NEAR EAST UNIVERSITY

Faculty of Engineering

Department Of Computer Engineering

MANAGING SNMP AGENTS WITH SNMP (SIMPLE NETWORK MANAGEMENT PROTOCOL)

Graduation Project
COM – 400

Student : Khalid Khurshid

Supervisor: Asst. Prof. Dr Rahib Abiyev

Nicosia – 2002

ACKNOWLEDGEMENTS

All glory to Almighty ALLAH, the Lord of universe, who is the entire source of all knowledge and wisdom endowed to mankind. All thanks are due to ALLAH who enabled me to complete this project.

I want to express my gratitude and indebtedness to my project supervisor Dr Rahib Abiyev for his deep interest, continuous guidance, assistance and cooperation at every stage of the project.

Then I want to say thanks to my family for their encouragement and support.

Finally I would like to thank all of my friends for their help.

ABSTRACT

Two types of network management issues exist: software related, such as data security and access permissions; and hardware related such as workstations, servers, network cards, routers, bridges and hubs. For hardware related network management, ISO offers a framework for network management that divides the functions of network management into five Specific Management Functional Areas.

Network management is divided into four categories: Managed Nodes, Agents, Network Management Station and Network Management Protocol. SNMP is a family of protocol suits and specifications that provides a means for collecting network management information from devices on the network. It includes Management Information Base (MIB) that is a database for keeping information in the managing and managed devices, Structure of Management Information (SMI) and Simple Network Management Protocol (SNMP) that provides a way for the devices to report problems and errors to the network management station.

There are several vendors that support SNMP-based network management including Asante Technologies' IntraSpection, Cabletron Systems' SPECTRUM, Hewlett-Packard OpenView, Novell's ManageWise, Sun Microsystems' Solstice Domain Manager and Tivoli Systems' TME 10 NetView.

TABLE OF CONTENTS

INTRODUCTION

Since it was developed in 1988, the Simple Network Management Protocol (SNMP) has become the de facto standard for internetwork management. SNMP has a number of advantages that contribute to its popularity. Because it is a simple solution, requiring relatively little code to implement, vendors can easily build SNMP agents into their products. SNMP is extensible, allowing vendors to easily add network management functions. And SNMP separates the management architecture from the architecture of the hardware devices, which broadens the base of multivendor support. Perhaps most importantly, unlike other so-called standards, SNMP is not a mere paper specification, but is an implementation that is widely available today.

In order to fully understand the depth of network management, let's discuss these concepts one chapter at a time. Chapter 1 provides an overview of the concepts of network management. Individual sections discuss the OSI, IEEE, and Internet network management standards. Other sections consider architectures from key vendors that support these standards: Asanté Technologies, Cabletron Systems, Hewlett-Packard, Novell, Sun Microsystems, and Tivoli Systems. SNMP is only part of what is known as the Internet Network Management Framework. Chapters 2, 3, and 4 discuss individual sections of that framework. In order, these topics are the structure of management information (SMI), management information bases (MIBs), and SNMP itself.

The SMI provides a mechanism for describing and naming the objects being managed. This structure allows the values of these objects to be retrieved and manipulated, that is, managed. It accomplishes this by using a message description language, defined by ISO 8824, known as the Abstract Syntax Notation One (ASN.1). ASN.1 is used to define the syntax, or form, of a management message. Once this syntax has been specified with ASN.1, the Basic Encoding Rules (BER)—from ISO 8825—encode that message into a format that can be transmitted on a LAN or WAN. The MIBs more precisely delineate the managed objects and organize these objects for ease of use. Different types of MIBs are available, including the Internet-standard MIB, defined in Request for Comments (RFC) documents 1212 and 1213; the remote monitoring MIBs, defined in RFCs 1513, 1757, and 2021; and numerous private enterprise MIBs that vendors define specifically for their products.

SNMP completes the story by providing a mechanism for the manager to communicate with the agents. This communication involves reading the values of the objects within a MIB and altering the values as appropriate—in other words, managing the objects. Enhancements, known as SNMP version 2 (SNMPv2), extend the capabilities of this popular protocol. Chapter 5 provides an overview of the management and security improvements found in SNMPv2. Since SNMP is an Application Layer protocol, it must rely on other protocols at the lower OSI layers for other communication functions. Chapter 6 studies these protocols. For example, the User Datagram Protocol (UDP) transports the SNMP message through the internetwork. The Internet Protocol (IP) provides Network Layer functions, such as addressing, for the datagram. A third protocol, such as Ethernet or token ring, then delivers the information to the local network.

CHAPTER ONE

NETWORK MANAGEMENT ARCHITECTURES

This chapter gives an overview of the currently available network management technologies and explains how the Simple Network Management Protocol (SNMP), fits into the big picture.

1.1 Three Decades of Network Evolution

The 1970s was the decade of the centralized network. In a decade dominated by mainframe processing, data communication allowed terminals to talk to the mainframe (Figure 1.1). Low speed, asynchronous transmission was the norm. Mainframe providers such as IBM and communication circuit providers such as AT&T or the local telephone company managed the network for those systems.

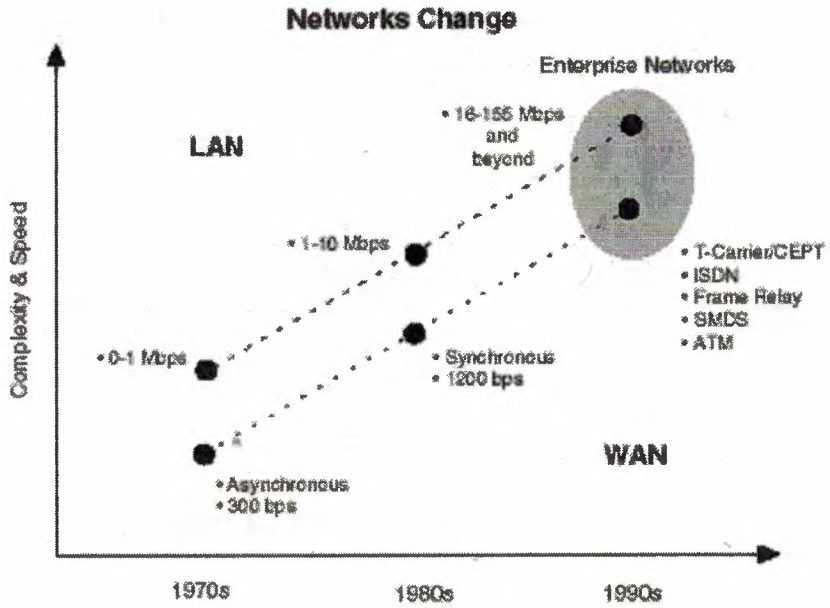


Figure 1.1 Evolution in networking complexity and speed (Courtesy Wandel & Goltermann)

The 1980s saw three significant changes in data communications. Microprocessors came onto the scene, offering significant price and performance advantages over mainframes.

The number of microcomputer-based LANs increased. And high-speed wide area transmission facilities, such as T-carrier circuits, emerged to connect microcomputer-based LANs. The proliferation of LANs gave rise to distributed processing and moved applications off the mainframe and onto the desktop. And as data communication shifted to distributed networks, network management became distributed as well (Figure 1.2). Further shifts are coming from the use of World Wide Web–based technologies, which utilize widely available Web browsers to access network management information.

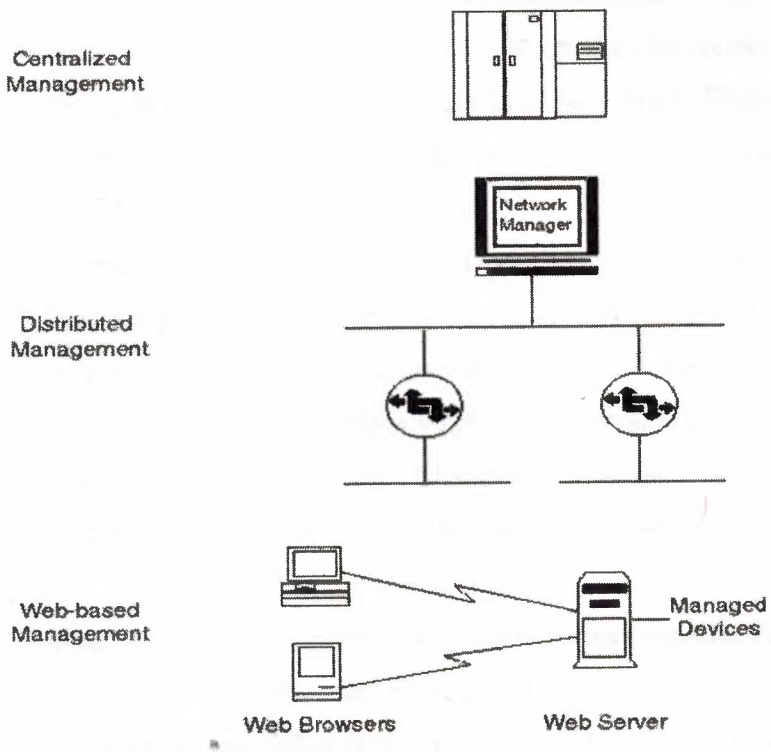


Figure 1.2 Evolution in distributed systems

Today, LANs and distributed computing have matured. Wide area network (WAN) technologies such as Asynchronous Transfer Mode (ATM), Switched Multimegabit Data Service (SMDS), and Frame Relay are meeting the needs of high-speed applications. Network management capabilities have matured as well.

1.2 The Challenge of Distributed Network Management

Network management has two parts: the network and the management. To manage a network properly, all of the people involved must agree on the meaning of network management and on its objectives.

Network management can mean different things to the different individuals in an organization, such as the chief executive officer (CEO), the chief information officer (CIO), and the end users. The CEO tends to view the network (and its manager) as a line item on the expense budget. CEOs consider computing and data communications as a way to manage orders, inventory, accounting information, and so on. As long as overall corporate revenues hit their target, these budget items are likely to remain intact. Therefore, the CEO would define network management as the financial management of the corporate communications network.

The CIO must look at network management from the theoretical perspective of the CEO and the corporate budget as well as from the practical perspective of the end users. The goal is to keep the corporate network running 99.99 percent of the time and to schedule periods of downtime on weekends and holidays when few are around to notice. The CIO would, therefore, define network management as the ability to balance increasing end-user requirements with decreasing resources—that is, the ability to provide more service with less money.

End users spend their days in the network trenches, designing airplanes, writing dissertations, and attending boring meetings. Their jobs depend on the network remaining operational. Thus, end users would define network management as something that keeps the data communication infrastructure on which they depend working at all times. A network failure could threaten their livelihood.

1.3 The System Being Managed

Now, let's shift to a systems-engineering perspective on network management. Figure 1.3 shows the big picture. On the left side of the diagram are centralized applications such as an inventory control system or the corporate financial database. The right side illustrates distributed applications, such as those that run on client-server LANs. In the middle is the glue that connects the different types of systems—the wide area transport.

This transport may consist of public and private networks and software defined networks (SDN).

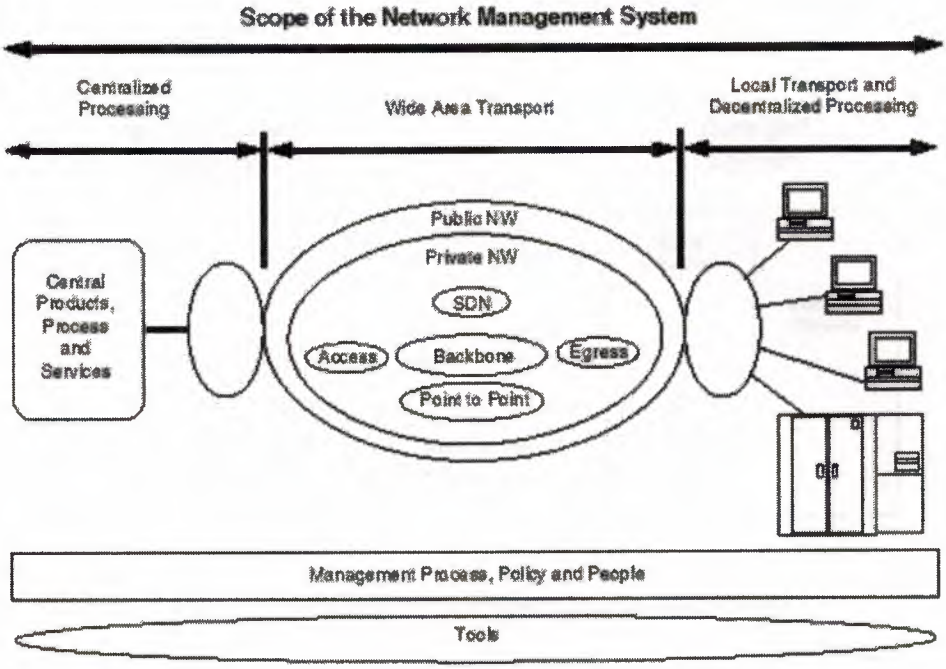


Figure 1.3 The scope of network management systems (Courtesy EDS)

1.4 Elements of a Network Management Architecture

The network management system, called the *manager/agent model*, consists of a manager, a managed system, a database of management information, and the network protocol (Figure 1.4).

The manager provides the interface between the human network manager and the devices being managed. It also provides the network management process. The management process performs tasks such as measuring traffic on a remote LAN segment or recording the transmission speed and physical address of a router's LAN interface.

As Figure 1.4 shows, the managed system consists of the agent process and the managed objects. The agent process performs network management operations such as setting configuration parameters and current operational statistics for a router on a given segment. The managed objects include workstations, servers, wiring hubs, communication

circuits, and so on. Associated with the managed objects are attributes, which may be statically defined (such as the speed of the interface), dynamic (such as entries in a routing table), or require ongoing measurement (such as the number of packets transmitted without errors in a given time period).

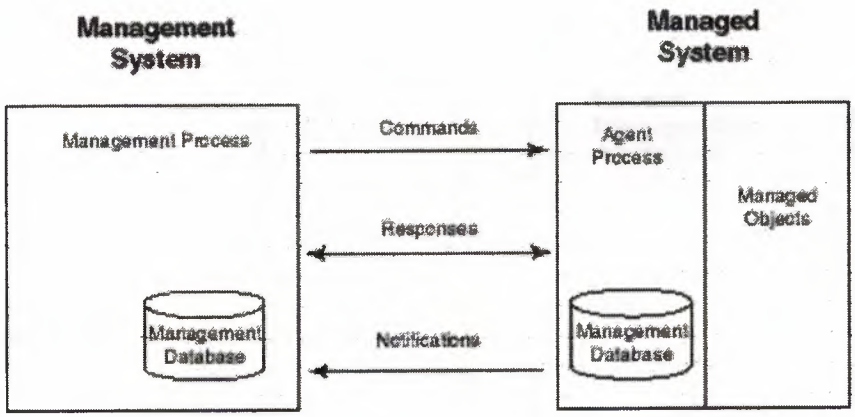


Figure 1.4 Network manager/agent relationships

A database of network management information, called the *management information base* (MIB), is associated with both the manager and the managed system. Just as a numerical database has a structure for storing and retrieving data, a MIB has a defined organization. This logical organization is called the *structure of management information* (SMI). The SMI is organized in a tree structure, beginning at the root, with branches that organize the managed objects by logical categories. The MIB represents the managed objects as leaves on the branches.

The network management protocol provides a way for the manager, the managed objects, and their agents to communicate. To structure the communication process, the protocol defines specific messages, referred to as commands, responses, and notifications. The manager uses these messages to request specific management information, and the agent uses them to respond. The building blocks of the messages are called *protocol data units* (PDUs). For example, a manager sends a GetRequest PDU to retrieve information, and the agent responds with a GetResponse PDU.

1.5 The OSI Network Management Architecture

The ISO/OSI model has been a benchmark for computer networking since it was first published in 1978. Figure 1.5 shows the familiar seven-layer structure. Following is a summary of the seven layers:

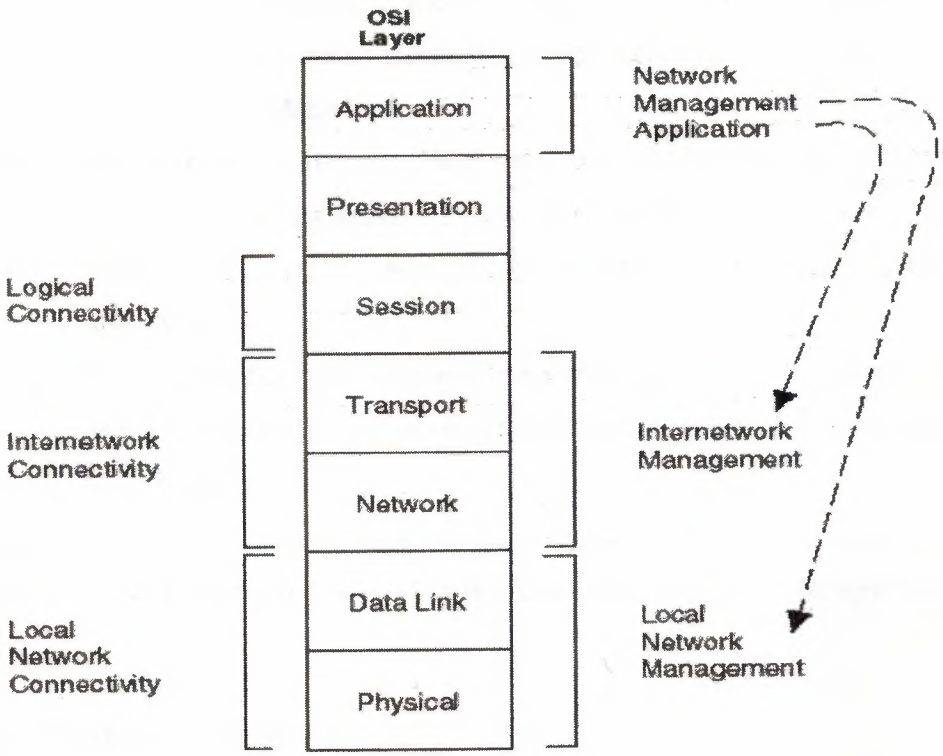


Figure 1.5 Network management within the OSI framework

Layer	Description
Physical	Provides the physical transmission medium for carrying the raw data, such as electrical or optical impulses, from one network node to the next.
Data Link	Provides reliable communications on the link; that is, it creates the channel between adjacent nodes on a LAN, MAN, or WAN. Functions include addressing, framing, and error control on the link.

Network	Provides communications functions for an internetwork. These include tasks such as the global addressing, routing, and switching that take data from its source to its destination via an internetwork of LANs, MANs, and WANs.
Transport	Assures the reliable end-to-end delivery of data. Its functions include error control and sequence control.
Session	Establishes the logical connection between end-user applications. These functions include mechanisms that synchronize the data transfer once a connection is established.
Presentation	Represents the application data so that it can be properly interpreted at the distant location. Examples of these functions include data compression/decompression, encryption, or ASCII to EBCDIC code conversion.
Application	Includes the functions responsible for end-user applications, such as file transfer, electronic mail, or remote terminal access. SNMP is an Application layer protocol.

OSI standards include a model of network management and a network management protocol.

1.5.1 The OSI Management Model

The manager/agent model includes a number of interactive components. The OSI network management framework defines the roles of those components. The organizational model uses a management domain. The domain may contain one or more management systems, managed systems, and subdomains. The managed system may, in turn, contain one or more managed objects. Each object is a network resource that one of the management systems may monitor and/or control.

An information model associated with the organizational model defines the structure of the management information and the management information base (MIB). It is a tree structure that groups objects sharing similar characteristics into classes. These objects are represented as an entry in the management information tree; each entry has defined attributes and values. The functional model defines five areas of network management used

for specific purposes. Figure 1.6 demonstrates how the various elements work together. This model relates the system management application process (SMAP) to the management information base (MIB) and the seven layers of the network management system. It defines interfaces for system management (the system management interface, or SMI) and layer management (the layer management interface, or LMI). The layer management functions are specific to a particular OSI layer entity. The model also specifies a protocol for manager/agent communication, known as the Common Management Information Protocol, or CMIP.

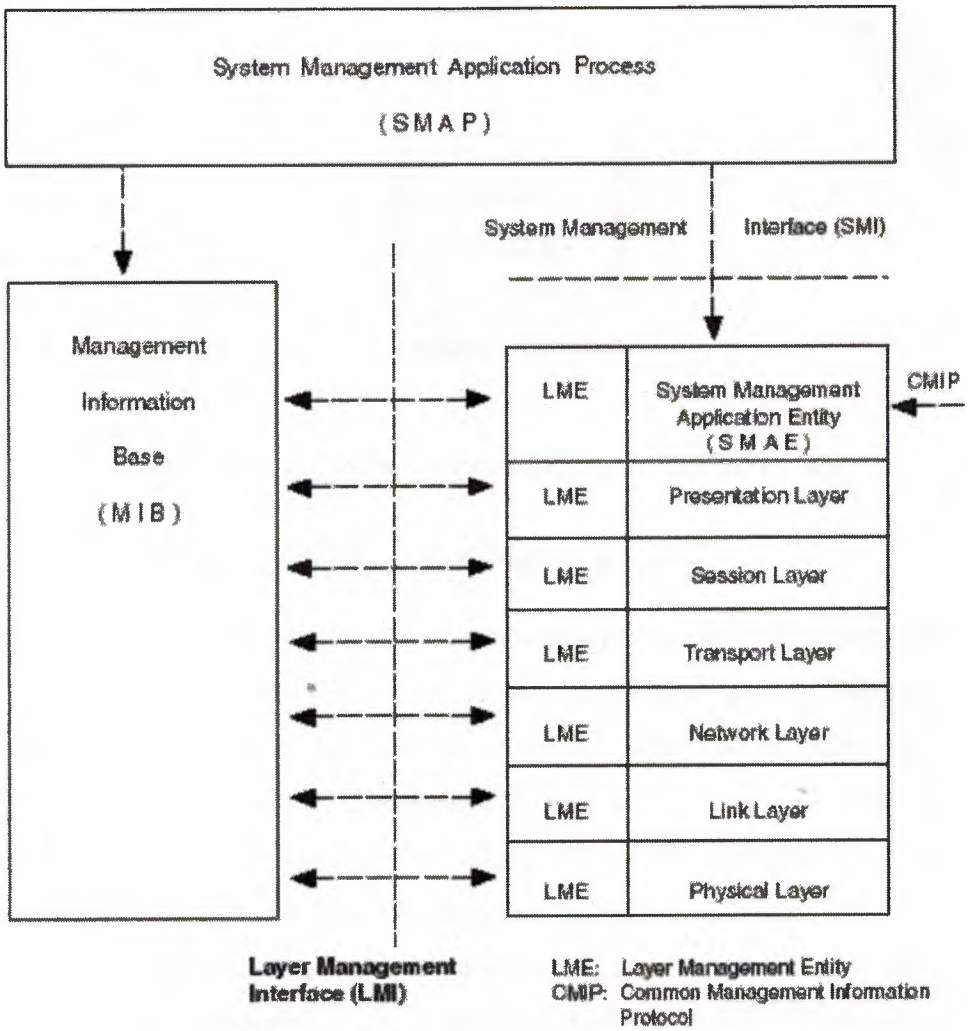


Figure 1.6 Architectural model of OSI management (©1988, IEEE)

1.5.2 OSI Specific Management Functional Areas (SMFAs)

The OSI management environment includes five areas of network management, which are called the OSI specific management functional areas (SMFAs) (Figure 1.7). These are fault management, accounting management, configuration management, performance management, and security management.

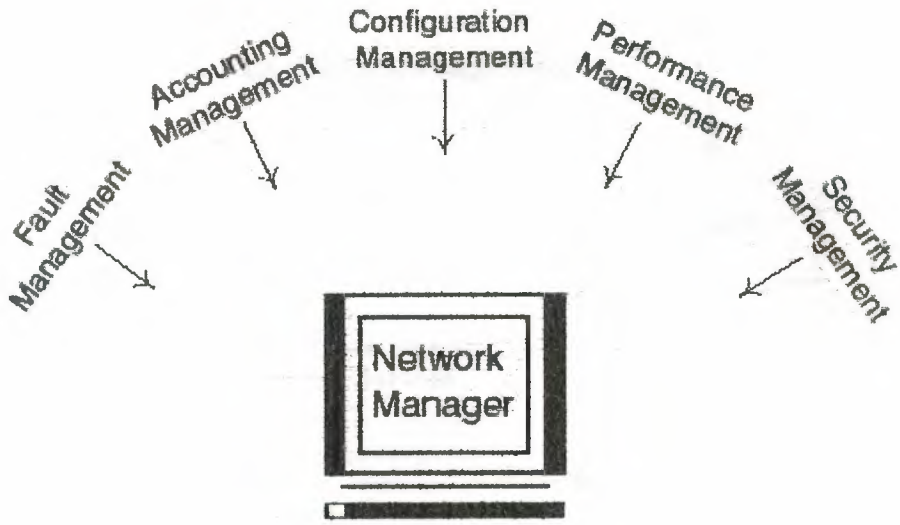


Figure 1.7 OSI network management functional areas

1.6 The IEEE Network Management Architecture

The Institute of Electrical and Electronics Engineers (IEEE) is perhaps best known for developing the 802 series of LAN standards. The IEEE Project 802 addresses the Physical and Data Link layers and extends into the higher layers of the architecture where appropriate. The IEEE LAN/MAN management standard uses ISO's CMIP, which was discussed in Section 1.5, to extend into the higher layer. This architecture includes three elements (Figure 1.8): the LAN/MAN Management Service (LMMS), the LAN/MAN Management Protocol Entity (LMMPE), and the Convergence Protocol Entity (CPE). The LMMS defines the management service available to the LAN/MAN Management User (LMMU). The LMMPE communicates management information via protocol exchanges. LMMS and LMMPE use the ISO CMIS and CMIP standards and

enable two LMMUs to exchange management information. The CPE allows LAN/MAN environments to provide LMMS. The CPE adds functions of reliable and sequential data delivery on top of the unacknowledged connectionless service provided by the IEEE 802.2 Logical Link Control (LLC) layer. The unacknowledged connectionless service is known as LLC Type 1.

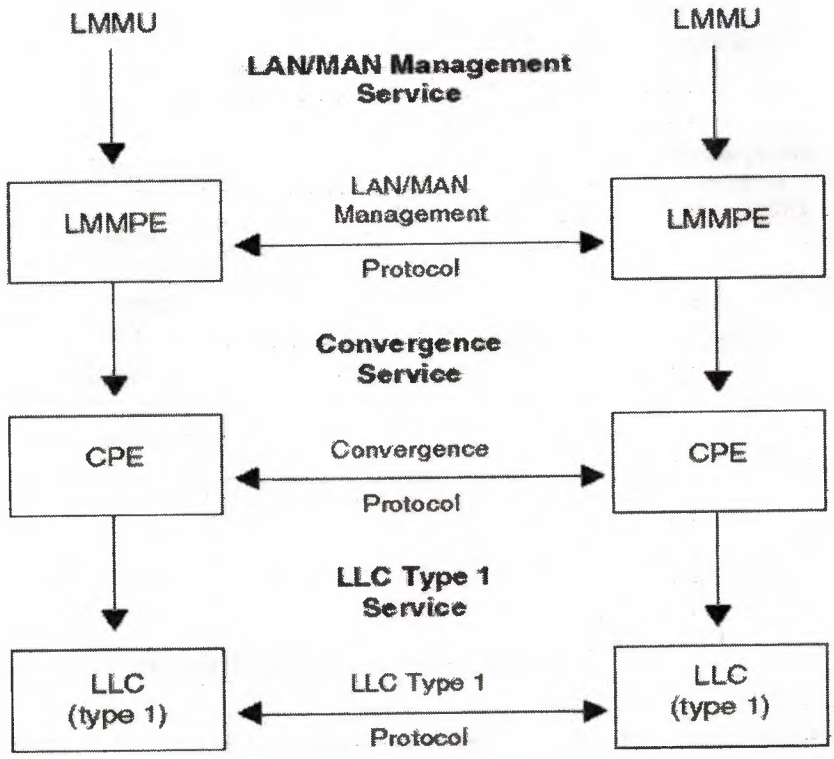


Figure 1.8 LAN/MAN management communication architecture (©1992, IEEE)

Figure 1.9 compares the IEEE architecture with the OSI model. The complexity of the two protocol stacks varies significantly. While CMIP uses all seven layers of the ISO model, the IEEE model runs CMIP and the CPE directly over the LLC layer—hence the acronym CMOL, which stands for CMIP over LLC. Because LLC provides connectionless service to the management application, some of the Association Control Service Element (ACSE) functions in the full CMIP stack are unnecessary. The CPE fills in and performs some, but not all, of the Network through Presentation layer functions. The benefit of the reduced CMOL stack is that it minimizes the memory requirements for agents. The

disadvantage is that you cannot route CMOL across internetworks because it lacks Network layer functionality. This is not surprising, since CMOL was designed from a LAN and not an internetwork perspective.

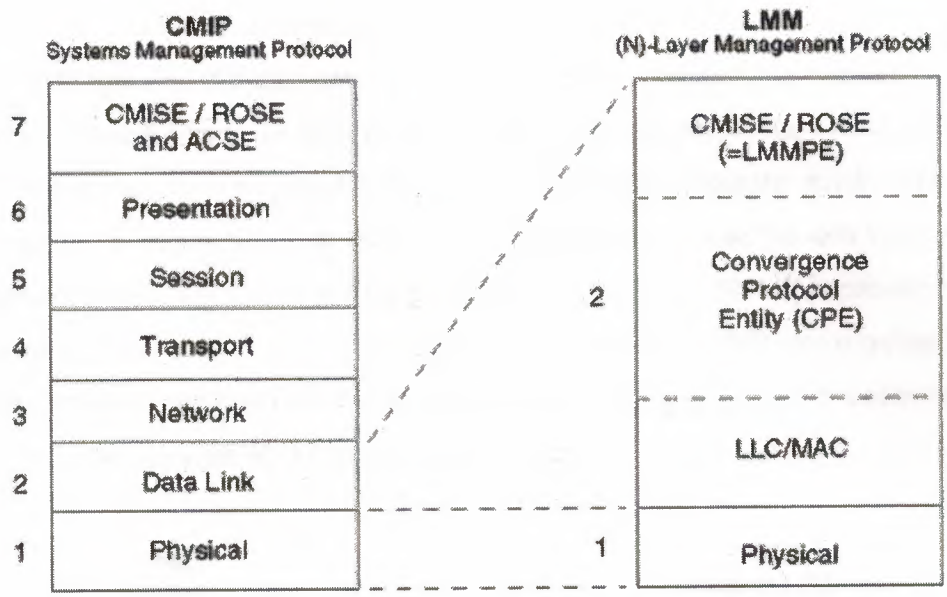


Figure 1.9 Comparing CMIP and LMMP protocol stacks (©1992, IEEE)

1.7 The Internet Network Management Framework

The Internet Activities Board (IAB) decided to take a two-step approach to Internet management. Enhancements to the SGMP, which became known as the Simple Network Management Protocol (SNMP), would provide a short-term solution. The long-term solution would be based on the CMIS/CMIP architecture, and was called CMOT (CMIP over TCP/IP).

1.7.1 SNMP, the Simple Network Management Protocol

SNMP is based on the manager/agent model. SNMP is referred to as "simple" because the agent requires minimal software. Most of the processing power and data storage resides on the management system, while a complementary subset of those functions resides on the managed system. To achieve its goal of being simple, SNMP includes a limited set of management commands and responses (Figure 1.10). The

management system issues Get, GetNext, and Set messages to retrieve single or multiple object variables or to establish the value of a single variable. The managed system sends a Response message to complete the Get, GetNext, or Set. The managed system sends an event notification, called a trap, to the management system to identify the occurrence of conditions such as a threshold that exceeds a predetermined value.

SNMP assumes that the communication path is a connectionless communication subnetwork. In other words, no prearranged communication path is established prior to the transmission of data. As a result, SNMP makes no guarantees about the reliable delivery of the data; however, in practice most messages get through, and those that don't can be retransmitted. Reviewing Figure 1.10, the primary protocols that SNMP implements are the User Datagram Protocol (UDP) and the Internet Protocol (IP). SNMP also requires Data Link layer protocols, such as Ethernet or token ring, to implement the communication channel from the management to the managed system.

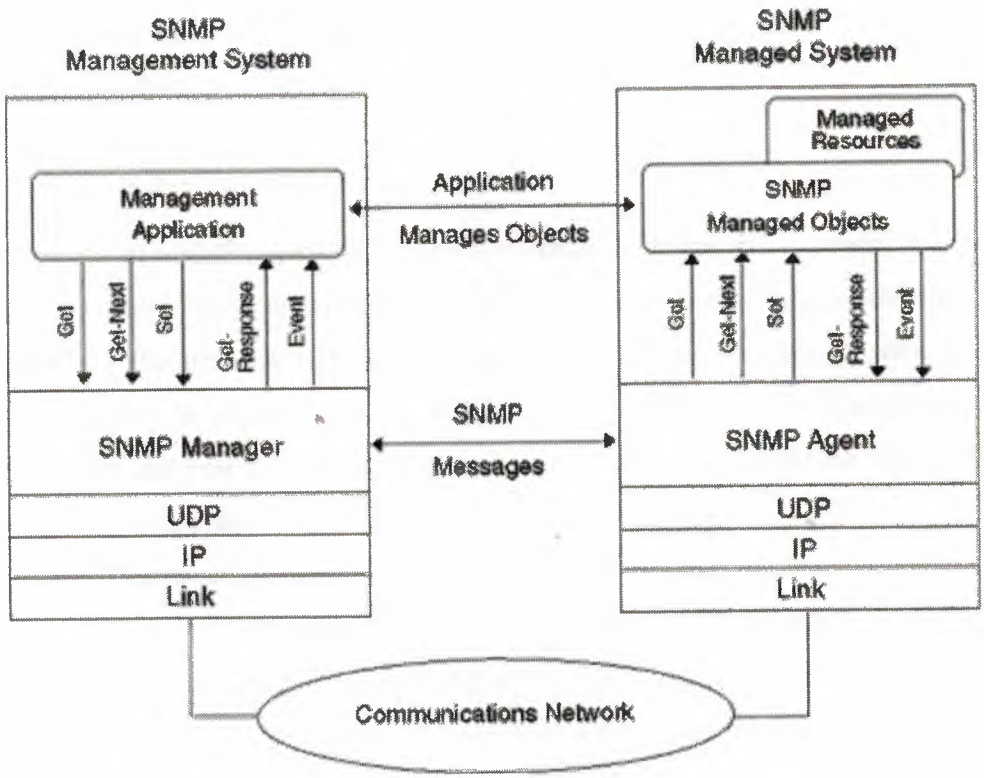


Figure 1.10 SNMP architecture (©1990, IEEE)

SNMP's simplicity and connectionless communication also produce a degree of robustness. Neither the manager nor the agent relies on the other for its operation. Thus, a manager may continue to function even if a remote agent fails. When the agent resumes functioning, it can send a trap to the manager, notifying it of its change in operational status.

1.7.2 CMIP over TCP/IP (CMOT)

Architecturally, CMOT fits the manager/agent paradigm (Figure 1.11). Unlike SNMP, which provides connectionless service using UDP/IP, however, CMOT uses an association-oriented communication mechanism and the TCP/IP protocol to assure reliable transport of data. To guarantee reliable transport, CMOT systems establish Application layer connections prior to transmitting management information. CMOT's Application layer services are built on three OSI services: the Common Management Information Service Element (CMISE), the Remote Operation Service Element (ROSE), and the Association Control Service Element (ACSE). A Lightweight Presentation Protocol (LPP) provides Presentation layer services.

1.8 Supporting SNMP: Agents

The use of SNMP agents within internetworking devices has increased dramatically in the last few years. There are five general categories of devices in which you'll find agents: wiring hubs; network servers and their associated operating systems; network interface cards and the associated hosts; internetworking devices, such as bridges and routers; and test equipment, such as network monitors and analyzers. Other devices, such as uninterruptible power supplies, have also become SNMP compatible.

Each of these categories makes a significant contribution to the overall network management scheme. Thus, network administrators who practice proactive network management should seriously consider using network devices that have these imbedded agents. In conclusion, we can find SNMP agents in almost every internetworking device.

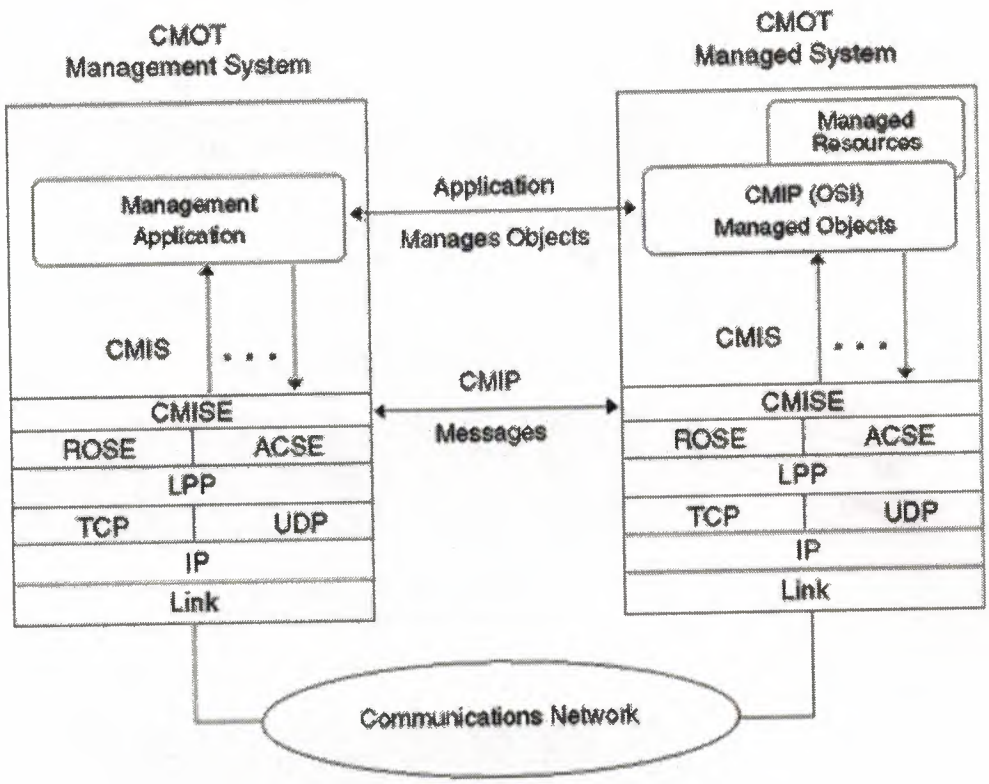


Figure 1.11 CMOT architecture (©1990, IEEE)

Some vendor implementations are better than others. As a result not all of these agents are interoperable.

1.9 Desktop Management Task Force

The Desktop Management Interface (DMI) technology is the management architecture developed by the DMTF (Figure 1.12). The focus of the DMI is on desktop and LAN management, independent of the system, operating system, or network operating system. DMI is designed to be integrated with all network management protocols and consoles, such as SNMP or CMIP. The DMI architecture is divided into three layers: the Management Applications Layer, which interfaces with various agents; the Service Layer, which includes the Management Information File (MIF) database; and the Hardware/Software Components Layer, which interfaces with the actual components being managed.

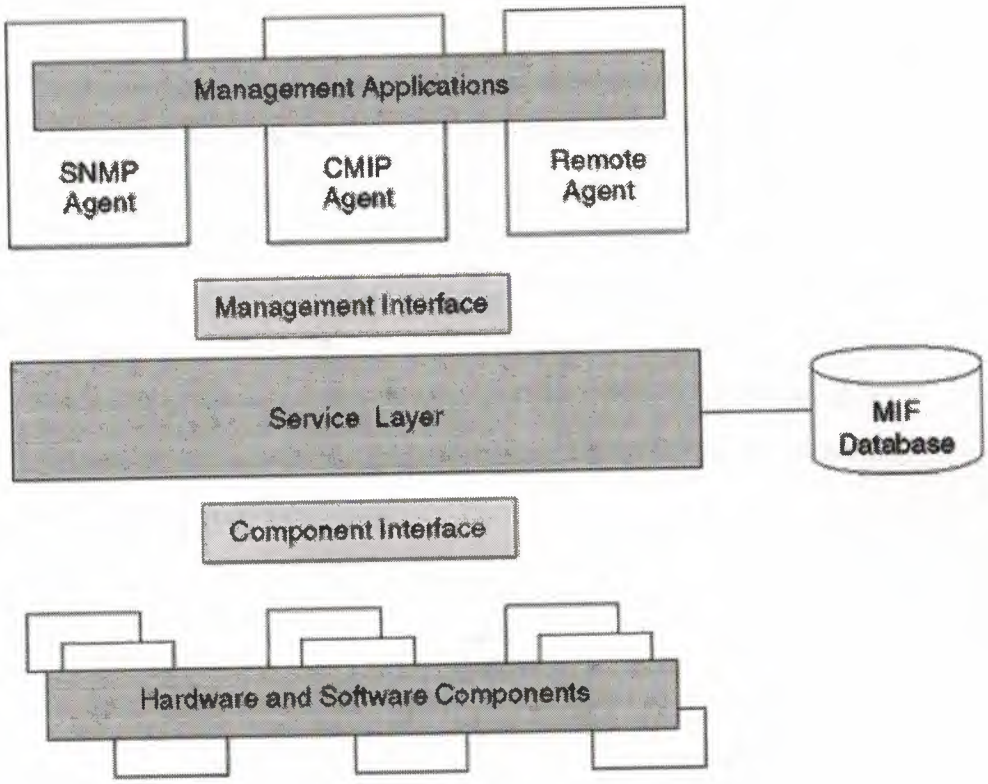


Figure 1.12 Desktop Management Interface (DMI)

1.10 Web-based Network Management

One of the most common interfaces that has evolved in recent years is the World Wide Web, or simply the Web. Web-based systems consist of a server that stores "pages" of information that are typically formatted using the Hypertext Markup Language, or HTML. The client accesses the information using software called a Web browser, which may have integrated capabilities for printing, file retrieval and storage, email, and so on. The communication protocol between the server and client is the Hypertext Transfer Protocol, or HTTP, which is a transaction-oriented protocol that makes use of the Transmission Control Protocol (TCP). One of the advantages of this architecture is its platform independence, as Web browsers from a number of client platforms, including Macintosh, Windows, UNIX, and other workstations, can access the Web server in a

similar manner. Web-based traffic now consumes a large portion of the traffic on the Internet.

The popularity of these Web-based systems has created another application for this technology—storing network management information on a Web server so that it can be accessed and disseminated to distributed users in a platform-independent fashion. Web-based network management can take on one of several forms (Figure 1.13):

- Web-enabled agents that can be managed through a browser using the HyperText Transfer Protocol (HTTP) for communication.
- Web-enabled managers, which may include a Web server front end to an existing platform, or a stand-alone manager running on a Web server, either of which may use HTTP for communication.

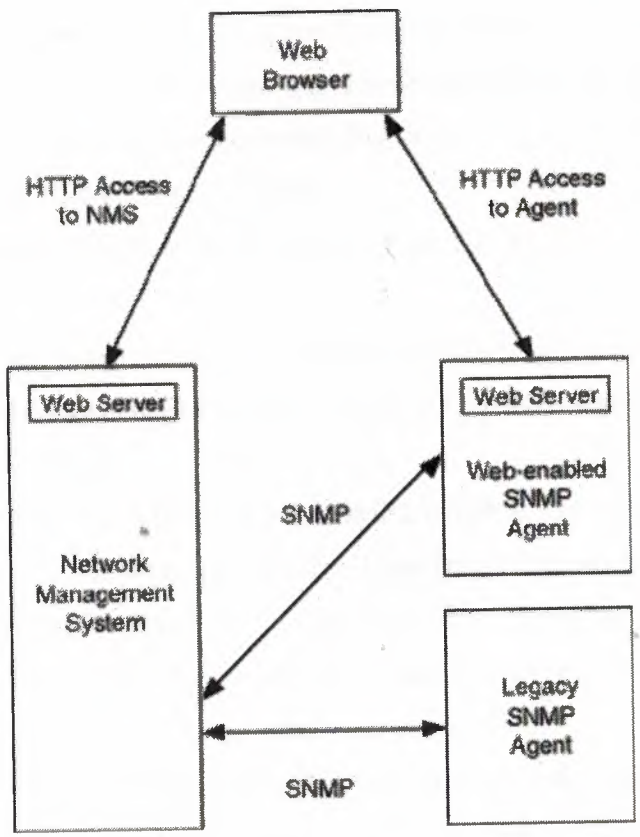


Figure 1.13 Web-based management architecture

In addition, there are two standardization efforts underway in this area:

- The Web-based Enterprise Management (WBEM) proposal, from a consortium of vendors which include Microsoft, Compaq Computer, Cisco Systems, and many others.
- The Java Management Application Programming Interface (JMAPI) proposal from SunSoft.

In any event, however, SNMP still enters into the equation, either from the perspective of communication with existing (legacy) SNMP agents and/or managers, or the need to provide technical functionality that other solutions do not adequately cover.

1.10.1 Web-based Enterprise Management

The Web-Based Enterprise Management (WBEM) initiative was launched by vendors BMC Software, Cisco Systems, Compaq Computer, Intel, and Microsoft to address the challenge of distributed networks using emerging Web-based technologies. Other goals included the integration of network, systems, and application management; platform and management environment independence; scalability to grow as networks expand; plus leveraging the low cost of Web-enabled clients.

The WBEM proposal consists of several elements (Figure 1.14):

- Hypermedia Management Schema (HMMS), an extensible data model which can be used to describe the managed objects. The DMTF was chartered with further defining of the HMMS.
- HyperMedia Management Protocol (HMMP), which is a communication protocol that embodies HMMS and runs over the HyperText Transport Protocol (HTTP), with interfaces to SNMP and DMI in the future. The HMMP allows the aggregated data to be queried across the network and shared among top-level applications. The IETF was chartered with further refinement of the HMMP.
- HyperMedia Managed Object (HMMO) is a managed entity, containing at least one URL, that contains data that can be managed by a client browser, either directly or through some type of management schema.
- HyperMedia Object Manager (HMOM) is a generic definition for management applications that combines information from multiple sources and uses a

communication protocol to present that information to the client (browser) using the HyperText Markup Language (HTML). It is anticipated that the HMOM could be implemented using a number of development platforms, such as Java, Active X, Common Gateway Interface (CGI), the Common Object Request Broker Architecture (CORBA), and others.

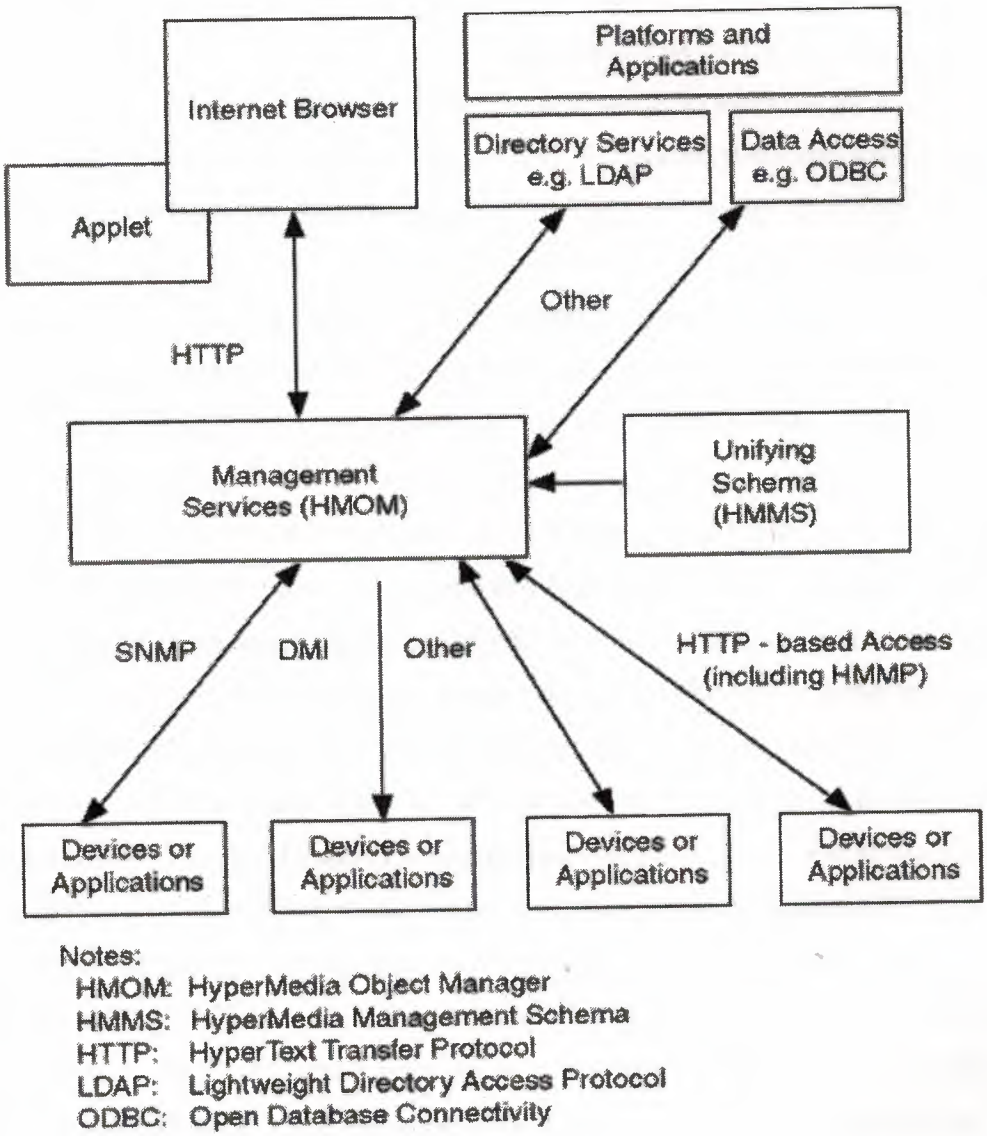


Figure 1.14 WBEM architecture

1.10.2 Java Management API

SunSoft's Java is a simple yet robust object-oriented programming language that has been implemented across a wide variety of platforms and operating systems. The Java Management API (JMAPI) is a set of objects and development tools for creating network management solutions that can be utilized by a wide variety of heterogeneous networks. Thus, the JMAPI leverages the platform-independence of the Java computing environment, extending Java's "write once, run everywhere" capabilities to the traditionally proprietary architectures of network management systems and consoles. In addition, the JMAPI allows for the integration of SNMP agent information into the Java environment, thus leveraging classic network management solutions with the emerging technology of Web-based network management. The JMAPI consists of three functional components: a Browser User Interface, an Admin Runtime Module, and Appliances (Figure 1.15).

The Browser User Interface (BUI) is the means by which the network administrator issues the management queries and commands. The BUI requires a Java-enabled Web browser that has the capabilities to run Java applets. *Applets* are Java programs that can be included in a HyperText Markup Language (HTML) page, in much the same way that graphics, such as .GIF files, may be included in a page. When a Java-compatible browser views a page containing an applet, the applet code is transferred to, and executed on, the browser. The BUI uses the HyperText Transfer Protocol (HTTP) for communication with an HTTP server within the Admin Runtime Module, which loads the initial Java applet and JMAPI objects. Other communication across machine boundaries uses the Remote Method Invocation (RMI). The JMAPI applet consists of the Administrative View Module (AVM), which provides a set of building blocks for user interface and application-level functionality. The Managed Object Interfaces perform remote management functions. The Admin Runtime Module (ARM) is the focus of the administration efforts; it consists of several elements. The HTTP Server provides bootstrap services for the Java elements. After the Java applets take control, the Managed Objects Interfaces in the BUI provide the communication link to the ARM. The Managed Object Factory implements the management operations and interacts with the Agent Object Interfaces and the Managed Data Interfaces. The Managed Data Interfaces access a relational database through the Java

Database Connectivity (JDBC) Interface, which provides the repository of management information.

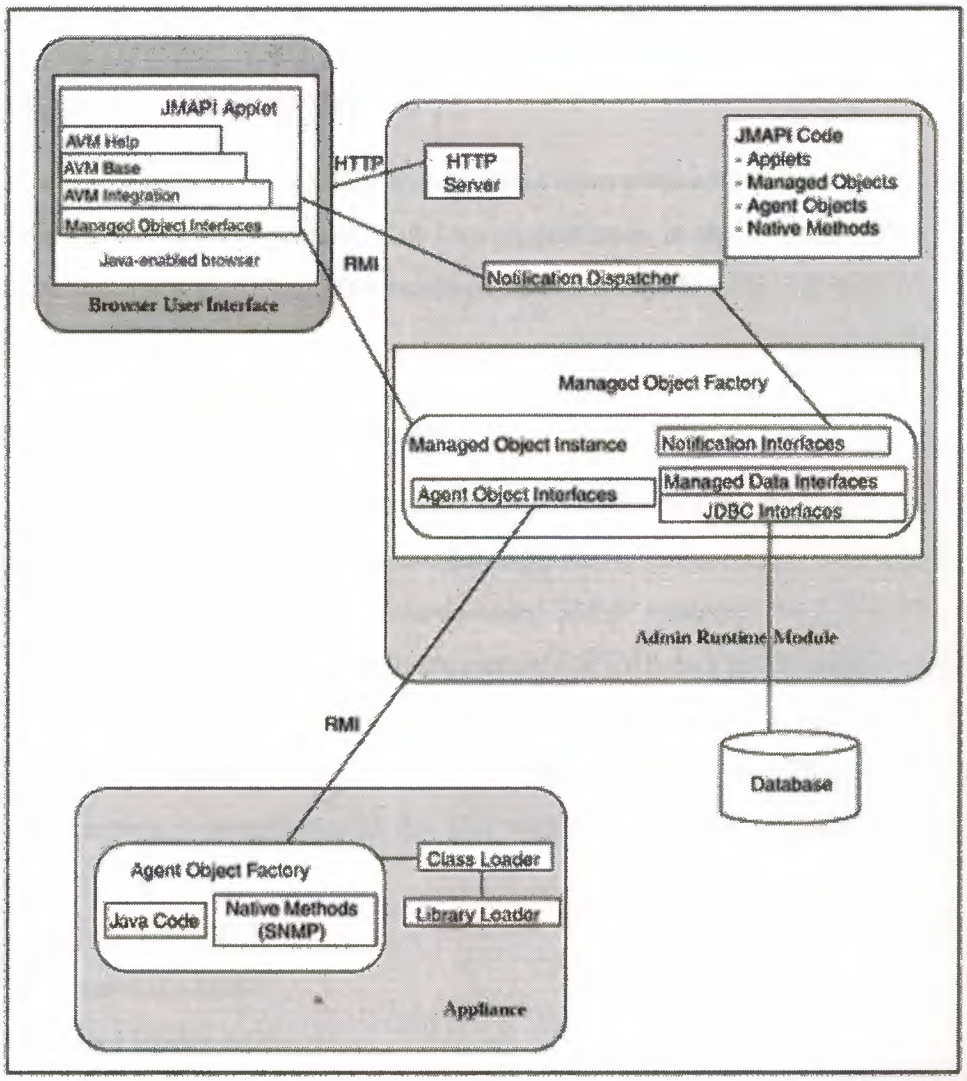


Figure 1.15 Java Management API architecture components (Copyright 1996, Sun Microsystems, Inc.)

The Appliances are the devices being managed by the Admin Runtime Module. An Appliance contains an Agent Object Factory which creates and maintains instances of agent objects. When the objects are invoked, they may download Java code to implement the

management operations. Integration with SNMP agents, which implements the protocol and handles traps, has also been designed into the system.

Thus, the Java Management API provides the tools for developing network and service management systems that can operate across a diversity of systems and platforms.

1.11 Supporting SNMP: Managers

We can describe management architectures from prominent vendors that support SNMP-based network management. This section discusses, in alphabetical order, offerings from Asante Technologies, Inc., Hewlett-Packard Company, Novell, SunSoft, Inc., and Tivoli Systems.

1.11.1 Asante Technologies' IntraSpection

Asante Technologies Inc.'s IntraSpection is the first SNMP management product based entirely on Intranet technology, a technology that is being widely and rapidly adopted. IntraSpection is an open, standards-based SNMP management platform that runs on a Windows NT Web server and delivers standard SNMP data graphically to any Java-enabled Web browser. Thus, IntraSpection provides network management capabilities for your entire network anytime, anywhere you have access to the World Wide Web (Figure 1.16). IntraSpection is compliant with the following SNMP-based management standards:

- MIB II
- Standard Repeater MIB
- Ethernet-Like MIB
- Standard Bridge MIB

IntraSpection is comprised of five software modules. The Map Manager builds a topology diagram of the network. The Device Manager graphically represents each network element. The Trap Manager gathers device statistics and stores that information on a third-party database that is running on the same server. A HyperText Transfer Protocol (HTTP) module can be used to turn the IntraSpection server into a Web server. Finally, the Common Gateway Interface (CGI) module translates the HyperText Markup Language (HTML) to/from SNMP commands/responses.

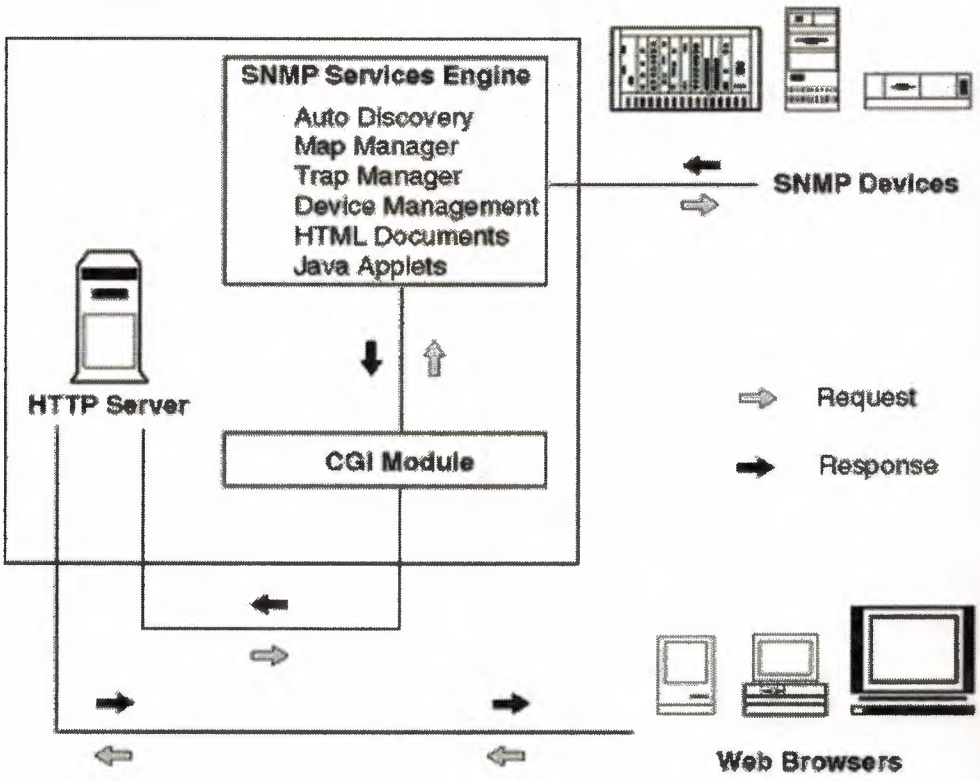


Figure 1.16 Asante Technologies' IntraSpection architecture (Courtesy of Asante Technologies, Inc.)

1.11.2 Cabletron Systems' SPECTRUM

Cabletron Systems' SPECTRUM Enterprise Manager is designed as an open system to be implemented in multivendor environments. The architecture is based on a client/server paradigm, with various interfaces to other systems. SPECTRUM consists of two principal elements. A graphical user interface (GUI), called SpectroGRAPH, provides a Motif-based interface for the end user. The management server, called SpectroSERVER, consists of two sections. The Virtual Network Machine (VNM) creates models of the various network entities, such as cables or network devices. The Device Communication Manager (DCM) is a multiprotocol communications engine with protocol support for SNMP, IEEE 802.1, and ICMP/PING commands, and with future support planned for CMIP as well as extensions (via a tool kit) for any proprietary protocol (Figure 1.17).

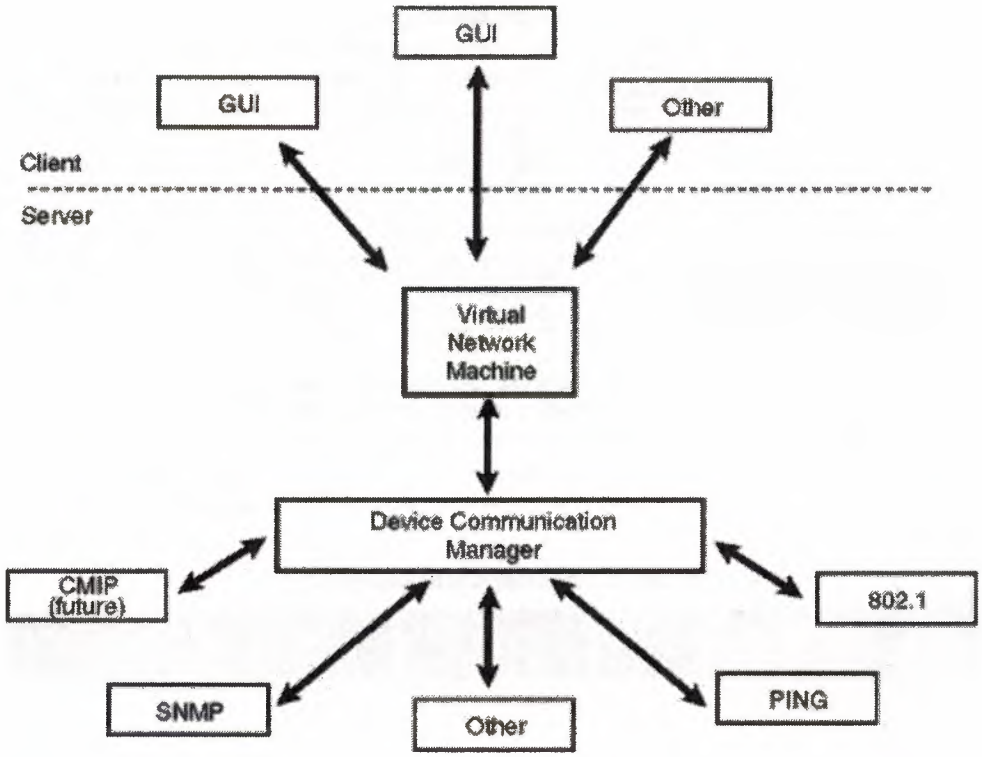


Figure 1.17 Cabletron Systems' SPECTRUM architecture (Cabletron Systems)

1.11.3 Hewlett-Packard OpenView

The Hewlett-Packard OpenView family provides an integrated network and systems management solution for end-to-end service management of the complete information technology environment. Solutions consist of a broad portfolio of management products from HP and OpenView Solutions Partners, and a complete set of services that help customers improve service and reduce operations cost (Figure 1.18).

1.11.4 Novell's ManageWise

Novell's ManageWise is a comprehensive, integrated management solution that lets you successfully manage and optimize a heterogeneous network. It reduces the cost of owning and managing a network and enhances business operations by increasing network reliability and user productivity (Figure 1.19).

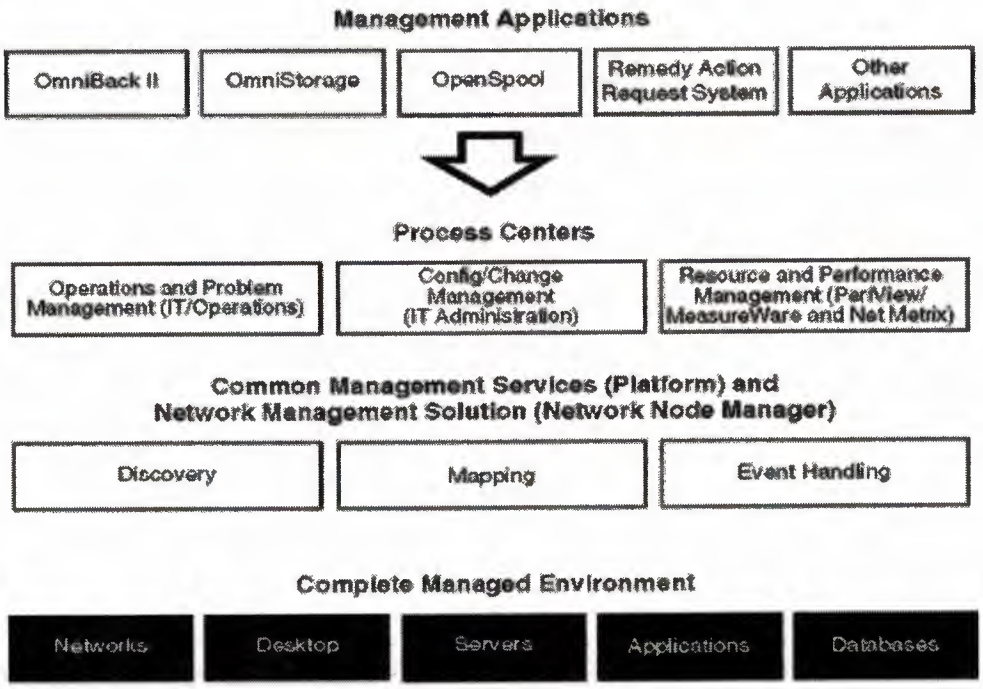


Figure 1.18 Hewlett-Packard OpenView solution framework (Hewlett-Packard)

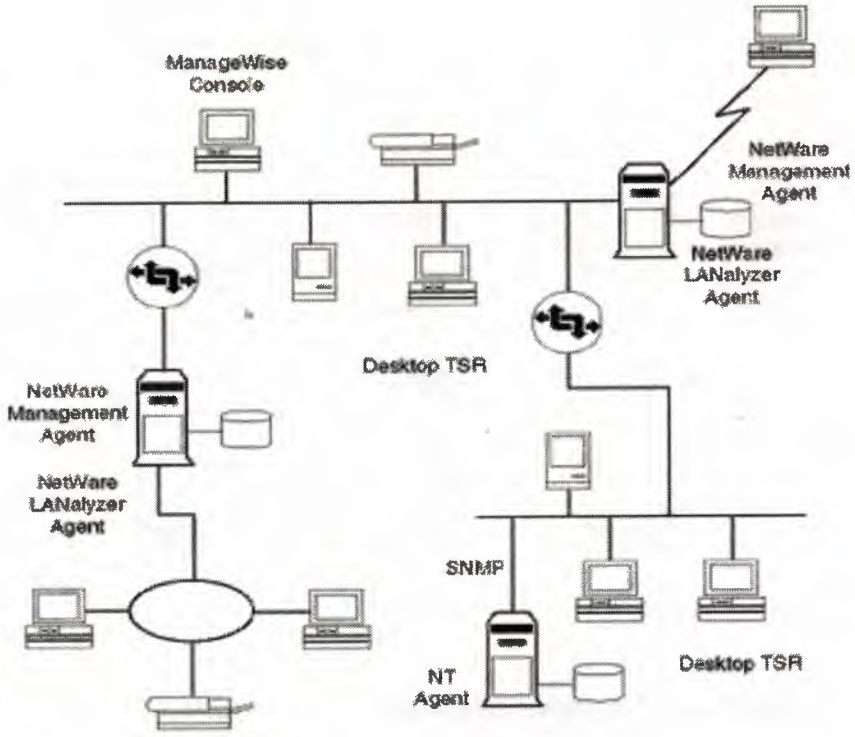


Figure 1.19 Novell's ManageWise solution framework (Courtesy of Novell, Inc.)

ManageWise lets you proactively manage an entire network through NetWare and Windows NT server management, desktop management, network traffic analysis, automated network inventory, remote control, virus protection, and software management. In addition, ManageWise continuously works in the background to monitor network activity and trend performance. As a result, we can easily plan for future network changes and growth, predict future bottlenecks, and plan for resegmentation of the network before problems occur:

1.11.5 Sun Microsystems' Solstice Domain Manager

Sun Microsystems' network management product family includes Solstice Site Manager (SM), Solstice Domain Manager (DM), and Solstice Enterprise Manager (EM) (Figure 1.20).

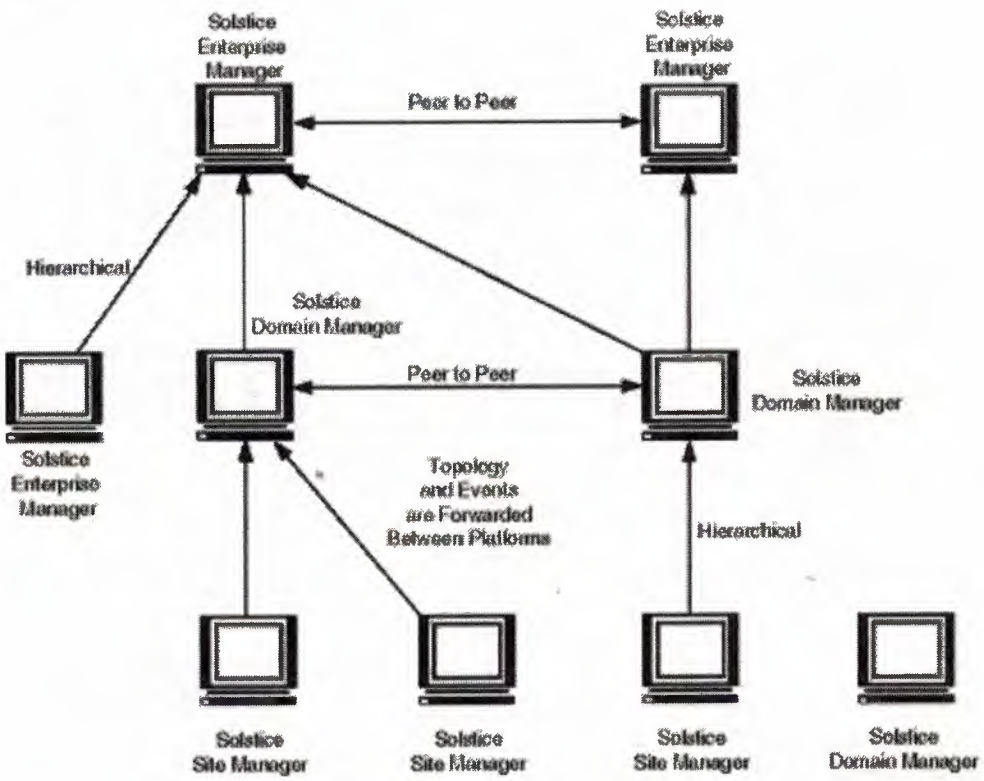


Figure 1.20 Sun Microsystems' Solstice Enterprise management architecture (Courtesy of Sun Microsystems, Inc.)

Solstice Domain Manager is designed to meet the requirements of larger or multisite environments. Key features of the Domain Manager include: event management, including event-based actions, scheduled requests, and alarm reports; and user tools, including the console, topological map, link management, as well as discover, layout, browser, and grapher tools. Domain Managers includes a number of integrated SNMP features, including: the Proxy Agent, Trap Daemon to translate and forward traps, the mib2schema utility for MIB translation, and support for the protocol operations enhancements for SNMPv2.

1.11.6 Tivoli Systems' TME 10 NetView

TME 10 NetView breaks down the traditional barriers between network management and systems management by providing tight integration with Tivoli's complimentary TME 10 management applications.

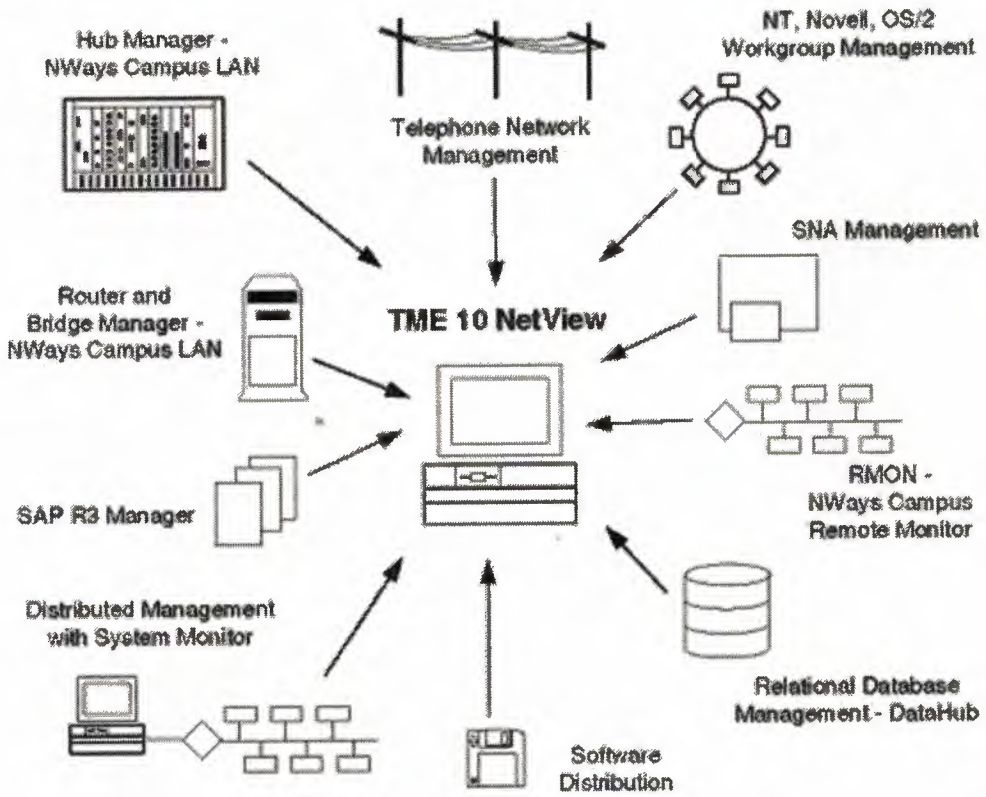


Figure 1.21 Tivoli and IBM application integration (Courtesy of Tivoli Systems)

Combined with its ability to easily effect changes on many devices, a global support infrastructure, and the backing of hundreds of third-party vendors, Tivoli's TME 10 NetView is a widely implemented management platform (Figure 1.21). Further, TME 10 NetView not only enables you to manage your network, it also positions you for planned and future growth with a complete systems management solution.

1.12 Fitting SNMP into the Role of Network Management

SNMP is a protocol that communicates network management information. Therefore, SNMP fits into the Application layer of the OSI model. But if we only look at SNMP in this context, we are ignoring the structure that supports it—and that fills out the remaining layers of the OSI model. In order to study SNMP in detail, you need to thoroughly understand the supporting structures.

CHAPTER TWO

THE STRUCTURE OF MANAGEMENT INFORMATION

In this chapter, we will learn about the structure of management information (SMI), which defines the rules for identifying managed objects.

2.1 Managing Management Information

In the manager/agent paradigm for network management, managed network objects must be physically and logically accessible. The term *physically accessible* means that some entity must physically check the address, count the packets, or otherwise quantify the network management information. Logical accessibility means that management information must be stored somewhere and, therefore, that the information must be retrievable and modifiable. (SNMP actually performs the retrieval and modification.) The structure of management information (SMI) organizes, names, and describes information so that logical access can occur.

The SMI states that each managed object must have a name, a syntax, and an encoding. The *name,* an object identifier (OID), uniquely identifies the object. The *syntax* defines the data type, such as an integer or a string of octets. The *encoding* describes how the information associated with the managed objects is serialized for transmission between machines.

2.2 Presenting Management Information

In terms of the ISO/OSI model, the ASN.1 syntax is a Presentation-layer (layer 6) function. The Presentation layer defines the format of the data stored within a host computer system. In order for managers and agents to exchange data, both must understand it, regardless of the way either machine represents data internally. For this to occur, two items must be standardized: the abstract syntax and the transfer syntax. The *abstract syntax* defines specifications for data notation. The *transfer syntax* defines (transmittable) encodings for the elements in the abstract syntax.

The Internet SMI specifies that ASN.1 (Abstract Syntax Notation One) define the abstract syntax for messages; that is, ASN.1 defines the basic language elements and

provides rules for combining elements into messages. The Basic Encoding Rules (BER) provide the transfer syntax. The BER are associated with the abstract syntax and provide bit-level communication between machines. Thus the SMI and SNMP use the ASN.1 formalizations to define various aspects of the Internet network management framework.

2.3 ASN.1 Elements

Abstract Syntax Notation One (ASN.1) is designed to define structured information (messages) in a machine-independent (or host-independent) fashion. To do this, ASN.1 defines basic data types, such as integers and strings, and new data types that are based on combinations of the basic ones. The BER then define the way the data is serialized for transmission. ASN.1 defines *data* as a pattern of bits in computer memory, just as any high-level computer programming language defines data that the language manipulates as *variables*. The BER define a standard way to convert ASN.1 definitions into bit patterns for transmission, and then they actually transfer the data between computers. The BER are necessary because the ASN.1 description is "human-readable" and must be translated differently for each type of computer. The BER representation, however, is always the same for any ASN.1 description, regardless of the computers that send or receive that information. This assures communication between machines, regardless of their internal architecture. ASN.1 uses some unique terms to define its procedures, including *type* definitions, *value* assignments, *macro* definitions and evocations, and *module* definitions

2.3.1 Types and Values

A *type* is a class of data. It defines the data structure that the machine needs in order to understand and process information. The SMI defines three types: Primitive, Constructor, and Defined. ASN.1 defines several *Primitive types* (also known as Simple types), including INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL. By convention, types begin with an uppercase letter. (ASN.1 also defines the four types listed here as reserved character sequences, and therefore represents them entirely in uppercase.) *Constructor types* (also known as Aggregate types) generate lists and tables. *Defined types* are alternate names for either simple or complex ASN.1 types and are usually more descriptive. Examples of SNMP-defined types include IpAddress, which represents a 32-bit

Internet address, and TimeTicks, which is a time-stamp.

The *value* quantifies the type. In other words, once we know the type, such as INTEGER or OCTET STRING, the value provides a specific instance for that type. For example, a value could be an entry in a routing table. By convention, values begin with lowercase letters.

Some applications allow only a subset of the possible type values. A subtype specification indicates such a constraint. The subtype specification appears after the type and shows the permissible value or values, called the *subtype values*, in parentheses. For example, if an application uses an INTEGER type and the permissible values must fit within an 8-bit field, the possible range of values must be between 0 and 255. We would express this as:

INTEGER (0..255)

The two periods (..) are the range separator and indicate the validity of any integer value between 0 and 255.

2.3.2 Macros

A macro notation allows us to extend the ASN.1 language. By convention, a macro reference (or macro name) appears entirely in uppercase letters. For example, MIB definitions make extensive use of the ASN.1 macro, OBJECT-TYPE. The first object in MIB-II is a system description (sysDescr). RFC 1213 uses the OBJECT-TYPE macro to define sysDescr, as follows:

```
sysDescr OBJECT-TYPE
  SYNTAX  DisplayString (SIZE (0..255))
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
"A textual description of the entity. This value should
include the full name and version identification of the
system's hardware type, software operating-system, and
networking software. This must contain only printable ASCII
characters."
```

::= { system 1 }

Thus, one concise package defines the object sysDescr.

2.3.3 Modules

ASN.1 also collects descriptions into convenient groups, called *modules*. For example, the remote monitoring (RMON) MIB is a discrete unit that is also part of MIB-II. The module starts with a module name, such as RMON-MIB. Module names must begin with an uppercase letter. The BEGIN and END statements enclose the body of the module. The body may contain IMPORTS, which are the names of types, values, and macros, and the modules in which they are declared. In the following example, the first line after IMPORTS specifies that the Counter type, which will be used in this MIB module, is from another MIB module, RFC1155-SMI.

Following is the header section of the RMON MIB (from RFC 1757), which represents a MIB module. Comment lines within ASN.1 syntax begin with a double hyphen (--):

```
RMON-MIB DEFINITIONS ::= BEGIN
  IMPORTS
    Counter            FROM RFC1155-SMI
    DisplayString      FROM RFC1158-MIB
    mib-2              FROM RFC1213-MIB
    OBJECT-TYPE        FROM RFC-1212
    TRAP-TYPE          FROM RFC-1215;
-- Remote Network Monitoring MIB
  rmon   OBJECT IDENTIFIER ::= { mib-2 16 }

  -- textual conventions
       .
       .
       .
END
```

In the preceding example, we can see the OBJECT IDENTIFIER value notation for RMON. Note that the value of RMON is the sixteenth defined object under the mib-2 object tree. The curly brackets ({}) indicate the beginning and end of a list—in this case a list of the OBJECT IDENTIFIER values defining RMON.

2.4 Details of ASN.1—Objects and Types

This section focuses on the ASN.1 objects and data types used within the Internet Network Management framework.

2.4.1 Defining Objects in the MIBs

A MIB contains the objects to be managed. The OBJECT-TYPE macro defines these objects in a standard format that is consistent across various public and private MIBs. The MIB-II ASN.1 definitions (RFC 1213) appear as follows:

```
tcpInSegs OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    ::= { tcp 10 }
```

This ASN.1 definition means: This defines an object named tcpInSegs that contains Counter information. The Counter type is a nonnegative number that increases monotonically. This object is read-only and is mandatory for all managed devices that support its parent, mib-2.tcp. When a management protocol accesses this object, it uses the name { tcp 10 }, which identifies the tenth defined object within the tcp group.

2.4.2 Primitive (Simple) Types

To maintain SNMP's simplicity, the Internet SMI uses a subset of the ASN.1 data types. These are divided into two categories, the Primitive types and Constructor types. Primitive data types (also called Simple types) include INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL. The following examples come from MIB-II (RFC 1213).

INTEGER is a Primitive type with distinguished (or unique) values that are positive

and negative whole numbers, including zero. The INTEGER type has two special cases. The first is the *enumerated integer type*, in which the objects have a specific, nonzero number such as 1, 2, or 3. The second, the *integer-bitstring type*, is used for short bit strings such as (0..127) and displays the value in hexidecimal. An example of INTEGER would be:

```
ipDefaultTTL   OBJECT-TYPE
    SYNTAX    INTEGER
    ACCESS    read-write
    STATUS    mandatory
    DESCRIPTION
        "The default value inserted into the
        Time-To-Live field of the IP header of
        datagrams originating at this entity, whenever
        a TTL value is not supplied by the transport
        layer protocol."
    ::= { ip 2 }
```

The OCTET STRING is a Primitive type whose distinguished values are an ordered sequence of zero, one, or more octets. SNMP uses three special cases of the OCTET STRING type: the DisplayString, the octetBitstring, and the PhysAddress. In the DisplayString, all of the octets are printable ASCII characters. The octetBitstring is used for bit strings that exceed 32 bits in length. (TCP/IP frequently includes 32-bit fields. This quantity is a typical value for the internal word width of various processors—hosts and routers—within the Internet.) MIB-II defines the PhysAddress and uses it to represent media (or Physical layer) addresses.

An example of the use of a DisplayString would be:

```
sysContact    OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The textual identification of the contact
        person for this manage node, and information on
```

how to contact this person."

::= { system 4 }

Note that the subtype indicates that the permissible size of the DisplayString is between 0 and 255 octets.

The OBJECT IDENTIFIER is a type whose distinguishing values are the set of all object identifiers allocated according to the rules of ISO 8824-1. The ObjectName type, a special case that SNMP uses, is restricted to the object identifiers of the objects and subtrees within the MIB, as for example:

```
ipRouteInfo    OBJECT-TYPE
    SYNTAX   OBJECT IDENTIFIER
    ACCESS   read-only
    STATUS   mandatory
    DESCRIPTION
        "A reference to MIB definitions specific
        to the particular routing protocol responsible
        for this route, as determined by the value
        specified in the route's ipRouteProto
        value. If this information is not present, its
        value should be set to the OBJECT IDENTIFIER
        { 0 0 }, which is a syntactically
        valid object identifier, and any conforming
        implementation of ASN.1 and BER must be able to
        generate and recognize this value."
    ::= { ipRouteEntry 13 }
```

NULL is a type with a single value, also called *null*. The null serves as a placeholder, but is not currently used for SNMP objects. NULL is used as a placeholder in the variable bindings field of the SNMP GetRequest PDU. The NULL is assigned to be the value of the unknown variable, that is, the value the GetRequest PDU seeks.

2.4.3 Constructor (Structured) Types

The Constructor types, SEQUENCE and SEQUENCE OF, define tables and rows

(entries) within those tables. By convention, names for table objects end with the suffix *Table*, and names for rows end with the suffix *Entry*. The following discussion defines the Constructor types. The example comes from MIB-II.

SEQUENCE is a Constructor type defined by referencing a fixed, ordered, list of types. Some of the types may be optional, and all may be different ASN.1 types. Each value of the new type consists of an ordered list of values, one from each component type. The SEQUENCE as a whole defines a row within a table. Each entry in the SEQUENCE specifies a column within the row.

SEQUENCE OF is a Constructor type that is defined by referencing a single existing type; each value in the new type is an ordered list of zero, one, or more values of that existing type. Like SEQUENCE, SEQUENCE OF defines the rows in a table; unlike SEQUENCE, SEQUENCE OF only uses elements of the same ASN.1 type.

The TCP connection table that follows illustrates both the SEQUENCE and SEQUENCE OF:

```
tcpConnTable  OBJECT-TYPE
    SYNTAX  SEQUENCE OF TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "A table containing TCP connection-specific
        information."
    ::= { tcp 13 }
tcpConnEntry  OBJECT-TYPE
    SYNTAX  TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "Information about a particular current TCP
        connection. An object of this type is transient;
        it ceases to exist when (or soon after) the
        connection makes the transition to the CLOSED
```

```
        state."
    INDEX { tcpConnLocalAddress,
        tcpConnLocalPort,
        tcpConnRemAddress,
        tcpConnRemPort }
::= { tcpConnTable 1 }
TcpConnEntry ::=
SEQUENCE {
tcpConnState
  INTEGER,
tcpConnLocalAddress
  IpAddress,
tcpConnLocalPort
  INTEGER (0..65535),
tcpConnRemAddress
  IpAddress,
tcpConnRemPort
  INTEGER (0..65535)
}
```

The sequence name, TcpConnEntry, is the same as the row name, except that it begins with an uppercase letter. The INDEX clause defines the construction and order of the columns that make up the rows.

2.5 Encoding Rules

This section discusses the encoding rules that allow that information to be transmitted on a network. The Basic Encoding Rules (BER) define this transfer syntax, and ISO 8825-1 specifies it.

2.5.1 Encoding Management Information

Each machine in the management system can have its own internal representation of the management information. The ASN.1 syntax describes that information in a standard

form. The transfer syntax performs the bit-level communication (the external representation) between machines. For example, assume that the host needs management information from another device. The management application would generate an SNMP request, which the BER would encode and transmit on the network media. The destination machine would receive the information from the network, decode it using the BER rules, and interpret it as an SNMP command. The SNMP response would return in a similar, but reverse, manner. The encoding structure used for the external representation is called *Type-Length-Value encoding* (Figure 2.1).

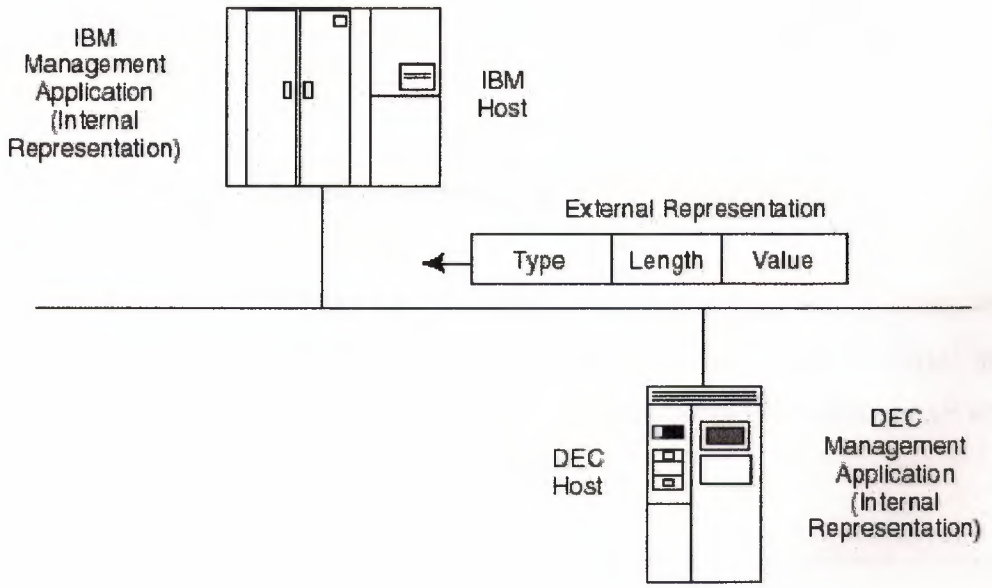


Figure 2.1 Internal and external data representations

2.5.2 Type-Length-Value Encoding

To define the external data representation, the BER first specify the position of each bit within the octets being transmitted. Each octet transmits the *most significant bit* (MSB) first and defines it as bit 8 on the left-hand side of the octet. The octet defines the *least significant bit* (LSB) as bit 1 on the right-hand side (Figure 2.2).

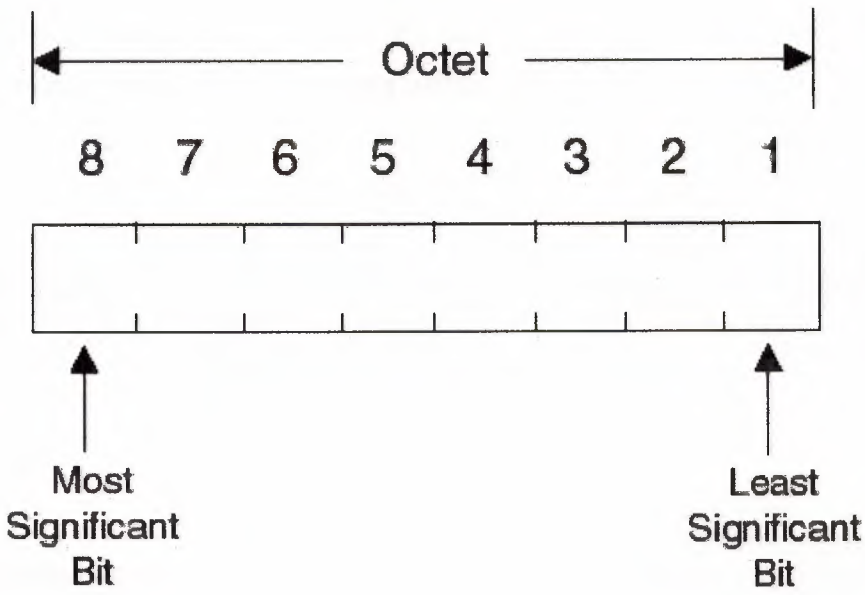


Figure 2.2 BER bit ordering, as defined in ISO 8825-1

The data encoding structure itself has three components: the Type, Length, and Value (TLV). Note that in the literature you will run across other names for Type-Length-Value, including Tag-Length-Value and Identifier-Length-Contents. The structure of a TLV encoding used with SNMP is shown in Figure 2.3.

Figure 2.3 Type-Length-Value (TLV) encoding

By defining the order and structure of the bits, the BER guarantee that both ends of the communication channel interpret the bit stream consistently.

2.6 Object Names

Each object, whether it's a device or a characteristic of a device, must have a name by which it can be uniquely identified. That name is the object identifier. It is written as a

sequence of integers separated by periods. For example, the sequence {1.3.6.1.2.1.1.1.0} specifies the system description, within the system group, of the mgmt subtree.

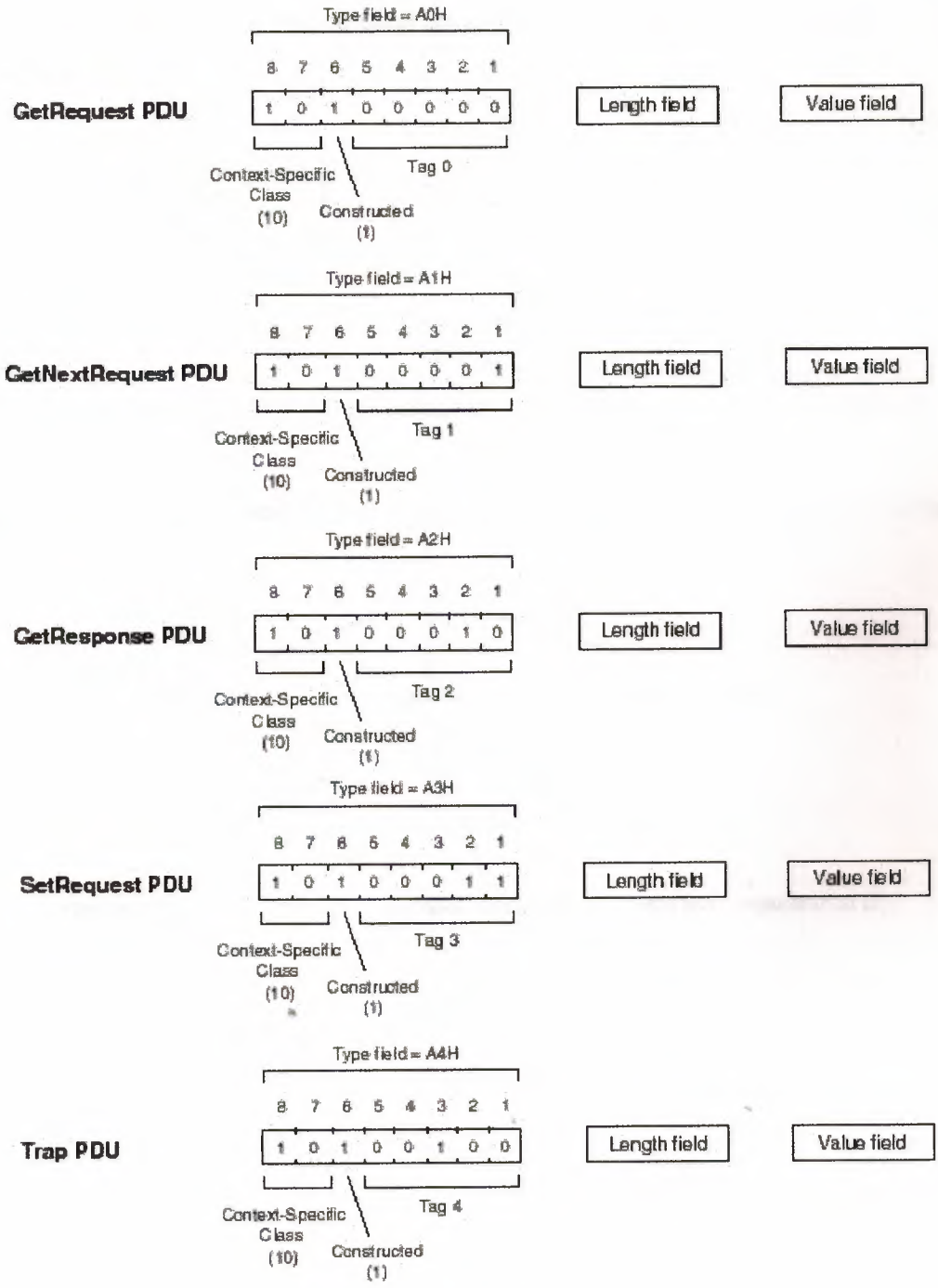


Figure 2.4 Encoding for the context-specific types used with SNMP

Annexes B, C, D and E of ISO 8824-1 define the numerical sequences; they resemble a tree with a root and several directly attached branches, referred to as *children* (Figure 2.5). These branches connect to other branches. We can use the structure of root, branches, subbranches, and leaves to diagram all of the objects within a particular MIB and their relationships.

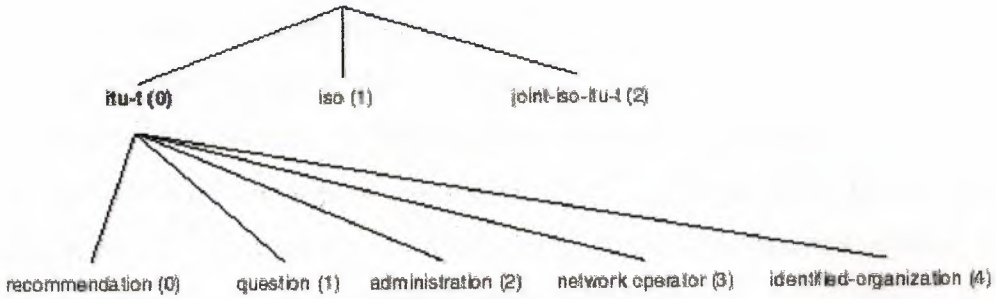


Figure 2.5 The root node and ITU-T-assigned OBJECT IDENTIFIER component values

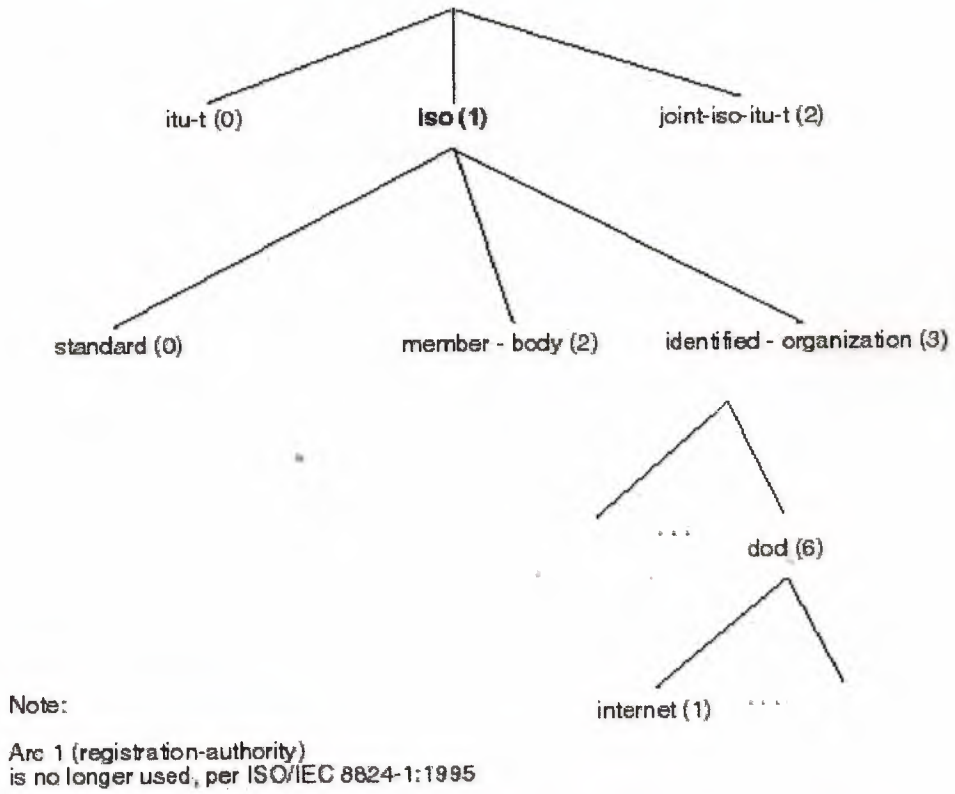


Figure 2.6 The root node and ISO assigned OBJECT IDENTIFIER component values

The root does not need a designation, but a specific numeric value designates the three connected arcs, or branches. The ISO branch (Figure 2.6) has three children: standard (0) designates international standards; member-body (2) is a three-digit numeric country code that ISO 3166 assigns to each member of ISO/IEC; and identified-organizations (3) have values of an international code designator (ICD), defined in ISO 6523. (Branch (1) was previously assigned to registration -authority (1), but it is no longer in use, per ISO 8834-1.) The U.S. Department of Defense is assigned to one of the children under 1.3, and is designated as 6. On this tree, the Internet community has designation 1.

To identify a particular position on the tree, we list the numeric values in a string, separated by periods. For example, to identify the position of the Internet subtree, we start at the root and move down until you reach position {1.3.6.1}.

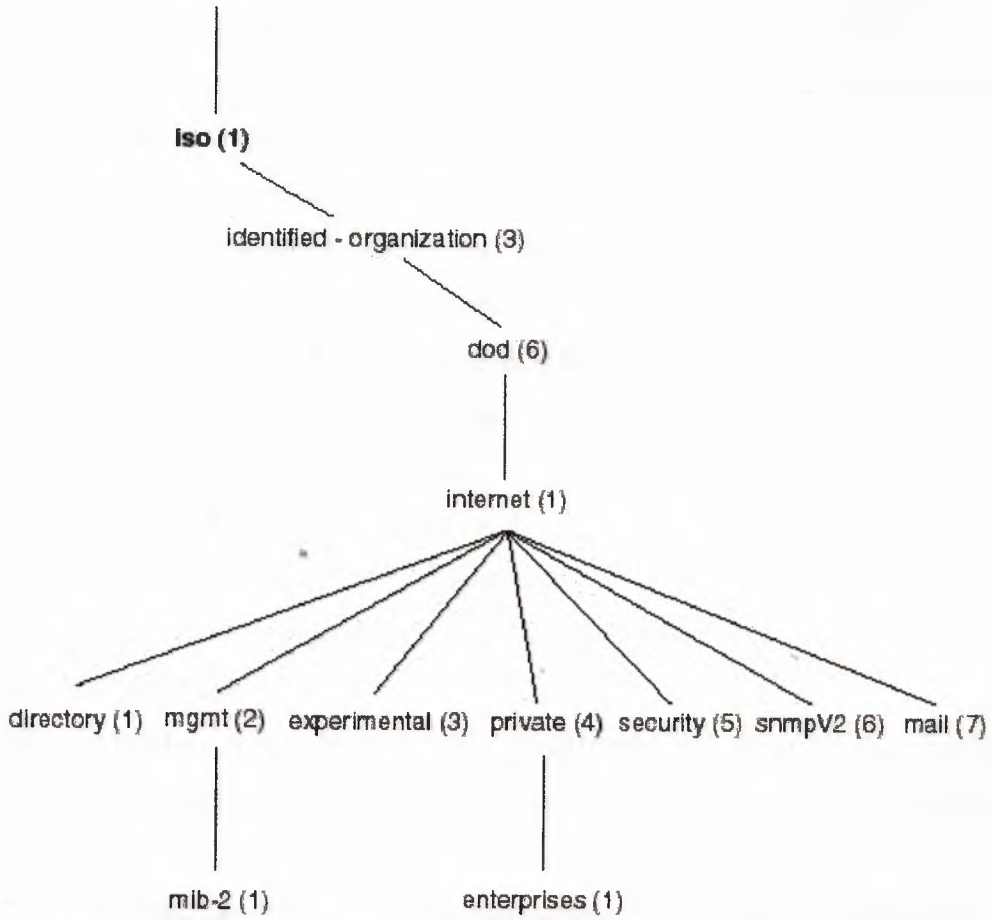


Figure 2.7 Internet assigned OBJECT IDENTIFIER component values

At the Internet level (Figure 2.7), we begin to see details germane to network management and SNMP. The Internet subtree has seven branches:

- The directory (1) subtree, {internet 1} or {1.3.6.1.1}, is reserved for future use by the OSI directory within the Internet.
- The mgmt (2) subtree, {internet 2} or {1.3.6.1.2}, is managed by the Internet Assigned Numbers Authority, and includes the standard MIBs.
- The experimental (3) subtree, {internet 3} or {1.3.6.1.3}, is used for Internet experiments.
- The private (4) subtree, {internet 4} or {1.3.6.1.4}, allows vendors to register objects.
- The security (5) subtree, {internet 5} or {1.3.6.1.5} for security-related objects.
- The snmpV2 (6) subtree, {internet 6} or {1.3.6.1.6}, for SNMP version 2 objects.
- The mail (7) subtree, {internet 7} or {1.3.6.1.7}, for mail objects.

The Internet Assigned Numbers Authority (IANA) administers these subtrees and publishes them in the current Assigned Numbers document (currently RFC 1700).

Let's now return to the example given at the beginning of this section. Now that we know the identities of the individual tree structures, we can construct the following sequence:

```
internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1 }
mgmt OBJECT IDENTIFIER ::= { internet 2 }
mib OBJECT IDENTIFIER ::= { mgmt 1 }
system OBJECT IDENTIFIER ::= { mib-2 1 }
sysDescr OBJECT IDENTIFIER ::= { system 1 }
```

When these tree structures are combined, the result becomes:

```
sysDescr OBJECT IDENTIFIER ::= { 1.3.6.1.2.1.1.1 }
```

To the OID we need to add one last element—a suffix that identifies whether a particular variable occurs just once (a *scalar*) or whether the variable occurs multiple times (as in *columnar* entries).

Since sysDescr is a scalar, not columnar, object, there is only one instance of it. (In other words, we can have only one description of the system being managed.) Therefore, a

.0 is added to the end of the OID:

{1.3.6.1.2.1.1.1.0.}

If the object was a columnar entry, which could have multiple instances, an index plus a nonzero suffix (.1, .2, an IP address, and so on) would identify the object within the table. Experimental codes, with prefix {1.3.6.1.3}, have been assigned for many LAN and WAN objects and MIBs, such as ISO CLNS (Connectionless Network Service), the Synchronous Optical Network (SONET) objects, and Asynchronous Transfer Mode (ATM) objects, while the technologies and their MIBs were in the testing phase of development.

2.7 The Concise SMI Definition

The best way to summarize this chapter is to include a module entitled RFC1155-SMI from RFC 1155. This module defines all of the constructs discussed in this chapter. In the interest of timeliness, the SMI definition includes the new OBJECT-TYPE macro. Comment lines enclosed within angle brackets (<...>) indicate the beginning and end of the revised section from RFC 1212.

Definition 2-1. Concise SMI Definition

```
RFC1155-SMI DEFINITIONS ::= BEGIN
  EXPORTS — EVERYTHING
  internet, directory, mgmt,
  experimental, private, enterprises,
  OBJECT-TYPE, ObjectName, ObjectSyntax,SimpleSyntax,
  ApplicationSyntax, NetworkAddress, IpAddress,
  Counter, Gauge, TimeTicks, Opaque;
  — the path to the root (from RFC 1155)
  internet  OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
  directory  OBJECT IDENTIFIER ::= { internet 1 }
  mgmt  OBJECT IDENTIFIER ::= { internet 2 }
  experimental  OBJECT IDENTIFIER ::= { internet 3 }
  private   OBJECT IDENTIFIER ::= { internet 4 }
  enterprises  OBJECT IDENTIFIER ::= { private 1 }
```

```
< definition of object types (taken from RFC 1212) >
OBJECT-TYPE MACRO ::=
  BEGIN
    TYPE NOTATION ::=
          — must conform to
          — RFC1155's ObjectSyntax
          "SYNTAX" type(ObjectSyntax)
          "ACCESS" Access
          "STATUS" Status
          DescrPart
          ReferPart
          IndexPart
          DefValPart
    VALUE NOTATION ::= value (VALUE ObjectName)
    Access ::= "read-only"
          | "read-write"
          | "write-only"
          | "not-accessible"
    Status ::= "mandatory"
          | "optional"
          | "obsolete"
          | "deprecated"
    DescrPart ::=
          "DESCRIPTION" value (description DisplayString)
          | empty
    ReferPart ::=
          "REFERENCE" value (reference DisplayString)
          | empty
    IndexPart ::=
          "INDEX" "{" IndexTypes "}"
          | empty
```

```
    IndexTypes ::=
          IndexType | IndexTypes ","        IndexType
    IndexType ::=
          -- if indexobject, use the SYNTAX
          -- value of the correspondent
          -- OBJECT-TYPE invocation
        value (indexobject ObjectName)
          -- otherwise use named SMI type
          -- must conform to IndexSyntax below
        | type (indextype)
  DefValPart ::=
      "DEFVAL" "{" value (defvalue ObjectSyntax) "}"
        | empty
END
IndexSyntax ::=
  CHOICE {
    number
      INTEGER (0..MAX),
    string
      OCTET STRING,
    object
      OBJECT IDENTIFIER,
    address
      NetworkAddress,
    ipAddress
      IpAddress
 }
< names of objects in the MIB (taken from RFC 1155) >
ObjectName ::= OBJECT IDENTIFIER
-- syntax of objects in the MIB
  ObjectSyntax ::=
```

```
    CHOICE {
      simple
          SimpleSyntax,
— note that simple SEQUENCEs are not directly
— mentioned here to keep things simple (i.e.,
— prevent mis-use). However, application-wide
— types which are IMPLICITly encoded simple
— SEQUENCEs may appear in the following CHOICE
          application-wide
              ApplicationSyntax
    }
SimpleSyntax ::=
   CHOICE {
     number
       INTEGER,
     string
         OCTET STRING,
     object
         OBJECT IDENTIFIER,
     empty
         NULL
 }
ApplicationSyntax ::=
     CHOICE {
       address
          NetworkAddress,
       counter
          Counter,
       gauge
          Gauge,
       ticks
```

```
        TimeTicks,
      arbitrary
        Opaque
— other application-wide types, as they are
— defined, will be added here
    }
— application-wide types
NetworkAddress ::=
    CHOICE {
     internet
        IpAddress
    }
IpAddress ::=
 [APPLICATION 0]   — in network-byte order
   IMPLICIT OCTET STRING (SIZE (4))
Counter ::=
 [APPLICATION 1]
   IMPLICIT INTEGER (0..4294967295)
Gauge ::=
 [APPLICATION 2]
   IMPLICIT INTEGER (0..4294967295)
TimeTicks ::=
 [APPLICATION 3]
   IMPLICIT INTEGER (0..4294967295)
Opaque ::=
 [APPLICATION 4]   — arbitrary ASN.1 value,
   IMPLICIT OCTET STRING  —  "double-wrapped"
END
```

This concludes the discussion of the SMI for SNMP version 1.

CHAPTER THREE

MANAGEMENT INFORMATION BASES

This chapter extends naming mechanisms to include *management information bases* (MIBs), which store management information. We can think of a MIB as a virtual information warehouse. Like a physical warehouse with specific floors, aisles, and bins, the MIB must implement an inventory control scheme. SMI defines the scheme for the MIBs. Just as a large company can have several warehouses, there are several different types of MIBs. Some, such as Internet standards, are for public use; specific organizations have developed others for private use for their products.

3.1 MIBs within the Internet Object Identifier Subtree

A tree represents the management structure, with branches and leaves representing the managed objects (Figure 3.1). In the figure, we can see seven subtrees under Internet: directory (1), mgmt (2), experimental (3), private (4), security (5), snmpV2 (6), and mail (7). The directory (1) subtree is reserved for future use of the OSI directory within the Internet. The mgmt (2) subtree handles Internet-approved documents, such as the Internet standard MIBs, which are MIB-I (RFC 1156) and MIB-II (RFC 1213). An object identifier (OID) with a prefix of {1.3.6.1.2.1} denotes managed objects within MIB-I and MIB-II. Internet experiments use the experimental subtree (3).

The private subtree (4) allows vendors to register a MIB for their equipment. The enterprise subtree, whose branches are private organizations, falls under the private subtree. The IANA assigns "enterprise codes" to branches representing private organizations and publishes them in the current assigned numbers RFC. Enterprise OIDs begin with the prefix {1.3.6.1.4.1}.

3.2. MIB Development

MIBs address the need for a standard network management platform by the Internet as a whole and by private enterprises. These MIBs require a consistent objective and format to realize this objective.

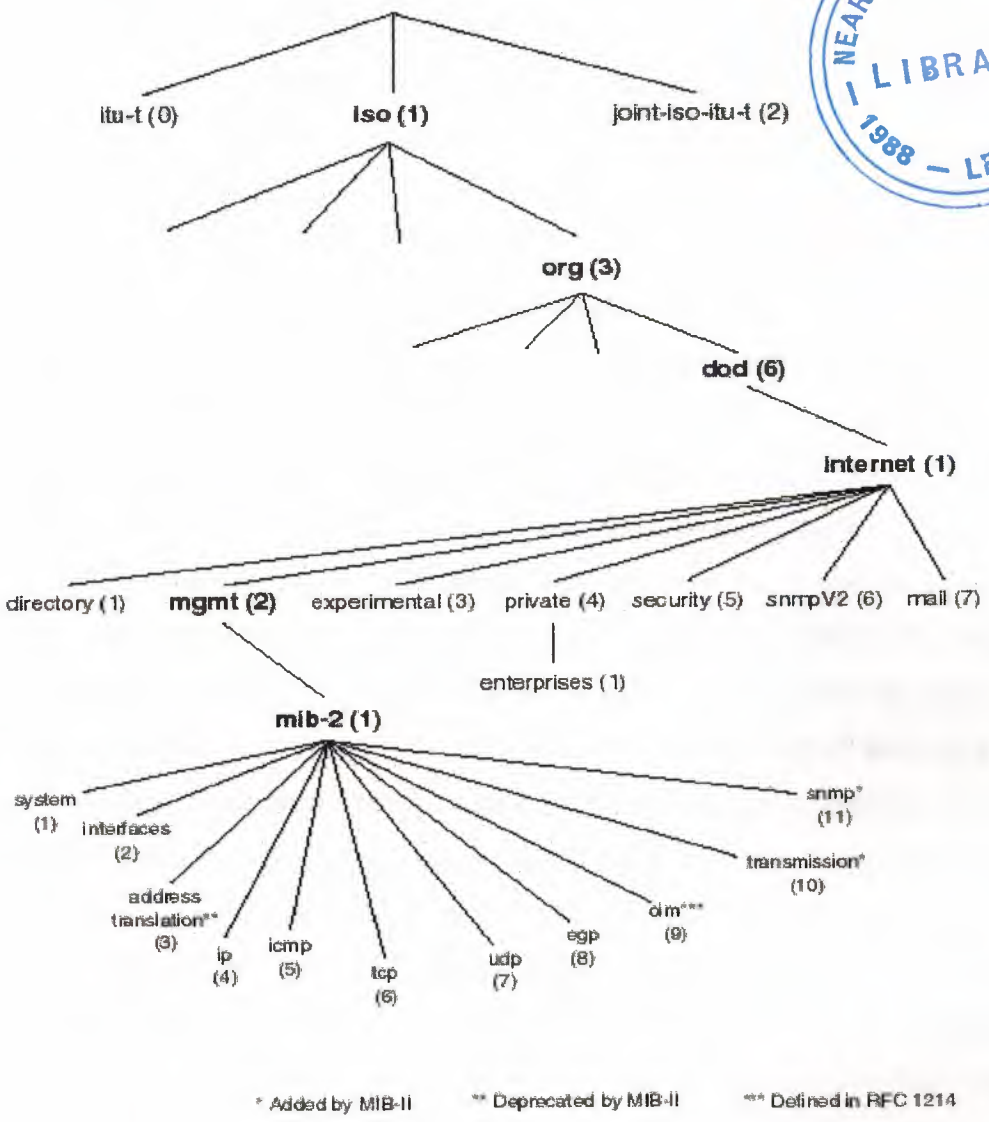


Figure 3.1 The Internet OID tree

3.2.1 MIB-I—RFC 1156

The first MIB, MIB-I (RFC 1156), was published in May 1990. MIB-I divided managed objects into eight groups in order to simplify OID assignment and implementation (that is, the SMI "structure"). Those groups were System, Interfaces, Address Translation, IP, ICMP, TCP, UDP, and EGP.

3.2.2 Concise MIB Definitions—RFC 1212

Prior to the publication of RFC 1212, there were two ways to define objects: a

textual definition and the ASN.1 OBJECT-TYPE macro. RFC 1212 embedded the textual definition within the OBJECT-TYPE macro, reducing the amount of documentation. The Concise SMI Definition includes this macro.

3.2.3 Elements of the OBJECT-TYPE macro

Since the OBJECT-TYPE macro seems cryptic to most people, a few words of explanation are in order. Each object has a number of attributes: SYNTAX, ACCESS, STATUS, DESCRIPTION, REFERENCE, INDEX, and DEFVAL. SYNTAX defines the object's data structure. Simple data types such as INTEGER, OCTET STRING, or NULL are examples of these data structures. SYNTAX also defines special cases of the simple objects, including an enumerated integer that defines an integer value, and a DisplayString restricted to printable ASCII characters. Table objects use the SEQUENCE OF syntax. ACCESS defines the minimum level of access to (or support of) an object. ACCESS may have values of read-only, read-write, not-accessible, or write-only. SNMP does not permit the write-only value. Table or row objects define ACCESS to be not-accessible. STATUS defines the implementation support for the object, which may be mandatory, optional, deprecated (discouraged), or obsolete. When STATUS defines a level of support for a particular group, that level applies to all objects within the group. Objects that have been replaced by backwards-compatible objects are "deprecated." Objects that are no longer supported are "obsolete." DESCRIPTION, which is not always present, provides a textual definition of an object type. REFERENCE, also not necessarily present, is a textual cross-reference to an object defined by another MIB module. INDEX works only with row objects. It indexes the order in which objects appear in a row, that is, the column order. Agents use DEFVAL, also optional, to populate values of columnar objects. For example, when an SNMP agent creates a new row, the DEFVAL clause assigns a default value to the objects within the row. For example, an OCTET STRING object may have a DEFVAL clause of 'FFFFFFFFFFFF'H.

3.2.4 Defining Table Structures in MIBs

Definition 3-1 (taken from RFC 1213) dissects the elements of a table. The italicized text after each section is my explanation. Double hyphens (--) indicate a comment

line within the table structure. The comment defines the purpose of the table.

Definition 3-1. Defining the UDP Listener table from RFC 1213

```
-- the UDP Listener table
-- The UDP listener table contains information about this
-- entity's UDP end-points on which a local application is
-- currently accepting datagrams.
udpTable     OBJECT-TYPE
    SYNTAX  SEQUENCE OF UdpEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "A table containing UDP listener information."
    ::= { udp 5 }
```

The object name (or table name) udpTable identifies a table object. Note that this name begins with a lowercase letter. The SYNTAX defines a SEQUENCE OF UdpEntry. This refers to a type definition (listed below) that defines the objects that make up each row of the table.

```
udpEntry     OBJECT-TYPE
    SYNTAX  UdpEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "Information about a particular current UDP listener."
    INDEX { udpLocalAddress, udpLocalPort }
    ::= { udpTable 1 }
```

The object name (or row name) udpEntry defines each row of the table. The INDEX clause specifies instances for columnar objects in the table. The instance values determine the order in which the objects are retrieved.

```
UdpEntry ::=
```

SEQUENCE {

udpLocalAddress IpAddress,

udpLocalPort INTEGER (0..65535)

}

The type definition UdpEntry identifies the objects that make up the row. Note that the type definition, often called a sequence name, is the same as the row name except that it begins with an uppercase letter. Each row has two columns, the udpLocalAddress (an IpAddress type) and the udpLocalPort (an INTEGER type).

udpLocalAddress OBJECT-TYPE

SYNTAX IpAddress

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The local IP address for this UDP listener. A UDP listener willing to accept datagrams for any IP interface associated with the node, uses the value 0.0.0.0."

::= { udpEntry 1 }

The notation { udpEntry 1 } indicates the first column in the table. The SYNTAX is a defined type, IpAddress. The description provides the address {0.0.0.0}.

udpLocalPort OBJECT-TYPE

SYNTAX INTEGER (0..65535)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The local port number for this UDP listener."

::= { udpEntry 2 }

The notation { udpEntry 2 } indicates the second column in the

table. The SYNTAX is an INTEGER type, with values ranging from 0 to 65,535.

An example of this table would be:

Local Address	Local Port
0.0.0.0	69 (TFTP)
0.0.0.0	161 (SNMP)
0.0.0.0	520 (Router)

In this example, the table contains three rows and two columns. All local addresses are [0.0.0.0], which indicates that the table is willing to accept IP datagrams from any address on this port.

3.3 MIB I and MIB II Groups

Managed objects are arranged into groups for two reasons. First, a logical grouping facilitates the use of the object identifiers and tree structure. Second, it makes the SNMP agent design more straightforward because the implementation of a group implies the implementation of all objects within the group. Thus, both the software developer and the end user can clearly understand a statement of support for, say, the TCP Group. MIB-I contained 114 objects. MIB-II, which is backward-compatible with MIB-I, contains these 114 objects plus 57 more, for a total of 171 objects.

3.3.1 The System Group

The System group provides a textual description of the entity in printable ASCII characters. This text includes a system description, OID, the length of time since the reinitialization of its network management entity, and other administrative details. Implementation of the System group is mandatory. The OID tree for the System group is designated {1.3.6.1.2.1.1}.

3.3.2 The Interfaces Group

The Interfaces group {1.3.6.1.2.1.2} provides information about the hardware interfaces on a managed device. This information is presented in a table. The first object (ifNumber) indicates the number of interfaces on the device. For each interface, a row entry is made into the table, with 22 column entries per row. The column entries provide information about the interfaces, such as the interface speed, physical (hardware) address, current operational state, and packet statistics.

3.3.3 The Address Translation Group

MIB-I included the Address Translation group but it was deprecated in MIB-II. The "deprecated" status means that MIB-II includes the Address Translation group for compatibility with MIB-I, but will probably exclude it from future MIB releases. The Address Translation group provided a table that translated between IP addresses and physical (hardware) addresses. In MIB-II and future releases, each protocol group will contain its own translation tables. The Address Translation group is designated {1.3.6.1.2.1.3}. It contains one table with three columns per row.

3.3.4 The IP Group

The Internet Protocol (IP) group is mandatory for all managed nodes and provides information on host and router use of the IP. This group includes a number of scalar objects that provide IP-related datagram statistics and the following three tables: an address table (ipAddrTable); an IP to physical address translation table (ipNetToMediaTable); and an IP forwarding table (ipForwardTable). Note that RFC 1354 defined the ipForwardTable, which replaces and obsoletes the ipRoutingTable in MIB-II. The IP subtree is designated {1.3.6.1.2.1.4}.

3.3.5 The ICMP Group

The Internet Control Message Protocol (ICMP) group is a mandatory component of IP and is defined in RFC 792. The ICMP group provides intranetwork control messages and represents various ICMP operations within the managed entity. The ICMP group contains 26 scalar objects that maintain statistics for various ICMP messages, such

as the number of ICMP Echo Request messages received or ICMP Redirect messages sent. This group is designated {1.3.6.1.2.1.5} on the OID tree.

3.3.6 The TCP Group

The Transmission Control Protocol (TCP) group is mandatory and provides information regarding TCP operation and connections. This group contains 14 scalar objects and one table. The scalar objects record various TCP parameters and statistics, such as the number of TCP connections that the device supports, or the total number of TCP segments transmitted. The table, tcpConnTable, contains information concerning a particular TCP connection. The OID for this group is {1.3.6.1.2.1.6}.

3.3.7 The UDP Group

The User Datagram Protocol (UDP) group is mandatory and provides information regarding UDP operation. Because UDP is connectionless, this group is much smaller than the connection-oriented TCP group. It does not have to compile information on connection attempts, establishment, reset, and so on. The UDP group contains four scalars and one table. The scalar objects maintain UDP-related datagram statistics, such as the number of datagrams sent from this entity. The table, udpTable, contains address and port information. The OID for this group is {1.3.6.1.2.1.7}.

3.3.8 The EGP Group

The Exterior Gateway Protocol (EGP) group is mandatory for all systems that implement the EGP. The EGP communicates between autonomous (self-contained) systems, and RFC 904 describes it in detail. The EGP group includes 5 scalar objects and one table. The scalars maintain EGP-related message statistics. The OID for this group is {1.3.6.1.2.1.8}.

3.3.9 The CMOT (OIM) Group

At one time, during the development of the Internet Network Management Framework, there was an effort to use SNMP as an interim step in the push for a network management standard, and to make the Common Management Information Protocol

(CMIP) over TCP/IP (CMOT) the long-term, OSI-compliant solution. As a result, the CMOT group was placed within MIB-II. Experience has shown, however, that SNMP is not an interim solution, and that the OSI-related network management protocol requires unique MIBs. Therefore, it's unlikely that we will encounter the OIM group within any commercially available SNMP managers or agents.

3.3.10 The Transmission Group

The Transmission group designated {1.3.6.1.2.1.10}, contains objects that relate to the transmission of the data. RFC 1213 defines none of these objects explicitly. However, the document does say that these transmission objects will reside in the experimental subtree {1.3.6.1.3} until they are "proven."

3.3.11 The SNMP Group

Since this book is about SNMP, we should be especially interested in the SNMP group, which provides information about SNMP objects (Figure 3.2). There are a total of 30 scalar objects in this group, including SNMP message statistics, the number of MIB objects retrieved, and the number of SNMP traps sent. This group is designated {1.3.6.1.2.1.11}.

3.4. The Ethernet RMON MIB

As networks have become increasingly distributed, geographically and logically, network management has become more challenging. One solution is to place remote management devices, sometimes called probes, on remote segments. The probes act as the eyes and ears of the network management system, providing managers with statistical information. The remote network monitoring (RMON) MIB standardizes the management information sent to and from these probes; it is presented in RFC 1757. A vendor-specific RMON implementation is the focus of "Continuous Monitoring of Remote Networks. The RMON MIB is assigned OID {1.3.6.1.2.1.16} and contains 9 groups. All of these groups are optional (not mandatory), but the implementation of some groups requires other groups. For example, the Filter group requires the Packet Capture group. The following is a summary of the nine Ethernet groups:

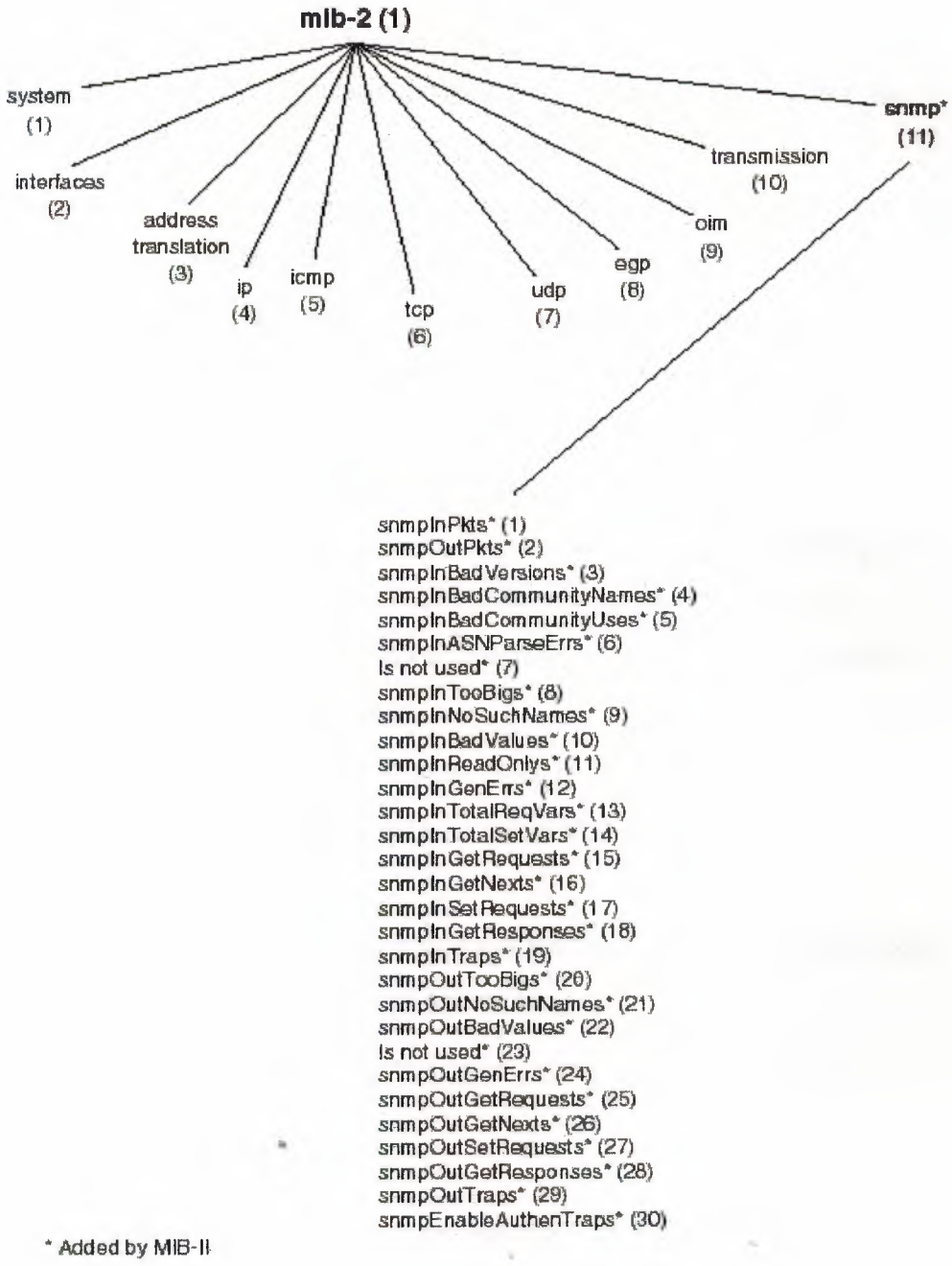


Figure 3.2 The SNMP group

Group	Description
statistics (1)	Provides probe-measured statistics, such as the number and sizes of packets, broadcasts, collisions, and so on.
history (2)	Records periodic statistical samples over time that you can use to analyze trends.
alarm (3)	Compares statistical samples with preset thresholds, generating alarms when a particular threshold is crossed.
host (4)	Maintains statistics of the hosts on the network, including the MAC addresses of the active hosts.
hostTopN (5)	Provides reports sorted by host table statistics, indicating which hosts are at the top of the list in a particular category.
matrix (6)	Stores statistics in a traffic matrix that tracks conversations between pairs of hosts.
filter (7)	Allows packets to be matched according to a filter equation.
capture (8)	Allows packets to be captured after they pass through a logical channel.
event (9)	Controls the generation and notification of events, which may include SNMP trap messages.

3.5 The Token Ring RMON MIB

The token ring RMON MIB is under development as an extension to the Ethernet RMON MIB. Because of the popularity of token ring networks, this MIB has received a great deal of attention. The Ethernet RMON MIB defines nine groups, Statistics through Events. The token ring RMON MIB extends two of these groups, Statistics and History, and adds one unique group. This new group is called tokenRing, with object identifier { rmon 10 }. The statistics extensions allow an RMON-compatible device to collect token ring MAC-Layer errors and promiscuous errors. The MAC-Layer errors, such as token

errors and frame-copied errors, are specific to the token ring protocol; the promiscuous errors, such as counting number of broadcast packages or data packets between 512 and 1023 octets in length, are more general. Similarly, the history information is divided into MAC-Layer and promiscuous details. The token ring group contains four sub groups: ring station, which monitors station- and ring-specific events; ring station order, which tracks the network topology; ring station configuration, which controls the removal and/or configuration of stations on the ring; and source routing, which details source routing bridging information.

3.6 RMON2

The original RMON MIBs for Ethernet and token ring networks are primarily concerned with the operation and management of the Physical and Data Link Layers of a remote network. As such, they can compile statistics and historical information regarding Ethernet collisions, token ring frame copied errors, and so on, but they cannot look into the operation of the OSI Network through Application layers of that remote network.

RMON2, defined in RFC 2021, extends the RMON capabilities to those higher layers by adding 10 new groups, designated {rmon 11} through {rmon 20}. Figure 3.3 illustrates the OID branches for both RMON and RMON2. Thus, the higher layer protocols, such as TCP/IP or SPX/IPX, can be monitored for greater management visibility within the internetwork.

The ten groups within RMON2 are:

Group	**Description**
protocolDir (11)	Protocol Directory: lists, in a table, the inventory of protocols that the probe has the capability of monitoring. Each protocol is described by an entry in the table.

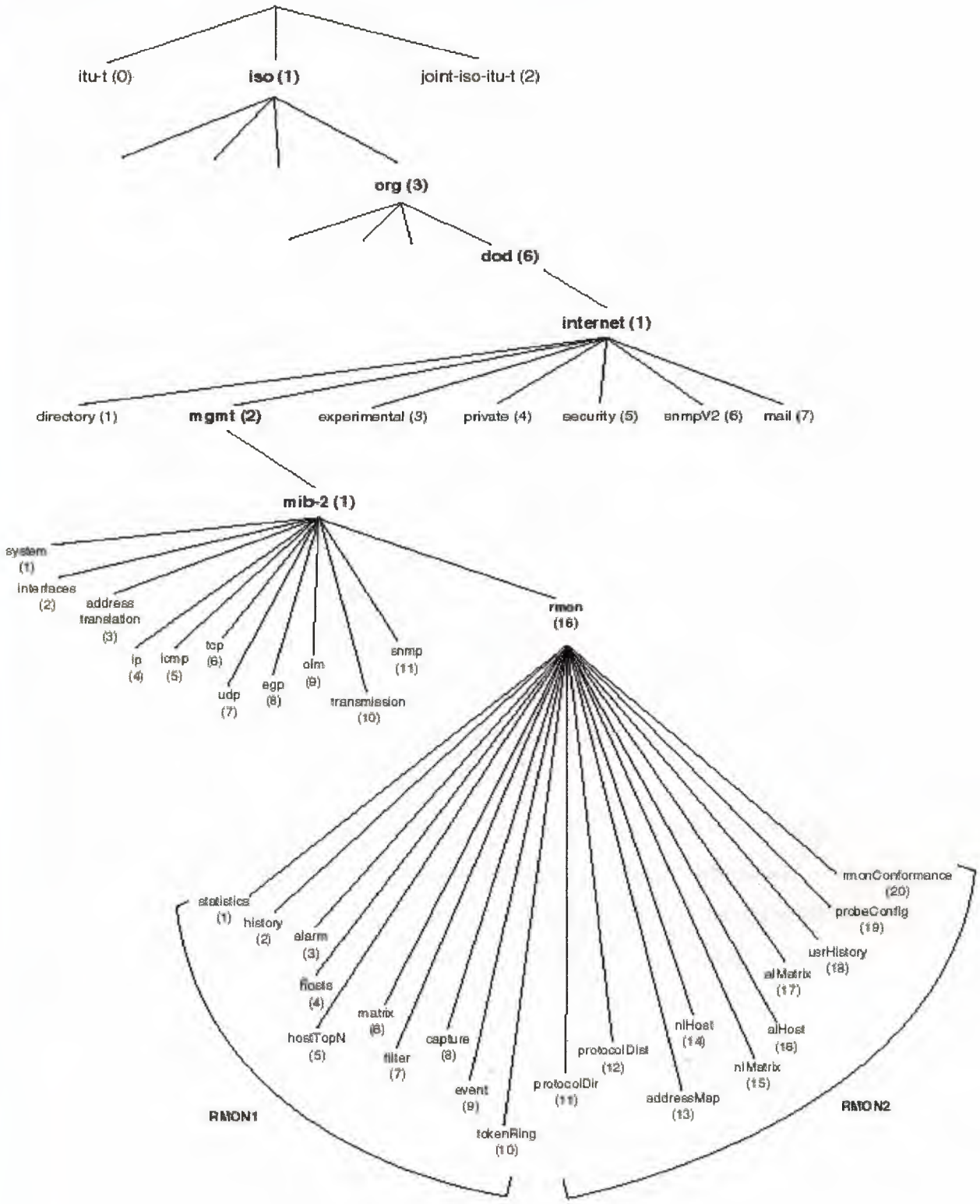


Figure 3.3 RMON1 and RMON2 object trees.

ProtocolDist (12)	Protocol Distribution: collects the relative amounts of octets and packets for the different protocols that are detected on a network segment. Each protocol is described by an entry in a table, and the network management station can easily determine the bandwidth consumed per protocol by accessing the information in that table.
addressMap (13)	Address Map: correlates Network Layer addresses and MAC Layer addresses, and stores the information in tables.
nlHost (14)	Network Layer Host: counts the amount of traffic sent from and to each Network Layer address discovered by the probe, and stores the information in tables.
nlMatrix (15)	Network Layer Matrix: counts the amount of traffic sent between each pair of network addresses discovered by the probe, and stores the information in tables from both source to destination and destination to source.
alHost (16)	Application Layer Host: counts the amount of traffic, by protocol and by host, that is sent from and to each network address discovered by the probe.
alMatrix (17)	Application Layer Matrix: counts the amount of traffic, by protocol, sent between each pair of network addresses discovered by the probe, and stores this information in tables. This group is similar to the nlMatrix group, but the focus is on the protocol in operation.
usrHistory (18)	Combines mechanisms seen in the alarm (3) and history (2) groups to provide user-specified history collection, and storing that information in tables.

probeConfig (19)	Controls the configuration of various operational parameters by the probe, such as the Ethernet and token ring RMON groups that are supported by the probe, software and hardware revision numbers of the probe, a trap destination table, and so on.
rmonConformance (20)	Describes the requirements for conformance to the RMON2 MIB.

3.7 Private MIBs

Many vendors have developed private MIBs that support hubs, terminal servers, and other networking systems. We can find these MIBs under the enterprises subtree, {1.3.6.1.4.1.A}. The A indicates a private enterprise code, defined in the "Assigned Numbers" RFC (RFC 1700) in the network management section. Because of these private MIBs are vendor-specific, interoperability is not always possible.

3.8. Accessing a MIB

This section gives an example of an SNMP management console retrieving values for MIB objects from a remote SNMP agent. In this case, the manager is a Sun Microsystems' SunNet Manager, and the agent is located in a Proteon's p4100+ router. Both devices connect to an Ethernet backbone. A Network General Corp. Sniffer protocol analyzer captured the data shown in Trace 3.9.

A protocol analyzer captures, then decodes, frames of data as they are transmitted on the LAN or WAN. These frames are numbered sequentially and stored in the same order. The analyzer can display these frames several ways; it can show all of the protocol layers, or just one. The example in this section shows only the highest layer, SNMP. The analyzer also lets us choose the amount of detail included. The minimum detail is a single summary line, and the maximum is the hexadecimal representation of the bits received on the wire. This exchange between the manager and the agent (Trace 3.9) involves two frames of information. Frame 109 contains an SNMP GetRequest PDU (protocol data unit,

the core of the SNMP message) and Frame 110 contains a GetResponse PDU.

The manager sends the GetRequest to the agent asking for the values of the objects within the system subtree, OID {1.3.6.1.2.1.1}. The PDU requests information about all seven of the objects: sysDescr, sysObjectID, sysUpTime, sysContact, SysName, sysLocation, and sysServices. On the trace, we can see two coding elements for each of these objects. First, the manager requests the sysUpTime object to determine whether the agent within the router has restarted (warm or cold boot). Second, the manager asks for the values of each individual object in order. This trace also illustrates the use of the SEQUENCE type encoding of VarBinds. Each object is encoded with an OBJECT IDENTIFIER type, for example {1.3.6.1.2.1.1.2.0}. The Object Value field is encoded with a NULL type because the manager does not know this information.

Frame 110 gives the agent's GetResponse. The response returns each object and its associated value in the order that Frame 109 requested. The sysDescr provides a textual description of the device (Portable I80386 C Gateway ...). The sysObjectID has a value of {1.3.6.1.4.1.1.1.1.41}. From the prefix {1.3.6.1.4.1}, we know that this is a private enterprise subtree. The next digit (.1) is the enterprise code for Proteon, Inc. The sysUpTime object has a value of 263,621,778 hundredths of a second, which translates to roughly 30 days because the router's network management system was restarted. Two of the objects, sysContact {system 4} and sysLocation {system 6} appear not to have a value. In reality, they have a value of a zero-length string, but the network manager entered no values for those objects in the router's configuration file. The sysName is the domain name of the node (boulder.org). Finally, the sysServices {system 7} is a calculated sum that indicates the services this node performs. In this case, the value is 72, indicating a host offering application services (RFC 1213).

Trace 3.9 Browsing the system subtree (SNMP protocol decode)

```
Sniffer Network Analyzer data 10-Nov at 10:42:04 file ASAN_SYS.ENC Pg 1
---------------------------- Frame 109 ----------------------------
SNMP: ----- Simple Network Management Protocol -----
SNMP:
SNMP: Version = 0
SNMP: Community = boulder
```

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.1.0} (sysDescr.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.2.0} (sysObjectID.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.4.0} (system.4.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.5.0} (system.5.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.6.0} (system 6.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.7.0} (system 7.0)

SNMP: Value = NULL

SNMP:

---------- Frame 110 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP: Version = 0

SNMP: Community = boulder

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0] (sysUpTime.0)

SNMP: Value = 263621778 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.1.0} (sysDescr.0)

SNMP: Value = Portable I80386 C Gateway BOULDER.ORG S/N XXX V12.0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.2.0} (sysObjectID.0)

SNMP: Value = {1.3.6.1.4.1.1.1.1.41}

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 263621778 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.4.0} (system.4.0)

SNMP: Value =

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.5.0} (system.5.0)

SNMP: Value = BOULDER.ORG

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.6.0} (system.6.0)

SNMP: Value =

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.7.0} (system.7.0)

SNMP: Value = 72

SNMP:

CHAPTER FOUR

THE SIMPLE NETWORK MANAGEMENT PROTOCOL

This chapter completes the discussion of the Internet Network Management Framework by looking at SNMP, the protocol that communicates management information.

4.1 SNMP Objectives and Architecture

RFC 1157 states that "SNMP explicitly minimizes the number and complexity of management functions realized by the management agent itself". In other words, SNMP is designed to be simple. SNMP does this in three ways. By reducing the development cost of the agent software, SNMP has decreased the burden on vendors who wish to support the protocol, thereby increasing the protocol's acceptance. Second, SNMP is extensible, allowing vendors to add network management functions. Third, it separates the management architecture from the architecture of hardware devices, such as hosts and routers, widening the base of multivendor support.

SNMP has a very straightforward architecture. Figure 4.1a compares the SNMP architecture to the ISO/OSI model and the Advanced Research Projects Agency (ARPA) model, around which the Internet protocols and TCP/IP were developed. The four layers of the ARPA model do not map evenly to the seven layers of the OSI model.

Let's use an example to see how the processes within the SNMP architecture interact. Suppose a management console requests information about one of the managed nodes. The SNMP processes in both the manager and the agent respond to the console. The ASN.1 encoding at the Application layer provides the proper syntax for the SNMP message. The remaining functions authenticate the data (attach the SNMP header) and communicate the information request.

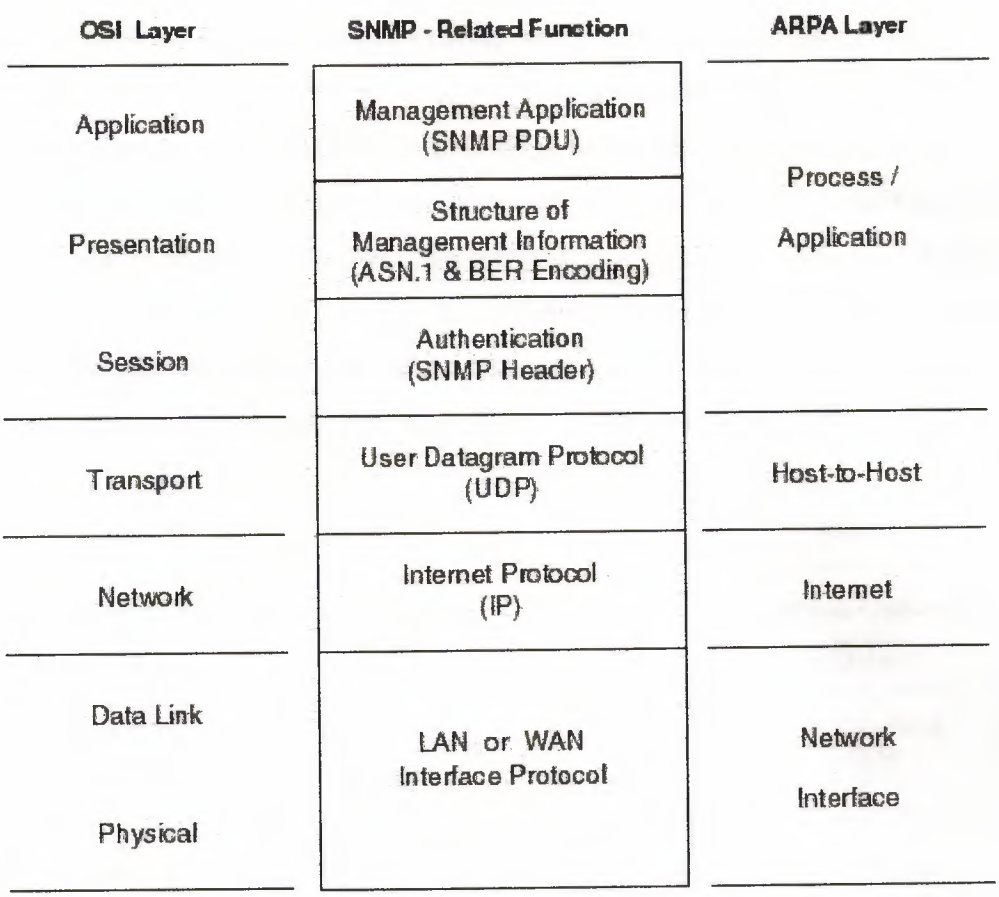


Figure 4.1a Comparing the SNMP architecture with the OSI and ARPA models

Because most management information does not demand the reliable delivery that connection-oriented systems provide, the communication channel between the SNMP manager and the agent is connectionless. When we compare the SNMP model to the ISO/OSI model, SNMP's connectionless communication mechanism removes some of the need for a Session layer and reduces the responsibilities of the lower four layers. For most implementations, the User Datagram Protocol (UDP) performs the Transport layer functions, the Internet Protocol (IP) provides the Network layer functions, and LANs such as Ethernet or token ring or WANs such as a leased line or a frame relay connection provide the Data Link and Physical layer functions.

If we compare SNMP to the Internet (or ARPA) architectural model (Figure 4.1b), we will notice that the ARPA model uses four layers to describe the entire communication function. In the ARPA model, SNMP would reside at the Process/Application layer.

However, while the ARPA Host-to-Host layer provides end-to-end communication reliability, SNMP's use of UDP assures only proper port addressing and a checksum; it does not provide octet-by-octet error control. IP provides the Internet layer functions, such as addressing and fragmentation, that are necessary to deliver an SNMP message from the source to the destination. Finally, the Network Interface layer deals with the LAN or WAN hardware, such as an interface to an FDDI or Frame Relay network connection. Figure 4.1b also shows the relative complexities of the host and router functions. Hosts implement all four layers of the ARPA model, whereas routers implement only the lower two.

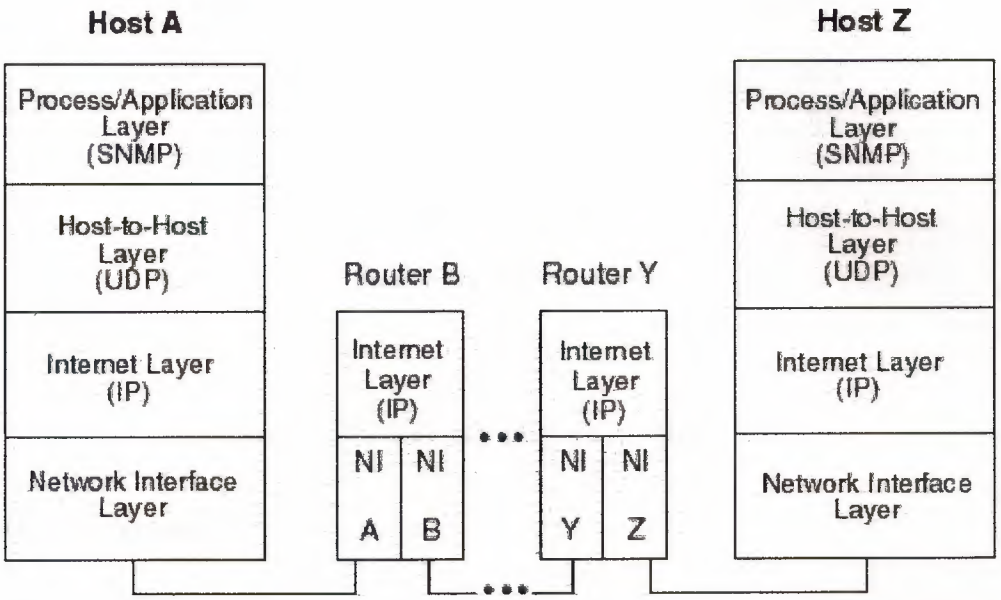


Figure 4.1b. Application-to-application connection

Comparing the SNMP architecture to the ISO/OSI and ARPA architectural models provides a theoretical basis for this discussion. But from a practical perspective, the SNMP model works as shown in Figure 4.2. This model contains several elements. It includes a management system that uses the SNMP manager, an SNMP agent, and managed resources, and the SNMP messages communicate management information via five SNMP protocol data units (PDUs). The management application issues the Get, GetNext, or Set PDUs. The managed system returns a GetResponse PDU. The agent may initiate a Trap (sometimes called an Event) PDU when predefined conditions are met.

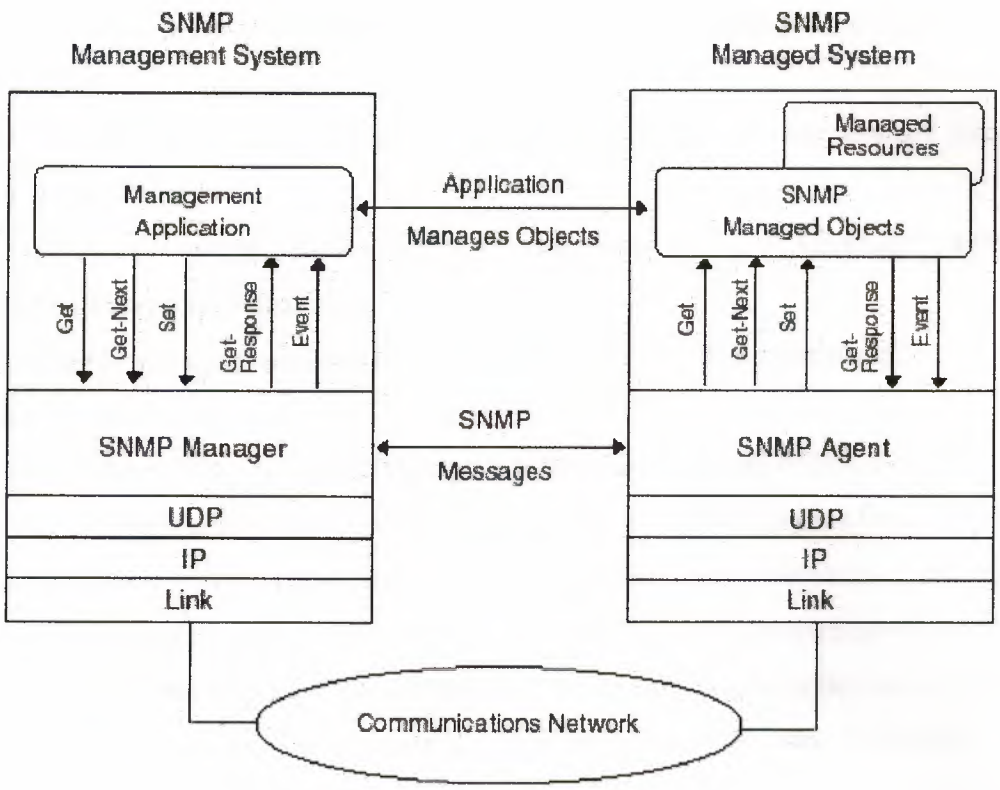


Figure 4.2 SNMP architecture (c 1990, IEEE)

4.2 SNMP Operation

The SNMP processes must occur in physical devices. For example, a router must have a physical processor that implements the software acting as an SNMP agent. Two sets of logical processes occur within those physical elements: the relationships that are specified between various network management entities, and the way network management information is communicated.

4.2.1 Network Management Relationships

The SNMP standard, RFC 1157, and the "SNMP Administrative Model," RFC 1351, define a number of terms. Many of these definitions describe relationships between management entities:

- *Network management stations* are devices that execute the management applications that control and monitor the network elements.

- *Network elements* are devices such as hosts, bridges, routers, and hubs that contain an agent and perform the network management functions that the network management stations request.
- The *SNMP* allows network management stations and the agents in the network elements to communicate.
- *SNMP application entities* reside at either a management station or a managed node, and use SNMP as a communication mechanism.
- *Protocol entities* are peer processes that implement SNMP, thus supporting the SNMP application entities.
- The *SNMP community* pairs an SNMP agent with an arbitrary set of SNMP application entities. The network administrator assigns the community a name (called the *community name*) which is essentially a password with associated rights and privileges. A management application with multiple community names may belong to multiple communities.
- *Authentic SNMP messages* are SNMP messages sent from an application entity to a specific SNMP community. The message contains the community name of interest.
- The *authentication scheme* is the method by which an SNMP message is identified as belonging to a specific SNMP community.
- The *MIB View* is the subset of MIB objects, which may be contained within several subtrees, that pertain to a network element.
- The SNMP *access mode* determines the level of access to objects that a particular application entity is allowed. The choices are read-only and read-write.
- The *community profile* pairs the SNMP access mode with the SNMP MIB View. The community profile represents specific access privileges for the variables in a MIB view.
- The SNMP *access policy* pairs an SNMP community with a SNMP community profile. The access policy represents the specific community profile that an agent permits the other members of the community to have.
- The SNMP *proxy agent* provides management functions on behalf of network elements that would otherwise be inaccessible.

Figure 4.3 illustrates some of the definitions described above.

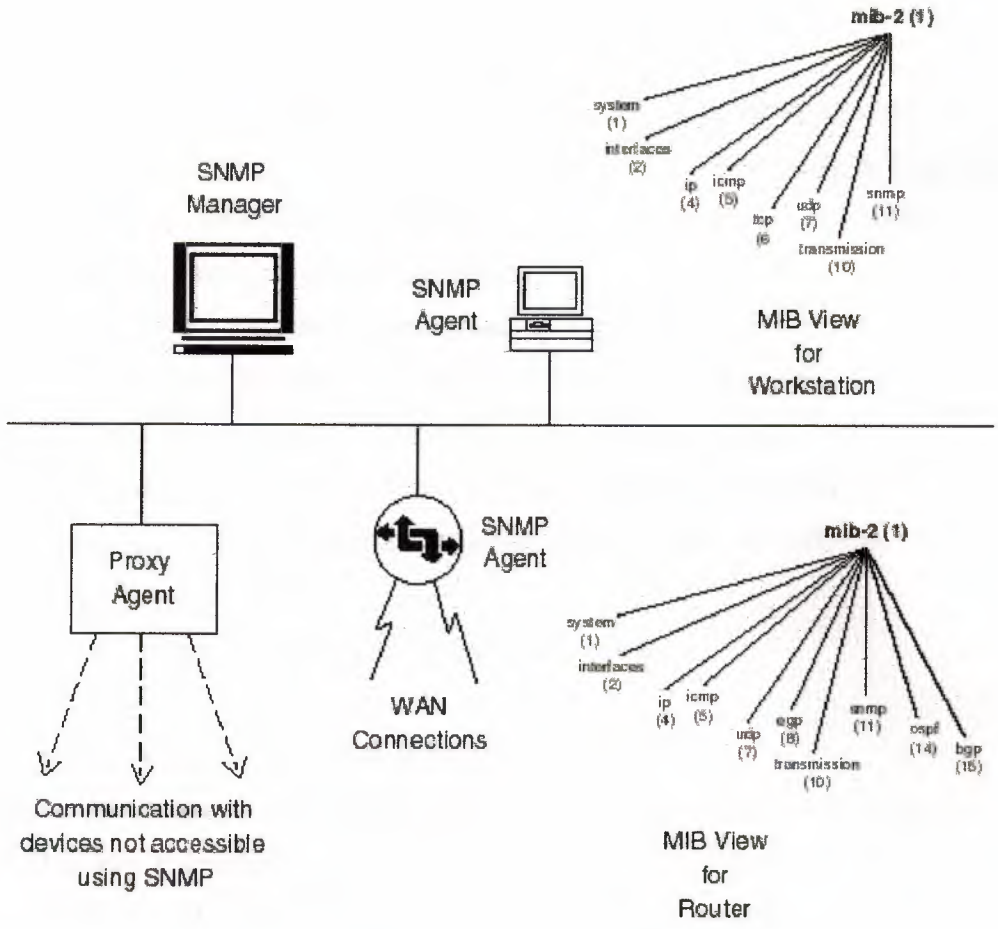


Figure 4.3 Network management relationships

4.2.2 Identifying and Communicating Object Instances

SMI managed object types have an object identifier (OID) that uniquely names them and locates their place on the object tree. An instance of an object type is an occurrence of that object type and has an assigned value. For example, the object sysDescr {1.3.6.1.2.1.1.1.0} might have a value of "Retix Remote Bridge Model 2265M." Suppose a network management station wishes to retrieve an instance of a specific object. The management station must use SNMP to communicate its question to the agent. Now, suppose multiple instances (or occurrences) of that object are possible. For example, say a router's routing table contains a number of entries. How would the network management station retrieve just the value of the third entry in the table?

For these SNMP operations, a *variable name* uniquely identifies each instance of an object type. This name consists of two parts of the form *x.y*. The *x* portion is the object type defined in the MIB, and the y portion is an OID fragment that identifies the desired instance. The following example should clarify this.

Consider a scalar object that has one instance. The objects contained in the System group are all scalar objects. For example, the sysServices object has an OID of {1.3.6.1.2.1.1.7} and occurs once. The *x* portion of the variable name is the OID, and the *y* portion has been assigned to 0. We can derive this by following the OID tree down to the object sysServices and adding the appropriate instance suffix (with the suffix, or *y* portion, shown in boldface type):

iso org dod internet mgmt mib-2 system sysServices Instance

1 3 6 1 2 1 1 7 **0**

Thus, the variable name for sysServices is {1.3.6.1.2.1.1.7.0}.

The variable name for a columnar object is more complicated because it must identify the location of an object within a two-dimensional data structure, such as a table having both rows and columns. (Within the RMON MIB, three-dimensional data structures are added, making the identification even more complex.) Using the familiar speadsheet as an example, the identification of a particular cell requires two coordinates, X and Y, which describe the horizontal and vertical positions, respectively. With columnar objects, an indexing scheme, specified in the INDEX clause in the ASN.1 definition for that object, provides a means for identifying the specific instance. The INDEX clause then further identifies the syntax to be used. And as one might expect, some of the indexing schemes are more complicated than others.

A final example (derived from RFC 1157) is from the TCP Connection Table, tcpConnTable. Suppose we wish to retrieve the state of the connection between port 575 on local address {a.b.c.d} and port 441 on remote address {w.x.y.z}. The OID for tcpConnState is {1.3.6.1.2.1.6.13.1.1}. The INDEX clause consists of four parts: tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress, and tcpConnRemPort. The *y* suffix would therefore be expressed as {a.b.c.d.575.w.x.y.z.441}. Therefore, the complete variable name would be:

```
{1.3.6.1.2.1.6.13.1.1.a.b.c.d.575.w.x.y.z.441}
```

The following examples show specific variable names for both scalar and columnar object types:

- The description of this system's services:

```
sysServices ::=
{1.3.6.1.2.1.1.7.0}
```

- The speed of interface 3:

```
ifSpeed.3 ::=
{1.3.6.1.2.1.2.2.1.5.3}
```

- The physical address associated with interface 2 and IP address {a.b.c.d} (Note that the first component is a .1, which indicates an IP address [see RFC 1157, page 13]):

```
atPhysAddress.2.1.a.b.c.d ::=
{1.3.6.1.2.1.3.1.1.2.2.1.a.b.c.d}
```

- The maximum IP datagram reassembly size associated with IP address {a.b.c.d}:

```
ipAdEntReasmMaxSize.a.b.c.d ::=
{1.3.6.1.2.1.4.20.1.5.a.b.c.d}
```

- The number of ICMP Echo (request) messages received at this device:

```
icmpInEchos ::=
{1.3.6.1.2.1.5.8.0}
```

- The state of a TCP connection between local port e, local address {a.b.c.d}, and remote port j, remote address {f.g.h.i}:

```
tcpConnState.a.b.c.d.e.f.g.h.i.j ::=
{1.3.6.1.2.1.6.13.1.1.a.b.c.d.e.f.g.h.i.j}
```

- Verification that a UDP listener is operational on port e of local IP address a.b.c.d:

```
udpLocalAddress.a.b.c.d.e ::=
{1.3.6.1.2.1.7.5.1.1.a.b.c.d.e}
```

- The neighbor state for the IP address a.b.c.d:

```
egpNeighState.a.b.c.d ::=
{1.3.6.1.2.1.8.5.1.1.a.b.c.d}
```

- The number of SNMP messages delivered to this device with unknown community names (a scalar):

```
snmpInBadCommNames ::=
{1.3.6.1.2.1.11.4.0}
```

4.3 SNMP Protocol Data Units (PDUs)

We will begin the discussion of PDUs by describing the position of the SNMP message within a transmitted frame. The frame is the unit of information transmitted between network nodes. For example, an IEEE 802.5 frame format defines the transmission between token ring nodes, and an ANSI T1.617 format defines the transmission between Frame Relay nodes. The local network header and trailers defined by the LAN or WAN protocol delimit the frame (Figure 4.4). The transmitted data is called an *Internet Protocol (IP) datagram*. The IP datagram is a self-contained unit of information sent from the source host to its intended destination via the internetwork. Inside the datagram is a destination IP address that steers the datagram to the intended recipient. Next, the User Datagram Protocol (UDP) header identifies the higher-layer protocol process (SNMP) that will process the datagram, and provides error control using a checksum. The SNMP message is the innermost part of the frame, carrying the actual data from the manager to and from the agent.

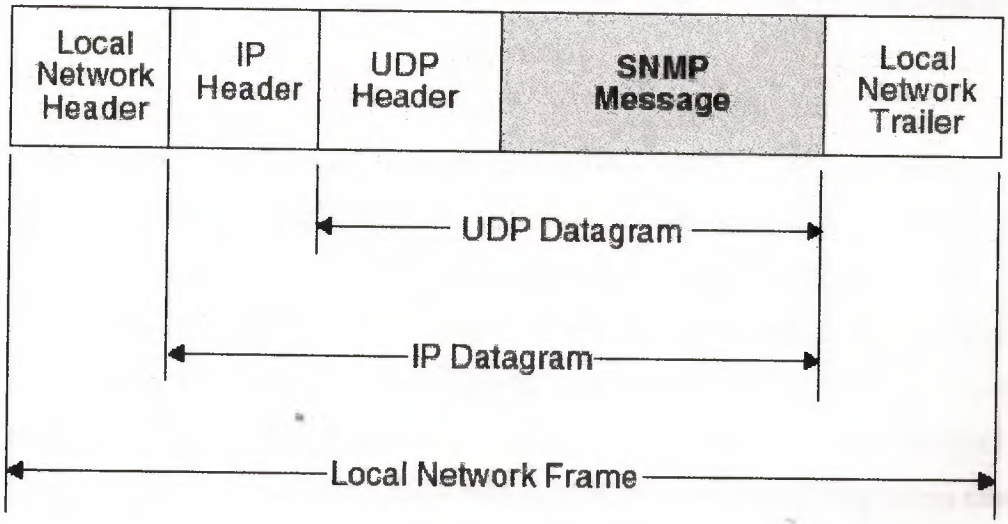


Figure 4.4 SNMP message within a transmission frame

When the IP is too long to fit inside one frame, it may be divided (or fragmented) into several frames for transmission on the LAN. The SNMP message itself is divided into two sections: a version identifier plus community name, and a PDU. The version identifier

and community name are sometimes referred to as the SNMP authentication header. There are five different PDU types: GetRequest, GetNextRequest, GetResponse, SetRequest, and Trap. The Get, Set, and Response PDUs have a common format (Figure 4.5), while the Trap PDU format is unique.

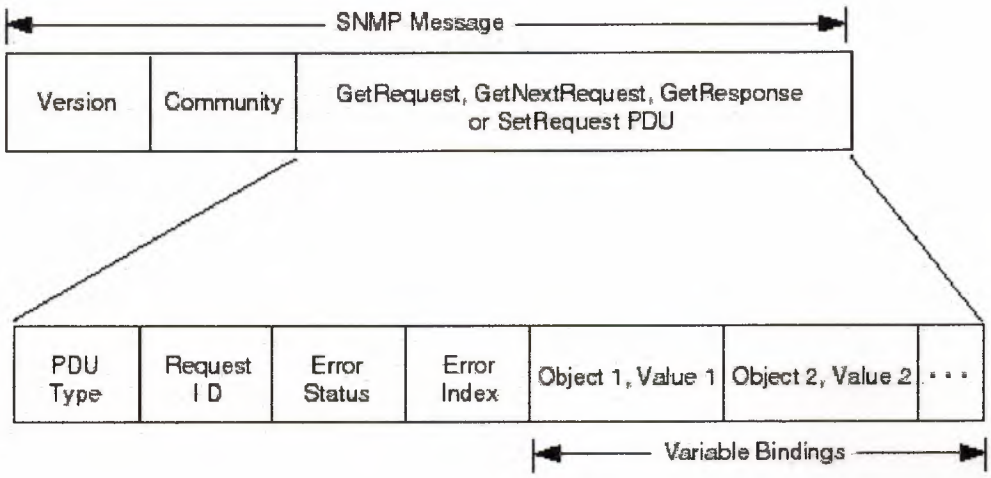


Figure 4.5 The SNMP GetRequest, GetNextRequest, GetResponse, and SetRequest PDU structures

The version number (an INTEGER type) assures that both manager and agent are using the same version of the SNMP protocol. Messages between manager and agent containing different version numbers are discarded without further processing. The community name (an OCTET STRING type) authenticates the manager before allowing access to the agent. The community name, along with the manager's IP address, is stored in the agent's community profile. If there's a difference between the manager and agent values for the community name, the agent will send an authentication failure trap message to the manager. If both the version number and community name from the manager match the ones stored in the agent, the SNMP PDU begins processing.

4.3.1 Get, Set, and Response PDU Formats

The GetRequest, GetNextRequest, SetRequest, and GetResponse PDUs share a common format (Figure 4.5). The first field, PDU Type, specifies the type of PDU the message contains:

PDU	PDU Type Field Value
GetRequest	0
GetNextRequest	1
GetResponse	2
SetRequest	3
Trap	4

The Request ID field is an INTEGER type that correlates the manager's request to the agent's response. The Error Status field is an enumerated INTEGER type that indicates normal operation (noError) or one of five error conditions. The possible values are:

Error	Value	Meaning
noError	0	Proper manager/agent operation.
TooBig	1	The size of the required GetResponse PDU exceeds a local limitation.
noSuchName	2	The requested object name did not match the names available in the relevant MIB View.
badValue	3	A SetRequest contained an inconsistent type, length, and value for the variable.
readOnly	4	Not defined in RFC 1157. (Historical footnote: this error is listed, but the description of the SetRequest PDU processing does not describe

		how this error is generated. The standard interpretation is that this error should not be generated, although some vendor's agents nevertheless do.)
genErr	5	Other errors, not explicitly defined, have occurred.

A Variable Binding (VarBind) pairs a variable name with its value. A VarBindList is a list of such pairings. Note that within the Variable Bindings fields of the SNMP PDUs (Figures 4.5 through 4.10), the word Object identifies the variable name (OID encoding of object type plus the instance) for which a value is being communicated. Also note that GetRequest or GetNextRequest PDUs use a value of NULL, which is a special ASN.1 data type.

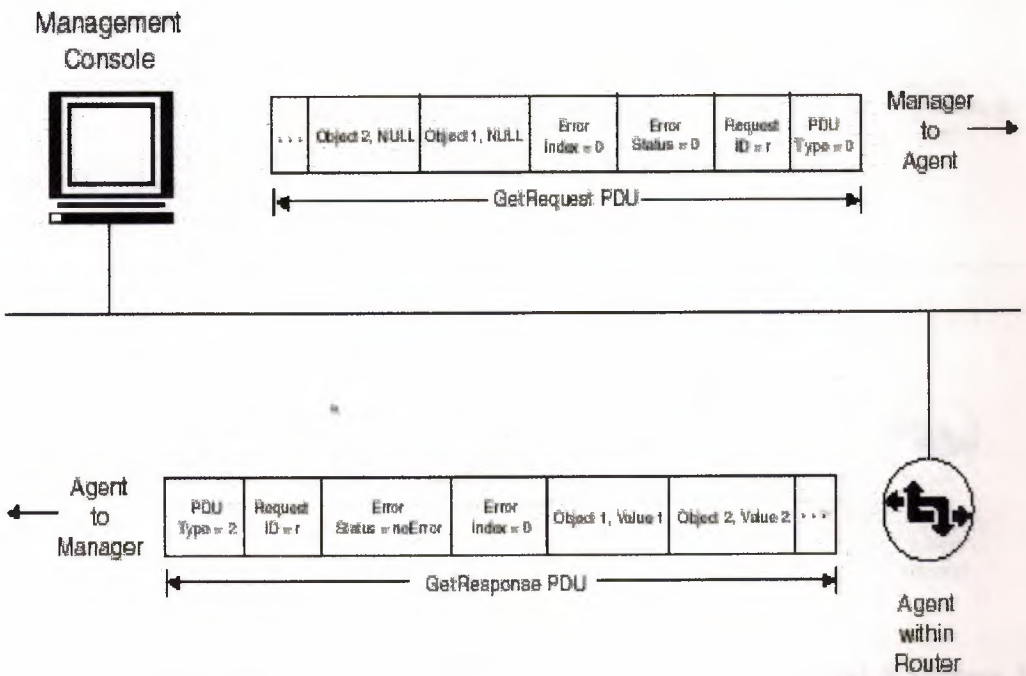


Figure 4.6 GetRequest/GetResponse PDU transmission (with no errors) *(Courtesy 3Com Corp.)*

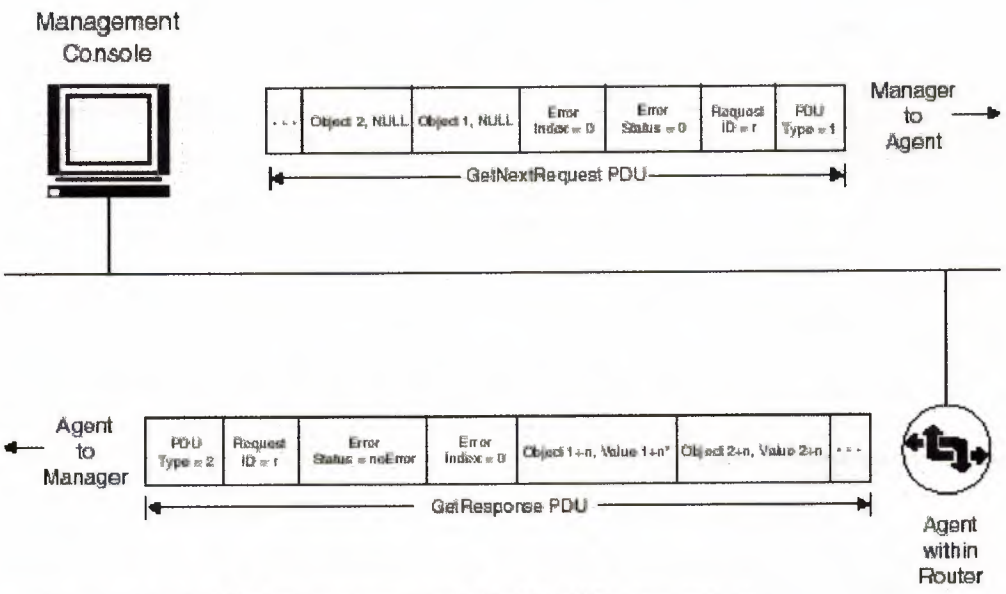


Figure 4.7 GetNextRequest/GetResponse PDU transmission (with no errors) *(Courtesy 3Com Corp.)*

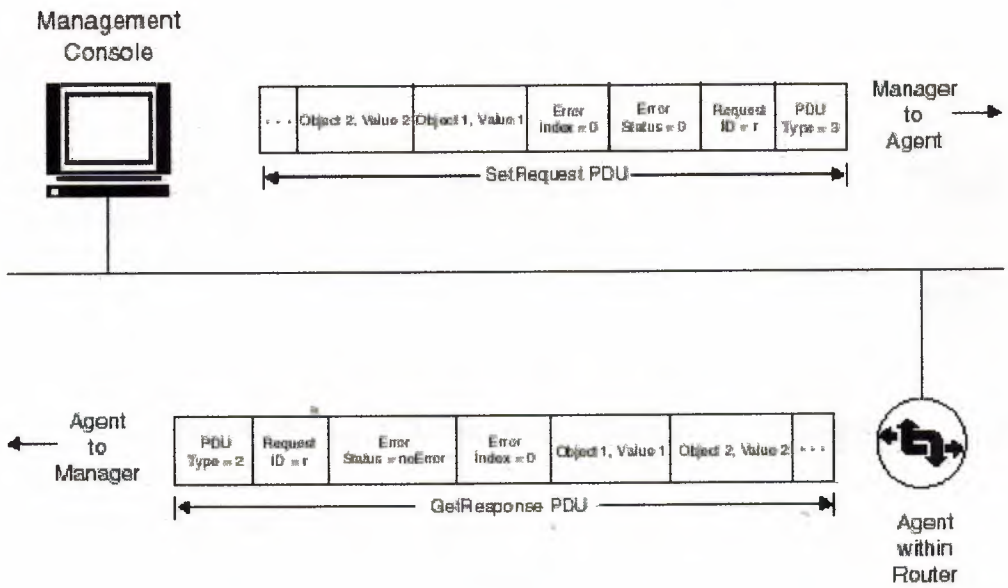


Figure 4.8 SetRequest/GetResponse PDU transmission (with no errors) *(Courtesy 3Com Corp.)*

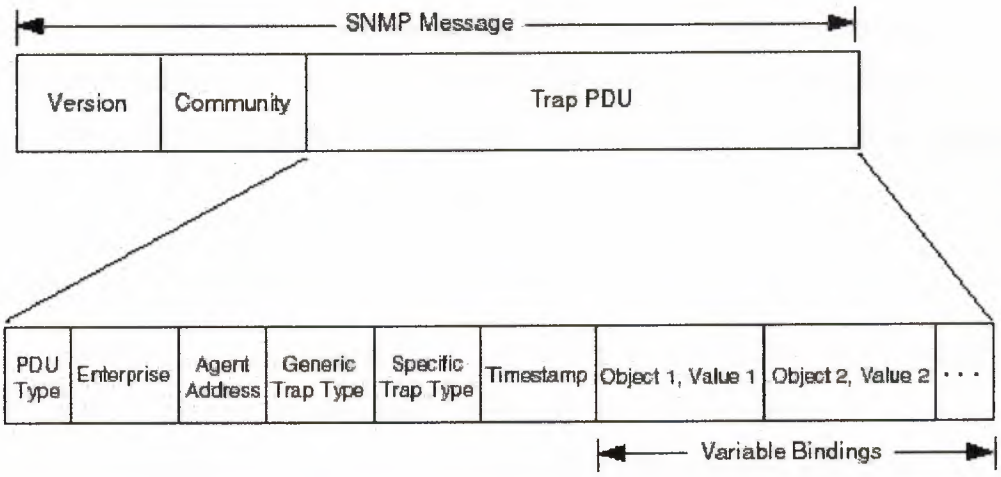


Figure 4.9 SNMP Trap PDU structure

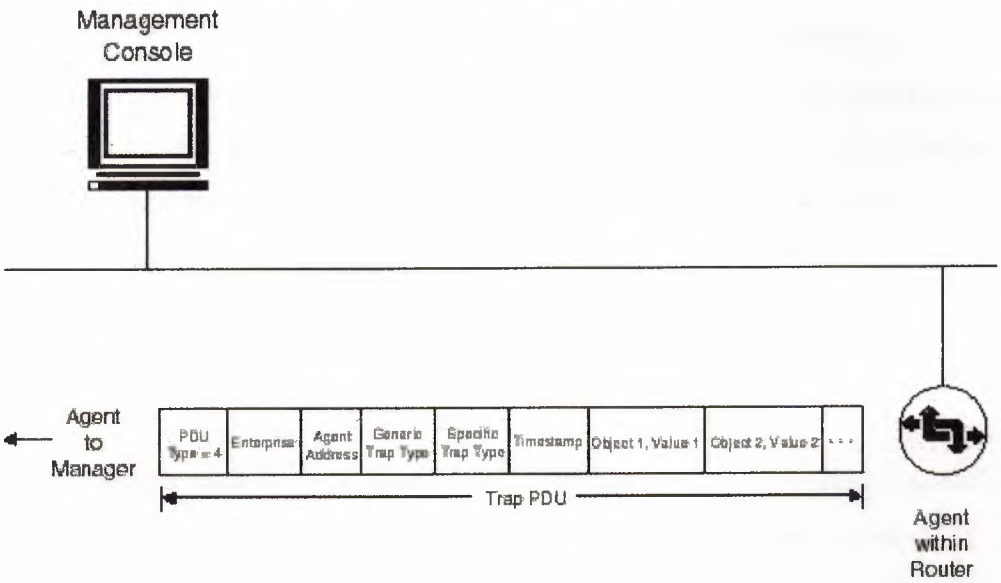


Figure 4.10 Trap PDU operation *(Courtesy 3Com Corp.)*

4.3.2 Using the GetRequest PDU

The manager uses the GetRequest PDU to retrieve the value of one or more object(s) from an agent. In most cases, these are scalar, not columnar, objects. To generate the GetRequest PDU, the manager assigns PDU Type = 0, specifies a locally defined Request ID, and sets both the ErrorStatus and ErrorIndex to 0. A VarBindList, containing the requested variables and corresponding NULL (placeholder) values, completes the PDU. Under error-free conditions, the agent generates a GetResponse PDU, which is assigned

PDU Type = 2, the same value of Request ID, Error Status = noError, and Error Index = 0. The Variable Bindings now contain the values associated with each of the variables noted in the GetRequest PDU (Figure 4.6). Recall that the term variable refers to an instance of a managed object.

4.3.3 Using the GetNextRequest PDU

The manager uses the GetNextRequest PDU to retrieve one or more objects and their values from an agent. In most cases, these multiple objects will reside within a table. As in Figure 4.7, to generate the GetNextRequest PDU the manager assigns PDU Type = 1, specifies a locally defined Request ID, and sets both the ErrorStatus and the ErrorIndex to 0. A VarBindList, containing the OIDs and corresponding NULL (placeholder) values, completes the PDU. These OIDs can be any OID (which may be a variable) that immediately precedes the variable and value returned. Under error-free conditions, the agent generates a GetResponse PDU, which is assigned PDU Type = 2, the same value of Request ID, Error Status = noError, and Error Index = 0. The Variable Bindings contain the name and value associated with the lexicographical successor of each of the OIDs noted in the GetNextRequest PDU.

4.3.4 Using the SetRequest PDU

The manager uses the SetRequest PDU to assign a value to an object residing in the agent. As you can see in Figure 4.8, to generate that PDU the manager assigns PDU Type = 3, specifies a locally defined Request ID, and sets both the ErrorStatus and ErrorIndex to 0. A VarBindList, containing the specified variables and their corresponding values, completes the PDU. When the agent receives the SetRequest PDU, it alters the values of the named objects to the values in the variable binding. Under error-free conditions, the agent generates a GetResponse PDU of identical form, except that the assigned PDU Type = 2, Error Status = noError, and Error Index = 0

4.3.5 The Trap PDU Format

The Trap PDU has a format distinct from the four other SNMP PDUs, as in Figure 4.9. The first field indicates the Trap PDU and contains PDU Type = 4. The Enterprise field

identifies the management enterprise under whose registration authority the trap was defined. For example, the OID prefix {1.3.6.1.4.1.110} would identify Network General Corp. as the Enterprise sending a trap. The Agent Address field, which contains the IP address of the agent, provides further identification. If a non-IP transport protocol is used, the value 0.0.0.0 is returned. The Generic Trap type provides more specific information on the event being reported.

4.3.6 Using the Trap PDU

The agent uses the Trap PDU to alert the manager that a predefined event has occurred. To generate the Trap PDU, the agent assigns PDU Type = 4 and fills in the Enterprise, Agent Address, Generic Trap, Specific Trap Type, and Timestamp fields, as well as the Variable Bindings list. By definition (and convention), Traps are application specific. Therefore, it would be difficult to cover the range of uses for this PDU. Figure 4.10 illustrates how an agent in a router could use a Trap to communicate a significant event to the manager.

4.3.7 SNMP PDU Encoding

SNMP PDUs are encoded using the context-specific class, with a tag that identifies the PDU. The Length and Value fields are then constructed to convey a particular structure and quantity of information. Now that we have discussed the structure of the SNMP PDUs, we can revisit these encodings in more detail.

Figure 4.11 shows an example of a TLV encoding of an SNMP PDU. Note that the entire encoding begins with a SEQUENCE OF type. The version is an INTEGER type, and the community name is an OCTET STRING type. A context-specific type then indicates the specific PDU and its length. Three INTEGER types provide the Request ID, Error Status, and Error Index. The VarBind list, consisting of multiple SEQUENCE OF encodings, completes the PDU. The following examples illustrate the details of this encoding structure.

4.4 Application Examples

To illustrate the SNMP PDUs discussed in this chapter, this section presents some examples of the protocol in use. The network analyzer captured each sample from an Ethernet backbone, which contained several other Ethernet segments connected by bridges and routers (Figure 4.12). For these cases, the SNMP manager was a Sun workstation running SunNet Manager, and a Proteon router contained the SNMP agent. In all of these examples, the traces are filtered to show only the SNMP protocol interaction.

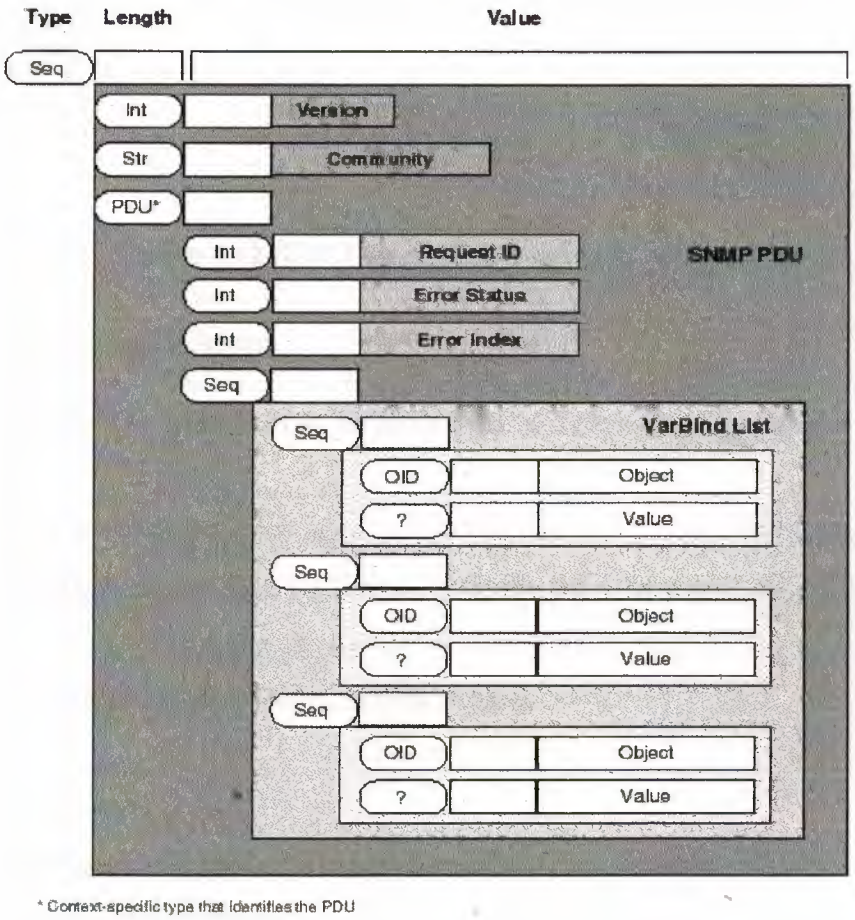


Figure 4.11 TLV encoding of an SNMP PDU *(Courtesy Network General Corp.)*

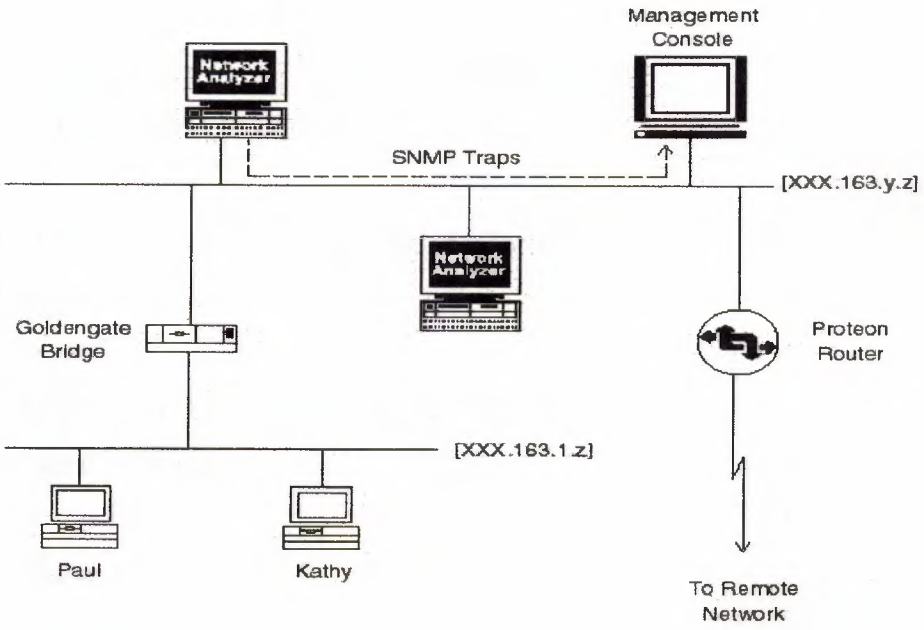


Figure 4.12 SNMP traps from a network analyzer

4.4.1 SNMP GetRequest Example

GetRequest PDU retrieves one or more objects. Trace 4.4.1 illustrates how the UDP group does this.

Trace 4.4.1. Retrieving scalar data using the GetRequest PDU: The UDP Group

Sniffer Network Analyzer data 10-Nov at 11:03:08, file UDP.ENC, Pg 1

---------- Frame 61 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.1.0} (udpInDatagrams.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.2.0} (udpNoPorts.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.3.0} (udpInErrors.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.4.0} (udpOutDatagrams.0)

SNMP: Value = NULL

SNMP:

---------- Frame 62 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 263748621 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.1.0} (udpInDatagrams.0)

SNMP: Value = 573894 datagrams

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.2.0} (udpNoPorts.0)

SNMP: Value = 419103 datagrams

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.3.0} (udpInErrors.0)

SNMP: Value = 0 datagrams

SNMP:

SNMP: Object = {1.3.6.1.2.1.7.4.0} (udpOutDatagrams.0)

SNMP: Value = 288892 datagrams

SNMP:

Trace 4.4.1 consists of two SNMP PDUs: the GetRequest (Frame 61) and the GetResponse (Frame 62). Both frames illustrate their respective PDU structures: Version = 0, Community = Brutus, Command (PDU Type 0 or 2), Request ID = 0, Error Status = 0, and Error Index = 0. Next, the VarBindList indicates the variables and associated values being requested or supplied.

We can observe two things here. First, the SunNet Manager always asks for the sysUpTime before requesting other objects. (Other management consoles may construct the VarBindList in another fashion.) The sysUpTime provides a time-stamp update for the Sun console, and is an input to the Sun graphical display of the network management statistics. Second, the values associated with the objects in the GetRequest have a Value = NULL. Recall that NULL is the ASN.1 type used as a placeholder in the data stream. When you look at the GetResponse in Frame 62, you'll see that each NULL value has been replaced with a measured value. For example, the number of UDP datagrams that have been delivered to UDP users, udpInDatagrams {1.3.6.1.2.1.7.1.0}, has a value of 573,894 datagrams.

4.4.2 SNMP SetRequest Example

This example issues a SetRequest PDU for an object on the Proteon router, then issues a GetRequest for the same object (Trace 4.4.3) to verify that the action was properly completed. Frames 1 and 2 retrieve the current value of ipDefaultTTL; Frames 3 and 4 set a new value for that object; Frames 5 and 6 verify the new value; Frames 7 and 8 set the value back to the original; and finally, Frames 9 and 10 verify the previous operation. The GetResponse PDU (Frame 2) contains the requested value (60) of ipDefaultTTL (the default value of the Time-to-Live field within the IP header). Frame 3 contains a

SetRequest PDU, assigning Value = 64 to ipDefaultTTL. The router sends a confirming GetResponse PDU in Frame 4. Frame 5 issues a GetRequest PDU to verify that the SetRequest changed the value of ipDefaultTTL to 64 (Frame 6). Frame 7 issues a second SetRequest, this time with Value = 60, which is acknowledged in Frame 8. Frames 9 and 10 confirm that the operation was successful.

Trace 4.4.2. SNNP Set ipDefaultTTL details

Sniffer Network Analyzer data 11-Dec at 15:16:52 file SETIPTTL.ENC

Pg 1

---------- Frame 1 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = NULL

SNMP:

---------- Frame 2 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 16862273 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)

SNMP: Value = 1 (gateway)

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 60

SNMP:

---------- Frame 3 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Set request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 64

SNMP:

---------- Frame 4 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 64

SNMP:

---------- Frame 5 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = NULL

SNMP:

---------- Frame 6 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 16863228 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)

SNMP: Value = 1 (gateway)

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 64

SNMP:

---------- Frame 7 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Set request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 60

SNMP:

---------- Frame 8 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = 60

SNMP:

---------- Frame 9 ----------

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = Brutus

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)

SNMP: Value = NULL

SNMP:

```
------------------------------ Frame 10 ------------------------
SNMP: ----- Simple Network Management Protocol -----
SNMP:
SNMP: Version = 0
SNMP: Community = Brutus
SNMP: Command = Get response
SNMP: Request ID = 0
SNMP: Error status = 0 (No error)
SNMP: Error index = 0
SNMP:
SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)
SNMP: Value  = 16863846 hundredths of a second
SNMP:
SNMP: Object = {1.3.6.1.2.1.4.1.0} (ipForwarding.0)
SNMP: Value  = 1 (gateway)
SNMP:
SNMP: Object = {1.3.6.1.2.1.4.2.0} (ipDefaultTTL.0)
SNMP: Value  = 60
SNMP:
```

4.4.3 SNMP Trap Example

The final example shows how a Trap PDU indicates an alarm condition to the network manager. In this case, the agent generating the trap is a Network General Sniffer protocol analyzer (Figure 4.12). One set of network statistics is network utilization. Network utilization is a ratio between the total number of bits transmitted in a period of time (in this case five seconds) divided by the total number of bits that could theoretically be transmitted during the same period. A typical network would have a network utilization in the 5 to 20 percent range. For this example, we set the threshold to the unrealistically low value of 1 percent over a five second period. When the network reaches that threshold, the Sniffer generates a Trap PDU and sends it to the SunNet Manager. Another Sniffer analyzer captured the results.

Trace 4.4.3. An enterprise-specific trap: Network utilization exceeded 1 percent during a five second period.

```
Sniffer Network Analyzer data 11-Dec at 16:13:26 file SNIFTRAP.ENC
Pg 1
------------------------------ Frame 1 ------------------------
SNMP: ----- Simple Network Management Protocol -----
SNMP:
SNMP: Version = 0
SNMP: Community = public
SNMP: Command = Trap
SNMP: Enterprise = {1.3.6.1.4.1.110.1.1.1.0}
SNMP: Network address = [132.163.128.102]
SNMP: Generic trap = 6 (Enterprise specific)
SNMP: Specific trap = 7
SNMP: Time ticks = 244894900
SNMP:
SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.1.1.1}
          (Network General Corp.1.1.1.1.1.1.1.1)
SNMP: Value  = 53 (counter)
SNMP:
SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.1.2.1}
          (Network General Corp.1.1.1.1.1.1.2.1)
SNMP: Value  = 1
SNMP:
SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.1.3.1}
          (Network General Corp.1.1.1.1.1.1.3.1)
SNMP: Value  = Abs usage exceeded 1%
SNMP:
SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.1.4.1
          (Network General Corp.1.1.1.1.1.1.4.1)
SNMP: Value  = 5
```

SNMP:

SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.5.1}

(Network General Corp.1.1.1.1.1.1.5.1)

SNMP: Value = 0

SNMP:

SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.6.1}

(Network General Corp.1.1.1.1.1.1.6.1)

SNMP: Value = 7

SNMP:

SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.7.1}

(Network General Corp.1.1.1.1.1.1.7.1)

SNMP: Value = 724119640 (counter)

SNMP:

SNMP: Object = {1.3.6.1.4.1.110.1.1.1.1.1.8.1}

(Network General Corp.1.1.1.1.1.1.8.1)

SNMP: Value = Global Network

SNMP:

4.5 The ASN.1 SNMP Definition

To conclude the discussion of SNMP protocol operation, Definition 4.1 is the ASN.1 definition of SNMP (RFC 1157). Of special interest are the constructs of the various SNMP PDUs. Those constructs summarize the variables used within the PDUs, plus the values that those variables may assume.

Definition 4.1 The ASN.1 definition of SNMP

```
RFC1157-SNMP DEFINITIONS ::= BEGIN
IMPORTS
   ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
      FROM RFC1155-SMI;
-- top-level message
    Message ::=
       SEQUENCE {
          version              -- version-1 for this RFC
             INTEGER {
```

```
                version-1(0)
            },
        community           -- community name
            OCTET STRING,
        data                -- e.g., PDUs if trivial
            ANY             -- authentication is being used
    }
-- protocol data units
PDUs ::=
  CHOICE {
        get-request
            GetRequest-PDU,

        get-next-request
            GetNextRequest-PDU,
        get-response
            GetResponse-PDU,
        set-request
            SetRequest-PDU,
        trap
           Trap-PDU
        }
-- PDUs
GetRequest-PDU ::=
   [0]
      IMPLICIT PDU
GetNextRequest-PDU ::=
   [1]
      IMPLICIT PDU
GetResponse-PDU ::=
   [2]
      IMPLICIT PDU
SetRequest-PDU ::=
   [3]
      IMPLICIT PDU
PDU ::=
      SEQUENCE {
         request-id
```

```
        INTEGER,
      error-status          -- sometimes ignored
        INTEGER {
          noError(0),
          tooBig(1),
          noSuchName(2),
          badValue(3),
          readOnly(4),
          genErr(5)
        },
      error-index           -- sometimes ignored
        INTEGER,
      variable-bindings     -- values are sometimes ignored
        VarBindList
    }
Trap-PDU ::=
    [4]
    IMPLICIT SEQUENCE {
      enterprise            -- type of object generating
                            -- trap, see sysObjectID in [5]
      OBJECT IDENTIFIER,
      agent-addr            -- address of object generating
        NetworkAddress,     -- trap
      generic-trap                -- generic trap type
        INTEGER {
          coldStart(0),
          warmStart(1),
          linkDown(2),
          linkUp(3),
          authenticationFailure(4),
          egpNeighborLoss(5),
          enterpriseSpecific(6)
        },
      specific-trap         -- specific code, present even
        INTEGER,            -- if generic-trap is not
                           -- enterpriseSpecific
      time-stamp            -- time elapsed between the last
        TimeTicks,          -- (re-)initialization of the
```

```
                                    network
                                -- entity and the generation of
                                    the trap
            variable-bindings    -- "interesting" information
               VarBindList
      }
  -- variable bindings
  VarBind ::=
     SEQUENCE {
        name
          ObjectName,
        value
          ObjectSyntax
     }
  VarBindList ::=
     SEQUENCE OF
        VarBind
END
```

CHAPTER FIVE

SNMP VERSION 2

The original version of SNMP (SNMPv1) was derived from the Simple Gateway Monitoring Protocol (SGMP) and published as an RFC in 1988. At that time, the industry agreed that SNMP would be an interim solution until OSI-based network management using CMIS/CMIP became more mature. Since then, however, SNMP has become more popular while the OSI solution has been less widely adopted than was anticipated originally. As a result, it became appropriate to revise and improve SNMPv1.

5.1 The SNMPv2 Structure of Management Information

MIB modules provide a mechanism for grouping similar objects. The SMI for SNMPv2 defines the subset of the ASN.1 language that describes various MIB modules. SNMPv2 has two documents that support the SMI: the Conformance Statements and the Textual Conventions. The Textual Conventions define the data types used within these MIB modules and make it easier to read the modules. The conformance statements provide an implementation baseline and include, for example, a lower bound on what agents must support. The SMI also defines two new branches of the Internet OID tree: security {1.3.6.1.5} and snmpV2 {1.3.6.1.6}. Under snmpV2 are the Transport domains (snmpDomains); Transport proxies (snmpProxys); and Module identities (snmpModules). Defined under the snmpModules are the SNMPv2 MIB (snmpMIB); the Manager to Manager MIB, (snmpM2M); and the Party MIB (partyMIB). Figure 5.1 illustrates the positions of these new elements of the OID tree.

The SMI is divided into three parts:

- Module definitions which are used to describe information modules, such MIB modules, compliance statements for MIB modules and capability statements for agent implementations
- Object Definitions, which are used to describe managed objects
- Notification definitions, which are used to describe unsolicited transmissions of management information, such as traps.

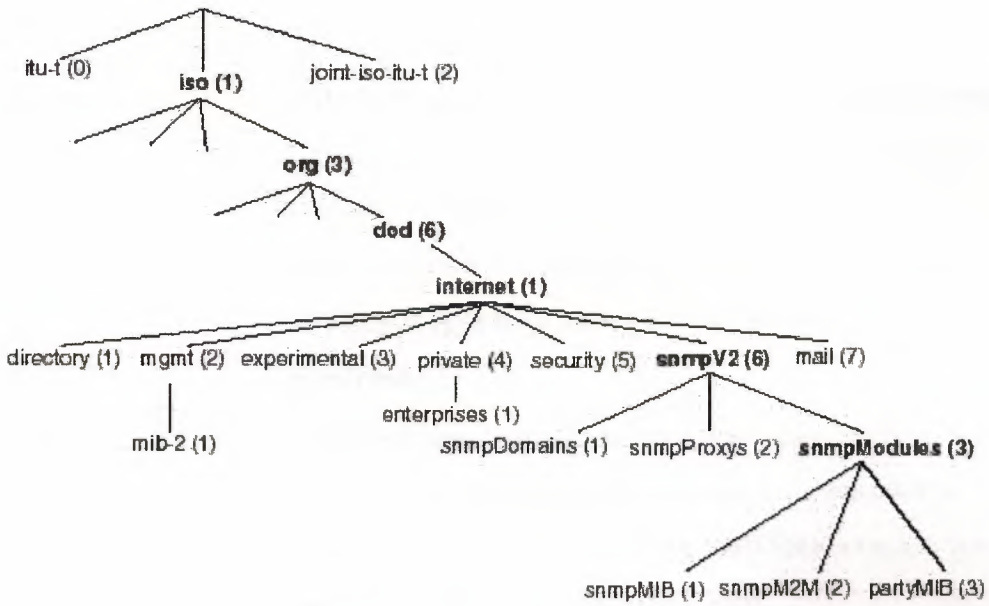


Figure 5.1 SNMPv2 elements within the OID tree

5.1.1 SNMPv2 SMI Module Definitions

Module definitions are used when describing information modules. An ASN.1 macro, called MODULE-IDENTITY, is used to convey the semantics of an information module. More specifically, it conveys the contact and revision history for each information module, using the following clauses: LAST-UPDATED, ORGANIZATION, CONTACT-INFO, DESCRIPTION, and REVISION.

5.1.2 SNMPv2 Object Definitions

The object definitions in SNMPv2 are enhanced from SNMPv1 through the use of the new OBJECT-IDENTITY module, some new data types that were not used in SNMPv1, and a revised OBJECT-TYPE module. The OBJECT-IDENTITY module is used to define information about an OBJECT IDENTIFIER assignment. This module includes the following clauses: STATUS, DESCRIPTION, and REFERENCE.

New SNMPv2 data types include:

Data Type	Description
Integer32	A defined type that represents integer-valued information between -2^{31} and $2^{31}-1$ inclusive (-2147483648 and 2147483647 decimal). (Note: This type is indistinguishable from the INTEGER type, although the INTEGER type may have different numerical constraints.)
Counter32	A defined type that represents a non-negative integer that monotonically increases until it reaches a maximum value of $2^{32}-1$ (4294967295 decimal) then wraps around and starts increasing again from zero.
Counter64	A defined type that represents a nonnegative integer that monotonically increases until it reaches a maximum value of $2^{64}-1$ (18446744073709551615 decimal), then wraps around and starts increasing again from zero. Counter64 is used for objects for which the 32-bit counter (Counter32) is too small, or which would wrap around too quickly. RFC 1902 states that the Counter64 type may be used only if the information being modeled would wrap in less than one hour using the Counter32 type.
Unsigned32	A defined type that represents integer-valued information between 0 and 232-1 (4294967295 decimal), inclusive.
Gauge32	A defined type that represents a nonnegative integer which may increase or decrease, but which never exceeds a maximum value ($2^{32}-1$, as above).
BITS	A construct which represents an enumeration of named bits.

5.1.3 SNMPv2 SMI Notification Definitions

The SNMPv2 SMI's new NOTIFICATION-TYPE macro defines the information contained within the unsolicited transmission of management information. This includes an SNMPv2-Trap-PDU or an Inform-Request-PDU. SMIv2 also references three other new ASN.1 macros: MODULE-COMPLIANCE, OBJECT-GROUP, and AGENT-CAPABILITIES.

5.2 SNMPv2 Conformance Statements

The Conformance Statements are used to define acceptable lower bounds of implementation, along with the actual level of implementation for SNMPv2 that is achieved by the device. The Conformance Statements document, RFC 1904, defines the notations, along with ASN.1 macros, that are used for these purposes. Two kinds of notations are used:

- *Compliance statements,* which describe requirements for agents with respect to object definitions. The MODULE-COMPLIANCE macro is used to convey a minimum set of requirements with respect to implementation of one or more MIB modules. In other words, the MODULE-COMPLIANCE macro conveys a minimum conformance specification, including objects and groups required, which may come from different MIB modules.
- *Capability statements,* which describe the capabilities of agents with respect to object definitions. The AGENT-CAPABILITIES macro describes the capabilities of an SNMPv2 agent. It defines the MIB modules, objects, and values implemented within the agent. A description of the precise level of support that an agent claims is bound to the instance of the sysORID object. (See the SNMPv2 MIB, RFC 1907, for a complete definition of the sysORID object and other objects that convey object resource information.)

5.3 SNMPv2 Protocol Operations

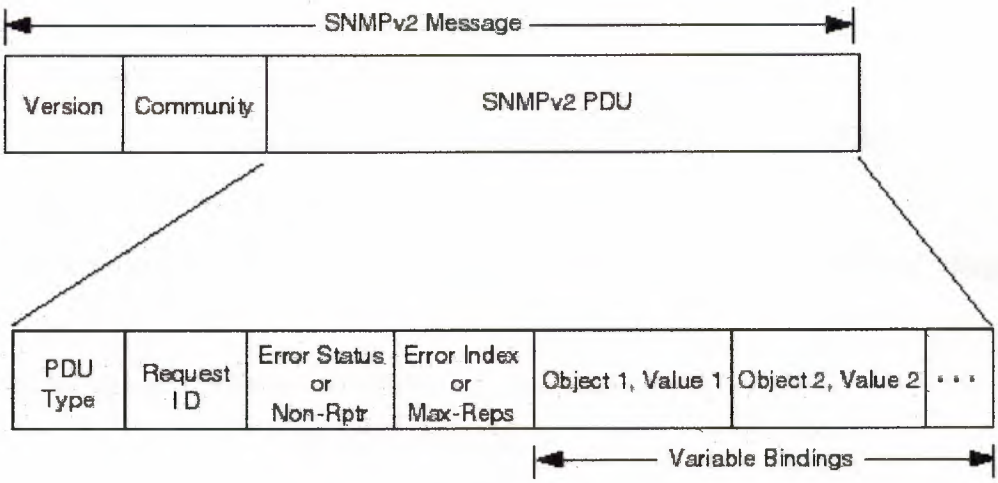
When it comes to processing protocol messages, an SNMPv2 entity may act as an agent, a manager, or both. The entity acts as an agent when it responds to protocol messages (other than the Inform notification, which is reserved for managers) or when it

sends Trap notifications. The entity acts as a manager when it initiates protocol messages or responds to Trap or Inform notifications. The entity may also act as a proxy agent. SNMPv2 provides three types of access to network management information: these types are determined by the network management entity's role and relate to the Manager-to-Manager capabilities. The first type of interaction, called request-response, is where an SNMPv2 manager sends a request to an SNMPv2 agent, which responds. The second type of interaction is a request-response where both entities are SNMPv2 managers. The third type is an unconfirmed interaction, where an SNMPv2 agent sends an unsolicited message, or trap, to the manager and no response is returned. SNMPv2 has significantly enhanced the PDUs that convey this management information (Figure 5.2). SNMPv2 offers new PDUs and adds error codes and exception responses. The latter allows a management application to easily determine why a management operation failed.

5.3.1 SNMPv2 PDUs

SNMPv2 defines eight PDU types, of which three are new: the GetBulkRequest, the InformRequest, and the Report. In addition, the SNMPv2-Trap PDU format has been revised from the SNMPv1 Trap to conform to the format and structure of the other PDUs (In SNMPv1, the Trap PDU had a unique format).
The following is a list of all the SNMPv2 PDUs, along with their assigned tag numbers:

PDU/Tag Number	Description
GetRequest [0]	Retrieves values of objects listed within the variable bindings field.
GetNextRequest [1]	Retrieves values of objects that are the lexicographical successors of the variables, up to the end of the MIB view of this request.



Wrapper : Contains authentication and privacy information

PDU Type : Specifies the PDU being transmitted:
0 = GetRequest
1 = GetNextRequest
2 = Response
3 = SetRequest
4 = obsolete
5 = GetBulkRequest
6 = InformRequest
7 = SNMPv2–Trap
8 = Report

Request ID : Used to correlate the Request and Response

Error Status : Exception Condition for the request. Values are:
0 = noError
1 = tooBig
2 = noSuchName
3 = badValue
4 = readOnly
5 = genErr
6 = noAccess
7 = wrongType
8 = wrongLength
9 = wrongEncoding
10 = wrongValue
11 = noCreation
12 = inconsistentValue
13 = resourceUnavailable
14 = commitFailed
15 = undoFailed
16 = authorizationError
17 = notWritable
18 = inconsistentName

Error Index : Pointer to the Variable Binding that caused the error

Non-Rptr : Non-Repeaters, how many of the requested variables will not be processed repeatedly, e.g. single instances of variables. Used in GetBulkRequests only.

Max-Reps : Maximum-Repetitions, the maximum number of repeated executions to retrieve specific variables. Used in GetBulkRequests only.

Variable Bindings : Pairing of object name and value

Figure 5.2 SNMPv2 PDU structure

Response [2]	Generated in response to a GetRequest, GetNextRequest, GetBulkRequest, SetRequest, or InformRequest PDU.
SetRequest [3]	Establishes the value of a variable.
GetBulkRequest [5]	Retrieves a large amount of data, such as the contents of a large table.
InformRequest [6]	Allows one manager to communicate information in its MIB view to another manager.
SNMPv2-Trap [7]	Used by an SNMPv2 agent to provide information regarding an exceptional condition. The Trap PDU, defined for SNMPv1 with tag [4], is now considered obsolete. The Coexistence document, RFC 1908, discusses conversion from the Trap PDU to the SNMPv2-Trap PDU.
Report [8]	Included in SNMPv2, but its usage is not defined in RFC 1905. It is expected that any Administrative Framework that makes use of this PDU would define its usage and semantics (see RFC 1905, page 6).

The PDUs that the SNMPv2 entity generates or receives depend on the entity's role as an agent or manager:

SNMPv2 PDU	Agent Generate	Agent Receive	Manager Generate	Manager Receive
GetRequest		x	x	
GetNextRequest		x	x	
Response	x		x	x
SetRequest		x	x	
GetBulkRequest		x	x	
InformRequest			x	x

SNMPv2-Trap	x			x	

5.3.2 SNMPv2 PDU syntax

The SNMPv2 message consists of the wrapper that encapsulates an SNMPv2 PDU. The wrapper is determined by the administrative framework and may contain the authentication and privacy information. The syntax of the SNMPv2 PDUs is similar to the structures defined in SNMPv1. Significant enhancements include error status codes that detail why protocol operations were unsuccessful. The PDU consists of four fields—the PDU Type field, the Request ID field, the Error Status field, and the Error Index field (Figure 5.2)—plus the variable bindings. The PDU Type field specifies which one of the eight PDUs is being transmitted. The Request ID correlates the request and response PDUs. The Error Status field includes new exception conditions. When errors occur in the processing of the GetRequest, GetNextRequest, GetBulkRequest, SetRequest, or InformRequest PDUs, the SNMPv2 entity prepares a Response PDU with the Error Status field set to help the manager identify and correct the problem. The following table shows how the PDUs use these error codes:

SNMPv2 Error	Get	GetNext	GetBulk	Set	Inform
noError	x	x	x	x	x
tooBig	x	x		x	x
noSuchName[1]					
badValue[1]					
readOnly[1]					
genErr	x	x	x	x	
noAccess				x	
wrongType				x	
wrongLength				x	

wrongEncoding				x	
wrongValue				x	
noCreation				x	
inconsistentValue				x	
resourceUnavailable				x	
commitFailed				x	
undoFailed				x	
authorizationError	x[2]	x[2]	x[2]	x[2]	x[2]
notWritable				x	
inconsistentName				x	

The Error Index field is used with the Error Status code. When errors occur in the processing of the variable bindings, the Error Index field identifies the binding that caused the error. An error in the first binding would have Index = 1, an error in the second binding would have Index = 2, and so on.

5.4 SNMPv2 Transport Mappings

SNMP version 1 was originally defined for transmission over UDP and IP. Subsequent research explored the use of SNMP with other transport protocols, including OSI transport, AppleTalk's Datagram Delivery Protocol (DDP), and Novell's Internetwork Packet Exchange (IPX). SNMPv2 formally defines implementations over these other transports in the Transport Mapping document, RFC 1906 (Figure 5-3).

5.4.1 SNMPv2 over UDP

SNMPv2 over UDP is the preferred transport mapping. UDP provides compatibility with SNMPv1 at both the Transport and Network layers, although other higher-layer issues, such as SNMPv2 PDU structures, remain. RFC 1906 also suggests that SNMPv2 agents continue the practice of listening on UDP port 161, and that notifications listen on UDP port 162. (UDP port 162 was previously defined for SNMP traps.) Figure 5.4 illustrates the

details of the UDP header, which precedes the SNMP message within the transmitted frame.

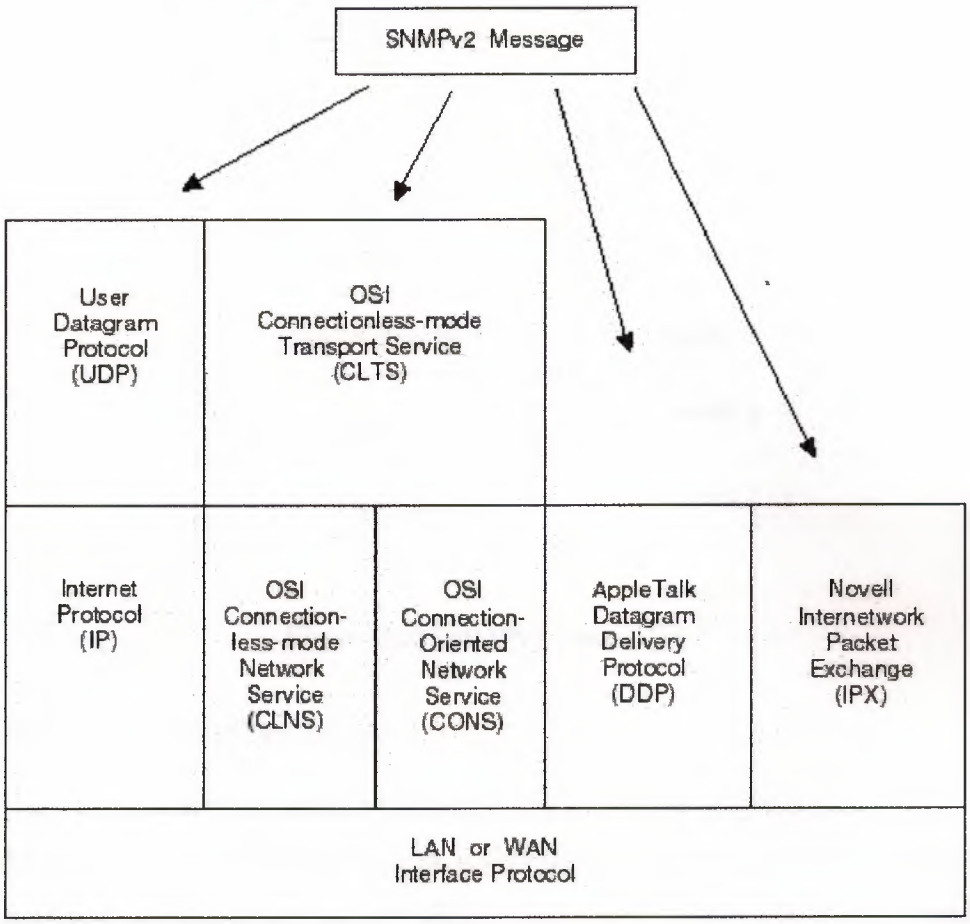


Figure 5.3 Transport mappings for SNMPv2

5.4.2 SNMPv2 over OSI

RFC 1449 defines two options for transmitting SNMPv2 messages over OSI protocols. Both send the SNMPv2 message in a single transport service data unit (TSDU) using the provisions of the OSI Connectionless-mode Transport Service (CLTS). Then at the Network layer, either a Connectionless-mode Network Service (CLNS) or a Connection-oriented Network Service (CONS) may be used.

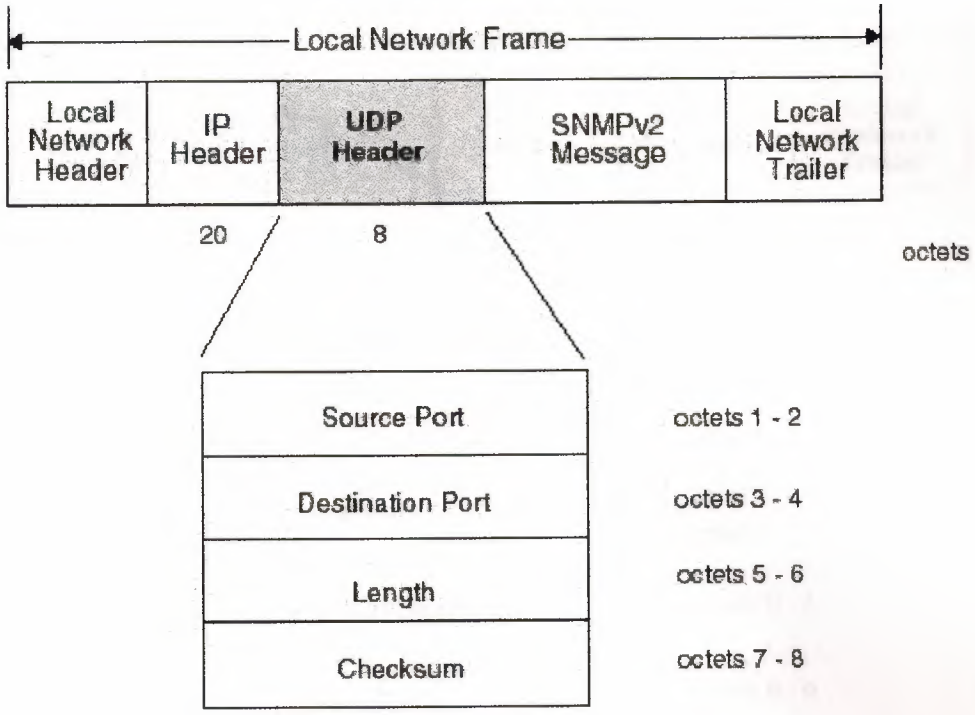


Figure 5.4 SNMPv2 over UDP

5.4.3 SNMPv2 over AppleTalk DDP

Apple Computer's AppleTalk protocol suite is another option available for SNMPv2 transport. The SNMPv2 message is sent in a single Datagram Delivery Protocol (DDP) datagram, which operates at the OSI Network layer. Figure 5.6 shows the details of the DDP header and the position of the SNMPv2 message within the transmission frame. The final octet of the DDP header specifies the DDP Type, indicating the protocol in use. SNMPv2 messages use DDP Type = 8, since Apple has previously defined types 1 through 7. Other DDP parameters, such as socket numbers, are also defined for SNMPv2 use. SNMPv2 entities acting in the agent role use DDP socket number 8; notification sinks, which are entities receiving a notification, use DDP socket number 9.

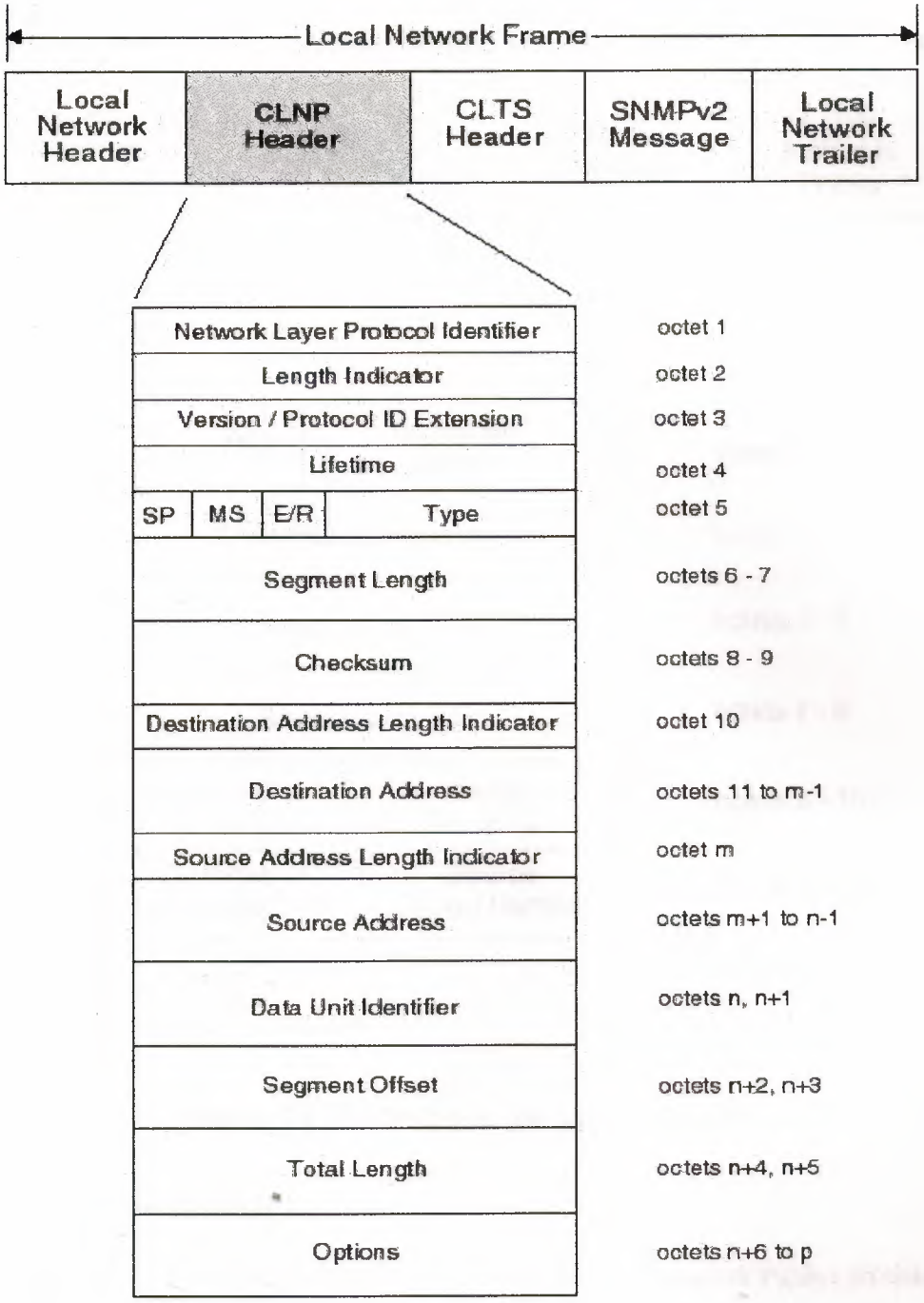


Figure 5.5 SNMPv2 over ISO CLNP

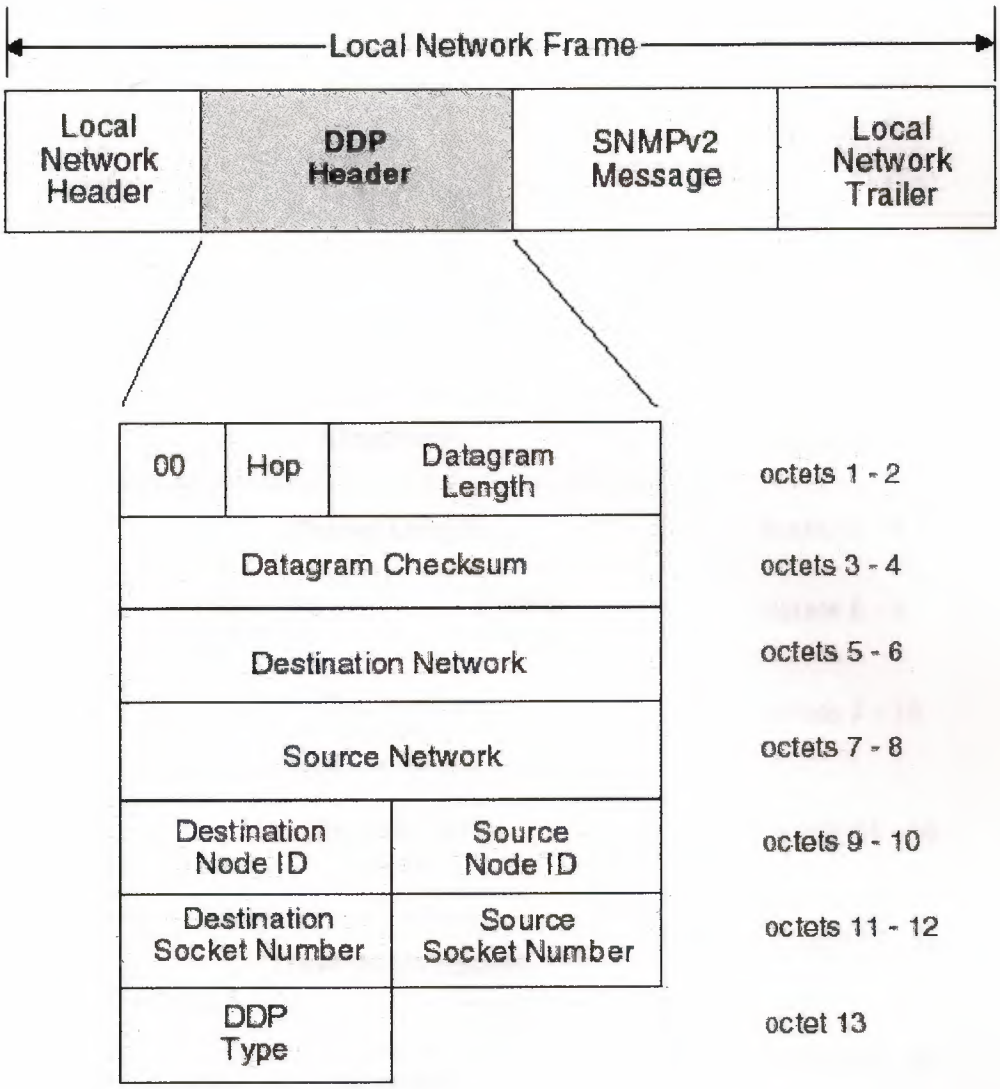


Figure 5.6 SNMPv2 over the AppleTalk DDP

5.4.4. SNMPv2 over Novell IPX

Novell Inc.'s NetWare protocol suite defines the Internetwork Packet Exchange (IPX) protocol at the Network layer. SNMPv2 messages are serialized into a single IPX datagram, as shown in Figure 5.7. Within the IPX header is a Packet Type parameter that specifies the protocol in use. SNMPv2 messages use Packet Type = 4, which is defined as a Packet Exchange Protocol packet. SNMPv2 entities acting in the agent role listen on IPX socket number 36879 (900FH), while notification sinks listen on socket 36880 (9010H).

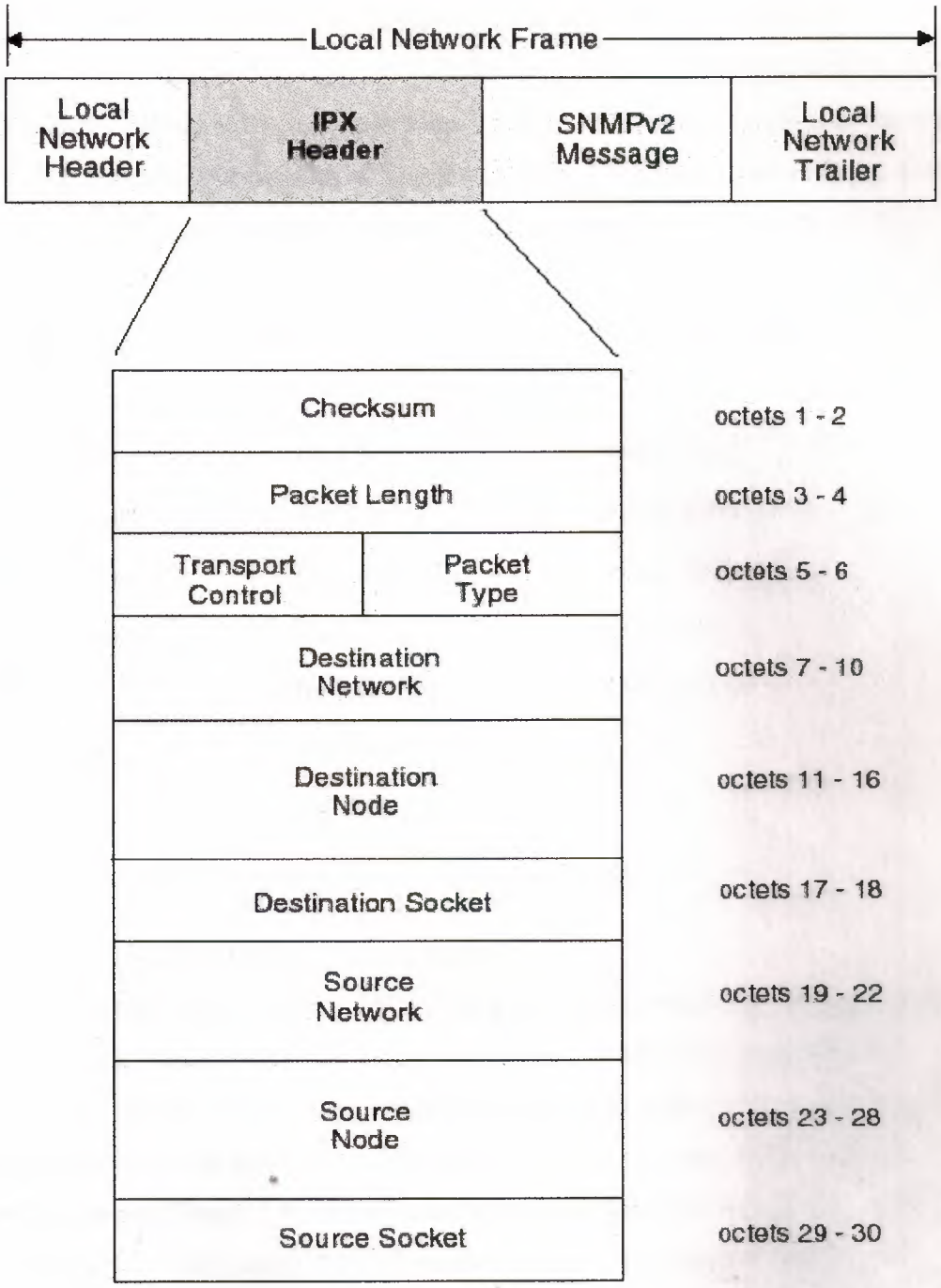


Figure 5.7 SNMPv2 over Novell IPX

5.5 The SNMPv2 MIB

The April 1993 version of SNMPv2 (RFCs 1441–1452) provided three MIB documents. The first, RFC 1450, described a MIB module for SNMPv2 objects, which was identified by {snmpModules 1}. The second, RFC 1451, coordinated multiple management

stations and was therefore called the Manager-to-Manager MIB, identified as {snmpModules 2}. The third module, RFC 1447, supported the SNMPv2 security protocols and was called the Party MIB {snmpModules 3}. With the removal of the security-related aspects in the January 1996 version of SNMPv2 (RFCs 1901-1908), the MIB required revision as well. The fundamental structure is still the same, however (Figure 5.1):

Branch	OID	RFC References
snmpV2	{ 1.3.6.1.6 }	1442, 1902
snmpDomains	□ 1.3.6.1.6.1 }	1442, 1902, 1906
snmpProxys	{ 1.3.6.1.6.2 }	1442, 1902, 1906
snmpModules	{ 1.3.6.1.6.3 }	1442, 1902
snmpMIB	{ 1.3.6.1.6.3.1 }	1450, 1907
snmpM2M	{ 1.3.6.1.6.3.2 }	1451
partyMIB	{ 1.3.6.1.6.3.3 }	1447

5.6 Coexistence of SNMPv1 and SNMPv2

The Coexistence document, RFC 1908, presents a number of guidelines that outline the modifications necessary for successful coexistence of SNMPv1 and SNMPv2. From a practical point of view, two methods are defined to achieve coexistence: a proxy agent and a bilingual manager. The proxy agent translates between SNMPv1 to/from SNMPv2 messages (Figure 5.8). When translating from SNMPv2 to SNMPv1, GetRequest, GetNextRequest, or SetRequest PDUs from the manager are passed directly to the SNMPv1 agent. GetBulkRequest PDUs are translated into GetNextPDUs. For translating from SNMPv1 to SNMPv2, the GetResponse PDU is passed unaltered to the manager. An SNMPv1 Trap PDU is mapped to an SNMPv2-Trap PDU, with the two new variable bindings, sysUpTime.0 and snmpTrapOID.0, prepended to the variable bindings field. The second alternative is a bilingual manager, which incorporates both the SNMPv1 and

SNMPv2 protocols. When the manager needs to communicate with an agent, it selects the protocol appropriate for the application

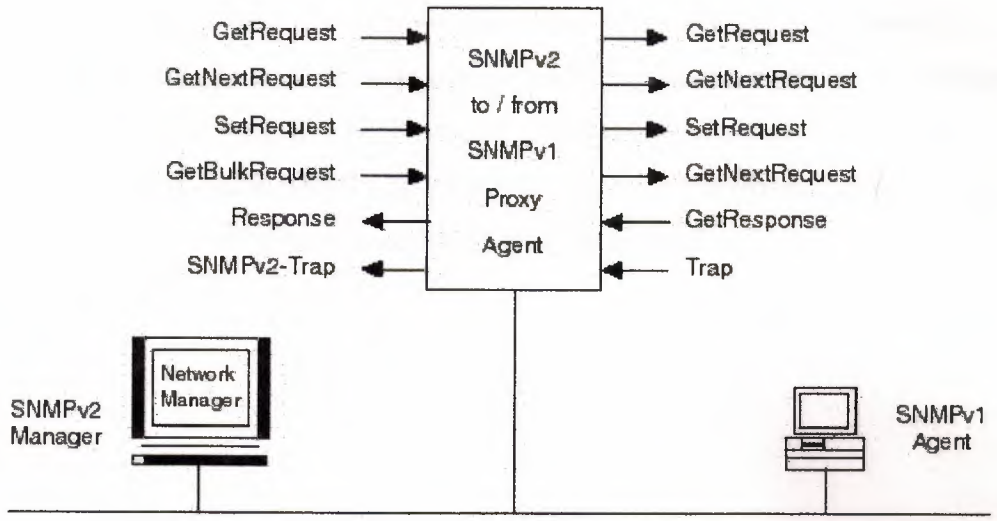


Figure 5.8 SNMPv1/SNMPv2 proxy agent operation

5.7 SNMPv2 Security

When SNMPv1 was first published, the community name and the version number in the SNMP header provided the only message security capabilities. This provision, known as the *trivial* protocol, assured that both agent and manager recognized the same community name before proceeding with network management operations. Additional research into security issues yielded three documents on the subject.

RFC	Title
1351	SNMP Administrative Model
1352	SNMP Security Protocols
1353	Definitions for Managed Objects for Administration of SNMP Parties

These RFCs were designed to address the authentication and privacy of network management communication. *Authentication* assures the appropriate origin of the message, while *privacy* protects the messages from disclosure. Unfortunately, implementing these enhancements proved to be more complex than either vendors or network managers anticipated; consequently, few products containing these improvements were developed. In addition, two alternatives have been proposed to address the security aspects in particular. The first is called SNMPv2U, which stands for a User-based security model. The second is called SNMPv2* (pronounced SNMP vee-two-star).

CHAPTER SIX

LOWER LAYER SUPPORT FOR SNMP

An underlying communication infrastructure is necessary for the manager and agent to communicate network management information. This infrastructure exists at the OSI Transport, Network, and Data Link layers, or at the ARPA Host-to-Host, Internet, and Network Interface layers. SNMP messages fit inside the OSI Data Link layer or ARPA Local Network layer frame. To send SNMP messages, the system requires the User Datagram Protocol (UDP) and the Internet Protocol (IP), as shown in Figure 6.1. Together, the SNMP message, plus UDP and IP headers, comprise an IP datagram. This chapter discusses these supporting protocols.

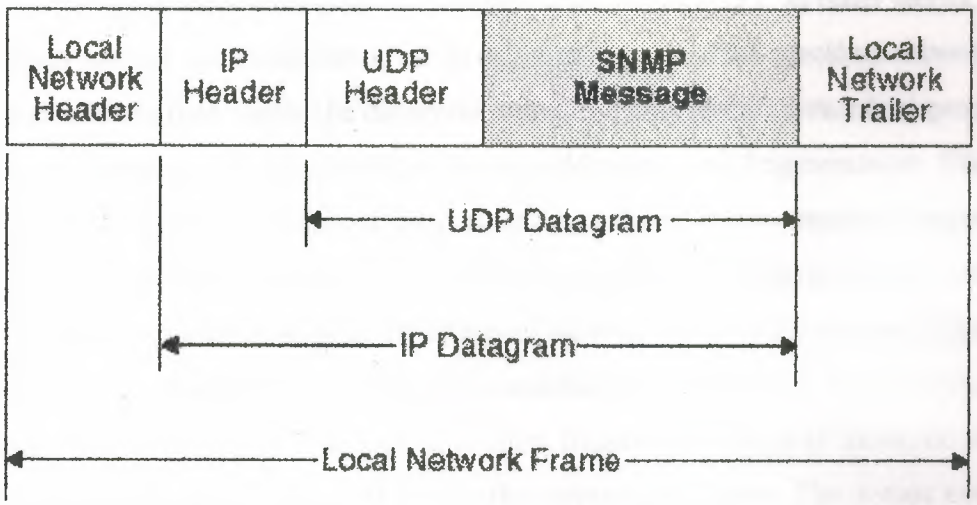


Figure 6.1 An SNMP message within a transmission frame

6.1 User Datagram Protocol (UDP)

UDP provides a connectionless host-to-host communication path for the SNMP message. A connectionless path is one in which the communication channel is not established prior to the transmission of data. Instead, the network transmits the data in a package called a datagram. The datagram contains all of the addressing information necessary for the SNMP message to reach its intended destination. The UDP service requires minimal overhead, and therefore uses the relatively small UDP header shown in Figure 6.2. Note in the figure that each horizontal group of bits, called a word, is

32 bits wide.

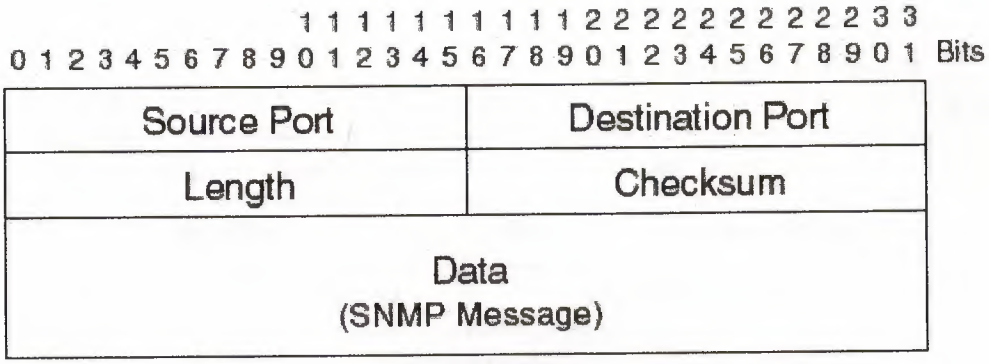


Figure 6.2 The User Datagram Protocol (UDP) header

6.2 Internet Protocol (IP)

IP works closely with UDP. IP handles datagram delivery. In other words, the IP destination address routes the datagram to the correct host on the specified network. The UDP port address then routes the datagram within the host to the correct host process. To deliver datagrams, IP deals with two issues: addressing and fragmentation. The *address* assures that the datagram arrives at the correct destination. *Fragmentation* is necessary because the sequence of LANs and WANs that any particular datagram may traverse can have differing frame sizes, and the IP datagram must fit within these varying frames (Figure 6.1). For example, if the endpoint is attached to an IEEE 802.3 LAN with a maximum data field size of 1500 octets, IP must fragment the large IP datagram into smaller pieces (fragments) that will fit into the constraining frame. The distant node then reassembles the fragments back into a single IP datagram.In Figure 6.3, the IP header contains at least 20 octets of control information.

6.3 Internet Addressing

Each 32-bit IP address is divided into Host ID and Network ID sections, and may take one of five formats, Class A through E addresses, as shown in Figure 6.4. The formats differ in the number of bits allocated to the Host and Network IDs and are identified by the first three bits. Class A addresses are designed for very large networks having many hosts; they are identified by Bit 0 = 0. Bits 1 through 7 identify the network, and Bits 8 through 31

identify the specific host on that network. With a seven-bit Network ID, only 128 class A addresses are available. Of these, addresses 0 and 127 are reserved.

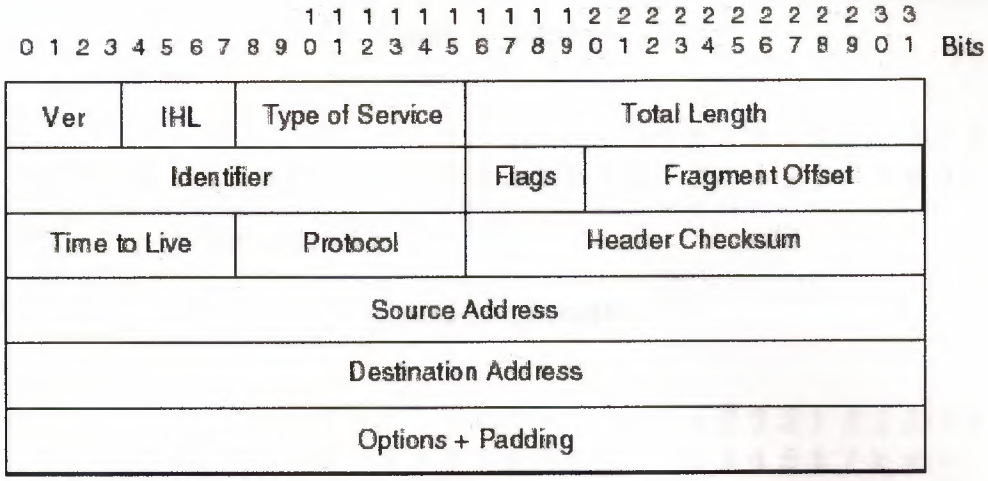


Figure 6.3 Internet Protocol (IP) header

Class B addresses are identified by the first two bits having a value of 10 (binary). The next 14 bits identify the Network and the remaining 16 bits identify the Host. As many as 16,384 Class B addresses are possible, with addresses 0 and 16,383 reserved. Class C addresses begin with a binary 110. The next 21 bits identify the Network, and the remaining 8 bits identify the Host. A total of 2,097,152 Class C addresses are possible, with addresses 0 and 2,097,151 reserved. Class D addresses begin with a binary 1110 and are intended for multicasting. Class E addresses begin with a binary 1111 and are reserved for future use. All IP addresses are written in *dotted decimal notation*, in which each octet is given a decimal number from 0 to 256. For example, network 10.56.31.84 is represented in binary as

00001010 00110111 00011111 1010100

- The first bit (0) indicates a Class A address.
- The next seven bits (0001010) represent the Network ID (decimal 10).
- The last 24 bits (00110111 00011111 1010100) represent the Host ID.

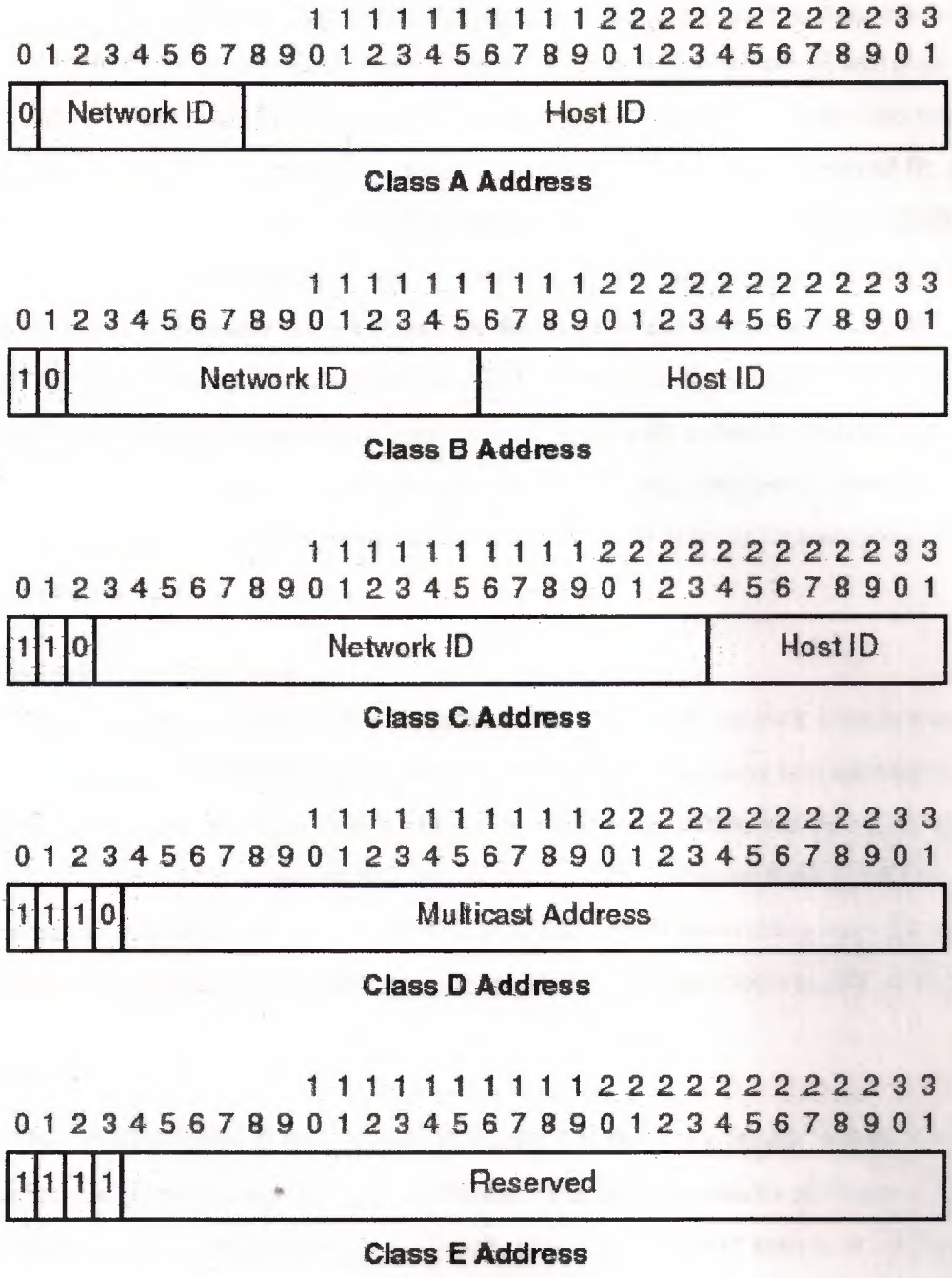


Figure 6.4 IP address formats

6.4 Internet Control Message Protocol (ICMP)

IP provides a connectionless service to the attached hosts but requires an additional module, known as the Internet Control Message Protocol (ICMP), to report any errors that may occur in the processing of those datagrams. Examples of errors would be undeliverable

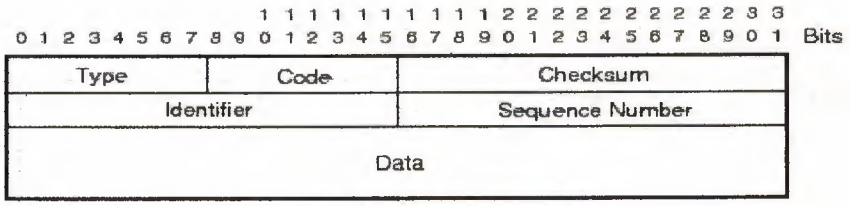
datagrams or incorrect routes. The protocol is also used to test the path to a distant host (known as a PING) or to request an address mask for a particular subnet. ICMP is an integral part of IP and must be implemented in IP modules contained in hosts and routers. IP datagrams contain ICMP messages. In other words, ICMP is a user (client) of IP, and the IP header precedes the ICMP message. The datagram would thus be IP header, ICMP header, and finally ICMP data. Protocol = 1 identifies ICMP within the IP header. A Type field within the ICMP header further identifies the purpose and format of the ICMP message. Any data required to complete the ICMP message follows the ICMP header. Thirteen ICMP message formats have been defined, each with a specific ICMP header format. Two of these formats (Information Request/Reply) are considered obsolete, and several others share a common message structure. The result is six unique message formats, as shown in Figure 6.5.

6.5 Network Interface Protocols

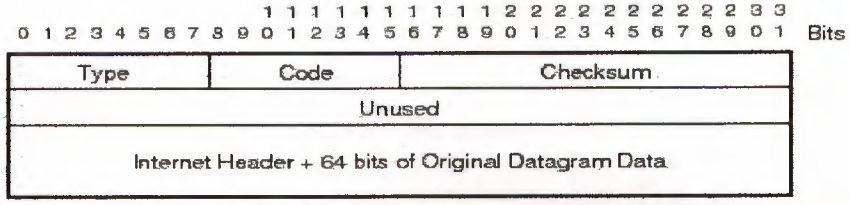
The lowest layer of the ARPA architectural model is the Network Interface layer, which encompasses the OSI Data Link and Physical layers. This layer is responsible for the network hardware and topology, such as Ethernet, token ring, FDDI, and so on. WAN protocols, such as dial-up or leased-line connections, X.25, or Frame Relay, can also be implemented at this layer. Because most SNMP implementations involve local, not remote manager/agent relationships, we will concentrate on the LAN protocols in this section.

6.5.1 Ethernet

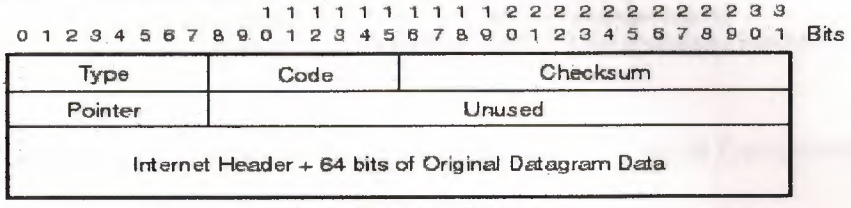
The Ethernet frame format, shown in Figure 6.6, defines a length between 64 and 1518 octets, including the header, data, and trailer. The header consists of Destination and Source addresses that are 6 octets (48 bits) each, plus a 2-octet field known as the Type (or Ethertype) field. The Ethernet-designated destination address for broadcast frames is all ONES (FFFFFFFFFFFFH). The type designates the higher-layer protocol in use within the Data field.



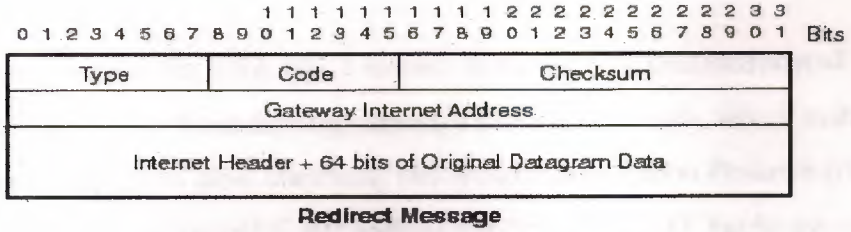
Echo and Echo Reply Messages



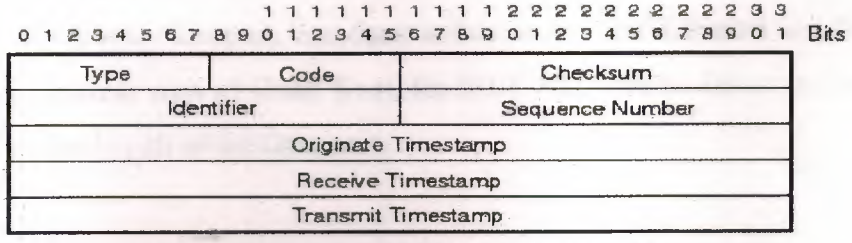
Destination Unreachable, Source Quench and Time Exceeded Messages



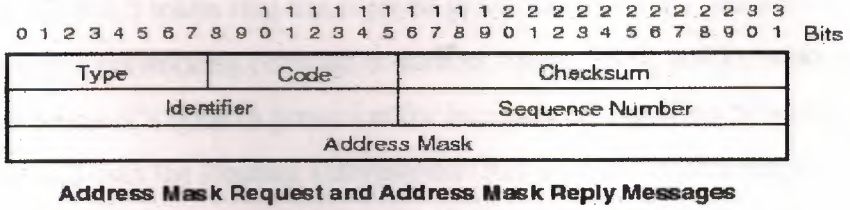
Parameter Problem Message



Redirect Message



Timestamp and Timestamp Reply Messages



Address Mask Request and Address Mask Reply Messages

Figure 6.5 ICMP message formats

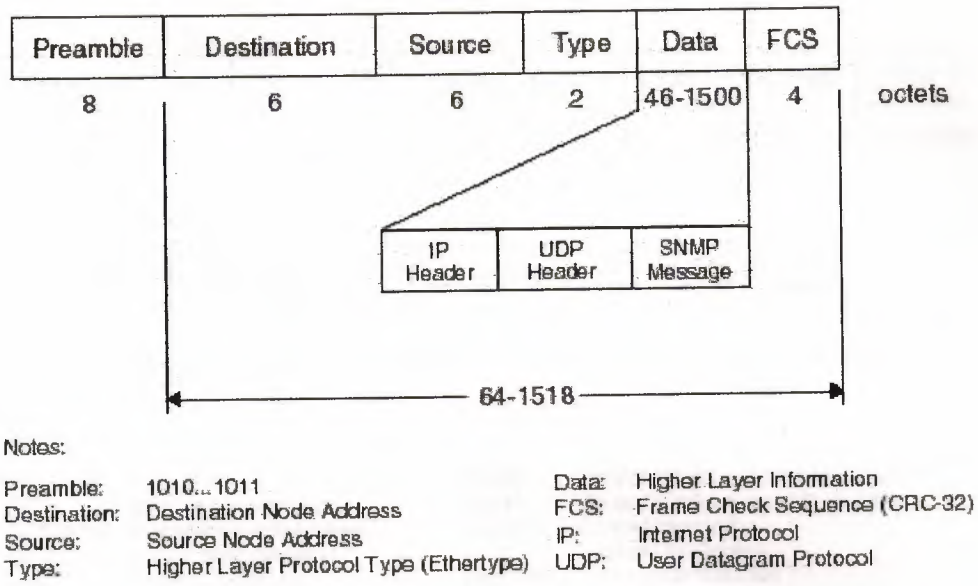


Figure 6.6 Ethernet frame with SNMP message (©1982 Digital Equipment Corp.)

6.5.2 IEEE 802.3

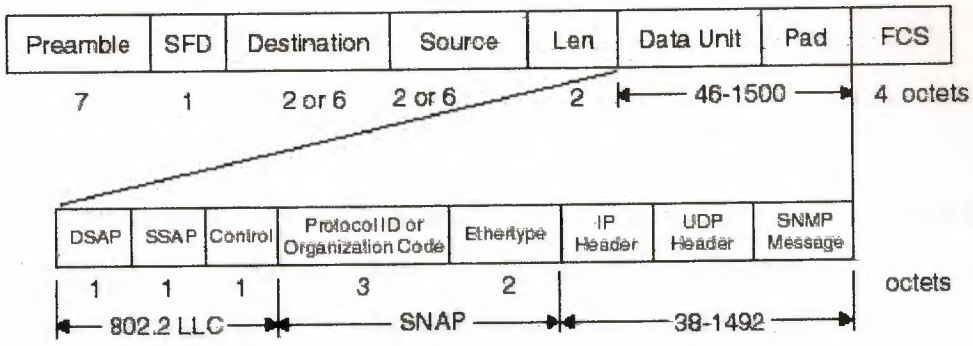
Figure 6.7 shows the IEEE 802.3 format. IEEE 802.3's Destination and Source address fields may be 2 or 6 octets long, although the 6-octet length, which matches the Ethernet address lengths, is most common. The Address Resolution Protocol (ARP) maps the IP address (32 bits) to the IEEE 802 address (48 bits). The ARP hardware code for IEEE 802 networks is 6. However, broadcast addresses for both Ethernet and IEEE 802 networks are consistent with all Ones. Next, the IEEE 802.3 frame defines a Length field, which specifies the length of the Data unit.

6.5.3 IEEE 802.5

The IEEE 802.5 token ring has enjoyed great success, partly because of strong support from major networking companies such as Apple, IBM, and Proteon, and partly because of the protocol's built-in provision for internetworking. This provision is known as *source routing* and uses the Routing Information (RI) field to connect rings via bridges. The RI field specifies the path the frame must take from its source to its destination. The mechanism for determining that path is called *route discovery*.

The IP Datagram occupies the Information field of the token ring frame, as shown in Figure

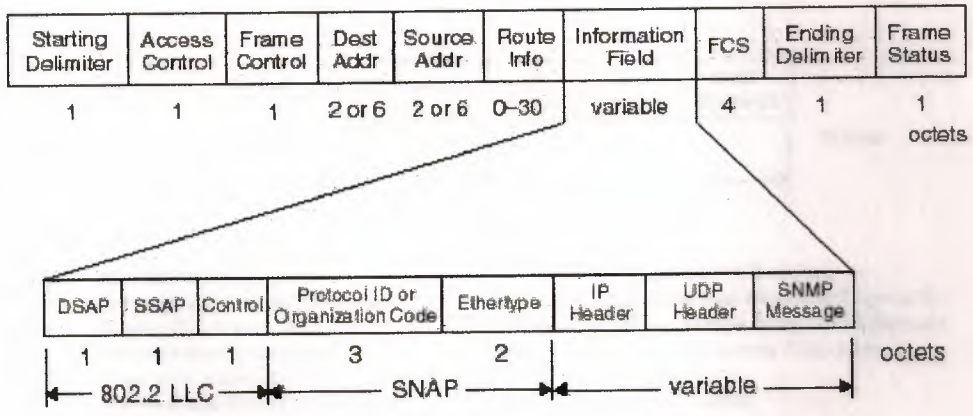
6.8.



Notes:

Preamble:	1010...1011	DSAP:	Destination Service Access Point Address
SFD:	Start Frame Delimiter	SSAP:	Source Service Access Point Address
Destination:	Destination Node Address	Control:	Control Information
Source:	Source Node Address	LLC:	Logical Link Control
Len:	Length of Data Unit	Protocol ID:	Protocol Identification
Data Unit:	Higher Layer Information	Ethertype:	Ethernet Protocol Type
Pad:	Pad Characters (to reach minimum frame size)	SNAP:	Subnetwork Access Protocol
		IP:	Internet Protocol
FCS:	Frame Check Sequence (CRC-32)	UDP:	User Datagram Protocol

Figure 6.7 IEEE 802.3 frame with SNMP message (©1990, IEEE)



Starting Delimiter:	Beginning of Frame	DSAP:	Destination Service Access Point Address
Access Control:	Transmission Parameters	SSAP:	Source Service Access Point Address
Frame Control:	Frame Type	Control:	Control Information
Dest Addr:	Destination Node Address	LLC:	Logical Link Control
Source Addr:	Source Node Address	Protocol ID:	Protocol Identification
Route Info:	Routing Information Field	Ethertype:	Ethernet Protocol Type
Information:	Higher Layer Information	SNAP:	Subnetwork Access Protocol
FCS:	Frame Check Sequence (CRC-32)	IP:	Internet Protocol
Ending Delimiter:	End of Frame	UDP:	User Datagram Protocol
Frame Status:	Receiver-provided Feedback		

Figure 6.8 IEEE 802.5 frame with SNMP message

6.5.4 FDDI

ANSI developed the Fiber Data Distributed Interface (FDDI) as a standard for fiber-optic data transmission. FDDI is a token-passing ring architecture that operates at 100 Mbps. (The actual data rate for FDDI is 125 Mbps, but 1 out of 5 bits handles overhead.) Because of its transmission rate, FDDI may emerge as a significant alternative to Ethernet or token ring for local data transport. The FDDI frame structure, shown in Figure 6.9, is similar to IEEE 802.6.

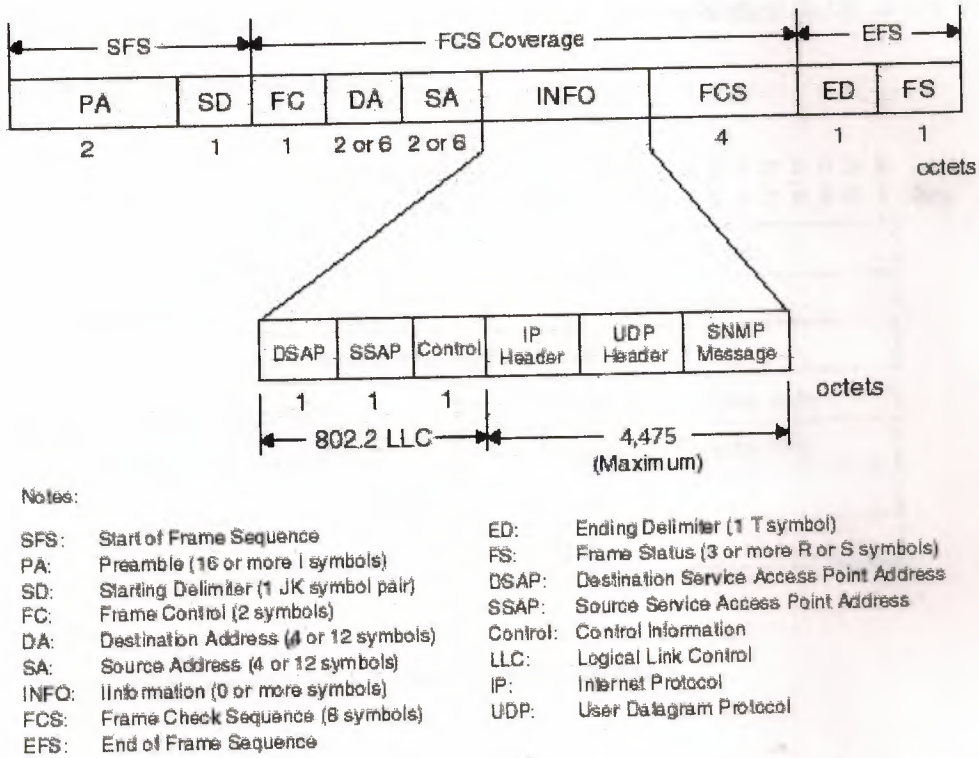


Figure 6.9 FDDI frame with SNMP message *(Courtesy American National Standards Institute)*

6.6 Address Translation

The translation between the physical and logical addresses is necessary. The

Address Resolution Protocol (ARP) described in RFC 826 translates from an IP address to a hardware address. The Reverse Address Resolution Protocol (RARP), detailed in RFC 903, does the opposite, as its name implies.

6.6.1 Address Resolution Protocol (ARP)

Assume that a device on an Ethernet, Host X, wishes to deliver a datagram to another device on the same Ethernet, Host Y. Host X knows Host Y's destination protocol (IP) address, but does not know Host Y's hardware (Ethernet) address. Host X would therefore broadcast an ARP packet (shown in Figure 6.10) on the Ethernet to determine Host Y's hardware address. The packet consists of 28 octets, primarily addresses, contained within the Data field of a local network frame. A device that recognizes its own protocol address responds with the requested hardware address. The individual fields of the ARP message show how the protocol operates.

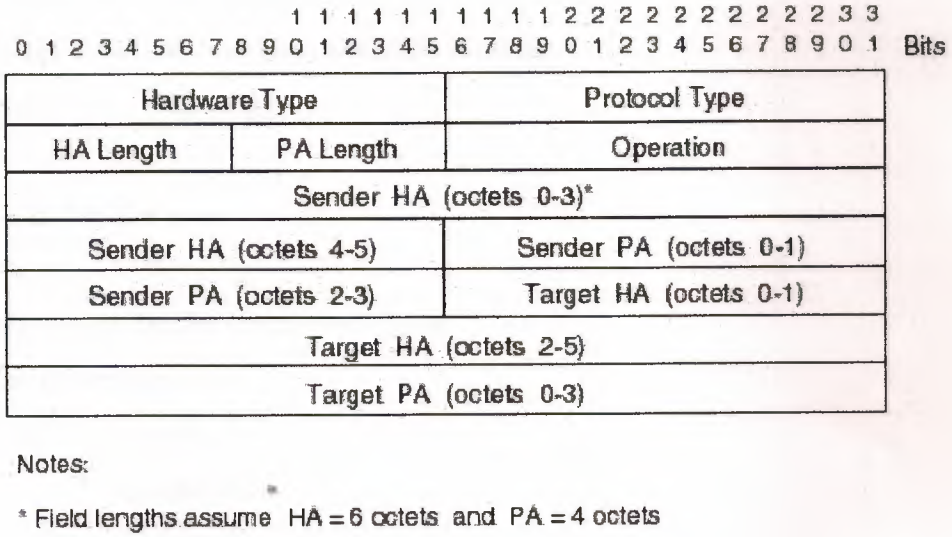


Figure 6.10 Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP) packet formats

6.6.2 Reverse Address Resolution Protocol (RARP)

The process of determining an unknown protocol address is similar to that of finding an unknown hardware address. The same packet structure is used, with only minor

modifications to the field values required. The Operation field adds two new values, 3 (RARP Request) and 4 (RARP Reply).

6.7 Using SNMP with UDP and IP

This section shows how the SNMP GetRequest and GetResponse PDUs fit within the structure of an Ethernet frame. In this section we will examine the Ethernet frame format, the IP header, the UDP header, the SNMP PDUs. Trace 6.7a shows four layers of protocol operating in two Frames, 7 and 8.

Trace 6.7a. Using SNMP with Ethernet, IP, and UDP

```
Sniffer Network Analyzer data 10-Nov at 10:29:36 file GOLD_SYS.ENC Pg 1
------------------------------ Frame 7 ------------------------------
DLC:  ----- DLC Header -----
DLC:
DLC:  Frame 7 arrived at 10:29:37.30; frame size is 138 (008A hex) bytes
DLC:  Destination = Station Retix 034CF1, GoldGate
DLC:  Source      = Station Sun   0900C8, SunMgr
DLC:  Ethertype   = 0800 (IP)
DLC:
IP:   ----- IP Header -----
IP:
IP:   Version = 4, header length = 20 bytes
IP:   Type of service = 00
IP:         000. .... = routine
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 124 bytes
IP:   Identification = 20055
IP:   Flags = 0X
IP:   .0.. .... = may fragment
IP:   ..0. .... = last fragment
```

IP: Fragment offset = 0 bytes

IP: Time to live = 60 seconds/hops

IP: Protocol = 17 (UDP)

IP: Header checksum = A5C5 (correct)

IP: Source address = [XXX.YYY.128.4]

IP: Destination address = [XXX.YYY.1.10]

IP: No options

IP:

UDP: ----- UDP Header -----

UDP:

UDP: Source port = 3234 (SNMP)

UDP: Destination port = 161

UDP: Length = 104

UDP: No checksum

UDP:

SNMP: ----- Simple Network Management Protocol (Version 1) -----

SNMP:

SNMP: Version = 0

SNMP: Community = public

SNMP: Command = Get request

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.1.0} (sysDescr.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.2.0} (sysObjectID.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = NULL

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.6.0} (system.6.0)

SNMP: Value = NULL

SNMP:

---------- Frame 8 ----------

DLC: ----- DLC Header -----

DLC:

DLC: Frame 8 arrived at 10:29:37.33; frame size is 195 (00C3 hex) bytes

DLC: Destination = Station Sun 0900C8, SunMgr

DLC: Source = Station Retix 034CF1, GoldGate

DLC: Ethertype = 0800 (IP)

DLC:

IP: ----- IP Header -----

IP:

IP: Version = 4, header length = 20 bytes

IP: Type of service = 00

IP: 000. .... = routine

IP: ...0 .... = normal delay

IP: .... 0... = normal throughput

IP: .... .0.. = normal reliability

IP: Total length = 181 bytes

IP: Identification = 0

IP: Flags = 0X

IP: .0.. .... = may fragment

IP: ..0. .... = last fragment

IP: Fragment offset = 0 bytes

IP: Time to live = 16 seconds/hops

IP: Protocol = 17 (UDP)

IP: Header checksum = 1FE4 (correct)

IP: Source address = [XXX.YYY.1.10]

IP: Destination address = [XXX.YYY.128.4]

IP: No options

IP:

UDP: ----- UDP Header -----

UDP:

UDP: Source port = 161 (SNMP)

UDP: Destination port = 3234

UDP: Length = 161

UDP: Checksum = 6417 (correct)

UDP:

SNMP: ----- Simple Network Management Protocol (Version 1) -----

SNMP:

SNMP: Version = 0

SNMP: Community = public

SNMP: Command = Get response

SNMP: Request ID = 0

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 240267300 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.1.0} (sysDescr.0)

SNMP: Value = Retix Local Ethernet Bridge Model 2265M

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.2.0} (sysObjectID.0)

SNMP: Value = {1.3.6.1.4.1.72.8.3}

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.3.0} (sysUpTime.0)

SNMP: Value = 240267300 hundredths of a second

SNMP:

SNMP: Object = {1.3.6.1.2.1.1.6.0} (system.6.0)

SNMP: Value =

SNMP:

CONCLUSION

A network management system contains two primary elements: a manager and agents. The manager is the console through which the (human) network administrator performs network management functions. Agents are the entities that interface to the actual devices being managed. Bridges, routers, switches, or network servers are examples of managed devices that contain managed objects. These managed objects might be hardware, configuration parameters, performance statistics, and so on, that directly relate to the current operation of the device in question. These objects are arranged in what is known as a virtual information database, called a management information base (MIB). SNMP allows managers and agents to communicate for the purpose of accessing these objects.

There are a number of advantages of SNMP that contribute to its popularity. Because it requires relatively little code to implement, vendors can easily build SNMP agents into their products. SNMP is extensible, allowing vendors to easily add network management functions. Also SNMP separates the management architecture from the architecture of the hardware devices, which broadens the base of multivendor support.

REFERENCES

[1] Klerer, S. Mark. "The OSI Management Architecture: An Overview." *IEEE Network* (March 1988): 20–29.

[2] Yemini, Yechiam. "The OSI Network Management Model." *IEEE Communications Magazine* (May 1993): 20–29.

[3] Institute of Electrical and Electronics Engineers. LAN/MAN Management. ANSI/IEEE Std 802.1B, 1995.

[4] http://www.intraspection.com.

[5] http://www.cabletron.com/spectrum.

[6] http://www.hp.com/go/openview/.

[7] http://www.sun.com.

[8] http://www.tivoli.com.

[9] Rose, M.T., and K. McCloghrie. "Structure and Identification of Management Information for TCP/IP-based Internets." *RFC 1155*, May 1990.

[10] Steedman, Douglas. *Abstract Syntax Notation One (ASN.1), the Tutorial and Reference*. Isleworth, Middlesex, UK: Technology Appraisals, Ltd. ISBN 1-871802-06-7, 1990.

[11] McCloghrie, K., and M.T. Rose. "Management Information Base for Network Management of TCP/IP-based Internets." *RFC 1156*, May 1990.

[12] Case, J.D., M. Fedor, M.L. Schoffstall, and C. Davin. "Simple Network Management Protocol (SNMP)." *RFC 1157*, May 1990.

[13] Davin, J., J. Galvin, and K. Mcloghrie. "SNMP Administrative Model." *RFC 1351*, July 1992

[14] McCloghrie, K., editor. "An Administrated Infrastructure for SNMPv2." RFC 1909, February 1996.

[15] Finlayson, R., et.= al. "A Reverse Address Resolution Protocol." *RFC 903*, June 1984.

[16] Postel, J. "User Datagram Protocol." *RFC 768*, ISI, August 1980.

[17] http://www.us-epanorama.net