the network, the enterprise network manager often has to resort to restricting access to the network to avoid such problems. A better solution would be to provide specific chunks of bandwidth (via MPLS LSPs) to the developers. WAN links normally have strictly limited bandwidth and provide an even stronger case for QoS management.

Traffic engineering is set to become a mandatory element of converged layer 3 enterprise networks [[MPLS&Profits](#)]. MPLS provides resource advertisements in its routing protocols (extended from the regular IP operation). Link bandwidth states are included as extensions of the standard IP routing protocols such as OSPF and IS-IS. These are traffic-engineering enhancements, and the modified protocols are referred to as OSPF-TE and IS-IS-TE. The purpose of the enhancements is to allow MPLS routers to construct a complete link-state database of the network, including available/allocated bandwidth.

QoS

The need for definable levels of QoS is due to the increasingly mission-critical, real-time applications being deployed on enterprise and SP networks. Overengineering of the core gets you only so far, but the WAN may then become a bottleneck (remember that a network is only ever as strong as its weakest link). Once the overengineered cores become depleted, there is a need for IP QoS. The issue of SP core depletion is interesting and slightly controversial because it is estimated [[TimesMarch2002](#)] that between 1998 and 2002, service providers invested about \$500 billion in fiber-optic networks and that only four percent of the new fiber is in use. The bulk of the fiber is idly sitting underground. Quite possibly, core depletion is therefore not an immediate issue for service providers, but this is not the case for enterprises.

An allied problem is due to the way in which enterprise and SP networks are inextricably interwoven. When an enterprise wants to send, say, email, Web, FTP, and VoIP traffic through an SP network, then a number of approaches can be taken by the enterprise:

1. It can rate all of its traffic as being equally important.
2. It can rate the VoIP traffic as being the most important.

Unfortunately, most enterprises choose the first option. This may be because:

- The enterprise equipment is not capable of marking traffic passed to the service provider.
- There is no incentive from the service provider.
- It is difficult to manage from within the enterprise network.

Whatever the reason, this causes a downstream problem for the service provider because there may be no easy way of differentiating the incoming traffic streams. In practice, it matters little if an email message arrives at its destination one minute or one second after sending it. The same would not be true of real-time, delay-sensitive traffic such as voice or video. As traffic levels increase, these issues become more pressing for service providers, prompting the need for QoS to be applied in both the enterprise and SP networks. Service providers often deploy (and manage) customer premises equipment (CPE) to execute such functions.

There are essentially three approaches that can be adopted for providing different levels of network service [[CrollPackmanBW](#)]:

- Best effort, as provided by the Internet
- Fine granularity QoS, such as Integrated Services (IntServ)
- Coarse granularity QoS, such as Differentiated Services (DiffServ)

We're all familiar with the best-effort model from surfing the Internet. The Internet does its best to deliver traffic between senders and receivers, but there is no guarantee. This works well for non-real-time traffic, such as email and Web site access. Internet telephony is also reasonable as long as the users aren't too fussy about quality.

Another approach to service provision is fine granularity QoS of which the IntServ model is an implementation. The IntServ model allows path (called microflow) reservation across the network, and traffic is pushed into these paths in order to get to its destination. The paths have to be explicitly reserved, and they must also be periodically refreshed. IntServ implementations have two elements of overhead made

up of path handling and refreshing.

The third approach is coarse granularity QoS, and it uses the technique of traffic marking. This is the model used by DiffServ. It is often considered to be more scalable than IntServ because no microflows are required. Packets are marked (at the point of origination or by a downstream NE such as a customer edge router) with a specific value. This value is then used as the packet is forwarded through the network. For DiffServ, the values are called DiffServ Code Points (DSCP), each of which corresponds to a given traffic-handling treatment. As the marked traffic makes its way through the network, it is processed at each node in accordance with its DSCP. The DSCP specifies what is called a behavior aggregate; that is, all packets with this DSCP fall into the same category. Each category represents a distinct class of service. When a router receives a packet marked for a given class of service, it imposes what is called a per-hop-behavior (PHB) on the packet. This is the way the packet is treated by the node.

The PHB consists of queuing strategy, policing (dropping the packet or remarking it), shaping, classifying, and so on. It has two main functions:

1. Scheduling the packet into the egress interface
2. Discarding the packet using a drop priority

Scheduling refers not to how packets are stored in the queues, but how they are pulled from the queues by the scheduler. The QoS experienced is the final result of all this activity as this packet (and associated packets) makes its way through the network.

In [Chapter 4](#) we will see how these models are used in the context of MPLS. For the moment, we illustrate how IP packets are marked with DSCPs in the Differentiated Services (DS) field (formerly known as the Type of Service) in the IP header. This is illustrated in [Figure 3-5](#), where the IP header DS field is located in the second octet position. The value assigned to this field can be used to provide specified forwarding treatment for the packet as it traverses the DiffServ domain.

Figure 3-5. The IP header.

Bit Positions																																																								
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																									
Version	Header Length	Differentiated Service (DS)						ECN	Total Length																																															
Identification										Flags	Fragment Offset																																													
Time To Live (No. of hops)					Protocol (e.g., UDP)					Header Checksum																																														
Source IP Address (32 bits)																																																								
Destination IP Address (32 bits)																																																								
Options																																																								

RFC 3260 provides an update, clarifications, and new elements of DiffServ. Among other things:

- It clarifies that the DSCP field occupies only the first six bits of the DS field.
- The other two bits will be used for Explicit Congestion Notification (ECN) [[DavieRehker2000](#)], as can be seen in [Figure 3-5](#).

Downstream switches and routers then take the DS value as the cue for special treatment—that is, differentiated services. Packet service metrics can include:

- Bandwidth allocation
- Packet loss rate
- Delay
- Jitter (variation in delay)

The value in the DS field can be used to dictate these metrics. Clearly, the nodes traversed by the marked packets must know the meaning of the DSCPs. This is an important part of DiffServ: the mapping between DSCP and the forwarding treatment. Ultimately, this is reflected in the end-user service level.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

MPLS and Scalability

In the standard MPLS MIBs, the tunnels on a given NE reside in the `mplsTunnelTable`. [Figure 3-6](#) illustrates an extract from the MPLS Traffic Engineering MIB [[IETF-TE-MPLS](#)].

Figure 3-6. The MPLS tunnel table.

```

MplsTunnelEntry ::= SEQUENCE {
    mplsTunnelIndex          MplsTunnelIndex,
    mplsTunnelInstance        MplsTunnelInstanceIndex,
    mplsTunnelIngressLSRId   MplsLsrIdentifier,
    mplsTunnelEgressLSRId   MplsLsrIdentifier,
    mplsTunnelName            DisplayString,
    mplsTunnelDescr           DisplayString,
    mplsTunnelsIf             TruthValue,
    mplsTunnelIfIndex         InterfaceIndexOrZero,
    mplsTunnelXCPPointer      RowPointer,
    mplsTunnelSignallingProto INTEGER,
    mplsTunnelSetupPrio       INTEGER,
    mplsTunnelHoldingPrio     INTEGER,
    mplsTunnelSessionAttributes BITS,
    mplsTunnelOwner            INTEGER,
    mplsTunnelLocalProtectInUse TruthValue,
    mplsTunnelResourcePointer RowPointer,
    mplsTunnelInstancePriority Unsigned32,
    mplsTunnelHopTableIndex   MplsPathIndexOrZero,
    mplsTunnelARHopTableIndex MplsPathIndexOrZero,
    mplsTunnelCHopTableIndex  MplsPathIndexOrZero,
    mplsTunnelPrimaryInstance MplsTunnelInstanceIndex,
    mplsTunnelPrimaryTimeUp   TimeTicks,
    mplsTunnelPathChanges     Counter32,
    mplsTunnelLastPathChange  TimeTicks,
    mplsTunnelCreationTime    TimeStamp,
    mplsTunnelStateTransitions Counter32,
    mplsTunnelIncludeAnyAffinity MplsTunnelAffinity,
    mplsTunnelIncludeAllAffinity MplsTunnelAffinity,
    mplsTunnelExcludeAllAffinity MplsTunnelAffinity,
    mplsTunnelPathInUse       MplsPathIndexOrZero,
    mplsTunnelRole             INTEGER,
    mplsTunnelTotalUpTime     TimeTicks,
    mplsTunnelInstanceUpTime  TimeTicks,
    mplsTunnelAdminStatus     INTEGER,
    mplsTunnelOperStatus      INTEGER,
    mplsTunnelRowStatus       RowStatus,
}

```

1 Sequence or Table

```
mplsTunnelStorageType    StorageType }
```

[Figure 3-6](#) shows the objects contained in the `mplsTunnelTable`. The `mplsTunnelTable` is made up of instances of `MplsTunnelEntry`, as seen by arrow 1 in [Figure 3-6](#).

Each entry in this table should be seen as a column in a row; for example, `mplsTunnelIndex` can be considered a key value (in the relational database sense). This is depicted in [Table 3-1](#), where some of the columns are arranged and assigned specific values. The exact meanings of the entries in [Table 3-1](#) are explained in [Chapter 8](#). For the moment, a short description is given.

Table 3-1. MPLS Tunnel Table Excerpt

MPLSTUNNEL INDEX	MPLSTUNNELHOP TABLEINDEX	MPLSTUNNEL INGRESSLSRID	MPLSTUNNEL NAME
1	1	LER A	TETunnel_1
2	1	LER A	TETunnel_2
3	1	LER A	TETunnel_3
5	1	LER A	TETunnel_5

The column `mplsTunnelIndex` provides a unique key value for each tunnel on the node in question, starting at 1 and increasing with each entry added to the table (tunnel instances are described in [Chapter 8](#)). The column `mplsTunnelHopTableIndex` provides an index into a hop table that describes the path taken through the MPLS cloud by the tunnel. The column `mplsTunnelIngressLsrid` is the designated ingress node for the tunnel and has the value LER A for all the tunnels listed in [Table 3-1](#). This column would most likely be either an IP address or a router ID, but a name is chosen here for simplicity. The column `mplsTunnelName` is simply a text descriptor for the tunnel. One notable feature of [Table 3-1](#) is that there is no entry for index 4. This can occur when the user deletes the fourth entry. The table entries are *not* then moved up to fill the gap.

This table can typically include millions of rows (as mentioned earlier in the Light Reading Trials). Let's assume that each row is roughly 300 bytes in size. That means the overall size of the `mplsTunnelTable` for an SNMP agent containing 3 million LSPs is $3,000,000 * 300$, or just under 9MB. This would assume a network containing possibly tens or hundreds of thousands of MPLS nodes. It is not practical to try to read or write an object of this size using SNMP. Unfortunately, such an operation might be necessary if a network is being initially commissioned or rebalanced after adding new hardware. Also, many NMS provide a connection discovery feature that must retrieve all virtual circuits (ATM/MPLS) from the network and figure out details for each circuit, such as traffic resource allocations and links traversed. One way to assist in improving scalability is to indicate in the MIB which objects have changed. A scheme for this would be to provide a second table called a tunnel-change table linked to the tunnel table. The tunnel-change table could have summarized boolean entries for a block of tunnel table entries. Let's say we have 1,000,000 tunnels and we assign a block size of 10,000 to the tunnel-change table. This means that any changes in the first 10,000 tunnels are reflected in the first entry in the change table. Any change in the next 10,000 tunnels are reflected in the second change table entry. With a block size of 10,000 we would then have 100 entries in the change table, or $100 * 10,000 = 1,000,000$ tunnels. The NMS could then consult the change table in order to see which blocks in the tunnel table have changed. This would help avoid the problem of reading back all the tunnel table entries.

[[Team LiB](#)]

[PREVIOUS](#) [NEXT](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Summary

There are some serious problems affecting network management. Bringing managed data and code together is one of the central foundations of computing and network management. Achieving this union of data and code in a scalable fashion is a problem that gets more difficult as networks grow. MIB tables expand as more network-resident managed objects such as virtual circuits are added. Our first MIB note (on scalability) records a useful object that can help in managing additions to large (integer-indexed) MIB tables. The increased size of networks is matched by ever more dense devices. The latter both help and hinder operators.

The designers of management systems need a rarified skill set that matches the range of technologies embedded in NEs and networks. More emphasis is needed on solutions than on technology, particularly as the components of the technology are combined in new and complex ways, for instance, in layer 2 and layer 3 VPNs. Solutions should try to hide as much of the underlying network complexity as possible. NMS technology can help in hiding unnecessary complexity.

The liberal use of standards documents and linked overviews are some important tools for tackling the complexity of system development, managed object derivation, and definition. Standards documents can be used in conjunction with UML to inform and open up the development process to stakeholders. Like management system developers, network operators also require an increasingly impressive range of skills. Just as for developers, this is both a challenge and an opportunity.

MPLS was described in greater detail, introducing some of its managed objects. Networks must increasingly support a growing range of traffic types. These traffic types require specific traffic engineering and QoS handling in layer 2 and layer 3 networks. The different types of QoS provide the means of implementing the required QoS. While MPLS helps solve many problems, it also can suffer from scalability issues as the number of LSPs increases. Scalability can be improved by incorporating techniques such as change tables in the agent.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Chapter 4. Solving the Network Management Problem

Having seen some of the most pressing network management problems, we now examine possible solutions, including:

- Filling the development skills gap
- Smarter NMS
- Smarter MIBs
- Smarter NEs
- One data model
- Distributed servers
- Policy-based network management
- Directory-enabled networking

Solutions engineered with zero-defect software are the ultimate goal. These solutions should enable end users to efficiently and economically operate their network. Little if any recourse should be needed to the use of NE-level, command-line interface (CLI) utilities such as telnet. The NMS should fit seamlessly into the business processes and workflows of its enterprise or SP customer. These issues are described in the following sections.

[Team LiB]

◀ PREVIOUS NEXT ▶

Filling the Development Skills Gap

[Chapter 3](#), "The Network Management Problem," described a solution mindset as a way of approaching NMS development. This involves a mix of perspectives that combine to cross the boundaries of pure development, such as GUI, backend/server, middleware, database, and so on. Instead, the central focus is on the overall solution rather than on the increasingly diverse component technologies. This is of increasing relevance because of the growing number of technologies being packed into NEs. Managing any given technology resolves down to the associated:

- Managed objects
- Management software

Arriving at these requires a broad range of skills. Linked overviews can help to identify the key elements of a managed technology (such as MPLS). A comprehensive understanding of all aspects of the technology, while useful, is rarely needed for providing management solutions. In spite of this, solution engineers will tend to acquire a deep understanding of the managed technology as they work with it. The key elements of the managed technology can be used to determine the associated managed objects. If a MIB has been supplied for the technology, then the main challenge lies in incorporating the relevant objects into the NMS. In solution engineering, the overall focus is not on a single item of work, such as producing:

- A complete GUI feature consisting of one or more screen pages
- The provisioning code for a feature such as IP traffic engineering
- A topology backend combined with fault management
- Performance monitoring software

Each of these is important, but it is the combination of all the NMS components that forms the overall solution seen and experienced by the end user. For this reason, solution engineering considers the end-to-end solution even beyond the needs of the current software release. Project work on a specific feature, such as adding Gigabit Ethernet management, is continuously viewed with questions such as the following:

- How will this set of options look to a user?
- Does the proposed implementation have real value?
- How easy is it to test and install?
- What are the upgrade implications for adding this feature?
- Are there reusable components in this feature?
- Can any components be adapted for use in other features or products?
- Can test code be saved and used for future enhancements?
- What other infrastructure can be added to help future work?
- Will this feature help users to manage their networks?
- Does the new feature make it easier and faster to perform a task than would using a CLI?
- How will the other features, new and old, be affected by this one?
- In fulfilling the release requirements, can anything else be added to the feature to marginally exceed the remit (i.e., the required

development work)?

The second-to-last point is essential for maintaining continuity of existing function in the NMS. Adding new features always risks breaking existing code and, for this reason, solution engineering encourages broader vision and attention to risk so that unnecessary breakages are minimized. This is a persuasive reason for minimizing changes to existing code (as mentioned in [Chapter 3](#)).

It is perhaps the last point that most clearly characterizes solution engineering: trying to incrementally and safely exceed the requirements within an agreed timescale. This is particularly important as development cycles become shorter because it allows for the grouped releases (starting with the foundation release as described in [Chapter 3](#)) to dovetail into each other. If requirements A, B, and C are needed in the full release, then add A and B in the first release. In the second release add part of C, and finish it in the final release along with enhancements that have emerged during development.

The test cycle required for foundation releases should be short—a truncated version of that required for a major release. This pushes responsibility for testing the foundation releases onto the developers, but it helps to streamline the process. If possible, a skeleton QA team can assist in testing the foundation releases. The emphasis should be on successful, short work cycles achieved using a combination of careful software engineering and lightweight development processes. Major releases, on the other hand, should use the full test cycle and heavyweight development and test processes.

As an example of solution engineering, let's look at adding third-party support to an existing NMS feature, such as MPLS. This consists of adding software to manage not only the vendor's MPLS NEs but also those of other vendors. This is a contentious and interesting issue because adding support for *any* device tends to add value to it as well as to the NMS. So, many vendors may be a little unenthusiastic about supporting and effectively enhancing competitors' NEs. While this is an important issue, we note in passing that enterprise networks tend to be multivendor environments, so third-party support may be mandatory. An important point about the MPLS standard MIBs is that NMS support for compliant third-party NEs may be relatively easy to implement.

MPLS equipment vendors recognize the need for compatibility and interoperability because trials take place regularly in various independent test sites, such as Isocore and the University of New Hampshire.^[1] In these trials, vendors get a chance to competitively test their MPLS implementations. The reasoning is that if all the NEs are demonstrably compatible, then there is little reason for network operators to fear deployment of MPLS devices from different vendors. A consequence of this is the need for NMS to support more than just one vendor's hardware. So, if vendor X decides to add support for its own MPLS devices to its proprietary NMS, then a good solution would also add at least limited third-party support at the same time. This can take the form of allowing the user to create LSPs/tunnels that:

[1] The MPLS Forum also carried out interoperability tests in 2002 involving traffic engineering (using explicit routes), RFC 2547 VPN, and Ethernet over MPLS.

- Originate on vendor X devices and
- Traverse or terminate on vendor Y devices

This means that while the network hosts multiple vendors' NEs, the MPLS content is driven by vendor X devices. However, the limited third-party support should enable the vendor X NMS to auto-learn (discover) NEs from other vendors. It should also be able to process traps from these devices.

We mention in passing a further objection that may be raised to adding third-party device support: It competes with standard NMS such as HP OpenView Network Node Manager. Considerations like these show why adding this type of support is contentious. Existing development resources may already be stretched in providing support for the vendor's own NEs. So, why use precious development time in supporting other vendors' NEs? One reason is that third-party support can greatly assist end users in running their networks. Similar considerations apply to processing traps from third-party NEs because such facilities help users to manage their networks more effectively.

Adding special-purpose NMS software infrastructure for in-field use, such as tracing facilities that can be turned on and off by the end user, can also be beneficial. This is particularly useful when software problems occur on user sites. The user can generate trace files and email them back to the developers for analysis. Alternatively, the developers can dial into the site and generate the trace files themselves. This helps avoid the need for developers to travel and can result in fast problem identification and accurate resolution. If the developers have taken ownership and inserted meaningful trace messages, then most problems can be quickly resolved.

Developer Note: Training for a Solution Mindset—Key Abstractions

There are many ways of adopting a solution engineering mindset apart from the ones already mentioned. (Experience is perhaps the best teacher, but this section describes some additional approaches). Another way of becoming used to solutions in general is to gain expertise in as many as possible of the excellent general-purpose, desktop software packages available, such as those for:

- Web browsing
- Virus detection
- Document processing
- Software development

These are powerful, increasingly GUI-based applications that are applicable across a wide variety of uses, including network management. Identifying and learning the constituent components usefully leverage the intellectual property freely available with these applications. This is beneficial because many of the components have become standard desktop objects, such as:

- Pull-down menus
- Dialog boxes
- Toolbars
- Icons
- Task bars

The more standard the look and feel is for an NMS, the easier it is to use. Ease of use translates into time, effort, and money saved in managing networks.

Most standard applications do not provide source code, so they can be used only to learn the visual and processing paradigms. But even this can help to ensure that the NMS developer adheres to known usability guidelines. The Open Source community [[OpenSourceWeb](#)], however, does provide access to source code, which potentially opens up a treasure trove of standard components for developers. Interestingly, some NMS products [[NMSOpen](#)] are now open sourced. [Linux continues to pose a significant threat to the big commercial software vendors (i.e., Microsoft and Oracle, among others) and server vendors (such as Sun Microsystems)]. Many organizations now deploy Linux on cheap PC-based platforms. It remains to be seen whether Open Source will have a large impact on enterprise and SP network management products. It certainly can't be ignored.

Apart from the visual aspect, just about every application supports some type of document-view architecture [[MicrosoftWeb](#)]. A document in this context is the application data stored in some persistent format, such as disk files or a database. The view is the application infrastructure (GUI, command consoles, etc.) provided for looking at, creating, and modifying the data. This is a useful abstraction for NMS development in which the persistent data is distributed between databases, disk files, and SNMP agent MIBs.

Another important aspect of solution engineering is the ability to think in chunks. We saw this earlier in our ongoing discussion on MPLS, but it can be extended to other areas of development. It represents a form of abstraction similar to the way strong chess players think during chess games—in certain positions specific patterns of pieces need not be considered individually because they are seen as interrelated chunks. In a similar fashion, rather than looking at the overall NMS and trying to think about it in its entirety, it is better to break it up into chunks. The chunk of interest at any given time is then selected out of the set of possibilities, while the others are temporarily left out of consideration. The different layers of an NMS (mentioned earlier) are a good example of these types of chunks.

We mentioned that it is a good idea to (as far as possible) decouple technology-specific code (for provisioning, auto-learning, and trap handling) from other components. This modularization is not only good practice, it also illustrates what might be called *chunk orientation*. Technology-specific code can be organized into a backend block. This can take the form of (among others):

- Windows dynamic-link libraries (DLLs)
- Unix shared libraries

- Java packages
- C++ classes
- C modules

The form taken is not so important; the key is the functional demarcation. This backend code can then present a simple interface to the rest of the software and also to standalone test programs. These simple test programs can then be used to exercise all of the backend code well in advance of full database and system integration.

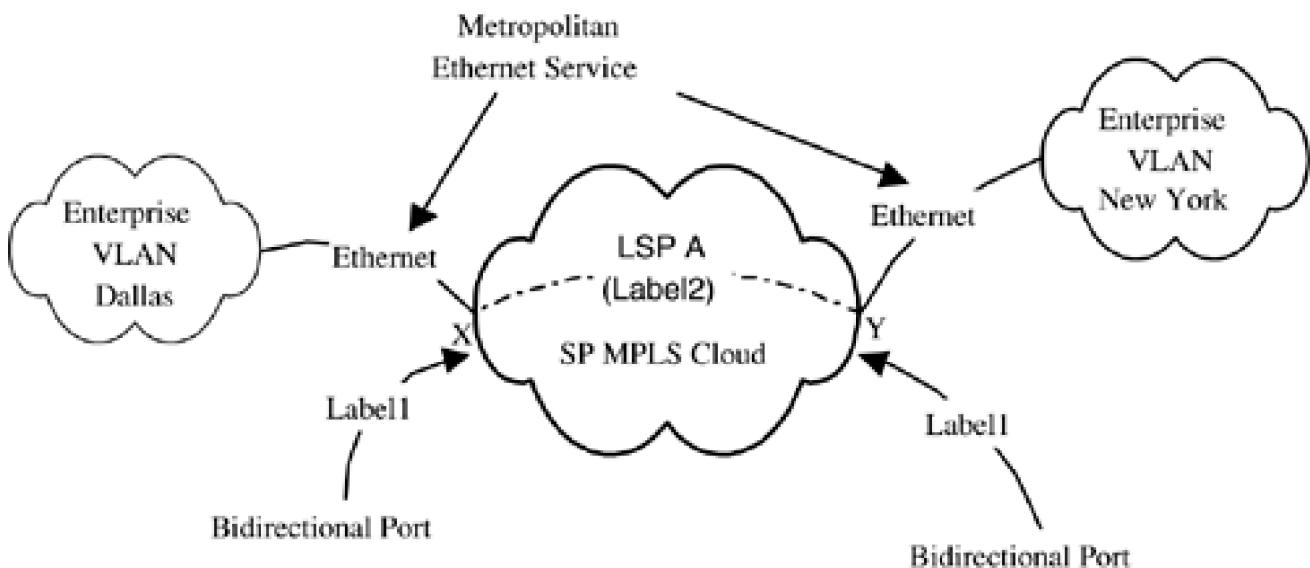
The backend chunk then assumes an identity of its own. It can be modified and independently tested, and as long as the interface does not change, the rest of the system need not change. Modifications to the backend can then be assigned to different developers, and if it is reasonably well-structured, it should be easy to learn. Chunks like this serve to de-mystify the NMS software codebase and impose a known structural pattern on it.

Components

The industry has embraced object orientation as a healthy move away from monolithic (procedural) systems development. Component development is a further refinement of this concept in which software products are made up of reusable objects. An important aspect of components is that they should model real-world objects as closely as possible, including relationships to other objects. To illustrate this concept, we now look at a multisite VLAN implemented using an intermediate SP ATM/MPLS cloud.

[Figure 4-1](#) illustrates an Ethernet service provided by a central site joining two branch offices of an enterprise. The two branch offices could be in the same VLAN or in two different VLANs. However, the WAN link (provided by the SP) allows for traffic to pass between the two sites as required. The links between these two sites are transparent to the end users. This type of arrangement is called Virtual Private LAN Service (VPLS).

Figure 4-1. Multisite Ethernet-to-MPLS interworking.



The bidirectional transfer of Ethernet through an SP core is sometimes called Transparent LAN Service (TLS) and allows for multisite, single broadcast domains. The central site would typically be an SP network but it could also be an enterprise headquarters. The connections between the enterprise sites in [Figure 4-1](#) can be Ethernet encapsulated over ATM.

We have created an LSP (called LSP A) in the SP cloud that connects the ingress and egress interfaces X and Y. The label that corresponds to this LSP is Label2. So, traffic at interface X that is MPLS-encapsulated with Label2 can then get to interface Y. However, we have a problem because interface Y may be shared by more than one client (i.e., sites other than the New York VLAN might also use interface Y). So, some means has to be found to determine where to send the traffic at interface Y. The TLS solution is to use a second

label (Label1) for this purpose. This means that traffic landing on interface X pushes both Label1 and Label2 during MPLS encapsulation. This gives us a label stack depth of two with the outermost label being Label2. The value of Label2 is then used to forward the traffic from interface X to interface Y, at which point Label2 is popped off the stack. We now have a label stack depth of one with Label1 as the outermost (and the only) label. The LER at interface Y pops Label1 and uses it to forward the traffic on to the New York VLAN site. One merit of using two labels like this is that it reduces the number of LSPs.

The VLAN traffic must be mapped to Label1, for example, by mapping the associated IEEE 802.1Q VLAN ID to Label1. The following components are needed for this model:

- A unidirectional port for the incoming Dallas VLAN traffic
- A mapping between an MPLS label and an IEEE 802.1Q VLAN ID
- A unidirectional port for the outgoing New York VLAN traffic

[Figure 4-1](#) illustrates only half of the story—that is, getting traffic from Dallas to New York. To get traffic sent in the opposite direction, we would need two more interfaces, another LSP with a VLAN ID mapping.

To manage this type of Ethernet interworking it is necessary to combine all of the above elements using an NMS. The overall managed object set is four ports, two outer labels, and two label-to-VLAN ID mappings. The components should be easy to combine, provision, and monitor so that the appropriate relationships are maintained between the two enterprise sites.

NMS also have other components that add value by being as loosely coupled as possible, such as:

- Scheduling facilities
- Virtual connection creation, modification, and deletion facilities
- NE firmware backup and restore features
- NE configuration database backup and restore, including network inventory details like port configuration settings, IP addresses, protocol settings, and virtual connections

Network operators often perform certain actions at specific times. If the NMS scheduling facilities are freestanding and not tied to one particular application feature (such as connection management or backup), then the user does not have to use external software (such as running UNIX cron jobs to schedule NE configuration database backups). Instead, the NMS can provide a scheduling function for all such operations. In other words, the user benefits because scheduling is implemented as a loosely coupled NMS component. This allows for executing the scheduled operations in a secure and controlled fashion (rather than using telnet). For operations that are repeated regularly and for which a permanent record is required, it is generally better to use the NMS.

Crossfunctional Cooperation

The complexity of NMS software development is such that many different people are generally involved in its production, including:

- Sales and marketing executives
- Specification and release planning experts
- Designers and domain experts
- Developers
- QA/Test personnel
- IT managers (or system administrators)—perhaps not often mentioned in the context of NMS development, IT is increasingly an essential component of keeping NEs operational (because of NE complexity)
- End users, through alpha and beta test sites, which can provide valuable initial end-user response to the system about

look-and-feel, function, and so on

The efforts of these groups can assist in improving the quality of the end product. GUI developers can combine their visual controls as early as possible (rather than waiting for the integration phase) in order to call into the backend or middleware layers. QA can also assist in the early stages of development by first testing paper models (based on design documents) and later testing stubbed software builds. These builds provide the skeleton of the end system with much of the function not yet implemented. This is in advance of receiving fully functional builds. An early cycle of tests can provide useful feedback and at the same time helps QA become conversant with the new software. Regular cycles of such releases help all parties (including developers) in a project become accustomed to new features. This is somewhat related to the release early, release often model [[OpenSourceWeb](#)] that has been an important part of the development of Linux. Another important partner in the development process is IT. The setup and maintenance of modern NEs increasingly require professional system administration skills. The contribution that IT can make to this is significant, particularly when many users are sharing the NEs. They can quickly resolve issues such as NEs suddenly failing because a signaling protocol has been inadvertently disabled. IT staff are well used to managing complex devices as part of their day-to-day work, and this skill can be leveraged by development.

Software developers can also assist crossfunctional cooperation by actively moving around the different areas of development and avoiding excessive specialization. As mentioned earlier, chunk-orientation in software structure helps to achieve this. Ownership and moving around the development environment need to be balanced against the needs of the developers (some will like to move around more than others). These areas include:

- GUI
- Backend
- Middleware
- Database
- Deployment facilities for software installation and upgrade

The last area mentioned is noteworthy because it is sometimes perceived to be the least glamorous role. Deployment software (such as those based on InstallShield or UNIX scripts) is the first thing the end user sees during installation of a given NMS. Traditionally, the bulk of project development effort goes into feature coding and testing, so a good implementation of deployment infrastructure is essential to the overall solution. Deployment is a very interesting and challenging area because it requires not only intimate knowledge of the product but also an excellent understanding of the host environments. Following are guidelines for successful deployment:

- Ease of use—the NMS should install/upgrade easily.
- Speed—deployment should be fast, minimizing downtime.
- Function—deployment should, if possible, not require any downtime as new code is applied.
- Auditable—it should be possible to verify that new code has been successfully (or unsuccessfully) deployed.
- Ease of removal—the NMS should uninstall cleanly and efficiently.

A short stay (of a month or two) in QA can also greatly assist developers in understanding the overall software development business processes. QA test plans can be written based on the content in software design documents—this is an interesting and useful perspective for developers to acquire. This last point is particularly important, since testing can amount to nearly half the total cost of a complex project.

Developers' moving between different areas helps spread knowledge and expertise around a team. This can then improve the accuracy of design and implementation because decisions are informed by broad experience. The cost of a given developer learning a new area in this way can be offset against the acquired expertise.

Task-based development is closely allied to solution engineering. It is most clearly explained in the context of either bug fixing or maintenance code changes. A given task is assigned to a developer who applies the necessary changes, tests, and then submits the changes into a configuration management system such as Telelogic CM Synergy. Many code changes can relate to just one item of work (or task), and this is why a task orientation is extremely helpful. Software tools, such as Telelogic CM Synergy, allow for a task to be treated as an object in its own right rather than as a collection of source code file changes. The task can then easily be imported into or quarantined from a given development code stream. This is a very powerful facility and provides a substantial advantage over traditional version control systems.

[Team LiB]

[PREVIOUS] [NEXT]

Smarter NMS

There is always scope for improving technical software products, and NMS are no exception. NMS must increasingly support high levels of

- Reliability
- Availability— failover for the entire system or just a critical component such as the database
- Maintainability—the software should be written to easily support future extensions

Much can be done to improve NMS so that they consume minimal NE resources:

- Preprocessing NE requests in order to reduce the number of messages sent to the network
- Discovering static NE data once and then rediscover only as necessary
- Minimizing the amount of data retrieved from NEs—every byte read from an NE uses up agent machine cycles

When the NMS user issues commands that result in network-bound SNMP messages, it may be advantageous to create batches. These are condensed SNMP messages that seek to minimize the overall number of **gets** and **sets** by aggregating MIB objects. So, rather than sending ten **getRequest** messages to one NE, it is more efficient to send one message with ten MIB object bindings. A specific NMS middleware layer would execute this function transparently.

When static data is discovered, there may be a large number of **gets** sent to the network. Rather than an expensive (and ongoing) rediscovery of all data, the MIBs should allow for the indication of modified objects. This allows the NMS to rediscover only data that has changed (similar to the way disk backup software can apply incremental backups of only files that have changed rather than of all files). This reduces the cost of maintaining parity between the network and the NMS.

Deterministic performance is another important NMS aspect. This requires a fully profiled codebase so that adding 100 new NEs to the user network adds a known load to the NMS. To a large extent it is almost impossible to know the exact resources required for supporting a given NE. Any device can suddenly suffer a failure (e.g., several port cards failing at the same time) and start to emit large numbers of notifications. Likewise (in the opposite direction), provisioning operations may start to time out because the NE/agent is heavily loaded, the network is congested, or timeout thresholds have been set too low.

Adding Services Management

Service management is an increasingly popular topic with vendors [[NMSGotham](#)], [[NMSHPOV](#)]. In this the service offered to the user is managed rather than the operator being concerned with just the underlying connections in its network domains (access, distribution, and core). The NMS offers a higher-level service management capability, such as layer 2 (e.g., Ethernet) over an MPLS backbone. The rationale is that users want services rather than technology like ATM connections (this is closely allied to the above-mentioned solution orientation). So, it makes sense for NMS to be able to deal in terms of services rather than just connections and devices. Service management introduces a new paradigm that is not restricted to a single connection. A service can be made up of more than one connection and even more than one type of cross-connection object. For this reason, service management generally requires a new type of managed object for:

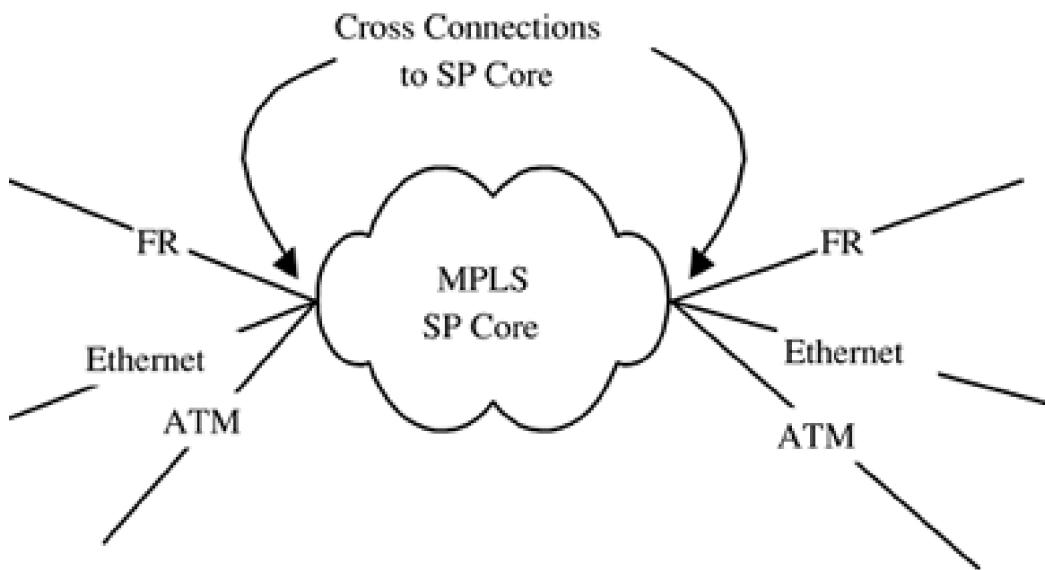
- Visual representation
- Provisioning

- Monitoring
- Auto-learning

An interesting aspect of MPLS is that it will allow enterprise network cores to become more generic. Regardless of traffic type (layer 2 or 3), it will be transported over LSPs/tunnels. A more generic core allows for more easily offering different service levels.

Multiservice switches help facilitate this type of arrangement by allowing easy cross-connection of technologies such as Ethernet, FR, and ATM into an SP core. This is illustrated in [Figure 4-2](#) with a range of access link types terminating on the edge of an SP network. The various links terminate at multiservice switch ports and are then cross-connected into the core.

Figure 4-2. SP core network with cross connections.



Service management still involves connections in the underlying network, but this is deliberately obscured (or abstracted away) by the NMS in order to simplify the network picture. A number of issues arise if scalable service management is to be offered because NEs must provide:

- Unified signaling across multiple domains
- Service-level traps

Implementing services is difficult in a network comprised of several different domains, such as ATM, IP, MPLS, Frame Relay, and X.25. Connections, such as ATM/Frame Relay PVCs, can be manually joined together as they are created (by first creating an ATM PVC on port X and then an associated Frame Relay interworking). However, automatically creating end-to-end, cross-domain connections requires special signaling protocols in the NEs. These protocols would allow the creation of multidomain, managed service connections. This becomes a more pressing problem as MPLS starts to be deployed in network cores (along with ATM and SONET/SDH) because more and more protocols will then move to the edge of the network and have to be interworked or cross-connected. The emerging standard Pseudo Wire Emulation Edge-to-Edge (PWE3) ([IETF-PWE3](#)) will allow generic layer 2 emulation across MPLS cores. In time, this may well assist service management.

In advance of service management infrastructure such as multidomain signaling and PWE3, existing NMS can still offer a limited and fairly scalable form of service management. This consists of discovery and provisioning of just the first leg of a cross-connected service. This is just the head end of the service, which in [Figure 4-2](#) is the leftmost cross-connection. If the head-end host node maintains the operational status of the overall service, then this can be monitored by the NMS. However, the individual downstream segments of the service are not monitored for status changes in the interests of scalability (there may be many downstream segments, so the NMS essentially ignores them and takes its cue from the head end only).

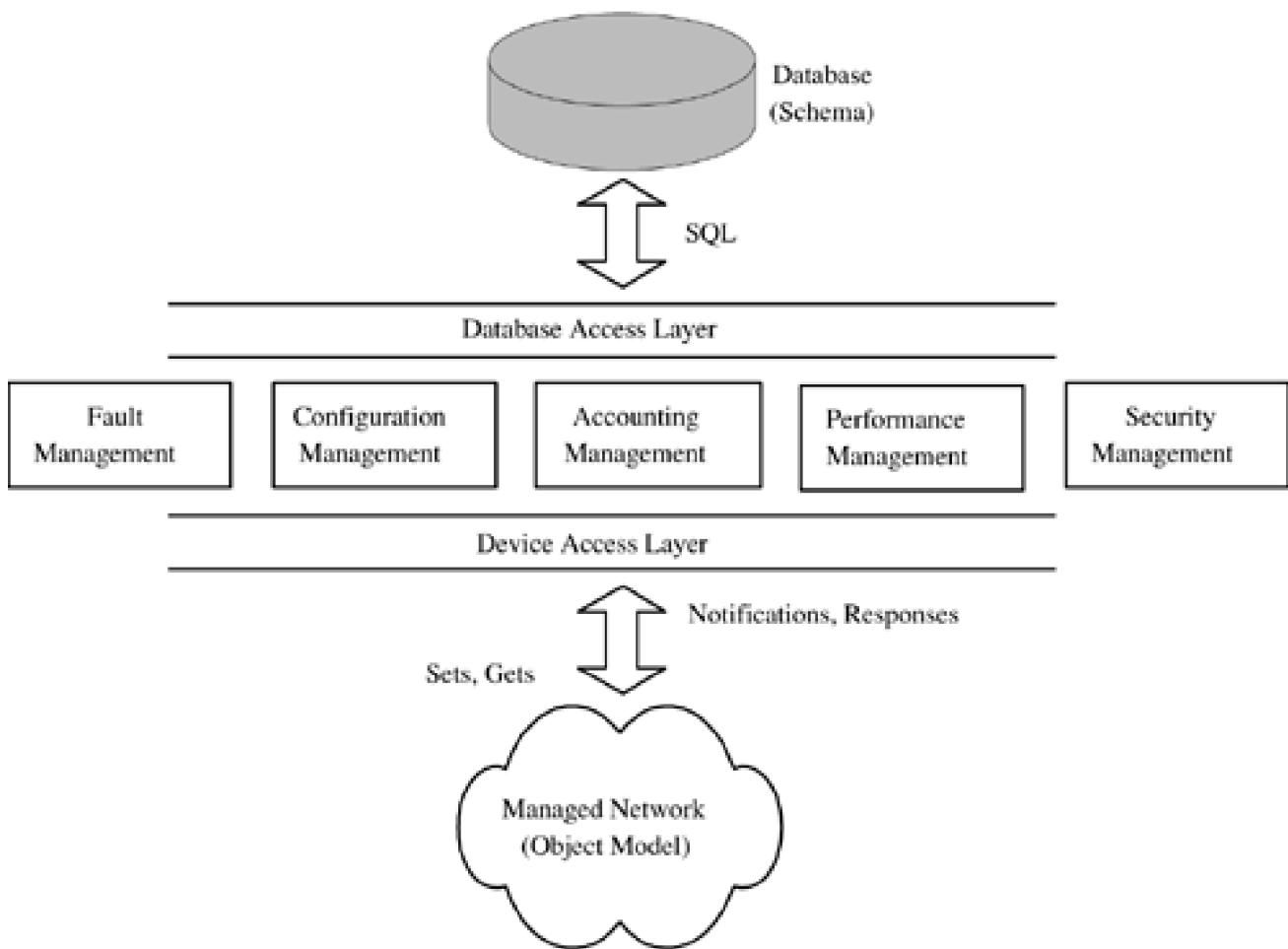
Once the network becomes service-aware (with multidomain signaling, PWE3, etc.), its NEs can then emit service-level traps. In [Figure 4-2](#), a multiservice connection starts as Frame Relay, crosses an MPLS core, and exits as Frame Relay again. If a node or link in the path of this connection fails, then only one trap is needed by the NMS. The head-end node at the point of origination of the Frame Relay service can emit this trap. It is not absolutely necessary for all affected nodes to generate traps for this. Reducing the number of traps improves

scalability, particularly if the traps accurately describe the problem and its location.

NMS Structure

Most NMS are vertical applications. Generally implemented in what is often called a *stovepipe* fashion, data flows are up and down the paths illustrated in [Figure 4-3](#).

Figure 4-3. NMS stovepipes.



Each application in [Figure 4-3](#) tends to be distinct, sharing some data (such as node details) about the network but essentially being standalone. There is little, if any, horizontal communication between the stovepipes because they are both database- and NE-centric and fulfill specific FCAPS functions.

[Figure 4-3](#) illustrates the baseline FCAPS structure. Real NMS tend to have additional software for facilities, such as

- Topology management supporting multiple clients and fault-based coloring (explained below)
- NE firmware backup and restore
- NE configuration database backup and restore

The version of firmware running on a given NE is an important piece of information. As new features (or bug fixes to existing ones) are required in the network, it frequently is necessary to distribute new firmware versions to the NEs. An NMS can provide this facility as well as back up the existing version (in case the new code has problems, e.g., insufficient NE RAM/flash, software bugs) and store it in a safe

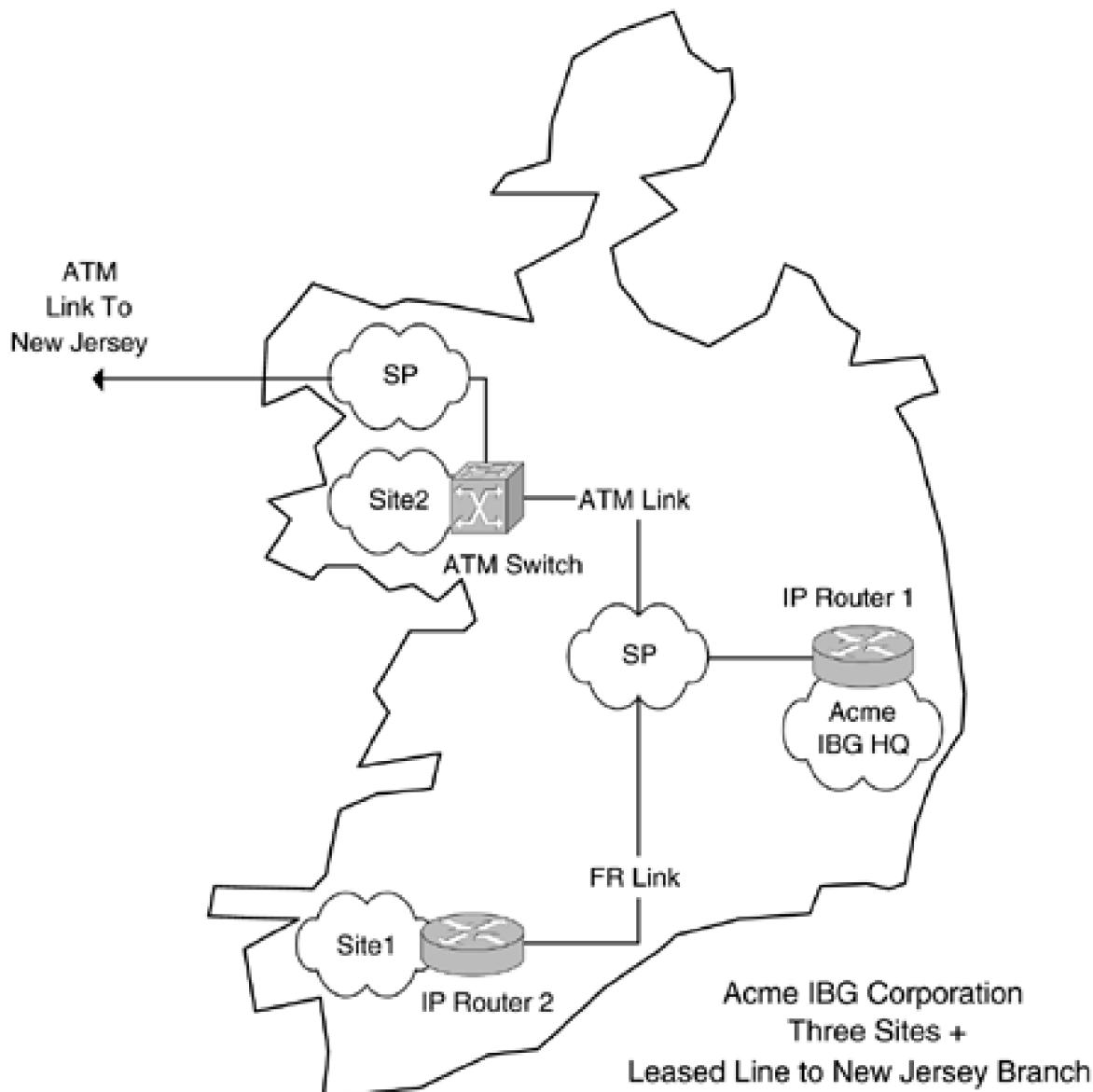
location. The topology data can then be updated to indicate the new firmware version. The NMS can also distribute firmware to more than one NE simultaneously if required (this can reduce network/NE downtime).

NE configuration data is also vital because it dictates how the features operate in the network. Many NMS provide the ability to backup and restore configuration data.

Topology management can take the form of supporting a pictorial representation of the network. Often, this is implemented against a geographical background showing the location of NEs relative to their physical location. This is illustrated in [Figure 4-4](#) for a hypothetical three-site enterprise called Acme IBG Corporation located in Ireland. This enterprise has a central site (HQ), and WAN connections go from it to the branch offices. The intersite links are SP-owned. Any of the nodes, links, and clouds in [Figure 4-4](#) can become faulted:

- Nodes can go up (operational) as well as down (nonoperational)
- Interfaces on either end of a link can go down
- Network faults can occur inside the site network clouds

Figure 4-4. Network topology with geographical mapping.



Faults can be visually indicated by means of changing the color of some GUI object. Topology components may contain subordinate objects. For example,

- Nodes contain interfaces
- Interfaces carry virtual connections
- Links join together adjacent interfaces

Above the level of abstraction associated with a node are clouds. Clouds can contain any number of subordinate network objects (including other clouds). An important consideration for reflecting network status in a topology application is the way in which faults are visually rendered. If a topology application is displaying just network clouds—that is, the topmost level of the hierarchy—then a node fault (notification) in one of the clouds should be visible. Normally, the fault results in a color change in its container cloud. As usual, the quality of the overall NMS dictates the speed with which the fault is registered. The user should be able to select (using a mouse or other pointing device) the faulted cloud and drill down to see the exact location of the problem. Some NMS may employ additional methods of announcing faults, for example, by sending mobile telephony short text messages, pager alerts, or email to operators.

Usually, an NMS deployed inside one of the clouds monitors and controls the NEs in [Figure 4-4](#). [Chapter 5](#), "A Real NMS," examines the components of a commercial NMS.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

Smarter MIBs

MIBs represent a shared name space between SNMP agents and managers. They allow an operator to leverage the management facilities deployed in the network. Because of their central role in network management, MIBs should be as well written as possible. The following MIB notes provide some guidelines.

Our MIB notes are primarily intended to promote NE manageability (one of the four M's mentioned in the preface). High levels of NE manageability offer a number of benefits:

- The NE is easier (and cheaper) to incorporate into a network and an NMS.
- The specific NE features are easier to access and manage.
- End-user confidence levels are raised with regard to the vendor.
- The NE acquires a degree of product differentiation.

The enterprise networking market is highly competitive with tight margins. If a given NE is easy to install, configure, and operate in an NMS environment, then that is a considerable advantage for the vendor. If the NE feature set (e.g., MPLS, Frame Relay) is easy to access (via both the EMS and SNMPv3), then the cost of integration and ownership are reduced. Happy end users are likely to upgrade or purchase new equipment. All of these are good things from the vendor's perspective, particularly in an industry where NEs are increasingly similar. We now describe some aspects of MIBs that improve manageability.

MIB Note: Avoid MIB Object Semantic Dependencies

A surprisingly difficult thing to do in SNMP is create new rows in MIB tables. The problem lies in having to understand the semantic relationships between the table columns. If a table (such as the MPLS tunnel table from [Chapter 3](#)) has, say, 37 columns, then not all columns have to be set in order to create valid entries. Loose coupling between columns is a good thing because the user knowledge burden is lower. A good way of coupling together blocks of related MIB objects is to provide multiple tables (similar to relational database normalization). This is particularly so when the tables can be reused (e.g., as in the MPLS tunnel hop table for EROs and the MPLS tunnel resource table for bandwidth reservations. These two tables can be reused by many tunnels, so by not including their objects in the tunnel table, the overall data set is less redundant).

Additional tables can then be linked into the parent table using integer indexes. The external tables can then be shared (e.g., more than one tunnel instance can share the same ERO).

Where intercolumn relationships do exist, this should be indicated clearly in the MIB using comments.

MIB table column semantic dependencies complicate provisioning code because the NMS software has to understand the columnar relationships. Another problem is that they also complicate the database schema when the columnar relationships have to be duplicated. The latter point illustrates the problem of modeling the NE MIBs in the NMS. Since the NMS must track the state of the NEs, it has to store NE data usually in a relational database. It is in the schema of the latter that it is often required to duplicate the MIB data. Intercolumn MIB dependencies make for a more complex database schema.

MIB Note: Provide Default MIB Object Values

Providing default values for all MIB table objects is useful for facilitating a thin provisioning code layer. In other words, the caller provides specific values for the required MIB table columns and relies on defaults for the rest. When the NMS request arrives at the provisioning code layer, all the data can be simply written to the associated NE MIB objects with little need for checking. In other words, default values help simplify network-facing code.

Default values can be given to extraneous MIB objects (column objects that are not needed in a given SNMP operation), for example, `mplsTunnellIncludeAffinity` in the MPLS tunnel table. This object is used when creating a tunnel in which the user wants to constrain the signaling path through an MPLS cloud so that it uses only a specific type of interface on each hop. This type of tunnel may reflect a higher (or lower) class of service reserved for certain end users—in short, much of the time, this type of tunnel may not be required, so the `mplsTunnellIncludeAffinity` object will, in many cases, not be set. Providing a default value for this and similar objects can help to prevent agent exceptions caused by the insertion of bad values; these can occur if the sender inadvertently uses an invalid value for a given column but the agent tries to interpret the value anyway and generates an exception. This guideline indicates that only mandatory columns are set; the rest are given safe, default values.

To illustrate this, the allowed values of `mplsTunnellIncludeAffinity` are integer bitmasks; each bitmask represents an interface color code value, for example, 0x00001 for gold, 0x00010 for silver, and 0x00100 for bronze. The network operator must configure these colors on all NEs where color selection will be used. In [Chapter 3, Figure 3-3](#), the nodes LER A, LSR D, LSR E, LSR F, and LER B could be configured to support the colors silver and bronze on their ingress interfaces. Then a tunnel could be created with a path constrained to use only interfaces with the colors silver and bronze by setting `mplsTunnellIncludeAffinity` to 0x00110. The tunnel so created would follow the path described. This is another example of traffic engineering, where the path is reserved in advance by special actions on the part of the network operator.

A sensible default value for `mplsTunnellIncludeAffinity` (and its associated objects) could be 0 to indicate non-use of this MIB object. Since the default provisioning value has been chosen to match the MIB default value, there is no need to validate before updating the MIB. This reduces the size of the provisioning code layer. New MIBs should always be written to include default values for all objects.

MIB Note: Centralize MIBs to Match NE Features

A good example of a centralized feature is the MPLS tunnel table. Most of the tunnel-related content is located in this table. This avoids the need for expensive MIB walks in order to manage tunnels. The single table allows for tunnels to be created, modified, monitored, and deleted—in other words, full management using just a few interrelated MIB tables. This helps to keep provisioning code (and connection discovery code) simple and avoids the need for the simultaneous loading of multiple MIB files in the NMS.

This is similar to solution engineering considerations. MIBs should be structured so that solution components are clearly expressed and easily accessed. Again, the MPLS tunnel table is a good example because it illustrates the main MPLS managed objects (such as tunnels, EROs, and resource blocks) in a fairly clear and concise manner.

Issues like the above MIB notes provide good reason for the developers of NMS maintaining the MIBs to work in close cooperation with the NE developers. In other words, authority and responsibility for MIBs should be shared between the main interested engineering parties. Poor-quality MIBs can result in unnecessarily complex NMS software and possibly even NEs that are difficult if not impossible to properly manage.

One Data Model

The data stored and maintained in the managed network must, at some point, be imported (in whole or in part) into the NMS and stored in some type of persistent repository. Normally, this is a relational (or even an object-oriented) database. The schema for this repository represents the NMS data model. Repository data is manipulated by the NMS and, for actions such as provisioning, is written to the network as MIB object instance values. Similarly, traps received from the network must be processed and stored for reporting purposes. The data model is the glue for bringing together the managed NE data and the user's view of the network. In a sense, NEs implement a type of distributed database in their MIBs. It is this database that the NMS tracks and modifies. Extracting data from NEs is achieved using SNMP **gets** and traps. Similarly, pushing data into NEs is achieved using SNMP**sets**. These three message types all consume precious NE and network resources. So, maintaining parity between an NMS and its managed network is fundamentally limited by:

- Network size and bandwidth
- NE density—the number of managed objects (connections, interfaces, protocols, etc.)
- NE agent resources (speed, allocated CPU cycles, and I/O)

The NMS must try to maintain data parity and, at the same time, minimize NE access.

The database model (or schema) should be a superset of the MIB. All the applications in [Figure 4-3](#) could benefit from the deployment of a single data model. This applies particularly to the bidirectional applications that both read from and write to the network. The single data model allows for flow-through to and from the user if MIBs are simple. This helps to keep the device-access layer thin and fast because all (or almost all, if access is needed to an object like `mplsTunnelIndexNext`, mentioned in [Chapter 3](#)) the required data for NE write operations can be gathered from the database and written to the network. There is then no need for intermediate processing at the device-access layer.

One issue with the stovepipe structure is that the FCAPS applications are written to share a single data repository hosted on one server. This can give rise to database contention between the applications and possibly even deadlocks where multiple applications update the same tables. Careful code and data design is needed to avoid this. Another problem closely allied to this is host resource contention between the applications. This occurs when the applications are written to run essentially independently of each other on a single host machine. The result can be an unbalanced host machine with high levels of CPU and disk activity. This can be improved using either some scheme for interapplication cooperative multitasking (rather than depending on the host operating system) or by distribution (discussed in the next section).

Distributed Servers and Clients

NMS are increasingly large, complex application suites. Rather than using a single server host with multiple distributed clients, more than one server machine can be used. This helps to distribute the processing among a number of host machines. The availability (in standard programming languages) and function of technology such as RPC, Java RMI, and middleware products based on CORBA considerably ease the task of integrating multiple NMS application hosts. The applications shown in [Figure 4-3](#) could therefore be deployed on different machines; this reflects the fact that computing power is increasingly inexpensive. This approach can help to offload some processing from a single host, but it may increase the interserver network traffic. This tradeoff between saving host resources and consuming network bandwidth is a problem typical of networked applications in general. Clients can also be distributed, accessing the NMS via standard, desktop Web browsers.

NMS can also be operated in redundant mode. This consists of deploying a primary server with one or more backup servers. Failure of the primary results in a switchover (or failover) to a secondary server. This allows for the entire NMS to be backed up in a number of configurations:

- Hot standby: The secondary takes over with no data loss.
- Warm standby: The secondary takes over with some data loss.
- Cold standby: The secondary is started up and switched into service.

Hot standby is used for critical systems that require 99.999% (the five 9s) uptime. A good example of this is an SS7 protocol stack used for signaling in a mobile (or fixed) telephony network. Two copies of the SS7 stack run in parallel, but only one of them writes to the database and network. If the primary system fails, then the standby takes over. This primary and secondary configuration often provides a convenient means for applying software upgrades. When the operator wants to upgrade both primary and secondary, the primary is stopped, which causes a changeover to the secondary. The primary system software is then upgraded and started up (to back up the secondary). Then the secondary is stopped, causing a switch back to the original primary. Then the secondary software can be updated. Some DBMS vendors, such as Oracle and Informix, also provide standby support in their products.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Smarter NEs

The scalability issues associated with NEs discussed in [Chapter 3](#) require more agent intelligence. This is because NEs are becoming denser, essentially compressing what have been entire networks into firmware. When a device can host millions of virtual connections, the issue of trap generation becomes problematic because a device or high-capacity link failure (such as a fiber cut) can result in a storm of traps. Rather than many NEs generating traps in parallel, one dense device may not generate as many traps. This is a limitation of the processing speed and computational resources allocated to an agent on one of the new high-end devices. However, in time it is likely that the number of traps emitted by the next generation of NEs will probably exceed that of an equivalent set of low-end devices. Trap storms can cause network and NMS congestion (blocking). There is therefore a need for traps to be aggregated, buffered, and possibly even compressed. For this reason, denser NEs require more preprocessing at the agent level before emitting traps to the NMS. In the opposite direction, the provisioning operations initiated by the NMS must increasingly push more complex settings onto the NEs in order to support the increasingly advanced network services.

The need for advanced, real-time services, such as voice- and video-over-IP, on enterprise and SP networks is also resulting in a need for greater NE intelligence. Policies provide an elegant solution to the problem of managing ever-denser NEs. We discuss policies in detail in the next section. For now, they are introduced simply as being objects (rules, conditions, and actions; e.g., if the number of IP packets received at a router interface exceeds 10,000 per second, then increase the bandwidth of an associated virtual circuit) that are pushed onto NEs from an NMS. The NEs then subsequently:

- Follow the installed policy guidelines
- Watch for the indicated conditions
- Execute the required actions

Policies are a little like SNMP notifications in that the NE performs work independently of the NMS. They differ from notifications in that policies can result in quite complex NE computation (such as real-time traffic engineering). In effect, this brings management code much closer to the managed object data (providing a partial solution to one of the big management problems mentioned in [Chapter 3](#)).

[Team LiB]

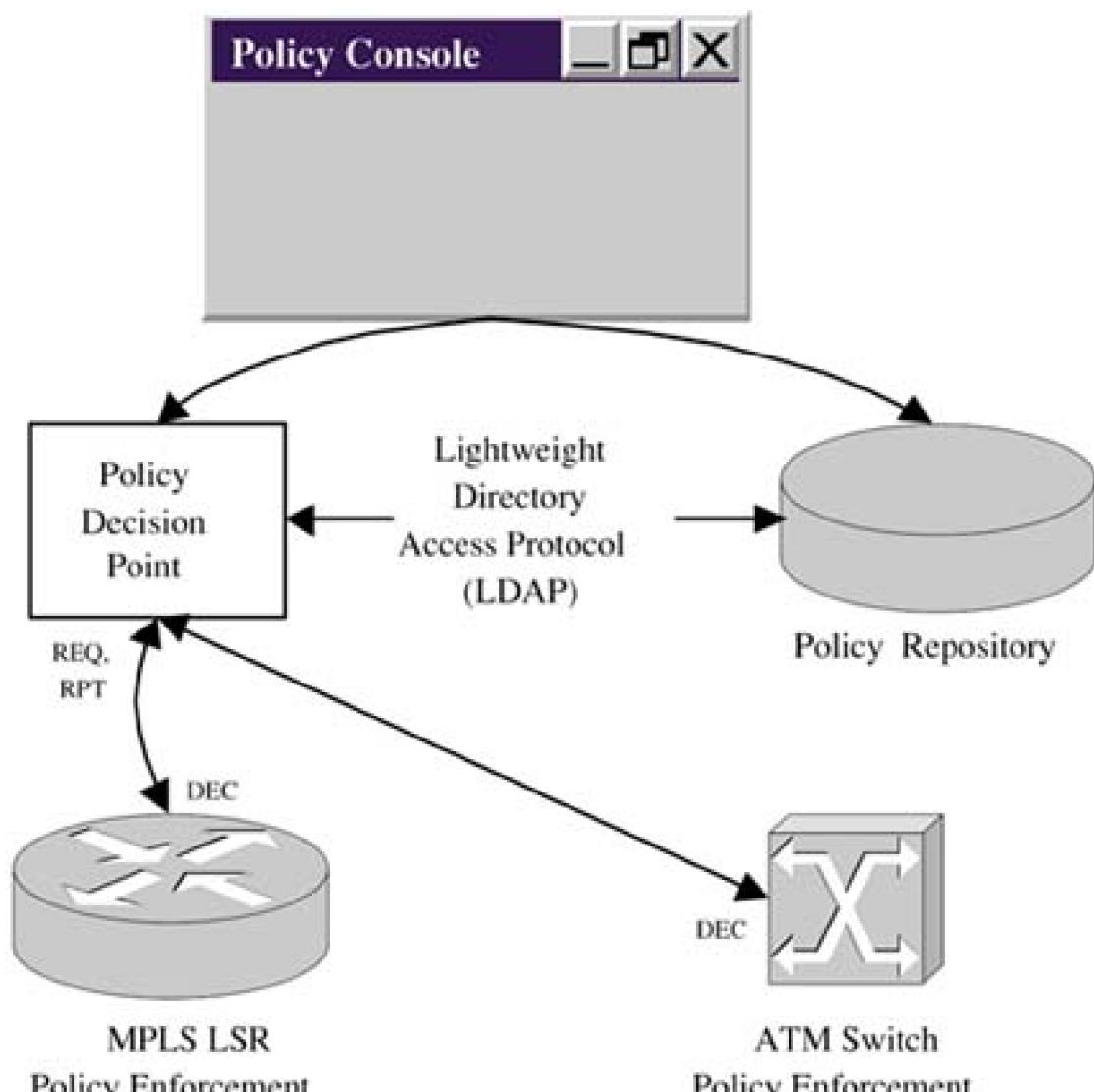
◀ PREVIOUS ▶ NEXT ▶

Policy-Based Network Management (PBNM)

PBNM is one of the most important directions being taken in network management. It recognizes that trying to manage individual devices and connections using a simple get/set/notification model is no longer sufficient because of the demands increasingly being placed on networks. Devices must become a lot more self-reliant in allowing the network operator to divide up and control the underlying resources in a deterministic fashion. PBNM introduces a number of new and interesting entities into network management, as shown in [Figure 4-5](#), which illustrates the main PBNM architectural elements:

- Policy console
- Policy repository
- Policy server/decision point (PDP)
- Policy enforcement point (PEP)

Figure 4-5. PBNM architecture.



Policy Point (PEP)

Policy consoles are employed to manage user-generated policies. The user creates, deletes, and modifies policies, and these are saved into the repository. The PDP or policy server is responsible for pushing (or installing) policies onto the various NEs. PEPs are NEs (such as IP routers, switches, or MPLS nodes) that execute policies against network resources like IP traffic. Policies can be installed by the PDP without any prompting from PEPs; alternatively, PEPs may initiate requests to the PDP to download device-specific policies; for example, if the PEP is an MPLS node, then it can download traffic engineering policies. The architecture in [Figure 4-5](#) is flexible enough to support both modes of operation. The PDP retrieves policies from the repository using the Lightweight Directory Access Protocol (LDAP). COPS-PR is the protocol used to move policy configuration from a PDP to a PEP. A simple protocol is used for policy manipulation by PEPs consisting of these messages: **REQ**(uest), **DEC**(ision), and **RPT**(report)—as illustrated in [Figure 4-5](#).

The PBNM elements in [Figure 4-5](#) form an adjunct to (not a replacement for) the NMS we have discussed so far. Policies installed on NEs provide a very fine-grained control mechanism for network operators.

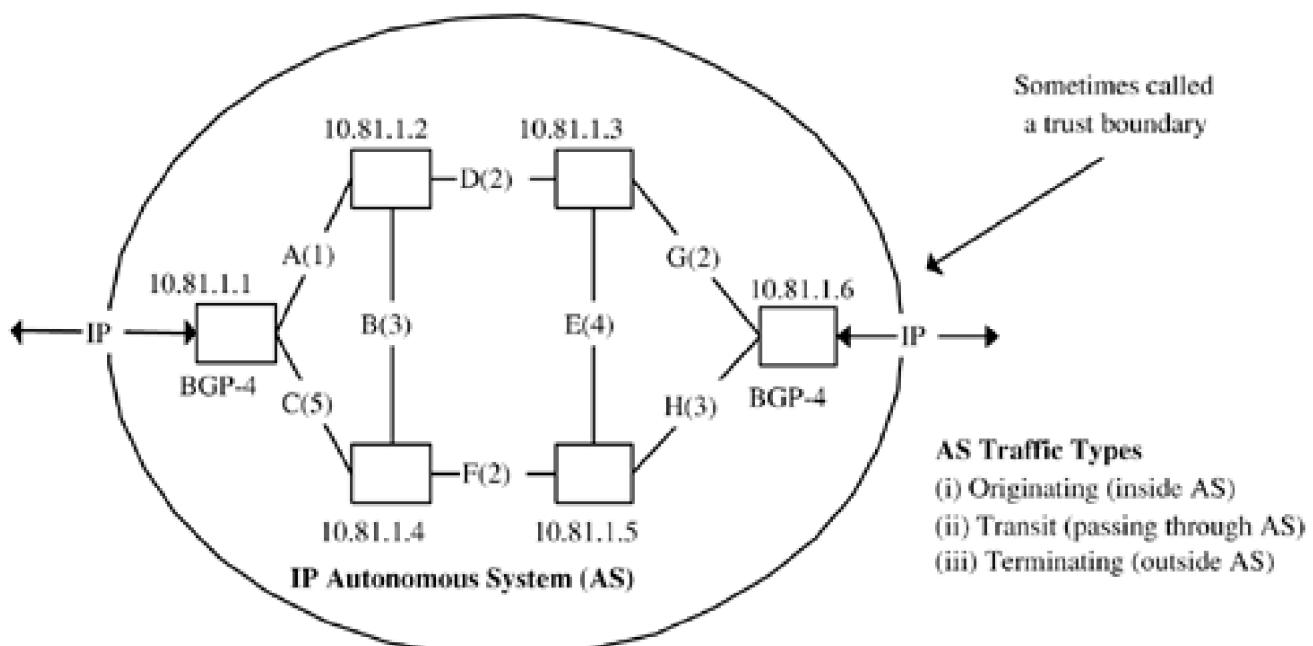
What Is a Policy?—Pushing Intelligence into the Network

Policies add intelligence to NEs; in effect, the network becomes almost like a computer, performing advanced functions with no further prompting needed from an external NMS. Policies are simply rules that contain essentially two components:

- A trigger or condition in the network
- An action to take when the condition occurs

Policies are in widespread use in computing. A simple example is that of IP router table control. A network of IP routers is a dynamic entity because nodes and links in the network can go up and down, potentially resulting in changes to the paths taken by traffic. All of the routers try to maintain a current picture of the network topology—similar to the way an NMS tries to maintain a picture of its managed objects. [Figure 4-6](#) illustrates an autonomous system (AS) comprised of four interior (intra-AS) routers and two exterior (inter-AS) routers. A real AS could contain hundreds or thousands of nodes.

Figure 4-6. An IP autonomous system.



The interrouter links in [Figure 4-6](#) are A, B, C, D, E, F, G, and H respectively, with administrative weights shown in brackets. Each router records those IP addresses reachable from itself in an optimal (in this case, least-cost) fashion; for example, the cheapest route from

10.81.1.2 to 10.81.1.6 is via links D and G respectively. [Table 4-1](#) illustrates a conceptual extract from the routing table on 10.81.1.2. The clever way routers manage traffic is that they simply push it to the next hop along the path. In this case, 10.81.1.2 has a packet destined for 10.81.1.6, so the packet is sent to 10.81.1.3. Once this occurs, 10.81.1.2 forgets about the packet and moves on to its next packet. When 10.81.1.3 receives the packet, it also pushes the packet to the next hop indicated in its routing table (in this case the destination 10.81.1.6). By operating in this way, IP routers distribute the intelligence required to move packets to their destinations.

Table 4-1. IP Routing Table for Router 10.81.1.2

DESTINATION	NEXT HOP	ADMINISTRATIVE WEIGHT
10.81.1.6	10.81.1.3	4

The next hop for a packet, destined for 10.81.1.6, at router 10.81.1.2 is illustrated in [Table 4-1](#) as 10.81.1.3. The administrative weight (or cost) of getting a packet from 10.81.1.2 to 10.81.1.6 is the sum of the intermediate link weights; this is the sum of link weights for D and G, or 4.

When a change occurs in the network topology, the routers detect this and initiate a process called convergence. If link D in [Figure 4-6](#) fails, then the shortest path to 10.81.1.6 (from 10.81.1.2) is recalculated to yield B-F-H. [Table 4-1](#) would then be updated so that the next hop in the direction of 10.81.1.6 is 10.81.1.4 with a cost of 8. It is actually the interface that leads to 10.81.1.6. A number of steps precede this:

- 10.81.1.2 retires the original route to 10.81.1.6 once it discovers it to be invalid.
- 10.81.1.4 passes (advertisises) the new route on to 10.81.1.2.
- 10.81.1.2 consults its import policy to decide if the new route to 10.81.1.6 is to be accepted.

This is the way in which the routers update their picture of the network. Routing information exchanges can also be encrypted, though many core Internet routers do not employ this facility, which leaves them open to attacks [CERTWeb](#). The Routing Policy Specification Language [\[RFC2622\]](#) provides a specification for this.

Our discussion of IP routing has left out some important details. [Figure 4-7](#) illustrates an extract from the route table for a Windows 2000 host with the IP address 10.82.211.29. To see this on a Windows machine, just open a DOS command prompt and type `netstat -r`.

Figure 4-7 Host routing table.

```
C:\>netstat -r

Route Table
=====
Active Routes:
Network Destination Netmask     Gateway       Interface     Metric
0.0.0.0      0.0.0.0      10.82.211.1  10.82.211.29    1
127.0.0.0     255.0.0.0    127.0.0.1    127.0.0.1      1
10.82.211.0   255.255.255.0 10.82.211.29  10.82.211.29    1
10.82.211.29  255.255.255.255 127.0.0.1  127.0.0.1      1
10.82.255.255 255.255.255.255 10.82.211.29  10.82.211.29    1

Default Gateway: 10.82.211.1
=====
```

[Figure 4-7](#) illustrates a number of interesting features:

- This is a host router table, so this machine does not support transit packets; that is, it is not a router (rather, it is a client in the routing scheme).
- The Network Destination column indicates the IP destinations reachable from this host.

- The Netmask column is used to isolate the network number in the IP address (IP addresses are made up of the combination of network number and node number, so by multiplying the IP address by the netmask, we isolate the network number).
- The gateway indicates the IP address of the next handler for a packet, with the address shown in the Network Destination column.
- The Interface column indicates the outgoing interface on the host machine that will be used as the next hop for a packet with this network destination.
- The Metric column indicates the cost of each of the operations.
- The Default Gateway is the IP address used for all packets that do not match any entries in the Network Destination column.

From all this we can see that a route table is just a set of rules for how to handle IP packets with specific destination addresses. This is the context we use for explaining policy^[2] here.

^[2] BGP4 is an example of an EGP that supports policies in relation to exporting routes from itself to an IGP (such as OSPF or IS-IS). However, we limit our discussion to describing the simple rules that are followed in basic IP routing. For this reason, BGP4 policies are outside the scope of this discussion.

To illustrate the way IP routing works, let's try using the ping command, `ping 10.82.211.29`, to produce the listing illustrated in [Figure 4-8](#). Ping is an application layer entity that sends an ICMP (another protocol in the TCP/IP suite) request packet to the specified host. Ping is very useful for determining if a given node is IP-reachable because when the ICMP request is received, a host must send a response similar to that in the listing. Let's now trace the message exchange.

Figure 4-8 ICMP request and reply messages.

C:\>ping 10.82.211.29

Pinging 10.82.211.29 with 32 bytes of data:

```
Reply from 10.82.211.29: bytes=32 time<10ms TTL=128
```

Ping statistics for 10.82.211.29:

 Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

 Approximate round trip times in milli-seconds:

 Minimum = 0ms, Maximum = 0ms, Average = 0ms

When the ICMP request packet arrives at the host (10.82.211.29), the route table is searched for the longest match. This is row four in [Figure 4-7](#), shown in bold. The packet is relayed to the loopback interface 127.0.0.1—this has the effect of sending the packet back to the local host (we will see the loopback interface again in [Chapter 7](#), "Rudimentary NMS Software Components," [Figure 7-9](#)). The local host then responds to the ping request with four packets.

Other interesting information on the various TCP/IP protocols can be gleaned using related DOS commands such as `arp` and `tracert`. [Appendix B](#), "Some Simple IP Routing Experiments," has some details.

Network Management Policies

Network management policies can be simple resource allocations such as the following:

- Give traffic from IP address a.b.c.d the highest priority forwarding treatment.
- Assign email traffic the lowest priority forwarding treatment.
- Assign VoIP traffic assured (guaranteed) forwarding treatment.

Other network management policies can be in the form of NE configuration information, such as

- Protocols—signaling and routing
- Interfaces
- Networkwide settings

Each of these ultimately translates into some action or set of actions at an NE. Policies allow a far more abstract view of the network than is possible with NMS that simply allocate virtual connections. With abstraction comes increased operational power because it is then possible to express complex rules, such as those above, and to combine these into business objects such as service level agreements. This is a type of fly-by-wire way of running a network, and it is entirely possible for policies to conflict—for example, two policies that attempt to allocate the same bandwidth twice or erroneously assign an unsupported forwarding behavior. The power of PBNM must be used with caution.

High-level policies, such as those above, will generally be reflected in some type of NE configuration changes:

- Priority settings for a given IP traffic stream
- Queue settings for a given IP traffic stream
- Security levels
- Access control lists

Communication of policies to PEPs can be made using SNMP, COPS, telnet, and so on. Once the policies have been installed in the PEP, they can then be locally executed when the associated conditions have been met.

The Common Open Policy Service Protocol (COPS)

COPS [RFC2748] is a simple client/server protocol for the communication of policies between policy clients and servers. A policy client is also called a PEP, and a policy server is also called a PDP. PEPs may request policy provisioning configuration data (using **REQ** messages) from the PDP. (PDPs may also autonomously send policies.) The PDP replies with **DEC(ision)** messages, as shown in [Figure 4-5](#). These exchanges are made using TCP.

In [Figure 4-5](#) we see NEs that are policy-enabled; that is, they can directly host COPS policies. This need not be the case. Policies can be expressed in any NE technology; the standard is COPS-based, but even SNMP tables can be used if required (as we'll see below). This is particularly the case if existing NE products already use SNMP.

SNMP uses SMIv2 for the following:

1. Module definitions
2. Object definitions
3. Notification definitions
4. Textual convention definitions
5. Conformance definitions

The COPS equivalent to Structure of Management Information (SMI) is called Structure of Policy Provisioning Information (SPPI) and is based on SMI. Apart from notifications, SPPI uses all items in the list above to create Policy Information Bases (PIBs) [[RFC3084](#)]. PIBs are very similar to MIBs: They specify a name space that is common to both the PDP and PEP (MIBs specify a name space common to both managers and agents). A PIB is a tree-structured name space containing provisioning classes and provisioning instances. PIB module definitions [[RFC3159](#)] provide a starting point for PBNM. Because PIBs and MIBs are based on SMI, it is relatively easy to convert a PIB into a MIB. An extension to COPS—called COPS-PR—is used to configure policies on NEs.

Network Processors

Network communication is increasingly a problem not of bandwidth but of managing mixed traffic types. Network management is driven by the need to be able to manipulate network resources so that all traffic types receive the required forwarding treatment. This is assisted by the development of network processor products from organizations such as Intel and IBM. These are network-facing devices that support advanced features such as the following:

- Classifiers identify a packet stream and steer it to a traffic conditioner for special treatment.
- Meters measure the timeliness of packets against an agreed arrival rate profile.
- Markers set the Differentiated Services Code Point (DSCP, described in the previous chapter and also at the end of this chapter) of packets.
- Queue mappings move a packet stream into a specific hardware queue for special forwarding treatment.
- Shapers introduce some delay in a packet stream in order to make it comply with a profile. Delay is applied using buffers; if the buffers fill up, then packets may be dropped if they are still out of profile.
- MPLS labelers apply MPLS labels to packets in advance of pushing them into an appropriate LSP/tunnel.
- PIBs are used for storing policies written by the operator.
- SPPI is the language in which policies are expressed.

Network processors form an important part of the migration towards the use of policy. PBNM systems can define networkwide policies for connection management, QoS, and traffic engineering, and then push these onto all relevant devices in the network (including network processors). Typically, devices such as routers and switches can be equipped with the new-generation network processors.

Directory-Enabled Networking (DEN)

One of the most time-consuming aspects of NMS development is producing the information model. This is the way in which the managed objects, such as nodes, interfaces, links, virtual connections, network clouds, routes, and resource blocks (bandwidth and other traffic parameters), are represented inside the NMS. Rather than reinventing the wheel every time a new set of managed objects is needed (as occurs when an NE feature set is extended), another approach is to use an existing standard information model. DEN provides this and is not a product: It is more of a philosophy. Technically, DEN is a specification of an object-oriented information model. This is a set of:

- Classes, such as generic nodes (e.g., IP routers, FR switches)
- Class attributes, such as location, owner/operator
- Class methods, such as **delete** a specified node or**monitor** a specified node for notifications
- Class relationships—for example, a node that owns connections that originate on it and terminate on other nodes

This models NEs and services as part of a managed environment in a repository-independent fashion. The purpose of DEN is to bind users and services to NEs, network paths, bandwidth, and other network parameters. To really leverage the concepts of PBNM, it is necessary to combine it with DEN. Central to DEN is the deployment of a strong directory technology product, from organizations such as Novell, Netscape, and Microsoft. Both the information model instantiation and the policies are stored in the directory.

The focus of DEN lies in providing a type of single system image for a managed network by combining the disparate elements of:

- A technology-independent information model
- A directory system for storing policies, the object model, and its instantiation (the network, its devices, users, services, etc.)
- A policy system as discussed earlier
- A traditional SNMP-based NMS

This is an ambitious undertaking combining products from a range of vendors. We now briefly discuss the principal elements.

The Information Model

There are currently two important standard information models:

- The Common Information Model (CIM)
- An extension to CIM called Directory-Enabled Networking (DEN)

CIM is an object-oriented model that describes how a system and its components may be managed. A central tenet of CIM is the presentation of a consistent view of the managed network, independent of any protocols (such as SNMP) and data formats supported by NEs. CIM is a layered model that starts with generic classes. These classes can be built upon and refined (e.g., starting with devices at the top level, a router is a refinement of the basic device type, as is a switch). Some CIM model components are:

- System

- Device
- Application
- Network

By describing these objects in a standard fashion, NMS designers can leverage existing work. This avoids the task of inventing a vendor-specific information model. The use of a standard model can also assist in interworking with other products.

DEN is an extension of CIM. It describes the physical and logical characteristics of NEs, service, and the policies used for their management. DEN provides a mapping of the information to a format that can be stored in an LDAP-based directory system.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

IP QoS and the Enterprise

Converged enterprise networks have a critical need for controllable IP QoS levels. The various voice, video, and data applications in common use must be assigned specific service levels. In this section we examine the solution that MPLS offers in the provision of IP QoS.

A service can be defined [[RFC2475](#)] as a set of characteristics of packet transmission in one direction through a network. A VLAN is a type of service in that it gives members a single broadcast domain, possibly distributed across a large enterprise network. A network operator views a service as a kind of pipe through its underlying network. The end user simply pushes traffic into the pipe (in this case, a VLAN) and may in fact be unaware of the existence of any service. SLAs (in relation to network management) dictate parameters such as:

- Maximum allowed bandwidth
- Source IP addresses permitted to send traffic
- Destination IP addresses permitted to receive traffic
- Maximum packet burst sizes, for instance, 1000 packets per second
- Traffic handling scheme
- Average packet size

Traffic that does not conform to the SLA may be either tagged or dropped. Tagged traffic may be forwarded if resources permit, or it may be dropped at a subsequent point in the path through the network. Enterprise network operators require the ability to easily create and manage different services in order to fulfill SLAs. This is an important NMS requirement, particularly as many organizations adopt a customer-supplier relationship with their IT departments. It also applies to organizations that outsource IT functions.

A number of approaches can be adopted in the provision of IP QoS (i.e., a domain that offers more than best-effort service):

- IntServ
- DiffServ
- IntServ over DiffServ
- MPLS and IntServ
- MPLS and DiffServ

IntServ provides an end-to-end QoS model using microflows between specified endpoints. The routers in the path must maintain state information. IntServ provides a coarse-grained approach to traffic management. The RSVP protocol implements the IntServ model. Incoming traffic is pushed into a given microflow based on its destination address. The need for state maintenance and path reservation is often raised as a scalability concern with the IntServ approach.

DiffServ (as we saw in the previous chapter) adopts a divide-and-conquer approach. Traffic is marked (with a DSCP) prior to entering the QoS domain, and routers apply specific per-hop-behavior (PHB) based on the marked values.

MPLS and IntServ can be used in conjunction to create LSPs. Also, MPLS can use DiffServ to provide more advanced QoS capabilities (often in conjunction with special-purpose network processors) by the use of:

- Special-purpose edge nodes (such as MPLS LERs)
- Per-hop forwarding behaviors for specific traffic types

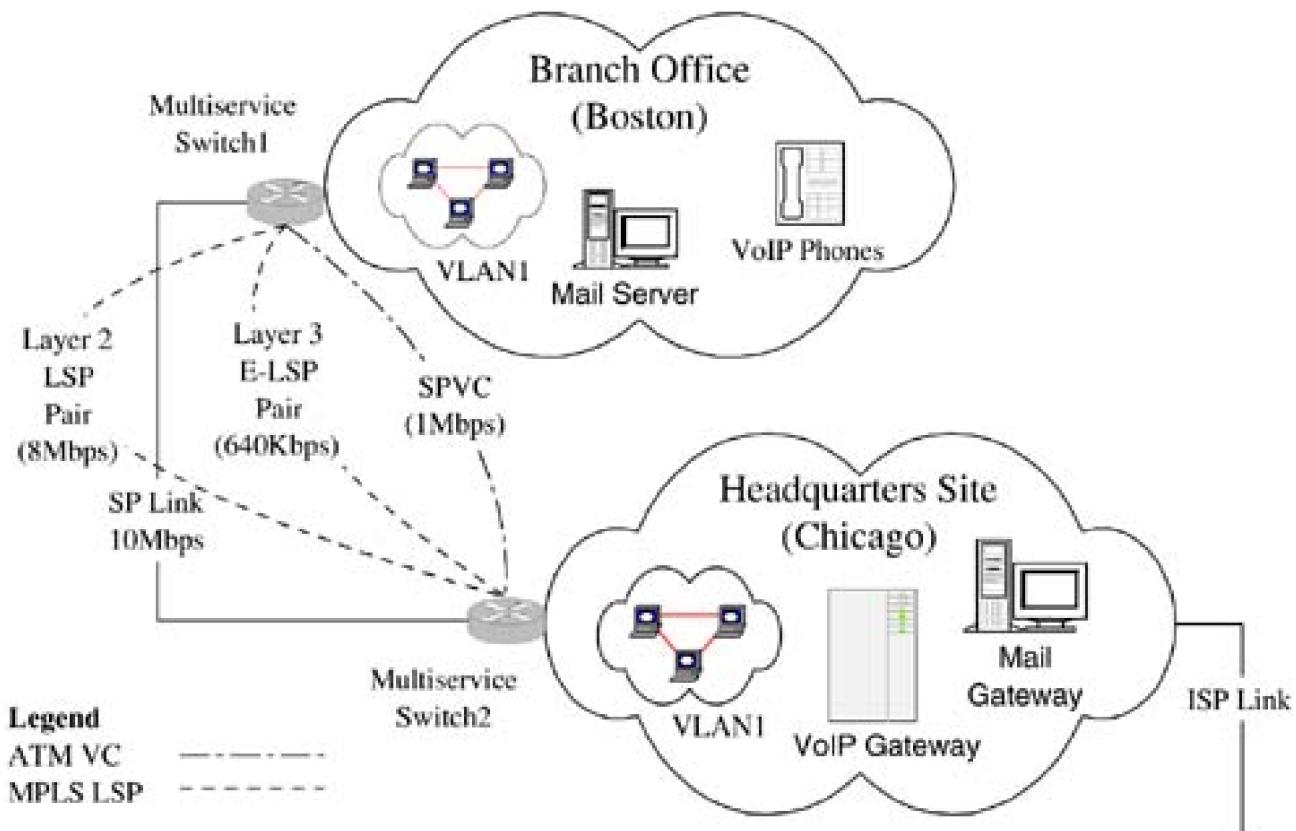
- Packet classification
- Traffic conditioning—metering, marking, shaping, and policing (tagging or dropping out-of-profile traffic)

DiffServ allows a useful separation between traffic conditioning and service provisioning functions from the forwarding behaviors implemented in the network. The end-user IP traffic is marked with a DSCP in the DS IP header field (as we saw in [Chapter 3, Figure 3-5](#)). The DSCP corresponds with a specific set of downstream traffic handling criteria. Scalability concerns dictate that there are a limited number of forwarding behaviors associated with the DSCPs. This reduces the amount of QoS-related decision-making needed in NEs. In fact, the six bits provide for 64 (2^6) different DSCP. Out of these, 32 are set by the standard (RFC 2474), 16 are reserved, and 16 are available for experimental use. Each one corresponds to a PHB—expedited forwarding, assured forwarding, and so on. The standard PHBs consist of:

- Default—No special forwarding treatment, that is, best effort.
- Expedited forwarding (EF)—Packets should be forwarded with minimal delay and low loss. RFC 2598 indicates that the EF PHB DSCP is 101110 (this value is written into the DS field of the IP header, as seen in [Chapter 3, Figure 3-5](#)).
- Assured forwarding (AF)—Packets are forwarded based on queuing class and drop precedence; for example, packets marked AF11 and AF12 would be pushed into the same queue. However, AF12 packets are more likely to be dropped if congestion occurs.

[Figure 4-9](#) illustrates two enterprise sites called Headquarters (Chicago) and Branch Office A (Boston) joined by a 10Mbps leased line (SP Link).

Figure 4-9. Multisite enterprise traffic management.



The two sites group functionally related users (such as NMS developers) into a VLAN1—the latter forms a multisite broadcast domain. Multiservice switches (containing both ATM and IP/MPLS hardware) are located at each end of the intersite link, and all traffic passes through these NEs. The traffic emitted by the Boston office originates from:

- VLAN1—Ethernet (i.e., layer 2) frames on their way to VLAN1 in Chicago

- VoIP— IP telephony traffic on its way to a VoIP gateway in Chicago
- Mail Server— SMTP traffic on its way to a mail gateway in Chicago for distribution

The Chicago site acts as the termination point for both Internet and telephony traffic. The three traffic types (originating in Boston) must cross the intersite link to get to Chicago, and to achieve this we use three virtual circuits (or connections):

- VLAN1 traffic passes over a pair of layer 2 LSPs.
- VoIP traffic passes over a pair of layer 3 LSPs (specifically E-LSPs, as described below).
- SMTP traffic passes over an ATM SPVC.

Creating these circuits requires us to instantiate rows in the MIB tables of the SNMP agent in Multiservice Switch1. We will see how this is done in [Chapter 8](#), so for the moment, let's assume we use an NMS to magically create the circuits for us (after we specify the two endpoints and the required resources). Each of the virtual circuits has an associated bandwidth reservation, and the allocated value has been derived by IT. This allows for the link bandwidth to be carved up for use by the applications in both sites. If the bandwidth calculations are correct, then there is no need for overengineering. We note in passing that this is referred to as static traffic engineering (i.e., analyze traffic patterns, set the allocated bandwidth, and reconfigure later if required). Dynamic traffic engineering attempts to modify resources on the fly.

Traffic from VLAN1 is forwarded from Boston to Chicago (and vice versa) across the Layer 2 LSP pair. One of the LSP pair handles traffic from Boston to Chicago while the other handles traffic in the opposite direction. VoIP traffic passes over the Layer 3 LSP pair. Finally, SMTP traffic passes over the SPVC.

The only real-time traffic passing between the two sites is VoIP telephony, and we assume that all such traffic is marked to receive DiffServ EF forwarding. The DSCP in the IP header of these packets is then mapped into the EXP bits of the associated MPLS label (as in [Figure 4-10](#)). This ensures that when the VoIP packets are MPLS-encapsulated, they are pushed into the LSP where they receive the required PHB. Because the LSPs use the contents of the EXP field for scheduling priorities, they are referred to as E-LSPs. The E-LSPs in [Figure 4-9](#) are dimensioned to carry 10 uncompressed voice channels of 64Kbps each.

Figure 4-10. MPLS shim header structure.

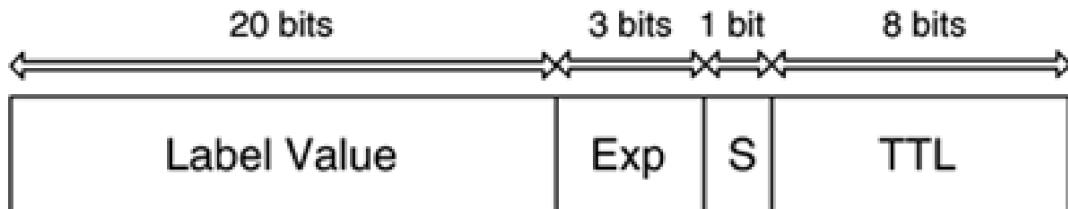
Legend

Label Value = MPLS Label Value, e.g. 0 = IPv4 Explicit Null

Exp = Experimental Bits

S = Stack Position

TTL = Time To Live



Once the virtual circuits have been created and are carrying traffic between the two sites, the NMS can examine the statistics MIB tables to verify correct operation. This consists of checking that packets are not being dropped excessively, bandwidth levels are adequate, and so on.

MPLS Differentiated Services Support

[Chapter 3](#) discussed the IP header DS field. This can be mapped into the label associated with an MPLS LSP. Such an LSP is known as an E-LSP (as described in RFC 3270 and [\[DavieRehker2000\]](#)) because its forwarding treatment is derived from the EXP(erimental) bits in the label. The MPLS label structure is illustrated in [Figure 4-10](#).

MPLS nodes use the label as the packet travels across an LSP. For an E-LSP, when the associated IP packet crosses the boundary of the MPLS cloud, the value of the DS field is mapped into the MPLS label EXP field. This field then dictates the PHB for the packet as it is forwarded through the MPLS cloud. The S bit dictates the position of this header in a hierarchy of LSPs. The topmost label in the stack is the one used for making forwarding decisions, and this is indicated by a value of zero in the S bit. Labels lower down the stack are used only when the ones above them have been popped. The TTL (Time to Live) field is used to ensure that the number of hops in the MPLS cloud has some meaningful value. Its value may be decremented at each hop, and once it reaches zero, the packet may be dropped. The standards also describe another option that uses the entire MPLS cloud as a single hop.

Where the scheduling behavior is derived solely from the label value, the LSP is known as an L-LSP. This type of LSP can be used on links (such as ATM) that cannot encode the entire label structure. There will be one L-LSP for each supported PHB, and the drop precedence can be encoded in the link layer header (e.g., in the cell loss priority bit field for ATM). RFC 3260 contains more information on these topics.

Attacks Against DiffServ Networks

DiffServ networks provide the means for advanced QoS features to be deployed at layer 3. This introduces a new level of vulnerability to external attacks because fraudulent use of such networks can have a potentially more destructive effect than is the case for best-effort service. We mention, in passing, that two types of attacks can be directed specifically at DiffServ networks:

- Denial of service
- Theft of service

If a network provides a limited set of PHBs, then denial of service can occur if a user fraudulently marks his or her traffic with a higher level of service than the one allocated. This may cause the network to consume more of its resources in handling the fraudulent traffic. This in turn can cause a denial of service to legitimate users by reducing the available bandwidth. The fraudulent marker is also guilty of theft of service in this case. More detail on the growing threat of network attacks can be found at [\[CERTWeb\]](#).

[\[Team Lib \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Summary

We covered a lot of ground in this chapter. Network management problems *can* be solved. Creating NMS that incorporate standard GUI paradigms is very important for increasing usability and reducing end-user training time. Providing NMS that incorporate well-engineered solutions is a useful guide to developers. Components form a useful abstraction for building durable management features. Components should provide ease of use for developers, and relationships between them should match those found in real-world objects, such as Frame Relay interworking across ATM, virtual connections, and EROs. The broad range of expertise in NMS vendor organizations can be leveraged to produce high-grade solutions. NMS can themselves be improved to support increasingly popular features like end-to-end services, particularly when standards like PWE3 emerge (an important point about PWE3 is that it is edge-to-edge but enables end-to-end services and QoS).

The stovepipe structure of many NMS was described along with some consequences of the fact that FCAPS applications usually share a database and a host machine. MIBs can also stand some improvements in the shape of default values, loose column coupling, and centralized features. The merits of a single data model were enumerated, with an unexpected benefit being thinner (simpler) provisioning code. Less code helps in speeding up development.

Distributed NMS applications provide some advantages in the form of less heavily loaded hosts, but this may be at the expense of more network traffic. Raising the intelligence of network devices is another good way of improving overall network function and manageability. This involves getting NEs to do more work, including preprocessing (or aggregating) traps and supporting policy-based management.

PBNM was introduced as an emerging model for management. By pushing policies onto NEs, the philosophy of PBNM is to treat the network as a type of computer. This allows for a more fine-grained management model than is available using a connection- and node-based approach. The QoS mechanisms in the IP and MPLS networks were discussed. Policies were described prior to introducing COPS and COPS-PR for policy distribution between clients and servers.

An increasingly advanced range of network processors is helping to encourage adoption of PBNM by allowing advanced traffic management functions. PBNM can be leveraged to the greatest extent when it is deployed in conjunction with DEN. This is a powerful means of modeling and controlling the managed network.

Integrated and Differentiated Services were described in the context of IP and MPLS. Layer 3 QoS is efficiently enabled using these technologies. Attacks on networks continue to pose a substantial risk, and this is perhaps even more serious with DiffServ networks.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Chapter 5. A Real NMS

Up to now we have looked at network management in a fairly theoretical way, with the discussion centering on:

- MIBs, agents, and managers
- Enterprise and SP networks
- Connection management at layers 2 and 3
- NMS/EMS layers
- Network migration to layer 3
- MPLS
- Scalability
- Development skills needed for producing high-grade NMS
- Operational skills needed for running networks and using NMS

In this chapter we look at a real NMS product—Hewlett-Packard's OpenView Network Node Manager—and describe some of its principal features. No endorsement of HP OpenView is intended. It is selected purely because it is relatively well-known and also to illustrate the concepts described in this and previous chapters. This product is special in that it is in widespread use by both enterprises and service providers. The headings under which we study it are:

- FCAPS support—the basic NMS application areas
- MIB support features, such as loading new MIBs for third-party NEs
- MPLS support
- Policy support
- Reliability features, such as support for failover
- Integration with other software, for example, data export/import, trouble ticketing, and workflow
- Programmability, for example, adding additional software for new features such as provisioning

In this chapter we leave the design and development domains, and look more closely at the way organizations actually use real network management products. A good description of this and many other SNMP management features can be found in [\[EssentialSNMP\]](#). Following these sections is a description of the typical business processes and workflows in enterprises and how these are facilitated using NMS.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

HP OpenView Network Node Manager (NNM)

HP OpenView [[HPOpenView](#)] is a broad family of network and system management products. This reflects the fact that network management needs in general are sophisticated and wide-ranging, and no single product can generally satisfy any more than a small subset of the overall user requirements. Central to the function of NNM is the management station—the command and control center. This is the computer (UNIX or Windows) that performs the bulk of the data:

- Collection of information from NEs via SNMP notifications/polling
- Processing and archiving in a database
- Distribution to registered applications

For large networks, there can be more than one management station, and these can exchange information with one another. A management station can also delegate NE data acquisition to collection stations. The latter connect to NEs and collect status and configuration data, which is then passed to the management station. Typically, HP suggests that a single management station can handle 5,000 managed objects. Up to 60,000 objects can be managed if collection stations are deployed along with the management station [[HPNMScale2002](#)]. This type of distribution helps to reduce the load on the NMS. In its default-installed state, NNM provides the following main features:

- Discovery and mapping
- Monitoring
- Notification processing
- Reporting
- Data warehousing
- Backup and restore of firmware and configuration data
- Java interface for remote access to management features
- Remote administration of the NMS

As these are quite generic NMS facilities, each of them is discussed in general terms in the following sections. Each section is then followed by an NNM-specific description. These facilities are all based on manually retrieving or asynchronously receiving data from the network. Pushing data onto the network can be achieved only by using special-purpose, third-party, add-on software. In this sense, NNM is a platform on which additional software can be layered in order to achieve full-featured network management. This again reflects that network management is complex and requires a range of software applications that go together to make up an overall solution. As networks grow, they can be effectively managed only using advanced software.

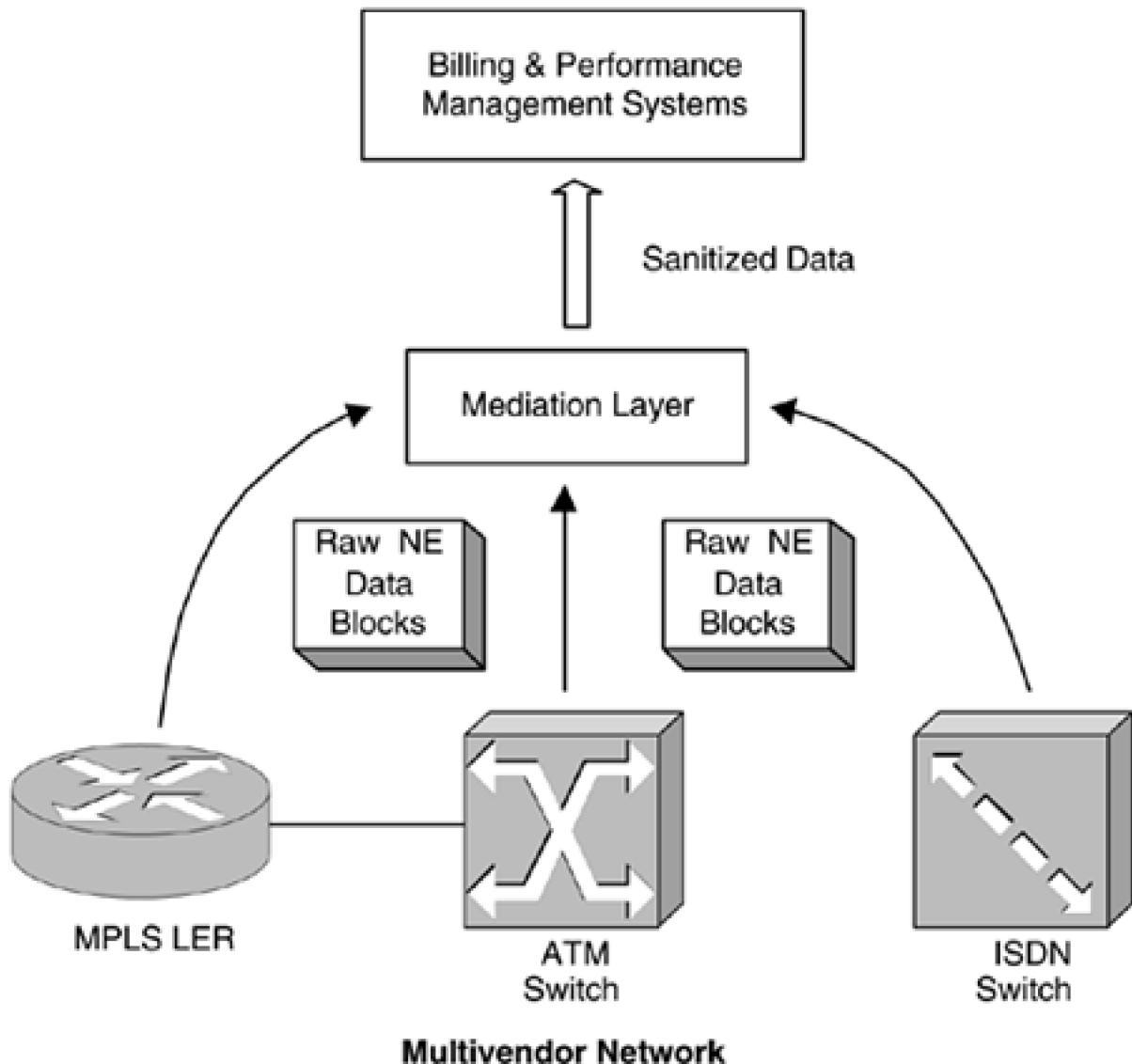
An interesting aspect of managing modern networks is that there is no single solution to all the network management needs. This is similar to the fact that there is no single desktop software application that satisfies all possible needs (word processing, email, Web browsing, spreadsheet, etc.). Desktop user workflows and business processes are generally too complex and varied for it to be feasible to produce one application capable of handling them all. Instead, a range of software packages are deployed and used. In the same way, the many commercial network management tools go together to make up a continuum of packages. This then forms the basis for the management solution. Substantial teams of people are needed in vendor organizations to service the ongoing (and changing) needs of large enterprise customers.

Before looking at NNM, we take a brief detour into the important area of mediation.

Mediation

Once we start to look at the way NMS are used in practice, the area of mediation becomes relevant. This is another type of layering, similar to the ones we have mentioned in previous chapters. Mediation software exists to protect application layer software from proprietary configuration data. [Figure 5-1](#) illustrates a multivendor network operating in conjunction with a mediation layer that feeds network-originated data into a set of NMS applications. Raw data is processed by the mediation layer and passed up to the applications for further processing.

Figure 5-1. Mediation in a multivendor network.



Multivendor Network

An example of one of these applications is billing—the A (accounting) in FCAPS. The NEs in [Figure 5-1](#) generate data relevant to billing, such as:

- The number of ATM-encapsulated IP packets received by the MPLS LER
- The number of ATM cells received by the ATM switch
- The number of ATM cells forwarded onto an ATM virtual circuit
- The number of voice calls made using the ISDN switch

Typically, the NEs generate billing data in a proprietary format, and this must be transferred from the network into the mediation layer. The processed data emitted by the mediation software is presented in a standard format such as Billing Automatic Message Accounting Format (BAF) that can be handled using a standard commercial off-the-shelf (COTS) accounting package.

[Team LiB]

◀ PREVIOUS NEXT ▶

Network Discovery and Mapping

Discovery is a useful NMS feature by which NEs are automatically detected (or learned) and recorded. It fits into the C (configuration) part of the FCAPS areas. Automatic discovery frees the user from the potentially error-prone and tedious task of manually entering and maintaining the details of the deployed NEs. Large service providers tend to deploy NMS from the point in time that they start to build their networks (smaller service providers may wait for some revenues before adding an NMS). So, the burden of data entry may not be so great for large service providers; that is, they add an NE to the network and simultaneously add all the necessary details to the NMS. Once the deployed NEs are known and stored in a persistent repository, they can be managed. Discovery typically follows three main stages:

- Initial discovery of previously unknown NEs
- Incremental discovery of some change that has occurred to previously discovered data—for example, new cards (hardware) are added to a switch or additional protocols (software) are configured
- Discovery of removal, where a device is taken out of the network and is then automatically removed from the NMS

Initial discovery occurs when a device is encountered about which nothing is known by the NMS. Its details are read by the discovery application and recorded in the database. Examples of such details are:

- IP address of the SNMP agent on the device
- IP address of the device interfaces
- Device type; for example, for multiservice switches, this might be some combination of SONET/SDH, DWDM, ATM, MPLS, Frame Relay, and so on
- Inventory details, such as configured software and cards deployed in the device
- Protocols and technologies running on the device, such as ATM PNNI, MPLS, X.25, IS-IS, and so on
- Links to other devices
- Virtual connections, traffic profiles, route objects, and so on

NE software configuration is equally as important as the hardware details. An example is the set of ATM and MPLS virtual circuits we saw in [Chapter 4](#), "Solving the Network Management Problem," in [Figure 4-9](#). These circuits facilitated the exchange of a range of traffic types between the two geographically distant sites.

The results of discovery are extremely useful, but even automated, it can be a time-consuming and expensive operation requiring SNMP (not all NEs support SNMP, e.g., many optical devices use the OSI management protocols for historical/technical reasons) messaging, and much database activity. Discovery provides the information needed for the other functional areas of management (such as provisioning, fault handling, performance analysis, and billing), but it should try not to do this at the expense of the overall solution. A balance is needed between supporting all the FCAPS features and the computational discovery effort required in trying to maintain parity between the actual network and the list of discovered entities.

A network-mapping feature further processes the discovered NEs and attempts to understand and depict the logical (and sometimes geographical) interconnections between them. Knowing the interconnections allows for a more comprehensive understanding of the network operation.

NNM Discovery and Mapping

NNM provides an automatic discovery mechanism. This reflects the fact that many deployments of NNM are in enterprise networks with a wide range of layer 2 (switches, bridges, repeaters, etc.) and layer 3 devices. The discovery process uses SNMP-based polling and ICMP requests (over UDP and IPX) to build a picture of the network. For networks connected to the Internet, a seed file must be provided in order to avoid trying to discover NEs outside the organizational boundary.

The discovery process populates an IP topology database using a series of tables such as:

- Network-level connectivity
- Segments
- Nodes
- Interfaces

This grouping allows NNM to create logical maps of the NEs and to graphically indicate operational status using a color, such as green for up, red for down, and so on. An icon representing a network can be expanded to show the constituent nodes. Similarly, nodes can be viewed in terms of their interfaces. In other words, containment relationships are depicted clearly and intuitively.

Updates to the topology database occur continuously as a result of information received from *managed* nodes. These are nodes that are specially designated by the network operator for regular polling. Both status and configuration changes are recorded for such nodes. On the other hand, the operator must explicitly initiate on-demand, polling of *unmanaged* nodes. These are nodes deemed to be either relatively unchanging or not as important (from a management perspective) as their managed counterparts. This reduction in the number of managed nodes assists in improving scalability of both the network and the NMS.

Monitoring

Monitoring is the process of recording temporal changes in the status of managed objects such as:

- Nodes
- Interfaces
- Links
- Virtual connections (e.g., ATM PVCs, LSPs)
- Ethernet VLANs

Status changes can be simple transitions such as link/interface up or down, or more complex, such as when an LSP path is being signaled through the network. In the latter case, a complex and dynamic state transition is occurring. Many such status changes can have an important bearing on the service received by the associated end user. An example is when an interface that is part of an ATM PVC goes down. The interface is no longer able to handle traffic. Such a status change may be service-affecting if there is no backup connection. For this reason, monitoring functions are an important part of an NMS, and the faster they record changes, the better.

The same process that carries out discovery also executes NNM monitoring. This is convenient because both discovery and monitoring can use the same set of objects—lists of NEs, interfaces, links, connections, and so on. Status changes are reflected back into the topology.

[Team LiB]

◀ PREVIOUS NEXT ▶

Notification Processing

Notification processing is an important part of network fault management—this is the F (fault) part of the FCAPS areas—arguably the most critical part of any NMS because faults generally reflect problems in the network. Network problems can in turn affect end users. Notifications are the means by which SNMP agents asynchronously communicate problems with their NMS. From a scalability perspective, notifications provide a cue for remedial action from the NMS in response to some change in the network. This reduces the need for polling by the NMS. A number of issues arise in relation to SNMP notifications:

- Notifications are not acknowledged by the NMS (unless they are informs).
- Notifications are transported using the UDP protocol and hence are unreliable.
- Faulty NEs can generate many notifications.
- Aggregated services that become faulty can result in notification storms.
- New hardware being added to (or reconfigured in) a network can produce notification storms.

When an NMS receives an SNMP trap over an unreliable transport, it never acknowledges it. This is in the interests of scalability and keeping the management protocol as lightweight as possible. It also helps avoid exacerbating situations such as network congestion. When an agent detects a problem, it sends a best-effort notification message and delegates resolution of the underlying problem to the NMS. Networks are often designed to leave an absolute minimum of about 25 percent bandwidth free to allow for routing, signaling, and management protocols to continue to operate at all times. If this is adhered to, then in theory agent notifications should always get through to the NMS. This enables the latter to carry out some meaningful remedial action.

Faulty NEs can generate large numbers of notifications; for example, if a node interface is flapping up and down, then each status transition results in a new notification. The NMS user should quickly try to resolve this by Downing the associated link or resolving the underlying problem with the interface.

Aggregated services, such as layer 2 VPNs (as we saw in [Chapter 3](#), "The Network Management Problem," [Figure 3-2](#)), may have thousands of underlying connections. If a major fault occurs, such as a fiber cut, then the originating node for each affected connection may legitimately emit a notification. This can result in a great many notifications, particularly for the increasingly dense next-generation NEs (described in [Chapter 3](#)). If the NEs are aware they are participating in a VPN, then it should be possible to intelligently reduce the number of notifications, as discussed next.

MIB Note: Scalable Aggregated Services

Managed objects that are constituents of an aggregated service (such as all the virtual circuits in a layer 2 VPN) can be logically grouped by an NMS. This allows the network operator to view the service as a whole rather than as a collection of objects. The NEs are not generally aware of such a grouping. This means that the associated NEs cannot act in concert in the event of faults. This can lead to problems like notification storms.

A MIB table that expresses membership of aggregated services like VPNs could help prevent such notification storms. MIB indexes of members (e.g., virtual circuits) could be entered in the table, and the NEs could then negotiate overall service status before issuing notifications. This would have the effect of pushing more intelligence into the network and reducing the burden on the NMS. Given the trend towards increasingly dense NEs with more complex component objects (such as layer 2 and layer 3 VPNs), this type of issue may become more important.

NNM Notification Processing

NNM uses the term *event* to describe NE notifications as well as messages from other sources (e.g., external applications). NNM provides an alarm browser for all such events. Important (service-affecting) events can then be configured to show up as alarms so that operator intervention is prompted.

NNM distinguishes between SNMP notifications and events. The lifecycle for a notification is as follows:

- An NE sends a notification.
- The notification is received by NNM and logged.
- NNM then distributes the notification to applications that registered for it.

NNM allows notifications to be paired so that notification A indicates a problem (e.g., link down) and notification B indicates problem rectification (e.g., link up). Not all notifications are symmetric like this; for example, if an LSR receives an MPLS-encapsulated packet with an invalid (or unknown) label—this is called a label fault—then there is no correction for this. It is a once-off, hopefully transient type of error. Paired notifications assist a network operator because they reflect those situations when the network self-heals. Likewise, when the corrective notification does not occur, then the fault remains active.

NNM also supports event correlation in which a given notification is processed before it is forwarded to one of the applications. This helps in situations where the same notification keeps recurring. As mentioned in the previous chapter, a very useful NE facility would be one that allows for notifications to be staggered or paced in order to avoid flooding the network with unnecessary traffic. This is particularly relevant during network reconfigurations. Some MIBs support notification throttling (RFC 1224) by using a sliding window of a specific duration (in seconds) and limiting the number of notifications allowed in this window.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Reporting

Reporting is one of the most important features of an NMS. Data is retrieved from the network and presented in a specified report format. This can include:

- Deployed NEs
- NE configuration
- Interfaces
- Links
- Virtual connections

The data gathered is presented in a manner useful to the operator.

NNM Reporting

NNM reporting is GUI-based and accessible through a browser. The main options are:

- Report configuration: Create, delete, and modify reports
- Report presentation: View reports

NNM reports can be:

- Scheduled daily/monthly
- Configured/viewed using a standard browser
- Automatically emailed to a recipient

NNM comes configured with standard reports that can be used immediately or extended. Examples of such reports are:

- Accounting
- Availability
- Congestion, for example, FR PVCs
- Historical details
- Inventory
- Performance
- Real-time details

- Thresholds
- Trend analysis
- Utilization

These provide a useful foundation for creating additional reports if required.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Data Warehousing

NMS provide a persistent repository, typically, a commercial database product such as Informix, Oracle, or Ingres. It is this database that the NMS tries to keep synchronized with the network state. This is also the database used by remote management clients. By storing most of the managed data centrally, the outlying components of the NMS can be made as thin as required, even hosted on standard COTS applications such as Web browsers. It also facilitates data security to a degree because the database can be hosted in a secure location with access granted only to authorized clients.

Once a network topology has been discovered and stored in the database, the user can execute management operations against it such as:

- Viewing the configuration of a given node
- Provisioning an ATM virtual connection between two nodes
- Viewing the active faults on a node interface

The database is therefore a key component of the NMS. All of the FCAPS applications use it. Commercial database products are often complex, sophisticated software products in their own right.

NNM Data Warehousing

NNM provides an embedded relational database into which it stores its management data.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Backup and Restore of Firmware and Configuration

Backup and restore are important functions in any network environment. In the case of network management, both firmware and NE configuration are high value items. The firmware version on a given NE is subject to change as new features and technologies are added. It is important to be able to back up the existing version of firmware, allowing for it to be reapplied if required. One application for this would be firmware redistribution as part of disaster recovery, such as a lightning strike.

Configuring NEs is also an increasingly nontrivial task as more and more technologies and protocols are packed into devices. Once the NEs have been configured and are operational, it is important to be able to back up the settings—often called a configuration database. As for the firmware case, the configuration details can then be restored if necessary. In providing the backup and restore capabilities, equipment vendors can use protocols such as File Transfer Protocol (FTP) and Trivial File Transfer Protocol (TFTP).

NNM Backup and Restore

NNM provides a backup facility that allows a snapshot to be taken of the topology and maps. These are frozen during the time it takes to copy the data into a backup directory. After this, normal service ensues. At no point during this process do the alarm handling and data collection procedures stop.

[Team LiB]

◀ PREVIOUS NEXT ▶

Java Interface for Remote Access

Standard Java-enabled Web browsers can play an increasingly important role in network management because they are COTS products. This reduces the investment required for producing clientside NMS solutions because almost no development is required for the client application framework. The browser provides the execution environment for client applications. A Java-based client application can be executed using the browser to download a Java applet. The use of Java has a number of additional merits:

- Java is object-oriented.
- Java provides built-in security.
- Java is a multiplatform programming language.
- Java supports field-replaceable packages.

The needs of network management solution developers are well served by the object-oriented features of Java [[JavaDev](#)]. The multiple views of UML fit easily into a Java environment, and some development tools (e.g., Telelogic Tau and Rational Rose) can generate Java code.

Security is an increasingly important issue in network management as the incidence of attacks increase [[CERTWeb](#)], so it is with good reason that security increasingly concerns the operators of all types of networks. All levels of network management infrastructure have to be protected from the NEs/EMS, all the way up to the business-management layer. Multiplatform support is also important, because two of the main competing platforms are:

- The many different versions of UNIX, such as SunOS, Solaris, and HP-UX
- Windows NT/2000

Java allows for full compatibility across these platforms: Code written for NT should run unmodified on the others. This can help reduce development and testing time.

As NEs deploy increasingly advanced features, it is often necessary to upgrade deployed NMS components. Java provides an elegant means for doing this by the use of packages. These are logical Java code and data entities that can be signed for security and swapped in and out of systems as required. Packages also provide many advantages for developers, such as encapsulation and interfaces.

NNM Java Interface

The Java-based Web interface in NNM allows the user to connect to the NNM management station. It is also possible to offload the server processes from the management station. This permits the remote viewing of:

- Topology
- Alarms
- Node status
- MIBs
- Maps

- Graphs

The Java interface exports much of the management station function onto desktop systems equipped with little more than standard browsers. This is a very powerful use of COTS software and illustrates the merit of thin clients based on standard browsers. It allows for easy remote access to NNM. Remote operation provides several advantages:

- Clients can be geographically remote.
- Scalability is improved because client applications execute remotely, offloading NMS servers.

Geographic distribution means that network administrators do not have to be located close to the management station. They can dial in and access the functions and features. This raises an important issue of security. Network topology details are sensitive matters from the perspective of security, infrastructure protection, and commercial advantage.

Offloading management station functions is useful for freeing central resources. This can facilitate deployment of more advanced third-party software features on the central server.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

MIB Support Features

MIBs are the cornerstone of SNMP-based NMS. They provide details concerning the network managed objects and form the basis for the NMS data model. In an ideal world, the MIBs would be sufficiently flexible to provide the bulk of the data model, but (as discussed in [Chapter 3](#)) for the current generation of MIBs, this is generally not the case. NMS should provide a number of baseline features for MIB support, including the ability to:

- Load new MIBs in addition to the existing set
- Support multiple versions of the same MIB
- Unload MIBs no longer in use
- Browse and analyze MIBs

When new devices are added to networks, their associated MIBs must be loaded into the NMS in order to provide managed-object access. It should be a relatively simple matter of adding new MIB files to the existing set. The new MIB files should be automatically compiled by the NMS and verified for correctness. Following the addition of the files, the associated managed objects should then become visible to the NMS. This latter would almost certainly be a manual configuration step, but the inclusion of marked objects could help in automating it, as described next.

MIB Note: Principal Managed Objects

Mandatory managed objects in a MIB should be clearly indicated and marked for easy inclusion in anNMS. These MIB objects reflect the core management features supported by a host NEs, and ease of inclusion facilitates importing new NEs into existing managed networks. (This point was made in the preface).

If the mandatory objects are marked (or tagged), then this can also facilitate automatic parsing. A sample marking could be a simple MIB comment like "--M" placed just before the mandatory object with "--/M" just after it. Coupled with default object values, this serves to improve the device manageability. The MPLS MIBs already include this with MODULE-COMPLIANCE clauses. RFC 2580 provides useful SMIv2 conformance guidelines; for example, related objects can be indicated using the OBJECT-GROUP macro.

It was mentioned in [Chapter 1](#), "Large Enterprise Networks," that not all NEs deployed in a network will necessarily host the same firmware version. In some cases, later firmware revisions may require extra memory or even special-purpose hardware. This reflects the ongoing problem of feature creep as NEs become more complex. Denser NEs require more RAM, flash, and more powerful processors to support higher levels of intelligence. So, it is a fact of life that a given network operator may not have a common firmware revision on all its NEs. Since the MIB set is generally compiled into the executable firmware image, it follows that there may then be numerous versions of the same MIB deployed in the network. This adds up to a broader range of managed network objects. The NMS must be able to support all deployed MIB versions. Providing this support can be difficult, particularly when (as is often the case) there exist substantial differences between the various MIB versions. MIB authors (and implementers) can greatly reduce the burden on NMS developers and users by following guidelines such as those in RFC 2578. Examples of the latter include the DESCRIPTION clause and not using reserved keywords (some MIB compilers may not complain about reserved keywords).

Where entire MIBs have been deprecated or the associated managed objects are no longer in use, it is useful to be able to retire them from the NMS. This helps to free up resources and can take the form of unloading the relevant MIB files—the reverse process of manually loading MIB files.

NNM MIB Support Features

NNM supports the loading of both standard and third-party MIBs. This helps in extending NNM to support additional and modified existing NEs. As long as all MIB objects have unique object identifiers, it is also possible to support different versions of the same MIB. Unwanted MIBs can be unloaded as required. NNM also allows operators to browse and graph managed objects from any loaded MIB. This can be done either in real time or historically.

[[Team LiB](#)]

[[PREVIOUS](#)] [[NEXT](#)]

[Team LiB]

◀ PREVIOUS NEXT ▶

MPLS Support

MPLS operates at the network level; that is, the managed objects relevant to MPLS involve more than one node. MPLS nodes have a number of managed objects, the status of which can change over time:

- Interfaces can be up or down.
- Routing protocols, such as OSPF, IS-IS, and BGP4, can be operational or disabled.
- Signaling protocols, such as RSVP-TE and LDP, can be operational or disabled.
- Forwarding table entries can be active or inactive.
- LSPs can be up or down.

There are other MPLS objects that do not change because they are static in nature:

- EROs
- Resource blocks
- Cross-connects

All of these are highly relevant to managing MPLS; for example, LSPs may span any number of MPLS nodes and can be built using EROs, resource blocks, and cross-connects. Providing MPLS support requires software for managing these and other objects.

NNM MPLS Support

NNM provides no special support for MPLS but can be extended to use the appropriate MIBs.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Policy Support

[Chapter 3](#) introduced PBNM in the context of automatic traffic engineering and explained some of its advantages. In general network management terms, policies can be used to automatically solve recurring problems, such as certain types of failures or other important events. A simple example is traffic thresholds. The user can set a threshold on an interface for the number of packets received. If the threshold is exceeded, then a notification is generated and the NMS can divert some of the traffic. A simple rule is defined to achieve this based on the setting and crossing of a traffic threshold.

NNM Policy Support

Apart from some fairly basic policies (such as issuing emails in response to certain events), NNM requires third-party products to implement policy handling. Policy-based applications provide features such as

- Application prioritization, that is, certain key applications are assigned a specified level of service according to business priority.
- Enterprisewide policy distribution, for example, access to servers and NEs.
- Service-level mappings, such as those provided by technologies like IEEE 802.1p and IP DiffServ, can be used to deliver consistent quality of service across the network.
- User-based security policies similar to the user-based security model of SNMPv3.
- CIM/DEN, LDAP, SNMP, and so on.

An example of a third-party application providing such features is Avaya CajunRules Policy Manager. We mention this product for reference only in order to assist further research into this important area.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Reliability Features

System reliability is an estimate of the probability of failure. There are a number of ways of improving reliability using backup facilities. In the case of NMS, this can take the form of protecting the central database. Failure of the database is usually fatal, and for this reason, many database vendors provide a failover capability. The user can deploy a backup version of the database that runs in parallel with the primary system. Failure of the primary results in a full switchover to the secondary.

NNM Reliability

NNM collection stations can be configured to failover to remote management stations. This allows for continuous monitoring of the network.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Integration with Other Software

Supporting the wide variety of NEs in enterprise networks requires a rich mix of NMS and operational skills. Our primary concern in this chapter is the NMS, and in this section we briefly review the function of some ancillary systems to which it may be attached. These can include software applications that handle:

- Data export of network topology/inventory data for business asset analysis—for leaseback arrangements, departmental billing, and so on.
- Data export to software-based modeling packages—traffic analysis, network design, and capacity planning.
- Alarm generation—trouble tickets, audible/visual, email, mobile telephony text message, pager, phone, and so on.
- Performance analysis—the number of packets, frames, and cells transported in a given period by a node, interface, link, or connection.
- Billing—reports generated using call detail records and other data, such as connection type, class of service, and bandwidth consumed.
- Security—distribution of keys, user account names and passwords, digital certificates, encryption settings, authentication, and so on.
- Workflow—unsignaled virtual connections, such as ATM PVCs, require manual (or software-assisted) creation. A workflow system external to the NMS may provide this facility by handling task delegation, tracking, and completion.

Business asset analysis can be used for depreciation studies, lifecycle management, and department billing. NEs are expensive items that need to be recorded, managed, and maintained from initial deployment through decommissioning. Issues like upgrade and replacement have an important bearing on the network operator.

Specialized software applications can be used to carry out offline traffic analysis. A snapshot of the discovered topology can be exported into a modeling package, and the user can execute what-if scenarios, for example, increasing the bandwidth on a given link. The effects of any such changes can be viewed offline before making a change in the network. Modeling packages may also allow new topology details to be exported back into the NMS.

Alarms generated by the network can be routed via the NMS to trouble ticket applications. This allows for recording and directing any remedial work required to clear the fault. Another method of alarm annunciation is the simple audible variety, such as sounding a bell or a computer speaker. Visual indication can be a GUI topology object color change. Routing a message to an email recipient (or a short text message to a mobile phone or pager) can be used to indicate text-based fault indication. It is even feasible that a voice phone call could be initiated by the NMS to indicate a particularly serious fault.

Performance analysis can be facilitated by a combination of special reports and third-party applications. Many NEs generate performance data records (PDR) that provide utilization details for:

- Ports, links, and virtual circuit utilization
- Protocols
- Networking technologies

Mediation software can process PDRs to produce data ready for export to reporting. This is similar to billing. Billing is already a critical SP business requirement, particularly as IP service billing becomes increasingly important [[IPDR-ORG](#)]. Enterprises also need billing as SLAs become more common.

Many NEs generate call detail records (CDR) that are preprocessed by mediation software prior to export to billing. The raw NE data

blocks in [Figure 5-1](#) are generally CDRs and/or PDRs.

Security is an issue of grave concern to the owners and operators of all large networks. The distributed nature of managed networks provides possible targets for attack:

- SNMP agents can be flooded with request messages (denial of service).
- NMS hosts can be attacked by various means, such as by viruses or hackers.
- The network joining the NEs and the NMS can be broken or spied upon.
- The data passing between the network and the NMS can be stolen, modified, or destroyed.

Security is required at all levels of a managed network, and this may require additional specialized software, such as directories for secure storage of relevant data.

Networks generally do not remain static, and changes need to be applied in a controlled manner. Workflow systems can help achieve this by tracking and managing the tasks required for running a network. This is discussed in more detail later in the chapter.

NNM Integration

Many third-party applications can be integrated into NNM: This is one of its great strengths. This includes Microsoft Systems Management Server (SMS), which can be launched via the pull-down NNM menus. The reason for integrating applications in this way is to assist in solving operational problems. Clearly, it is a matter for the network operator to decide if such close integration is required. For example, if an enterprise application starts to misbehave by continuously sending out network messages (e.g., an NMS auto-discovery application), then the operator could locate the offending machine and then launch SMS to remotely remove the application.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Programmability

The facility of adding software to an NMS can greatly extend its usefulness. This can include anything from MPLS/ATM/SONET provisioning to special-purpose NE monitoring. Open source NMS are a relatively new phenomenon and represent what might be called the ultimate in end-user configurability—actually changing the base software itself. Network management requirements are unique to every network, so programmability is an important addition.

NNM Programmability

NNM allows for user-software to be added to enhance the base functions. Examples of this can be in any of the FCAPS areas. NNM allows for geographical map-level integration between itself and third-party applications. In this, a user can select an NNM map node (such as a router) and launch third-party software to carry out operations on that device. This can include reporting, alarm and event processing, and other applications. The merit of the integration is the value added to NNM and the ease with which the third-party software can be accessed from the operational context.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Workflows and Business Processes

Enterprises deploy an increasingly wide variety of NEs as their network requirements and business processes evolve. In this section we look at the workflows and business processes surrounding the way these networks are managed and operated.

Enterprise Workflows and Business Processes

Enterprise network customers are generally the employees of the organization except in cases where e-commerce applications and extranets exist. In such cases, the enterprise network boundary stretches outside the traditional limits and allows specially authorized and protected traffic inside, such as an extranet VPN. Beyond these capabilities, the everyday management of enterprise networks broadly follows a model of:

- NE deployment, upgrades, management, and (proactive) support.
- Trouble ticket-based (reactive) workflows; typically, an NE breaks or needs service (e.g., a VLAN link becomes congested), and an IT engineer fixes it in conjunction with details entered in the trouble ticketing system. The lifecycle of the problem is recorded in the ticketing system. Using an NMS helps in recording the details of the repair work.
- General policing—minimizing traffic-generating file downloads from the Internet (and other hazardous activities) and implementing policies such as virus protection.
- Connecting the network to other networks, such as SP networks and remote sites.

Some of the NE types deployed by enterprises include:

- Switches—ATM, Frame Relay, MPLS, X.25, and Ethernet
- IP Routers
- PABXs/soft switches
- SANs
- Servers
- Wireless devices (base stations, access points, and exchanges)

The types of issues that crop up in managing enterprise networks with devices like these are:

- Fault analysis and rectification
- Rebalancing traffic after adding new hardware
- Billing
- Performance analysis
- Security
- Creating virtual connections

To illustrate a typical problem, imagine that an enterprise PC has been inadvertently disconnected from a VLAN during a network reconfiguration. The result is an inability to log into the network, access email and the Web, other such problems. This can be quickly

fixed with no measurable loss of revenue (apart from wasted time).

An enterprise uses its network to carry out its day-to-day activities rather than as a means of generating revenue. However, network downtime can be very costly. Also costly is the maintenance of multiple incompatible management systems. Consolidation of these is a useful enterprise network management goal.

[\[Team LiB \]](#)

 PREVIOUS  NEXT 

[Team LiB]

◀ PREVIOUS NEXT ▶

Applications of NMS

Increasingly, NMS are required to assist in rapidly bringing up, keeping up, and downing large networks. One application of this is when a network is rebalanced after a significant hardware addition occurs, such as a new multiservice switch. The operator wants to execute this as smoothly as possible, and this may involve:

- Managing VLANs
- Tearing down existing virtual circuits
- Re-creating the virtual circuits on the new switch
- Generating CLI scripts or SNMP messages to move the connections to the new device
- Suppressing faults during the changeover

Major reconfiguration actions such as these can be recorded by the NMS and stored as command files. These can be reused at a later time when similar operations are required.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

The Network Is the True Database

It is sometimes said that the network is the database for any NMS. As we've seen, all NMS attempt to maintain parity between their database and the one stored in the network. Saying that the network is the database reflects the fact that the following may all be stored on the NEs and accessed at will via an NMS:

- NE configuration details
- Firmware
- Performance details
- Billing details
- Security settings
- NE Faults

The merit of this is that all such device-resident data is completely up to date because it exists in the network. The problem with retrieving this data is that there is so much of it. Getting the requisite data from the network for processing by an NMS requires expensive and possibly lengthy device retrievals. As we've seen, most NMS struggle to close the gap between their snapshot of the network and the real picture.

When data is discovered from the network, it is important that it is accurately reflected in the NMS. This can lead to anomalous results when NEs support multiple personalities, such as for MPLS SIN. As discussed earlier, SIN provides support for both MPLS and ATM at port level on the same NE. Should this node be described as an MPLS or an ATM entity? Whatever solution is found, the NMS should respect the data stored in the network, because in this context, the network is the true database.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

The Network Boundary

An important point about managed networks is the definition of the boundary. This is the point outside of which the NMS has no jurisdiction. NNM must have some idea about this in order to avoid trying to discover nodes unnecessarily. This can arise when a given network is connected to the Internet and the operator wants discovery to stop its data collection at the last node before the Internet.

The network boundary problem is also seen in multiservice networks that contain numerous clouds of different technologies, including:

- ATM switches
- MPLS switches
- Frame Relay switches
- IP routers

As for the Internet case, this may require careful administrative actions, such as specifying a seed file in NNM. In essence, this sets a boundary address for management beyond which it is not permitted (or meaningful) to operate.

One last point about this is that even when the boundary has been carefully and correctly designated, the traffic introduced by network management may be too high.^[1] In this case, it is important to be able to pace the management operations to suit the available bandwidth.

[1] This is a little like Heisenberg's Uncertainty Principle. By managing a network, its characteristics are modified. Each action that pushes data into the network instantaneously reduces the bandwidth and changes the dynamics of the affected NEs.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Summary

The theory and practice of network management are separate and distinct. Effective network management requires a broad range of software tools. Two approaches can be taken: Build one all-encompassing system or divide and conquer via a number of packages. The latter is the philosophy adopted in NNM and reflects the fact that computing power is increasingly inexpensive. Specialized software can be employed for areas such as mediation, billing, and performance analysis. The cost of ownership and development can be significantly lowered by the use of standard COTS packages. Java-based software can facilitate the use of standard browsers for (thin client-based) network management. Discovery, mapping, and monitoring are often inextricably interwoven, and along with fault management, these are the big "readers" from the network. They are the means by which the NMS attempts to keep pace with changes in the network.

Minimizing the gap between the network situation and that perceived by the NMS is crucial and provides a baseline for defining the quality of a given NMS. Provisioning both writes to and reads from the network as it executes the user's commands. Fault management can perhaps be considered the most crucial of all NMS facilities. The workflows associated with enterprise and SP networks share some similarities, but the financial cost of downtime may be more keenly felt in the latter. We briefly explored the reasons why it is often said that the network is the database. All NMS have a boundary, particularly those that encompass more than one service or technology.

[Team LiB]

◀ PREVIOUS NEXT ▶

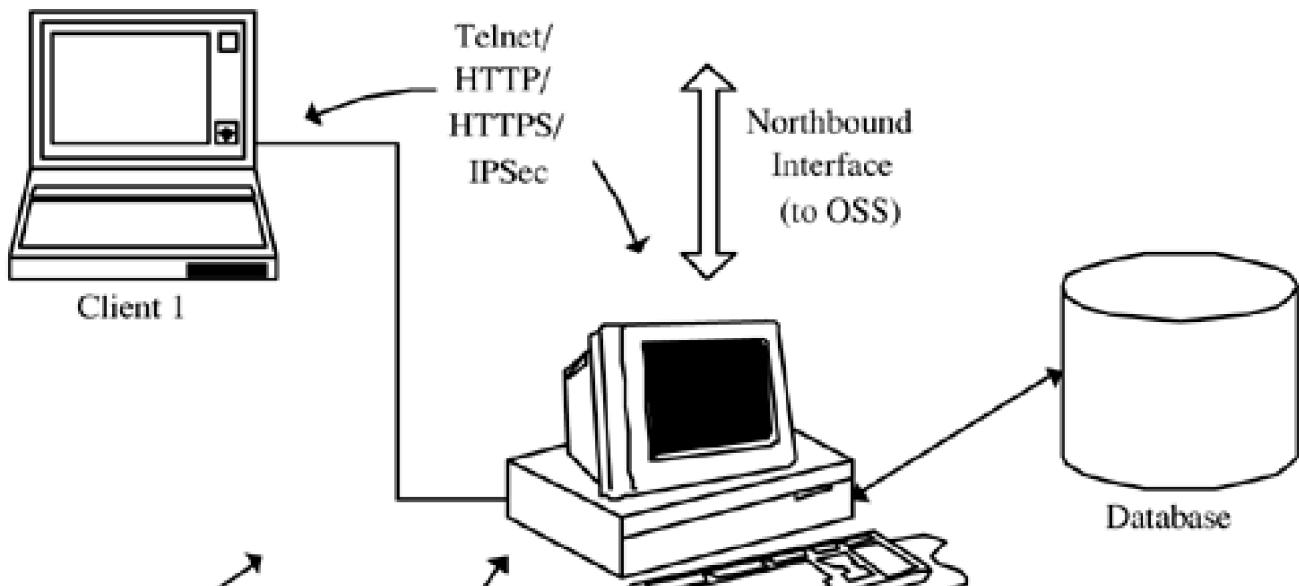
Chapter 6. Network Management Software Components

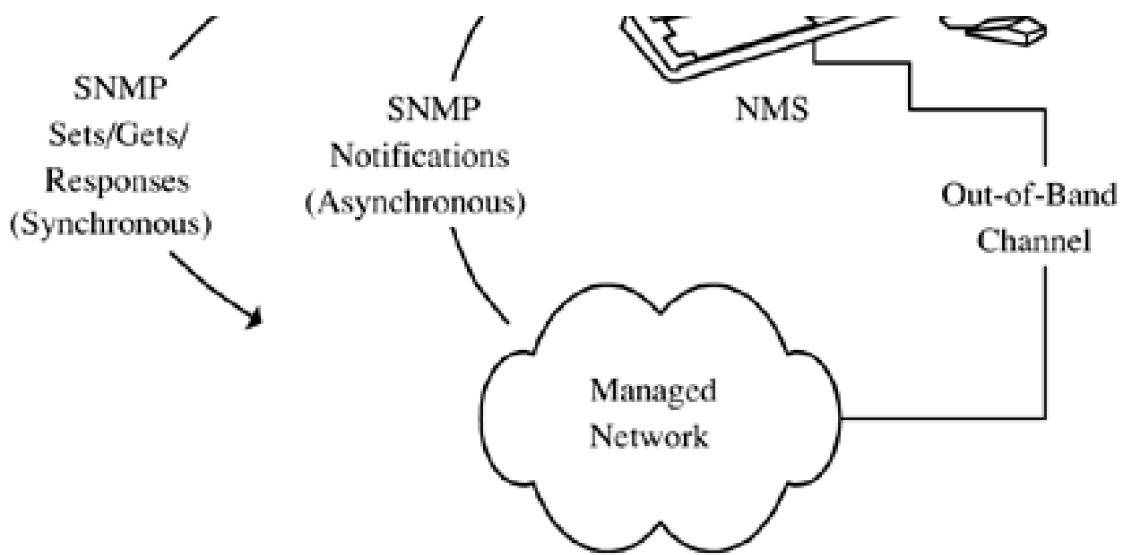
We now describe some of the internal software components that combine to make up an NMS. As we've seen, network management is complex, and the requirements of most organizations are normally filled using a range of software products. The discussion in this chapter uses the FCAPS areas as the pillars that support and logically separate the subject treatment. As usual, to illustrate principles, we liberally dip into the areas of device technology, MPLS, ATM, FR, and so on. There are many possible solutions to NMS development—this chapter describes just one possible structure. The following NMS software areas (and subareas) are described:

- Server-side components
- Network-receiving asynchronous
- Network-receiving synchronous
- Network-sending
- Database access
- Client-side components
- Middleware components
- Data representation, such as XML
- Northbound interface (NBI)

This rounds out the theoretical and practical discussion of NMS in preparation for the case study in [Chapter 8](#), "Case Study: MPLS Network Management." Some of the above are illustrated in [Figure 6-1](#); the rest are described later in the chapter.

Figure 6-1. NMS components and data flows.





The out-of-band channel is noteworthy because it allows for network management traffic to use a separate channel from the one used for data (conceptually similar to the way signaling is implemented in SS7 networks). This helps avoid the twin problems of:

- Network management traffic causing congestion in the service network
- Service traffic congestion starving off the management channel

Server components are the prime movers or workhorses in an NMS, performing the bulk of the workload. Typical servers provide the following functions:

- Servicing client user requests
- Issuing provisioning operations, such as writing to agent MIBs (inserting table entries, updating/deleting existing objects)
- Special-purpose listening operations, such as monitoring LSP operational state
- Providing generic services, such as scheduling
- Providing specific services, such as NE firmware and configuration database backup, restore, and distribution
- Handling incoming notifications from the network

All of these server-side functions can result in database access. The database forms the glue that ties together the major components of:

- Clients
- Middleware
- Servers
- NEs

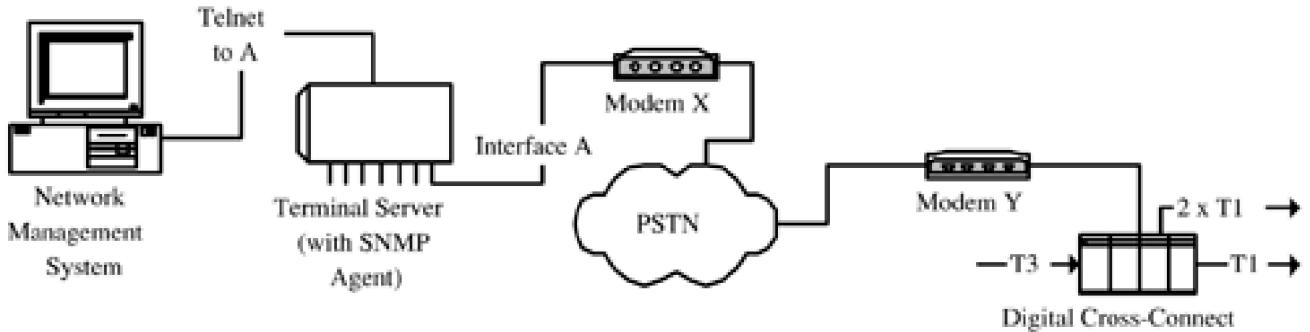
Thin clients tend not to use the database directly and instead rely on the servers to manage the database, for example,

- Recording client-initiated operations, such as creating FR or ATM virtual circuits
- Storing the details of scheduled operations and associated results

As we saw in previous chapters, thin clients can be based on standard Web browsers. Since there can be many such clients (potentially hundreds for large networks), where to carry out the bulk of the processing is an important design decision. If the principal requirement for client software is fast execution, then as much as possible of the MIB and database access should be carried out by the client rather than the server. This is a good way of offloading server capacity, but it does require more secure clients. Similarly, if the client software is required to be simple and intuitive to use, then it should be designed to be as generic as possible. Generic software hides complex

network data as much as possible and presents simple visual objects providing default values where appropriate. An example of this is terminal-server interface configuration (mentioned in passing in [Chapter 1](#), "Large Enterprise Networks"), as illustrated in [Figure 6-2](#).

Figure 6-2. Terminal-server interface management.



Let's assume the user wants to access the text-based menu system provided by the digital cross-connect on a remote site. A digital cross-connect is a device that allows incoming TDM channels to be groomed into higher or lower bandwidth circuits. [Figure 6-2](#) illustrates an incoming T3 out of which a T1 and two concatenated T1s are extracted and transmitted onward in another direction. In this example, the digital cross-connect is a legacy NE that provides a simple serial interface for management using a text menu system rather than SNMP. It can be reached via modem X connected to Interface A on the local terminal server. The user connects to Interface A using telnet and can then start sending commands to modem X, for instance, dialing out to modem Y. However, before Interface A can be used, it must be configured. Some terminal servers allow the use of SNMP to set and get the configuration of their serial interfaces. So, let's assume the user wants to configure Interface A. Typically, this involves setting the MIB object values for:

- Bit rate
- Parity
- Number of data bits
- Number of start bits
- Number of stop bits, and so on

To facilitate this, the NMS should present the user with a dialog that is as generic as possible and provides defaults for most of the above objects. The user can then apply the required changes and set the appropriate row in the terminal server agent MIB. If other interface types (beyond the serial variety) are set using the NMS, then the software should try to make them all look as similar as possible. This example also illustrates the way a flexible NMS, by incorporating extra infrastructure like terminal servers, can add value to (and extend the lifespan of) NEs that do not themselves host facilities like SNMP agents.

Middleware components provide a convenient means by which clients can communicate with server software. CORBA-based products are an example of middleware technology, though there are others, such as Java RMI, RPC, and even Java 2 Enterprise Edition (J2EE). Very often, client software may consist of Web browser-based Java applets. This approach leverages desktop COTS software (i.e., browsers) and the Java programming language. On the other hand, client software may consist of full-featured C/C++ standalone applications.

We now start the discussion of the FCAPS server components with the Fault Server.

Fault Server

The purpose of the Fault Server is to process NE notifications. It faces into the network and seeks to maintain parity between the NMS picture of network faults and the real situation in the network. In many ways this is the most critical element of an NMS because it ultimately determines if real problems exist. A Fault Server will generally provide the following features:

- Listening for notifications
- Determining the underlying problem (root-cause analysis)
- Updating persistent repositories and any GUI visual indicators

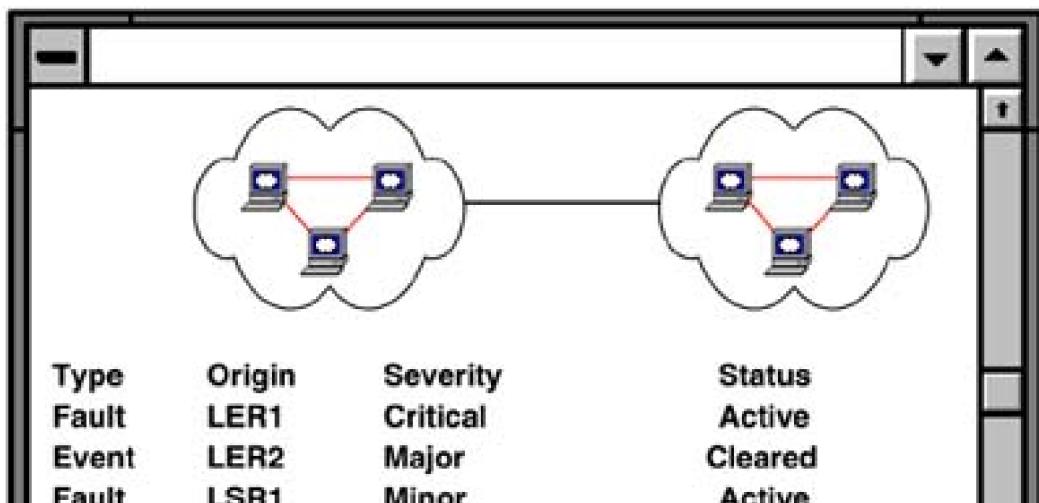
The first of these is straightforward enough: Notifications from the network are received, processed, and stored. Getting to the root of the associated problem may not be so simple and is often called root-cause analysis. This is the way in which notifications are analyzed and processed to determine what exactly is causing the problem. An example of this is a steady stream of link-up and link-down notifications that involve the same link. Without root-cause analysis, the NMS will process each notification, attempt to reflect it in the fault database, and propagate it to the other applications. This is an expensive operation, particularly when the actual problem may be due to a faulty interface on one end of the link. Root-cause analysis would attempt to examine the notifications and apply such reasoning. The best action might be to disable the link in question and investigate why the interface is faulty. In effect, root-cause analysis tries to impose semantics upon the data arriving from the network. A point about this description of multiple notifications and the recommended remedial action is that it would help in reducing the number of notifications received.

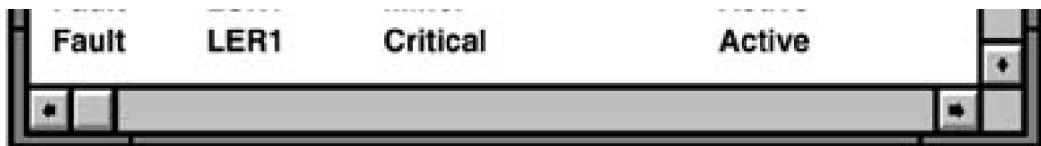
Once the Fault Server has determined that a real problem has been identified, it must then record it. This consists of at least two actions:

- Updating the database
- Updating registered clients

Database update takes the form of either inserting a new record or updating an existing record in a fault table. Updating registered clients consists of ensuring that the new fault is propagated to any users viewing network faults. This can take the form of a topology section changing color and/or a new entry appearing in a fault listing. This is illustrated in [Figure 6-3](#) with a combined topology view and a fault listing. As faults occur on the network, they appear in the listing window and in the topology view. Some systems may also provide a geographical map background for the topology.

Figure 6-3. Client topology view with a fault listing.

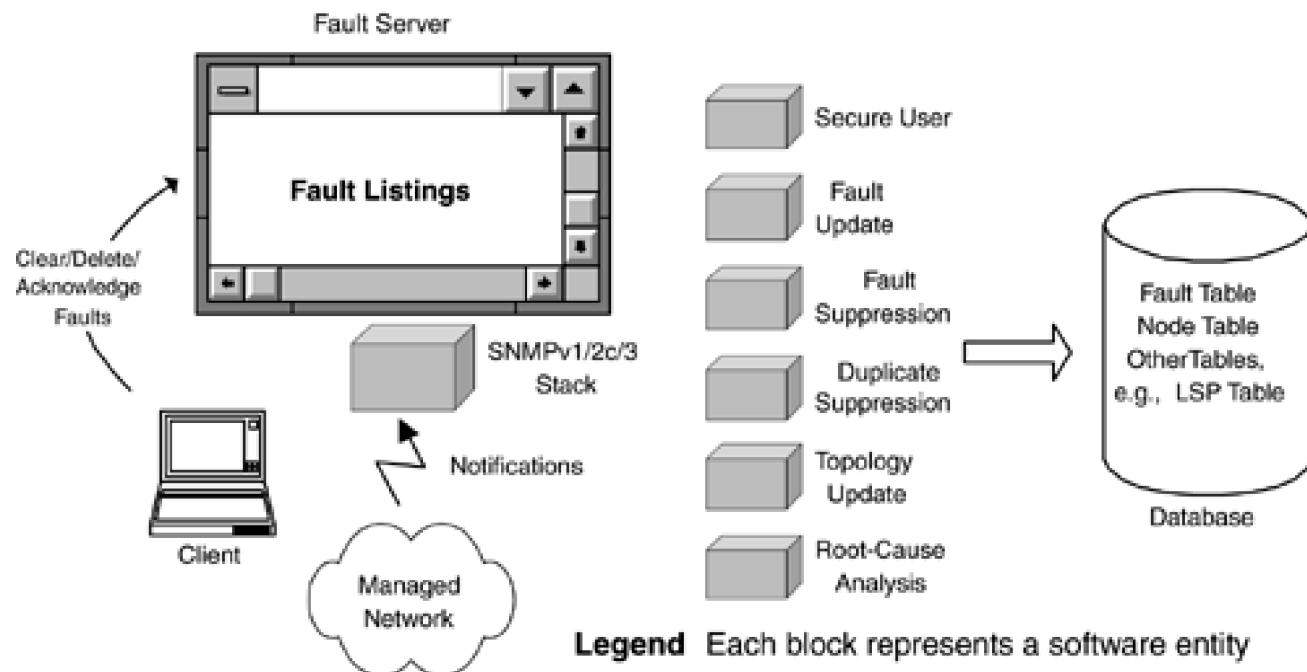




As we've observed before, the faster the latter occurs, the better. This last task can be nontrivial (involving temporarily locking out some database changes) if many clients are viewing the faults.

Figure 6-4 illustrates a possible Fault Server with its constituent software components.

Figure 6-4. Fault Server components.



Notable items in Figure 6-4 are:

- A multilingual SNMP stack
- Various (possibly optional) server application components and features
- Specific database tables for use in fault processing

The SNMP stack component is capable of accepting:

- SNMPv1 traps
- SNMPv3 notifications

Incoming messages pass into the SNMP stack, which listens on port 162 for all such messages. Once received, messages are processed by the stack and passed upwards into the Fault Server. If the fault is new—for example, if it indicates that LSP x has become operational—then a new entry is inserted in the fault table to this effect. Other affected tables may also be updated at this point, in this case, the LSP table, as illustrated in Figure 6-4. Alternatively, a message could be sent to the monitoring server (described later) expediting rediscovery of LSP x. If the LSP has become operational, then it is ready to receive IP traffic—this can be communicated to an external application that wishes to send IP traffic. Alternatively, once the LSP is operational, the IP traffic may start to flow across it immediately with no need for external communication. So, the simple case of an LSP becoming operational can result in IP traffic landing at our MPLS network boundary. This in turn can result in the sender being billed for the MPLS resources used.

The preceding discussion again shows the way the different areas of network management are often inextricably interwoven. All of our examples are carefully chosen to illustrate simple network changes and the aftereffects; real networks may present hundreds of

distributed changes occurring in quick succession. The NMS in general struggles to keep up. One additional point about this is that a change in LSP operating status from down to up is perhaps not exactly a "fault"; it is more of an event. For this discussion, we assume that faults and events are treated essentially in the same way even though in practice this is likely not to be the case.

Two other items in [Figure 6-4](#) relate to fault and duplicate suppression. During certain periods of reconfiguration or fault, the operator may wish to inhibit processing of notifications. This is in order to avoid overwhelming the NMS or filling up database fault tables. Also, if a given fault is recurring at an unreasonable rate—for instance, a given pair of link up/down notifications—then it may be desirable to not process (i.e., to suppress) the faults until the problem is resolved.

Another class of fault is the paired variety in which only two states are possible, such as a power supply that is either outside or inside its allowed operating temperature range. If a fault occurs to indicate that a power supply has exceeded its allowed temperature range, then when the device returns to the normal operating range, a second fault should be issued. This second fault should clear the first one—there should not be two unrelated faults.

Fault Server Database Tables

Basic fault storage can take the form of one or more relational tables keyed by node ID (unique number attached to every node in the NMS). Examples of columns in a fault table are:

- Node ID (the key)
- Description: A text string embedded in the notification explaining the fault
- Origin: The originating NE (processor, card, fabric, etc.) for the fault
- Status: Active, cleared, acknowledged (the user knows about the fault but has not cleared it)
- Color: Red for active, blue for acknowledged, green for clear

As described in the previous section, rows containing all of the above columns are created or updated as incoming faults are processed.

Fault Server Software Structure

The software components in [Figure 6-4](#) can be viewed as separate modules all using the central database. The main Fault Server component is the engine that processes incoming notifications. It stores faults in the database and signals to the rest of the NMS using the other components, such as topology update. The Fault Server can be hosted on its own platform (e.g., a Solaris, HP-UX, Windows 2000 system) or shared with other FCAPS servers.

Topology Update

Clients looking at faults generally want to see new notifications propagated into their views as quickly as possible. As mentioned earlier, there are many ways of achieving this:

- CORBA
- J2EE
- Java RMI

- RPC
- Database update

In order to use a CORBA-based solution, the Fault Server can simply alert registered clients by calling an appropriate remote object method. The topology update object can reside on each registered client and provide a method (or function call) called by the Fault Server to indicate the arrival of new faults. The client can then synchronize with the database for the new faults. It is even possible for the faults to be provided as parameters in the object method. Similar facilities can be provided using J2EE if, as may be the case with CORBA-based systems, there is no need to bridge different programming languages and environments. Java RMI and RPC provide a lower level interface for achieving remote synchronization. Alternatively, the clients can be relied upon to regularly poll the database for newly updated and inserted faults.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Configuration Server

The purpose of the Configuration Server is to execute client-initiated directives made against NEs. Like the Fault Server, it also faces into the network but operates in a less open-ended way because it is not required to process asynchronous NE-originated notifications. However, the complexity of the Configuration Server lies in the way it can both write to and read from the network. As we'll see, the bidirectional nature of this SNMP traffic puts additional demands on the Configuration Server software.

Many NMS do not provide a configuration feature and instead restrict their baseline functions to fault management and discovery. A full-featured NMS will include a Configuration Server as a key component.

Let's assume that a client user creates an LSP as was shown in [Chapter 2](#), "SNMPv3 and Network Management," [Figure 2-7](#). As we've seen, this involves the creation of an entry in the MPLS tunnel table ([Figure 2-5](#)). Depending on the type of LSP, this may require other tables to be updated as well, but for simplicity let's assume that the LSP is:

- Signaled
- Best-effort
- Unidirectional

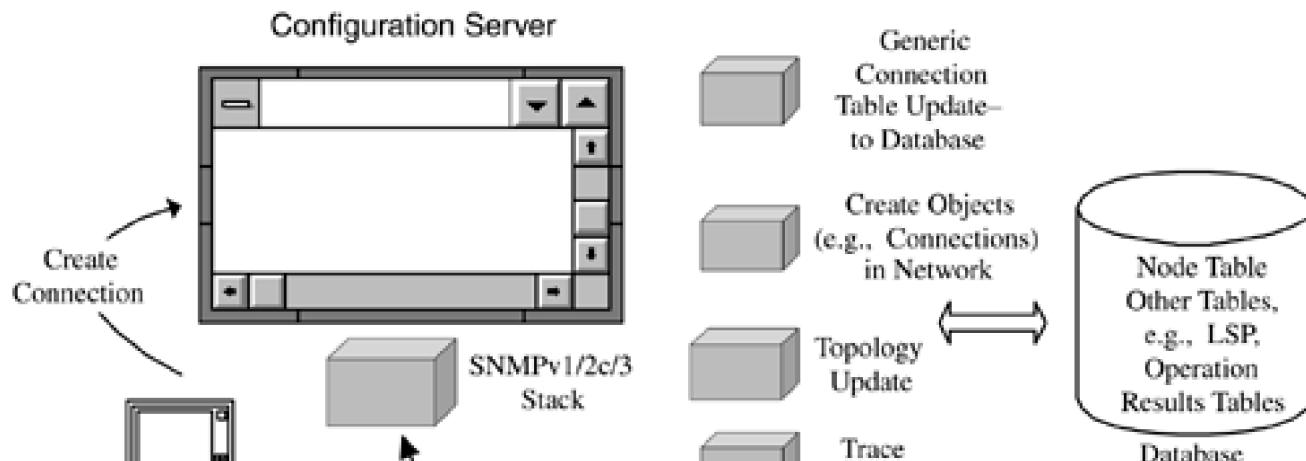
We also assume that no route object is needed because the ingress LER computes the required path. So, the NMS must create a new row in the LER1 MIB MPLS tunnel table. The user specifies the required data for this row, as shown in [Table 6-1](#).

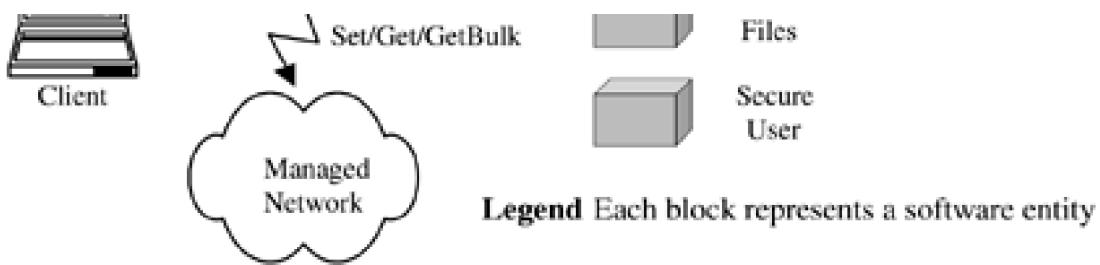
Table 6-1. LSP Configuration Data

ORIGIN	DESTINATION	SIGNALING PROTOCOL	REQUIRED QOS	EXPLICIT ROUTE OBJECT
LER1	LER2	LDP	Best-effort	None

This data is written to the database and submitted to the Configuration Server as a type of job, and from there it must be translated into a form suitable for the MIB of LER1. The Configuration Server must therefore translate the [Table 6-1](#) items into one or more SNMP `setRequest` messages. One possible structure for a Configuration Server is illustrated in [Figure 6-5](#).

Figure 6-5. Configuration Server components.





Secure User

Since the end user can access NE data, it is essential that security is in place. This may be required if many users can remotely connect to the Configuration Server. For SNMPv1/v2c, the security amounts to read and write community strings. For secure SNMP operations (version 3), the user may be required to supply security credentials (authentication and encryption) if these are not automatically supplied by the NMS. For SNMPv3, the security settings can be:

- No authentication and no encryption
- Authentication and no encryption
- Authentication and encryption

If authentication, encryption, or both are specified, then the client user may be requested to supply corresponding passwords. In addition, if authentication has been specified, then it is necessary to indicate the required authentication protocol: MD5 or SHA1.

The user may also be required to specify the SNMP timeout value (e.g., 5 seconds), the number of retries, and the port numbers to use (normally 161 for **sets/gets** and 162 for notifications).

Securing the user (i.e., ensuring that the user has security clearance to execute the required operations) can be achieved in two steps:

- The NMS imposes security against the client requests (i.e., user name, passwords, etc.).
- The NEs impose SNMP security against the data sent by the NMS.

After this, the network has been secured against an insecure user.

Trace Files

During software development, trace files are an indispensable means of tracking software execution paths. They can help in locating problems such as:

- Software bugs
- SNMP timeouts, such as a third-party NE that has a slightly slow (or heavily loaded) agent
- Bad values in MIB operations, such as trying to write an illegal value to a MIB object

It is very useful to be able to switch tracing facilities on and off even in deployed systems.

Generic Connection Table Update

The need for generic GUI components was mentioned earlier in the chapter in relation to terminal servers. Keeping connection types as generic as possible helps simplify the NMS. The front end presents similar screens for multiple connection types, such as:

- ATM virtual connections (PVX and SPVX)
- MPLS LSPs (signaled and unsignaled)
- FR cross connections into an MPLS core
- SONET paths

The user is allowed to select the endpoints for the connection, resources needed, and the route to take, and this data is then written to the database. In this way, there is a complete logical separation between the GUI and the provisioning backend (described next).

Create Network Objects

Once the requisite connection objects have been stored in the database, they must be written to the network. For signaled connections (ATM or MPLS), this may require just writing to the MIB of the originating node. This is relatively simple. For unsignaled connections (e.g., ATM PVX), the provisioning code may have to write data to each node in the path. This is a more complex exercise, particularly if errors occur. The latter raises difficult questions: Should the entire operation be aborted and rolled back, or should the partial data be left on the network and the user notified? One approach is to leave the network clean (roll back any MIB sets if all operations do not succeed) and flag the problem to the user visually and in a log file.

Topology Update

As for the Fault Server, many configuration changes will be of interest to clients, and again it may be necessary for a topology update to occur after important changes such as:

- Changing the administrative status of a connection from up to down
- Creating a new LSP
- Deleting an existing LSP

These changes are made in the central database and then applied to the network. Any registered, viewing clients will subsequently see the changes reflected in their topology GUI.

Configuration Server Database Tables

Typical tables used by a Configuration Server can be

- Generic connection tables: These contain data relevant to all connection types keyed by index value or origination/destination node IDs.
- Technology-specific connection tables: These contain data relevant to specific connection types, such as ATM PVX and LSPs.
- Operations log tables: These are for recording all configuration changes.
- Operations result log tables: These are for recording all configuration change results.

The generic connection tables may be split into a number of technology-specific sub-tables. Common elements of all connection types (e.g., source and destination nodes, resources used) can be stored in one table, while the technology-specific settings are stored in other tables. These tables are updated as configuration changes occur.

Configuration Server MIB Support

Since the Configuration Server interacts directly with NE MIBs, it must support possibly many different versions of the same MIB and a variety of other MIBs. This should be transparent to the end user.

Issues such as MIB holes should also be handled as transparently as possible. A MIB hole occurs when a given column in a table has no value, as illustrated in the extract from the MPLS tunnel table shown in [Table 6-2](#).

Table 6-2. MPLS Tunnel Table Extract with a MIB Hole

MPLS TUNNELINDEX	MPLS TUNNELSIGNALLINGPROTO	MPLS TUNNELSETUPPRIORITY	MPLS TUNNELHOLDINGPRIORITY
1	3	0	0
2	2		1
3	1	5	7

Before we describe the MIB hole, let's briefly review the way MIB object instances are accessed. Scalar objects have only a single instance value within a MIB. The object instance of a scalar is distinguished from the underlying object type by appending a zero to the OID. MIB table object instances, however, are accessed by appending the index value.

The columns in [Table 6-2](#) are referenced by using a notation made up of the value of the `mplsTunnelIndex` and the column name; for example, `mplsTunnelSignallingProto.3` has the value 1. We'll see more of this type of indexing in [Chapter 8](#).

The shaded entry in [Table 6-2](#) is a MIB hole. A `get-next` request on the object `mplsTunnelSetupPriority.1` will return the value 5, that is, the value of `mplsTunnelSetupPriority.3`. If the request is part of an NMS NE MIB query, then this may not be what was intended. It is up to the NMS software to either return, say, -1 or to give some indication that a hole was found. Also, a `get` request on `mplsTunnelSetupPriority.2` will result in a No Such Name exception. Holes can cause similar problems when trying to perform row-based operations.

The details of avoiding problems with MIB holes and retrying failed operations (e.g., due to agent timeouts) are all buried in the Configuration Server software. Keeping clear (layered) lines of demarcation between the various technologies helps to maintain a degree of simplicity in the software. In other words, it is generally a good idea to keep issues relating to SNMP access out of application code. This helps reduce clutter in the latter by isolating complexities such as retries in the event of timeouts or other exceptions. A clean API into the SNMP access code can help fulfill this need.

Configuration Server Software Structure

The Configuration Server can be hosted on its own platform (e.g., a Solaris, HP-UX, Windows 2000 system) or on a system shared with other FCAPS servers. The principal challenges facing the designers of a Configuration Server are as follows:

- Supporting many clients: This may give rise to multiple simultaneous NE operations. Not all NEs support multithreading, so the NMS may have to implement queuing.
- Supporting many MIBs and different versions of the same MIB.
- Keeping the different technologies separate—SNMP access code, database access code, provisioning code, and so on. SNMP access code makes calls into an SNMP API (e.g., Java, Visual C++) that ultimately results in sending messages to and receiving messages from NEs. Database access code makes calls into a database API (e.g., Java JDBC). Provisioning code is concerned with the reading and writing of data to and from NEs. All three of these layers tend to have interactions, so clear separation is important (using APIs) to cater for software upgrades (e.g., SNMPv1 upgraded to SNMPv3).
- Providing meaningful operations results.
- Bulk facilities for rapidly bringing up or down a large network.

Unlike the Fault Server, the Configuration Server does not generally have to cope with persistently high levels of (unsolicited) NE-originated traffic. However, during periods of high client activity (e.g., 30 operators each bulk-creating 100 PVX circuits that span the network), the Configuration Server must be able to withstand bursts of incoming traffic resulting directly from the provisioning actions. The latter is caused by reads from the NE MIBs and/or from set operation responses.

[\[Team LiB \]](#)

 PREVIOUS  NEXT 

Accounting Server

Accounting and performance software share a number of similarities. The Accounting Server faces into the network and receives data records periodically generated by NEs. Often, the data records are emitted based on a preconfigured time. It is also possible for an Accounting Server to poll MIBs for specific data. Ultimately, accounting data is concerned with billing users for network resource consumption. As usual, our discussion concerns connection-oriented resources, such as:

- ATM/FR virtual circuits
- LSPs

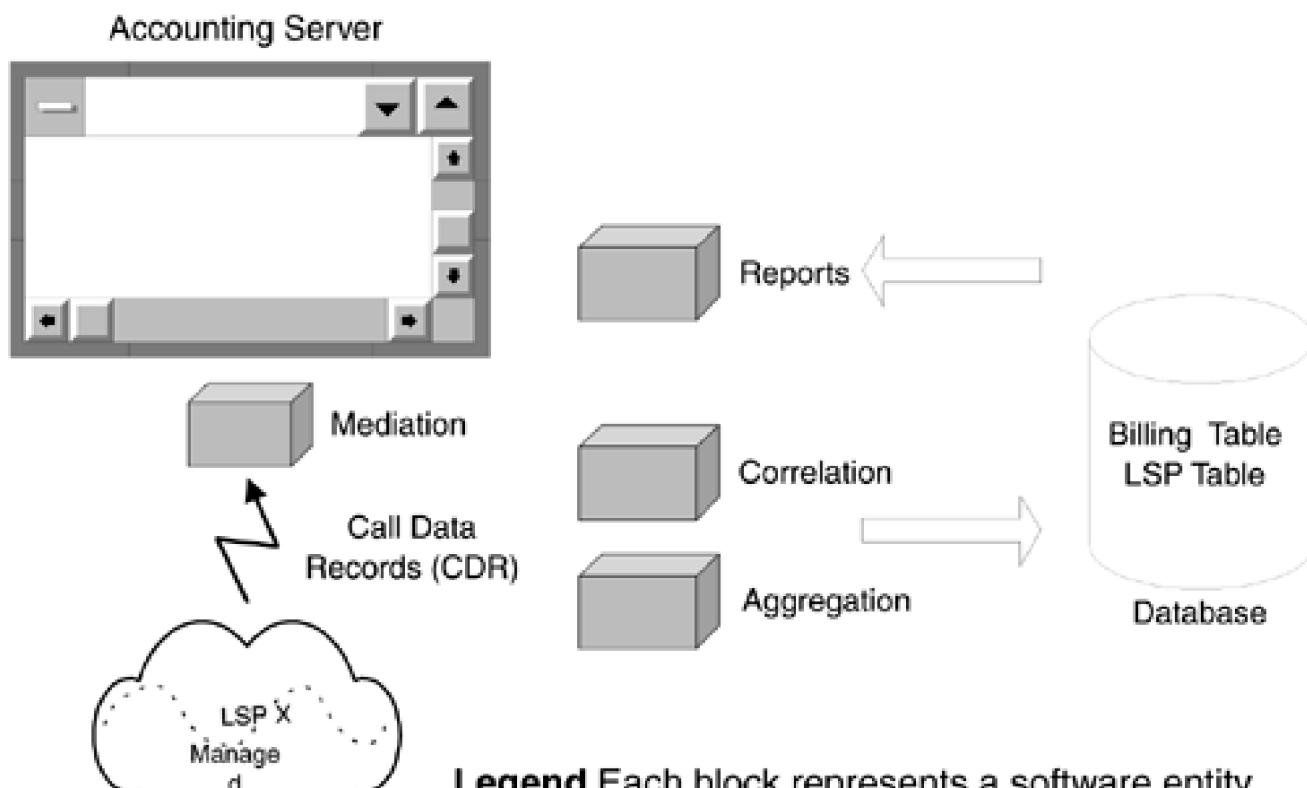
Billing for connection-oriented networks can present a number of difficulties, such as when a connection is rerouted because of a path failure. The original connection is automatically torn down, and a new one is created to replace it (hopefully not losing any data). The Accounting Server must recognize that a new connection is in place and ensure that the billing details for both connections are aggregated. Similar issues affect multicast connections, such as a point-to-multipoint video transmission network—a multiparty connection with billed clients potentially at each endpoint.

There is no reason why users can't be billed for the use of other objects such as:

- CPE routers
- Traffic landed on SP router ingress interfaces

Nodes from some vendors generate call detail records (CDR), as illustrated in [Figure 6-6](#).

Figure 6-6. Accounting Server components.



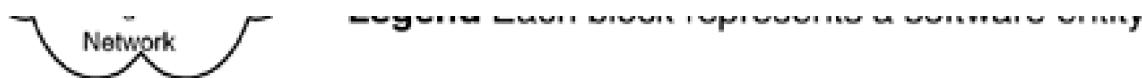


Figure 6-6 illustrates a managed network with one LSP. Let's assume that the LSP traverses five NEs, each of which generates associated CDRs that pass into mediation. Typical details contained in CDRs are:

- Correlation ID: unique large integer for tying together the connection segments
- Originating date and time
- Call type: signaled LSP, ATM/FR PVC
- Call state: starting, in progress, terminated
- Ingress node
- Ingress interface
- Egress node
- Egress interface
- Cells/packets/frames received, transported, and dropped
- Class of service

The purpose of this data is to characterize the network resource consumption made by the underlying connections. If the nodes do not generate such accounting data automatically, then the Accounting Server may have to poll the network for the requisite details.

We assume that CDRs are available and that they are presented to mediation in a proprietary format containing some or all of the above fields.

Mediation

Mediation is the process of analyzing the raw data generated by NEs to produce standard format billing details for downstream use by third-party applications (from organizations such as ACE*COMM). It is not necessary to use standard formats if the billing application is proprietary. However, standard formats have the merit of allowing different third-party applications to be swapped in as required. Mediation may contain two additional steps: aggregation and correlation (described next).

Aggregation

This is the process by which separate CDRs are combined. An example is an ATM PVC that spans a number of NEs (or the rerouted connection example described above). A given virtual circuit has certain performance parameters of interest:

- Number of IP packets transported (if the circuit is an LSP)
- Number of cells transported per second (if the circuit is an ATM connection)
- Number of cells dropped due to excessive input traffic
- Average bandwidth used by the cell traffic
- Number of SLA contract violations

Aggregation merges related data records so that they are tied to one billing entity, such as an LSP or an ATM connection.

Correlation

Correlation is the process of combining multiple units of aggregated data with the details of the ultimate bill recipient, that is, one customer. Let's take an example of an enterprise that has a 5Mbps ATM link to a service provider with one virtual circuit joining the enterprise's CPE to the SP network. Aggregate data on this circuit is collected across the network and then correlated with the associated enterprise user for billing. This can support usage-based billing rather than a flat-rate model. In the usage-based model, the committed traffic dictates the price.

Clearly, aggregation and correlation may be performed in a single step, but we separate them here for clarity. The billing elements can be details such as:

- Number of cells sent to or received from the SP network
- Bandwidth used in transporting the data across the ATM link

These reflect a pay-as-you-go billing model. Correlated data can be saved into the database in readiness for reporting and bill generation.

Reports

Reports are the means by which end users can view the billing details. Examples of reports are:

- Utilization of objects, such as LSPs
- The average and peak numbers of IP packets transported by the LSP
- The bandwidth consumed

Reports may be viewed by the network operator and may be accessible to customers via the Web. Actual bills can be based on report content.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Performance Server

The purpose of the Performance Server is to analyze network data in order to:

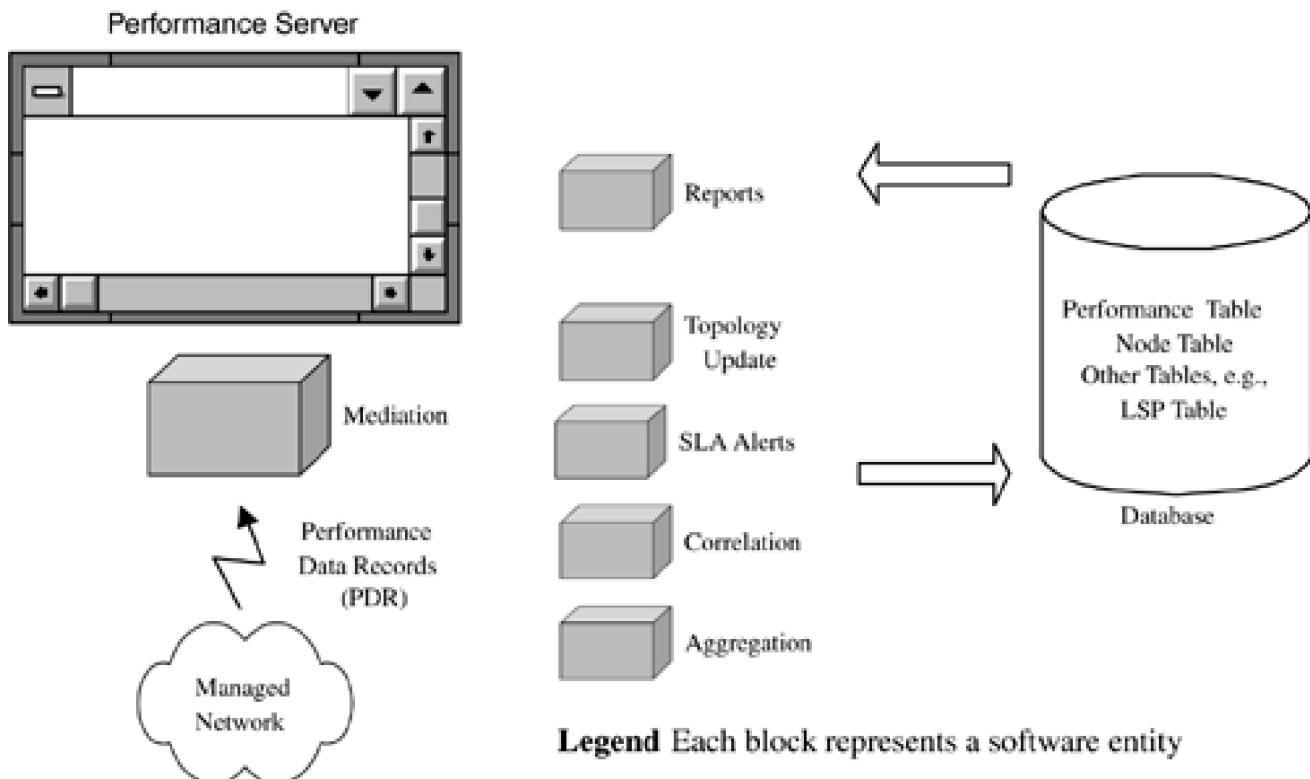
- Determine if problems exist prior to their affecting services
- Maximize network utilization
- Pre-empt the occurrence of congestion
- Demonstrate compliance with agreed SLAs
- Indicate when extra network investment is needed (capacity planning)

The Performance Server faces into the network and receives asynchronously generated NE data. It can also proactively retrieve from MIBs data such as the level of bandwidth consumption. Performance data, for example, the number of IP packets received by an LER, may change very rapidly. For this reason, performance data is often automatically generated and emitted by NEs rather than being polled. The number of IP packets landing at an MPLS LER ingress interface may be hundreds of thousands per second. For data such as this, the NE generally periodically creates a data record and emits it to a listening application (in this case, the Performance Server).

In this discussion, we focus on the data issued by the NEs and the way this is processed and used by the Performance Server. Also, SLAs are introduced.

[Figure 6-7](#) illustrates a possible structure for a Performance Server.

Figure 6-7. Performance Server components.



Mediation

We introduced mediation in this chapter and the previous one in the context of billing. It is a process of analyzing the raw data generated by NEs to produce standard format billing details. Mediation can also be applied to performance data to produce sanitized details for downstream use by third-party applications.

Aggregation

This is the process by which separate performance data records are combined. An example is an ATM SPVCC that spans a number of NEs. A given virtual circuit has certain performance parameters of interest:

- Number of cells transported per second
- Number of cells dropped due to excessive input traffic
- Average bandwidth used by the cell traffic
- Number of contract violations

Similarly, the performance details of other objects may be of interest:

- Nodes
- Interfaces
- Links
- LSPs
- Multiservice cross connections—Ethernet-to-MPLS, FR over ATM, and so on

Aggregation logically joins up the separate performance data records so that the relevant managed objects can be analyzed. An example is a count of all the cells transmitted by an ATM interface; at time T1 the number of cells may be x , and later at time T2 the number of cells may be $x + y$. Aggregation links this data together and stores it.

Correlation

The aggregated data is then correlated with the associated managed objects in readiness for reporting. Examples of correlated data are (for a single customer):

- The number of IP packets forwarded by an LSP end to end
- The number of Ethernet frames forwarded by an LSP end to end
- The number of cells carried by an ATM PVC
- The number of cells dropped by an ATM switch

These data should help give a clear picture of the performance of the underlying objects.

Reports

Reports are the means by which end users can view the performance details. Examples of reports are:

- Utilization of managed objects, such as interfaces, links, nodes, and virtual circuits
- The difference between actual and planned loads
- Real-time and historical SLA conformance

Reports may be viewed by the network operator and may be accessible to customers via the Web. Important aspects of performance reports are:

- Baseline measurements—a realistic snapshot of the performance
- Deviation from the baseline as network changes occur
- Utilization of network resources

Baseline measures are essential for effective performance management. This is a set of readings taken from the network during normal operation. As the network changes—for example, due to link failure, variations in traffic mix, or increased traffic—the baseline changes with it. The extent of the deviation is important for planning changes or additions to the network. It may also help to pinpoint problems before they become service-affecting.

SLA Alerts

It is very important for enterprises to avoid violating SLA terms because there may be financial penalties—particularly when the network management processes have been outsourced. SLA alerts can be issued based on ongoing analysis of trends in an effort to pre-empt violations before they actually occur. An example SLA is illustrated in [Table 6-3](#).

Table 6-3. An IP SLA

Source IP Address	10.81.1.45
Source Port	444
Destination IP Address	10.81.2.89
Destination Port	445
Link Bandwidth	10Mbps
Interpacket Delay	1ms
Ordered Delivery	Yes
Packet Loss	0.0001%
Jitter	No
Round Trip Delay	30ms

This SLA indicates that IP traffic from 10.81.1.45 port 444 will land in the SP network on a 10Mbps link destined for 10.81.2.89 port 445. The interpacket arrival time is specified to be no more than 1 millisecond with no packets arriving out of order. A tunneling technology such as MPLS or L2TP could be employed to achieve the latter. An SLA alert might be raised in the Performance Server if the link bandwidth increased up to or beyond 9.9Mbps.

Topology Update

Performance Server data changes (such as detection of a congested link) will be of interest to clients, and it may be necessary for a topology update to occur after changes such as the following:

- When congestion is imminent on a given link
- When a virtual circuit is being presented with excessive traffic—it may be necessary to add extra bandwidth to the circuit

These are important because, left unattended, they can have a serious impact on the network.

Performance Server Database Tables

Performance data can be related to the associated managed object; for example, there can be separate tables for each of the following:

- Nodes
- Interfaces
- Links
- Virtual connections

Each of these tables can have separate columns for the relevant performance data, such as:

- Number of incoming and outgoing packets, cells, and frames

- Bandwidth in use
- SLA status

The rows and columns of these tables are populated by the components of the Performance Server. Alternatively, the other servers can share the above tables.

[[Team LiB](#)]

 PREVIOUS  NEXT 

Security Server

If there is one area of network management that has moved to the very top of the operator's agenda, it is security. There are many aspects to security provision; for example, IPSec can be deployed to protect the underlying managed network if all nodes implement it. All management traffic on such a managed network is then protected. There are other aspects to securing NMS, and we now study the elements of a Security Server from a number of perspectives:

- Access application: SNMP, telnet, Secure Shell, Web, console (serial port)
- Authentication: Password, community string, Kerberos, user-based security, Remote Access Dial-In User Service (RADIUS)
- Privilege level: Superuser, Read-only, and User
- Permitted views: Specific objects and sources

Each of the above is described in the following sections. Up to this point we have tacitly assumed that all management interaction with NEs go via the NMS. In many cases this is not what may occur in practice as operators use a range of access methods to achieve the following tasks:

- NE configuration using a CLI
- Fault access and analysis

These operations may be made directly on the NEs themselves. There are advantages and disadvantages to the use of direct NE access. However, it should be noted that there is a certain minimum number of steps—such as IP address assignment and SNMP enabling—needed in order to configure an NE. In most cases, this set of steps has to be executed directly on the NE using a serial interface.

Access Applications

Access applications are the software facilities used to gain entry to the NMS. Depending on the design, the user can gain access either by direct connection to the NEs or via the NMS. Configuration using a CLI is a quick and generally easy way of bringing up a network. However, it presents a few security hazards:

- Limited or no logging apart from that provided by the NE or CLI
- Fairly open access to sensitive NE data
- It may be error-prone, and help facilities may be quite limited

On the other hand, configuration carried out via the NMS can be piped to a comprehensive logging facility. Also, scripts can be maintained in the NMS for subsequent reuse (HP OpenView NNM provides this). In addition, a good NMS may provide extensive context-sensitive help or automated assistance to the user.

Some popular access applications are:

- SNMP

- Telnet
- Secure Shell
- Web
- Console (serial port requires an intermediate device such as a terminal server)

The different versions of SNMP provide the means by which an NMS can conveniently access NE MIB objects. Telnet is a simple method of gaining access to the CLI of a given NE. Secure Shell provides a secure method of accessing the NE CLI. Web access uses HTTP (or possibly the secure version of HTTP) for gaining access to a mixed textual and graphical management interface. Console access is essentially the same as telnet except that connection is made directly to a serial interface on the NE. These are typical applications for gaining access to NEs, and in the next section we look at ways in which degrees of security are added to them.

Authentication

Once the access application has been chosen, some means of authenticating (or checking and authorizing) the user must be selected. The most basic level of security is none—that is, the user is given unrestricted access. Moving up the security food chain, passwords provide a first level of protection against unauthorized access. The SNMP community string is essentially a password that has to be supplied in the SNMP messages sent to remote agents. If the community is not correct, then the message is discarded. SNMPv1/2c community strings are clear text embedded in SNMP messages sent over the wire. They are open to interception and therefore don't really provide any protection. Kerberos provides stronger security mechanisms in the form of a secret key network authentication protocol that allows a user to communicate using a DES-encrypted telnet session. The SNMPv3 framework provides the user-based security model (USM). This consists of authentication, privacy, and timeliness (protects against the replaying of a captured message). RADIUS is a client/server protocol for the authentication of users trying to connect to a system via various access applications.

Privilege Levels

Some security schemes provide different levels of authority to users, such as:

- Read-only
- User-level
- Superuser

Read-only access allows only MIB gets; user-level allows **gets** and some **sets**; superusers can **get** and **set** all appropriate objects.

Permitted Views

It may be required to restrict the set of objects accessible to a given user. Two ways of doing this are:

- Access control lists
- Permitted object views

An access control list contains the source IP addresses allowed to connect to an NE. This is similar to the access control lists used in IP

routers. Permitted object views specify a subset of MIB objects accessible to a given NMS user. Both access control lists and permitted object views are stored on NEs. However, the NMS can either retrieve (or discover) them from NEs or provision them in the first place.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

[PREVIOUS] [NEXT]

Other Servers

The FCAPS servers form what may be considered a baseline for NMS products. Some NE and NMS software vendors tend to add extra servers for product differentiation and to enhance NE manageability. We now briefly describe some of these.

Discovery

A Discovery Server exists to keep up with the details of the deployed NEs. Discovery can be considered an extension of our Configuration Server and typically tries to maintain parity between the deployed NEs and the NMS. Discovery may keep track of objects such as the following:

- Nodes
- Interfaces and underlying stacks
- Links
- Virtual connections
- Cross connections between different technologies
- Routing protocols
- Routing tables
- Signaling protocols

Various methods are used for tracking these objects, including:

- ICMP (if the NEs support this)
- SNMP
- Standard and proprietary signaling protocols

Discovery software can make use of both the management and control planes, for example, SNMP and IP routing tables.

Monitoring

A Monitoring Server processes managed objects that change value in some fashion; for example, it monitors interface operational status. This is a facility by which the following can be achieved:

- Rapid updates of discovered topology data
- Expedited rediscovery of managed objects that have changed

Typically, discovered objects are subsequently processed by Monitoring. Consider the case of an LSP initially discovered in the operationally down state. On the next cycle, Monitoring detects no change in the LSP state. Next, the user sets the administrative status of

the LSP from down to up, and this is written to the MIB of the associated NE. Assuming there are no problems with the LSP (NEs are configured correctly, connectivity exists, etc.), the operational state then changes to up. When Monitoring comes around to the node again, it will note that the state has changed and mark up the associated database entry. In this way, Discovery can offload some of its processing. This process may be improved by making it notification-driven. In this, an NE sends a notification, such as a link up: This is processed by the Fault Server that in turn signals Discovery.

NE Software Distribution

NE software changes must be installed on deployed devices. This can be achieved using a variety of methods:

- FTP/TFTP
- Proprietary download mechanisms
- Using an NMS

Some NMS provide a means of distributing NE software. Typically, this can involve a number of steps:

- Preparing the NE for a new firmware load
- Rerouting traffic around any nodes to which downloads are pending
- Initiating the transfer
- Handling rollback if the transfer fails
- Verifying the transfer succeeded
- Starting up the new NE software (rebooting, copying the image from FLASH to RAM, etc.)

Preparing the NE may involve issues like:

- Bringing the NE into a quiescent state
- Closing down existing connections
- Ensuring that no resources are in use on the NE
- Determining the available FLASH and RAM free space
- Taking a copy of the existing firmware

Once the NE is ready, the transfer can start. Errors—for example, a loss of connectivity or an image that has become bad and will not allow a successful restart—must be catered to both during and after a transfer. The latter may require manual intervention. Rollback of the previous version is important for getting back to a working firmware build. Some NEs may provide sufficient FLASH to store the old image—this can be deleted if required, or left onboard. In all cases, the NMS should record the detailed results of the software distribution operation, for example,

- New version of NE software
- Old version of NE software

- Is old version still on NE?
- Was upgrade successful?
- Was rollback necessary?
- Was rollback successful?

An important point about this is that an NMS may carry out multiple simultaneous software distribution operations. This might be required for upgrading a large network, and it makes the operation quite complex.

NE Configuration Database Backup and Restore

Just as for firmware distribution, NE configuration data is increasingly important, particularly as device complexity increases. Some reasons for the importance of backing up configuration data are:

- New firmware builds may upgrade the configuration, making rollback difficult
- Disaster recovery
- Creating mirror networks
- Using a given configuration as a template

Probably the most critical of these is disaster recovery in which it is required to completely reinstate the configuration of one NE or even an entire network. Typically, the user would like to be able to use a GUI to point and click certain nodes and restore them. The NMS should handle the complexities of:

- Knowing where the appropriate configuration data files are located
- Handling the transfer via FTP, TFTP, and so on
- Restarting the NEs or reloading the data files
- Informing the operator when the operation is complete
- Rerouting traffic around the nodes being restored

The guiding principle here is that the NMS should help the operator to quickly and efficiently execute the task.

NMS Database Backup, Restore, and Upgrade

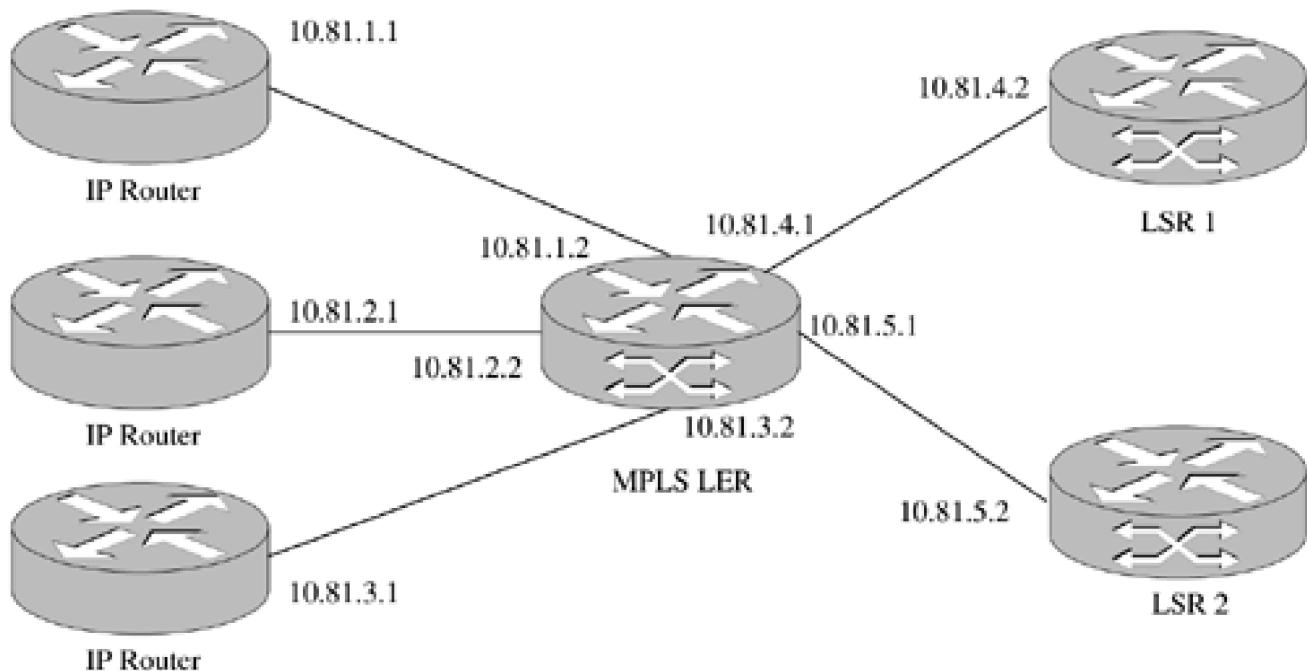
The overall NMS database is a high-value item. It requires regular maintenance, and two very important operations are backup and restore. Normally, the database is a third-party engine of some kind, such as Informix, Oracle, or SQL Server. Either the vendor tools can be used for the database maintenance or software can be written specially for this purpose.

Network operators don't like losing data, so when the network management software is updated, it is important that existing data is retained. In other words, upgrade is an important aspect of software releases. The NMS software vendor must take great care to maintain an upgrade path for user data.

Configuring NEs

The increasing complexity of NEs makes their configuration more difficult as technologies are added and entire networks are compressed into them. An MPLS LER will generally be connected to a minimum of two technologies: IP and MPLS. It may also be connected to a number of additional technologies, such as FR, SONET, or dense wavelength division multiplexing (DWDM). In this example we assume just IP and MPLS. Let's assume that our LER has incoming links from three high-speed IP routers and outgoing links to two MPLS LSRs, as shown in [Figure 6-8](#).

Figure 6-8. The problem of configuring multiservice networks.



Configuring the MPLS nodes in this network typically consists of the following steps:

- Assigning the interface IP addresses
- Assigning the neighbor IP addresses
- Assigning the interface subnet masks
- Configuring signaling protocols
- Configuring routing protocols
- Creating routing areas, levels, peers, and so on

These steps may be necessary for each of the MPLS nodes in [Figure 6-8](#).

In many cases, configuration of networks is done using CLI scripts. An NMS can achieve this relatively easily by the use of software-driven wizards. This applies particularly to the case where IP addresses are assigned in blocks (as is the case in [Figure 6-8](#)). Starting with the LER, we can assign seed addresses and just decrement in order to calculate the neighbor. The commands to set up the various protocols and routing objects can also be executed by the Configuration Server.

As networks grow, the problem of reconfiguration can be difficult. Renumbering large IP networks with thousands of nodes can be a tedious and error-prone task. By maintaining configuration in the NMS, the reconfiguration becomes a software function into which an

operator feeds data.

Middleware

Middleware is that part of an NMS that allows communication between the clients and servers. There is a broad range of software technology choices for achieving this, including RPC, Java RMI, CORBA, J2EE, and Microsoft .NET.

RPC and RMI allow the realization of simple distributed objects in a fairly low-level fashion. The developer must understand a lot about the mechanisms in order to use these facilities. However, they are very powerful. CORBA offers a more abstract interface and allows code in multiple languages to use a shared set of objects on any platform that has an Object Request Broker (ORB) available. CORBA provides a migration path—a compelling reason for organizations using legacy languages to continue doing so. In other words, CORBA products can help to defer the decision to invest in a new and perhaps relatively unproven technology such as (possibly) J2EE and Microsoft .NET. However, for NMS, CORBA may have issues in getting past firewalls.

Data Representation

XML is an increasingly widely deployed metalanguage for data representation. It can be used to encode any type of data in a platform-independent fashion, including:

- Multivendor MIB objects
- Fault MIB objects

XML is used in the OpenNMS product mentioned earlier. SOAP is a relatively new data-exchange and messaging framework based on XML over HTTP. HTTP is not a high-performance data protocol, but it can be used as an NMS client/server transport protocol. It also has the merit of not being blocked by firewalls. It is likely that Simple Object Access Protocol (SOAP) will come to have an important bearing on NMS technology, particularly as the concepts of Web Services permeate the area.

Northbound Interface

Integration between an OSS and an NMS is facilitated with an NBI [[Telcordia](#)]. There are many possible forms for this software interface:

- IDL
- TL1
- SNMP

The importance of the NBI cannot be overstated. It allows for data and commands and responses to flow between an OSS and the underlying NMS. A fully instrumented NBI allows an OSS to automate all the exposed major features of the NMS, including:

- Connection management: creation, modification, and deletion
- Route object management: MPLS EROs, and ATM DTLs
- Fault management: retrieval, and clearing
- Resource block management: creation, modification, and deletion of MPLS resources

Somewhat confusingly, the Telcordia Web site refers to the latter as an EMS when in fact it is generally an NMS because it has a broader context than just one NE (as is the case with an EMS).

The Trend Towards Java-Based NMS

The use of Web browser-based clients has enhanced the importance of Java [[JavaDev](#)] for NMS development. There are several reasons for this; for example, many browsers support a Java Runtime Environment by default. This allows the execution of Java applets during normal browsing. In other words, you can run Java code by interaction with Web site pages, such as pressing buttons and filling out and submitting forms, This capability fulfils many of the NMS client-side requirements. Java is:

- Platform-independent—runs on any platform with a Java Virtual Machine (JVM)
- Browser-enabled
- Object oriented
- Able to support field-replaceable packages
- Secure
- Reasonably good in terms of performance

Java code runs as JVM bytecodes that are interpreted in the JVM, (Just-In-Time) JIT compiled, or compiled entirely into native code. Perhaps the most compelling reasons for the use of Java is its combination of platform-independence and security.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Summary

The implementation of NMS software can take the form of servers. These are high-performance software objects that can support interactions with both external clients and NEs. It is essential that servers are resilient and designed so that they are unlikely to fail except in exceptional circumstances. They form the intermediate layer through which end users can securely communicate with NEs.

We reviewed a variety of such servers that implement the FCAPS areas of network management. We also discussed other servers, used for tasks such as firmware distribution, that can enhance the experience of managing networks.

The need for generic software components is growing with the increasing deployment of dense, multiservice NEs. Generic software attempts to abstract complex NE data as much as possible and present simple GUIs applicable across a broad range of devices. An example was briefly described of terminal-server interface configuration, and multiservice switch configuration was described as well.

Software to implement the FCAPS areas generally does not consist of standalone, isolated components. Instead, there is often a need for server interaction; for instance, a faulty link may be of interest to the Performance Server because it directly impacts an SLA. All the servers make use of the high-performance data storage and retrieval services of a central database engine product such as Oracle or Informix.

On the client side, GUI views are often depicted as network topologies accompanied by fault listings. It is a major challenge for the NMS software to keep these views synchronized with the network. It is always hard to escape from legacy NEs, and for this reason it is often necessary for server components to be SNMP multilingual, that is, able to use any of SNMPv1/2c/3. Security is an increasingly important aspect of managing networks, and it is essential that NMS accommodate this trend. It is possible that security considerations may even have a bearing on the way networks are configured and commissioned—it may be required in the future to set up networks *only* via the NMS rather than via a CLI. Depending on the implementation, this could:

- Help the operator achieve the configuration task via automatic help facilities
- Facilitate bulk configuration, such as of SNMPv3 security settings
- Enhance security
- Provide audit trails of commands sent to the NEs

However, the NMS and its underlying host must be secure. The choice of middleware is important, dictated by needs such as:

- Multilanguage support (not always required)
- Multiplatform support (Java provides this)
- Microsoft solution (.NET provides this)

XML provides an increasingly popular data facility. The NBI is a crucial component of an NMS because it allows for automation. Finally, Java offers many advantages, and it is interesting to note that its use for NMS development is growing. The platform-independent security model offered as part of Java may be one of the most compelling reasons for its adoption.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Chapter 7. Rudimentary NMS Software Components

It's one thing to talk about NMS components; it's quite another to actually build them. In this chapter we see how to build and use SNMP code components. Using some custom software in conjunction with two commercial SNMP APIs, we provide some low-level elements of an NMS. The software is far from production standard (improvements are suggested later) and was written purely as a learning aid to show some of the nuts and bolts of SNMP. Two example programs are provided: one using Microsoft Visual C++ and the other using the Sun Microsystems Java Development Management Kit (JDMK) version 4.2. It is interesting to compare the two APIs from the viewpoint of the SNMP versions supported, platform-independence, speed, and other features. Both programs are driven from user-supplied command-line parameters, and a set of DOS batch files is provided to facilitate ease of use.

The first item of prototype software we look at is a small program called **snmpmgr.exe**, written using the Microsoft Visual C++ V6.0 SNMPv1 API. This program provides the following services:

- Sends a **GetRequest/SetRequest** to a specified SNMPv1 agent
- Receives a **GetResponse** from the specified SNMPv1 agent
- Performs a walk of a specified agent MIB table
- Receives a Trap PDU from a specified SNMPv1 agent

The program is written so that it can be run from the command line with the user specifying:

- The remote SNMP agent IP address or DNS name
- The remote SNMP agent community name
- The required MIB object identifier instance
- The value (for **SET** operations only) of the object instance

To receive traps, the user just presents the word **TRAP** on the program command line. This registers the program to receive traps from an agent. The agent in question must be configured to send its traps to the host running the program. The configuration steps will be described.

The ability to run from the command line allows for the program to be incorporated into a batch file (i.e., a script file) so that many SNMP-based operations can be executed.

We hope that the source code examples will help to solidify some of the concepts introduced in the book so far. In order not to restrict the user to a PC (running Microsoft Windows) platform, we also provide a short sample of similar capability using the Sun Microsystems JDMK toolkit. Because the latter is built using Java, it has multiplatform capability. A noteworthy aspect of JDMK is the brevity of the example program.

All source code provided in this chapter is available for download on the Prentice Hall Web site: [PrenticeHall] in the location provided in the Preface.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Building a Rudimentary Management System

An NMS consists of a minimum of the following items:

- SNMP manager
- MIBs
- Database
- SNMP agents

We now focus on building the bare essentials of the first component only, the SNMP manager. This manager can target specific SNMP agents on any IP-reachable (and SNMP-reachable) machine for which the appropriate community rights exist. MIB objects are accessible to the manager by specifying an OID on the program command line. The program we construct in this section can be seen as the first step to producing a more complete NMS. For this reason, we don't provide a database facility but instead use the program command line as our data source. Similarly, the data sink is just the console output rather than a GUI or database. The examples use the standard SNMP services provided with the Windows NT/2000 platforms.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

Configuring SNMP on Windows NT/2000

The Windows NT/2000 product line includes an SNMPv1 agent as a service. This allows the user to switch on/off and configure the operation of the agent via the Windows GUI. We will configure the SNMP Service so that the public community name rights are read-write. This allows for SNMP **SET** operations to be performed against the agent. On NT, please follow these steps:

- Click on the desktop Start button, then click on Settings
- Select Control Panel
- Select Services and scroll down to the SNMP Service

The rights for the community name can be changed from here. If the SNMP Service is not present, then it can be added as follows: double-click the Control Panel Network icon and select the Services tab. Next, click the Add button, and then scroll down to the SNMP Service item. Click OK to add the service—it may be necessary to insert the NT/2000 installation CD.

On Windows 2000, the Services are accessed by double-clicking the Control Panel Administrative Tools icon. Double-click the Services icon, then right-click the SNMP Service and select the Properties option; this allows for the community rights to be set appropriately. On both NT and Windows 2000, the agent can be configured to send SNMP traps to a specified address. The trap destination for our purposes should be set as the host PC IP address (or DNS name)—the machine on which the sample program executes.

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Setup Required for the Visual C++ Program

The supplied software was tested in the following environments:

- Windows NT Workstation Version 4.00 Build 1381 Service Pack 6a
- Windows 2000 Version 5.0.2195 Service Pack 2 Build 2195

On both of these platforms, the TCP/IP protocol stack was functioning with the SNMP Service configured. The agent used was one both local to and remote from the host running the manager.

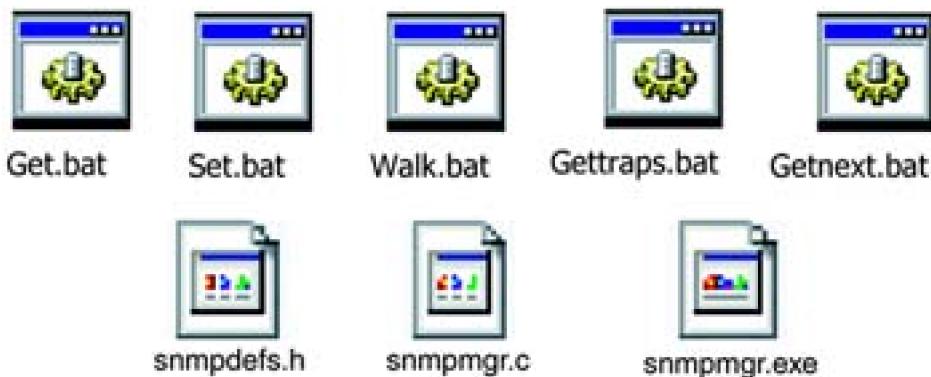
[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Building the Sample Visual C++ SNMP Manager

If it is required to build the Visual C++ program, then the files in [Figure 7-1](#) should be downloaded from the Prentice Hall Web site [[PrenHallCodeWeb](#)]:

Figure 7-1. The Visual C++ SNMP Manager files.



The purpose of these files is now briefly described:

1. **Get.bat** executes a single SNMP **GET** operation against the agent.
2. **Set.bat** executes a single SNMP **SET** operation against the agent.
3. **Walk.bat** walks the contents of a specified agent MIB table.
4. **Gettraps.bat** prepares the program for receiving SNMP traps.
5. **Getnext.bat** executes a single SNMP **getNext** operation against the agent.
6. **snmpdefs.h** is a header file containing data types, global variables, and function declarations.
7. **snmpmgr.c** contains the C code for the program.
8. **snmpmgr.exe** contains the executable file for the program.

The program **snmpmgr.exe** was built using Microsoft Visual C++ Version 6.0. To build the source code, the simplest method is to search Visual C++ for an SNMP example program called **snmputil**. Our example was originally based on this Microsoft sample program (our source code is a completely rewritten version). Open the Microsoft example and delete the C and header files (one of each), and then select Add Files and choose our two files (**snmpmgr.c** and **snmpdefs.h**). Change the target executable to **snmpmgr.exe**. The project should then build successfully.

To run the example (without building it), copy **snmpmgr.exe** into a directory called debug located *immediately underneath* the location of the above files. If the program is built using Visual C++, then the debug directory should be created automatically along with the target executable. In either case, the batch files should run successfully.

Our two files **snmpdefs.h** and **snmpmgr.c** are now described.

The Source Code Components of snmpmgr.exe

The files `snmpdefs.h` and `snmpmgr.c` provide the codebase for our elementary management system. We now describe the principal elements of this program: The source code listing is contained in [Appendix D](#), "Visual C++ Sample Program Source C." The contents of the header file `snmpdefs.h` are illustrated in [Figure 7-2](#). We added line numbers and section headings to make it easier to follow the description that follows. The header file `snmpdefs.h` is made up of three sections.

Figure 7-2 `snmpdefs.h`: The API for our rudimentary management system.

[View full width]

```
***** SECTION 1 *****
1 #define TIMEOUT 6000 /* milliseconds */
2 #define RETRIES 3
3 #define MAX_OID_NAME_LENGTH 50
4 #define FUNCTION_SUCCESS 0
5 #define FUNCTION_FAILED -1

***** SECTION 2 *****
6 enum Operations { GET, GETNEXT, SET, WALK, TRAP };
7 char * operationsArray[5] = { "GET", "GETNEXT", "SET", "WALK", "TRAP" };
8 INT programMode, operation;
9 LPSTR SNMPAgentName, SNMPCommunity, OIDString;
10 RFC1157VarBindList variableBindings;
11 LPSNMP_MGR_SESSION SNMPsession;
12 AsnAny retrievedInstanceValue;
13 INT timeout = TIMEOUT;
14 INT retries = RETRIES;
15 BYTE requestType;
16 AsnInteger errorStatus, errorIndex;

***** SECTION 3 *****
17 int allocateResources(LPSTR agentName, LPSTR community, char *objectIdentifier);
18 int deallocateResources();
19 int prepareForOp(enum Operations reqOperation, LPSTR agentName, LPSTR community, char
  *objectIdentifier, char *objectValue);
20 int prepareDataForOperation(enum Operations reqOperation, unsigned char
  *newObjectValue);
21 int prepareSetOperation(unsigned char *newObjectValue);
22 int prepareGetOperation();
23 int prepareGetNextOperation();
24 int dispatchOperation(int programMode, char * argumentVector[]);
25 int createSNMPSession();
26 int executeRequest();
27 int displayMIBInstanceValue();
28 int doSnmpOperation(enum Operations reqOperation, LPSTR agentName, LPSTR community,
  char *objectIdentifier, char *objectValue);
29 int executeMibWalk();
30 int waitForTraps();
31 int startupRoutine (int argc, char *argv []);
```

Snmpdefs.h Lines 1 to 5

Section 1 provides five symbolic constants for SNMP operations, comprised of the following:

- **Line 1:** Timeouts occur when an operation (such as `GET`) is executed and for some reason no response is received. There are many reasons for this; for example, the agent or network may be down, or the message may have been rejected due to incorrect security information. Timeouts are used to allow the manager to close down an operation rather than wait indefinitely.
- **Line 2:** Retries are needed for situations in which a message is lost or rejected due to some temporary problem. An example is a busy agent. Another is a lost message—remember that SNMP uses UDP, an unreliable datagram service. Typically, retries work in conjunction with timeouts, as we'll see in [Figure 7-11](#).
- **Line 3:** Maximum length permitted for an object identifier.
- **Lines 4–5:** Function success and failure codes.

The SNMP API uses the first three constants.

Snmpdefs.h Lines 6 to 16

Section 2 provides global variables. It is not necessary for many of these to have global scope, but they are made global in this example for simplicity. Some of the data objects are specific to the Microsoft SNMP API:

- **Line 6:** Operation indicator to store the SNMP operation type (`GET`, `SET`, etc.).
- **Line 7:** Operation array to store the SNMP operation name, (`GET`, `SET`, etc.).
- **Line 8:** Program mode integer.
- **Line 8:** Program operation integer.
- **Line 9:** String for storing the remote SNMP agent IP address or DNS name.
- **Line 9:** String for storing the SNMP community name.
- **Line 9:** String for storing the specified object identifier.
- **Line 10:** Container for the SNMP PDU variable bindings.
- **Line 11:** SNMP session pointer.
- **Line 12:** Container for retrieved objects.
- **Line 13:** Timeout global variable.
- **Line 14:** Global variable for storing the number of permitted retries.
- **Line 15:** Request type global variable for storing the required SNMP operation type.
- **Line 16:** SNMP error indicator—tells us if an error or exception occurred.
- **Line 16:** SNMP error offset value—tells us the variable bindings offset with which the problem is associated.

Snmpdefs.h Lines 17 to 31

Section 3 lists the available function calls and is comprised of the following:

- **Line 17: allocateResources()** dynamically allocates required memory, creates an SNMP session, and sets up the variable bindings structure for the SNMP operations.
- **Line 18: deallocateResources()** deallocates memory, destroys the SNMP session object, and frees up the variable bindings structure.
- **Line 19: prepareForOp()** calls the `allocateResources` function and prepares for the specific operation `GET`, `GETNEXT`, etc.).
- **Line 20: prepareDataForOperation()** sets up program mode for the required SNMP operation.
- **Line 21: prepareSetOperation()** indicates to the API to expect a`SET` operation and populates a variable bindings object ready for transmission to a remote SNMP agent as part of a `SET` operation.
- **Line 22: prepareGetOperation()** indicates to the API to expect a`GET` operation.
- **Line 23: prepareGetNextOperation()** indicates to the API to expect a`GETNEXT` operation.
- **Line 24: dispatchOperation()** issues the final call to the API for the required operation.
- **Line 25: doSnmpOperation()** makes the operation-specific call.
- **Line 26: createSNMPSession()** opens an SNMP session with the remote agent.
- **Line 27: executeRequest()** is the final port of call—the SNMP API is entered, and`SET/GET` message traffic is sent across the network.
- **Line 28: displayMIBInstanceValue()** presents the retrieved data to the user, showing the operation type, data type, and value.
- **Line 29: executeMibWalk()** carries out a lexicographic MIB walk using the supplied OID as the root.
- **Line 30: waitForTraps()** prepares the program for receiving incoming traps and displays them as they occur.
- **Line 31: startupRoutine()** parses the command line for program mode and any required parameter strings.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

The Structure of the Sample Visual C++ Program

The sample program is not complex; it has a `main()` structure made up of just the following two lines:

```
startupRoutine(argumentCount, argumentVector);
dispatchOperation(programMode, argumentVector);
```

The first function validates the command-line parameters and determines the program mode (**GET**, **SET**, etc.). The second function executes the required operation. The full source code listing is included in [Appendix A](#), "Terminal Server Serial Ports."

The Supported Operations

Five operations are allowed with this manager:

- GET
- GETNEXT
- SET
- WALK
- TRAP

The required operation is specified on the program command line. Overall, the external API provided by this program is a single function called:

[View full width]

```
int doSnmpOperation(enum Operations reqOperation, LPSTR agentName, LPSTR community, char
    *objectIdentifier, char *objectValue)
```

The program is invoked for a **GET** operation on the `ipInReceives` object value as follows, with a three-line response (commands typed by the user are shown in bold; system responses are delimited by angle brackets):

[View full width]

```
Debug\snmpmgr.exe GET myHostPC public .iso.org.dod.internet.mgmt.mib-2.ip.ipInReceives.0
    NULL
<***** THE SYSTEM RESPONSE NOW FOLLOWS ***** >
< SNMP Operation Type GET >
< MIB Object Instance = ip.ipInReceives.0 >
< Value   = Counter32 4929873 >
```

The parameters for this are as follows:

- `snmpmgr.exe`: the program name

- **GET:** the required SNMP operation type
- **myHostPC:** the target IP address or DNS name of the remote SNMP agent host
- **public:** the SNMP community name
- **.iso.org.dod.internet.mgmt.mib-2.ip.ipInReceives.0:** the required MIB object instance
- **NULL:** value for **SET**; not required for **GET**, hence the value is **NULL**

The remote (or local) agent responds to the message and the data sent back to **snmpmgr.exe** is presented as:

```
< SNMP Operation Type GET >
< MIB Object Instance = ip.ipInReceives.0 >
< Type and Value = Counter32 4929873 >
```

The last two lines indicate the required MIB object instance and its value (in this case, the object is our old friend **ipInReceives.0** and has the value 4929873). The supplied DOS batch files automate the process of running the program, freeing the user from having to type complex command lines. The batch files and their use are described in the following sections.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

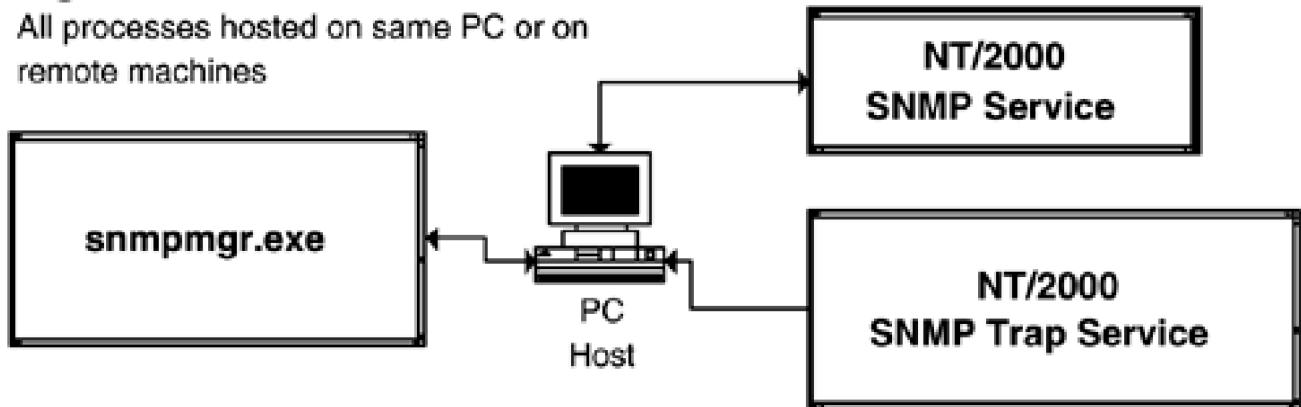
Using the Rudimentary Management System

The sample program can, if required, be run on the same host as the SNMP agent, as illustrated in [Figure 7-3](#).

Figure 7-3. Processes and hosts in the rudimentary management system.

Legend

All processes hosted on same PC or on remote machines



Alternatively, the program can be run using an SNMP agent on a remote machine.

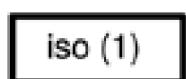
Please Note For Running The Visual C++ Program

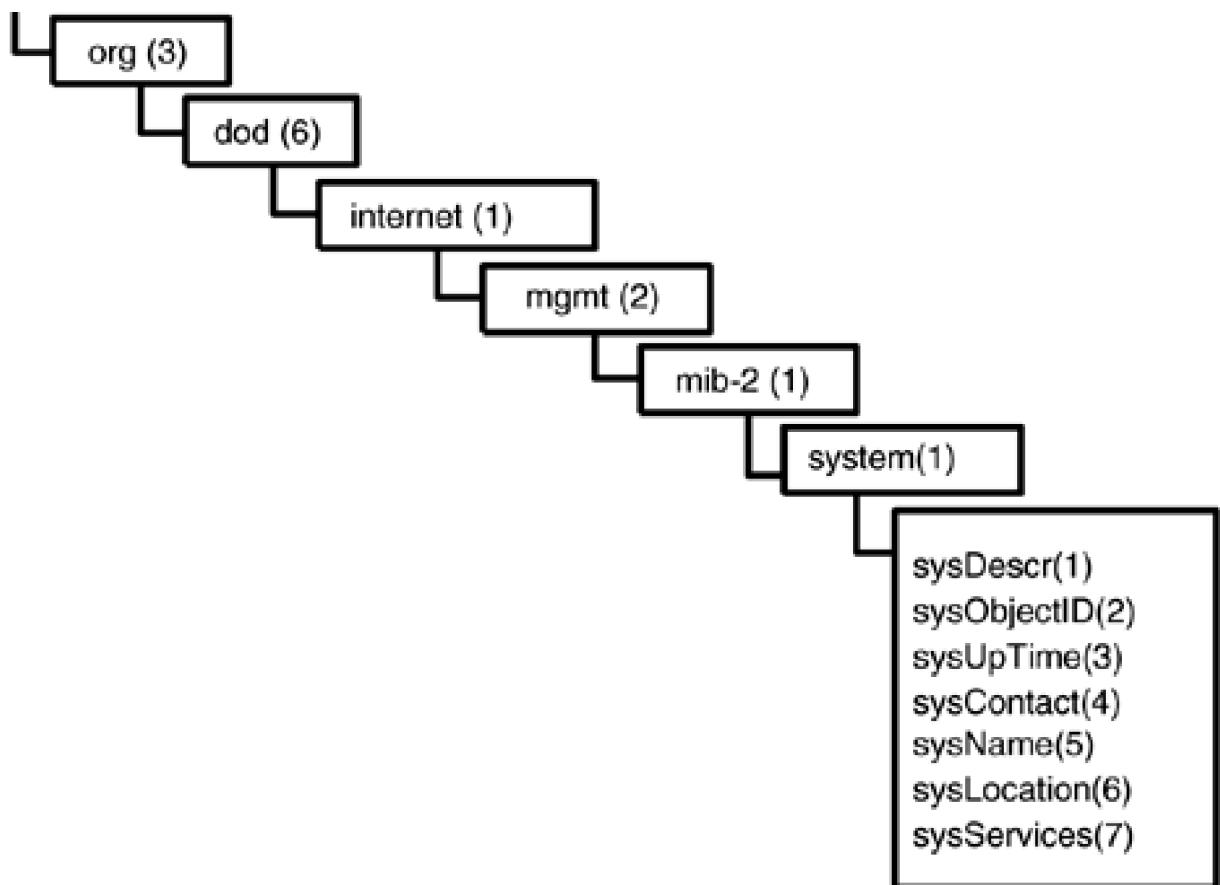
1. We use just a single SNMP community string called public. Normally, you can expect to see at least two community strings—one for gets, called public, and one for sets, called private. We omitted this to reduce the complexity. Changing the batch files for this would be a good exercise because it would also be required to create the private community string in the agent configuration and to assign read-write capability.
2. Only one change is required—the user must supply the IP address or DNS name for the SNMP agent host in each of the supplied batch files. For example, in the batch file Get.bat (see [Figure 7-5](#)), replace the word myHostPC with the IP address or DNS name of your SNMP agent host.
3. The required platform for running the program is either Windows 2000 or NT.

MIB Table Used in the Examples

The MIB-II system table is used in our examples and is illustrated in [Figure 7-4](#).

Figure 7-4. The MIB-II system table.





[Figure 7-4](#) can be consulted during the following sections. The syntax for specifying MIB objects in the Microsoft SNMP API is to prefix each leaf with a period, starting with `.iso`. This is seen in the next section.

An SNMP GET

Double-clicking the batch file `Get.bat` (or running it from a command line) should result in a display similar to that shown in [Figure 7-5](#).

Figure 7-5 `GET` operation on `system.sysContact.0`.

```
Debug\snmpmgr.exe GET myHostPC public .iso.org.dod.internet.mgmt.mib-2.system.sysContact.0
< SNMP Operation Type GET >
< MIB Object Instance = system.sysContact.0 >
< Type and Value = String StephenM >
```

[Figure 7-5](#) illustrates a `GET` operation on a specific instance (zero in this case) of a columnar object from the standard system table—in this case, `system.sysContact`. We must specify the instance explicitly by appending a zero. The value of the returned object is illustrated as `StephenM`.

An SNMP GETNEXT

Double-clicking the batch file `GetNext.bat` (or running it from a command line) should result in a display similar to that shown in [Figure 7-6](#).

Figure 7-6 GETNEXT operation on system.sysContact.0.

[View full width]

```
Debug\snmpmgr.exe GETNEXT myHostPC public .iso.org.dod.internet.mgmt.mib-2.system.  
  ↗ sysContact.0  
< SNMP Operation Type GETNEXT >  
< MIB Object Instance = system.sysName.0 >  
< Type and Value = String myHostPC >
```

Figure 7-6 illustrates a **GETNEXT** operation on the **sysContact** object. Instead of returning **StephenM**, as in the last example, we receive the host system name. This is the lexical successor to the **sysContact** object, as can be seen from [Figure 7-4](#).

An SNMP SET

We now move to a different category of SNMP operation, that of **setRequest**. This pushes data into the MIB of the remote agent. Interestingly, providing a set capability like this is often considered controversial because of security fears. These fears are generally well-founded, but we present sets here only in order to describe the principles, leaving aside any political issues. In this example, we set the value of the **sysContact** name to be **StephenMorris**. Recall from the **GET** example above that the value of this object is currently **StephenM**. Double-clicking the batch file **Set.bat** (or running it from a command line) should result in a display similar to that shown in [Figure 7-7](#). We can verify that the operation succeeded by simply running **Get.bat** again.

Figure 7-7 SET operation on the system.sysContact.0.

[View full width]

```
Debug\snmpmgr.exe SET myHostPC public .iso.org.dod.internet.mgmt.mib-2.system.sysContact.  
  ↗ 0 StephenMorris  
< SNMP Operation Type SET >  
< MIB Object Instance = system.sysContact.0 >  
< Type and Value = String StephenMorris >
```

Figure 7-7 illustrates a **SET** operation on the **system.sysContact.0** object.

Walking a MIB Table

Suppose we want to get an idea of the MIB objects supported by a given agent, starting at a specific point in the tree. An SNMP walk provides this capability by traversing the MIB tree leaf objects until the end of that branch is reached. This is useful for retrieving tables (of unknown extent). Double-clicking the batch file **Walk.bat** (or running it from a command line) should result in a display similar to that shown in [Figure 7-8](#). This is achieved by a dialog between the manager and the agent in which the former keeps sending **GETNEXT** messages until the end of the MIB table is reached.

Figure 7-8 A MIB WALK on the system table.

```
Debug\snmpmgr.exe WALK myHostPC public .iso.org.dod.internet.mgmt.mib-2.system
< Variable = system.sysDescr.0 >
< Value   = String Hardware: x86 Family 6 Model 8 >
< Variable = system.sysObjectID.0 >
< Value   = ObjectID 1.3.6.1.4.1.311.1.1.3.1.1 >
< Variable = system.sysUpTime.0 >
< Value   = TimeTicks 1889968 >
< Variable = system.sysContact.0 >
< Value   = String StephenMorris >
< Variable = system.sysName.0 >
< Value   = String myHostPC >
< Variable = system.sysLocation.0 >
< Value   = String Europe >
< Variable = system.sysServices.0 >
< Value   = Integer32 76 >
< End of MIB subtree. >
```

[Figure 7-8](#) illustrates a **WALK** of the system table. Referring to [Figure 7-4](#), it can be seen that all the columns in this table are retrieved up to and including the last object, **sysServices**.

An SNMP TRAP

The manager program can also be configured to listen for SNMP traps. This is illustrated by running the batch file **GetTraps.bat**. This puts the manager into listening mode, as can be seen in [Figure 7-9](#). Next, we must simulate a trap. This can be done by simply stopping and starting the SNMP service, which results in the agent sending three traps, as illustrated in the bottom half of [Figure 7-9](#).

Figure 7-9 Listening for and receiving SNMP traps.

```
Debug\snmpmgr.exe TRAP
< snmputil: listening for traps. >
***** THE PROGRAM IS NOW AWAITING THE ARRIVAL OF TRAPS *****
***** WE NOW STOP AND THEN START THE SNMP SERVICE *****
snmputil: trap generic=0 specific=0 from -> 127.0.0.1 *** TRAP 1
snmputil: trap generic=3 specific=0 from -> 127.0.0.1 *** TRAP 2
Variable = interfaces.ifTable.ifEntry.ifIndex.1
Value   = Integer32 1
snmputil: trap generic=3 specific=0 from -> 127.0.0.1 *** TRAP 3
Variable = interfaces.ifTable.ifEntry.ifIndex.16777219
Value   = Integer32 16777219
```

Let's take a closer look at what's going on in [Figure 7-9](#). Trap 1 has both generic and specific code values of zero, indicating that this is a **coldStart**, that is, the agent has restarted. This is a direct response to stopping and starting the SNMP service. The origin of this trap is the loopback IP address (127.0.0.1), indicating that the agent is located on the same host as the manager. Traps 2 and 3, respectively, have generic and specific codes of three and zero, indicating **linkUp** events. These relate to entries in the host system interfaces MIB table (**ifTable**). Two interfaces are present on the host machine (indicated by the MIB object **ifNumber**, which has the value 2 and is not shown); one is the software loopback interface (127.0.0.1), and the other is an Ethernet interface (16777219). When these interfaces go into the up state, the agent detects it and emits traps 2 and 3.

Combining the Batch Files

It is easy enough to chain these batch files together (or write entirely new ones) to carry out multiple operations in sequence—for example, to execute a **SET** followed by a **GET**. A real NMS tends to use a rich mix of all types of SNMP operations, for example, manipulating a range of scalar and tabular objects on a number of agents (as we'll see in [Chapter 8](#)).

A Security Violation

Security is increasingly important in network management. We now acknowledge this with a specific example of a simple security violation by using the wrong community name. This is equivalent to entering the wrong password during a login procedure. Modifying the community name in one of the batch files achieves the desired result. The remote agent should do two things:

- Discard the message so that no reply should occur—this gives no clue to a hacker that he or she has supplied an incorrect community name.
- Emit an **authenticationFailure(4)** trap—this informs the NMS operator that an intruder alert has occurred.

In a real network with stronger security (such as SNMPv3), the intruder might be attempting more sophisticated actions, such as replaying a captured message into an agent. In this case, the security system in the agent would rely on more advanced protection facilities, such as message timeliness checking. However, we illustrate the simple (wrong community string) case just to give a flavor of what might occur in a real network.

To see this, we must configure one session of the program to listen for traps and another to issue an improper SNMP Get. [Figure 7-10](#) illustrates the **GET** with a modified (and invalid) community string of **public1** (instead of **public**).

Figure 7-10 Security failure and associated actions.

```
Debug\snmpmgr.exe GET myHostPC public1 .iso.org.dod.internet.mgmt.mib-2.system.sysContact.0
< error on SnmpMgrRequest 8 >
< SNMP Operation Type GET >
< MIB Object Instance = system.sysContact.0 >
< Type and Value = Null value >
```

In [Figure 7-10](#), we see a Microsoft Visual C++ SNMP API error value of 8, indicating a problem with the submitted message.

If we initiate another instance of the program in trap mode (before running the **Get** in [Figure 7-10](#)), we see an interesting sequence of events, as illustrated in [Figure 7-11](#). Four traps are received from the loopback address. So, what's happening here, and why do we see four traps?

Figure 7-11 Security violations and retries.

```
Debug\snmpmgr.exe TRAP
< snmputil: listening for traps... >
< snmputil: trap generic=4 specific=0 from -> 127.0.0.1 >
< snmputil: trap generic=4 specific=0 from -> 127.0.0.1 >
< snmputil: trap generic=4 specific=0 from -> 127.0.0.1 >
< snmputil: trap generic=4 specific=0 from -> 127.0.0.1 >
```

Recall from [Figure 7-2](#) that the program implements a retry mechanism (the **RETRIES** symbolic constant in **snmpdefs.h**). A maximum of three retries is allowed. So, the first **GetRequest** is sent to the agent. This message has an invalid community string and is discarded. The agent also issues a trap with a generic code of 4 (indicating an **authenticationFailure**). This explains the first trap. Our manager code is not very sophisticated and fails to correlate the authentication failure trap with the bad **GetRequest** message. Having failed to get the expected response, the manager then transparently (via the SNMP library code) retries the operation a total of three more times; this results in three more traps before the manager code finally gives up. This explains the sequence of four traps in [Figure 7-11](#). One final point about [Figure](#)

[7-11](#) is that the timeout mechanism is also initiated as a result of the (attempted) security breaches. Again from [Figure 7-2](#), we note that there is a timeout defined as 6,000 milliseconds or 6 seconds. This is the time between retries and explains the small delays between the occurrence of the traps in [Figure 7-11](#).

[Team LiB]

[PREVIOUS] [NEXT]

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

A Note On Security

The preceding discussion leads into another very important area: that of attacks against networks and management system components. The sheer quantity of attacks against networks is surprising—thousands are independently recorded annually. An entire book could be dedicated to this subject. Networks can be subjected to the following types of attacks, among many others:

- Hacking: Attempting to gain unauthorized access to agent data (as above)
- Denial of service: Flooding an agent with bogus messages, such as `GetRequests`
- Message interception: Listening for and capturing messages in transit
- Message modification: Modifying the contents of messages in transit
- Message replay: Capturing and then resending messages

It is surprising how unprotected many networks are [[CERTWeb](#)], and the emphasis on security is certain to increase. Protection must be applied throughout the network from the very top of the management system software pyramid (introduced in [Chapter 1](#)) to the NEs and agents deployed in the network.

We now move onto our second sample program, this time using Java.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

The Sample JDMK Java SNMP Manager

Evaluation copies of the JDMK toolkit can be downloaded from Sun Microsystems' Web site [[Sun Microsystems](#)]. Included with JDMK is a rich mix of sample Java programs that use the internal class library. In this section, we describe a handcrafted program that uses JDMK to implement the following SNMP requests:

- GetRequest
- GetNext

In effect, this is a cutback version of our earlier Visual C++ program. JDMK programs can interact with either the JDMK sample agent provided by Sun or with any other legacy agent. To illustrate this interoperability, we use the standard Windows 2000/NT SNMP agent as the target of our JDMK SNMP **Get/GetNext** messages (rather than using the JDMK sample agent). This is an interesting illustration of the mature and standard nature of SNMP technology in general. The full source code for the Java program is located in [Appendix B](#), "Some Simple IP Routing Experiments."

Just as for the C++ sample, we provide companion DOS batch files to simplify use of the program.

Installing JDMK

A number of Sun Microsystems' software applications are needed for our Java example. The following installation steps were followed (for Windows NT) in order to set up JDMK:

1. Install the Java Development Kit (JDK): We used the Java 2 SDK Standard Edition v1.3.1_02.
2. Install JDMK: We used JDMK 4.2 for JDK 1.1.8.
3. Install JDMK: We used JDMK 4.2 for Java 2 Platform.

Step 2 may not be necessary.

Installation of JDMK involves running two downloaded batch files called **Setup.bat**—this extracts the JDMK class files (**jdmk42_nt_12.class** and **jdmk42_nt_11.class**, respectively).

Next, the environment variables must be set. We created one variable (**JDMKPATH**) and modified two others for this:

1. JDMKPATH
2. The system PATH
3. The Java CLASSPATH

The values we assigned to these variables are:

[View full width]

JDMKPATH=C:\Program Files\SUN\Wjdmk\jdmk4.2\1.2

```
PATH=%JDMKPATH%\bin;C:\jdk1.3.1_02\bin  
CLASSPATH=%JDMKPATH%\lib\collections.jar;%JDMKPATH%\lib\jdmkrt.jar;%JDMKPATH%\lib\jdmktk.  
jar;
```

The syntax **%JDMKPATH%** is simply a shorthand form for the fully expanded contents of **JDMKPATH**. One last point is to make sure to include the period (for the current directory) at the end of the **CLASSPATH**.

Last Steps in the Setup

To use the NT SNMP agent, make sure to apply NT Service Pack 6a. Also, ensure that the SNMP agent has two configured community strings (the Visual C++ program used just one community string):

- **public** (read-only)
- **private** (read-write)

It should be possible to successfully run the examples once these steps have been taken. Please note that the two managers were run only on a Windows NT Workstation (Version 4.00 Build 1381) with Service Pack 6a. Windows 2000 was used only for the agent.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Building the Sample Java Manager

If it is required to build the Java program, then the files in [Figure 7-12](#) should be downloaded from the Prentice Hall Web site [[PrenticeHallCodeWeb](#)]:

Figure 7-12. The JDMK SNMP manager project files.



The purpose of these files is now briefly described:

1. `mib_II.txt` contains the definition of the MIB II standard objects.
2. `SynchronousManager.java` contains the Java class that sends either an `SNMPGET` or `getNext` message.
3. `Get.bat` executes a single `SNMP GET` operation against the agent.
4. `GetNext.bat` executes a single `SNMP getNext` operation against the agent.

Once the Java class files are created in the next section, it is possible to run the batch files.

To Build the Java Program

Two commands are required to build the Java program:

1. `mibgen -mo -d . mib_II.txt`
2. `javac -d . *.java`

The first command builds a file called `RFC1213_MIBOidTable.java`. The second command creates the bytecode file `SynchronousManager.class`. The latter is the binary (executable) file for the examples that follow.

An SNMP GET

To run the program, double-click on `Get.bat` to invoke Java and execute a `GetRequest` operation. This should result in a display similar to that shown in [Figure 7-13](#).

Figure 7-13 GET operation on the `system.sysContact.0`.

```
java SynchronousManager GET myHostPC public sysContact.0 NULL 161
< Sent GET request to agent on myHostPC port 161 >
```

```
< Result: [Object ID : 1.3.6.1.2.1.1.4.0 (Syntax : String) >
< Value : StephenMorris] >
```

[Figure 7-13](#) illustrates a **GET** operation on the **sysContact** object from the standard system table. The object instance is hardcoded (i.e., there is no way for the value to be changed other than by a program source code change) internally to the program. The value of **system.sysContact.0** is illustrated as **StephenMorris**. Recall that this object instance was set to its current value in [Figure 7-7](#).

An SNMP GETNEXT

To run the program, double-click on **GetNext.bat** to invoke Java and execute a **GetRequest** operation. This should result in a display similar to that shown in [Figure 7-14](#).

Figure 7-14 **GETNEXT** operation on **system.sysContact.0**.

```
java SynchronousManager GETNEXT myHostPC public sysContact.0 NULL 161
< Sent GETNEXT request to agent on myHostPC port 161 >
< Result: [Object ID : 1.3.6.1.2.1.1.5.0 (Syntax : String) >
< Value : myHostPC] >
```

[Figure 7-14](#) illustrates a **GETNEXT** operation on the **sysContact** object from the standard system table. As was the case with the C++ program, the **GETNEXT** response provides the lexical successor to the **system.sysContact.0** object. This is the **system.sysName.0** object and has the value **myHostPC**.

The Structure of the Synchronous Manager

The Java program is very simple indeed. All of the code is contained in one file that in turn contains a Java class called **SynchronousManager**. The command-line parameters are validated, and the required operation type is recorded as either **GET** or **GETNEXT**. The next nine lines prepare the API for making SNMP calls. The actual SNMP request is made in the method called:

```
public static void issueRequest(SnmpPeer agent, String operation,
    SnmpVarBindList list, SnmpSession session,
    String host, String port)
```

It is this method that issues the message and processes the agent response. Any exceptions that occur are caught in an overall **try/catch** block.

The Synchronous Java Manager

The JDMK API provides both synchronous and asynchronous operation. Synchronous operation simply waits for responses from the network, whereas asynchronous operation uses callbacks to allow the program to proceed with other tasks. A production-standard NMS tends to use asynchronous operation, or it may use synchronous calls in conjunction with multiple threads. In either case, the NMS should not simply block until a response is received. As we've seen, there is generally much ongoing work to be done by a given NMS as it strives to keep up to date with the managed network. So, execution time should never be squandered waiting for (possibly) tardy devices to respond.

Our examples are all synchronous in nature in order to illustrate the software components in as simple a fashion as possible.

Comparing the Visual C++ and JDMK 4.2 APIs

[Table 7-1](#) shows a brief comparison between the Visual C++ and JDMK APIs.

Table 7-1. API Feature Comparisons

SNMP API	MULTI-PLATFORM	SNMPv1	SNMPv2	SNMPv3
JDMK 4.2	Yes – Java	Yes	Yes	No 
Visual C++	No	Yes	No	No

[*] SNMPv3 support is provided as part of JDMK 5.0.

An interesting exercise in interoperability is to mix and match the operation of the two programs, for example,

1. Set the C++ program to listen for traps.
2. Execute a security violation (using the wrong community name) with the Java program.
3. Watch the traps coming back from the SNMP agent.

To compare the performance of the two APIs, a new batch file can be written to make multiple calls (e.g., 10 GETs) to one of the example batch files. The system time can be printed at the beginning and end of the new batch file, and from this, a very simple idea of the overall execution time can be derived.

In this case, the agent is (happily) communicating with two entities, one written in C++ and the other in Java. Though a relatively trivial test, this shows the platform-independent power of standardized protocols.

Ways to Improve the Two Sample Programs

Our two efforts are far from being production-standard code! There are many ways in which they both can be materially improved:

- Make the operations asynchronous to free up resources when waiting on replies.
- Move the parameters off the command line or provide them in encrypted form.
- Make the programs independent of MIB object specifications—the existing structure is inflexible because a MIB change would require a code change.
- Provide a facility for adding support for new MIBs.
- Allow multiple OIDs in one PDU—this reduces traffic for multi-object operations.
- Move all SNMP API code into a separate module or even a separate server. The latter would help to thin down the client programs.
- Remove global variables.

- Provide a non-debug version of the C++ program (faster and smaller).
- Allow table-based operations, such as for getting and setting rows in a MIB table.
- Provide an external data source other than the command line.
- Support SNMPv3.

It is better to fulfill the above from the ground up, that is, before writing any code. The examples were written purely to try to make concrete some of the major SNMP concepts.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

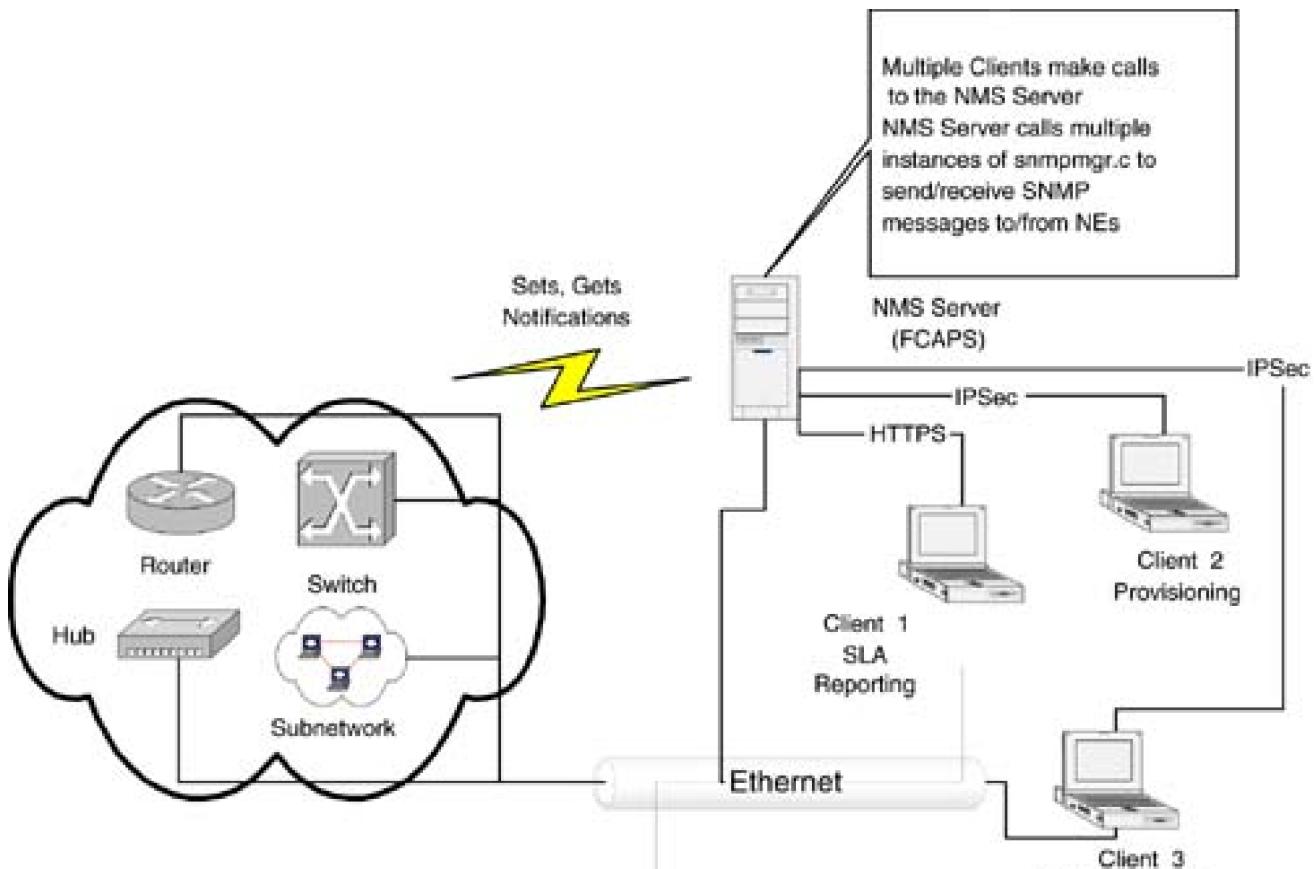
Extending the Sample SNMP Programs

The C++ or Java program samples can form the foundation of a more complex product. NMS typically take the form of a server with a number of distinct (possibly distributed) components, such as (in the FCAPS order):

- A fault server that listens for traps from the managed network
- A configuration (or provisioning) server that executes SNMP **SET** and **GET** operations
- An accounting server that can apply rules (such as quotas or limits) to specific MIB object counters, such as setting an upper limit on the value of **ipInReceives**
- A performance server that can perform mediation and also read the various counters supported in the NE MIBs
- A security server that can be used to manage community strings, access control rights, encryption, and authentication details

[Figure 7-15](#) illustrates one such arrangement with a central NMS server and multiple clients. Each client is dedicated to a specific network management function that it executes using the services of the central server. Client 1 is tasked with SLA reporting; that is, it retrieves data from the network relevant to SLAs. This data is then presented as reports, for instance, in HTML format. Client 2 configures the managed NEs by issuing requests (via the central server) to set and get managed objects in the network. Client 3 provides a billing/accounting function—usually this amounts to reading many objects. Clients 2 and 3 access the server using IPSec, and Client 1 uses HTTPS; all the clients are employing secure communications.

Figure 7-15. Extended `snmpmgr.c` handles multiple clients.



The important point to note about [Figure 7-15](#) is that it is relatively straightforward to build a complex management system once the basic agent and manager entities are in place. Additional server components are:

- Topology manager to handle icons, network maps, and device status
- An access server that provides a queuing mechanism for handling simultaneous requests from more than one client user (this server could be part of the server in [Figure 7-15](#))

The last two points move us out of the SNMP domain and are included just for completeness.

[Team LiB]

[PREVIOUS] [NEXT]

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Summary

We've now seen two simple programs that were used to interact with the Windows NT/2000 SNMP agents. Written in Visual C++ and Java, respectively, these programs provide what might be considered the low-level bricks or widgets that can be brought together to form a more complex product. The latter could in turn be built upon to become an NMS. The choice of programming languages was driven by considerations such as multiplatform support (Visual C++ is restricted to Windows, Java is multiplatform), the version of SNMP that can be used, and so on. It is surprisingly straightforward to produce such software programs. Once written, the two example programs can be combined using the target SNMP agent as a type of programming language-independent traffic cop; for example, set the C++ program to receive traps, then send a bad `getRequest` from the Java program, and the agent will send an authentication failure trap message to the C++ program. In effect, the agent is indifferent to the sending program just as long as the messages it receives conform to the SNMP protocol.

We saw a security violation caused by the use of an incorrect community string. The receiving agent discards the message and emits an authentication failure trap. The manager should not then resend the erroneous message because the same failure will occur. In a sense, this could be construed as a type of unintentional attack on the agent.

The example programs can be extended and made into more complex NMS components that use, for example, database services and a multiclient GUI. An important design goal is providing support for SNMPv3—JDMK allows this.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Chapter 8. Case Study: MPLS Network Management

In [Chapter 7](#), "Rudimentary NMS Software Components," we saw some simple software tools that can be used to experiment a little with MIBs and SNMP agents. Beyond experimentation, the software provides some of the bricks that might be used to construct the foundations of a basic NMS. In addition, we hope that this exercise has helped to crystallize many of the concepts introduced earlier in the book. Readers with access to MPLS NEs could, with a little effort, extend the [Chapter 7](#) tools to read (and possibly write) to the MIBs described in the current chapter. In this chapter, we try to bring together as many as possible of the separate MPLS strands running through the book; specifically, we:

- Look more closely at the IETF MPLS MIBs
- Describe how the MIB elements relate to MPLS in general
- Explain how to combine these MIB elements in an operational MPLS network

Without further ado, let's get started on these MIBs.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

The (Internet Draft) Standard MPLS MIBs

The two MIBs described in the IETF drafts "Multiprotocol Label Switching (MPLS) Label Switch Router (LSR) Management Information Base" and "Multiprotocol Label Switching (MPLS) Traffic Engineering Management Information Base" [[IETF-LSR-MPLS](#)] and [[IETF-TE-MPLS](#)] provide a framework for managing MPLS NEs. (There are other MPLS MIBs, but we have selected these for our current discussion). At the time of writing, these MIBs are draft standards, but it is highly likely that they will proceed to full IETF standard status. Broadly speaking, the two MIBs can be used to achieve the following:

- Manage the low-level MPLS objects, such as interfaces, cross-connects, and segment tables
- Create LSPs
- Manage the high-level MPLS objects, such as traffic-engineered tunnels, EROs, and resource blocks

These two MIBs are now described. The LSR MIB objects include tables that describe:

- MPLS interface configuration
- In-segments
- Out-segments
- Cross-connects
- Label stacks
- Traffic parameters
- Performance parameters

These objects are described in the following sections. Similarly, the TE MIB objects include tables that describe:

- Traffic-engineered tunnels
- Tunnel resources
- Tunnel paths
- Tunnel performance counters

We now start the discussion with MPLS devices.

MPLS Devices

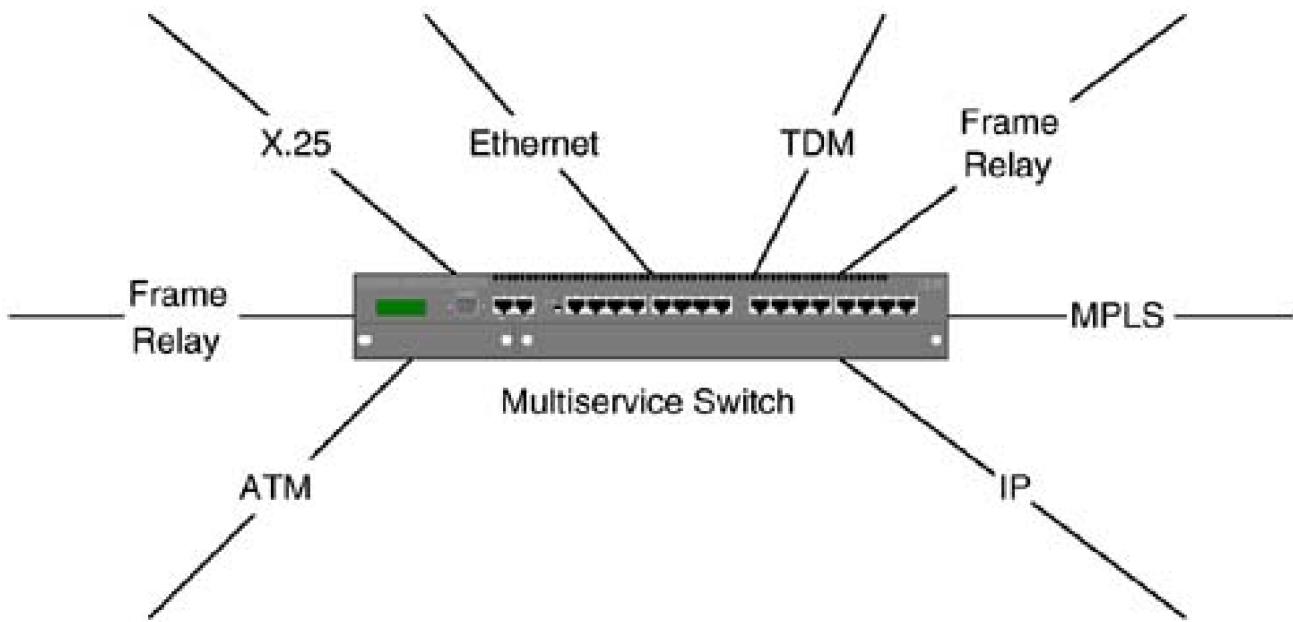
MPLS devices are NEs on which MPLS technology is deployed, and they can include:

- IP routers
- ATM switches operating in SIN mode

- Multiservice switches

MPLS technology may be added as a firmware upgrade to such devices, or it may be included as a standard product component. This reflects the migration approach adopted for MPLS deployment: It can be switched on/off and used on an as-needed basis. In other words, a network operator can phase in the use of MPLS in conjunction with existing technologies such as ATM and FR. As the deployment (and planned deployment) of MPLS increases, the ability to smoothly (and slowly) apply its use in production networks is very useful. This is illustrated in [Figure 8-1](#), where a multiservice switch supports a number of different technologies.

Figure 8-1. A multi-service switch that supports MPLS.



The multiservice switch in [Figure 8-1](#) can originate and terminate a range of service types, such as Ethernet, X.25, TDM, IP, FR, and MPLS. Clearly, the switch is part of a broader network that supports these services. Over time, it is likely that such networks may migrate to just IP or possibly IP and MPLS. For this reason, it is important that the switch be capable of moving over to supporting just these service types without the need for expensive hardware upgrades. So, MPLS NEs implement the MPLS technology in firmware, and access to it is made through MPLS interfaces. The latter are described in the next section.

MPLS Interfaces

An MPLS interface is one on which MPLS has already been configured and may include the following:

- An IP routing interface (recall interfaces from [Chapter 1](#), "Large Enterprise Networks").
- An IGP routing protocol with traffic engineering extensions, such as OSPF-TE, IS-IS-TE. An IGP routing protocol is not mandatory—static routes can be used instead.
- Possibly an EGP protocol if the node faces out of an autonomous system. Typically, IGP and EGP protocols are not used on the same interface. This is to avoid leaking routing information between adjacent networks.
- A signaling protocol such as LDP or RSVP-TE.

In the next section, we will see MPLS interfaces.

MPLS Network Example

Figure 8-2 illustrates MPLS interfaces with the letters A, B, C, and D, respectively. The lower half of the diagram has four more MPLS interfaces that have the following IP addresses: 5.5.4.1; 5.5.4.2; 5.5.5.1; and 5.5.5.2. This is the network that will be used for our MIB investigation. The `ifIndex` values for these interfaces are illustrated in parentheses. Also, interfaces in the lower half of the diagram are deliberately not labeled.

Figure 8-2. An LSP and tunnel in an MPLS network.

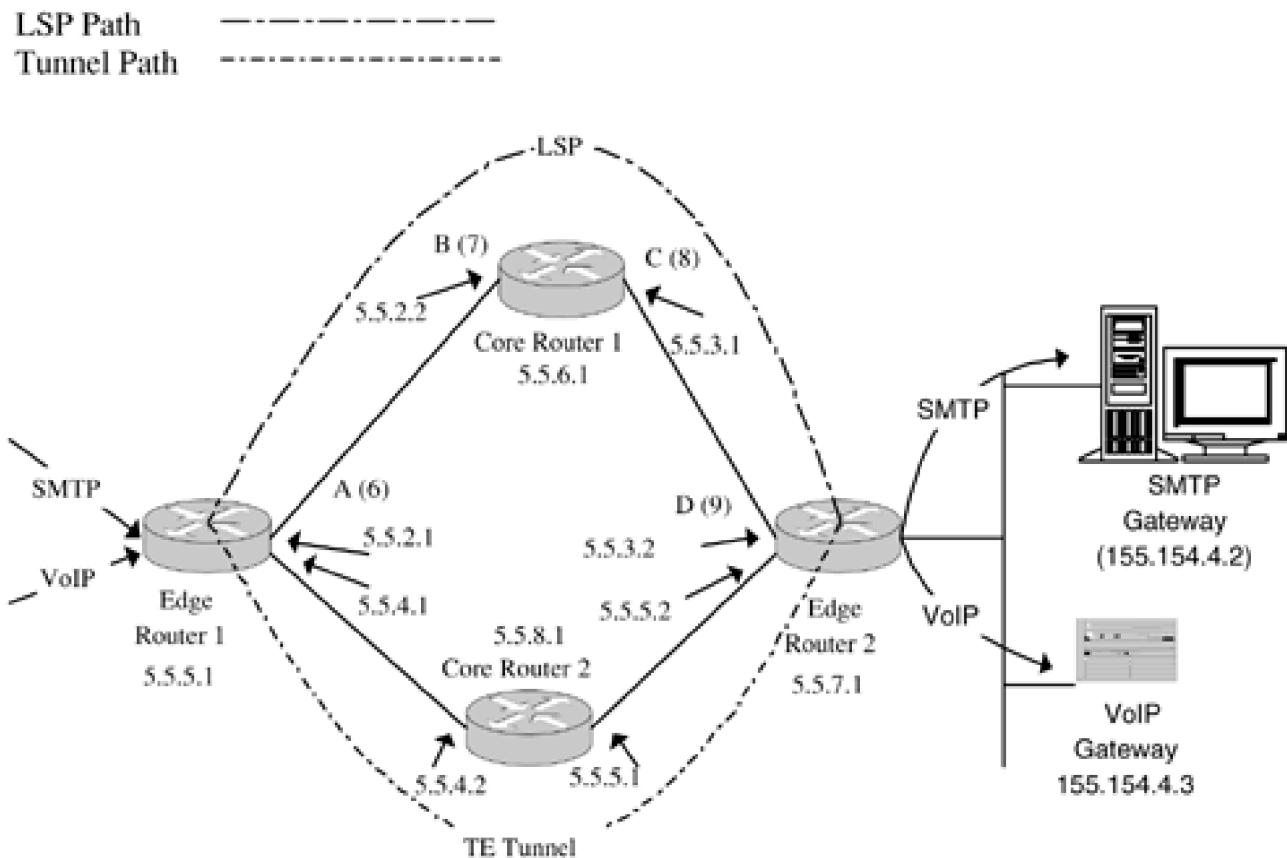


Figure 8-2 illustrates a four-node MPLS network that shares a boundary with an IP network. The MPLS network forwards real-time VoIP and non-real-time SMTP (email) traffic from one edge of the IP network in the direction of an adjacent subnetwork containing two gateways. Both sets of traffic are terminated on the latter devices. An LSP and a traffic-engineered tunnel have been configured in the MPLS network between the two edge nodes (LERs) with the core routers (LSRs) acting as transit nodes. The tunnel (called TE Tunnel in the diagram) is created using the TE MIB and the LSP is created using both the TE MIB and the LSR MIB. The TE Tunnel has been allocated sufficient bandwidth (640kbps) to simultaneously carry 10 uncompressed voice channels in its path. The LSP has no reserved bandwidth and offers a best-effort service level. Later in this chapter we show how the MIB is manipulated to create these entities.

A noteworthy item of interest about the LSP and tunnel is that they originate and terminate inside the LERs rather than on the external interfaces. Each of them serves a destination IP address (or prefix). Incoming IP traffic lands on the Edge Router 1 and is then pushed into the appropriate tunnel or LSP. Which one should be used? That depends on the nature of the IP traffic; if it has been marked to receive better than best effort (hop-by-hop) forwarding, then it may take the path provided by the tunnel. The ingress LER makes the decision about the path taken by the packet by encapsulating it with an appropriate MPLS label—one label for the LSP and another for the tunnel. The labeling decision may also be made based on any or all of the following:

- The contents of the IP header DS field (and even the two Explicit Congestion Notification bits)
- The destination or source IP address
- The destination or source port

The important point to note is that the labeling decision can be based on a rich combination of parameters. In the example of [Figure 8-2](#), we take the most basic option because IP traffic is pushed into either the tunnel or LSP based only on the destination IP address. The policies that dictate traffic treatment are generally the network operator's responsibility.

Each of the MPLS interfaces A, B, C, and D has a corresponding entry in the MIB table `mplsInterfaceConfTable`. The same is true of the unmarked interfaces in the lower half of [Figure 8-2](#). The latter are not annotated in order to reduce clutter. An MPLS node would automatically populate this table with a row for each MPLS-capable interface. An entry in this table is illustrated in [Figure 8-3](#). Please note that the MIB excerpts in the rest of the chapter take the form of **SEQUENCEs** of objects. These are ASN.1 constructs and are copied straight from the MIB definitions. The listed objects should be visualized as columns in a table (conceptually similar to a spreadsheet or a relational database table). Index objects are commented and appear in bold.

Following a description of the MIB tables, we illustrate how the tables would be manipulated to create the LSP and tunnel objects in [Figure 8-2](#). The software provided in [Chapter 7](#) could be extended to achieve this—that is, instead of addressing single MIB objects, the programs could address entire tables.

Figure 8-3 The MPLS interface MIB table.

```
MplsInterfaceConfEntry ::= SEQUENCE {  
    mplsInterfaceConflIndex      InterfaceIndexOrZero, -- Index  
    mplsInterfaceLabelMinIn     MplsLabel,  
    mplsInterfaceLabelMaxIn     MplsLabel,  
    mplsInterfaceLabelMinOut    MplsLabel,  
    mplsInterfaceLabelMaxOut    MplsLabel,  
    mplsInterfaceTotalBandwidth MplsBitRate,  
    mplsInterfaceAvailableBandwidth MplsBitRate,  
    mplsInterfaceLabelParticipationType BITS }
```

There is a relationship between the MPLS interface table and the `interfaces.ifTable`. This relationship is provided by the value of the `mplsInterfaceConflIndex` object. The range of MPLS label values that this interface can receive is indicated by the `mplsInterfaceLabelMinIn` and `mplsInterfaceLabelMaxIn` objects. The range of MPLS label values that this interface can send is indicated by the `mplsInterfaceLabelMinOut` and `mplsInterfaceLabelMaxOut` objects. The `MplsLabel` object is represented by four octets. Bits 0 to 19 represent the label with values supported, as we saw in [Chapter 4](#), "Solving the Network Management Problem," [Figure 4-10](#)—for example, Explicit Null (0), Router Alert (1). The remaining 12 bits encode the Exp, Stack, and TTL fields.

The total amount of usable bandwidth on this interface is indicated by `mplsInterfaceTotalBandwidth` and is specified in units of kilobits per second. The amount of bandwidth available at any given time is indicated by `mplsInterfaceAvailableBandwidth`; this is the difference between `mplsInterfaceTotalBandwidth` and the amount of bandwidth in use. The `mplsInterfaceLabelParticipationType` object dictates whether the label space is distributed across the platform or the interfaces. Per-platform label participation indicates that labels are globally allocated across the platform. Per-interface label participation indicates that each interface shares the label space with a specified range.

In-Segments

An in-segment is the ingress leg of an LSP segment on a given MPLS NE. This is an object that controls the forwarding of packets into the LSP. Each of the in-segments on an MPLS node has a corresponding entry in the MIB table `mplsInSegmentTable`. An entry in this table is illustrated in [Figure 8-4](#).

Figure 8-4 The MPLS in-segment MIB table.

```
MplsInSegmentEntry ::= SEQUENCE {  
    mplsInSegmentIfIndex      InterfaceIndexOrZero, -- Index  
    mplsInSegmentLabel        MplsLabel, -- Index  
    mplsInSegmentNPop         Integer32,
```

```
mplsInSegmentAddrFamily      AddressFamilyNumbers,
mplsInSegmentXCIndex        Unsigned32,
mplsInSegmentOwner          MplsInitialCreationSource ,
mplsInSegmentTrafficParamPtr RowPointer,
mplsInSegmentRowStatus       RowStatus,
mplsInSegmentStorageType    StorageType }
```

This table is indexed by a combination of the `ifIndex` of the incoming interface and the topmost label, that is, `mplsInSegmentIfIndex` and `mplsInSegmentLabel`. The number of labels to pop is indicated by the value of `mplsInSegmentNPop`; if this value is 2, then the node pops two labels off the stack. The `mplsInSegmentAddrFamily` gives the Internet Assigned Numbers Authority (IANA) address number; for instance, IPv4 has the value 1 and IPv6 is 2. The cross-connect associated with this segment is provided by the `mplsInSegmentXCIndex`. This is an index into the `mplsXCTable`. The `mplsInSegmentOwner` identifies the entity that created and owns this segment. The `mplsInSegmentTrafficParamPtr` indicates the entry (if any) in the `mplsTrafficParamTable` that contains the traffic details for this segment. The `mplsInSegmentRowStatus` is used when creating, modifying, or deleting an entry in this table. Its type is `RowStatus`, and the ways it can be used are described later in the section where we create an LSP. Finally, the storage type for the segment is described by `mplsInSegmentStorageType`. If this object has the value `readOnly(5)`, then a `setRequest` cannot delete or modify it.

Out-Segments

An out-segment is the egress leg of an LSP segment on a given MPLS NE. This is an object that controls the forwarding of packets along the path of the LSP. Each of the out-segments on an MPLS node has a corresponding entry in the MIB table `mplsOutSegmentTable`. An entry in this table is illustrated in [Figure 8-5](#).

Figure 8-5 The MPLS out-segment MIB table.

```
MplsOutSegmentEntry ::= SEQUENCE {
  mplsOutSegmentIndex      Unsigned32, -- Index
  mplsOutSegmentIfIndex     InterfaceIndexOrZero,
  mplsOutSegmentPushTopLabel TruthValue,
  mplsOutSegmentTopLabel    MplsLabel,
  mplsOutSegmentNextHopIpAddrType InetAddressType,
  mplsOutSegmentNextHopIpAddr   InetAddress,
  mplsOutSegmentXCIndex     Unsigned32,
  mplsOutSegmentOwner       MplsOwner ,
  mplsOutSegmentTrafficParamPtr RowPointer,
  mplsOutSegmentRowStatus    RowStatus,
  mplsOutSegmentStorageType  StorageType }
```

Entries in the out-segment table can be created based on index values obtained from the `mplsOutSegmentIndexNext` object. This object type is described later in this chapter. Once the index value is acquired, we can assign it to `mplsOutSegmentIndex`. The interface index of the outgoing interface is contained in `mplsOutSegmentIfIndex`. The boolean `mplsOutSegmentPushTopLabel` indicates if a label (the value of this label is found in `mplsOutSegmentTopLabel`) should be pushed onto the stack of an outgoing MPLS packet. The `outSegment` is concerned with where to send an outgoing MPLS packet; the type of destination is indicated by the value of `mplsOutSegmentNextHopIpAddrType` and can be IPv4 (1) or IPv6 (2). The `mplsOutSegmentNextHopIpAddr` contains either the IPv4 or IPv6 address of the next hop, depending on the value of `mplsOutSegmentNextHopIpAddrType`. The `mplsOutSegmentXCIndex` indicates the cross-connect table entry with which this segment is associated. The `mplsOutSegmentOwner` identifies the entity that created and owns this segment. The `mplsOutSegmentTrafficParamPtr` indicates the entry (if any) in the `mplsTrafficParamTable` that contains the traffic details for this segment. The `mplsOutSegmentRowStatus` has semantics identical to the corresponding object in the in-segment table. The same is true for the `mplsOutSegmentStorageType`.

Cross-Connects

Cross-connects are used to create associations between LSP segments. These associations serve as instructions for the MPLS NE to switch between the specified segments. The LSR MIB supports point-to-point, point-to-multipoint, and multipoint-to-point connections (we consider only point-to-point). Each of the cross-connects on an MPLS node has a corresponding entry in the MIB table **mplsXCTable**. An entry in this table is illustrated in [Figure 8-6](#).

Figure 8-6 The MPLS cross-connect MIB table.

```
MplsXCEntry ::= SEQUENCE {
    mplsXCIndex          Unsigned32, -- Index
    mplsXCInSegmentIndex InterfaceIndexOrZero, -- Index
    mplsXCInSegmentLabel  MplsLabel, -- Index
    mplsXCOutSegmentIndex Unsigned32, -- Index
    mplsXCLspId           MplsLSPID,
    mplsXCLabelStackIndex Unsigned32,
    mplsXCIsPersistent    TruthValue,
    mplsXCOwner            MplsOwner ,
    mplsXCRowStatus         RowStatus,
    mplsXCStorageType       StorageType,
    mplsXCAdminStatus      INTEGER,
    mplsXCOperStatus        INTEGER }
```

Entries in **mplsXCTable** can be created based on index values obtained from the **mplsXCIndexNext** object. The unique index value is assigned to **mplsXCIndex**. The **mplsXCTable** has an index made up of the first four objects in [Figure 8-6](#). The object **mplsXCInSegmentIndex** represents the in-segment interface index for LSPs not originating on this node. For LSPs originating on this node, **mplsXCInSegmentIndex** is zero. The incoming label value on the cross-connect is **mplsXCInSegmentLabel**. The object **mplsXCOutSegmentIndex** is the out-segment index for LSPs passing through this node. For LSPs terminating on this node, **mplsXCOutSegmentIndex** is zero.

The LSP to which this cross-connect belongs is indicated by the value of **mplsXCLspId**. The object **mplsXCLabelStackIndex** indicates an entry in the label stack table. This indicates the label stack that should be pushed onto the MPLS label. If this cross-connect must be restored after a failure (e.g., a faulty port card or a switch power failure), then the value of **mplsXCIsPersistent** should be set to **true(1)**. The value of **mplsXCOwner** identifies the entity that created and owns this cross-connect. The **mplsXCAdminStatus** object dictates the required administrative state of the cross-connect: **up(1)** means that packets can be forwarded. The value of **mplsXCOperStatus** is set only by the NE to indicate the actual operational status. If a failure occurs, then the value of **mplsXCOperStatus** should reflect it. This means that if an IP port card fails, then the LSP can no longer forward packets and the operational status should change to from **up(1)** to **down(2)**.

Label Stacks

The **mplsLabelStackTable** specifies the label stack to be pushed onto a packet. Entries to this table are referred to from **mplsXCTable** (via the **mplsXCLabelStackIndex** object). The topmost label is the one used by MPLS NEs for forwarding treatment. Labels beneath the topmost label become accessible when the topmost one is popped. This is useful when hierarchical routing behavior is required for a given packet; for example, let's say our label stack has two labels, label X and label Y. An IP packet arrives at MPLS Edge Router 1 in [Figure 8-2](#). At this point the packet is MPLS-encoded and two labels are pushed, first X and then Y. The MPLS packet then proceeds to the next NE, but only the topmost label (Y) is used for forwarding treatment—X remains unchanged. When the MPLS packet reaches the edge of our domain at MPLS Edge Router 2, the topmost label is popped and the remaining label (X) can then be popped and used for additional routing. This type of hierarchical arrangement could be used when routing packets across transit SP networks, such as Interexchange Carriers (IXCs). An entry in this table is illustrated in [Figure 8-7](#).

Figure 8-7 The MPLS label stack MIB table.

```
MplsLabelStackEntry ::= SEQUENCE {
    mplsLabelStackIndex      Unsigned32, -- Index
    mplsLabelStackLabelIndex Unsigned32, -- Secondary Index
    mplsLabelStackLabel      MplsLabel,
    mplsLabelStackRowStatus  RowStatus,
    mplsLabelStackStorageType StorageType }
```

Again, `mplsLabelStackIndexNext` is sampled to give the next free index in this table. This value can then be assigned to `mplsLabelStackIndex`. The object `mplsLabelStackLabelIndex` is a secondary index indicating position within the label stack. A smaller value of `MplsLabelStackLabelIndex` indicates entries higher up the stack. `MplsLabelStackLabel` is the label to be pushed onto the packet.

Traffic Parameters

This table specifies the characteristics of traffic parameter objects for in-segments and out-segments. An entry in this table is illustrated in [Figure 8-8](#).

Figure 8-8 The MPLS traffic parameter MIB table.

```
MplsTrafficParamEntry ::= SEQUENCE {
    mplsTrafficParamIndex      Unsigned32, -- Index
    mplsTrafficParamMaxRate    MplsBitRate,
    mplsTrafficParamMeanRate   MplsBitRate,
    mplsTrafficParamMaxBurstSize MplsBurstSize,
    mplsTrafficParamRowStatus  RowStatus,
    mplsTrafficParamStorageType StorageType }
```

Entries in this table can be created by use of the `mplsTrafficParamIndexNext` object. The value of the latter can be assigned to `mplsTrafficParamIndex`. Each entry in this table can be viewed as a profile that describes the bandwidth characteristics of the associated LSP. The maximum rate in units of kilobits per second is indicated by the value of `mplsTrafficParamMaxRate`. This is the maximum required rate of packet forwarding. Similarly, the mean rate in units of kilobits per second is indicated by the value of `mplsTrafficParamMeanRate`. This is the required average rate of packet forwarding. The maximum burst size in bytes is indicated by the value of `mplsTrafficParamMaxBurstSize`. This is the required maximum burst size expected.

Performance

The LSR MIB includes a number of performance counters. One of these is the `mplsInterfacePerfTable`, which provides an entry for every interface on the LSR capable of supporting MPLS. This table augments the `mplsInterfaceConfEntry` discussed in [Figure 8-3](#). An entry in this table is illustrated in [Figure 8-9](#).

Figure 8-9 The MPLS interface performance MIB table.

```
MplsInterfacePerfEntry ::= SEQUENCE {
    mplsInterfaceInLabelsUsed    Gauge32,
```

```
mplsInterfaceFailedLabelLookup Counter32,  
mplsInterfaceOutLabelsUsed Gauge32,  
mplsInterfaceOutFragments Counter32 }
```

The **mplsInterfaceInLabelsUsed** object counts the number of labels that are in use at this point in time on this interface in the incoming direction. The object **mplsInterfaceFailedLabelLookup** counts the number of MPLS packets that have been received on this interface and were discarded because no matching cross-connect entry was found. Each such occurrence is commonly called a label fault. The object **mplsInterfaceOutLabelsUsed** counts the number of top-most labels in the outgoing label stacks that are in use at this point in time on this interface. The object **mplsInterfaceOutFragments** counts the number of outgoing MPLS packets that required fragmentation before transmission on this interface.

[\[Team LiB \]](#)

[!\[\]\(da8d36bc09d8416b111b870b5e435683_img.jpg\) PREVIOUS](#) [!\[\]\(92ba579d8d63e8879e297bec3f339561_img.jpg\) NEXT !\[\]\(206da16d1cc0fb238bd26898d093f19f_img.jpg\)](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Configuring LSPs Through an LSR

The creation of LSPs, such as the one illustrated in [Figure 8-2](#), involves the following steps:

- Enabling MPLS on MPLS-capable interfaces ([mplsInterfaceConfTable](#)). Our examples assume that this administrative step has been executed.
- Configuring in-segments (via the [mplsInSegmentTable](#)) on LSRs and egress LERs (not needed on ingress LERs).
- Configuring out-segments (via the [mplsOutSegmentTable](#)) on LSRs and ingress LERs (not needed on egress LERs).
- Setting up the cross-connect table to associate segments and/or to indicate connection origination and termination ([mplsXCTable](#)).
- Configuring a tunnel object to point to the cross-connect on the ingress LER.
- Optionally specifying label stack actions ([mplsLabelStackTable](#)).
- Optionally specifying segment traffic parameters ([mplsTrafficParamTable](#)).

We now look at these steps in more detail.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

[PREVIOUS] [NEXT]

Creating an LSP Using the LSR MIB

We now describe the way in which the LSR MIB tables are combined in order to create an LSP. This description is based on that in the IETF draft "Multiprotocol Label Switching (MPLS) Label Switch Router (LSR) Management Information Base." [[IETF-LSR-MPLS](#)]. Our goal with the LSP in [Figure 8-2](#) is to carry the non-real-time SMTP traffic through the MPLS cloud. Since there are no stringent real-time requirements, we can make do with a best-effort QoS for the LSP.

We can fulfill these requirements by creating a best-effort, unidirectional LSP segment that originates in a tunnel on the MPLS Edge Router 1 in [Figure 8-2](#) and exits on interface A via an out-segment. This is not a complete LSP (in the end-to-end sense) but rather an LSP segment. Similar segments must be created in the MIBs of the neighboring devices in order to create the full end-to-end LSP. As can be seen from [Figure 8-2](#), interface A has an `ifIndex` value of 6. The configuration is carried out in three steps. It is important to note that LSPs can be signalled (this takes away many of the manual steps we now describe).

Step 1: Edge Router 1 Cross-Connect and Out-Segment Tables

As we mentioned earlier, three objects are required on an ingress LER:

- A cross-connect
- An out-segment
- A tunnel

We now create one of each of these entities in turn on Edge Router 1.

Edge Router 1 Cross-Connect Creation

A cross-connect entry is required between the tunnel and out-segment objects, respectively. In `mplsXCTable`, we insert the following values:

```
{ mplsXCIndex      = 1, -- The first entry in this table
  mplsXCLsPid     = '05050501'H, -- 5.5.5.1 IP address of the node
  mplsLabelStackIndex = 0, -- only a single outgoing label
  mplsXCIsPersistent = false (1),
  mplsXCRowStatus   = createAndGo(4),
  mplsXCAdminStatus = up(1) }
```

An `mplsOutSegmentTable` row must now be created to point to the appropriate device interface (interface A in [Figure 8-2](#)) and any associated traffic parameter (not relevant for our best-effort LSP).

Edge Router 1 Out-segment Creation

Setting `mplsInSegmentTrafficParamPtr` to zero-dot-zero indicates a best-effort LSP. The same applies to the `mplsOutSegmentTrafficParamPtr` object. In `mplsOutSegmentTable`, we create a row with the following values:

```
{ mplsOutSegmentIndex      = 1,
  mplsOutSegmentIfIndex    = 6, -- outgoing interface A
  mplsOutSegmentPushTopLabel = true(1),
  mplsOutSegmentTopLabel   = 22, -- outgoing label (fed to adjacent node)
  mplsOutSegmentAddrType   = IPv4(1)
  mplsOutSegmentNextHopIpv4Addr = '05050601'H, -- 5.5.6.1, -- Figure 8-2
  mplsOutSegmentXCIndex = mplsXCTable.1, -- Cross-connect just created above
  mplsOutSegmentTrafficParamPtr = 0, -- Best effort
  mplsOutSegmentRowStatus   = createAndGo(4) }
```

Our cross-connect and out-segment objects are now logically associated with each other. To link this aggregate object to the IP domain, we now need a tunnel.

Edge Router 1 Tunnel Creation

To associate IP traffic with our LSP, we must now create a tunnel. In `mplsTunnelTable`, we create a row with the following values:

```
{  mplsTunnelIndexIndex     = 1,
  mplsTunnelInstance       = 1,
  mplsTunnelIngressLSRId  = 5.5.5.1,
  mplsTunnelEgressLSRId   = 5.5.5.1,
  mplsTunnelName          = "LSP",
  mplsTunnelDescr         = "Best-effort for SMTP",
  mplsTunnelIfsIf          = true (1),
```

The following setting assigns the cross-connect 1, ingress interface 0, ingress label 0, and out-segment 1 to the `mplsTunnelXCPPointer` column. The LER then decides which tunnel to use. Label 0 indicates that unlabeled IP traffic is to be received.

```
mplsTunnelXCPPointer     = mplsXCIndex.1.0.0.1,
mplsTunnelSignallingProto = none (1),
mplsTunnelSetupPrio       = 0,
mplsTunnelHoldingPrio     = 0,
mplsTunnelSessionAttributes = 0,
mplsTunnelOwner           = snmp (1),
mplsTunnelLocalProtectInUse = false (0),
mplsTunnelResourcePointer = 0,
mplsTunnelInstancePriority = 1,
mplsTunnelHopTableIndex   = 1,
mplsTunnelPrimaryInstance = 0,
mplsTunnelIncludeAnyAffinity = 0,
mplsTunnelIncludeAllAffinity = 0,
mplsTunnelExcludeAllAffinity = 0,
mplsTunnelRole            = head (1),
-- Mandatory parameters needed to activate the row go here
mplsTunnelRowStatus       = createAndGo (4) }
```

This completes the configuration required for the creation of an originating LSP on the MPLS Edge Router 1 in [Figure 8-2](#).

We now move to the next node in line, Core Router 1 ([Figure 8-2](#)).

Step 2: Core Router 1 Segment and Cross-Connect Tables

We must now create an in-segment, out-segment, and cross-connect on Core Router 1.

Core Router 1 In-segment Creation

In the MIB on Core Router 1, we set the following values in the `mplsInSegmentTable`:

```
{ mplsInSegmentIfIndex = 7, -- interface index (B) value for the transit LSP
  mplsInSegmentLabel = 22, -- incoming label value from Edge Router 1
  mplsInSegmentNPop = 1, -- default value
  mplsInSegmentAddrFamily = IPv4(1)
  mplsInSegmentXCIndex = mplsXCTable.6, -- Please see Cross-connect section
  mplsInSegmentTrafficParamPtr = 0, -- Best effort
  mplsInSegmentRowStatus = createAndGo(4) }
```

The first two objects—`mplsInSegmentIfIndex` and `mplsInSegmentLabel`—are set to the values 7 and 22, respectively, to tie in with the originating segment of the LSP on Edge Router 1.

Core Router 1 Out-Segment Creation

In `mplsOutSegmentTable`,

```
{ mplsOutSegmentIndex = 1,
  mplsOutSegmentIfIndex = 8, -- outgoing interface C
  mplsOutSegmentPushTopLabel = true(1),
  mplsOutSegmentTopLabel = 0, -- outgoing label explicit null, 3 = Implicit null
  mplsOutSegmentAddrType = IPv4(1)
  mplsOutSegmentNextHopIpv4Addr = '05050701'H, -- 5.5.7.1, -- Figure 8-2
  mplsOutSegmentXCIndex = mplsXCTable.6, -- Please see Cross-connect section
  mplsOutSegmentTrafficParamPtr = 0, -- Best effort
  mplsOutSegmentRowStatus = createAndGo(4) }
```

The next step consists of configuring the cross-connect table.

Core Router 1 Cross-Connect Creation

A cross-connect entry is now created, thereby associating the newly created segments together. In `mplsXCTable`, we insert the following values:

```
{ mplsXCIndex = 6,
  mplsXCLspId = '05050601'H, -- 5.5.6.1
  mplsLabelStackIndex = 0, -- only a single outgoing label
  mplsXCIsPersistent = false (1),
  mplsXCRowStatus = createAndGo(4),
  mplsXCAdminStatus = up(1) }
```

This completes the configuration required for the creation of a transit LSP segment on the MPLS Core Router 1 in [Figure 8-2](#). We now create the last remaining segment on Edge Router 2.

Step 3: Edge Router 2 Cross-Connect and In-Segment Tables

We must now create a cross-connect and in-segment on Edge Router 2.

Edge Router 2 Cross-Connect

In `mplsXCTable`, we insert the following values:

```
{ mplsXCIndex      = 8,
  mplsXCLspId     = '05050701'H, -- 5.5.7.1
  mplsLabelStackIndex = 0, -- only a single outgoing label
  mplsXCIsPersistent = false (1),
  mplsXCRowStatus   = createAndGo(4),
  mplsXCAdminStatus = up(1) }
```

Finally, an in-segment must be created.

Edge Router 2 In-segment Table

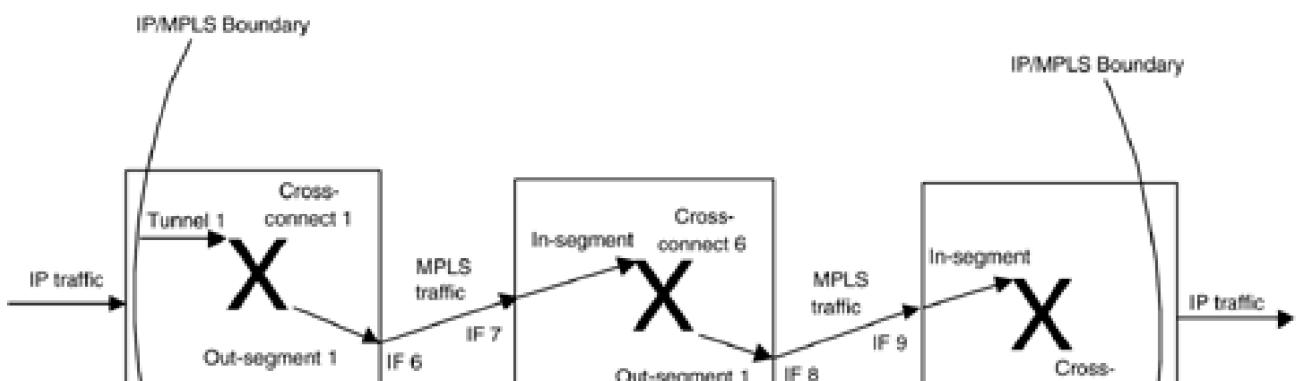
In the MIB on the Edge Router 2, we set the following values in the `mplsInSegmentTable`:

```
{ mplsInSegmentIfIndex = 9, -- interface (D) index value for the terminating LSP
  mplsInSegmentLabel  = 0, -- incoming label value from Core Router 1
  mplsInSegmentNPop   = 1, -- default value
  mplsInSegmentAddrFamily = IPv4(1)
  mplsInSegmentXCIndex = mplsXCTable.8, -- Please see Cross-connect section
  mplsInSegmentTrafficParamPtr = 0, -- Best effort
  mplsInSegmentRowStatus = createAndGo(4) }
```

The first two objects—`mplsInSegmentIfIndex` and `mplsInSegmentLabel`—are set to the values 9 and 0, respectively, to tie in with the out-segment configuration on Core Router 1.

This completes the configuration required for the creation of a terminating LSP on the MPLS Edge Router 2 in [Figure 8-2](#). After these steps, the LSP looks something like that illustrated in [Figure 8-10](#).

Figure 8-10. Logical description of the LSP.





We now describe a brief walkthrough of what happens to the IP traffic landing on the IP/MPLS boundary on the left-hand side of [Figure 8-10](#):

1. A packet with destination IP address 155.154.4.2 arrives at Edge Router 1 ingress interface.
2. The ingress interface pushes the packet into Tunnel 1.
3. Tunnel 1 serves this IP address via the LSP illustrated in [Figure 8-10](#).
4. The packet is MPLS-encapsulated with the label value 22.
5. Label value 22 is associated with outgoing interface index 6.
6. The MPLS packet is pushed out interface index 6.
7. At Core Router 1, the incoming packet with a label value of 22 arrives at interface index 7.
8. The label value of 22 is used to direct the packet to Core Router 1 interface index 8. The label value of 22 is then replaced with a new label value of zero (explicit null—some MPLS devices may use 3 for implicit null).
9. The MPLS packet arrives at Edge Router 2 interface index 9 with a label value of zero. The latter is a signal to Edge Router 2 to strip off the label and perform a normal IP lookup.
10. Edge Router 2 forwards the packet to its original destination, the SMTP Gateway at 155.154.4.2.

We now turn to the TE MIB to take a look at the traffic-engineering MPLS objects.

Traffic-Engineered Tunnels

The TE MIB differs from the LSR MIB in a number of ways. The TE MIB provides a model for a traffic-engineered tunnel through an MPLS cloud; in other words, it provides an end-to-end picture. The LSR MIB deals only in terms of segments and cross-connects, and lacks an end-to-end perspective (though use of a signaling protocol, such as LDP, can compensate for this). The only real difference between a tunnel and an LSP is that the former is explicitly routed. Both can reserve resources and (depending on the implementation) both may support IntServ, DiffServ, and any other QoS models.

Tunnels can be created at the head-end (or originating) node (Edge Router 1 in [Figure 8-2](#)), and the MIBs for all nodes in the path are automatically updated via a signaling protocol (such as RSVP). This is a powerful facility because it provides a simple way of managing tunnels, using just a few MIB tables. The end result is a traffic-engineered tunnel that traverses an entire network. Tunnels can also be created with or without the assistance of a signaling protocol (such as LDP or RSVP-TE). The TE MIB supports five tables that are used for creating tunnels:

- Tunnel table ([mplsTunnelTable](#)), used for recording the tunnel parameters
- Resource table ([mplsTunnelResourceTable](#)), used for configuring end-to-end tunnel resources, such as required bandwidth
- Tunnel hop tables for the specified, actual, and computed route ([mplsTunnelHopTable](#), [mplsTunnelARHopTable](#), and [mplsTunnelCHopTable](#)) for strict and loose source routed tunnels

The tunnel hops indicate the exact route that will be taken. We now describe these tables.

The mplsTunnelTable

[Figure 8-11](#) illustrates a row from the mplsTunnelTable definition.

Figure 8-11 The MPLS TE tunnel table.

```
MplsTunnelEntry ::= SEQUENCE {
1   mplsTunnelIndex      MplsTunnelIndex, -- Index
2   mplsTunnelInstance    MplsTunnelInstanceIndex,
3   mplsTunnelIngressLSRId  MplsLsrlIdentifier, Index
4   mplsTunnelEgressLSRId  MplsLsrlIdentifier, Index
5   mplsTunnelName        DisplayString,
6   mplsTunnelDescr       DisplayString,
7   mplsTunnelIsIf         TruthValue,
8   mplsTunnelIfIndex     InterfaceIndexOrZero,
9   mplsTunnelXCPPointer  RowPointer,
10  mplsTunnelSignallingProto  INTEGER,
11  mplsTunnelSetupPrio    INTEGER,
12  mplsTunnelHoldingPrio  INTEGER,
13  mplsTunnelSessionAttributes  BITS,
14  mplsTunnelOwner        INTEGER,
15  mplsTunnelLocalProtectInUse  TruthValue,
16  mplsTunnelResourcePointer RowPointer,
17  mplsTunnelInstancePriority Unsigned32,
18  mplsTunnelHopTableIndex  MplsPathIndexOrZero,
19  mplsTunnelARHopTableIndex MplsPathIndexOrZero,
20  mplsTunnelCHopTableIndex MplsPathIndexOrZero,
21  mplsTunnelPrimaryInstance  MplsTunnelInstanceIndex,
22  mplsTunnelPrimaryTimeUp  TimeTicks,
23  mplsTunnelPathChanges    Counter32,
24  mplsTunnelLastPathChange TimeTicks,
25  mplsTunnelCreationTime  TimeStamp,
26  mplsTunnelStateTransitions Counter32,
27  mplsTunnelIncludeAnyAffinity MplsTunnelAffinity,
28  mplsTunnelIncludeAllAffinity MplsTunnelAffinity,
29  mplsTunnelExcludeAllAffinity MplsTunnelAffinity,
30  mplsTunnelPathInUse      MplsPathIndexOrZero,
31  mplsTunnelRole          INTEGER,
32  mplsTunnelTotalUpTime   TimeTicks,
33  mplsTunnelInstanceUpTime TimeTicks,
34  mplsTunnelAdminStatus   INTEGER, -- Set by the operator
35  mplsTunnelOperStatus    INTEGER, -- Set by the NE
36  mplsTunnelRowStatus     RowStatus,
```

```
37 mplsTunnelStorageType     StorageType }
```

Because this is a large table, we inserted line numbers to make the description a little easier to follow. The objects are described as far as possible in related groups.

Tunnel Table Lines 1 to 6

Each row in the tunnel table has a unique index identified by **mplsTunnelIndex**. This (in conjunction with the value of **mplsTunnelInstance**) is how each tunnel is differentiated by the NMS. For tunnel configurations that require protection (or load sharing), a tunnel instance can represent a backup copy of another tunnel. The object **mplsTunnelInstance** is used for this purpose. The latter then refers to another completely separate entry in the tunnel table. Multiple tunnel instances can also be used for load sharing. Each such tunnel instance can have its own path and reserved resources. The tunnel is also identified in the network by the **mplsTunnelIngressLSRId** object. The egress router for this tunnel is identified by **mplsTunnelEgressLSRId**. The tunnel name is **mplsTunnelName** and is useful (in conjunction with **mplsTunnelIndex**) for differentiating between many tunnels on a given node. Other information about the tunnel can be stored in **mplsTunnelDescr**.

Tunnel Table Lines 7 and 8

A tunnel that corresponds to an interface is indicated by the value of the boolean **mplsTunnelIsIf**. If **mplsTunnelIsIf** is set to true, then **mplsTunnelIfIndex** contains the value of **ifIndex** from the associated entry in **theIfTable**.

Tunnel Table Lines 9 and 10

The variable **mplsTunnelXCPPointer** points to a row in the **mplsXCTable** (not needed for signaled tunnels). The **mplsTunnelXCPPointer** serves to join a tunnel to the cross connect on the ingress node. We saw this in the LSP configuration example. Signaled tunnels store the value of the signaling protocol in **mplsTunnelSignallingProto**, for example, **none(1)** and **rsvp(2)**.

Tunnel Table Lines 11 to 15

The setup priority of the tunnel is stored in **mplsTunnelSetupPrio**—a high value for this means that a tunnel instance may pre-empt (i.e., tear down) another tunnel instance. A related object is **mplsTunnelHoldingPrio**, which sets the resistance to being torn down by another tunnel instance. Additional characteristics for the tunnel instance are specified in **mplsTunnelSessionAttributes**; for instance, **isPersistent(2)** indicates that this tunnel instance should be restored automatically following a failure (e.g., power down). The tunnel instance creator is indicated by the value of **mplsTunnelOwner**; for instance, **admin(1)** indicates a manual administrator such as an NMS user. **mplsTunnelLocalProtectInUse** indicates that the tunnel instance can be repaired in the event of a link failure on this node.

Tunnel Table Line 16

The **mplsTunnelResourcePointer** indicates the entry in the **mplsTunnelResourceTable** used by this tunnel. Assigning

`mplsTunnelResourcePointer` the value `zeroDotZero` indicates best-effort treatment. When we create a traffic-engineered tunnel, we will see how the resources are used.

Tunnel Table Line 17

Zero is the lowest value that can be assigned to `mplsTunnelInstancePriority`. This object can be used to differentiate between groups of tunnel instances; for example, if all instances have the same priority value, then they can load-share traffic with the same destination address.

Tunnel Table Lines 18 to 20

The `mplsTunnelHopTableIndex` indicates the entry in the hop table used by this tunnel. This indicates the administratively defined, required route for the tunnel. The `mplsTunnelARHopTableIndex` indicates the entry in the actual hop table used by this tunnel (this indicates the real route through the network taken by the tunnel). The `mplsTunnelCHopTableIndex` indicates an index into the computed hop table.

Tunnel Table Lines 21 to 25

The primary instance index of this tunnel is indicated by `mplsTunnelPrimaryInstance`—this can be used to logically tie together all the tunnels instances. The time the primary instance has been active is indicated by `mplsTunnelPrimaryTimeUp`—this might be useful for billing and accounting purposes. Changes in the network (such as link failures) may result in a tunnel rerouting; `mplsTunnelPathChanges` records the number of such changes. The time since the last such path change is recorded in `mplsTunnelLastPathChange`. The `mplsTunnelCreationTime` records the value of `SysUpTime` (a member of the system table illustrated in the example programs in the previous chapter) when the first instance of this tunnel was created.

Tunnel Table Line 26

The overall number of tunnel state transitions (e.g., administrative and operational status changing from up to down) is recorded in `mplsTunnelStateTransitions`.

Tunnel Table Lines 27 to 29

To provide finer control over which links a tunnel traverses, the `mplsTunnelIncludeAnyAffinity` constraint can be employed. Links are administratively assigned constraint values (often called colors). The tunnel uses a given link only if either the constraint is zero (i.e., no constraint) or the link fulfills at least one of the specified constraints. For example, in a tunnel that uses only links that have *any* of the colors gold and silver, any links that have either gold or silver can be included in the tunnel. The object `mplsTunnelIncludeAllAffinity` is similar except that it specifies the colors that a link must have for inclusion. For example, in a tunnel that uses only links that have *all* of the colors gold and silver, any links that have both gold and silver can be included in the tunnel. The object `mplsTunnelExcludeAllAffinity` has similar semantics to `mplsTunnelIncludeAllAffinity` except that it indicates the constraints that *must not* be fulfilled. For example, a tunnel that uses only links that do *not* have all of the colors gold and silver, depending on the implementation, link coloring may only apply to

computed-path tunnels. These are tunnels with paths that are ultimately dictated by the network rather than by the values in an ERO.

Service providers could use colors to differentiate service—for instance, gold service can mean a higher level of service in terms of bandwidth, latency, jitter, and delay.

Tunnel Table Line 30

The `mplsTunnelPathInUse` object provides an index into the `mplsTunnelHopTable` and indicates the path that was chosen for the tunnel.

Tunnel Table Lines 31 to 33

The value of `mplsTunnelRole` reflects the role played by the host node in the overall tunnel—an originating tunnel has `mplsTunnelRole` set to `head(1)`, a transit tunnel has the value `transit(2)`, while a terminating tunnel has the value `tail(3)`. The aggregate up time for all instances of a tunnel is indicated by the value of `mplsTunnelTotalUpTime`. Similarly, `mplsTunnelInstanceUpTime` indicates the up time for this tunnel instance.

Tunnel Table Lines 34 and 35

The administratively assigned operational status of the tunnel is indicated by the value of `mplsTunnelAdminStatus`. This can have values such as `up(1)`, `down(2)`, or `testing(3)`. The actual operational status of the tunnel is indicated by the value of `mplsTunnelOperStatus` and can have values such as `up(1)`, `down(2)`, `testing(3)`, and `unknown(4)`.

Tunnel Table Lines 36 and 37

The `mplsTunnelRowStatus` is used to manage SNMP row operations against an entry in the `mplsTunnelTable` and has the same semantics as for the other tables. The object `mplsTunnelStorageType` also follows the usage from other tables.

The `mplsTunnelResourceTable`

The `mplsTunnelResourceTable` is used to indicate the resources (bandwidth, burst size, etc.) required for a given tunnel. When the tunnel is created across the network, the required resources are explicitly reserved for its use. An entry in this table is illustrated in [Figure 8-12](#).

Figure 8-12 The MPLS TE tunnel resource table.

```
MplsTunnelResourceEntry ::= SEQUENCE {
    mplsTunnelResourceIndex          Unsigned32, -- Index
    mplsTunnelResourceMaxRate        MplsBitRate,
```

```
mplsTunnelResourceMeanRate      MplsBitRate,
mplsTunnelResourceMaxBurstSize  MplsBurstSize,
mplsTunnelResourceMeanBurstSize MplsBurstSize,
mplsTunnelResourceExcessBurstSize MplsBurstSize,
mplsTunnelResourceFrequency    INTEGER,
mplsTunnelResourceWeight       Unsigned32,
mplsTunnelResourceRowStatus    RowStatus,
mplsTunnelResourceStorageType  StorageType }
```

The `mplsTunnelResourceIndex` object uniquely identifies a tunnel resource block (i.e., a row) in this table. The maximum traffic rate is indicated (in units of bits per second) by the value of `mplsTunnelResourceMaxRate`. The average traffic rate is indicated by `mplsTunnelResourceMeanRate`. The `mplsTunnelResourceMaxBurstSize` in bytes specifies the maximum expected burst size. The `mplsTunnelResourceMeanBurstSize` specifies the average expected burst size. Another important traffic characteristic is `mplsTunnelResourceExcessBurstSize`. The availability of the tunnel resources is indicated by `mplsTunnelResourceFrequency` and can have the values `unspecified(1)`, `frequent(2)`, or `veryFrequent(3)`. If the resources are not available at a given point in time, the result might be dropped or marked packets. The latter depends on the underlying platform network hardware. The `mplsTunnelResourceWeight` indicates the relative weight for using excess bandwidth above the reserved level. The `mplsTunnelResourceRowStatus` and `mplsTunnelResourceStorageType` follow the semantics seen in the other tables.

The `mplsTunnelHopTable`

LSPs can be given pre-assigned routes or paths. These are referred to as Explicit Route Objects (EROs) and serve to indicate a set of hops that is traversed by a tunnel instance. An entry in this table is illustrated in [Figure 8-13](#).

Figure 8-13 The MPLS TE tunnel hop table.

```
MplsTunnelHopEntry ::= SEQUENCE {
  mplsTunnelHopListIndex      MplsPathIndex, -- Index
  mplsTunnelHopOptionIndex    MplsPathIndex, -- Index
  mplsTunnelHopIndex          MplsPathIndex, -- Index
  mplsTunnelHopAddrType       INTEGER,
  mplsTunnelHopIpv4Addr      InetAddressIPv4,
  mplsTunnelHopIpv4PrefixLen Unsigned32,
  mplsTunnelHopIpv6Addr      InetAddressIPv6,
  mplsTunnelHopIpv6PrefixLen Unsigned32,
  mplsTunnelHopAsNumber       Unsigned32,
  mplsTunnelHopLspId          MplsLSPID,
  mplsTunnelHopType           INTEGER,
  mplsTunnelHopIncludeExclude INTEGER,
  mplsTunnelHopPathOptionName DisplayString,
  mplsTunnelHopEntryPathComp  INTEGER,
  mplsTunnelHopRowStatus      RowStatus,
  mplsTunnelHopStorageType    StorageType }
```

New index values in the `mplsTunnelHopTable` can be obtained from the `mplsTunnelHopListIndexNext` object. The `mplsTunnelHopListIndex` object uniquely identifies an ERO in this table. A particular set of hops (called a path option) is indicated by `mplsTunnelHopPathOptionIndex`. A specific hop in the table is indicated by the value of `mplsTunnelHopIndex`. The type of a tunnel hop is indicated by `mplsTunnelHopAddrType` and can have the value `ipV4(1)`, `ipV6(2)`, `asNumber(3)`, or `lspid(4)`. The value of `mplsTunnelHopAddrType` dictates the values in the next six objects.

If `mplsTunnelHopAddrType` is `ipV4(1)`, then `mplsTunnelHopIpv4Addr` and `mplsTunnelHopIpv4PrefixLen`, respectively, contain the value of an IPv4 address and its prefix length. If `mplsTunnelHopAddrType` is `ipV6(2)`, then `mplsTunnelHopIpv6Addr` and `mplsTunnelHopIpv6PrefixLen`, respectively, contain the value of an IPv6 address and its prefix length. If `mplsTunnelHopAddrType` is `asNumber(3)`, then `mplsTunnelHopAsNumber` contains the autonomous system number of this hop. Finally, if `mplsTunnelHopAddrType` is

`lspid(4)`, then `mplsTunnelHopLsPid` contains the LSPID of a tunnel of this hop.

The `mplsTunnelHopType` indicates if this tunnel hop is routed in either a strict (every hop is configured) or loose (where not all hops are configured—the path may include other networks) fashion.

The `mplsTunnelHopIncludeExclude` indicates if the current hop is to be excluded from the tunnel route computation. It can have the value `include(1)` or `exclude(2)`.

The `mplsTunnelHopPathOptionName` describes the sequence of hops in relation to the tunnel path. If the operator requires a dynamically computed tunnel path, then the value of `mplsTunnelHopEntryPathComp` should be set to `dynamic(1)`. This setting passes the responsibility for path determination over to the network. A value of `explicit(2)` indicates that the operator is providing the path via an ERO. Finally, the `mplsTunnelHopRowStatus` and `mplsTunnelHopStorageType` follow the semantics for these object types in other tables.

[\[Team LiB \]](#)

[◀ PREVIOUS](#)

[NEXT ▶](#)

Creating a Tunnel Using the TE MIB

An MPLS tunnel is an explicitly routed LSP. Tunnels are created using the TE MIB and can reserve resources as well as follow strict or loose explicit routes. The NEs in the tunnel path configure the appropriate LER and LSR MIB objects in order to fulfill the required constraints. Tunnels can be created on just a single NE, and all the other components are automatically generated across the network.

Configuring the `mplsTunnelTable`

Referring to [Figure 8-2](#), we now construct an entry in the `mplsTunnelTable`. Since this is a bigger table than those in the previous sections, comments are included along with the column assignments. Most of the settings correspond to default MIB values.

```
{
  mplsTunnelIndex      = 1, -- We assume this is the first table entry
  mplsTunnelInstance   = 1,
  mplsTunnelIngressLSRId = 5.5.5.1, -- Edge Router 1
  mplsTunnelEgressLSRId = 5.5.7.1, -- Edge Router 2
  mplsTunnelName       = "TE Tunnel", -- as in Figure 8-2
  mplsTunnelDescr      = "A traffic-engineered tunnel",
  mplsTunnelIsIf       = true (1), - Tunnel will appear in ifTable
  mplsTunnelXCPPointer = 0, -- Not needed for signaled tunnels
  mplsTunnelSignallingProto = rsvp (2),
  mplsTunnelSetupPrio   = 0,
  mplsTunnelHoldingPrio = 0,
  mplsTunnelSessionAttributes = 0,
  mplsTunnelOwner       = snmp (1),
  mplsTunnelLocalProtectInUse = false (0),
  mplsTunnelResourcePointer = mplsTunnelResourceIndex.5, -- Created below
  mplsTunnelInstancePriority = 1,
  mplsTunnelHopTableIndex = 1, -- This ERO is created below
  mplsTunnelPrimaryInstance = 0,
  mplsTunnelIncludeAnyAffinity = 0, -- No link coloring
  mplsTunnelIncludeAllAffinity = 0, -- No link coloring
  mplsTunnelExcludeAllAffinity = 0, -- No link coloring
  mplsTunnelPathInUse     = 1,
  mplsTunnelRole          = head (1), -- The originating end of the tunnel
  mplsTunnelRowStatus     = createAndGo (4)
}
```

Since this is a traffic-engineered tunnel, we must specify both the resources and the nodes required in the path. This is done next. Strictly speaking, the `mplsTunnelTable` entry would be created *after* the resources have been specified and the hop list created.

Configuring the `mplsTunnelResourceTable`

Our tunnel will have associated with it a set of resources that provides it with the ability to carry the traffic pushed into it. The resources are defined by a row in the `mplsTunnelResourceTable`. We arbitrarily select the index entry for this table as number 5 (e.g., this could be the next free value obtained from sampling the `mplsTunnelResourceIndexNext` object). This is set as follows:

```
{ mplsTunnelResourceIndex      = 5,
  mplsTunnelResourceMaxRate    = 640000, -- 10 * 64kbps voice channels
  mplsTunnelResourceMeanRate   = 640000,
  mplsTunnelResourceMaxBurstSize = 2000,
  mplsTunnelResourceRowStatus  = createAndGo (4) }
```

All five of these objects could be included in an SNMP `setRequest` message variable binding list. In our examples in [Chapter 7](#), we included just a single object, but it is possible to include many objects in a single message. The last object to be set is the row status. This is an indication to the remote agent that the operation is a row creation. The value 4 is called `createAndGo` and indicates that the row is to be created with a status of active; that is, the row is to be ready for service.

Configuring the `mplsTunnelHopTable` (ERO)

The following steps create the hops associated with an ERO that is used by the tunnel. In `mplsTunnelHopTable`, the following entries are created:

[\[View full width\]](#)

```
{ mplsTunnelHopListIndex  = 1,
  mplsTunnelPathOptionIndex = 1,
  mplsTunnelHopIndex      = 1,
  mplsTunnelHopAddrType   = 1,
  mplsTunnelHopIpv4Addr   = 5.5.5.1, -- Node IP address of Edge Router 1
  mplsTunnelHopIpv4PrefixLen = 9,
  mplsTunnelHopType       = strict (1),
  mplsTunnelHopRowStatus  = createAndGo (4) }
{ mplsTunnelHopListIndex  = 1,
  mplsTunnelPathOptionIndex = 1,
  mplsTunnelHopIndex      = 2,
  mplsTunnelHopAddrType   = 1,
  mplsTunnelHopIpv4Addr   = 5.5.4.2, -- Ingress interface IP address on
  ➔ Core Router 2
  mplsTunnelHopIpv4PrefixLen = 9,
  mplsTunnelHopType       = strict (1),
  mplsTunnelHopRowStatus  = createAndGo (4) }
{ mplsTunnelHopListIndex  = 1,
  mplsTunnelPathOptionIndex = 1,
  mplsTunnelHopIndex      = 3,
  mplsTunnelHopAddrType   = 1,
  mplsTunnelHopIpv4Addr   = 5.5.5.2, -- Ingress interface IP address on
  ➔ Edge Router 2
  mplsTunnelHopIpv4PrefixLen = 9,
  mplsTunnelHopType       = strict (1),
  mplsTunnelHopRowStatus  = createAndGo (4) }
```

This ERO can then be incorporated into the tunnel by setting `mplsTunnelHopTableIndex = 1`.

The packet processing for the traffic-engineered tunnel is, in fact, very similar to that shown in [Figure 8-10](#). The main differences are that:

- A different tunnel is chosen.
- The tunnel has an associated resource reservation.
- Signaling and an ERO dictated the tunnel path.

This completes our description of creating LSPs and traffic-engineered tunnels.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Creating LSPs and Tunnels Using an NMS

The detailed steps we've described for creating LSPs and tunnels are manual in nature. An NMS would substantially reduce the workload involved in creating these objects. We now describe a possible workflow for both cases:

1. Open a topology map containing the MPLS nodes in [Figure 8-2](#).
2. Click on the two edge nodes, selecting each as a connection endpoint.
3. Select the connection type (LSP or tunnel).
4. If the tunnel type is selected, then the GUI should allow the creation of an ERO.
5. Optionally request a path computation (trivial in [Figure 8-2](#) but more difficult when hundreds, thousands, or even hundreds of thousands of nodes are involved) between the two end nodes.
6. Select the required resources (bandwidth, maximum burst size, etc.).
7. View the computed path and modify it if necessary.
8. Accept the path and send the connection for provisioning

At this point, all the necessary MIB tables are updated by NMS provisioning software. For the case of the tunnel, just the ingress node (Edge Router 1) MIB needs to be updated, with the remainder being signaled. For the LSP, if no signaling is used, then the NMS provisioning code must painstakingly update the MIBs of all the nodes in the path (Edge Router 1, Core Router 1, and Edge Router 2). Finally, the NMS should then report on the success or failure of the overall operation. Beyond this, the NMS should then start to manage the created connections as *owned* objects—objects that it created. This differentiates between connections provisioned using the EMS (i.e., outside the NMS, using the CLI) rather than the NMS.

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

NextObjectIndex and Synchronization

Several of the MPLS MIB tables support a `nextObjectIndex` counter (this was introduced in [Chapter 3](#), "The Network Management Problem"). This provides a convenient means of learning the next available free entry in the table. A manager simply has to poll the value of this object to learn the next free index. An important aspect of this is synchronization when more than one user may attempt to use the same free index. This could occur if two clients simultaneously attempted to use the same value. The result of this would be that one of the two clients would inadvertently cause an SNMP bad value exception to occur. One solution to this would be for the clients to channel their requests for this object value through the NMS. The latter could then implement a synchronization procedure to ensure that multiple requests are queued. In this way, the clients are guaranteed that the index value they receive is unique. Alternatively, the intelligence can be put in the MIB so that when the object is retrieved, it is also automatically incremented. This is perhaps slightly less desirable because it means that a read of the object value results in a change. Another approach is to use the `TestAndIncr` type. In this, an object (let's call it `indexSpinLock`) of type `TestAndIncr` is used to store the current next available free index. The NMS reads the value of `indexSpinLock` and creates a new table row. At the same time, we also set the retrieved value `dhdexSpinLock` (into the object `indexSpinLock`). If the current value of `indexSpinLock` and the value we set it to do not match, then the SNMP request will fail.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

A Note About Standards

In the above discussion, we have used two of the IETF MPLS MIBs to illustrate the use of LSPs, tunnels, and their associated objects. An important point to note about standards in general is that they are detailed specifications that form engineering guidelines for implementers. If all vendors implement their MPLS products to conform to the standards, then the reward should be interoperability and uniformity of features in deployed technology. Mature standards such as IEEE 802.3 and ATM (though there are proprietary variants of the ATM control plane) provide a good example of the power of standards to produce interoperable NEs. The same can possibly be said for many of the SS7 protocol standards (though there are national variants of these). Let's hope MPLS proves to be both interoperable and generally easy to use.

The great merit of standard MIBs is that the network management software required to manipulate them need not change as new NEs are added. This facilitates ease of development of management solutions and potentially far fewer incompatibility problems in production networks.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Summary

We've now seen a detailed description of two of the IETF MPLS MIBs, indicating how these MIB objects might be manipulated by an NMS in order to create LSPs and tunnels. These MIBs differ in the level of abstraction offered—the TE MIB provides a useful tunnel metaphor. The LSR MIB requires that up to three tables be manipulated on each agent in the LSP path. These are the in-segment, out-segment, and cross-connect tables respectively, with the addition of the tunnel table on the ingress node. Manipulating just one head-end (or ingress LER) agent can achieve the same result with the TE MIB. The tables used in the latter case consist of a tunnel table, a hop table, and a resource table. The user of an NMS would generally not see the detailed manipulation of these MIB tables. Instead, the NMS would provide a simple interface that offers a limited set of choices, for example, either an LSP or a tunnel as the required connection type. Simplifying such operations is one of the many merits of effective NMS products. The many tables needed for MPLS provisioning underlines the importance of good table insertion discipline. This is particularly so in mult-client provisioning situations. The merit of vendor adoption of standard MIBs is considerable because NEs become easier to manage and to understand.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Chapter 9. Network Management Theory and Practice

As with many areas of human endeavor, the gap that separates the theory and practice of NMS development is very wide. Viewed against the backdrop of rapidly evolving networks and increasingly sophisticated end-user requirements, the problems facing the vendors of NMS products are considerable. Not least is the simple problem of scale in the management plane: Emerging NEs are big, dense, and complex, incorporating a growing range of technologies. In this final chapter, we try to draw together the main threads running through the book and revisit some of them now that our foundation chapters are complete. The discussion covers the following main areas:

- MIBs—how careful design can greatly assist management ([nextFreeIndex](#), single table on originating node, and default values).
- MIBs and scalability—the size of emerging NEs may generate a need for some form of compressed MIB data.
- Decision-making in the network—pushing more decisions out of the NMS and into the network. We examine the MPLS FTN MIB to see an example of this; also, Policy-Based Network Management (PBNM) is useful in this context.
- Pushing FCAPS into the network.
- Generic objects realized using software abstraction.
- The increasing need for end-to-end security.
- Shrink-wrapped solutions or consultancy buy-in.
- Integration with OSS layers.
- The roles of QA, IT, and developers.
- Solutions revisited—thin software layers.
- Facilitating a solution mindset.

We start with yet another MIB-related detour: The issue of storing policies in MIBs is introduced, followed by a description of intercolumn relationships.

[Team LiB]

◀ PREVIOUS NEXT ▶

MIBs Again

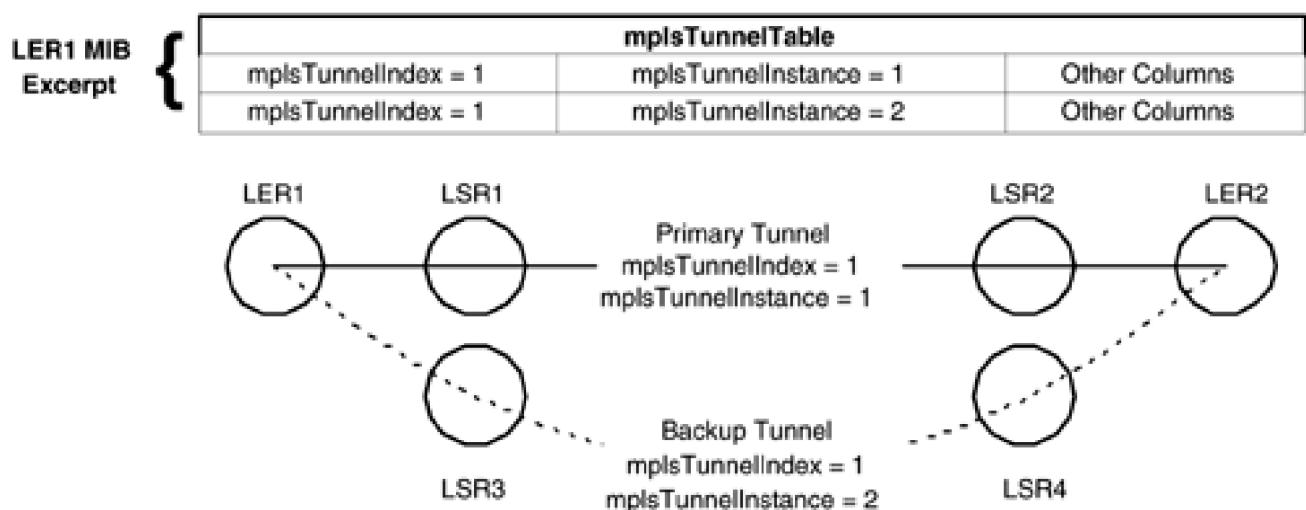
In [Chapter 8](#), "Case Study: MPLS Network Management," we studied in some depth the structure and use of two of the IETF MPLS MIBs. These MIBs have been well-designed; for example, a signaled traffic-engineered tunnel created using the `mplsTunnelTable` can be managed with reference just to the originating node MIB. In other words, it is not necessary to browse every node in the path of the tunnel, because the necessary details are stored in the originating node. This helps improve the manageability and scalability of the MPLS network. In effect, the NMS can manage such tunnels via the LERs in an MPLS network. This has other benefits: LERs are often more powerful devices than the LSRs in the core of the network, so they are potentially more able to withstand large bursts of management plane traffic, for instance, when the tunnels are being discovered. Another important point is that the path taken by the tunnels through the network is also stored in the LER in the (optional) `mplsTunnelARHopTable`. If this table is supported, then the originating node knows the path. This again avoids the need for delving into the transit LSR MIBs.

All of the MIBs we've studied so far have been essentially passive in nature; that is, they serve to record details of the managed network. There is another use that can be made of a MIB: policy storage. In this, the MIB is used to store rules and actions. Policies consist of conditions (or rules) and actions that should be taken when the conditions are met. Later in this chapter, we study the FTN MIB because it provides a framework for storing policies that can be used to manage IP traffic as it enters an MPLS network.

Tightly Coupled Intercolumn Dependencies

An important area of MIB design is that of intercolumn dependency in which the value of column X provides a context for column Y, or vice versa. An example is where a tunnel instance is a backup for a primary tunnel as illustrated in [Figure 9-1](#).

Figure 9-1. A primary tunnel with a backup instance.



In [Figure 9-1](#), we see an MPLS network with two tunnels. One is a primary and the other is a backup. This can be brought about by creating two entries in the `mplsTunnelTable`, one of which is an instance of the other. By instance, we mean a copy in all things except the path taken. The two tunnels can be configured to share the same set of resources, such as bandwidth, or they can each duplicate the resources. The primary tunnel follows the path {LER1, LSR1, LSR2, LER2}, while the backup follows the path: {LER1, LSR3, LSR4, LER2}. In [Figure 9-1](#), we also see an excerpt from the `mplsTunnelTable` in LER1. The primary tunnel has the value 1 in both `mplsTunnelIndex` and `mplsTunnellInstance`. In other words, it is the first entry in the `mplsTunnelTable` and is not an instance of another

tunnel (i.e., it is an instance of itself). The backup tunnel, however, has the value 1 in `mplsTunnellIndex` and 2 in `mplsTunnellInstance`. This means that it occupies the second entry in the `mplsTunnelTable` and is an instance of `mplsTunnellIndex` 1. In other words, it is a backup instance of the primary tunnel.

Let's consider the steps that must be taken to bring this about using SNMP. To create the primary tunnel, we set the values of `mplsTunnellIndex` and `mplsTunnellInstance` both to 1 (as well as setting other mandatory columns, as was seen in [Chapter 8](#)). It is likely that we will have looked up the value of the `mplsTunnellIndexNext` object to get the next free `mplsTunnelTable` index. To create the backup tunnel, we must consult the MIB (or an external database) for the value of `mplsTunnellIndex` that corresponds to the primary tunnel—in this case, 1. We then create another entry in `mplsTunnelTable` with the `mplsTunnellIndex` and `mplsTunnellInstance` values 1 and 2, respectively.

So, the value of `mplsTunnellInstance` serves to indicate if the tunnel is an instance of another tunnel. If two (or more) rows in the tunnel table have the same value of `mplsTunnellIndex` and different values of `mplsTunnellInstance`, then they are instances of each other. The tunnel instances can act as backups to each other, or they can load-share incoming traffic.

This is an example of intercolumn dependencies in which the value of one column depends on the value of another. In the case of backup (or load-sharing) tunnels, the value of `mplsTunnellIndex` has the same value as `mplsTunnellIndex` from another row in the `mplsTunnelTable`. The two entries are differentiated by the value of `mplsTunnellInstance`. Such dependencies contribute some complexity to the MIB. For example, should we be allowed to delete a primary tunnel before deleting the backup? Usually not, because the backup generally exists only to protect the primary tunnel. So, the agent on LER1 should enforce this, and the NMS should follow suit.

As with many engineering decisions, the best way to implement this is with clear rules, such as precluding deletion of a primary tunnel until all instances have been deleted. The agent should enforce these rules along with the NMS (e.g., if the user attempts to delete a primary tunnel before deleting the backup). It is a bad practice for the NMS to rely on the agent to enforce such rules—the agent may erroneously permit inconsistencies. For this reason, it is better for the NMS to infer relationships like tunnel instances and enforce rules concerning the order of deletion without relying on the agent.

Another important issue is that of providing default values for MIB objects. This can have an important impact on the complexity of the SNMP-handling software in an NMS, as we'll see in the next section.

Default Values and Thin Software Layers

If two MIB columns depend semantically on each other, then it is good practice to have default values via the `DEFVAL` clause. To illustrate, let's assume we have a MIB where two columns, X and Y, have a relationship with each other. Let's say X has no default value and can take the values 1 and 2, but these have meaning only if Y has a nonzero value. In other words, if Y has the value zero, then we should not set any value in X. This is a bad MIB design for a few reasons:

- The correct value of X is undefined if Y is zero.
- SNMP-handling software code in the NMS must check the value of Y before setting X.
- Not setting the value of X (e.g., when Y is zero) may give rise to holes in the MIB.

Having to check the value of an object makes the SNMP code unnecessarily complex. It makes flow-through operations more difficult to achieve because the incoming data has to be validated—this should already have occurred at the user interface layer. It also introduces special cases into the NMS software. If such intercolumn relationships are necessary, then it should be possible to use default values in the MIB object definition via the `DEFVAL` clause. Once this is done, the values of X and Y are irrelevant to the SNMP (provisioning) code: It simply sets the values passed to it regardless of whether or not they are defaults. This is so because the values will either be defaults or valid settings.

The issue of holes is important. These can arise if a `SET` operation is completed on a table row without setting all the columns. It is up to the agent to provide some default value if it deems it necessary. The problem with MIB holes (as we saw in [Chapter 6](#), "Network Management Software Components") is that a `getRequest` on a MIB hole can result in an exception; likewise `getNextRequest` on a MIB hole results in getting the lexical successor to the specified object skipping any holes. This can result in unexpected data being presented to the NMS. Providing default values can also help in avoiding MIB holes. When holes are avoided, it becomes easier to navigate around a MIB.

An allied problem occurs in the use of relational database products (e.g., Informix, SQL Server) when null values have been written into

tables. Retrieving such null data using Java can result in exceptions being thrown. This can be highly inconvenient because it then becomes necessary to catch such exceptions.

An added difficulty is that exception handling in languages such as Java can be quite slow. Just as for MIBs, it is generally better practice to avoid the problem altogether by the use of default (i.e., not null) schema values in the table definitions.

MIBs and Scalability

The crucial role played by MIBs in network management has been stated many times. MIBs are in fact so crucial that they can greatly simplify both the structure of the NMS and the ease with which the network can be managed.

The scale of emerging NEs is such that SNMP may be approaching a physical limit—navigating tables with millions of entries is almost certainly not a practical proposition. MIB designs must incorporate this trend and allow for possible techniques such as data compression. Compressed PDUs could use standard data compression techniques (e.g., LZ77) in order to manipulate larger amounts of data. In effect, larger PDUs could be used because each field could be compressed. On the downside, this would complicate PDU handling and make for slower NE responses because of compression overhead. A more permanent solution to this is to push more management decision-making capability into the NEs themselves, as discussed in the next section.

Decision-Making in the Network

The mapping of IP packets into the MPLS domain is a nontrivial task. The increasingly high speed and volume of IP packet feeds across enterprise networks is a compelling reason for moving individual packet-handling decisions outside of the NMS. Yet another important MPLS MIB—the FTN MIB [[IETF-MPLS-FTN](#)][IETF-MPLS-FTN](#)—provides a framework for this and is now described.

The MPLS FTN MIB

The full title of this MIB is a little unwieldy: MPLS FEC-To-NHLFE Management Information Base. An understanding of this MIB should help us gain a deeper appreciation of the MPLS examples described in [Chapter 8](#). It will also illustrate a way of storing policies in MIBs. These policies are created by the NMS user and executed by the NE (usually in conjunction with special-purpose network hardware). Before starting to describe the MIB, we define the term Forwarding Equivalence Class (FEC).

A FEC is a group of IP packets that receive the same forwarding treatment. A FEC dictates that packets follow the same path through the network and experience a defined quality of service. A FEC might correspond to a destination IP subnet or address prefix (e.g., 10.81/16), but it also might correspond to any traffic class that a given Edge-LSR (or LER) considers significant. For example, all traffic to a given destination with a certain value of IP DS field might constitute a FEC.

An analogy for a FEC is international immigration in an airport. Non-nationals are separated out from nationals by the passport they hold. Two queues are formed, one for nationals and another for non-nationals. The nationals queue is usually much faster than the one for the non-nationals. In this case, the FEC is dictated by passport and the forwarding behavior is much faster for nationals, that is, a faster queue. A conceptually similar mechanism exists at the point of ingress to an MPLS network.

FEC Definition

At the point of ingress to an MPLS network, packets are assigned to a forwarding equivalence class or FEC.

A FEC is a group of IP packets that are forwarded in the same manner, that is, over the same path, and with the same traffic-handling treatment. In conventional IP routing, FEC assignment occurs at each hop; in MPLS it occurs just once at the ingress. Once a packet has been assigned to a FEC, it is labeled. The labeled packet is then ready for MPLS forwarding.

Once an IP packet has been labeled, the MPLS node must decide where to send (or forward) it. The next hop label forwarding entry (NHLFE) is used for this purpose and contains the following details:

- The next hop for the packet—an LSP or a tunnel
- The operation to be performed on the label stack

Recall from [Chapter 8](#) that an LSP is an object created using the LSR (and TE) MIB. MPLS-encoded packets pushed onto an LSP then follow the path associated with that LSP. Similarly, the next hop can be a traffic-engineered tunnel (created using our old friend, the MPLS tunnel table MIB). The label stack operation can be one of the following:

- Replace the label at the top of the label stack with a specified new label.
- Pop the label stack.
- Replace the label at the top of the label stack with a specified new label, and then push one or more specified new labels onto the stack.

When a packet matches a particular rule, a corresponding action is executed, such as forwarding or discarding it. Other actions are possible, such as modifying the DS field (in a process called remarking) or redirection of the packet to a specific outgoing interface.

The next part of the FTN MIB concerns the association between the packet-handling rules and specific NE interfaces. The last table in the MIB provides performance-related statistics—useful for checking the speed of packet handling, throughput, and so on.

This is the broad functional description of the FTN MIB; we now look at the details of the following three tables:

1. mplsFTNTTable
2. mplsFTNMapTable
3. mplsFTNPerfTable

The **mplsFTNTTable** is used to store mappings between FECs and NHLFE. Each row defines a rule to apply to incoming IP packets and an action to take if the rule applies. The criteria for rule construction can consist of the following objects:

- Source IP address (version 4 or 6)
- Destination IP address (version 4 or 6)
- Source port
- Destination port
- DS value

These are all fields in the IP packet header, as we saw in [Chapter 3](#), "The Network Management Problem," [Figure 3-5](#). Another object, called the action pointer, serves to point at an entry in either the LSR MIB (**mplsXCEEntry**) or the TE MIB (**mplsTunnelEntry**).

The **mplsFTNMapTable** is used to activate or map FTN entries defined in **mplsFTNTTable** to specific interfaces. FTN entries are compared with incoming packets in the order in which they are applied on an interface. The **mplsFTNMapTable** supports a linked-list structure of FTN entries. The order of this list dictates the order of application of the associated policies on a given interface. So, if two FTNs, ftn1 and ftn2, are associated with an interface, then IP packets are processed against the settings in ftn1 and then ftn2.

Finally, the **mplsFTNPerfTable** provides performance counters for each FTN entry on a per-interface basis. Because LERs are located at the boundary of IP and MPLS networks, the traffic levels can be very high (e.g., an SP boundary connected to a large corporate site), so there is a need for high-capacity counters in order to avoid 32-bit counters wrapping around (although wraparound is clearly still possible).

Example Using the FTN MIB

This example illustrates the FTN MIB setup required for pushing MPLS-encoded IP traffic into either an LSP or a tunnel. [Figure 9-2](#) illustrates two IP traffic streams feeding into an MPLS LER (Edge Router 1). One IP source is sending voice-over-IP (VoIP) telephony traffic, and the other is SMTP (email distribution). We want to push the SMTP traffic through the LSP and the VoIP traffic through the tunnel. The VoIP traffic has real-time requirements, so let's assume that we have created the tunnel with adequate bandwidth and an appropriate assigned QoS (as we saw in [Chapter 8](#)). The SMTP traffic requirements are less stringent, so we use an LSP for this purpose, with no bandwidth resource allocation and a best-effort QoS. The tunnel, however, has to carry real-time telephony data, so we assume that the tunnel has dedicated resources (e.g., 640kbps as we saw in [Chapter 8](#)).

Figure 9-2. FTN MIB setup for incoming IP traffic.

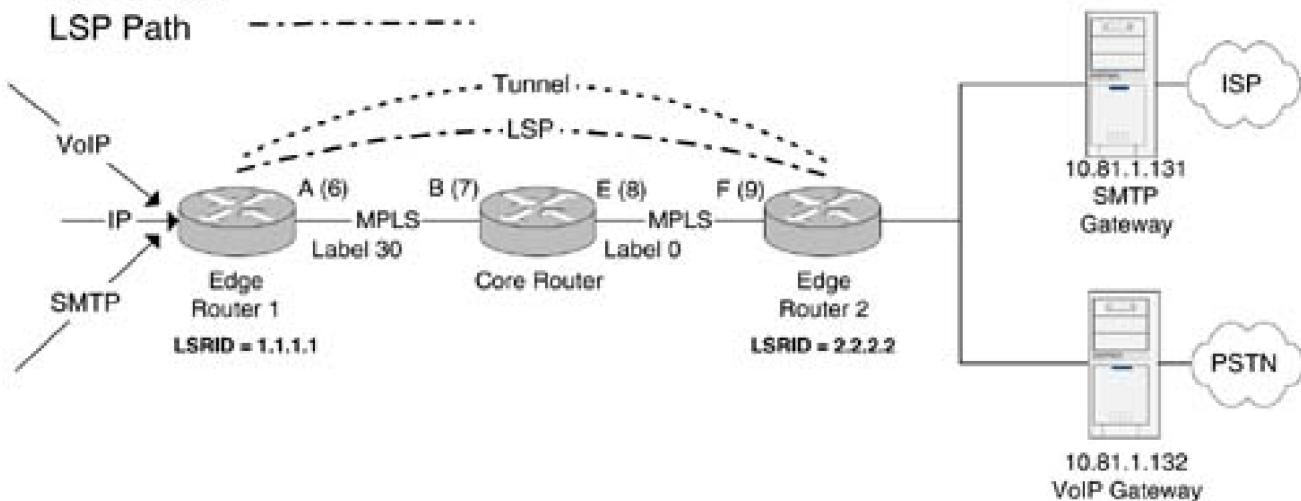
Legend

MPLS Interfaces are marked alphabetically as A, B, C, D, E, and F

ifIndex values are shown in brackets

Tunnel Path -----

LSP Path - - - - -



The LSP and tunnel are capable of transferring MPLS-encapsulated IP packets through the core network and delivering them as IP at the point of egress (Edge Router 2).

In this case, we have two IP destinations: the SMTP Gateway at IP address 10.81.1.131 and a VoIP Gateway at 10.81.1.132, respectively. The setup we illustrate in [Figure 9-2](#) is unidirectional (a telephony application would require bidirectional connections); to complete the VoIP picture, we would need another tunnel (or LSP) to forward traffic in the opposite direction.

As can be seen in [Figure 9-2](#), the egress MPLS label used by the core router has the reserved value 0. This value is called Explicit Null and is used in order to indicate to the next MPLS node (Edge Router 2) that the MPLS data must be stripped off the packet and a normal IP lookup performed. In other words, the label value of 0 tells the next node that the packet must be returned to the IP domain. The following example illustrates how the different IP traffic types are pushed into either the LSP or tunnel.

Setting Up the mplsFTNTTable for LSP Redirection

In order to push IP traffic into the LSP in [Figure 9-2](#), an entry is required in the `mplsFTNTTable`. The LSP setup requires the network administrator to know in advance the values of the following objects at each hop:

- Ingress interface index (not needed at the ingress LER)
- Incoming label (if applicable—not needed at the ingress LER)
- Egress interface index (not needed at the egress LER)
- Egress label (if applicable—not needed at the egress LER)

[Figure 9-2](#) illustrates the MIB objects needed for setting up the `mplsFTNTable`. These objects are required only for Edge Router 1 and consist of the following:

- The incoming label is not applicable because IP traffic lands at the ingress interface (and hence has no attached MPLS label)
- Egress interface index value of 6 (interface A)
- Egress label value of 30

Given these details, we now have enough information to populate a row in `mplsFTNTable`:

```
{ mplsFTNIndex = 1,
  mplsFTNDescr = "FTN-ENTRY-1 for IP subnet 10.81.0.0",
  mplsFTNMask = 0x01, -- Look up destination address only
  mplsFTNAddrType = ipv4,
  mplsFTNDestIpv4AddrMin = 10.81.1.131,
  mplsFTNDestIpv4AddrMax = 10.81.1.131,
  mplsFTNActionType = redirectLsp,
  mplsFTNActionPointer = mplsXCLsId.5.0.0.3 }
```

The value of `mplsFTNActionPointer` indicates the LSP to which packets should be redirected. It is set to point to the first column object of the XC entry that corresponds to this LSP. This is `mplsXCIndex.5.0.0.3`, which represents the following `mplsXCTable` entry:

```
{ mplsXCIndex = 5,
  mplsInSegmentIfIndex = 0, -- originating LSP
  mplsInSegmentLabel = 0, -- originating LSP
  mplsOutSegmentIndex = 3, -- pointer to a row in mplsOutSegmentTable
  mplsXCLabelStackIndex = 0 }
```

This `mplsXCTable` entry in turn points to the following row in the `mplsOutSegmentTable`:

```
{ mplsOutSegmentIndex = 3,
  mplsOutSegmentIfIndex = 6,
  mplsOutSegmentPushTopLabel = true,
  mplsOutSegmentTopLabel = 30 -- Egress label value }
```

As can be seen, the values in `mplsOutSegmentTable` match those illustrated in [Figure 9-2](#). Finally, we have `mplsFTNMapTable`, which activates the FTN entry:

```
{ mplsFTNMapIfIndex = 1,
  mplsFTNPrevIndex = 0, -- The first FTN entry on this interface
  mplsFTNMapCurIndex = 1 }
```

IP packets with the destination address 10.81.1.131 are now redirected into the LSP as required.

Setting Up the `mplsFTNTable` for Tunnel Redirection

In order to push IP traffic into the Tunnel in [Figure 9-2](#), another entry is required in the `mplsFTNTable`. We give this row the index value 2.

```
{ mplsFTNIndex = 2,
  mplsFTNDescr = "FTN-ENTRY-2 for IP subnet 10.81.0.0",
  mplsFTNMask = 0x01, -- Look up destination address only
  mplsFTNAddrType = ipv4,
  mplsFTNDestIpv4AddrMin = 10.81.1.132,
  mplsFTNDestIpv4AddrMax = 10.81.1.132,
  mplsFTNActionType = redirectTunnel,
  -- We assume that the ingress and egress LSR IDs are 1.1.1.1 and
  -- 2.2.2.2 respectively for this tunnel as seen in Figure 9–2
  mplsFTNActionPointer = mplsTunnelIndex.4.0.1.1.1.2.2.2.2 }
```

In `mplsTunnelTable`, we have the following row with index 4:

```
{ mplsTunnelIndex = 4,
  mplsTunnelInstance = 0, -- primary tunnel
  mplsTunnelIngressLSRID = 1.1.1.1,
  mplsTunnelEgressLSRID = 2.2.2.2 }
```

Finally, we have `mplsFTNMapTable`, which activates the FTN entry:

```
{ mplsFTNMapIfIndex = 1,
  mplsFTNPrevIndex = 1,
  mplsFTNMapCurrIndex = 2 }
```

IP packets with the destination address 10.81.1.132 are now redirected into the traffic-engineered tunnel as required.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Intelligence in the Network: Manufacturing

In many ways the present generation of NMS (i.e., NMS/EMS and SNMP entities/agents) exhibit some similarity to the problems attached to the automation and control of manufacturing systems in the 1980s and 1990s. The paucity of local intelligence in manufacturing systems put a great strain on centralized management and control systems, and the need for distributed intelligence was compelling.

One solution to those problems was to use local intelligence in networked controllers (similar to SNMP agents). The latter would then use local sensors and low-cost processing power wherever it was needed rather than in a central location. These distributed controllers then only reported serious problems back to a central supervisory management system. This arrangement freed the central management system to perform more complex (and centralized) calculations, such as scheduling production runs and reporting on scrap.

It is increasingly likely that a similar approach will be needed for NMS, that is, more and more agent intelligence. PBNM provides a basis for this by allowing for NEs to take some control responsibility. The FTN MIB provides an SNMP-based example of policy usage.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Pushing FCAPS into the Network

The FTN MIB provides an SNMP-based example of policy usage. Other types of decision-making can be pushed into the network. One area is that of billing and accounting. Many service providers use a flat-rate billing model because the deployed NEs and management facilities cannot provide a usage-based approach. Usage-based billing allows for improved SP margins and network resource use. The Lightweight Flow Accounting Protocol (LFAP) from Riverstone is an effort to provide for more accurate billing and accounting in the NEs themselves. This is similar to the provision of CDR/PDR facilities that we saw in [Chapter 6](#). The merit of LFAP is that it may become a standard.

Over time, we may expect to see more of the FCAPS being standardized and implemented in NEs. This would free the NMS/OSS to do very advanced functions, such as:

- Root-cause analysis
- Asset management
- Usage-based billing
- Capacity planning and forecasting

In other words, as FCAPS capability is pushed into the network (along with policy-based facilities), we may see some OSS (or possibly business management system) facilities being pushed in turn into the NMS.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Service-level Network Components

Throughout this book, we have seen a range of aggregate objects, including VPNs, VLANs, and LSPs. Aggregate objects combine base-level components to create some type of higher level service. An example is an NE that supports IEEE 802.1Q-based VLANs. Such an NE facilitates the creation of VLANs that span more than one device. In other words, several components in the network combine together into an aggregate object that provides a network-level service (in this case, a VLAN). Service providers now offer layer 3 VPNs (e.g., based on RFC 2547) as a revenue-generating service to corporate customers.

Managing these complex services is a major challenge, and doing so in a scalable fashion remains one of the biggest problems faced by the industry. It is possible that new MIBs will be needed to represent these aggregate objects, and realizing them in the network may well require new signaling protocols. An example scenario might be when a service provider wants to add a customer site to a VPN. The steps might include the following:

- Create a virtual circuit (e.g., MPLS, ATM, and FR) from the CPE to a PE router.
- Map the traffic on this circuit to an MPLS core.
- Map the QoS characteristics of the incoming traffic to the provide core.
- Ensure that this traffic goes to a specified set of destinations inside the VPN.
- Provide a verifiable SLA to the customer.
- Finally, before creating the service, ensure that encryption/authentication is in place.

The NMS will almost certainly be called upon to provide this type of multitechnology solution.

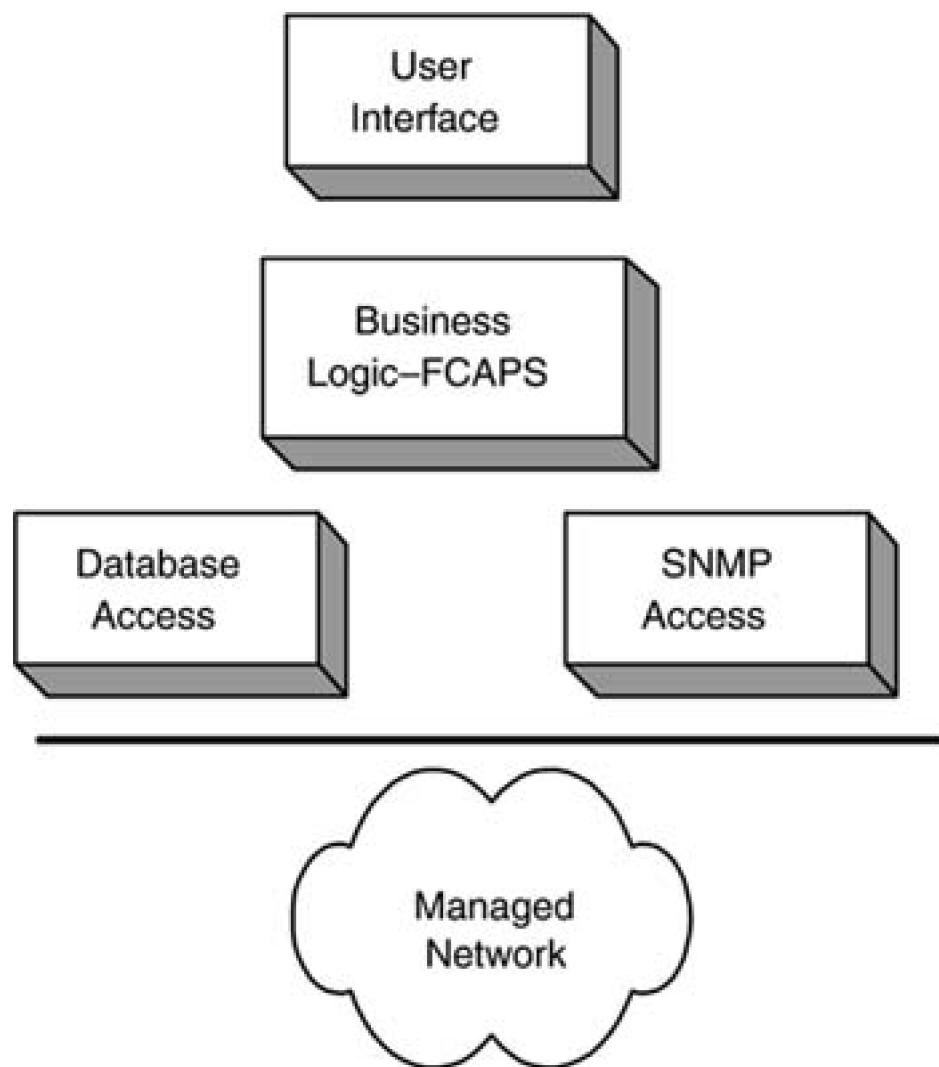
[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Generic Objects Realized Using Software Abstraction

The increasing mix of technologies deployed in enterprise networks places a growing burden on NMS. The software components used to realize NMS must become increasingly abstract. This needs to occur at all levels of the software, with technology specifics cleanly separated in their own layers. So, when application code needs access to NEs via SNMP, all calls should be made to separate code. In other words, business logic should not be mixed with network device technology access code. The simple structure illustrated in [Figure 9-3](#) provides an idea of this demarcation.

Figure 9-3. Demarcation between NMS components.



All code written to access specific technology should also be as generic as possible; for example, it is better to name a class method (assuming the code is written in C++ or Java) `getLabelValue()` than `getMPLSLabelValue()` because the former can be used for a number of label-based technologies, such as ATM, MPLS, FR, and Pseudo-Wires, whereas the latter is tied to MPLS. In a similar vein, code used to create virtual connections should use naming conventions such as `createConnection()` rather than `createFrameRelayConnection()`. The latter is again tied to one technology, whereas the former is more flexible and can be extended to support other technology types as they arise. Parameters in these function calls (or Java methods) can be used to distinguish the underlying technology types. So, we might have a top-level provisioning code method (with an abbreviated parameter list), such as:

```
createConnection(int technologyType, addressType sourceAddr, addressType destAddr, ...)
```

Internal to `createConnection()`, we can distinguish between the technologies using the `technologyType` parameter. The value of this parameter is then passed to another (technology-specific) method that actually creates the required low-level virtual connection using a Java switch statement as follows:

```
switch (technologyType)
{
case (ATM_SPVX)
createATMConnection(ATM_SPVX, sourceAddr, destAddr, ...);
break;
case (FRAME_RELAY_PVX)
createFrameRelayConnection (FRAME_RELAY_PVX, sourceAddr, destAddr, ...);
break;
case (MPLS_LSP)
createMPLSConnection (MPLS_LSP, sourceAddr, destAddr, ...);
break;
case (MPLS_TE_TUNNEL)
createMPLSConnection (MPLS_TE_TUNNEL, sourceAddr, destAddr, ...);
break;
}
```

The key point is generic outer code. We get specific about technology only at a well-defined point in the code rather than mix in potentially many different technology types at an early stage. An added bonus of this is that subsequent changes required to one of the low-level methods, such as `createMPLSConnection()`, do not have any effect on the other methods (such as the ones for Frame Relay or ATM).

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

The Need for End-to-End Security

The international terrorist threat has had a significant effect on the area of network management. The need for security is now paramount (along with the need for disaster recovery planning and service survivability). Security is needed at every level of a network, from the NE and subtended (i.e., attached) equipment all the way up to the NMS server and client machines. This can be called end-to-end security, the purpose of which is to protect the network and its data from unauthorized access. Connecting to an NE EMS should employ authentication and encryption, making certain that little or no clear text is exchanged. The same holds true for exchanges between an NMS and EMS, OSS and NMS, and so on.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Shrink-Wrapped Solutions or Consultancy Buy-In

One of the consequences of globalization is a greater degree of skills specialization and homogeneity of products. In modern corporations, it is increasingly easy to relocate and move between different host countries, often working in the same (or a similar) job and for the same organization. This means that software development skill sets can be used to move fairly easily between different geographical locations. In other words, software developers can become quite specialized. In a similar fashion, the vast range of commercially available NMS products share a great many functional characteristics. So, we see a great many products competing for a finite number of customers. Software developers with broadly similar skill sets create these products. Increasingly specialized software development skill sets may have a downside. This is particularly the case as the number of developers in vendor organizations is reduced. Those developers who remain may not have a broad enough vision for effectively creating NMS products.

Even without reduction in numbers, skill set specialization also has its own problems when adopting solution engineering, such as:

- Not taking an end-to-end or customer-type system view—for example, a developer creates a Frame Relay virtual circuit, verifying that the data is written to the database but not the network (a customer generally sees the network as the true database and is more interested in verifying that the connection is created in the network).
- Not taking consideration of a feature beyond the current release cycle.

Also, NMS products (and NEs) are increasingly homogeneous, often offering base-level features such as fault and performance management. Many vendor (and system integrator) organizations make sizeable sums of money in selling consultancy to network operators. This consultancy is often geared toward assisting a network operator in incorporating a given NMS product into its workflows and business processes. In effect, if consultancy is offered as part of a product sale, then the vendor is trying to add differentiation in this way. This seems a cumbersome and very expensive approach, given the relative ease with which modern software tools can be used to create highly useable software.

A better deployment model results if NMS products are well-designed with characteristics such as:

- High-quality (or standard) MIBs
- Generic software components, such as GUIs that allow the management of generic connections rather than technology-specific objects, whether they be optical light paths or Frame Relay virtual circuits
- Flow-through provisioning features with thin software layers
- Adherence to standard NBIs (discussed in the next section)

We believe that products fulfilling these requirements have a much better chance of fitting into enterprise networks, workflows, and business processes, and standing on their own merits. In other words, incorporating such products into large enterprise networks should not be such a daunting and expensive task as is perhaps the case at present.

[Team LiB]

◀ PREVIOUS NEXT ▶

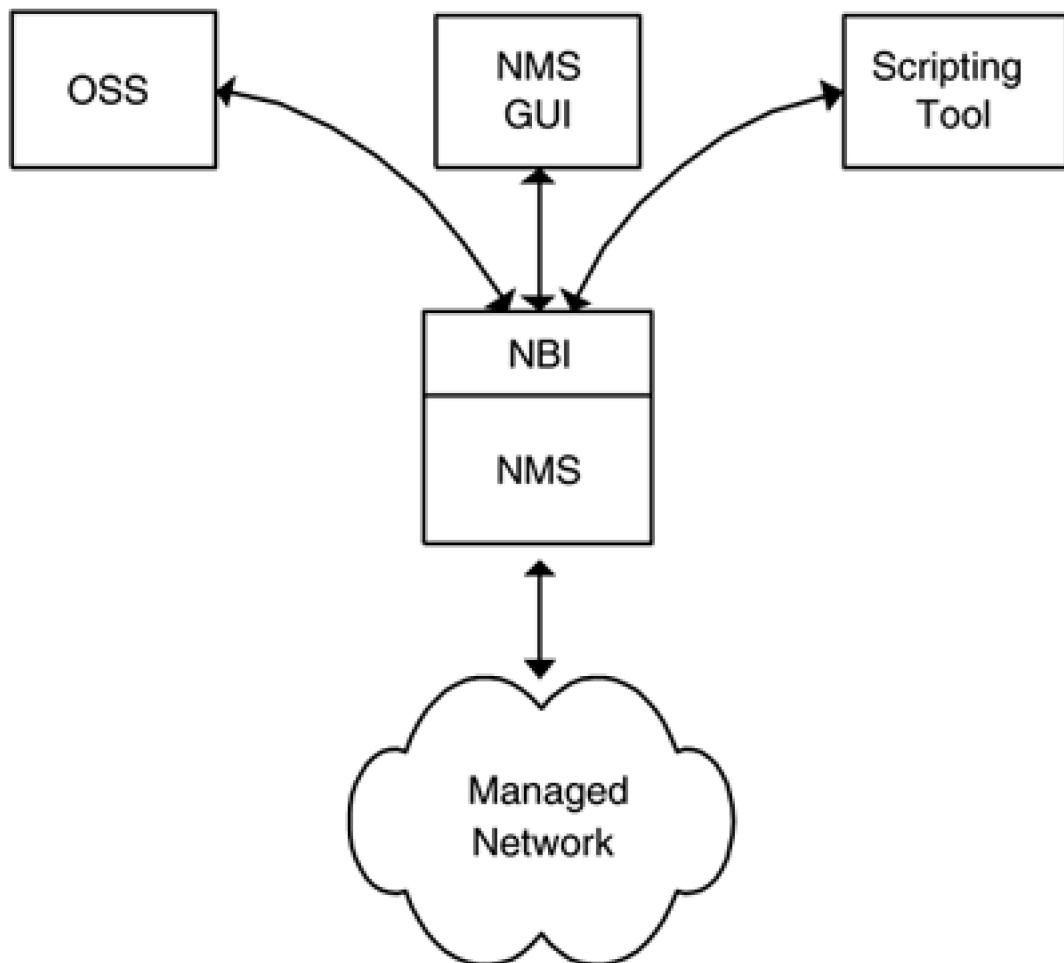
Integration with OSS Layers: Northbound Interface (NBI)

Communication between the OSS and NMS is crucial to the successful management of large SP networks. In the same way as NMS needs to communicate with EMS, the OSS needs to be able to communicate with the NMS. The latter is facilitated using the NBI. Broadly speaking, there are two ways of implementing an NBI layer:

- Put software in the OSS layer
- Put software in the NMS

A recurring theme in this book is the need to keep software layers as thin as possible. This applies to all layers, including the section of the OSS that interacts with the NMS. To this end, the ideal arrangement is for the NMS and OSS to use the same code. If no OSS is present, as would be the case for an enterprise network, then the user interface is the NMS GUI or a scripting tool. If an OSS is present, then the user interface is the NBI. The key point is that the underlying code is the same in both cases. This is illustrated in [Figure 9-4](#).

Figure 9-4. Code sharing between OSS and NMS layers.



The investment required in introducing the NBI layer in [Figure 9-4](#) is worthwhile because of the ease with which OSS integration can occur.

[Team LiB]

◀ PREVIOUS NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Developer Note: The Roles of QA, IT, and Developers

Close cooperation is needed in vendor organizations to deliver NMS products. Developers should delegate the administration of NEs to IT and also involve QA in every step of the development process. Developers should also learn to acquire deep testing ability in order to ensure that the software delivered to QA does not fail. In such a scheme, QA really is assuring quality rather than simply carrying out what often amounts to software integration testing. Developers should already have successfully completed the latter. Developers then become true knowledge workers—delegating NE administration to the experts (IT) and partnering with QA to ensure solution development. A further development of this might be to involve customer engineers in the QA process. The motivation for this is the rapid development of high-grade network management solutions.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Solutions Revisited: Thin Software Layers

Much mention has been made in this book about thin software layers in the client, middleware, and server components of NMS. Why is this a desirable proposition?

- Thin software has a small number of lines of code.
- Thin software is simple—little or no spaghetti (excessively complex) code.
- Thin software is fast and easy to modify, maintain, and test.
- Thin software spreads complexity over adjacent layers just as is done in network protocol layers (as seen [Figure 9-3](#)).

Thin software strikes a balance between form and function—the code size and complexity are minimized while the overall function is optimized. Code size is minimized by the use of details like meaningful default database values and flow-through provisioning. These in turn help avoid spaghetti code because, for example, the data sent for provisioning is valid for passing directly into SNMP `setRequest` messages. That is, the provisioning code does not need to validate the data it receives; instead, it can be written straight out to the network. The same applies where the NMS carries out extensive reads from the network, for example, during an IP discovery procedure. MIB objects are read from NEs and these match expected column values in the database. So, in the same way as for provisioning, the discovery code does not have to carry out large amounts of validation and data manipulation.

[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Facilitating a Solution Mindset

We have spoken extensively of solutions and the mindset needed for their creation. Since solutions form such an important element of NMS products, it is important to describe how to facilitate such a mindset. To do this, we now delve a little more deeply into the way solution engineers operate:

- Engineers focus on products and not just projects.
- Ownership is taken of large functional product areas, such as reporting subsystems, provisioning, and security (i.e., one or more of the FCAPS areas).
- A strategic interest is adopted beyond the current software release cycle.

Modern NMS products provide much scope for solution engineering on account of their complexity. The great challenge for solution engineers lies in harnessing the power of software abstraction to provide a simple, flexible interface to possibly nonexpert end users. The labyrinthine complexity of emerging NEs can be hidden by such generic NMS software designs. This notion extends the philosophy and spirit of SNMP into the NMS architecture itself.

Engineers who focus on products rather than individual projects tend to take the time to master their chosen area. This can be any or all of the FCAPS areas, for example. A product is a freestanding body of software that exists as either a substantial element of a product or as a product in its own right. An example of such a product is an accounting subsystem that allows for billing of IP packet traffic, ATM cell traffic, and so on. Product engineers tend to adopt a broad perspective.

Project engineers tend to focus on many small, well-defined pieces of work, and they often play an extremely useful role in getting software releases successfully out to customers. Project engineers differ in enjoying the strategic context of several release cycles and use this to inform their implementation decisions. Project engineers generally produce the best solutions. An added bonus of project engineers is that they can also materially contribute to strategy groups.

Facilitating a solution engineer can consist of little more than interviewing them and asking them about their desired work area and preferred working method. We regard product and solution engineers as being essentially the same.

[Team LiB]

◀ PREVIOUS NEXT ▶

A Final Word

It is increasingly difficult for enterprises to countenance throwing out all of their legacy hardware and management software in order to install the latest device offerings. There is a need for allowing such users to migrate slowly and steadily towards the packet-based networks of the future. The cost of hardware is falling, and for this reason good-quality NMS provide a degree of vendor product differentiation.

Another interesting aspect of consolidation and ongoing procurement is that SP equipment inventory will become increasingly similar. Service providers will find it more difficult to differentiate their services using hardware alone. NMS will offer a useful means of gaining competitive advantage. These considerations apply to both network operators and NE vendors.

The central role of MIBs in network management has been a major theme of this book, and we hope that MIBs now hold no surprises for readers. Vendor and standards organizations can do much to promote manageability by creating well-designed MIBs. Wherever possible, standard MIBs should be used.

Pushing more intelligence into NEs can be readily accommodated with MIBs such as the FTN MIB discussed in this chapter. It is possible that specialized networking hardware such as the network processors from Intel and IBM may be required for such MIBs. However, the pattern is clear: NEs will become increasingly sophisticated and capable of autonomously executing highly complex management functions. This will help to improve the scalability of networks that encompass many (i.e., hundreds or thousands) such devices.

The running example used in this book was MPLS. This was done to provide an interesting backdrop for the NMS discussions and also for a number of other reasons:

- MPLS allows for a connection-oriented layer 3 network.
- Phased migration of layer 2 technologies (such as ATM) to layer 3 becomes feasible.
- Layer 2 skills can be de-emphasized.
- Emerging standards such as PWE3 and Ethernet over MPLS pave the way for generic cores, moving complexity to the network edge.
- Consolidation of multiple technologies may also help in reducing the number of incompatible management systems.

Even if MPLS is not deployed in a large enterprise network, the benefits of NMS are considerable:

- An overall network perspective is provided by the NMS.
- NMS provides centralized management rather than using numerous EMS and proprietary products.
- It becomes possible to proactively manage the network using policies, that is, damage prevention rather than damage control.

The need for solutions in network management technology is a challenge for software developers. It becomes necessary to acquire a working knowledge of many different technologies, including IP, MPLS, ATM, Frame Relay, and SONET/SDH. The linked overview technique described in [Chapter 3](#) may be of assistance in rapidly getting a handle on these different areas. While there is rarely a substitute for experience, a willingness on the part of software developers to learn new technologies quickly can help vendor companies in shoring up skills shortages. This should have a direct and positive impact on product revenues.

On the commercial side, the global economic downturn (that started around March 2000) has forced most enterprise/SP network operators to assess their options. Investment does not tend to occur unless there is a proven return to be made.

As vendor organizations return to the path of profitability, it will become essential for them to produce good-quality differentiated solutions. This will translate into products that generate the cash needed to provide training for crossfunctional cooperation. It is only the

latter that will permit the long-term solution development and maintenance capability needed for the emerging NEs. Customers may be able to assist in this process by providing some of their own engineering capability, thereby extending crossfunctional cooperation outside the vendor organization. The creation of value is the difference between the cost of producing and selling solutions.

In [Chapter 7](#), we saw how straightforward it is to rapidly create NMS building blocks. The available development tools (e.g., Visual C++ and JDMK) are very easy to use and are supplied with reasonably good sample code. These provide base-level components that can be incorporated into larger bodies of NMS function—that is, FCAPS. The really hard problems are, as usual, related to scale, though usability and generic software are also crucial.

While vendors and enterprises have their problems, we also should not forget SP operator problems. The TeleManagement Forum [[TeleMgmtForum](#)] has reported that adding a new NE to an SP network can cost in excess of \$20 million. This is most likely due to some combination of:

- NMS changes required for the new hardware and associated NMS modules
- Interoperability problems with existing devices
- Firmware bugs in the new devices
- Integrating management for the NEs into existing OSS workflows and business practices

Similar costs apply to large enterprise networks. Many technologies, such as MPLS, are implemented long before the standards are complete. This is a necessary part of vendors keeping up with their competitors. Competitive advantage may go to those vendors whose products match the adopted standards. SNMP is an established standard that is widely deployed. Developers of NMS and NEs can use standard tools such as UML and SDL in conjunction with standard programming languages to create increasingly open systems. By open, we mean that UML and SDL allow for the development process to be opened up to all stakeholders. This can result in a better mapping between user requirements and deployed solutions.

Security is critical to successful network management, and SNMPv3 facilitates this. The SNMPv3 security model allows for extensions if necessary. It is likely that 3DES, AES (Advanced Encryption Standard), and their eventual successors will be required.

Network management is a very broad, exciting field. The trend toward favoring solutions over technology puts network management in a prominent position. The industry equation is simple: Good-quality NMS technology will help network operators to provide reliable, high-performance networks that meet organizational needs.

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

Appendix A. Terminal Server Serial Ports

The following is an excerpt from RFC 1317 and illustrates the way in which serial interfaces are modeled in this MIB. The object instances supported by a real device that implements this MIB can be viewed using a MIB browser and pointing it at an appropriately configured terminal server.

RFC1317-MIB DEFINITIONS ::= BEGIN

IMPORTS

Counter
 FROM RFC1155-SMI
transmission
 FROM RFC1213-MIB
OBJECT-TYPE
 FROM RFC-1212;

-- this is the MIB module for RS-232-like hardware devices

rs232 OBJECT IDENTIFIER ::= { transmission 33 }

-- the generic RS-232-like group
-- Implementation of this group is mandatory for all
-- systems that have RS-232-like hardware ports
-- supporting higher level services such as character
-- streams or network interfaces

rs232Number OBJECT-TYPE

SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION

"The number of ports (regardless of their current state) in the RS-232-like general port table."

::= { rs232 1 }

rs232AsyncPortTable OBJECT-TYPE

SYNTAX SEQUENCE OF Rs232AsyncPortEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION

"A list of asynchronous port entries. The maximum entry number is given by the value of rs232Number. Entries need not exist for synchronous ports."

::= { rs232 3 }

rs232AsyncPortEntry OBJECT-TYPE

SYNTAX Rs232AsyncPortEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION

"Status and parameter values for an asynchronous port."

INDEX { rs232AsyncPortIndex }
 ::= { rs232AsyncPortTable 1 }

```
Rs232AsyncPortEntry ::=  
SEQUENCE {  
rs232AsyncPortIndex INTEGER,  
rs232AsyncPortBits INTEGER,  
rs232AsyncPortStopBits INTEGER,  
rs232AsyncPortParity INTEGER,  
rs232AsyncPortAutobaud INTEGER,  
rs232AsyncPortParityErrs Counter,  
rs232AsyncPortFramingErrs Counter,  
rs232AsyncPortOverrunErrs Counter }
```

This MIB provides all of the above objects for use when managing serial devices. Each entry in the table corresponds to a serial interface, and the other columns provide access to configuration data such as the number of bits in a data word (`rs232AsyncPortBits`), the number of stop bits (`rs232AsyncPortStopBits`), and the port parity(`rs232AsyncPortParity`). A network management system reads the values of all entries in the `rs232AsyncPortTable` and sets the values of the port parameters as required. An example would be an appropriately configured 10-port terminal server, which would have 10 entries in this table. By sending a block of SNMP `GetRequest` messages, the entire table can be retrieved. If required, the user can also send a block of SNMP `SetRequest` messages in order to modify the rows in `rs232AsyncPortTable`. The following is the complete definition of the columns of this table:

[\[View full width\]](#)

```
rs232AsyncPortIndex OBJECT-TYPE  
SYNTAX INTEGER  
ACCESS read-only  
STATUS mandatory  
DESCRIPTION  
"A unique value for each port. Its value is the same as rs232PortIndex for the port."  
 ::= { rs232AsyncPortEntry 1 }
```

```
rs232AsyncPortBits OBJECT-TYPE  
SYNTAX INTEGER (5..8)  
ACCESS read-write  
STATUS mandatory  
DESCRIPTION  
"The port's number of bits in a character."  
 ::= { rs232AsyncPortEntry 2 }
```

```
rs232AsyncPortStopBits OBJECT-TYPE  
SYNTAX INTEGER { one(1), two(2), one-and-half(3),  
dynamic(4) }  
ACCESS read-write  
STATUS mandatory  
DESCRIPTION  
"The port's number of stop bits."  
 ::= { rs232AsyncPortEntry 3 }
```

```
rs232AsyncPortParity OBJECT-TYPE  
SYNTAX INTEGER { none(1), odd(2), even(3),  
mark(4), space(5) }  
ACCESS read-write  
STATUS mandatory  
DESCRIPTION  
"The port's sense of a character parity bit."  
 ::= { rs232AsyncPortEntry 4 }
```

```
rs232AsyncPortAutobaud OBJECT-TYPE  
SYNTAX INTEGER { enabled(1), disabled(2) }  
ACCESS read-write  
STATUS mandatory
```

DESCRIPTION

"A control for the port's ability to automatically sense input speed. When
 ↳ rs232PortAutoBaud is 'enabled', a port may autobaud to values different from the set
 ↳ values for speed, parity, and character size. As a result a network management system
 ↳ may temporarily observe values different from what was previously set."
::= { rs232AsyncPortEntry 5 }

rs232AsyncPortParityErrs OBJECT-TYPE

SYNTAX Counter

ACCESS read-only

STATUS mandatory

DESCRIPTION

"Total number of characters with a parity error, input from the port since system
 ↳ re-initialization and while the port state was 'up' or 'test'."
::= { rs232AsyncPortEntry 6 }

rs232AsyncPortFramingErrs OBJECT-TYPE

SYNTAX Counter

ACCESS read-only

STATUS mandatory

DESCRIPTION

"Total number of characters with a framing error, input from the port since system
 ↳ re-initialization and while the port state was 'up' or 'test'."
::= { rs232AsyncPortEntry 7 }

rs232AsyncPortOverrunErrs OBJECT-TYPE

SYNTAX Counter

ACCESS read-only

STATUS mandatory

DESCRIPTION

"Total number of characters with an overrun error, input from the port since system
 ↳ re-initialization and while the port state was 'up' or 'test'."
::= { rs232AsyncPortEntry 8 }

[\[Team Lib \]](#)

[PREVIOUS](#) [NEXT](#) ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

Appendix B. Some Simple IP Routing Experiments

This appendix provides some examples of using the Windows NT/2000 console IP utilities.

[Team LiB]

◀ PREVIOUS NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

B.1 The IP Routing Table

Netstat is a very useful tool for looking at various aspects of the IP configuration. Here, we illustrate the IP routing table on a Windows 2000 PC.

C:\netstat -r

Route Table

Interface List

0x1 MS TCP Loopback interface
0x1000003 ...00 b0 d0 16 48 97 ... 3Com EtherLink PCI

Active Routes:

Network	Dest	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	255.255.255.0	142.159.65.17	142.159.65.29	1
127.0.0.0	255.0.0.0	255.255.255.0	127.0.0.1	127.0.0.1	1
142.159.65.0	255.255.255.0	142.159.65.29	142.159.65.29	142.159.65.29	1

Default Gateway: 142.159.65.17

Persistent Routes:

None

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

B.2 Ping

Ping is an indispensable tool for determining if specified IP interfaces are up.

C:\>ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<10ms TTL=128

Ping statistics for 127.0.0.1:

 Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

 Approximate round trip times in milliseconds:

 Minimum = 0ms, Maximum = 0ms, Average = 0ms

In the above session, four ICMP request packets are sent to the loopback interface, and four ICMP response packets are received.

[Team LiB]

◀ PREVIOUS NEXT ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

B.3 Traceroute

Traceroute is very useful for determining the route taken for a specified IP destination. Try the following and substitute something like www.microsoft.com (or an IP address internal or external to your intranet) to see the IP route traced to the destination.

C:\>tracert localServer

Tracing route to localServer [142.159.65.17]

over a maximum of 30 hops:

```
1 <10 ms <10 ms <10 ms hostServer [140.140.211.1]
2 <10 ms <10 ms <10 ms localServer [142.159.65.17]
```

Trace complete.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Appendix C. The IP MIB Table

This appendix provides a listing of the IP MIB table with real data retrieved from a Cisco 7200 router.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

MIB Walk on the IP Table

The following listing illustrates an excerpt from walking the IP routing table. It uses the example Visual C++ application described in [Chapter 6](#), "Network Management Software Components."

[\[View full width\]](#)

```
D:\Visual C++ Projects\ debug\snmputilv5.exe WALK 10.81.1.133 public .iso.org.dod.  
internet.mgmt.mib-2.ip  
Variable = ip.ipForwarding.0  
Value   = Integer32 1  
Variable = ip.ipDefaultTTL.0  
Value   = Integer32 255  
Variable = ip.ipInReceives.0  
Value   = Counter32 5608115  
Variable = ip.ipInHdrErrors.0  
Value   = Counter32 3  
Variable = ip.ipInAddrErrors.0  
Value   = Counter32 0  
Variable = ip.ipForwDatagrams.0  
Value   = Counter32 4  
Variable = ip.ipInUnknownProtos.0  
Value   = Counter32 10  
Variable = ip.ipInDiscards.0  
Value   = Counter32 0  
Variable = ip.ipInDelivers.0  
Value   = Counter32 5602411  
Variable = ip.ipOutRequests.0  
Value   = Counter32 3717989  
Variable = ip.ipOutDiscards.0  
Value   = Counter32 2489  
Variable = ip.ipOutNoRoutes.0  
Value   = Counter32 4  
Variable = ip.ipReasmTimeout.0  
Value   = Integer32 30  
Variable = ip.ipReasmReqds.0  
Value   = Counter32 0  
Variable = ip.ipReasmOKs.0  
Value   = Counter32 0  
Variable = ip.ipReasmFails.0  
Value   = Counter32 0  
Variable = ip.ipFragOKs.0  
Value   = Counter32 0  
Variable = ip.ipFragFails.0  
Value   = Counter32 0  
Variable = ip.ipFragCreates.0  
Value   = Counter32 0
```

[Team LiB]

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Appendix D. Visual C++ Sample Program Source Code

This appendix provides a listing of the Visual C++ sample program source code.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

D.1 snmpdefs.h

This is the header file that provides symbolic constants and function declarations for the main C file.

```
***** SECTION 1 *****
#define TIMEOUT 6000 /* milliseconds */
#define RETRIES 3
#define MAX_OID_NAME_LENGTH 50
#define FUNCTION_SUCCESS 0
#define FUNCTION_FAILED -1

enum Operations { GET, GETNEXT, SET, WALK, TRAP };
char * operationsArray[5] = { "GET", "GETNEXT", "SET", "WALK", "TRAP" };

***** SECTION 2 *****
INT programMode, operation;
LPSTR SNMPAgentName, SNMPCommunity, OIDstring;
RFC1157VarBindList variableBindings;
LPSNMP_MGR_SESSION SNMPsession;
AsnAny retrievedInstanceValue;
INT timeout = TIMEOUT;
INT retries = RETRIES;
BYTE requestType;
AsnInteger errorStatus, errorIndex;
***** SECTION 3 *****
int allocateResources(LPSTR agentName, LPSTR community,
                     char *objectIdentifier);
int deallocateResources();
int prepareDataForOperation(enum Operations reqOperation,
                           unsigned char *newObjectValue);
int prepareSetOperation(unsigned char *newObjectValue);
int prepareGetOperation();
int prepareGetNextOperation();
int dispatchOperation(int programMode, char * argumentVector[]);
int createSNMPSession();
int executeRequest();
int displayMIBInstanceValue();
int doSnmpOperation(enum Operations reqOperation, LPSTR agentName,
                   LPSTR community, char *objectIdentifier, char *objectValue);
int executeMibWalk();
int waitForTraps();
int startupRoutine(int argc, char *argv []);
```

[Team LiB]

◀ PREVIOUS NEXT ▶

[Team LiB]

◀ PREVIOUS NEXT ▶

D.2 snmpmgr.c

This is the main C file that provides function definitions and the implementation of the program.

[\[View full width\]](#)

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <snmp.h>
#include <mgtapi.h>

#include "snmpdefs.h"

INT _CRTAPI1 main(
    IN int argumentCount,
    IN char *argumentVector[])
{
    startupRoutine (argumentCount, argumentVector);

    dispatchOperation(programMode, argumentVector);

    return 0;
}

/****************************************
 * This routine checks the command line parameters and issues a
usage statement if there are insufficient entries.
****************************************/
int startupRoutine(int argc, char *argv [])
{
    int loop;

    programMode = 0;

    switch (argc)
    { //GET, GETNEXT, SET, WALK, TRAP
    case 2:
    {
        programMode = TRAP;
        // printf("Program Mode is TRAP %d\n", programMode);
        break;
    }
    case 5:
    {
        programMode = GET;
        // printf("Program Mode is GET %d\n", programMode);
        break;
    }
    case 6:
    {
        programMode = SET;
```

```
// printf("Program Mode is SET %d\n", programMode);
break;
}
default:
{
printf ("usage: %s <Mode> <Agent> <Community> <Object ID>
[<Value>]\n", argv [0]);
exit (0x01);
}
}

for (loop = 0; loop < TRAP; loop++)
{
    if (!strcmp(argv[1], operationsArray[loop]))
    {
        programMode = loop;
        break;
    }
}

// printf("Final Program Mode is %s value %d\n", operationsArray[loop],
// programMode);

return programMode;
}

/************************************************************
 * Issue the required operation
************************************************************/
int dispatchOperation(int programMode, char * argumentVector[])
{
    switch (programMode)
    {
    case GET:
    {
        doSnmpOperation(programMode, argumentVector[2],
            argumentVector[3], argumentVector[4], NULL);
        break;
    }
    case GETNEXT:
    {
        doSnmpOperation(programMode, argumentVector[2],
            argumentVector[3], argumentVector[4], NULL);
        break;
    }
    case WALK:
    {
        doSnmpOperation(programMode, argumentVector[2],
            argumentVector[3], argumentVector[4], NULL);
        break;
    }
    case SET:
    {
        doSnmpOperation(programMode, argumentVector[2],
            argumentVector[3], argumentVector[4], argumentVector[5]);
        break;
    }
    case TRAP:
    {
        doSnmpOperation(programMode, argumentVector[1],
            NULL, NULL, NULL);
        break;
    }
}
```

```
    }
    default:
    {
        printf("Unknown dispatchOperation mode\n");
        break;
    }
}
return 0;
}

/*************************************
* Prepare for the user-requested operation including traps
************************************/

int prepareForOp(enum Operations reqOperation, LPSTR agentName,
                LPSTR community, char *objectIdentifier, char *objectValue)
{
    if (reqOperation != TRAP)
    {
        allocateResources(agentName, community, objectIdentifier);
        prepareDataForOperation(reqOperation, objectValue);
    }

    return 0;
}

/*************************************
* The following function is for a complete SNMP Agent operation.
* The syntax for its use is as follows:
* (i) reqOperation may have the value GET/SET/TRAP
* (ii) agentName is the IP address or DNS name of the agent
* (iii) community is the name of the SNMP community
* (iv) objectIdentifier is the OID of interest
* (v) objectValue is the value of the OID for set operations
************************************/

int doSnmpOperation(enum Operations reqOperation, LPSTR agentName,
                    LPSTR community, char *objectIdentifier, char *objectValue)
{
    prepareForOp(reqOperation, agentName, community,
                 objectIdentifier, objectValue);

    switch (reqOperation)
    {
        case GETNEXT:
        case GET:
        case SET:
            {
                executeRequest();
                displayMIBinstanceValue(reqOperation);
                break;
            }
        case WALK:
            {
                executeMibWalk();
                break;
            }
        case TRAP:
            {
                waitForTraps();
                break;
            }
    default:
        {
```

```
        break;
    }
}

return 0;
}

//*********************************************************************
* The following function copies the parameters into
* the corresponding member variables. It does the following:
* (i) Copies the SNMP Agent name into a private data member,
* (ii) Copies the SNMP community name into a private data member,
* (iii) Converts the object name into the correct OID format,
* (iv) Sets up the variable bindings structure
*****/




int allocateResources(LPSTR agentName,
                      LPSTR community, char *objectIdentifier)
{
    AsnObjectIdentifier reqObject;

    // Allocate space for the SNMP agent name or IP address
    SNMPAgentName = (LPSTR)SNMP_malloc(strlen(agentName) + 1);
    strcpy(SNMPAgentName, agentName);

    // Allocate space for the agent community
    SNMPCommunity = (LPSTR)SNMP_malloc(strlen(community) + 1);
    strcpy(SNMPCommunity, community);

    // Open an SNMP session
    createSNMPSession();

    // Get specified object identifiers
    variableBindings.list = NULL;
    variableBindings.len = 0;

    // Convert the OID string representation to the internal representation.
    if (!SnmpMgrStrToOid(objectIdentifier, &reqObject))
    {
        printf("Error: Invalid oid, %s, specified.\n",
objectIdentifier);
        return FUNCTION_FAILED;
    }
    else
    {
        // Set up the variable bindings for the SNMP operation
        OIDstring = (LPSTR)SNMP_malloc(MAX_OID_NAME_LENGTH);
        if (!SnmpMgrOidToStr(&reqObject, &OIDstring))
        {
            printf("Error: Invalid oid, %s, returned\n", OIDstring);
            return FUNCTION_FAILED;
        }

        SNMP_free(OIDstring);

        // It is here that the size of the variable bindings needs to be
        // decided and acted upon. The number of objects in this operation
        // is determined by the value of variableBindings.len

        // Add the required OID to the variable bindings list.
        variableBindings.len++;
        if ((variableBindings.list = (RFC1157VarBind *)SNMP_realloc(
            variableBindings.list, sizeof(RFC1157VarBind) *
```

```
variableBindings.len)) == NULL)
{
    printf("Error: Error allocating oid, %s.\n", objectIdentifier);
    return FUNCTION_FAILED;
}

variableBindings.list[variableBindings.len - 1].name =
    reqObject; // NOTE! structure copy
variableBindings.list[variableBindings.len - 1].value.asnType =
    ASN_NULL;
}

return FUNCTION_SUCCESS;
}

/************************************************************
* The following function frees these parameters:
* (i) The SNMP Agent name,
* (ii) The SNMP community name,
* (iii) The object name,
* (iv) The variable bindings structure
* (v) The SNMP session
************************************************************/
int deallocateResources()
{
    // Free any dynamically allocated memory objects
    if(SNMPAgentName)
    {
        SNMP_free(SNMPAgentName);
        SNMPAgentName = NULL;
    }

    if(SNMPCommunity)
    {
        SNMP_free(SNMPCommunity);
        SNMPCommunity = NULL;
    }

    // Free the variable bindings that have been allocated.
    SnmpUtilVarBindListFree(&variableBindings);

    // Close the SNMP session with the remote agent
    if (!SnmpMgrClose(SNMPsession))
    {
        printf("error on SnmpMgrClose %d\n", GetLastError());
        return FUNCTION_FAILED;
    }

    return FUNCTION_SUCCESS;
}

/************************************************************
* Prepare for the required request
************************************************************/
int prepareDataForOperation(enum Operations reqOperation,
                           unsigned char *newObjectValue)
{
    int returnValue = FUNCTION_SUCCESS;

    switch (reqOperation)
    {
        case GET: {
```

```
prepareGetOperation();
break;
}
case GETNEXT: {
    prepareGetNextOperation();
    break;
}
case SET: {
    prepareSetOperation(newObjectValue);
    break;
}
case WALK: {
    break;
}
case TRAP: {
    break;
}
default: {
    returnValue = FUNCTION_FAILED;
    break;
}
}

return returnValue;
}

/*************************************
 * Prepare for a Set request
************************************/

int prepareSetOperation(unsigned char *newObjectValue)
{
requestType = ASN_RFC1157_SETREQUEST;

// Request that the API carry out the desired operation.
variableBindings.len = 1;

/* This is not sufficiently generalized, the type needs to be
   passed in as a parameter */
variableBindings.list[variableBindings.len - 1].value.asnType =
    ASN_OCTETSTRING;
variableBindings.list[variableBindings.len - 1].value.asnValue.string.stream =
    (unsigned char *)newObjectValue;
variableBindings.list[variableBindings.len - 1].value.asnValue.string.length =
    strlen((const char *)
variableBindings.list[variableBindings.len - 1].value.asnValue.string.
    stream);

return FUNCTION_SUCCESS;
}

/*************************************
 * Prepare for a Get request
************************************/

int prepareGetOperation()
{
requestType = ASN_RFC1157_GETREQUEST;

return FUNCTION_SUCCESS;
}

/*************************************
 * Prepare for a GetNext request
************************************/
```

```
******/  
int prepareGetNextOperation()  
{  
    requestType = ASN_RFC1157_GETNEXTREQUEST;  
  
    return FUNCTION_SUCCESS;  
}  
  
*****/* Create a session with a remote agent*****/  
int createSNMPSession()  
{  
    if ((SNMPsession = SnmpMgrOpen(SNMPAgentName, SNMPCommunity, timeout, retries)) ==  
        NULL)  
    {  
        printf("error on SnmpMgrOpen %d\n", GetLastError());  
        return FUNCTION_FAILED;  
    }  
    return FUNCTION_SUCCESS;  
}  
  
*****/* Execute a MIB walk*****/  
int executeMibWalk()  
{  
    // Walk is a common term used to indicate that all MIB variables  
    // under a given OID are to be traversed and displayed. This is  
    // a more complex operation requiring tests and looping in addition  
    // to the steps for get/getnext above.  
    AsnObjectIdentifier root;  
    AsnObjectIdentifier tempOid;  
    SnmpUtilOidCpy(&root, &variableBindings.list[0].name);  
    requestType = ASN_RFC1157_GETNEXTREQUEST;  
  
    while(1)  
    {  
        if (!SnmpMgrRequest(SNMPsession, requestType, &variableBindings,  
                            &errorStatus, &errorIndex))  
        {  
            // The API is indicating an error.  
            printf("error on SnmpMgrRequest %d\n", GetLastError());  
            break;  
        }  
        else  
        {  
            // The API succeeded, errors may be indicated from the remote  
            // agent. Test for end of subtree or end of MIB.  
            if (errorStatus == SNMP_ERRORSTATUS_NOSUCHNAME ||  
                SnmpUtilOidNCmp(&variableBindings.list[0].name,  
                                &root, root.idLength))  
            {  
                printf("End of MIB subtree.\n\n");  
                break;  
            }  
  
            // Test for general error conditions or sucesss.  
            if (errorStatus > 0)  
            {  
                printf("Error: errorStatus=%d, errorIndex=%d \n",  
                      errorStatus, errorIndex);  
            }  
        }  
    }  
}
```

```
        break;
    }
else
{
    // Display resulting variable binding for this iteration.
    char *string = NULL;
    SnmpMgrOidToStr(&variableBindings.list[0].name, &string);
    printf("Variable = %s\n", string);
    if (string)
        SNMP_free(string);

    printf("Value   = ");
    SnmpUtilPrintAsnAny(&variableBindings.list[0].value);
}
} // end if()

// Prepare for the next iteration. Make sure the returned oid is
// preserved and the returned value is freed.
SnmpUtilOidCpy(&tempOid, &variableBindings.list[0].name);

SnmpUtilVarBindFree(&variableBindings.list[0]);

SnmpUtilOidCpy(&variableBindings.list[0].name, &tempOid);
variableBindings.list[0].value.asnType = ASN_NULL;

SnmpUtilOidFree(&tempOid);

} // end while()

// Free the variable bindings that have been allocated.
SnmpUtilVarBindListFree(&variableBindings);
SnmpUtilOidFree(&root);

return 0;
}

/*****************/
/* Execute the required SNMP operation
*****************/
int executeRequest()
{
// Request that the API carry out the desired operation.
if (!SnmpMgrRequest(SNMPsession, requestType, &variableBindings,
    &errorStatus, &errorIndex))
{
    // The API is indicating an error.
    printf("error on SnmpMgrRequest %d\n", GetLastError());
    return FUNCTION_FAILED;
}

return FUNCTION_SUCCESS;
}

/*****************/
/* Display the retrieved OID instance value and type
*****************/
int displayMIBInstanceValue(enum Operations reqOperation)
{
// Display the resulting variable bindings.
UINT i;
char *string = NULL;
```

```
printf ("SNMP Operation Type %s\n", operationsArray[reqOperation]);
for (i = 0; i < variableBindings.len; i++)
{
    SnmpMgrOidToStr(&variableBindings.list[i].name, &string);
    printf("MIB Object Instance = %s\n", string);
    if (string)
        SNMP_free(string);
    printf("Type and Value = ");
    SnmpUtilPrintAsnAny(&variableBindings.list[i].value);
    // Save the value of the retrieved instance
    //retrievedInstanceValue = variableBindings.list[i].value;
}

return FUNCTION_SUCCESS;
}

/*****************
 * Initiate the process of listening for traps
*****************/
int waitForTraps()
{
    // Trap handling can be done two different ways: event driven or
    // polled. The following code illustrates the steps to use event
    // driven trap reception in a management application.
    HANDLE hNewTraps = NULL;

    if (!SnmpMgrTrapListen(&hNewTraps))
    {
        printf("error on SnmpMgrTrapListen %d\n", GetLastError());
    }
    else
    {
        printf("snmputil: listening for traps...\n");
    }

    while(1)
    {
        DWORD dwResult;

        if ((dwResult = WaitForSingleObject(hNewTraps, 0xffffffff))
            == 0xffffffff)
        {
            printf("error on WaitForSingleObject %d\n",
                  GetLastError());
        }
        else if (!ResetEvent(hNewTraps))
        {
            printf("error on ResetEvent %d\n", GetLastError());
        }
        else
        {
            AsnObjectIdentifier enterprise;
            AsnNetworkAddress IPAddress;
            AsnInteger genericTrap;
            AsnInteger specificTrap;
            AsnTimeticks timeStamp;
            RFC1157VarBindList variableBindings;
            UINT i;
            char *string = NULL;

            while(SnmpMgrGetTrap(&enterprise, &IPAddress, &genericTrap,
```

```
&specificTrap, &timeStamp, &variableBindings))  
{  
    printf("snmputil: trap generic=%d specific=%d\n",  
          genericTrap, specificTrap);  
    if (IPAddress.length == 4)  
    {  
        printf(" from -> %d.%d.%d.%d\n",  
              (int)IPAddress.stream[0], (int)IPAddress.stream[1],  
              (int)IPAddress.stream[2], (int)IPAddress.stream[3]);  
    }  
    if (IPAddress.dynamic)  
    {  
        SNMP_free(IPAddress.stream);  
    }  
  
    for (i = 0; i < variableBindings.len; i++)  
    {  
        SnmpMgrOidToStr(&variableBindings.list[i].name, &string);  
        printf("Variable = %s\n", string);  
        if (string)  
            SNMP_free(string);  
  
        printf("Value = ");  
        SnmpUtilPrintAsnAny(&variableBindings.list[i].value);  
    } // end for()  
    printf("\n");  
  
    SnmpUtilOidFree(&enterprise);  
    SnmpUtilVarBindListFree(&variableBindings);  
}  
}  
}  
} // end while()  
}
```

[[Team LiB](#)]

[PREVIOUS](#) [NEXT](#) ▶

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Appendix E. JDMK Sample Program Source Code

This appendix provides a listing of the JDMK sample program source code.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

E.1 synchronousManager.Java

The following is a listing of the Java program:

[\[View full width\]](#)

```
import javax.management.snmp.SnmpDefinitions;
import javax.management.snmp.SnmpOid;
import javax.management.snmp.SnmpVarBindList;
import javax.management.snmp.manager.SnmpPeer;
import javax.management.snmp.manager.SnmpParameters;
import javax.management.snmp.manager.SnmpRequest;
import javax.management.snmp.manager.SnmpSession;
//import javax.management.snmp.manager.SnmpEventReportDispatcher;
import com.sun.jdmk.Trace;
import com.sun.jdmk.snmp.SnmpOidTableSupport;

/**
 * To use the SNMP agent provided as part of JDMK, use port 8085. */

public class SynchronousManager {
    /**
     * The main entry point. When calling the program, the      command
     * line must contain:
     * 1) operation: GET, GETNEXT.
     * 2) target: IP address or DNS name of the remote agent
     * 3) community: string for the operation
     * 4) OID: the OID of interest
     * 5) value: Object instance value (for set operations)
     * 6) port: port number to use
     */
    public static void main(String argv[]) {

        final int numParameters = 6;

        if (argv.length != numParameters) {
            usage();
            java.lang.System.exit(1);
        }

        String snmpOp = argv[0];
        String host = argv[1];
        String community = argv[2];
        String requiredOid = argv[3];
        String oidValue = argv[4];
        String port = argv[5];

        try {
            /* Initialize the JDMK SNMP Manager API.
             Specify the OidTable containing the SNMP MIB II data.
             Use the OidTable generated by mibgen after compiling MIB II. */
            SnmpOidTableSupport oidTable = new RFC1213_MIBOidTable();
```

```
SnmpOid.setSnmpOidTable(oidTable);

// Create a SnmpPeer object for representing the remote entity.
SnmpPeer agent = new SnmpPeer(host, Integer.parseInt(port));

// Create required parameters for the remote entity.
// We expect the remote agent to support a public and private community
SnmpParameters params = new SnmpParameters(community, "private");

// Associate the parameters with the remote agent.
agent.setSnmpParam(params);

// Create the session for controlling the request.
SnmpSession session= new SnmpSession("SyncManager session");

// A default peer (agent) is now associated with an SnmpSession.
session.setDefaultPeer(agent);

// Build the varbind list.
SnmpVarBindList list = new SnmpVarBindList("SyncManager varbind list");

list.addVarBind(requiredOid);

// Issue the required SNMP request and wait for the result.
issueRequest(agent, snmpOp, list, session, host, port);

// Destroy the session
session.destroySession();

java.lang.System.exit(0);

} catch(Exception e) {
    java.lang.System.err.println("Exception occurred:" + e );
    e.printStackTrace();
}

}

/** 
 * Return command line usage of the program.
 */
public static void usage()
{
    java.lang.System.out.println("java SynchronousManager <op> <host> <community>
    <OID> <value> <port>");
    java.lang.System.out.println("where");
    java.lang.System.out.println("\t-op: required SNMP operation.");
    java.lang.System.out.println("\t-host: hostname or IP address of the SNMP agent.
    ");
    java.lang.System.out.println("\t-community: string for the operation.");
    java.lang.System.out.println("\t-OID: required object instance identifier.");
    java.lang.System.out.println("\t-value: of the object instance (for set only).");
    java.lang.System.out.println("\t-port: port number for the remote agent.");
}

/** 
 * Issue the required SNMP message.
 */
public static void issueRequest(SnmpPeer agent, String operation,
        SnmpVarBindList list, SnmpSession session,
        String host, String port)
{
    final String GEToperation = "GET";
```

```
final String GETNEXToperation = "GETNEXT";
final String SEToperation = "SET";
SnmpRequest request = null;

try {
    // Make the SNMP request and wait for the result.
    if (operation.compareToIgnoreCase(GEToperation) == 0)
    {
        request = session.snmpGetRequest(null, list);
    }
    else
    if (operation.compareToIgnoreCase(GETNEXToperation) == 0)
    {
        request = session.snmpGetNextRequest(null, list);
    }
    else
    {
        java.lang.System.out.println("Unknown operation. Exiting...");
        java.lang.System.exit(0);
    }

    java.lang.System.out.println("Sent " + operation + " request to agent on " +
    host + " port " + port);
    boolean completed = request.waitForCompletion(10000);

    // Check for a timeout of the request.
    if (completed == false) {
        java.lang.System.out.println("Request timed out. Check reachability of
    agent");

        // Print request.
        java.lang.System.out.println("Request: " + request.toString());
        java.lang.System.exit(0);
    }

    // Check if the response contains an error.
    int errorStatus = request.getErrorStatus();
    if (errorStatus != SnmpDefinitions.snmpRspNoError) {
        java.lang.System.out.println("Error status = " + SnmpRequest.
    snmpErrorToString(errorStatus));
        java.lang.System.out.println("Error index = " + request.getErrorIndex());
        java.lang.System.exit(0);
    }

    // Display the result.
    SnmpVarBindList result = request.getResponseVarBindList();
    java.lang.System.out.println("Result: \n" + result);

} catch(Exception e) {
    java.lang.System.err.println("Exception occurred:" + e );
    e.printStackTrace();
}
}
```

[Team LiB]

[PREVIOUS](#) [NEXT](#)

[Team LiB]

◀ PREVIOUS NEXT ▶

References

[Alcatel2001] R. Uebel & M. Verhoeven, "Strategy for Migrating Voice Networks to the Next-Generation Architecture," *Alcatel Telecommunications Review*, 3rd Quarter, 2001.

[ANSIWeb] www.ansi.org

[AT&T2002] *The Economist*, Feb. 25, 2002.

[ATM&IP2001] "The ATM & IP Report," *Broadband Publishing*, 8(3), July/August 2001.

[ATMObjects] www.ietf.org/rfc/rfc2515.txt

[CrollPackmanBW] Alistair Croll & Eric Packman, *Managing Bandwidth, Deploying QoS in Enterprise Networks*, Prentice Hall, 1999.

[CERTWeb] www.cert.org

[CiscoVoIP] www.cisco.com/en/US/tech/tk652/tk701/tech_protocol_family_home.html IP Telephony / VoIP.

[CiscoWWW] www.cisco.com

[ComerStevens] Douglas Comer, & David Stevens, *Internetworking with TCP/IP*, Vol. 2, Prentice Hall, 1993.

[DavieRehker2000] Bruce Davie & Yakov Rekhter, *MPLS Technology and Applications*, Morgan Kaufmann Publishers, 2000.

[DMTF1999] www.dmtf.org/standards/index.php

[EnterpriseIT] www.harriskern.com. Design and implementation of IT organizations.

[EssentialSNMP] Douglas R. Mauro & Kevin J. Schmidt, *Essential SNMP*, O'Reilly & Associates, 2001.

[GlobalCross2002] "Survival of the Slowest," *The Economist*, Jan. 31, 2002.

[HPNNMScale2002] Hewlett-Packard, "A Guide to Scalability and Distribution for HP OpenView Network Node Manager," Part number J1240-90081, May 2002.

[HPOpenView] Jill Huntington-Lee, Kornel Terplan, & Jeffrey Gibson, *HP OpenView: A Manager's Guide*, McGraw-Hill, 1996.

[IETF-MPLS-FTN] "Multiprotocol Label Switching (MPLS) FEC-to-NHLFE (FTN) Management Information Base," Current at the time of writing. Available: www.ietf.org/internet-drafts/draft-ietf-mpls-ftn-mib-05.txt

[IETF-MPLS-MGMT] "Multiprotocol Label Switching (MPLS) Management Overview," Current at the time of writing. Available: www.ietf.org/internet-drafts/draft-ietf-mpls-mgmt-overview-01.txt

[IETF-LSR-MPLS] "Multiprotocol Label Switching (MPLS) Label Switch Router (LSR) Management Information Base," Current at the time of writing. Available: www.ietf.org/internet-drafts/draft-ietf-mpls-lsr-mib-09.txt

[IETF-TE-MPLS] "Multiprotocol Label Switching (MPLS) Traffic Engineering Management Information Base," Current at the time of writing. Available: www.ietf.org/internet-drafts/draft-ietf-mpls-te-mib-09.txt

[IETF-PWE3] www.ietf.org/html.charters/pwe3-charter.html

[IETFWeb] www.ietf.org

[IPDR-ORG] www.ipdr.org An open consortium of service providers, equipment vendors, system integrators, and billing/mediation vendors. IPDR members aim to facilitate the exchange of both usage and control data between network and hosting elements and operations and business support systems.

[ISPArchitecture] John V. Nguyen, *Designing ISP Architectures*, Sun Microsystems, 2002.

[JavaDev] Clifford Berg, *Advanced Java Development for Enterprise Applications*, Prentice Hall, 1998.

[Juniper2001] "Profile: Scott Kriens, CEO Juniper," *The Euronet*, December 2001.

[LightReading] www.lightreading.com

[MIBPerkins] David Perkins & Evan McGinnis, *Understanding SNMP MIBs*, 1st ed., Prentice Hall, 1997.

[MicrosoftWeb] www.microsoft.com

[MPLS&Profits] "MPLS Is Key to Profits," March 21, 2002. Available: www.lightreading.com

[MPLSForum] www.mplsforum.org

[MultiserviceSwitch] "MultiService WAN Switch Market: 2001 and Beyond," March, 2002. Available: www.instat.com/index.htm

[NadeauMPLS2002] Thomas Nadeau, *MPLS Network Management*, Morgan Kauffman Publishers, 2002.

[NMSGotham] www.gothamnetworks.com

[NMSHPOV] www.openview.hp.com

[NMSOpen] www.opennms.com

[NovellSAN] Novell NetWare 6—SAN and Clustering Technical White Paper. Available: www.novell.com

[OMGWeb] www.omg.org

[OpenSourceWeb] www.opensource.org/docs/definition_plain.html

[PrenHallCodeWeb] <http://authors.phptr.com/morris>

[PrinRussell] D. Russel, *The Principles of Computer Networking*, Cambridge University Press, 1989.

[PSOS] www.windriver.com/products/html/psosystem.html

[Puzmanova2001] Rita Puzmanova, *Routing and Switching: Time of Convergence*, Addison-Wesley, 2001.

[RationalRose] "Rational Rose 98i Using Rose," Rational Software Corporation, December 1998.

[RFC2475] RFC2475, "An Architecture for Differentiated Services." Available: www.ietf.org

[RFC2622] RFC2622, "Routing Policy Specification Language (RPSL)." Available: www.ietf.org

[RFC2702] RFC2702, "Traffic Engineering." Available: www.ietf.org

[RFC2748] RFC2748, "The COPS (Common Open Policy Service) Protocol." Available: www.ietf.org

[RFC2749] RFC2749, "COPS Usage for RSVP." Available: www.ietf.org

[RFC3031] RFC3031, "MPLS Architecture." Available: www.ietf.org

[RFC3084] RFC3084, "COPS Usage for Policy Provisioning (COPS-PR)." Available: www.ietf.org

[RFC3159] RFC3159, "Structure of Policy Provisioning Information (SPPI)." Available: www.ietf.org

[RFC3260] RFC3260, "New Terminology and Clarifications for Diffserv." Available: www.ietf.org

[RiverstoneLFAP] Description of the Riverstone Lightweight Flow Accounting Protocol. Available: www.riverstonenet.com/pdf/accounting_for_profitability.pdf

[RouterScale2002] "Business Networks—Does Scale Matter?" *Telecommunications Online*, Feb. 2002.

[RoutingHuitema] Christian Huitema, *Routing in the Internet*, Prentice Hall, 1999.

[Stallings1999] William Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Addison-Wesley, 1999.

[SunMicrosystems] java.sun.com

[Sweeney2001] Dan Sweeney, "The Hard Sell of Softswitches" *America's Network*, Dec. 2001. Available: www.americasnetwork.com

[Tanenbaum1996] Andrew S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice Hall, 1996.

[Tanenbaum2003] Andrew S. Tanenbaum, *Computer Networks*, 4th ed., Prentice Hall, 2003.

[Telcordia] www.telcordia.com Research Section

[TeleMgmtForum] www.tmforum.org

[Tele2001] Ted McKenna, "Piecing Together the OSS," *Telecommunications Online*, Sept. 2001.

[TimesMarch2002] Paul Durman, "Banks Prepare to Take Control of Marconi," *Sunday Times* (UK & Ireland), Business Section, March 24, 2002.

[UMLRumbaugh] J. Rumbaugh, I. Jacobson, & G. Booch, *The Unified Modeling Language Reference Manual* Addison-Wesley, 1999.

[VXWORKS] www.windriver.com/products/html/vxwks54.html

[Zeltserman1999] David Zeltserman, *A Practical Guide to SNMPv3 and Network Management* Prentice Hall, 1999.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[Team LiB]

 PREVIOUS

Glossary

API

An Application Programming Interface is an entry point to a system of software. The API provides programmatic access to a set of software services. SNMP is included as part of many versions of UNIX, and access to it is made using the system API. Likewise, a third-party software product such as the Microsoft Visual C++ SNMP API or the Sun Microsystems JDMK product can provide an API.

ASN.1

Abstract Syntax Notation One is a formal language for the abstract (platform-independent) description of messages exchanged between machines. It is used to encode and decode messages in a wide range of applications, including SNMP. Objects such as integers are encoded in a manner called tag-length-value (TLV) that is independent of any processor architecture, such as big or little endian. The tag indicates the object type, the length is the object size, and the value is the encoded object. ASN.1 also allows structured (or nested) definitions.

AS

An Autonomous System is an IP network controlled by one administrator on behalf of a single administrative entity (such as a university, a business enterprise, or a business division). Also referred to as a routing domain, an autonomous system is assigned a globally unique number called an Autonomous System Number.

BGP4

The Border Gateway Protocol Version 4 is an exterior gateway protocol used for routing between different autonomous systems. It is mainly used for providing reachability between administrative domains that use different routing and protection methods. Interior gateway protocols, such as OSPF and IS-IS, are used inside autonomous systems.

CDR

Call Detail Records are created by NEs when a call setup (e.g., a Q.931 ISDN SETUP message) message is received. CDRs reflect NE resource consumption such as calling and called parties, bandwidth used, and processing time. CDR data fields are fully populated when the underlying call/transaction ends. At this point, NEs with CDR data can often be configured to emit their records for capture and processing by an external billing system. NEs may buffer some CDRs after transmission, but if call volume is high, then data may be quickly overwritten. Many PABXs create CDRs for use in billing software.

CIM

Common Information Model is a model for describing overall management information in a network environment. It is a vendor-independent, conceptual information model for describing management data. CIM consists of a specification and a schema. The specification defines the details for integration with other management models, while the schema provides the actual model descriptions. It allows for the interchange of management information between management systems and applications.

CLI

A Command-Line Interface is a means of interaction provided as standard with many NEs. The Cisco CLI has been a de facto standard. The CLI usually consists of a tree-structured menu system used to configure NEs. Many devices also allow scripting facilities in order to speed up the installation and configuration process. Some network operators favor one CLI purely because they have a lot of experience with the associated equipment.

CO

Central Office of a service provider. It consists of switches, routers, PABXs, collocated servers, power, connections to customer premises equipment via the local loop, SS7 stack, toll trunks, and so on.

COM

Component Object Model is a Microsoft-specific software architecture that allows applications to be built from binary software components. COM is the underlying architecture that forms the foundation for higher level software services such as support for compound documents, custom controls, interapplication scripting, and data transfer.

COS

Class of Service is a method of differentiating the treatment received by specific traffic elements as they traverse a network. It tends to work on the basis of marking the traffic in some fashion and then delegating to the NEs to use that marking to apply a specific forwarding treatment. This can include mechanisms such as queuing (higher priority queue for traffic marked with a high priority), scheduling, the path taken by the traffic, and so on. CoS is quite distinct from QoS, which tends to reserve network resources ahead of the arrival of traffic.

COTS

Commercial Off-The-Shelf is an expression used to describe standard software packages available through retail channels. COTS software generally contains no proprietary components and includes Web browsers, word processors, and desktop productivity tools. COTS is of great interest to network operators and equipment vendors for slightly different reasons. Vendors may be happy to incorporate COTS software into their solutions in order to reduce development costs. Network operators like it because it helps in providing a degree of vendor-independence for at least part of a solution. However, no

solution is ideal, and in uncertain times COTS suppliers may not have the degree of longevity that is generally required by large network operators.

CPE

Customer Premises Equipment is the term used to describe SP-owned equipment that is deployed on a customer's premises. Typically, CPE may consist of a switch or a router. In many cases, the service provider manages the CPE and the enterprise merely pushes its traffic into the device. This helps to reduce the workload on the enterprise IT staff and replaces this with a fixed monthly outlay. Another merit of CPE from the SP perspective is that it makes it a little hard for a customer to change from one service provider to another.

CTI

Computer Telephony Integration is a technique by which computers control telephony systems such as PABXs. Call centers are a good example of CTI—calls are queued up and routed to agents based on criteria such as caller ID and called number. A less obvious example of CTI is the way in which computers are used to control the operation of mobile telephony networks.

DECT

Digital Enhanced Cordless Telecommunications is a radio technology suited for voice data and networking applications with geographical range requirements up to a few hundred meters. It can be used for in-building enterprise mobile telephony as an add-on to an existing voice system.

DEN

Directory-Enabled Networking is a specification of an object-oriented information model that models network elements and services as part of a managed environment in a repository-independent fashion. It provides a mapping of this information to a form that is suitable for implementation in a directory that uses LDAP or X.500 as its access protocol.

DES

Data Encryption Standard is a privacy protocol used to protect messages in transit across networks.

DLCI

Data Link Connection Identifier is a field in a Frame Relay link layer header. It is used as part of the addressing data for forwarding frames.

DMI

Desktop Management Interface is an independent group devoted to providing desktop management.

DNS

Domain Name System is a hierarchical system for resolving names (such as www.microsoft.com) into IP addresses.

DS

Differentiated Services is an approach to providing quality of service in networks; it employs a small, well-defined set of building blocks from which a variety of aggregate behaviors may be built. A short bit pattern in each packet, in the IPv4 Type of Service (now called the DS octet) byte or the IPv6 Traffic Class byte, is used to mark a packet to receive a particular forwarding treatment, or per-hop behavior, at each network node. DS is thought to provide a scalable way of implementing layer 3 services because the requisite actions are distributed throughout the network without the need for networkwide state machines.

DSCP

Differentiated Services Code Point is an integer value encoded in the DS field of an IP header. The DSCP is an example of traffic marking because its value corresponds with a preferred QoS as the packet traverses the network. The DSCP value corresponds to a specific QoS.

DSL

Digital Subscriber Line is an access technology that allows simultaneous Internet connectivity and voice calls. The speed is much higher than a regular modem and generally no new wiring is needed, though there may be distance limitations.

DTL

Designated Transit List is a set of hops (or a path) in an ATM network used in creating a virtual circuit. A DTL is similar to an MPLS ERO.

DWDM

Dense Wavelength Division Multiplexing is a fiber-optic transmission technique that employs light wavelengths. Multiple incoming optical signals are combined into a group for transmission over a single fiber. DWDM can be used as a means of extending the lifespan of existing fibers.

ECN

Explicit Congestion Notification is a mechanism for informing routers that congestion may be about to occur. It is always best to avoid congestion if at all possible because of the need to provide defined levels of service (QoS) for different traffic types. Some traffic types, such as email, can recover from dropped (and subsequently retransmitted) packets. The same is not true for real-time services such as VoIP. So, if a router receives a packet with ECN content, then it should try to take some actions, such as dropping low-priority traffic (or traffic that has been marked droppable). ECN is yet another example of a type of policy, in this case, an emergency traffic management policy.

ELAN

Emulated LANs can be created using ATM LAN E(mulation) technology to join together existing LANs running protocols such as IP, Novell IPX, AppleTalk, and DECnet. ELANs leverage existing underlying ATM technology to provide connectivity.

EMS

Element Management Systems are often hosted on NEs and offer various configuration options such as where to send traps and notifications. In many cases, the EMS is a front end to an SNMP agent. Also, once a device is powered up for the first time, the user can configure it using the EMS to assign IP addresses, subnet masks, and so on. An EMS can also be hosted on an external computer system and used to manage NEs that provide only a simple CLI. Somewhat confusingly, EMS is also the term used by Telcordia to describe what is often in fact an NMS.

FCAPS

Fault, Configuration, Accounting, Performance, and Security are the OSI functional areas of network management. In the **Fault** area, network problems are found and corrected. Root cause analysis may be used to give an exact reason for a given fault. In the **Configuration** area, network operation is monitored and controlled. Hardware and NE software changes are recorded along with an inventory of deployed equipment and firmware. In the **Accounting** area, resources are shared out fairly among network users. This area ensures that end users are billed appropriately. The **Performance** area is involved with managing the overall performance of the network. The **Security** area is used to protect the network against hackers, unauthorized users, and physical or electronic tampering.

FEC

Forwarding Equivalence Class is a group of IP packets that are forwarded over the same path and with the same traffic handling treatment. An FEC can be a destination IP subnet or a traffic class that the LER considers significant.

FR

Frame Relay is a layer 2 technology similar in many ways to ATM. It provides a connection-oriented model.

FTN

FEC-to-NHLFE is a MIB table that maps FECs to a next-hop label-forwarding entry. The latter is used for labeled packet forwarding and contains the next hop, the label value to be replaced, and the label stack to be added. This MIB controls the transition between the IP and MPLS domains and contains rules for MPLS-encoding IP packets and pushing them into LSPs or tunnels.

GMPLS

Generalized Multiprotocol Label Switching is also referred to as multiprotocol lambda switching. GMPLS supports not only devices that perform packet switching, but also those that perform switching in the time, wavelength, and space domains. The development of GMPLS requires modifications to current signaling and routing protocols. It has also triggered the development of new protocols such as the Link Management Protocol.

IANA

Internet Assigned Numbers Authority is an organization that was responsible for the allocation of IP addresses, port numbers, character sets, and so on. This work is now performed by an organization called the Internet Corporation for Assigned Names and Numbers (ICANN).

IDL

Interface Definition Language is a means of exporting programmatic services over a network. IDL enables distributed applications to transparently invoke operations on remote networked hosts. An IDL file is not dissimilar from a C header file except that the actual code implementation (behind the IDL definitions) is located on a host that is remote to the caller.

IETF

Internet Engineering Task Force is a organization with thousands of members. It is the governing body for Internet Protocol (IP) standards. The IETF has IP and MPLS protocol ownership and is also responsible for originating the MPLS technology.

IN

Intelligent Networking is an expression used with reference to mobile telephony networks. One application of IN is providing extra network-based software, for example, to peek at the contents of SS7 messages as they traverse a network. Depending on the data in the messages, specific actions can be taken.

INTSERV

Integrated Services is a model used for providing traffic forwarding service levels in IP/MPLS networks. It allows for microflows to be created with reserved resources (such as bandwidth) and other traffic handling characteristics (maximum packet size, maximum burst size, etc.). Traffic is pushed into these microflows in the direction of the required destination. The disadvantages of IntServ are that the microflows must be explicitly traced and reserved, and thereafter they must be refreshed. This adds traffic into the network and can cause scalability problems.

IP

Internet Protocol provides for the transmission of datagrams from a source to a destination. The source and destination are hosts identified by fixed-length IP addresses. IP also provides for fragmentation and reassembly of long datagrams if necessary.

IS-IS

Intermediate System-Intermediate System is an International Organization for Standardization (ISO) dynamic routing specification. Often deployed in IP networks, IS-IS provides similar capabilities to OSPF.

ISDN

Integrated Service Digital Network is a set of CCITT/ITU standards for digital transmission. There are two levels of service: the Basic Rate Interface (BRI), intended for home users and small to medium enterprises, and the Primary Rate Interface (PRI), for larger users.

ISP

Internet Service Provider is an SP that provides Internet access. ISPs can also offer more than just Internet access—for example, VoIP.

IT

Information Technology represents a group of people dedicated to maintaining the technical infrastructure in an organization. Infrastructure includes switches, routers, PABXs, servers, hosts, and so on.

J2EE

Java 2 Platform, Enterprise Edition, takes advantage of many features of the Java 2 Platform, Standard Edition, such as portability, JDBC for database access, CORBA support, and a security model. J2EE adds support for Enterprise JavaBeans, JavaServer Pages, Java Servlets, and XML.

JDK

Java Development Kit contains all the software and tools needed to compile, debug, and run applets and applications written using Java language. It can also be seen as a software layer that resides between Java applets (and Java applications) and the underlying platform.

JDMK

Java Development Management Kit from Sun Microsystems provides a set of Java classes and tools for developing SNMP-based, management software (agents and managers). Programs can be created, deployed, updated, enhanced, or removed in real time.

JIT

Just-in-Time is a type of Java compiler that operates after reading in a class file for interpretation. It passes the class file to the JIT, which in turn compiles the bytecodes into native code for the platform. It may be faster to do this than to just interpret the bytecodes. The JIT is an integral part of the Java Virtual Machine. Some environments allow a choice whether or not to permit JIT code.

JVM

The Java Virtual Machine is the cornerstone of the Java programming language. It is the component of the Java technology responsible for crossplatform delivery. The JVM is an abstract computing machine and (just like a real computing machine) has a defined instruction set. The JVM knows nothing of the Java language, only of a particular file format, the class file format. A class file contains JVM instructions (or bytecodes) and a symbol table, as well as other required information.

LDAP

Lightweight Directory Access Protocol is a protocol used for communicating with a directory product.

L2TP

Layer 2 Tunneling Protocol is an emerging Internet Engineering Task Force (IETF) standard that combines the features of two existing tunneling protocols: Cisco's Layer 2 Forwarding and Microsoft's Point-to-Point Tunneling Protocol. L2TP is an extension to the Point-to-Point Protocol (PPP).

LER

The Label Edge Router is a router that sits at the boundary between an IP network and the MPLS domain. IP traffic is presented to an ingress LER, labels are pushed, and the resultant packets are forwarded over an LSP. LSRs carry the encoded packets and labels are swapped. At the egress edge of the MPLS domain, another LER removes all the MPLS encoding data, performs a normal IP lookup, and forwards the packet into the IP network.

LSP

Label Switched Paths are often also referred to as tunnels. LSPs are used to transport data, such as IP packets, across an MPLS network. An LSP is a set of hops across a number of MPLS nodes. At the edge of the MPLS network, the incoming traffic is encapsulated in an MPLS frame and the latter is then routed, using the embedded label for addressing. The path traversed by an LSP can be specifically engineered for traffic so that different incoming traffic streams receive different service levels.

LSR

Label Switching Router. In this book, an LSR is considered as a core device—that is, a node that resides inside the MPLS domain boundary and does no IP forwarding. An LER, on the other hand, performs the edge function: It applies the initial label to the packet after performing a conventional longest-match prefix lookup on the IP header. After the packet is labeled, the intermediate LSRs forward it using only the label. LSRs usually replace the label on an incoming packet with a new value as they forward it; thus, the forwarding mechanism is based on label swapping.

MAP

Mobile Application Part is a protocol that utilizes SS7 to allow roaming and other mobile telephony capabilities, such as text messaging.

MD5

Message Digest 5 is a standard algorithm that takes as input a message of arbitrary length and produces as output a 128-bit fingerprint or message digest of the input. Any modifications made to the message in transit can then be detected by recalculating the digest. Similar in concept to a CRC, the MD5 algorithm is used as part of the SNMPv3 security subsystem.

MIB

A Management Information Base is a formal description of a set of objects that can be managed using SNMP. MIB-I refers to the initial MIB definition, and MIB-II refers to the current definition. There exist MIB extensions for each set of related management entities, such as the MPLS TE and LSR MIBs. Standard MIBs can be extended to include proprietary objects.

MPLS

Multiprotocol Label Switching is a new technology designed to overcome some of the limitations of IP routing. IP packets are

assigned to an FEC at the edge of the MPLS domain. This occurs just once in contrast to IP routing, where it occurs at every hop. The assigned FEC is encoded as a short, fixed-length value known as a label that is prepended to the packet. When the packet is forwarded to its next hop, the label is sent along with it, and there is no further analysis of the network layer header. Instead, the label is used as an index into a label information base table that specifies the next hop and a new label. The old label is replaced with the new label, and the packet is forwarded to its next hop. This process continues until the packet arrives at the outer edge of the MPLS domain, where the label is stripped off and a normal IP forwarding operation is executed. Labels are flexible objects and can be included as part of sophisticated traffic engineering schemes; for example, a given label value can receive specific traffic handling.

MTP

Message Transfer Part is the part of a common-channel signaling system (such as SS7) that transfers signal messages between network nodes and performs functions such as error control and signaling link security.

MVNO

Mobile Virtual Network Operator is a relatively new breed of operator that uses the infrastructure of another operator to provide a competitive service. Unused capacity in the host network is sold to the MVNO and cross-selling of services may be possible.

NAP

A Network Access Point is an Internet interconnection point that is used to tie all the Internet access providers together. NAPs provide switching facilities

NBI

Northbound Interface describes an interface offered by many NMS products. The NBI allows for NMS features, functions, and data to be accessed by an OSS. TMF-814 (from the TeleManagement Forum) is one standardized model for an NBI based on CORBA. The OSS can use the NBI to retrieve information from the NMS in any of the FCAPS areas. It is also possible for an OSS to automate many of the NMS functions, such as provisioning. In this way, an OSS can avoid the need for a network operator to use the GUI provided with the NMS. This can help facilitate integration of the NMS into an SP environment.

NE

Network Element. This is a device that resides inside a managed network. Typically, an NE provides some services to a network operator, such as ATM or Frame Relay virtual circuits, MPLS, and IP. NEs host MIBs and the objects in these MIBs can be used by network management systems.

NGN

Next-Generation Network is a generic term used to describe the emerging packet-based networks. Such networks feature mixed traffic types such as voice, video, and data, and each traffic type receives an appropriate class of service. Realizing such converged networks requires special-purpose technology in the network, such as MPLS.

NHLFE

Next-Hop Label Forwarding Entry is a table that contains the next hop, the label value to be replaced, and the label stack to be added to an MPLS-encoded packet.

NIC

Network Interface Card is a term used to describe a peripheral circuit board or card installed in a computer that facilitates connection to a LAN. NICs may implement networking technology such as ATM, Ethernet, or token ring. NICs connect to some shared medium, such as an Ethernet cable, or they may connect to a switch.

NMS

Network Management System is a term that describes a computer-based software application suite dedicated to the management of networks of NEs. Typically, the NMS provides abstractions (such as signaling links and virtual connections) appropriate to the overall running of a network; that is, it is not exclusively concerned with the details of one NE. Communication between an NMS and NEs is typically executed via an EMS, where the latter may reside on the NE. Above the NMS, the OSS is found.

NOC

Network Operations Center. A NOC is a location where a telecommunications or data network is managed. Enterprises with large networks as well as service providers may use the services of a third-party NOC. This shifts the burden of management onto the third party and serves to explicitly define the financial outlay required for managing the network.

OID

Object Identifiers are strings of numbers organized in a hierarchical fashion. Every object in a MIB has a unique OID. The Internet OID is 1.3.6.1. The dot notation is an IETF invention, but the ITU preferred a notation using spaces and braces, with optional text labels, so that 1.3.6.1 would look like any one of the following:

{iso(1) org(3) dod(6) iana(1)}

{1 3 6 1}

OOD

Object-Oriented Design is a general field that seeks to provide an abstraction for software development based on real-world objects.

ORB

Object Request Broker is the software that functions as a broker (or intermediary) between a client request for a service from a distributed object or component and the completion of that request. Providing ORB support in a network means that client programs can request remote services without needing any knowledge concerning the location of the associated server. Likewise, it is not necessary for the client to have details of the interface to the server program.

OSPF

Open Shortest Path First OSPF is an IP routing protocol used inside autonomous systems. OSPF is controlled by the IETF as one of several Interior Gateway Protocols (IGPs). With OSPF, a host that detects a change to its routing table (such as an interface going up or down) immediately transmits the information to all other OSPF hosts in the network. What then follows is a process called convergence in which all OSPF hosts try to build the same routing picture of the network. Routing and network management are conceptually similar in that both try to maintain parity between external dynamic entities. In the case of routing protocols, the external entity is network topology. In the case of network management, the external entity is the set of remote SNMP agents.

OSI

Open Systems Interconnection is a complete suite of routing protocols developed by the International Organization for Standardization (ISO). OSI protocols include Intermediate System-to-Intermediate System (IS-IS), End System-to-Intermediate System (ES-IS), and Interdomain Routing Protocol (IDRP). Two important OSI networking terminology terms are nonrouting network nodes or end systems (ES) and routers or intermediate systems (IS). These two terms form the basis for the ES-IS and IS-IS OSI protocols. The ES-IS protocol enables ES and IS to discover each other. The IS-IS protocol provides routing between IS.

OSS

Operations Support System refers to the system that handles workflows, management, inventory details, capacity planning, and repair functions for service providers. Typically, an OSS uses an underlying NMS to actually communicate with the network devices. It is also possible for an OSS to communicate directly with devices. In like fashion, the OSS is itself often used by the business support system.

PBNM

Policy-Based Network Management technology provides the ability to define and distribute policies to manage enterprise and SP networks. Policies can reside either on devices themselves or in the network management system and exist to control essential network resources such as traffic engineering, bandwidth, and security.

PDP

Policy Decision Point is an entity in a policy-based system where decisions are made. PDPs can be devices such as routers carrying out traffic handling.

PDR

Performance Data Record (similar in concept to CDR) describes a block of data emitted by a network device to indicate some aspect of performance—for example, the number of IP packets sent or received. PDRs can be aggregated by external software to provide an overall picture of performance.

PDU

Protocol Data Unit is an expression that describes the basic information element of a given protocol; for example, SNMP has various PDUs, such as [get](#) and [get-next](#). The latter describe protocol operations and are encoded in the form of messages before being sent to another protocol entity.

PEP

Policy Enforcement Point is an entity in a policy-based system where decisions are enacted. PEPs can be devices such as routers carrying out traffic handling.

PHB

Per-Hop-Behavior is a way of describing the forwarding treatment experienced by a packet at each network node in a DiffServ domain. A bit pattern in each IP packet header, in the IPv4 TOS octet or the IPv6 Traffic Class octet, is used to mark a packet to receive a particular forwarding treatment, or per-hop behavior. The IETF has standardized a common layout for a six-bit field, called the DS (or Differentiated Services) field. RFC 2474 and RFC 2475 define the architecture and the general use of bits within the DS field (superseding the IPv4 TOS octet definitions of RFC 1349).

PIB

Policy Information Base is a virtual repository for policy information.

PLC

Programmable Logic Controller is a device used to automate monitoring and control of industrial plant.

PNNI

Private Network-To-Network Interface is an ATM Forum protocol that supports QoS and hierarchical operation in ATM networks. It supports routing and signaling in multivendor ATM networks. PNNI hierarchy is provided via peer groups—any nodes that share a given peer group ID elect a peer group leader, which then represents the peer group in the next level of hierarchy. Each PNNI node has a topology database that represents its view of the network. Signaling is used to create connections (e.g., SPVCCs) across the network.

POP

Point-of-Presence is an access point to the Internet. A POP must have a unique IP address. An ISP has one or more points-of-presence on the Internet. A POP may reside in rented space owned by a telecommunications carrier (e.g., AT&T) to which the ISP is connected. A POP usually includes switches, routers, servers, and so on.

POTS

Plain Old Telephone Service is a term often used to describe traditional telephone technology.

PSTN

Public Switched Telephone Network PSTN is the complete set of global voice-oriented public telephone networks. Often referred to as the Plain Old Telephone Service (POTS), the PSTN is a vast aggregation of circuit switching telephone networks that spans the globe.

PVC

Permanent Virtual Connections are software-created logical connections in a network such as ATM or Frame Relay. PVCs are generally created link-by-link and node-by-node in a set of manual steps. Once all the required PVCs have been created, it is possible for traffic to flow through the overall connection.

PVX

This is the generic name for all Permanent Virtual Connection types. For ATM, these include PVPs and PVCs.

QA

Quality Assurance is a group of people dedicated to testing software releases. Often amounting to half the total cost of a complex project, testing is a critical function. With development budgets increasingly squeezed, QA is often the last line of

defense before customer releases.

QoS

Quality of Service refers to the capability of a given network to provide preferential service to one type of traffic over another. It defines the ability of the network to deliver services other than, say, best effort (in the case of IP). A number of technologies, including ATM, Frame Relay, and MPLS, provide a degree of QoS such as priority, guaranteed bandwidth, and maximum burst size. In particular, ATM provides five service categories: *CBR* (constant bit rate used by connections that require a static quantity of bandwidth that is continuously available for the lifetime of the connection), *rtVBR* (real-time variable bit rate used by connections that require tightly constrained delay and delay variation, e.g., voice and video applications), *nrtVBR* (non-real-time VBR service category used by applications that are bursty in nature), *ABR* (available bit rate used by applications that can accommodate changes—subsequent to connection establishment—in the ATM layer transfer characteristics of the network), and *UBR* (unspecified bit rate used by non-real-time applications that do not require tightly constrained delay and delay variation). QoS is of increasing importance on layer 3 networks as time-constrained traffic grows.

RADIUS

Remote Access Dial-In User Service is a combination of a client/server protocol and software that enables remote access servers to communicate with a central server for the authentication of dial-in users, virtual private network users, and wireless network users. RADIUS allows for the storage of user profiles in a central database for sharing by remote servers. A central service also makes the collection of statistics and usage data for billing easier to manage.

RAS

A Remote Access Server is a combination of a computer and specific software that exists to provide remote network access to users. A RAS is often associated with a firewall to ensure security and may operate in conjunction with a router for forwarding the remote access requests to some other part of the network. A RAS may also be used as part of a virtual private network or a wireless network.

RFC

Request for Comments is a series of notes maintained by the IETF concerning the Internet. RFCs cover a broad range of topics with the principal focus on network protocols, procedures, programs, and concepts. RFCs are an indispensable tool for learning about Internet technology.

RMI

Remote Method Invocation provides a means for invoking the methods of remote Java objects. The caller must first acquire a reference to the remote object, for example, by looking it up in the RMI bootstrap naming service or by receiving a reference as an argument or a method return value. Using the object reference, a call can be made on the remote server object. The server can in turn be a client of other remote objects. RMI technology uses object serialization to marshal and unmarshal parameters between method calls; it does not truncate types, supporting true object-oriented polymorphism.

RPC

Remote Procedure Calls are made by software to functions hosted on remote machines. An RPC acts just like an ordinary function call except that it results in remote computation that occurs across a network in a transparent fashion. Parameters to the function call are passed across the network (a process called marshalling) to the called function; the remote function executes and returns a result. The result is then passed back across the network to the caller. All of this occurs in a transparent fashion.

SAN

Storage Area Networks are a means of separating storage from both host machines and corporate networks. SANs deploy storage inside the boundary of a dedicated, high-speed network. A boundary device then manages (reads and writes) the storage. Some commentators feel that once an organization's storage needs pass the terabyte mark, it is time to start using SAN technology.

SCCP

Signaling Connection Control Part is a component of the SS7 protocol suite that provides additional functions to those of the message transfer part (MTP). SCCP provides both connectionless and connection-oriented network services to transfer signaling information across telecommunication networks, such as GSM.

SDL

Specification and Description Language is an object-oriented, formal language defined by the International Telecommunications Union–Telecommunications Standardization Sector (ITU-T) as recommendation number Z.100. SDL can be used intended for the specification of complex, event-driven, real-time, and interactive applications involving many concurrent activities that communicate using discrete signals. Typically, SDL programs consist of many state machines controlled by signals. SDL programs can also communicate with the outside world through various programming languages (e.g., C) via what are called environment modules. Execution speed of these state machines is very often extremely impressive and faster than handcrafted C code.

SHA1

The US Secure Hash Algorithm takes a message of less than 2^{64} bits in length and produces a 160-bit message digest designed so that it is computationally very expensive to find a text string that matches a given hash.

SIN

Ships-in-the-Night allows for ATM Forum and MPLS control planes to both run simultaneously on the same hardware but isolated from each other; that is, they do not interact. The label space may be divided between the two technologies. SIN allows a single device to simultaneously operate as both an MPLS node and an ATM switch. This can be important when

migrating MPLS into an ATM network.

SLA

Service Level Agreement describes the performance provided to the customers of a given SP network. The items included in SLAs can include bandwidth, delay, uptime guarantees (anything from 0% to 99.999%), refund terms for SLA nonconformance, and so on. Some service providers provide Web-based facilities to allow customers to see how an SLA is being met. Other service providers provide reports on a monthly basis. An interesting trend is enterprise IT departments writing their own SLAs for their customers (other sites, divisions, and departments within the enterprise). These SLAs can then be measured and possibly even compared with an outsourced provider.

SMS

Short Message Service is a service for sending text messages to GSM mobile phones. GSM and SMS service is primarily available in Europe. SMS is similar to paging, and in some countries paging networks have been replaced by SMS. SMS has experienced an amazing upsurge in popularity in Europe and is used extensively by young subscribers.

SNMP

Simple Network Management Protocol is an IETF protocol used for network management. Even though SNMP is one of the TCP/IP protocols, it is not restricted to use in TCP/IP networks. The success of SNMP is mostly due to its simplicity and lightweight features. The managed objects supported by a given device are encoded in its MIB or schema description. SNMP entities include managers and agents (both proxy and non-proxy), and a simple messaging protocol is used between these entities. Operations from the manager side include set (modify) and get (retrieve), and agents can respond these with reference to a security framework. Agents can also issue notifications or traps to manager in order to indicate important events.

SOAP

Simple Object Access Protocol is a lightweight protocol for exchanging information in decentralized, distributed environments. It is an XML-based protocol and consists of three parts: an envelope for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a means of representing remote procedure calls and responses.

SP

Service Providers are companies that provide data and telecommunications services to end users. SPs sell an increasingly broad range of services, including products for access, aggregation, and transport. An interesting trend for SPs is the provision of software application infrastructure, SANs, content delivery networks—putting servers near the network edge to minimize the effects of congestion in the first mile (server-to-backbone), backbone, peering points (between carriers), and in the last mile (network-to-end user). These relatively new service offerings move SPs out of the simple bandwidth reselling area into a more diverse and complex marketplace. The amount of (and diversity of, i.e., non-real time, time-constrained) traffic crossing SP networks is increasing all the time. Many of the larger SPs are planning for a single packet-based network, and MPLS is an excellent contender for helping fulfill this.

SPPI

Structure of Policy Provisioning Information (RFC 3159) describes a provisioning model for policy information. The model views the policy information as a collection of provisioning classes and provisioning instances that reside in a virtual information store, termed the Policy Information Base (PIB). Collections of related provisioning classes are defined in a PIB module in a similar fashion to a MIB module. Just as for SNMP, PIB components can be written to (or read from) remote devices. The devices in turn implement any such policies.

SPVCC

Switched Permanent Virtual Channel Connection describes a connection with fixed endpoints but switched in the middle. ATM provides two types of SPVCs: switched permanent virtual path connections (SPVPCs) and switched permanent virtual channel connections (SPVCCs). The switched component of these connection types allows for more resilience than is the case with PVCs because intermediate node or link failure results in possible rerouting rather than just failure.

SPVX

This is the generic name for all Switched Permanent Virtual Connection types. These can be SPVPCs, SPVCCs, and so on.

SS7

Signaling System Number 7 is a network architecture that separates the management of telephone networks from the actual call switching. SS7 provides many benefits because the SS7 software applications can reside in computers rather than telephone exchanges. This allows for a healthy separation between software vendors and switch/exchange manufacturers.

TCAP

Transaction Capability Application Part is an SS7 protocol layer and allows applications to invoke software procedures at remote locations in an SS7 network. TCAP provides transaction and component handling capabilities as well as load sharing between multiple instances of the same application.

TCP

Transmission Control Protocol (RFC 793) is used as a reliable protocol between hosts in packet-based communication networks. TCP is a connection-oriented, end-to-end reliable protocol that sends and receives variable-length segments of information. TCP is conceptually located between the end user on one side and IP on the other side. Users can push and pull segments via TCP in a reliable fashion with flow control. Multiplexing and demultiplexing is provided using a socket abstraction. TCP also allows the creation of connections between processes (similar in concept to the connections we've seen in this book).

TDM

Time Division Multiplexing is a method of transporting voice communications.

TL1

Transaction Language One is a widely used telecommunications management protocol. TL1 is a vendor-independent and technology-independent man-machine language. TL1 facilities can be provided as part of an OSS for interacting with either underlying management systems or NEs. One popular application is for a management system (or NE) to package its trap/notification data in TL1 format and forward it to an OSS component. An OSS may also provide a TL1-based facility for sending commands to the lower layers.

TLS

Transparent LAN Service describes a method of providing a LAN service from one enterprise customer site to another through an intermediate SP cloud. The technology inside the SP is irrelevant to the enterprise users because their focus is using the LAN service. The latter is supplied transparently via the SP.

TMN

Telecommunications Management Network (defined in ITU M.3000 recommendation) provides a framework or model to support the management and deployment of telecommunications services. Methods are defined for managing networks using object-oriented principles, and standard interfaces facilitate communication between deployed management entities. The standard management interface for TMN is called the Q3 interface. Interoperability is a key aspect of TMN-compliant networks. TMN is based on the OSI network management framework and includes the familiar concepts of MIBs, agent, managers, and so on.

TMN describes network management from a number of interdependent viewpoints including:

- A logical or business model
- A functional model
- A set of standard interfaces

TMN consists of a small number of components that combine to provide a powerful management capability.

TOM

Telecommunications Operations Map is a model for representing common business processes from the customer's point of view. The TOM is independent of organizations, technologies, and services. It supports the implementation of end-to-end operations integration (often called flow-through operations) and automation. By modeling business processes, the TOM gives new service providers a starting point and existing service providers a reference point for the implementation and management of business-level processes. Management systems can then be built to support TOM. By using a common model, the overall business management systems can be more quickly developed and deployed. Also, it becomes easier to implement lower level management systems because interfaces are standardized.

TOS

Type of Service (RFC 1349) is a single-byte field in an IP packet header that specifies the service level required for the packet. It is now called the DS field and can have the following values:

- Bits 0–2: Precedence
- Bit 3: 0 = Normal Delay, 1 = Low Delay
- Bit 4: 0 = Normal Throughput, 1 = High Throughput.
- Bit 5: 0 = Normal Reliability, 1 = High Reliability
- Bits 6–7: Reserved for Future Use

UML

Unified Modeling Language is the industry-standard language for the specification, visualization, construction, and documentation of the components of software systems. UML helps to simplify the process of software design, making a model for construction with a number of different views. One of the great merits of UML is the way it helps open up the development process with what are called use cases. These serve to identify principal roles (actors) in the system, boundaries, actions, and so on. Software developers need not write use cases; instead, other stakeholders can provide them. Integrated tools allow for use cases to be incorporated into the development process.

USM

User-based Security Model is an administrative framework (or security model) that defines the mechanisms used to achieve a defined level of security for management protocol interactions (gets, sets, notifications, etc.). The USM for SNMPv2 defines a security model for this administrative framework that includes an access control model. The enforcement of access rights requires the means to identify the user who generates a network request. Data integrity protection is provided via a message digest algorithm. Authentication is provided via a secret value inserted into and appended to the message. Protection against message replay or delay is provided by time indicators and request-id counters.

VCI

Virtual Channel Identifier is one of the header fields in an ATM cell. The VCI is assigned during the virtual connection setup and used by each switch for forwarding.

VLAN

Virtual Local Area Network is a very flexible type of LAN in which machines located in the same physical area are *not* necessarily on the same LAN broadcast domain. VLANs can be implemented using sophisticated switches. Individual workstations are connected to switch ports (e.g., 10/100/1000Mbps), and it is the job of the latter to create the broadcast domain. VLANs can be of different types: port-based, MAC-based, or IEEE 802.1p/Q-based. Port-based VLANs relate to the switch port on which the end device is connected. MAC-based VLANs relate to the MAC address of the end device. 802.1p/Q-based VLANs use the contents of a 2-byte tag in the layer 2 header. Three bits are reserved for priority, and 12 bits are used for encoding VLAN IDs.

VoIP

Voice-over-IP is a telephony term describing the facilities for managing the delivery of voice using IP. It involves sending voice information in some digital form in discrete packets rather than in the traditional circuit-oriented format of the PSTN. One advantage of VoIP is toll bypass—that is, avoiding the tolls charged for ordinary telephone service. Many organizations use VoIP internally over WAN links to reduce telephony service costs. Another cost saving is that VoIP leverages existing IP infrastructure and reduces the need for traditional telephony equipment (PABX, TDM phones, etc.).

VPI

Virtual Path Identifier is one of the header fields in an ATM cell. The VPI is assigned during the virtual connection setup and used by each switch for forwarding.

WAN

A Wide Area Network is a geographically distributed telecommunications network. Often, a WAN is employed to interconnect LANs across a corporation. A WAN may be privately owned, leased, or rented, but normally includes some element of public networks.

XML

Extensible Markup Language is a flexible (text-based) way of creating common information formats. XML facilitates the sharing of both format and data.

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[Team LiB]

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

ATM

[PVC/SPVC/SPVCC](#)

[VCI 2nd](#)

[VPI 2nd](#)

[Team LiB]

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

Bandwidth

[ATM contract](#)

[overengineering](#) [2nd](#) [3rd](#) [4th](#)

[Billing](#)

[Billing Automatic Message Accounting Format \(BAF\)](#)

[CDR](#)

[Team LiB]

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[CLI](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

Components

[loose coupling](#)

[COTS](#) [2nd](#)

[CPE](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

DiffServ

[assured forwarding](#)

[DSCP](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[expedited forwarding](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

EMS

[difficulties](#) [2nd](#)

Enterprise

[billing](#)

[depreciation](#)

[funding](#)

[networks](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#)
[26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#) [45th](#) [46th](#) [47th](#) [48th](#) [49th](#) [50th](#)
[51st](#) [52nd](#) [53rd](#) [54th](#) [55th](#) [56th](#) [57th](#) [58th](#) [59th](#)

[Team LiB]

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[FCAPS](#)

[Five 9s](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

HP OpenView

[Network Node Manager \(NNM\)](#)

[product family](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[ICMP](#)

IETF

[MPLS MIB tables](#)

[MPLS MIBs](#)

[IntServ](#)

[microflow](#)

IP

[autonomous system](#)

[ipInReceives](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[loopback address](#) [2nd](#)

[IPX](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

Legacy Technologies

[ATM](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#)

[Ethernet](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[FR](#)

[Frame Relay](#)

[IEEE 802.1Q](#) [2nd](#)

[LAN](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[Linux](#) [2nd](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[MIB](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#)
[27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#) [40th](#) [41st](#) [42nd](#)

[central role of](#)

[columnar relationships](#)

[compressed object](#)

[default values](#) [2nd](#)

[extensions](#)

[keywords](#)

[manageability](#)

[modules](#)

[object instances](#)

[RFC 2578](#)

[Walk](#)

Middleware

[CORBA](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#)

[IDL](#)

[RMI](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[MPLS](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#)

[4-byte shim header](#)

[cloud](#)

[connection-oriented](#)

[control plane](#)

[creation of LSP](#)

[devices](#)

[DiffServ](#)

[E-LSP](#)

[ERO](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[forwarding equivalence class \(FEC\)](#)

[forwarding technology](#)

[FTN MIB](#)

[interface](#)

[interfaces](#) [2nd](#)

[interoperability](#)

[IntServ](#)

[IP/MPLS boundary](#) [2nd](#)

[L-LSP](#)

[label stack](#)

[LER](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[link coloring](#)

[LSP](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#)

[27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#) [45th](#) [46th](#) [47th](#) [48th](#) [49th](#) [50th](#) [51st](#)

[52nd](#) [53rd](#) [54th](#) [55th](#) [56th](#) [57th](#) [58th](#) [59th](#) [60th](#) [61st](#) [62nd](#) [63rd](#) [64th](#) [65th](#) [66th](#) [67th](#) [68th](#) [69th](#) [70th](#) [71st](#) [72nd](#) [73rd](#) [74th](#) [75th](#)
[76th](#) [77th](#) [78th](#) [79th](#) [80th](#) [81st](#) [82nd](#) [83rd](#) [84th](#) [85th](#)

[LSP path computation](#)

[LSR 2nd](#)

[LSR MIB 2nd](#)

[managed objects 2nd](#)

[resource block](#)

Signaling

[LDP 2nd](#)

[RSVP-TE 2nd](#)

[SIN 2nd 3rd 4th](#)

[TE MIB 2nd](#)

[tunnel](#)

[tunnel instances](#)

Multiservice Switches

[access link types](#)

[technologies](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

NE

ROI

Network Management

end-to-end

fault detection

fault indication

[FCAPS](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#)
[26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#)

lifecycles

monitoring

provisioning

root-cause analysis [2nd](#)

scheduling

the problems [2nd](#)

NMS

Accounting Server

backend/server

client/server design

Configuration Server

deterministic performance of

development skills

facilities

Fault Server

generic software

GUI [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

managing higher-level services

network discovery [2nd](#) [3rd](#) [4th](#)

networkwide perspective

operations

Performance Server

quality of an

reporting

scheduling

stovepipe [2nd](#)

thin clients

topology

NNM

as a platform

backup and restore

data warehousing

discovery and mapping

[Java interface](#)
[logical maps](#)
[managed nodes](#)
[monitoring 2nd](#)
[notification processing](#)
[remote administration](#)
[reporting](#)
[topology](#)
[unmanaged nodes](#)

[Team LiB]

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[OSS](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[NBI](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[PBNM](#)

[COPS-PR](#)

[DEN](#)

[LDAP](#)

[PDP](#)

[PEP](#)

Policy

[and IP](#)

[network intelligence](#)

[Policy Information Bases \(PIBs\)](#)

[Programmability](#)

[PWE3](#) [2nd](#) [3rd](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[QoS](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#)

[decision making](#)

[DiffServ](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[EXP field](#)

[Explicit Congestion Notification](#)

[IP](#)

[IP best effort](#)

[mapping](#)

[models](#)

[per-hop-behavior](#)

[Team LiB]

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[SAN](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

Scalability

[aggregate objects](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[defined](#)

[next free index](#)

[tunnel-change table](#)

Security

[3DES, AES](#)

Service provider

[networks](#) [2nd](#) [3rd](#) [4th](#)

[Signaling](#) [2nd](#)

[multidomain](#)

[PNNI](#)

[RSVP-TE](#) [2nd](#) [3rd](#) [4th](#)

[SLA](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[enterprise and SP](#)

[SNMP](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#)

[community](#)

[configuring on Windows](#)

[development with JDMK](#)

[exceptions](#)

[notification](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[OID](#) [2nd](#) [3rd](#)

[problems with](#)

[retries](#)

[scalability](#)

[security violation](#)

[SNMPv3](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#)

[timeouts](#)

[walk](#)

SNMP\

[community string](#)

SNMPv3 elements

[access control subsystem](#)

[applications](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[authentication](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[dispatcher](#)

[message format](#) [2nd](#)

[message subsystem](#)

[privacy 2nd](#)

[security subsystem](#)

Software

[components 2nd](#)

Solutions

[business processes](#)

[chess](#)

[component development](#)

[domain experts](#)

[exceeding the remit](#)

[foundation releases 2nd](#)

[institutional memory](#)

[linked overviews](#)

[mobile developers](#)

[task orientation](#)

[testing](#)

[thin software layers](#)

[think in chunks](#)

[trace files](#)

[UML 2nd 3rd 4th 5th 6th 7th 8th 9th 10th](#)

[zero-defect](#)

[Spanning Tree Protocol](#)

[SS7](#)

storage

[management](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

Timing

voice, video, and data

Traffic engineering [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

MPLS killer app

path creation

Trouble ticketing

[Team LiB]

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[Unix](#)

[\[Team LiB \]](#)

[Team LiB]

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[VLAN](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#)
[27th](#) [28th](#)

[broadcast domain](#)

[label-to-VLAN ID](#)

VoIP

[enterprise technology](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#)

[real time](#)

[VPLS](#)

VPN

[access, connecting sites](#) [2nd](#)

[layer 2](#)

[layer 3](#) [2nd](#) [3rd](#)

[N2 problem](#)

[Team LiB]

[\[Team LiB \]](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [S](#) [T](#) [U](#) [V](#) [W](#)

[WAN](#)

[connections](#)

[management](#)

[MPLS](#)

Web browser

[NMS component 2nd](#)

[Windows 2nd](#)

[service pack 6a](#)

[\[Team LiB \]](#)

Brought to You by



Like the book? Buy it!