MASTER IN COMPUTER SCIENCE

Network Management Systems
The management of TCP/IP networks

Heyvaert, Didier

Award date:
1991

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 1990-1991

Network Management Systems

The management of
TCP/IP networks

Didier HEYVAERT

Mémoire présenté en vue
de l'obtention du grade
de Licencié et Maître
en Informatique

Promoteur : Philippe van Bastelaer

Rue Grandgagnage, 21, B - 5000 NAMUR (BELGIUM)

Résumé

Lorsque, actuellement, on parle de réseaux, le sujet le plus abordé est celui de leur gestion. Ceci est dû à un besoin de plus en plus pressant du côté des responsables de réseaux qui voient leur tâche se complexifier. Cette complexité a pour cause, de manière générale, le développement des réseaux de télécommunications, tant du point de vue de leur taille que du point de vue de leur composition. Les responsables de réseaux ont donc besoin de nouveaux outils pour les aider à résoudre les problèmes qui peuvent se présenter.

Dans ce travail, nous présenterons d'abord un approche de ce qu'est, actuellement, la gestion des réseaux et des outils nécessaires à celle-ci. Puis, de manière à assimiler les concepts nécessaires à la compréhension du protocole SNMP, nous décrirons les principaux protocoles du réseau TCP/IP. Cela permettra de décrire le protocole de gestion des réseaux TCP/IP, SNMP (Simple Network Management Protocol). Nous pourrons, ensuite, après avoir brièvement parcouru le protocol ISO (Interconnexion de Systèmes Ouverts) de gestion de réseaux, nous attarder sur quelques programmes de gestion de réseaux.

Abstract

When, at the present time, it is spoken of networks, the most approached subject is their management. This is due to a more and more urgent need for persons responsible for the networks who see their task becoming complicated. This complexity is generally caused by the development of telecommunication networks, from the point of view of their size but also of their composition. Thus, the persons responsible for the networks need new tools to help them solve the problems that can arise.

In this work, we will first present a conception of what is, at the present time, the management of networks and of the tools useful for the management. Then, to assimilate the concepts needed to understand the SNMP protocol, we will describe the main TCP/IP network protocols. This will allow to describe the TCP/IP network management protocol, SNMP. Then, after having briefly glanced at the OSI (Open Systems Interconnection) network management protocol, we will expose some network management programs.

ACKNOWLEDGEMENT

I would like to thank all the persons who have contributed to the elaboration of this work.

First, I want to express my acknowledgement to M^{r} Philippe van Bastelaer, my work director at the Facultés Universitaires Notre-Dame de la Paix, Namur, who offered me the opportunity of a training at the European Centre for Nuclear Research (CERN), Geneva, during the first semester of the academic year 1990-1991.

I am also very grateful to all the members of the Computer and Networks / Communication Systems (CN/CS) section of the CERN with whom I spent six unforgettable months and especially M^{r} John N. Gamble, my supervisor, who introduced me to the problem of network management.

And finally I would like to thank M^{r} Rudi Simon who spent many hours of his time correcting all the English mistakes of this work.

CONTENTS

INTRODUCTION

At the present time, the size and the number of networks are expanding in an exponential way. And, thus, the control on the networks is proportionally harder to keep. Therefore, different mechanisms to help in the management of networks must be considered.

The aim of this work is to present the current state of the art in the management of networks, in general, and of TCP/IP networks, in particular.

Three approaches have been followed.

The first one is an intuitive approach. It consists in a reflexion on what network management could be. This reflexion led to the presentation of the people involved in network management, of the tasks to perform and of the tools useful for managing a network.

The second approach is practical. It has been realised through the testing of network management programs at the European Centre for Nuclear Research (CERN) in Geneva. The CERN is composed of a set of laboratories. The aim of the different laboratories is to provide particle accelerators to searchers. All the teams coming at CERN for carrying out experiments, have to take back their results with them for further analysis. But they all have different computers and different ways of communication. This involves that the CERN needs to have a very wide range of computers and networks. All this infrastructure must remain operational as long as possible. In order to no longer have the charge of maintaining its own management programs, the CERN has decided to buy one. To be sure that this program comes up to its expectation and responds to its needs, thorough tests had to be carried out.

The third approach is more formal. It consists in the analysis of the communication mechanisms that take place between computers for network management purposes. Therefore, some protocols useful for managing network will be described.

This work is divided into seven chapters. *Chapter 1* resumes the intuitive approach and lists the people, the tasks and the tools concerned with network management. *Chapter 2* gives a description of the main protocols used in TCP/IP networks. This will allow to expose concepts used in the next chapter. *Chapter 3* details the main protocol used for managing TCP/IP networks, the Simple Network Management Protocol (SNMP). *Chapter 4* briefly describes the OSI network management protocol, the Common Management Information Protocol (CMIP) and its application to TCP/IP networks. And *chapter 5*, *chapter 6* and *chapter 7* illustrate the practical approach by presenting the different tested programs.

CHAPTER 1 : NETWORK MANAGEMENT SYSTEMS

In the past, network management was limited to the only field of fault management. The networks were limited in dimension and in complexity. One person was able to know which components were used on the network and where they were placed. The job was limited to locating the component that was causing the trouble. This person only needed tools to help him or her in the location of faults.

Now, as the number of computers increases, the size of the networks grows in proportion. And their complexity grows much more because the networks pass from a single-vendor state to a multi-vendor state with all the related difficulties (different protocols, different machines). If the networks change, they also become more difficult to manage. More people will be involved in their management. And other, more sophisticated tools to help in the management of the networks will be indispensable. New tools must, therefore, be developed to cope with this complexity. These tools will be composed by the integration of other more specific tools: databases, monitors, etc. They will not only be used in the detection of network problems but they will also serve in the network planning, performance testing, inventory of network components, etc. These tools will be termed **Integrated Network Management Systems**.

The rest of this chapter will be divided into four parts. The *first* section will give a description of the different people involved in the management of the networks. The *second* section will list the areas in which network management is used. The "older" tools which helped and still help in the fault detection and location will be presented in the *third* section. And finally, a review of all the components that should be included in a performing integrated management tool are described in section *four*.

1.1. People involved in management

Now, as networks become more complex, their management is not the business of one person anymore. Many people are concerned with this activity. In this

section, the different people concerned with network management are listed. Information is also given about what they must know concerning the network. Dr J. N. Gamble [GAMB90] gave many ideas for this section.

1.1.1. The Higher Level Management Entity

The higher level management entity is the authority that has the power to manage the evolution of the network. This can be, for example, for a university the academic authorities, for a firm the board of directors, etc. It requires information on how well the network manager is doing his job. Figures, gathered over a long period, on the network equipments, on the services offered, on the quality of the service are needed for resource planning (manpower and equipment).

1.1.2. The Network Manager

The network manager is the person who is responsible for the good running of the network. Because he is also involved in the resource planning, he needs the same information as the Boss. Another job for the manager is to make the network run with as few down periods as possible. For this, he needs many other tools to find faults, to monitor performances, etc. These tools will be detailed later. The allocation of network addresses for the various protocols that are supported is another important role for the network manager. For this, database facilities are needed. For wide area networking, accounting and authorization are of importance. Access security to the network is also important. These are two other roles of the network manager.

1.1.3. Operators

The operators are the ones that users contact when there appears to be a problem and they are the ones who involve the maintenance experts (internal network engineers or maintenance staffs from the vendors) to fix faults. They need to know if there are faults on the network, where the faults are, who to call to repair them, etc. Help is needed because, if the network is healthy, the operators will not be in the habit of coping with fault situations and they must have quick reactions in case of problems.

1.1.4. Network Engineers

Network engineers are concerned with the maintenance of the network equipments, with the exact location of faults, with the repair of equipments, etc. For

maintenance, the engineers need information on the "health" of the network, particularly at the physical level. They need to see error rates, traffic rates, throughput rates of bridges, gateways, etc. For fault location, they need to know "who is where" on the network and have the tools to precisely identify the nature of the faults.

1.2. Network Management Areas

As has been seen in the previous section, network management is the business of more than one person. If, before, network management was limited to fault management, now it has grown to many other areas. For example, the higher level management entity needs performance information. The network manager must be able to configure his network, to perform accounting related tasks and to set up security mechanisms. The operators and network engineers need information concerning faults and must be able to repair the network. In order to give a formal description of all these needs, a management framework has been built. A lot of information was found in the Management Framework for Open Systems Interconnection [ISO7498-4]. Five network management areas have been defined: configuration management, performance management, fault management, accounting management and security management.

1.2.1. Configuration Management

The configuration management area can be divided into two sub-areas: the configuration management itself and the name management.

The purpose of network configuration and name management is to define, collect, manage and use configuration information (location, name, availability, reachability information) and to control the configuration of the network resources, in order to maintain the quality of service provided by the network environment.

"Some of the services that *configuration management* provides include setting network parameters, collecting data on network status"[JOSE88], changing the network configuration, displaying the network topology.

"Some of the services that *name management* provides include naming the resources to be managed and managing name assignments."[JOSE88] It provides services to help in the maintenance of name, address and location information and in the assignment of network addresses for the different supported protocols.

In short, configuration management comprises the "mechanisms to determine and control the characteristics and state of a network and to associate names with managed resources".[JOSE88]

1.2.2. Performance Management

Performance management involves monitoring the network, "obtaining feedback on its usage, identifying weaknesses and exercising control to correct these weaknesses. Typical weaknesses include processing bottle-necks, communication delays, high error rates."[SLOM88]

The performance management process can be split into the following functions.

The system gathers statistical information (error rates, throughput) on the network and on the important components of the network. In addition, the system maintains logs of system state histories. Using the gathered information and the logged historic information, it is possible to determine the system performance.

The following important function is the ability for the system to create performance reports. They show statistics and performance evaluation results.

Using these reports, the network manager must take performance optimization decisions. For example, if the traffic becomes too important for a routing device, the network manager can decide to add a new one and to share the traffic between the two routing devices.

When the optimization decision is taken, it is important to have the ability to implement it. The network manager must be able to tune the performance of the network. For instance, when the new routing device is installed, the manager must dispose of tools to change the routing tables of other network devices in order to share the traffic.

In brief, "performance management includes mechanisms to monitor and tune the network performances".[JOSE88]

1.2.3. Fault Management

Fault management is the activity which helps in the detection, the isolation and the correction of faults on the network.

The faults on a network can be of multiple origins. They can result from hardware or from software problems. The detection of faults can be made either by polling the important network components or by receiving event messages sent by the faulty components.

Once a fault has been detected, somebody must be advised, in general the operator, that a problem occurred. This will be made by the mechanism of alarms. The operator must react quickly and inform the network manager and/or network engineers, if necessary. Alarms notifications will be kept and saved in order to help in the network planning and in further corrections of network faults.

The network manager, in collaboration with engineers, will work in the isolation of the faults. Expert systems can also be very helpful in the diagnosis of faults.

Once a fault has been isolated, network engineers will be able to correct it. Mechanisms can be used by the network manager to bypass the fault if the network cannot be corrected quickly enough.

"In summary, fault management is the discipline of detecting, diagnosing, bypassing, repairing and reporting on network equipment and service failures."[MINO89c]

1.2.4. Accounting Management

Accounting management enables network managers to identify costs and establish charges for the use of communications resources. Accounting management can be divided into two functions.

The first one is the monitoring of all communications services. Accounting information must be collected. This information includes the amount of data received and data sent out by the network users. The network manager must be able to know which networks have been used and what costs have been incurred and by whom.

Therefore, all devices that provide a payable communication service must maintain statistics on all calls and call attempts.

These statistics will also permit the network manager to check the correct use of the network. For example, it would be useful for the network manager to have the possibility to fix budget limits for each network user and if the budget limit is reached, the user should automatically see his access limited to the free services on the network.

The second function of accounting management is the billing. The communication statistics will be transferred to a central computer. This computer collects together all the statistics from the different monitored entities. And, once a fixed period is reached, invoices are generated from them automatically. This billing function can also be combined with other services. For example, the users can receive bills containing their expenses for communications, printers, CPU time, disk usage, etc.[DALL88]

In short, "accounting management includes mechanisms for controlling and monitoring charges for the use of communications resources".[JOSE88]

1.2.5. Security Management

"Security management is an important issue: unauthorized or accidental access to strategic components must be eliminated or minimized."[MINO89c]

Security management supports the control and the distribution of information to various end systems that provide security services and mechanisms and reports on security-related events. Therefore, security management requires distribution of information to these services and mechanisms, as well as the collection of information concerning their operations. Examples are the distribution of cryptographic keys, the distribution of information on an entity's access rights, the reporting of both normal and abnormal security events (audit trails) and service activation and deactivation.[MINO89a]

In summary, security management functions include the creation, deletion and control of security services and mechanisms; the distribution of security-relevant information and the reporting of security-relevant events. It also concerns security of the network management system itself.

1.3. Physical Level Tools

Network interface cards, cables and other low-level hardwares all have an impact on the behaviour of a network. If an interface card is working improperly or a cable is not correctly terminated, errors may occur. In this case, the network will run significantly more slowly due to retransmissions of mutilated and lost packets.

There is a number of ways to check if interface cards and cables are working properly. Most network manufacturers include a basic diagnostic utility program integrated within their hardware. These programs are able to detect severe errors.

For diagnosing cable problems, there is a wide range of tools available. Different tools are available for copper-based (twisted pair or coaxial) networks and for fibre optic networks. The four first ones described hereunder are aimed at copper-based networks and the other three ones at fibre optic networks. The last one is much more powerful and it is not limited to the detection of physical-level problems.

The majority of the points developed in this section are borrowed from Daniel Minoli [MINO89b].

1.3.1. Ohmmeter

An ohmmeter is a simple tool that gives impedance measurement. An ohmmeter can be used to locate open or shorted cables. If the measured impedance matches the rated impedance of the cable, the cable is fine. If it does not match the rated impedance, then the network has a short circuit, a crushed cable or a cable break somewhere along the cable.

1.3.2. Outlet Tester

Sometimes the problem is not the cable but rather the electrical outlet. For example, if an outlet is not grounded properly, noise or even current may be introduced through the power supply into a workstation and then through the network interface card onto the copper-based network cable. An outlet tester can help detect this type of problem.

1.3.3. Oscilloscope

An oscilloscope allows to examine the cable's waveform. An oscilloscope helps detect the existence of noise or other disturbances on the wire, such as continuous voltage spikes.

1.3.4. Time Domain Reflectometer (TDR)

A time domain reflectometer operates by sending an electrical pulse over the network cable, waiting for signal reflections. On a good cable there will be no reflection. If there is a break or short circuit in the cable, the time it takes for the pulse reflection to return gives the TDR a very accurate idea of where the fault is located.

Fibre-based networks require different equipments. While they provide significant advantages over conventional networks, the fibre networks necessitate more sophisticated test equipment. Fibre optic instruments can be divided into three categories: power meters, optical time domain reflectometers and optical bandwidth test sets.

1.3.5. Power Meter

Power meters (or optical loss test sets) measure the optical power from a length of fibre in much the same way that conventional power meters measure electrical power. Two sets, both with transmit and receive capabilities, are used together to make measurements in both directions. A light source, typically at the point of origination, supplies the power which is detected at the end of the fibre link. The meter displays the power detected in decibels.

1.3.6. Optical Time Domain Reflectometer (OTDR)

OTDRs can be used to characterize a fibre wherein an optical pulse is transmitted through the fibre and the resulting light scattered and reflected back to the input is measured as a function of time. OTDRs are useful in estimating the attenuation coefficient as a function of distance and in identifying the location of defects and other losses. These devices operate on basically the same principles as a copper-based TDR.

1.3.7. Optical Bandwidth Test Set

Optical bandwidth test sets consist of two separate parts: the *source*, whose output data rate varies according to the frequency of input current applied to the source (specified by frequency range parameter); and the *detector*, which reads the changing signal, determines the frequency response and then displays a bandwidth measurement.

Another very important more sophisticated tool for fault management is the protocol analyzer. Even if this tool is directly connected at the physical level to the network, it is not only a physical level diagnostic tool.

1.3.8. Protocol Analyzer

A protocol analyzer is a specialized station that collects, analyzes and displays the data circulating in a network cable. An analyzer allows to see everything on the cable: PDUs[1], messages, files, passwords.

A protocol analyzer is not limited to hardware or cabling fault diagnosis. Using a protocol analyzer, it should be possible to find spurious broadcast packets, routing errors, addressing errors, etc. The protocol analyzer allows to examine packets where problems are occurring and it files information in a packet dump for later retrieval. It is then easier to track down the particular host or protocol which is causing the problems. Good protocol analyzers are able to identify packets from many different protocols and from every layer of each protocol. For example, they must be able to identify packets from TCP/IP, OSI, DECnet, SNA, Appletalk, NFS and they must be able to be placed on many supports (e.g. X25, Ethernet, Token Ring, etc).

If there is, for example, a broadcast storm, it is necessary to determine where it comes from. The analyzer is able to decode a packet to find the address of the sender. Then, the location of the faulty device can be found thanks to its address. And if the sender is known, it is possible, by decoding the packets, to see which application program sends these packets.

---

1. Protocol Data Unit

1.4. Integrated Network Management Systems

An integrated network management system (INMS) is a program or a collection of programs which integrates a number of network management functionalities within the same workstation or the same environment.

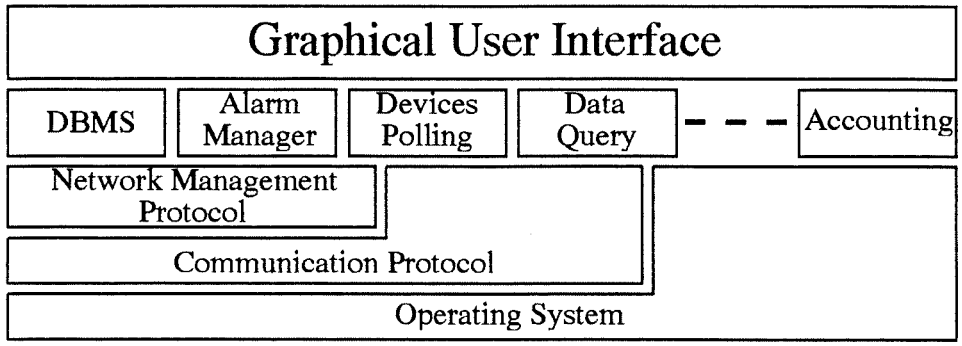


Fig. 1.1: Integrated Network Management System

The major tools that compose an INMS are listed in the following points. As shown in figure 1.1, two levels can be distinguished. The first level is composed of a graphical user interface which provides an access to the other facilities. And the second level contains all the other facilities, such as a DBMS, a topology DB builder, an alarm manager, a devices polling entity, a traffic monitor, reachability tests, an addresses monitor, a data query and values setting entity, a statistics monitoring and computing entity, an encipherment manager, an access control and authentication entity, an audit trail and a billing and accounting entity.

1.4.1. Graphical User Interface

It is necessary to provide an interface to access the management facilities. This interface will enable to perform the management functions. "The overall function of the human interface is to translate management information from the format convenient for internal use into a format suitable for people to understand and to translate commands and requests generated by a human into the form appropriate to the service being managed."[SLOM88]

The objectives of the user interface are: display of networks, near real time updates of network status, interactive query of network information and interactive control of network elements.[MEUN88] Tools should be able to display that information to the user in a form that augments his/her ability to work with the data.

"A simple menu-driven system facilitates learning of the functions, while a command-line mode allows the more accomplished user to by-pass the sometimes tedious redisplays necessitated by the menus."[MORR89]

The map representation should give a global view of all the structure of the network with the different segments, the bridges, the gateways, routing devices, etc. Some aspects of the presentation should be automatically changed (for example: change the colour of a node if it is down) regarding the information registered in the topology database. It should also be possible to obtain information about a node only by clicking on it with the mouse (type, name of device, state, throughout rate, error rate, etc).

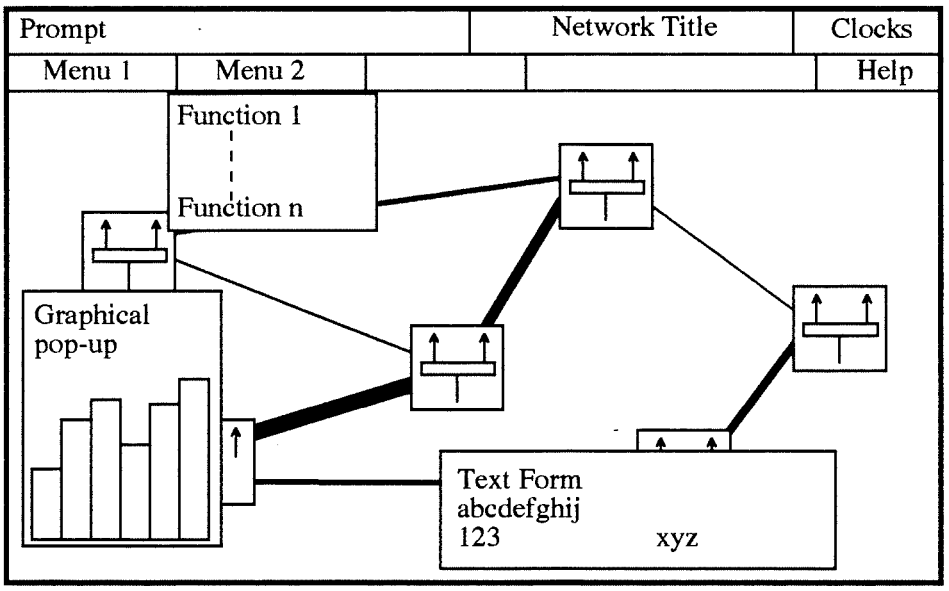


Fig. 1.2: Graphical user interface

The major elements of a graphical user interface are shown in figure 1.2. The different screen areas are the following ones: [MEUN88]

→ the display area where the network is displayed,

→ the context area which contains the title of what is being displayed in the display area, e.g. European OSI Network,

→ the clock area which gives both the current time and the time of the last network update,

→ the prompt area which is used to interact with the system,

→ the pull down menus which give the user choices between different commands,

→ pop-up forms which appear when the user requests specific information or when the system needs input from the user and

→ graphical pop-up forms are used to graphically present data.

A graphical workstation is interesting because it is much more convenient to work with a graphical representation of the network than, for example, with a file describing the topology of the network. It is also easier to see that a node is down, when its colour changes on the screen. Help screens are also needed because some tools will not be used very often and help is more convenient than documentation for rapid consultation.

Furthermore, other tools are needed to draw or edit the map, to add, delete, move objects and also to search for objects (by name, by address, etc).

1.4.2. Database

Database is a crucial resource for an Integrated Network Management System. Databases are made to store information. And in rather large networks, management information can run to a very respectable amount. That is why a good Database Management System (DBMS) must be integrated in a network management system. If possible, this DBMS must be well-known (e.g. Oracle) or must, at least, be accessible by a well-known language (e.g. SQL[2]), in order to allow the network responsible persons to write their own programs to exploit the recorded information or to record information. For example, for report generation, no product can anticipate all the different types of reports that may be requested of the data, so the ability to extract information from the database via a user-written program is a valuable asset.

Databases can be used for different applications in an INMS[3]. Some of the different databases which can be considered are listed below.

The first application is a ***network inventory***. For any large network, it is vital to have an inventory of objects connected to the network so that effective steps can be taken if an object gives problems. The inventory database should contain information on all the equipments that are connected to the network or form parts of it. This information is: the owner's name, the make and the type of the installed

---

2. Structured Query Language

3. Integrated Network Management System.

devices, their location, the type of software used, etc. The database should serve as directory of information for all classes of management but in particular as a tool for the maintenance (search for the problems on the network, etc).

A second application of database is a ***topology database***. This database contains all the information necessary to display the map of the network topology. Its information is the class of the devices, the map to which they belong, their places on the map, their current status (up, down, inactive) and some information concerning the traffic, for example, traffic passing through a link. This database will be directly queried in order to display, on-line, the state of the network on the graphical user interface.

Another application of databases in an INMS is the ***users directory***. A list of all the users can be very useful in many management areas. For example, this list could be used in fault management to be able to determine the user of a faulty device and where to find him. In accounting management, the invoices must be sent to the users. Therefore, it is important to know where to send them, e.g. their E-mail address. In security management, this directory could be used as repertory for passwords and services that a user can employ and therefore be useful for the authentication on the different computers of the network.

A ***vendor database*** is also of great importance in an INMS. This database would contain information about the vendors, the maintenance contracts, the selling contracts, people to contact, etc. This database would be very helpful in fault management. When a fault occurs on a device, it will be easy to find its vendor and then the maintenance contract, who to call to repair, etc.

The ***statistics database*** is the one in which all the information collected on the network is saved. This information concerns the measures done on the nodes but also on the network itself. An example of the application of this database would be the fault detection. Every hour, the error rates of all the devices are logged in the database. Then, it is possible to detect that a device is causing more and more errors on the network. This allows to repair the faulty device before it completely goes down. The statistics database can also be used to help in the network planning. It gives long-term figures in order to build graphs on the evolution of the network traffic. And for example, this allows to determine that, if the traffic increases in a part of the network, a new routing device is needed to reduce the traffic. A non restrictive list of statistics which can be collected, could contain network load, number of packets passing through devices, percentage of errors in the packets, etc.

A ***faults database*** is also very useful in an INMS, especially for the solving of later problems. This will mainly be helpful in the fault management area. The faults database can be divided in two. The first part is the *error log*. This error log contains a history of all the alarms sent to the operators. And it can be used to help in the resolution of problems on the network. For example, it can be useful when trying to isolate a problem to see that it has been caused by several other problems that occurred previously. The second part of the faults database can be called the *maintenance database*. This would mainly be used by the maintenance people. It contains, for each type of problem, the list of operations that should, in theory, be performed to cope with it. For each problem, a trouble ticket is issued. A trouble ticket contains information on the date and time of the problem, the nature of the problem, the specific devices and facilities involved, any short-term actions taken to relieve the problem and information such as visits from the vendor's maintenance staff, dates on which parts were returned for repair and the date of the problem's final resolution. "The manager can call up reports on all trouble tickets that are outstanding, are not solved within a given period and involve a specific device or specific vendor. With such reports, a manager has an objective handle on such factors as the reliability of a given component, the promptness of a given vendor's field service."[NM50-600-1]

A large variety of tools must be present to help managing this amount of information. For example, tools are needed to manually add or delete records on the databases. Other higher level tools should be available to query the databases with questions like "Give me information about all workstations using protocol XXX in building 31".

An important tool for an optimum use of the databases is the *reports generator*. To exploit the amount of information (mainly statistics), reports are needed. Reporting capabilities should support graphics. Graphics communicate information much better than texts or numbers. "INMS reporting features should include the ability to search on any field, sort in any order and subtotal in any way. As nobody can know in advance all the reports that will be required from the INMS, the INMS should include a generic report writer with which new reports can be built as time goes on."[NM20-300-1]

1.4.3. Topology DB Builder

For an INMS, an interesting tool is one that automatically builds a topology database with the nodes that can be found on the network. To build the database, this

tool can inquire into currently existing sources of data, such as network routing tables. It can also use promiscuous listening to the network traffic, looking at source and destination addresses. Different programs can be used to build databases for the different protocols used on the network. For example, in the case of a token ring network running TCP/IP[4], a program will be used to build a database containing all the TCP/IP nodes. This program will listen to all packets passing on the network and decode them to find the TCP/IP ones. When it has found valid TCP/IP addresses, it will be able to check the routing tables of the TCP/IP machines corresponding to the found addresses in order to detect new TCP/IP addresses and so on.

1.4.4. Alarm Manager

Alarm management is a crucial facility that is indispensable in an INMS. Networks are not completely reliable and, in some organizations, network down time can be very expensive. Therefore, errors must be detected as soon as possible. The operators must be alerted of an absolute failure and also of a degrading condition by using a set of alarms. When a problem is detected, an alarm signal, usually a message, must be displayed at the operator's console noting the type of malfunctioning device, the location and the nature of failure. Systems with colour displays use a special colour, usually red, for alarms. For example, if a line or a device goes down, an alarm indication should be visibly displayed on the screen. The map (or a part of it) containing the faulty object should be popped up and a window with the actions to perform if the operator can solve the problem or with the people to advise if he cannot solve it by himself, should be created.

The alarm indication should give information about all the events. The events should also be saved on file and on paper to keep a trace of what happened. The information is the date and time of the event, the type of event, the node or the line that is concerned, the program that detected the problem and the alarm message which explains the problem.

Automatic response to events is also a manner of not losing too much time when an error occurs. In the case of rather frequent problems, sequences of actions can be performed automatically by the INMS. For example, if the system detects that a bridge is not working anymore, before sending an alarm to the operator, the alarm manager can try to restart the bridge by itself and send a message to the operator to

---

[4]. Transmission Control Protocol/Internet Protocol.

explain what it has done. If this action does not work, then an alarm is displayed to warn the operator.

1.4.5. Devices Polling

An important goal of a network management system is to detect if all important devices on the network are working perfectly. If an object stops working as it should, somebody must be informed, to cope with the problem. Therefore, it is important to check, at regular intervals, if every object is working well. This function is in direct relation with the alarm manager. But, although the devices polling entity generates many alarms, it is not the only source of alarms. There are two ways of doing this.

The first way is the trap message. The object to which a problem occurs, sends a message to a well-known program, a trap handler. The trap handler will update the topology database if significant problems arise. But traps are not enough. Since an object can be unable to send a trap message, e.g. if it is not working anymore, another way must be used. A program can poll all important objects to check if every of them is all right. That is the second way.

This program should, at regular intervals, send requests to all devices. If one device does not answer, the program knows that there is a problem. Then it updates the topology database.

Nowadays, most networks are multivendor networks. Therefore there can be different types of equipments on a network and, sometimes, different types of protocols used by these equipments. A way for managing these objects is to use several programs. The objects will be divided into classes regarding the different protocols they support or the different makes. And a specific program will be run for every class. For example, XXX program will be run for devices supporting the XXX protocol, specific program for the equipments from one vendor, etc. All these programs will send the information they gather to a centralization program or they will directly update the topology database. This approach offers different advantages. It allows to share the CPU load used by the polling between several machines. It allows to easily modify a program if the protocol changes.

1.4.6. Traffic Monitor

In any network, bandwidth is an essential resource. Therefore, traffic should be continually monitored in order to determine the traffic levels on the network. This supervision can be made either on a line in the case of a point-to-point link or on a complete network, e.g. in the case of an Ethernet network. On-line graphs of the network traffic or the line load can be displayed on the graphical user interface. The measures can also be logged in the statistics database. If the traffic or the load becomes too high, the alarm manager must be warned.

1.4.7. Reachability Tests

The reachability test is indispensable for network testing, especially in a multivendor environment. It allows to detect if a specified device is reachable from the computer where the test has been performed. It consists in sending a packet and in waiting if a response is sent back by the tested device. This test can be performed at different levels. Three levels of testing can bring interesting information. The first one is the data-link layer test. It provides a method for determining if a working data-link level path exists between two adjacent hosts. Another test is targeted at the transport layer. This test allows additional levels of the communications layers to be controlled. Many failures may pass the data-link test and fail at the transport layer. Finally, the application layer test passes through the entire seven OSI layers. "This function loops data from a user-level task on host system to another user-level task on the target system and back. With the success of this test, the network manager is assured that a usable link from the host node to the target system exists."[MORR89]

1.4.8. Addresses Monitor

The allocation of network addresses is a problem in most of the networks. In theory, if a user wants to install, to move or to remove a device on the network, he should ask the authorization to the network manager. This will enable the manager to allocate addresses and to update the database with the location and address and with the name of the person responsible for the equipment. This is the only way to control the allocation of addresses. However in reality, people rarely warn the manager of the changes they make either for location moves or address changes. Therefore tools are needed to detect these changes automatically, to inform the network manager and prepare the databases for update.

On all the network, addresses appearing on the packets must be registered. This can be made by two means. The first one consists in adding equipments on every part of the network to register the addresses of all the packets they detect. The second means consists in using the tables built by network control devices. For example, the learning bridges build a table for their two interfaces to know if they must copy a packet on the other side. The gateways build routing tables to know where to send the packets they receive. The lists of addresses can be used to locate the different positions of the devices. It is possible, with the lists of addresses and the positions of the gateways and of the bridges, to know on which segment of the network an object with a given address can be found. It is also possible to compare these locations with the ones found in the database to detect all changes of the network topology.

Figure 1.3 gives the example of a small LAN composed of four Ethernet segments connected by learning MAC-level bridges.

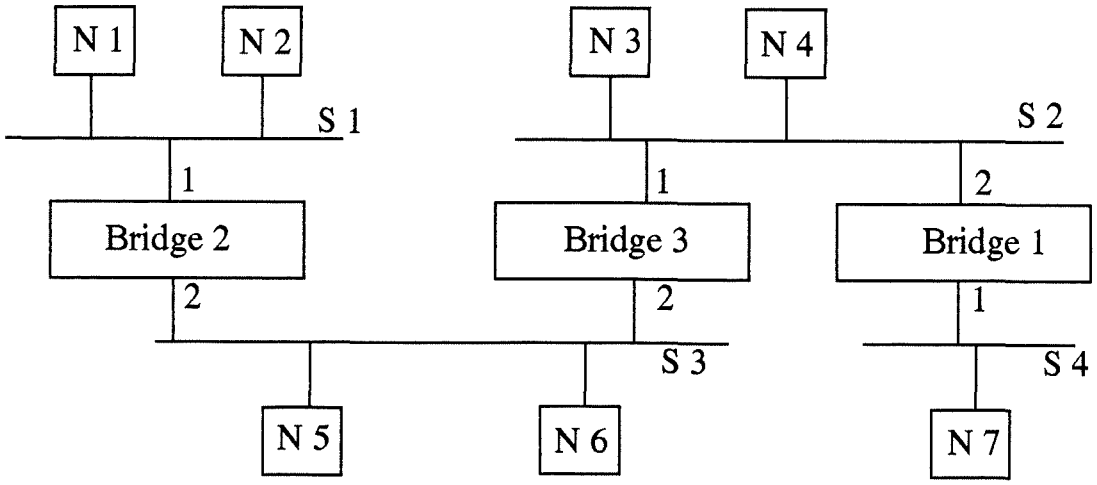


Fig. 1.3: Small LAN composed of MAC-level Bridges

The bridges need address tables to know if a packet coming from one side must be copied on the other interface. And by looking at the origin and destination addresses of every packet, they are able to build tables like the one shown on figure 1.4.

In the above example, it is possible for a program, with the tables as input, to determine that if the node N 5 appears on side 2 of bridge 2, on side 2 of bridge 3 and on side 2 of bridge 1, it is located on segment S 3.

Bridge 1	
I 1	I 2
N 7	N 1
	N 2
	N 5
	N 6
	N 3
	N 4

Bridge 2	
I 1	I 2
N 1	N 5
N 2	N 6
	N 3
	N 4
	N 7

Bridge 3	
I 1	I 2
N 7	N 1
N 3	N 2
N 4	N 5
	N 6

Fig. 1.4: Routing tables of bridges

Every address detected is checked to see if it corresponds to a valid address present in the database. The program should also check to see if two devices do not have the same address. If so, there is an error in the address allocation (duplication) or there is an unofficial use of addresses. In this case, the network manager must be advised.

1.4.9. Data Query and Values Setting

For an operator, information about the state of the objects or the behaviour of the network and about how to react is sufficient. But the network manager needs more. His task is more complex. If a problem occurs, the operator must only report it. But the network manager must find from where the problem came and why it occurred. Therefore, he needs detailed information about the hosts, the routers, the bridges and the network itself. For example, he must be able to retrieve information such as the number of messages that have been sent or received by a specific network device.

Tools must be available to collect, on demand, information about the objects on the network and to transform this information into a readable (graphs,...) and useful way. For example, errors counters are not useful if they are not transformed into error rates, either errors/second or errors/packet.

Furthermore, to be effective, a system must not only give the ability to read values but it must also allow to set values. The system must enable to modify the value of the configurable parameters in the network components. It could be, for example, very useful to modify routing tables, to change the value of a "reset" parameter of a given device. This mechanism could be used, as a first problem

solving attempt, to remotely restart a device which causes too many routing errors due to a software problem.

1.4.10. Monitoring and Statistics Computing

Monitoring is the means by which management entities obtain information on the system they are managing. In the management of a network, statistics on traffic, throughput and error rates are very useful. These provide basic information on how the network is performing. It is possible to see if there is a slow deterioration of a service. In addition, figures on the size of the network, on the services offered, the number of different computers in the network, the quality of the service (current problems, system reliability, etc) are needed for resource planning.

With all the data collected during a period of time, a program will calculate statistics about the network (error rate, load, users, failures, etc) and about the nodes (up/down time, maximum packet received per second, etc). The stored data can also be used to compute, for example, the load of a line and to modify the topology database. The manager must also be able to fix thresholds on variables. If the value of a variable becomes higher than the corresponding threshold, an alarm should be displayed.

Using the database, the user should be able to consult data about the history of the network and automatically create tables, reports, graphics, etc. This will be used to see the state and the evolution of the network and help the network planning.

1.4.11. Cryptographic Keys and Encipherment Manager

The cryptographic keys and encipherment manager mechanisms allow to encipher the information exchanged between two or more entities. The encipherment manager possesses a list with the different groups of entities that must talk to each other in a secure manner. The encipherment manager generates, at given intervals, keys as required and distributes them to the relevant entities in the system. The manager also determines the encipherment parameters. For example, the encipherment manager determines which encipherment algorithm must be used and when this algorithm must be changed. It must also distribute this information to the relevant entities. An example of encipherment activity would be the access to a file server containing secret information. The encipherment manager, every 15 minutes, generates new cryptographic keys and it distributes them to the file server and its

authorized clients. By doing this, the access to the information is very difficult if not registered in the manager's access tables.

1.4.12. Access Control and Authentication

Authentication is the mechanism by which a user identifies himself as a legitimate user of the network. This is, in general, made by using passwords. Access control is the mechanism by which the system controls which entity can access to which resource on the network. This is, in general, implemented using access control lists. The INMS allows to gather access control and authentication within a central entity. Every time a user logs on in any computer of the network, instead of authenticating the user with a local password, the computer will ask to the INMS if the user is allowed to log on. This will have as advantage for the users to work with only one password for all the services available on the network. And the mechanism is nearly the same for access control. Every time an entity (user, program) wants to access to a controlled service, the service asks the INMS if the entity is allowed to use the service. The INMS checks its access lists to see if the entity is allowed to do so and responds to the service that the access demand is accepted or rejected.

1.4.13. Audit Trails

Security audit is needed to record all exceptional events (e.g. attempts at unauthorized access), normal events such as logs-on and file accesses to enable future investigation in case of security problems. For an audit trail system to be effective, it must provide the network manager with streamlined and useful information (rather than mountains of raw data). The manager may wish to audit only certain users, operations on files with certain extensions or in certain subdirectories, only certain types of operations or certain servers. All file and directory creations, deletions and renames may need to be reported. A system error log report, listing all system error messages to alert the network manager of potential problems, may be advantageous.[MINO89d]

1.4.14. Billing and Accounting

Managers require accounting information to be able to charge for resources used and to control costs incurred by accessing remote resources.

On each device furnishing payable network services, an accounting program must be running. The accounting program must be able of maintaining statistics on all

calls and call attempts and these statistics should be adequate so that invoices can be generated from them. Beside accumulating these statistics, the program will also "tell each user at the end of the call, just how much that call has cost". The statistics are stored in a database. The stored information is: the username, the source address, the destination address, the date and time, the call statistics (call duration, number of segments transmitted and received), the cost of the call.[DALL88]

Then, another program, called billing program, is run once a given period. This program retrieves the accounting information in the database and then generates invoices from it automatically. The costs can also be combined on invoices with other payable services such as printing costs, CPU usage costs,...

CHAPTER 2 : THE TCP/IP PROTOCOL SUITE

2.1. Introduction

TCP/IP has been created in the US research community in response to a problem. The problem was to allow a collection of heterogeneous computers and networks to communicate. To solve this problem, it was decided to build a number of protocols.

The protocol architecture is divided into four layers. The protocols are derived from the ones developed by the Department of Defense (DoD). The standardization entity which oversees the development of these protocols, is the Internet Activities Board (IAB).

- The Network Access Layer describes the physical and data-link technologies used to realize transmission at the hardware level. It is concerned with the exchange of data between a host and the network to which it is attached. The protocol used at this layer will depend on the type of the network, for example X21, X25, IEEE 802.

- The Internet Layer provides the routing function across several networks. It describes the internetworking technologies in order to "forget" the different physical technologies. The major protocol developed for the internet layer is termed the *Internet Protocol* (IP).

- The Host-to-host (or Transport) Layer describes the end-to-end technologies to realize communications between hosts. For this layer, two protocols have been developed. The first one, the *Transmission Control Protocol* (TCP), provides a reliable mechanism to transfer data between computers. Whereas the second one, the *User Datagram Protocol* (UDP), provides only a non-reliable datagram-oriented transfer mechanism.

- The Application Layer contains protocols that provide end-user services. Examples of application protocols are the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP), etc.

As has been seen above, the aim of the internet layer is to interconnect different networks. This interconnection is realized through the Internet Protocol (IP). IP allows a computer to send messages to another computer wherever it may be (the

same or another network). A set of interconnected networks is called an Internet. An example of internet is shown in figure 2.1.

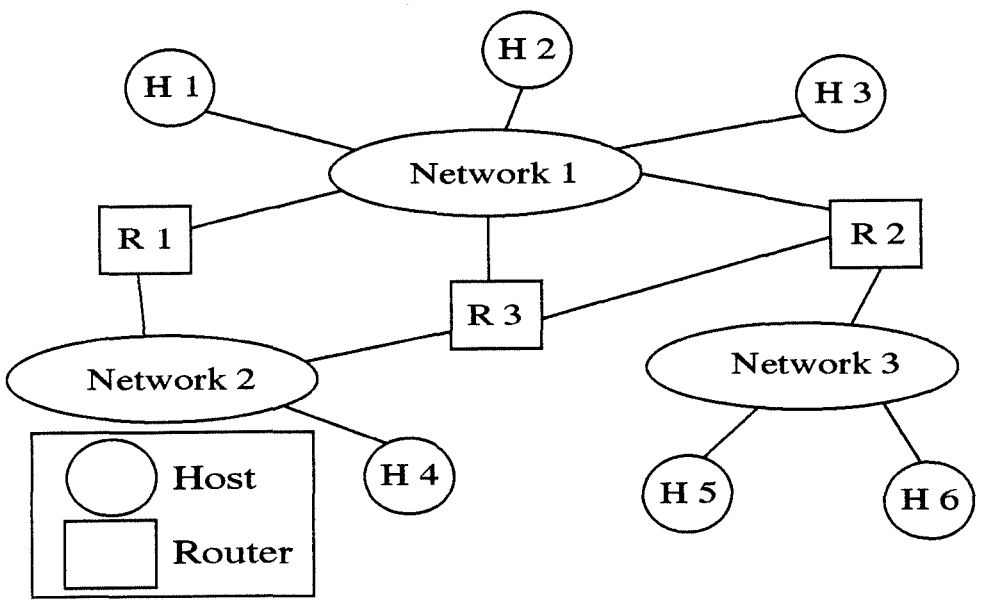


Fig. 2.1: Example of Internet

This internet is composed of three networks interconnected by three gateways (or routers in IP language). The routers are responsible for choosing the best way to reach the destination host and to transfer the messages between the different networks. In an internet, the hosts are, thus, virtually connected to the same big network.

In general, all the hosts in the internet must implement protocols from the four layers to provide services to the users. On the other end, the routers are not obliged to implement the four layers. Only the two first ones are needed because they are only concerned with the routing of datagrams which is made at the internet layer. Nevertheless, since network management takes place in many networks, the two upper layers are fairly often found in routers, as they are needed for management. In these routers, the Simple Network Management Protocol (SNMP) is running on top of UDP.

In the rest of this chapter, the major TCP/IP protocols will be described. After this introduction, the *second section* will concentrate on IP. The *third one* will explain a protocol which is, in general, found with IP; the Internet Control Message Protocol (ICMP) is used to report on the behaviour of the network. *Section four* and *section five* explain the two host-to-host protocols, respectively UDP and TCP. Many other protocols can be found in the TCP/IP protocol suite but they are not the subject

of this paper. Figure 2.2 gives an idea of all the protocols that can be found in the suite.

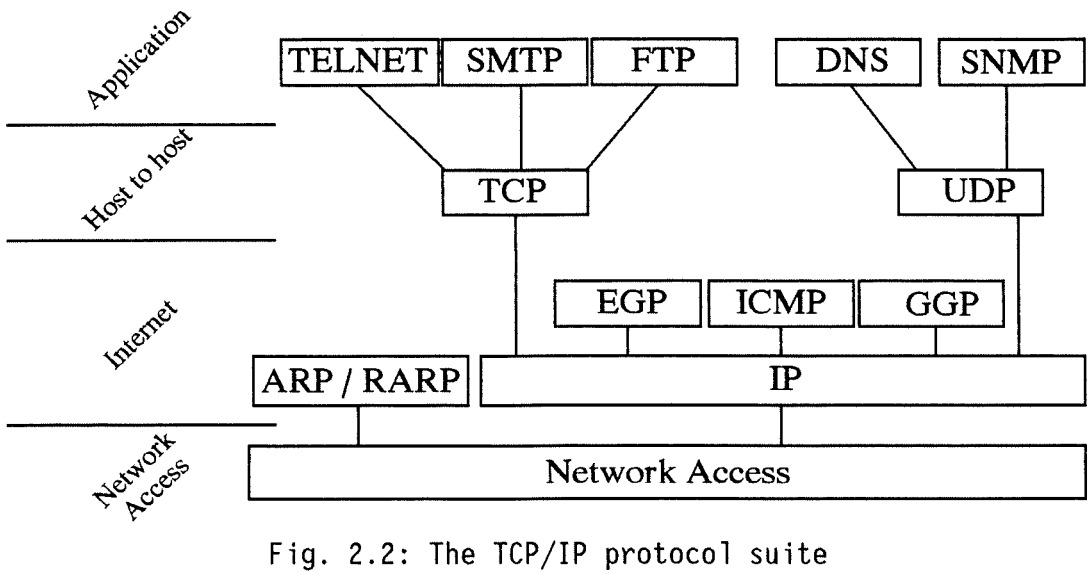


Fig. 2.2: The TCP/IP protocol suite

Next to the four protocols (IP, ICMP, UDP and TCP) that will be described hereunder in the following points of this second chapter, many others can be found. For example, the **Telnet** protocol allows a user to open a session on another computer. There is the **File Transfer Protocol** (FTP) which allows to transfer files between computers and the **Simple Mail Transfer Protocol** (SMTP) which allows users to exchange electronic mail. Above UDP, the **Domain Name Server** (DNS) protocol permits to translate high-level computer names to IP addresses and vice versa. The **Simple Network Management Protocol** (SNMP), used to manage networks, will be described, in detail, in the third chapter. At the internet layer, many protocols help IP in its delivery task. For example, the **Exterior Gateway Protocol** (EGP) is a way by which routers can exchange information about which networks are reachable via a particular router. The **Gateway-to-Gateway Protocol** (GGP) is a way by which routers exchange routing information. The **Address Resolution Protocol** (ARP) and **Reverse Address Resolution Protocol** (RARP) provide a mechanism to translate IP addresses into network specific addresses and vice versa.

2.2. IP Protocol

"The internet layer is responsible for providing transparency over both the topology of the internet and the transmission media used in each physical network comprising the internet."[ROSE91]

To be supported by the majority of media technologies, the internet layer must be as simple as possible. That is why the Internet Protocol has been designed as connectionless and for providing a non-reliable service. The aim of the internet protocol is to transport PDUs called *datagrams* between any hosts in the internet. For this, it provides a set of primitives to its users (layer above).

The service primitives are developed in the *first* point whereas the structure of an IP datagram are described in the *second* point. The *last three* points give a description of important mechanisms that are part of the IP protocol. These mechanisms are: addressing, routing and fragmentation. Other less important mechanisms are explained during the presentation of the primitives or during the presentation of the PDU.

2.2.1. IP Services

As IP provides a connectionless service to the upper layers, there are no needs for connection establishment and connection release primitives. Therefore only two IP primitives are defined by the IP specification [RFC791] : IP-DATA.request and IP-DATA.indication. These primitives are less precisely specified than, for example, the corresponding protocol data unit (PDU) because the vendors must be free to implement the layers in order to make them as efficient as possible. Only a functional specification is given for the IP services. The primitive preceded by an arrow pointing downwards represents a request primitive from the IP user to IP. And the one preceded by an arrow pointing upwards represents an indication primitive from IP to the IP user.

↓ IP-DATA.request (Source Address, Destination Address, Protocol, Type Of Service, Identifier, Don't Fragment, Time To Live, Data length, Option Data, Data)

↑ IP-DATA.indication (Source Address, Destination Address, Protocol, Data length, Option Data, Data)

The parameters associated with the two primitives are : [STAL89]

→ *Source & Destination Addresses*: Internetwork (or IP) addresses of sending and receiving IP entities. (see pt 2.2.3)

→ *Protocol*: It identifies the recipient protocol entity. (an IP user, e.g. TCP or UDP)

→ *Type Of Service*: It is used to specify the treatment of the data units during the transmission. The indicators are precedence, reliability, delay and throughput.

→ *Identifier*: It is used to identify the data units uniquely (along with the addresses and the protocol).

→ *Don't Fragment*: It indicates whether the various IP routers can segment data to accomplish delivery.

→ *Time To Live*: Maximum lifetime of data within the internet.

→ *Data Length*: Length of data being transmitted.

→ *Option Data*: Options requested by the user. The currently defined options are: security, source routing, route recording, stream identification and timestamping.

→ *Data*: User data to be transmitted.

The use of the primitives and the general model of operation for transmitting a datagram is illustrated in figure 2.3.

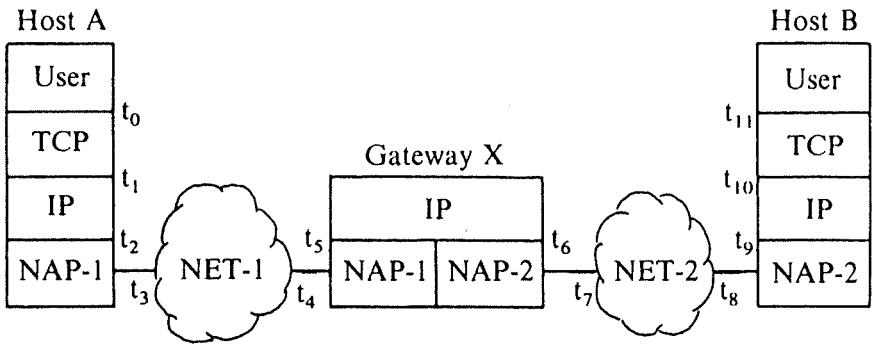


Fig. 2.3: Transmission of a datagram

In this example, data is sent from Host A to Host B. [RFC791, pp.5-6]

→ The sending IP user (here TCP[1]) uses the **IP-DATA.request** primitive to ask the sending of data across the network. *(t1)*

→ The IP entity builds an IP datagram which includes the final internetwork address. It determines a local network address (here the local address of a router). (see pt 2.2.4, the routing function)

---

1. TCP : Transmission Control Protocol

→ The IP entity gives the datagram and the local network address to the local network interface. *(t2)*

→ The local network entity creates a data unit and sends it via the local network. *(t3-t4)*

→ When the data unit arrives at the router, the local network entity strips off the header and turns the datagram over to the internet entity. *(t5)* By examining the destination IP address, the IP entity determines that the datagram is to be forwarded. It determines the local network address for the next IP entity and sends the datagram to the corresponding local network entity. *(t6)*

→ The local network entity creates a data unit and sends it via the local network to the destination host. *(t7-t8)*

→ At the destination host, the local network entity strips off the header and passes the datagram to the IP entity. *(t9)*

→ The IP entity determines that the datagram is for an application program in this host. It passes the data and other parameters to the destination entity (here TCP). For this, it uses the **IP-DATA.indication** primitive. *(t10)*

2.2.2. IP Datagram

A protocol data unit sent between IP entities is called a DATAGRAM. Its structure is shown in figure 2.4. The fields that are marked with an asterisk, come from or are derived from the IP-DATA.request primitive, whereas the others come from the IP entities themselves.

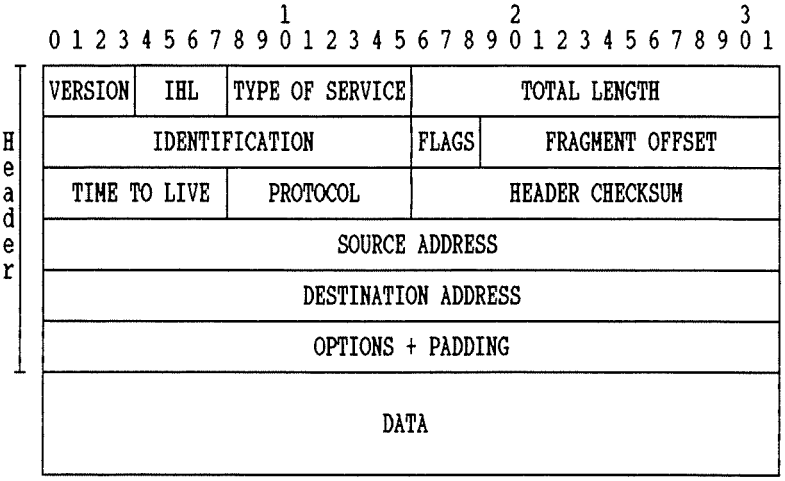


Fig. 2.4: Internet datagram

The fields in the IP datagram are: [STAL89] & [RFC791]

→ *Version (4 bits):* Version number of the protocol.

→ *IHL (Internet Header Length) (4 bits):* Length of the datagram header in 32-bit words (IHL ≥ 5).

→ *Type of service * (8 bits):* The type of service field provides an indication of the quality of service desired. The parameters are reliability, precedence, delay and throughput. This type of service indication will be used by routers to select the transmission parameters for a particular network, to select the network to be used for the next hop or to select the next router when routing an internet datagram.

→ *Total length (16 bits):* Length of the datagram measured in octets.

→ *Identification * (16 bits):* The identification field, together with the addresses and the user protocol, uniquely identifies a datagram.

→ *Flags (3 bits):* The first bit of the flag is unused. The second, the Don't Fragment (DF) flag *(*)*, if set, prohibits fragmentation. The third one, the More Fragment (MF) flag is used for fragmentation and reassembly.

→ *Fragment Offset (13 bits):* If the datagram has been fragmented, the fragment offset field indicates where in the primitive datagram this fragment belongs. It is measured in 64-bit units.

→ *Time to Live * (8 bits):* The Time to Live field indicates the maximum time the datagram is allowed to remain in the internet. It is an indication of an upper bound on the lifetime of a datagram. "It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit."[RFC791]

→ *Protocol * (8 bits):* The protocol field indicates the upper level protocol used in the data portion of the datagram.

→ *Header Checksum (16 bits):* Frame check sequence on the header only (except the header checksum field). It is used to perform error detection. When an IP entity receives a datagram, it recomputes the checksum. If it is not the same as the one in the header checksum field, then the datagram is discarded. The header checksum is recomputed each time the header is modified. For example, it must be recomputed when the Time to Live changes or when the datagram is fragmented.

→ *Source Address * (32 bits):* It represents the IP address of the sending IP entity. (see pt 2.2.3)

→ *Destination Address * (32 bits):* It represents the IP address of the recipient IP entity. (see pt 2.2.3)

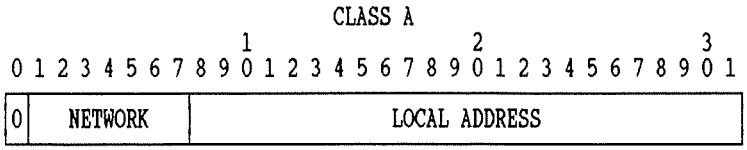
→ *Options * (variable):* The options field encodes the options requested by the sender. (see IP services) "The options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security and special routing."[RFC791]

→ *Padding (variable):* The header padding is used to ensure that the internet header ends on a 32-bit boundary.

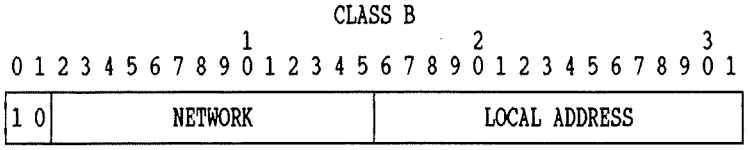
→ *Data * (variable):* The data field contains the data given by the IP user to be sent. It must be a multiple of eight bits in length. Total length of a datagram is a maximum of 65,535 octets.

2.2.3. Addressing

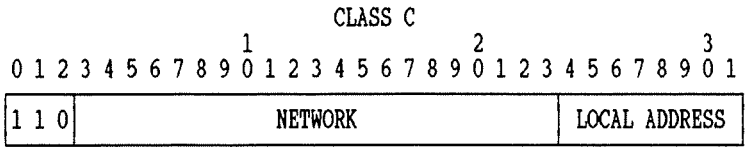
An IP address is a quantity of 4 octets (32 bits). It is divided into two fields: a *network number* and a *local address*. The network number refers to a particular physical network in an internet and the local address refers to a particular device attached to that physical network. A flexible scheme for allocating the 32 bits has been developed. IP addresses are divided into 5 classes, of which only three are used.[ROSE91]



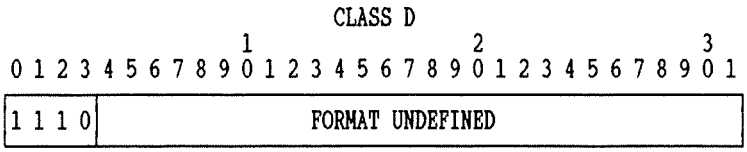
Class A specifies a small number of networks with a large number of hosts. This corresponds to 128 Class A networks, each containing up to 16777214 hosts.

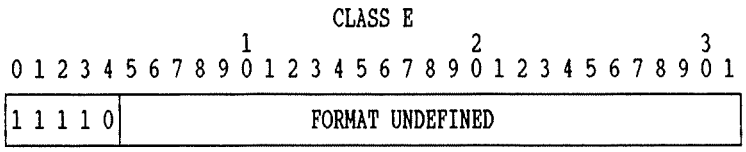


Class B specifies a moderate number of networks with a moderate number of hosts. This corresponds to 16384 Class B networks, each containing up to 65534 hosts.



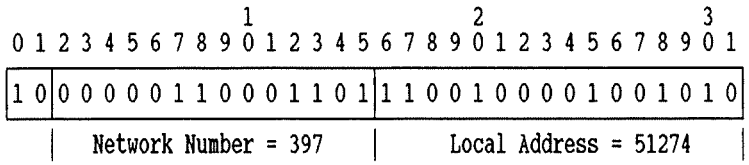
Class C specifies a large number of networks with a small number of hosts (2097152 networks with up to 254 hosts). And in addition, there exist two other classes which remain unused.



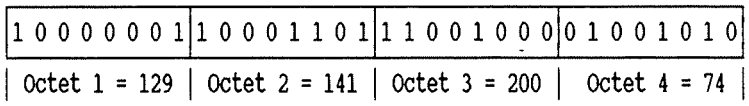


In each class, two values for the host-identifier are reserved for a special purpose. If all the bits are zero, then the resulting quantity refers specifically to the network identified in the IP address. And, if all the bits are one, then the IP address refers to all hosts attached to the network (the IP broadcast address).

To be easily understood by humans, the dotted quad notation is used for the representation of addresses. Each octet is expressed as a decimal number, separated by a dot. For example, a computer which has the local address 51274 and is connected to the class B network 397 would have the following address.



And in dotted quad notation, this address would be expressed by 129.141.200.74.



The two-level addressing seems sufficient. But in practice, it is impossible for a site to have more than one physical network (identified by the network number). If a site is running several physical networks, then it needs several IP network numbers, one for each physical network. This solution is not interesting since it increases the number of networks and the address space is of limited length. A better solution is to introduce a three-level addressing. "This allows each site to partition the host-identifier portion of their IP network address. A network so sub-partitioned is termed a *subnet*."[ROSE91] From the outside, the IP address appears to have two components, the network and host identifiers. Inside the site using subnets, the host-identifier is divided into two parts: a *subnet-number* and a *host-number*. The subnet-number refers to a particular physical network within the site's IP network and the host-number refers to a particular device on that subnet. To divide the host-identifier between the subnet- and host-numbers, a *subnet mask* is used. "The subnet mask is a 32-bit quantity which is logical-ANDed with an IP address in order to derive the actual physical network being identified."[ROSE91]

An example of the use of a subnet mask is shown in figure 2.5.

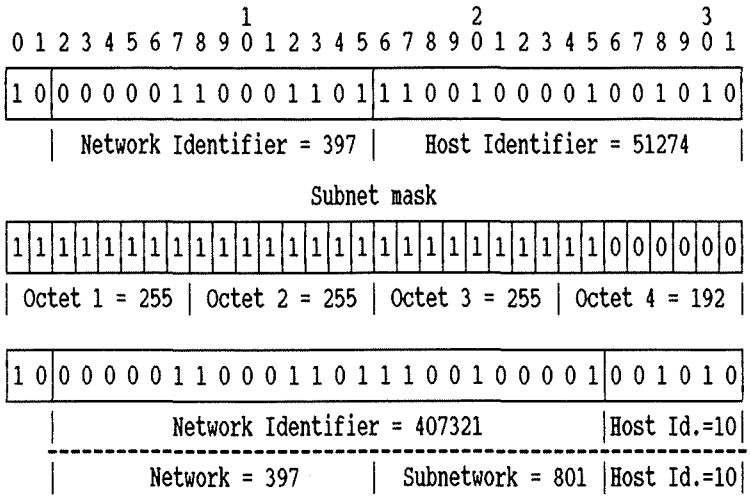


Fig. 2.5: Use of subnet mask

As the dotted quad notation is also used for subnet masks, this one would be 255.255.255.192. From the outside, the address will remain the same, 129.141.200.74 and refer to computer 51274 in class B network 397, as in the preceding example. But from the inside, the meaning has changed. The address refers to host 10 in the network 407321 or to host 10 in the subnetwork 801 from class B network 397. Network 397 appears to have 1024 subnetworks, each able to contain 64 devices. Thus, within this site, networks 406528 (for subnetwork 0) to 407551 (for subnetwork 1023) are available.

Thus, the two-level addressing (network identifier, host identifier) has become, with subnet masks, a three-level addressing (network identifier, subnet identifier, host identifier).

2.2.4. Routing

The routing problem concerns the choice about the way that the IP datagrams must take to reach their destination. The IP modules must decide to which router or to which host the datagram must be transmitted. This is called the "next hop". This problem is not invoked in the protocol specification [RFC791].

Routing can be either direct or indirect. "If the destination is on the same IP network, then the next hop is the destination IP address. This is termed *direct* routing. Otherwise, the next hop must be to a router, on the same IP network as the local device, which is somehow "closer" to the destination device. This is termed *indirect* routing."[ROSE91]

Each network device (router, host) maintains a *routing table*. The routing table gives, for each possible destination network, the next router to which the IP datagram should be sent. The routing table may be static or dynamic. A static table contains fixed routes to predefined destinations. A dynamic table is more flexible in responding both to error and congestion situations. For example, when a router goes down, all of its neighbours will send out a status report. This allows other devices to update their routing tables. This mechanism can also be used to control congestion.

"Usually both hosts and routers start with some initial configuration information on stable storage. Then, they dynamically learn about the network topology through protocol interactions (see other protocols of the suite). In addition, there is also the notion of a *default route*, which can be used to reach a destination if its IP network is not in the routing table."[ROSE91]

2.2.5. Fragmentation and Reassembly

An Internet is composed of many individual networks with their particularities. One of the particularities is the maximum packet size, also called maximum transmission unit (MTU). It would be impossible to fix a uniform packet size across the networks. Therefore, routers must, if necessary, be able to fragment datagrams into smaller segments before transmitting them. The reassembly of the fragments can be made at different moments. A solution is to make the reassembly only at the destination. Another solution is to make the reassembly as soon as possible in the routers. The second solution has the disadvantages that the routers need larger buffers and that dynamic routing is prohibited, whereas the principal disadvantage of the first is the reduced efficiency of the networks which must transmit many small datagrams. The first solution, reassembly at the destination host, has been chosen for IP.

A. Fragmentation

The fragmentation of an internet datagram follows the process described below. An internet protocol entity creates new internet datagrams and copies the contents of the internet header fields from the initial datagram into the new internet headers. The data of the initial datagram is divided into portions. The first portion of the data is placed in the first new internet datagram and the total length field is set to the length of the first datagram. The more-fragment flag is set to one. The second portion of the data is placed in the second new internet datagram and the total length field is set to the length of the second datagram. The more-fragment flag is set to

one... The last portion of the data is placed in the last new internet datagram and the total length is set to the length of the datagram. The more-fragment flag carries the same value as the initial datagram. The fragment offset field of the new internet datagrams is set to the value of that field in the initial datagram plus the total number of blocks copied in the preceding fragments.[RFC791]

An example of Fragmentation Procedure

An example of a fragmentation procedure is shown in figure 2.6. This procedure has been given, as an example, in the IP standard [RFC791]. The option chosen here is to cut the datagram into two non-equal parts. The length of the first one is equal to the maximum packet size and the second one contains the rest of the datagram. This last is reduced, if necessary, using the same process. Another option would have been to cut the datagrams into two equal parts and redo this until the length of each part is smaller than the maximum packet size.

Then: TL =< MTU	Else: DF = 1	
	T	E
SUB-MIT THE FRAG-MENT TO THE LAYER BELOW	DISCARD DATAGRAM	***Produce the First Fragment***
		Copy the original header
		OIHL <- IHL OTL <- TL OFO <- FO OMF <- MF
		NFB <- (MTU - IHL *4) / 8
		Attach the first NFB * 8 data octets
		Correct the header MF <- 1; TL <- (IHL*4) + (NFB*8) Recompute Checksum
		Submit the fragment to layer below
		Produce the Second Fragment
		Copy the header (selectively)
		Append the remaining data
		Correct the header IHL <- (((OIHL*4)-(length of options not copied))+3)/4 TL <- OTL -NFB*8 - (OIHL-IHL)*4) FO <- OFO + NFB MF <- OMF Recompute Checksum
Repeat Until (last fragment submitted) or (datagram discarded)		

(O)FO = (Old) Fragment Offset
(O)IHL=(Old) Internet Header Length
(O)MF = (Old) More Fragment flag
(O)TL = (Old) Total Length
DF = Don't Fragment flag
NFB = Number of Fragment Block
MTU = Maximum Transmission Unit

Fig. 2.6: Fragmentation Procedure

When an IP entity attempts to send the datagram, it checks the MTU of the network. If the MTU is greater than or equal to the total length of the datagram, then no further processing is required and the datagram is passed to the layer below. Otherwise, the IP entity checks to see if the flags field in the datagram permits fragmentation. If not, the datagram is discarded. Otherwise, the IP entity cuts the datagram into two fragments, the first fragment being the maximum size and the second fragment being the rest of the datagram. The fragment offset field contains a number corresponding to where the user-data belongs, in 8-octet increments, in the original datagram. Then, for each fragment, except for the last in the sequence, the more fragments bit is set in the flags field. The first fragment is submitted to the layer below to be transmitted, while the second fragment is submitted to this procedure in case it is still too large.[RFC791] & [ROSE91]

B. Reassembly

The reassembly of a fragmented IP datagram is performed by the IP module at the destination computer. "To reassemble the fragments of an internet datagram, an internet protocol module combines internet datagrams that all have the same value for the four fields: identification, source, destination and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero and the last fragment will have the more-fragments flag reset to zero."[RFC791]

An example Reassembly Procedure

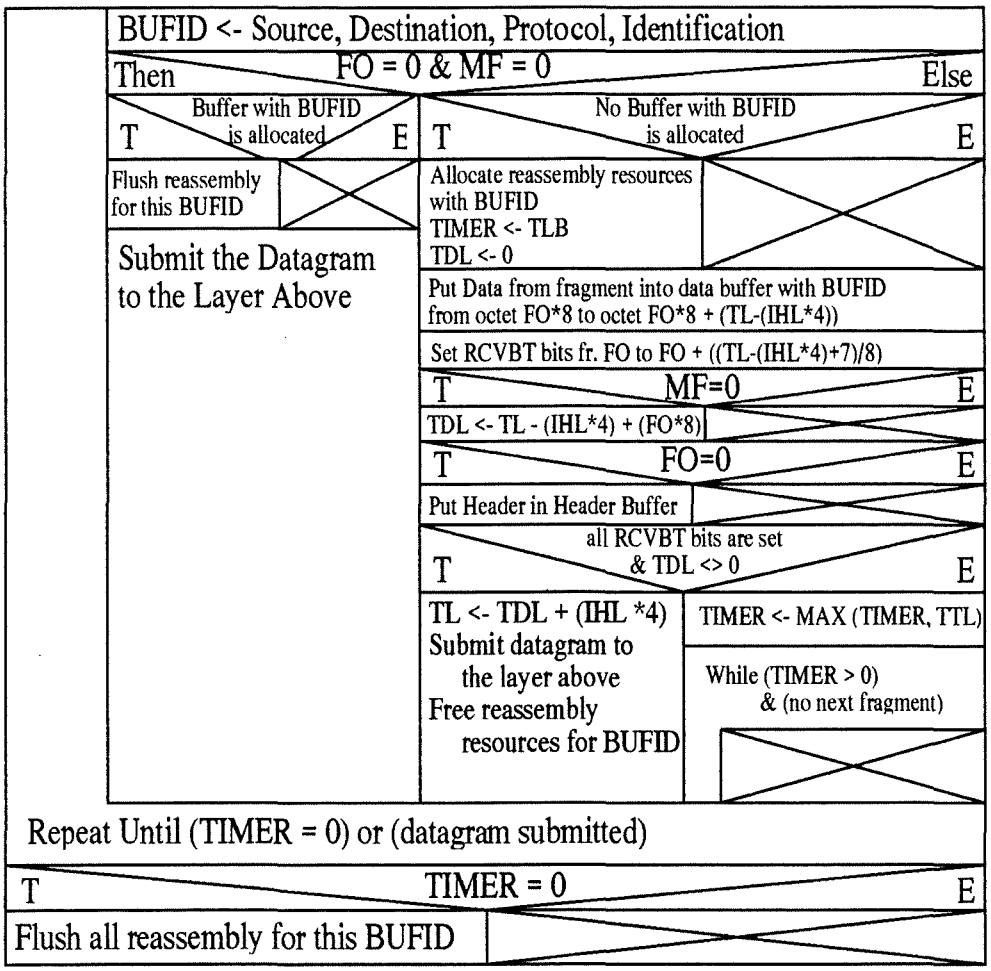
An example of a reassembly procedure is shown in figure 2.7. This procedure has been introduced in the IP specification [RFC791].

A datagram is identified by its source and destination addresses, its protocol and its identification field. The concatenation of these four values is termed the buffer identifier (BUFID).

If the received datagram is a whole datagram (FO = 0 & MF = 0), then the datagram is directly forwarded to the layer above and reassembly resources, if already allocated, are released.

If the received datagram is not a whole datagram and if no other fragment with this buffer identifier has already arrived, then reassembly resources are allocated.

The reassembly resources consist of a data buffer, a header buffer, a fragment block received bit table (RCVBT), a total data length field and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length and bits are set in the fragment block received bit table corresponding to the fragment blocks received.



BUFID = BUFfer IDentifier
FO = Fragment Offset
IHL = Internet Header Length
MF = More Fragment flag
RCVBT = frag. block ReCeiVed Bit Table
TDL = Total Data Length
TL = Total Length
TLB = Timer Lower Bound
TTL = Time To Live

Fig. 2.7: Reassembly Procedure

If the received fragment is the last one (MF = 0), the total data length (TDL) can be computed. If the fragment is the first one (FO = 0), its header is placed in the header buffer.

If all fragments are received (all RCVBT bits set to one), then the total datagram length (TL) is calculated, the datagram is passed to the layer above and reassembly resources are released.

Otherwise, the procedure waits until the timer expires or a new fragment arrives. If no new fragment arrives within a given time (TIMER = 0) or if the whole datagram is reassembled then the procedure stops. If the timer runs out, resources are released and the datagram is lost.[RFC791]

2.3. ICMP Protocol

The Internet Control Message Protocol (ICMP) is, in general, associated with each IP module. ICMP provides a low-level feedback on how the internet layer is operating and how it might behave. ICMP provides a small number of control messages for error-reporting. And even if they are at the same level, ICMP uses the delivery services of IP.

The ICMP module is activated each time the IP module detects an error or an event for which an ICMP message has been foreseen. The ICMP entity builds this message and gives it up to the IP module which sends it into the data field of an IP datagram to the entity which causes the problem.

The ICMP packet contains five different fields. Its format is given in figure 2.8.

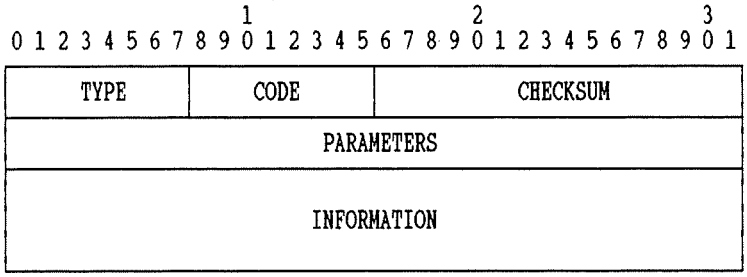


Fig. 2.8: ICMP Message

The meaning of the fields is straight-forward:

→ *Type (8 bits):* Identifies which control message is being sent.

→ *Code (8 bits):* Identifies a basic parameter for the control message.

→ *Checksum (16 bits):* Checksum computed over the entire ICMP packet.

→ *Parameters (32 bits):* Used to specify more lengthy parameters.

→ *Information (variable):* Provides additional information related to the message. In general, for errors concerning a particular IP datagram, the beginning of this datagram is copied in the information field of the related ICMP message.

The control messages supported by ICMP include:

→ *Destination unreachable:* This message is used in different circumstances. It is used to report that a datagram could not be delivered because a network or a host was unreachable, a protocol was not running or fragmentation was necessary but disallowed by the flags field.

→ *Time exceeded:* This message is sent to report that the time to live field of a datagram reached zero and that it has been discarded. A host can send this message if it cannot complete reassembly within a given time.

→ *Parameter problem:* This message is used to report an error in an IP header.

→ *Source quench:* The source quench message is used to report that a network device is discarding datagrams due to lack of resources (e.g. buffers). It provides a rudimentary form of flow control.

→ *Redirect:* This message is used to advise of a better route to a destination IP address.

→ *Timestamp/timestamp reply:* These messages are used to sample the delay in the network between two network devices.

→ *Information request/information reply:* These messages are used by a host to discover the address of the local IP network.

→ *Address mask request/address mask reply:* These messages are used to determine the subnet mask associated with the local IP network.

→ *Echo/echo reply:* These messages are used to test the reachability of an IP address, to test the communication between two hosts. An echo message is sent and when receiving such a message, the local IP entity responds by sending an echo reply message.

A useful network management tool can be built with ICMP messages.

This program is called Packet INternet Groper (PING). To test reachability of destinations, the PING program sends an ICMP echo message to an IP address and awaits an echo reply message. By repeating this operation, PING can be used to observe variations in network round-trip times and loss rates. This is useful to discover the source of network congestion. Furthermore, it can be combined with IP header options such as Source Routing and Record Routing to write other powerful programs in order to isolate network problems.

2.4. UDP Protocol

At the IP layer, an address identifies a router or a host. It corresponds, in OSI language, to a Network Service Access Point (NSAP). The User Datagram Protocol (UDP) adds a mechanism to distinguish between different destinations within a given host. This mechanism is a port addressing capability.

A port identifies a program executing on a machine and is equivalent to a Transport SAP. Whereas at the internet layer, an address is sufficient to deliver datagrams, at the UDP layer, a socket is needed. A socket is composed of an internet address (32 bits) and a port number (16 bits). Thus, a socket is a quantity of 48 bits.

Because two computers need to agree on port numbers before they can cooperate, a universal assignment mechanism must be used. An authority assigns some special port numbers. These are called *well-known ports*. For example, if a file has to be transferred between two computers, the two TFTP (Trivial File Transfer Protocol) programs know that the port number of the peer program is 69.

The UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. It provides a connectionless service. An example of the use of UDP is in the context of network management.

UDP sits on top of IP. Therefore it has very little to do. It provides the same unreliable, connectionless datagram delivery semantics as IP. UDP messages can be lost, duplicated or arrive out of order.

In summary, "the User Datagram Protocol (UDP) provides unreliable connectionless delivery service using IP to transport messages between machines. It adds the ability to distinguish among multiple destinations within a given host computer."[COME91]

The *first two* points of this section will present respectively the service primitives and the format of the UDP datagram, while the *last* point will describe the only additional mechanism provided by UDP, the checksum computation.

2.4.1. UDP Primitives

As UDP provides a connectionless service to the upper layers, there is no needs for connection establishment and connection release primitives. Therefore only

two UDP primitives are defined: UDP-DATA.request and UDP-DATA.indication. The primitive preceded by an arrow pointing downwards represents a request primitive from the UDP user to UDP. And the one preceded by an arrow pointing upwards represents an indication primitive from UDP to the UDP user.

↓ UDP-DATA.request (Source Address, Destination Address, Source Port, Destination Port, Protocol, Type Of Service, Data length, Option Data, Data)

↑ UDP-DATA.indication (Source Address, Destination Address, Source Port, Destination Port, Protocol, Data length, Option Data, Data)

The parameters associated with these primitives are the same as the ones defined for the IP primitives. The two new parameters (Source Port and Destination Port) represent the port number of the originating and destination UDP users.

2.4.2. UDP Datagram

The structure of the UDP datagram is shown in figure 2.9. To be transmitted on the network, a UDP datagram is included in the DATA field of an IP datagram (see fig. 2.4). The fields that are marked with an asterisk, come from or are derived from the UDP-DATA.request primitive, whereas the others come from the UDP entities themselves.

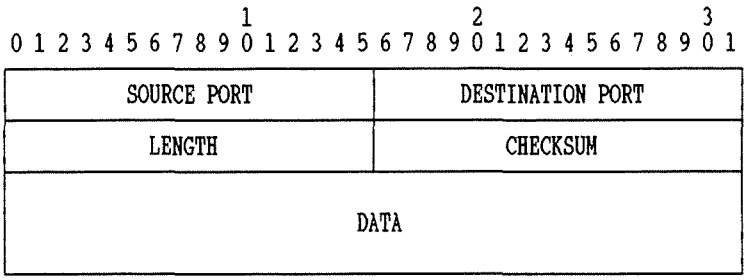


Fig. 2.9: UDP Datagram

The meaning of the fields is the following:

→ *Source Port * (16 bits):* Source Port is an optional field, when meaningful, it indicates the port of the sending process and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

→ *Destination Port * (16 bits):* Destination Port has a meaning within the context of a particular internet destination address.

→ *Length (16 bits):* Length is the length in octets of this user datagram including the header and the data. (This means the minimum value of the length is eight.)

→ *Checksum (16 bits):* Checksum on a pseudo-header of information from the IP header, on the UDP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

→ *Data * (variable):* The data field contains the data given by the UDP user to be sent.

2.4.3. UDP Checksum

UDP does not compute its checksum in the usual way. Before calculating the sum, UDP prefixes a pseudo-header to the datagram. The purpose of using a pseudo-header is to verify that the UDP datagram has reached its correct destination.

This verification is justified by the following reasoning. "The correct destination consists of a specific machine and a specific protocol port within that machine. The UDP header itself specifies only the protocol port number. Thus, to verify the destination, UDP on the sending machine computes a checksum that covers the IP addresses as well as the UDP datagram. At the ultimate destination, UDP software verifies the checksum using the IP addresses obtained from the header of the IP datagram that carried the UDP message. If the checksums agree, then it must be true that the datagram has reached the intended destination host as well as the correct protocol port within that host."[COME91]

The pseudo-header is described hereunder, in figure 2.10.

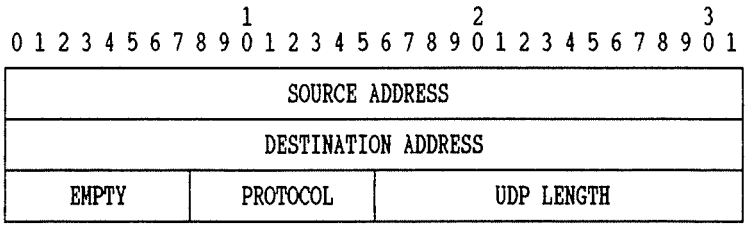


Fig. 2.10: UDP Pseudo-header

The fields of the pseudo-header are relatively self-explanatory: the empty field is simply a zero-valued octet, the protocol field is the value used by IP to identify UDP (17 decimal) and the UDP length field is the length of the UDP packet. TCP also uses this 96-bit pseudo-header in its checksum calculation when achieving user-data integrity.[ROSE91]

To compute the checksum, UDP first stores zero in the checksum field, then computes a 16-bit one's complement sum of the entire object, including the pseudo-header, UDP header and user data. After that, it stores the sum in the checksum field. Note that the pseudo-header is not transmitted with the datagram.

2.5. TCP Protocol

TCP is a transport protocol. But, unlike UDP, it provides a reliable mechanism for the exchange of data between processes in different computers.

Reliable mechanism means, for a host to host protocol, that data are delivered error free, in sequence, with no loss or duplication. A reliable mechanism was needed because the lower level is fundamentally unreliable and the higher level applications need to send large volumes of data. If the service is not reliable, error detection and recovery have to be build in each application program. So, a general purpose solution to the problem of providing a reliable stream delivery, has been developed in the form of the TCP protocol.

TCP is a connection oriented protocol. This means that the data transfer must be preceded and followed by connection establishment and connection termination phases. As already seen with UDP, a process within a host using TCP services is identified with a PORT (called TSAP in OSI). A port, when concatenated with an internet address, forms a 48 bit SOCKET, which is unique throughout the internet. When two TCP entities communicate, the exchanged units of data are termed *segments*. Segments are interpreted relative to a *connection*. In TCP, a connection is defined as the pairing of the two internet sockets. This 96-bit quantity uniquely identifies the connection in an internet. Two simultaneous connections cannot take place between the same pair of sockets.

TCP adds functionalities to the protocols presented above, but its implementation is also much more complex. To describe these functionalities, the same structure as in IP and UDP, will be used. The *first point* will explain the different service primitives. The *second one* will present the TCP segment and the *last point* will give a description of the mechanisms used during the three communication phases.

2.5.1. TCP Service Primitives

The TCP service primitives are richer than those provided by IP. The set of primitives and parameters is considerably more complex. As in every connection-oriented protocol, three classes can be identified: connection establishment, data transfer and connection release. A fourth class is used to report on errors and status. The primitives will be presented using this structure. The primitives preceded by an arrow pointing downwards represent request primitives from the TCP user to TCP.

And the ones preceded by an arrow pointing upwards represent indication primitives from TCP to the TCP user. A functional definition of the primitives is given in [RFC793]. The interface given hereunder is derived from the one proposed in [STAL89].

A. Connection establishment

↓ TCP-PASSIVE OPEN.request (source port, [destination port], [destination address], [timeout], [timeout-action], [precedence], [security-range])

↓ TCP-ACTIVE OPEN.request (source port, destination port, destination address, [timeout], [timeout-action], [precedence], [security-range], [data], [data length], [Push flag], [Urgent flag])

↑ TCP-OPENID.indication (local connection name, source port, destination port, destination address)

↑ TCP-OPEN FAILURE.indication (local connection name)

↑ OPEN SUCCESS.indication (local connection name)

B. Data transfer

↓ TCP-DATA.request (local connection name, data, data length, Push flag, Urgent flag, [timeout], [timeout-action])

↑ TCP-DATA.indication (local connection name, data, data length, Urgent flag)

C. Connection termination

↓ TCP-CLOSE.request (local connection name)

↓ TCP-ABORT.request (local connection name)

↑ TCP-CLOSING.indication (local connection name)

↑ TCP-TERMINATE.indication (local connection name, description)

D. Status and Error reporting

↓ TCP-STATUS.request (local connection name)

↑ TCP-STATUS.indication (local connection name, source port, source address, destination port, destination address, connection state, receive window, send window, amount awaiting ACK, amount awaiting receipt, urgent state, precedence, security, timeout)

↑ TCP-ERROR.indication (local connection name, description)

The signification of the parameters associated with the different primitives is obvious.

2.5.2. TCP Segment

A protocol data unit sent between TCP entities is called a SEGMENT. Its structure is shown in figure 2.11. To be transmitted on the network, a TCP segment is included in the DATA field of an IP datagram (see fig. 2.4). The fields that are marked with an asterisk, come from or are derived from TCP request primitives, whereas the others come from the TCP entities themselves.

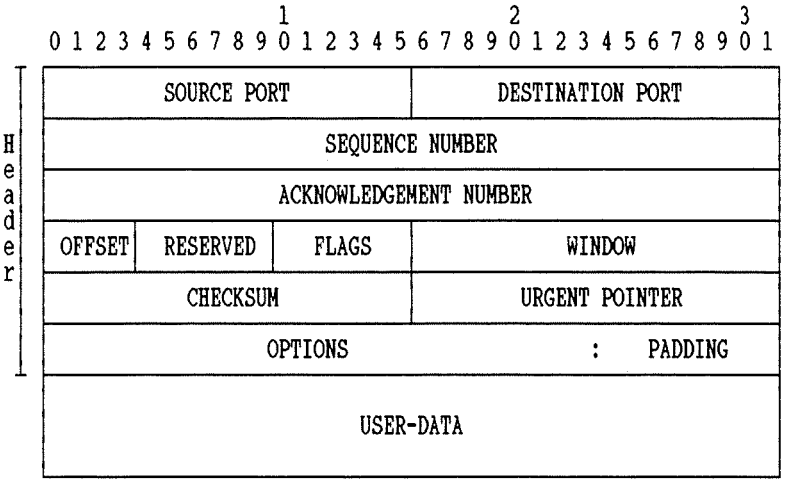


Fig. 2.11: TCP Segment

The fields in the segment are: [STAL89] & [RFC793] & [COME91]

→ *Source Port * (16 bits):* The source port number.

→ *Destination Port * (16 bits):* The destination port number.

→ *Sequence Number (32 bits):* The sequence number of the first data octet in this segment.

→ *Acknowledgement Number (32 bits):* If the ACK bit of the flags field is set, this field identifies the sequence number of the octet that the sender of the segment is expecting to receive next.

→ *Offset (4 bits):* The length of the segment header in 32 bit words.

→ *Reserved (6 bits):* Reserved for future use.

→ *Flags (6 bits):* Control bits indicating special functions for this segment: **URG** *: Urgent pointer field is valid, **ACK**: Acknowledgement field is valid, **PSH** *: This segment requests a push, **RST**: Reset the connection, **SYN**: Synchronize sequence numbers, **FIN**: No more data from sender.

→ *Window (16 bits):* The number of data octets which the sender of this segment is willing to accept.

→ *Checksum (16 bits):* A one's complement sum, computed over a pseudo-header and the entire TCP segment. (see UDP checksum)

→ *Urgent Pointer * (16 bits):* If the URG bit is set in the flags field, then this field when added to the sequence number field indicates the first octet of non-urgent data.

→ *Options (variable):* The only currently defined options is the maximum segment size that will be accepted.

→ *Padding (variable):* The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary.

→ *Data * (variable):* User-data to be sent.

2.5.3. TCP Mechanisms

This part explains the different mechanisms used to transfer data from a local TCP user to a remote TCP user. The three points give a description of the transmission operations in a connection oriented protocol: connection establishment, data transfer and connection termination. The second point, data transfer, explains more particular mechanisms. These mechanisms enhance the basic service offered by TCP. That is, for example the case for the urgent data mechanisms. Or they also fill in the gaps left by IP. That is the case for the retransmission and the flow control mechanisms. To easily understand the connection establishment and connection release mechanisms, a state diagram is given in the TCP specification [RFC793]. This diagram is reproduced in figure 2.12.

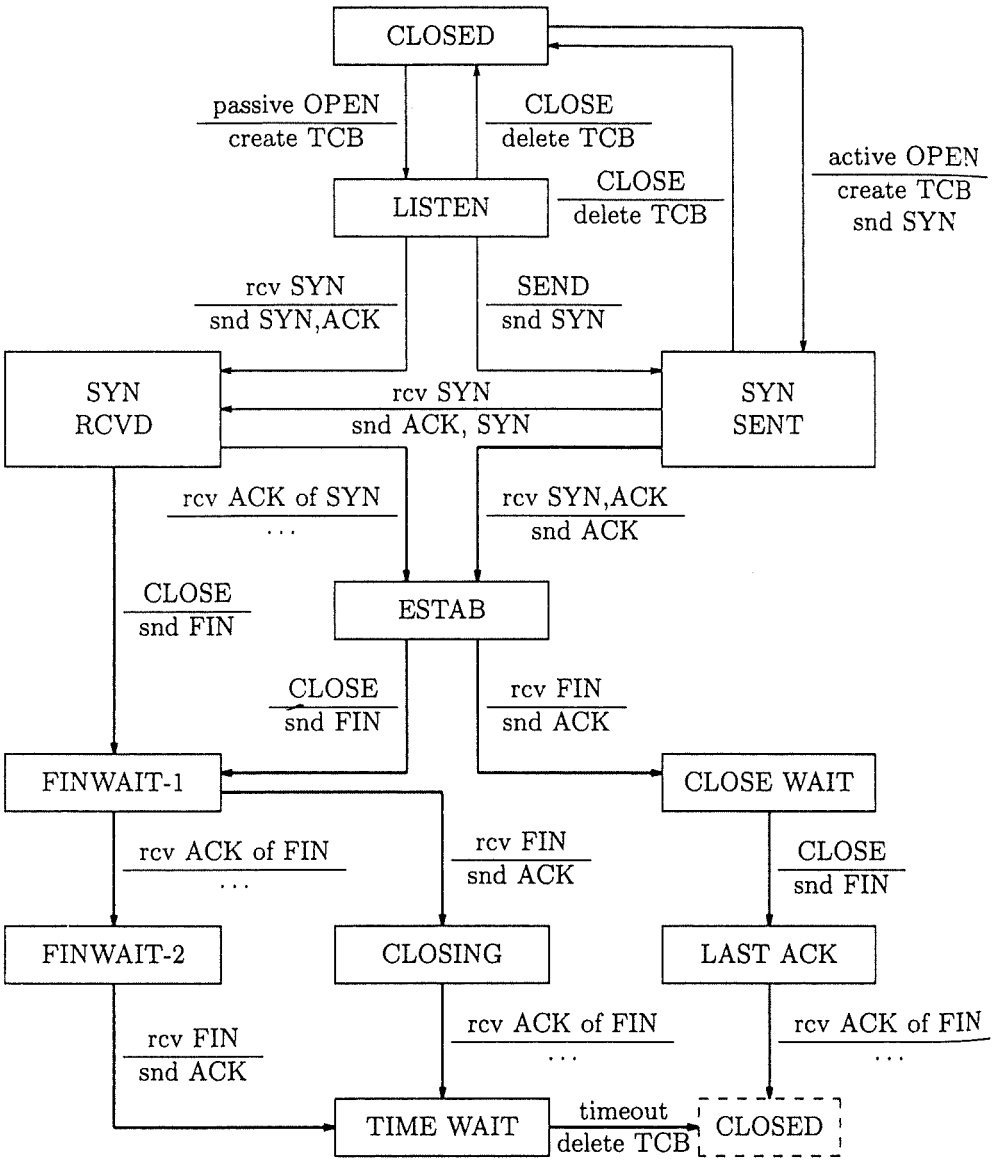


Fig. 2.12: TCP State Diagram

A. Connection establishment

The protocol interaction used to set up a logical connection between two TCP users is termed a *three-way handshake*. To keep state information relating to a connection, each TCP entity maintains a Transmission Control Block (TCB). "This is created during connection establishment, modified throughout the life of the connection and then deleted when the connection is released."[ROSE91] It serves in specifying the characteristics to be used for all data transfers on the connection and it enables each TCP entity to maintain state information concerning the connection.[STAL89]

The values given in parentheses represent the states of the two TCP users which want to establish a connection. The first one is the originator and the second one is the destination. **In the first case, the two entities do not want to establish a connection at the same time.** The mechanism of three-way handshake for connection establishment is illustrated in the diagram of figure 2.13.

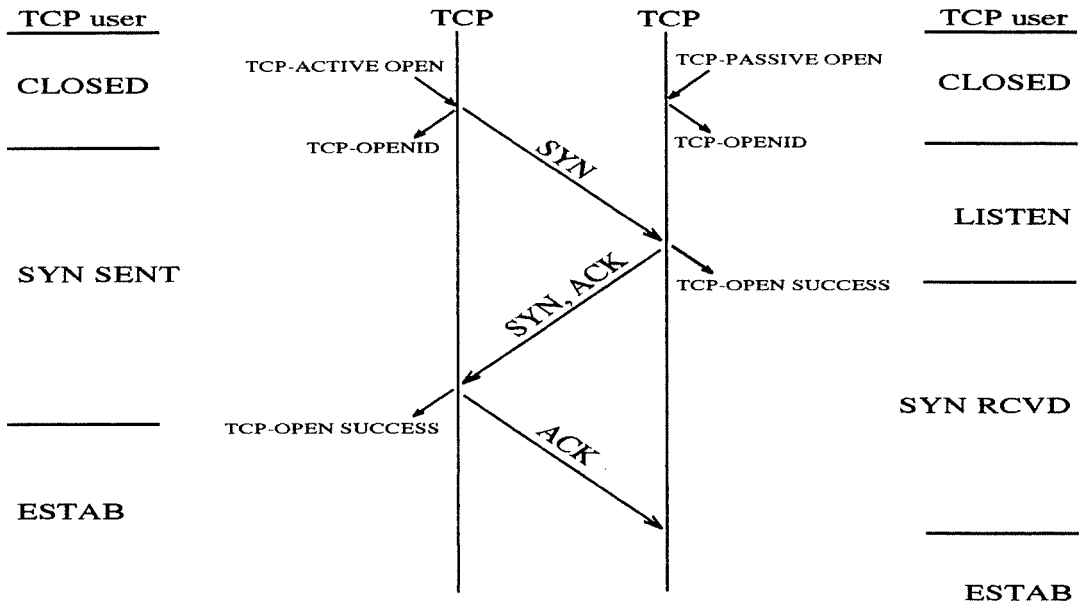


Fig. 2.13: Connection establishment

To begin, there is no connection. (CLOSED, CLOSED)

A connection enters the LISTEN state when a user signals that it will wait for a connection request with a TCP-PASSIVE OPEN.request primitive. Then, TCP issues an TCP-OPENID.indication primitive to the user. (CLOSED, LISTEN) The user can issue a TCP-CLOSE.request primitive if it changes its mind.

"From the CLOSED state, the user may also issue a TCP-ACTIVE OPEN.request primitive, which instructs TCP to attempt connection establishment to a designated user (socket). In this case, TCP also issues a TCP-OPENID.indication."[STAL89] The originating TCP entity computes an *initial sequence number*. "This must be chosen carefully so that segments from older instances of this connection, which might be floating around the network, will not cause confusion with this new connection."[ROSE91] The TCP sends a SYN (Synchronize) segment to the destination TCP entity. In the example, the user that initiated the TCP-PASSIVE OPEN.request has been chosen as destination. This connection request is carried by a TCP segment. (SYN SENT, LISTEN)

Upon receiving this segment, if no destination TCP user corresponding to the specified port is in the LISTEN state, the connection is aborted by sending an RST (Reset) segment. Otherwise, the destination TCP entity computes a sequence number, signals the user that a connection is open with a TCP-OPEN SUCCESS.indication primitive, sends a SYN/ACK segment and puts the connection in the SYN RCVD state. (SYN SENT, SYN RCVD)

Upon receiving the SYN/ACK segment, the original TCP entity sends an ACK (Acknowledgement) segment back, signals its user that the connection is open with a TCP-OPEN SUCCESS.indication primitive and moves the connection to the ESTABLISHED state. (ESTAB, SYN RCVD)

When the acknowledgement of the SYN is received by the responding TCP entity, it too can move the connection to an ESTABLISHED state. (ESTAB, ESTAB) The connection is prematurely aborted if either user issues a TCP-CLOSE.request primitive.

Once the three-way handshake has been successfully concluded, the connection enters the data transfer phase.

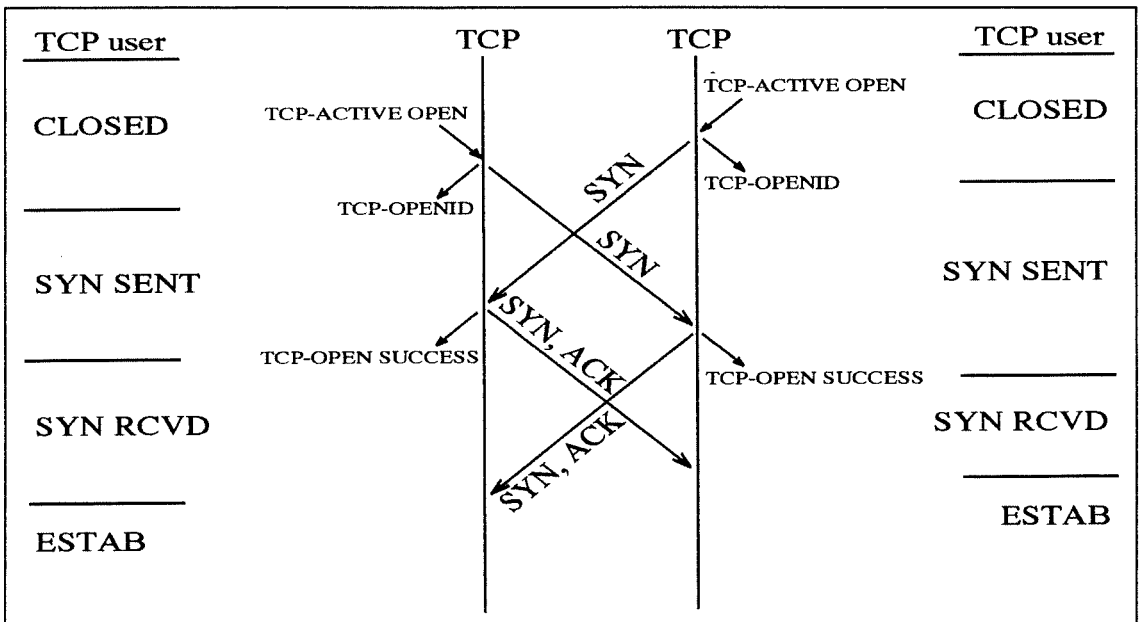


Fig. 2.14: Simultaneous Connection establishment

But the three-way handshake is also effective if **two entities try to set up a connection to each other at the same time.** This case is illustrated in figure 2.14.

To begin, there is no connection. (CLOSED, CLOSED)

Then, nearly at the same time, the users issue a TCP-ACTIVE OPEN.request primitive. The two TCP perform the same operations as was seen earlier. They both enter the SYN SENT state. (SYN SENT, SYN SENT)

Upon receiving the SYN segment, they acknowledge it with a SYN/ACK segment, signal their users that the connection is open (TCP-OPEN SUCCESS.indication) and enter the SYN RCVD state. (SYN RCVD, SYN RCVD)

When they receive the acknowledgement of the other TCP, the connection is established. (ESTAB, ESTAB)

B. Data transfer

Once the connection is established, each process can simultaneously send and receive segments. The connection is full-duplex. For sending data, a TCP entity cuts them, puts them in TCP segments and gives them up to IP to transmit them.

Without an appropriate mechanism, a TCP connection is not reliable. This is chiefly due to two events. "First, the segment may be damaged in transit but nevertheless arrive at its destination. TCP includes an error-detecting code in the segment header; therefore, the receiving TCP entity can detect the error and discard the segment. The second event is that a segment fails to arrive. In either case, the sending TCP does not know that the segment transmission was unsuccessful."[STAL89]

Retransmission

The reliability of a TCP connection is achieved through the mechanism of retransmission. This consists in retransmitting bad received or unreceived packets. Two segment fields are especially involved in retransmission: the sequence number field and the acknowledgement field.

A sequence number is attached to each data octet. The sequence number of the first octet contained in the segment data field is transmitted in the segment header. The acknowledgement field indicates the sequence number of the next data octet which can be sent in the other direction. An ACK n+1 is interpreted to mean that the TCP that issued the ACK has received all of the data up through sequence number n.

Each time a TCP entity sends a segment, it starts a retransmission timer. A timer is associated with each sent segment. Later, one of two events will happen: "either an acknowledgement for the segment will be received and the timer can be stopped; or the timer will expire. In this latter case, the TCP entity retransmits the segment and restarts the timer."[ROSE91]

The problem is knowing when to retransmit. In the case of lost data, if the sending transport entity retransmits too slowly, the throughput is reduced. If data is delayed due to network congestion and the transport entity retransmits too quickly, then it adds to the congestion and throughput gets even worse.[ROSE91]

Because of the variability of the networks that compose an internetwork system, TCP uses an adaptive algorithm to dynamically determine the retransmission timeout. The best value for this retransmission timer should be a bit longer than the round trip delay (send segment, receive ACK).

Flow control

"The flow control mechanism is used in TCP to allow a receiving TCP to regulate the rate at which data arrives from a sending TCP. One TCP entity would want to restrain the rate of segment transmission over a connection because (...) of a lack of receiving buffer space. Without some form of flow control, data may arrive faster than it can be processed. This leads to inefficiency, as the sender must retransmit a segment that successfully made it through to the receiver."[STAL89]

Each segment header contains a window field which represents the number of octets that may be sent in each direction before an acknowledgement is returned. If a TCP entity is able to receive many data, it will fix the window field to a large value. If its buffer space begins to fill up, it will have to reduce the value of the window field.

"Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted."[RFC793]

PUSH & URG

An application entity might need a mechanism for asking that all data it has previously sent has been delivered to the end-user and are not buffered anymore in the various communication entities. This is accomplished using a push function (PSH). An application entity may indicate that data previously sent should be pushed. Then the local TCP entity sets a PSH bit in the next new segment it sends. Upon receiving such a segment, the remote TCP entity knows that it has to push user-data up to its own application entity.

An urgent mechanism is also available with TCP. The semantics of urgent data are application-specific.

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data.

The mechanism employs an urgent pointer. The URG control flag indicates that the urgent pointer points to where urgent data ends in the stream. The absence of this flag indicates that there is no urgent data in the segment.

"The receiving application entity, upon being notified that urgent data is present in the stream, can quickly read from the stream until the urgent data is exhausted."[ROSE91]

C. Connection termination

When TCP users do not have any data to send anymore, they initiate a TCP-CLOSE.request primitive. Two cases can essentially be distinguished. The first case is when **a user initiates a close before the other does so**. The same principle as for connection establishment (parentheses) is used to represent the states of the TCP users which want to close a connection. The diagram illustrating the connection termination is given in figure 2.15.

To begin, a connection is established between the two users. (ESTAB, ESTAB)

When an application entity indicates that it has no more data to send by a TCP-CLOSE.request primitive, the local TCP entity ensures that all the segments it has sent have been acknowledged. Then the local TCP entity generates a FIN (Finish)

segment and it enters the FINWAIT-1 state. All segments preceding and including FIN will be retransmitted until acknowledged. (FINWAIT-1, ESTAB)

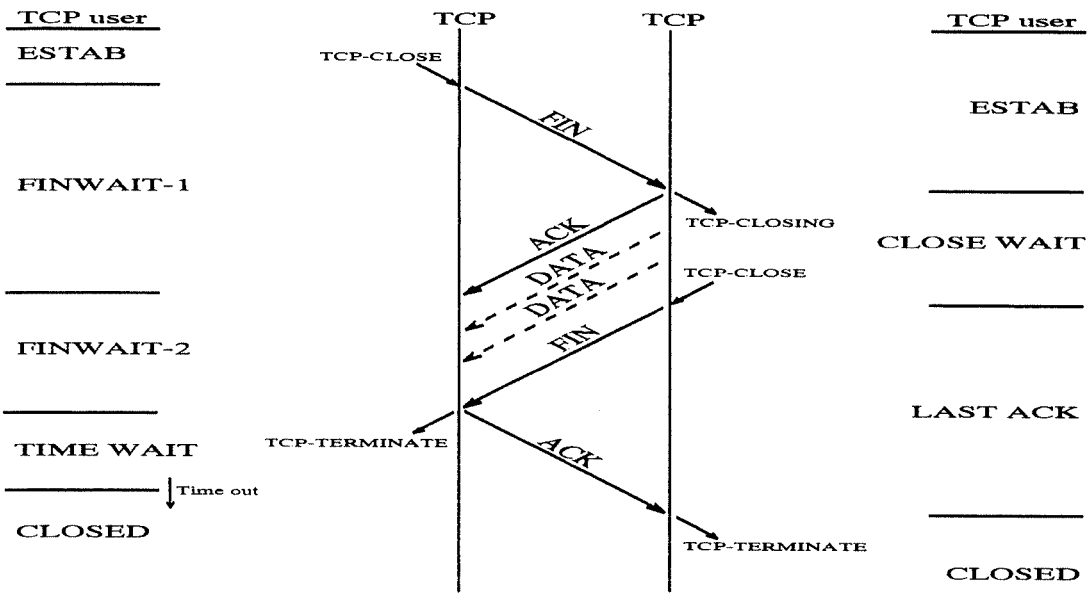


Fig. 2.15: Connection termination

Upon receiving the FIN segment, the other TCP acknowledges it with an ACK segment, notifies the reception to its user with a TCP-CLOSING.indication and enters the CLOSE WAIT state. (FINWAIT-1, CLOSE WAIT)

When the closing TCP receives the acknowledgement, it can enter the FINWAIT-2 state. At this point, only the TCP that does not initiate the close is allowed to send new segments. (FINWAIT-2, CLOSE WAIT)

When the TCP user that is in the CLOSE WAIT state has no more data to send, it transmits a TCP-CLOSE.request to its TCP entity. A FIN is also generated in this direction and it enters the LAST ACK state. (FINWAIT-2, LAST ACK)

Upon receiving the FIN, the first TCP acknowledges it by an ACK segment. It notifies its user that the connection is closing with a TCP-TERMINATE.indication primitive and can go in the TIME WAIT state. (TIME WAIT, LAST ACK)

When it receives the acknowledgement, the TCP that is in the LAST ACK state can directly return to the CLOSED state, after notifying its user by a TCP-TERMINATE.indication, while the other one must wait until a timer is run out. (CLOSED, CLOSED)

The second case is when **the two users initiate a CLOSE at the same time.** An illustration is given in figure 2.16.

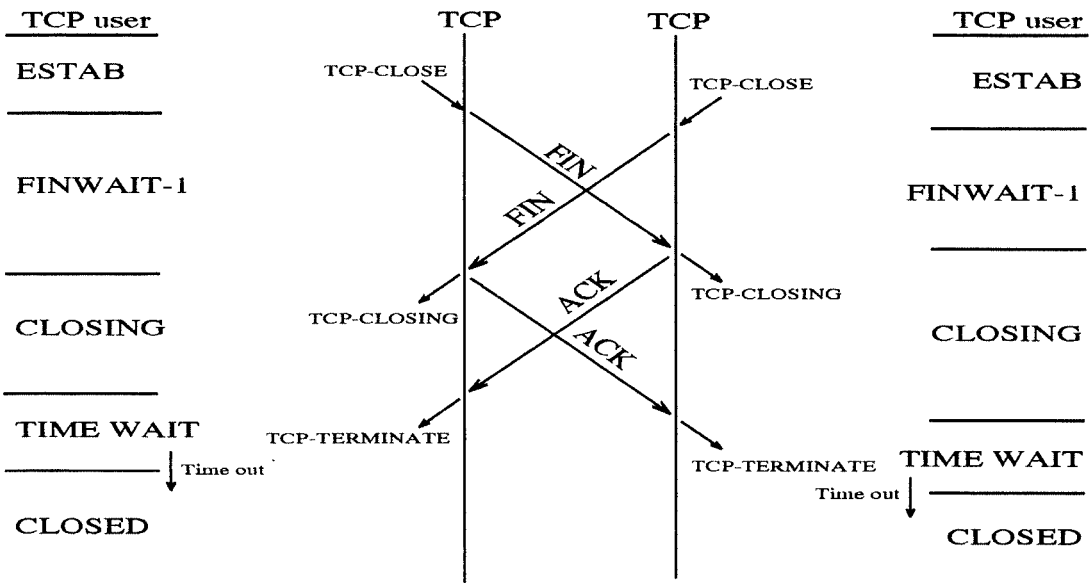


Fig. 2.16: Simultaneous connection termination

To begin, a connection is established between the two users. (ESTAB, ESTAB)

A nearly simultaneous TCP-CLOSE.request by users at both ends causes FIN segments to be exchanged and the TCPs to enter the FINWAIT-1 state. (FINWAIT-1, FINWAIT-1)

When all the segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received, notify its user that it received a FIN segment with a TCP-CLOSING.indication primitive and go in the CLOSING state. (CLOSING, CLOSING)

Both will, upon receiving these ACKs, enter the TIME WAIT state (TIME WAIT, TIME WAIT), say to their user that the connection is closed (TCP-TERMINATE.indication) and when a timer is run out delete the connection. (CLOSED, CLOSED)

Instead of requesting a graceful release, an application entity may determine that it wishes to immediately abort the connection. In this case, the local TCP entity generates a Reset segment (RST) and the connection is immediately released. Any data in transit is lost.

CHAPTER 3 : SIMPLE NETWORK MANAGEMENT PROTOCOL

The management of TCP/IP networks can be divided into 3 components: managed nodes, a management station and a network management protocol to exchange management information between managed nodes and management stations.

In TCP/IP, the management is ruled by 4 standards. Two of them define the information about network devices which are managed . These are the Structure of Management Information (SMI) [RFC1155] which defines rules to build an MIB and the Management Information Base (MIB) [RFC1156] which defines the information to manage. The two other ones define protocols that can be used to manage this information. The first one, the Simple Network Management Protocol (SNMP) [RFC1157] will be explained in this chapter. SNMP has been chosen as the short-term TCP/IP management protocol. And, now, it becomes to be widespread. CMOT (CMIP Over TCP/IP) [RFC1095], the second protocol, has been chosen as the long-term protocol. It is derived from the ISO network management protocol, the Common Management Information Protocol (CMIP). A migration from SNMP to CMOT is foreseen in the future, although not certain. This migration would make easier, from a management point of view, the migration from TCP/IP to OSI, as the management protocols would be nearly the same. These two protocols (CMIP and CMOT) will be briefly described in the next chapter.

The TCP/IP management protocols are running at the application level, above UDP or TCP. An example of the management architecture is illustrated in figure 3.1.

In TCP/IP, the management is exercised in the following way. Each managed device runs a server program, called *agent*, and maintains information about its state.

The management station (M.S. in figure 3.1) runs application programs which use the management protocol to contact the agents on managed devices in order

to perform management actions. These actions can be the retrieval of information, the change of parameters, etc.

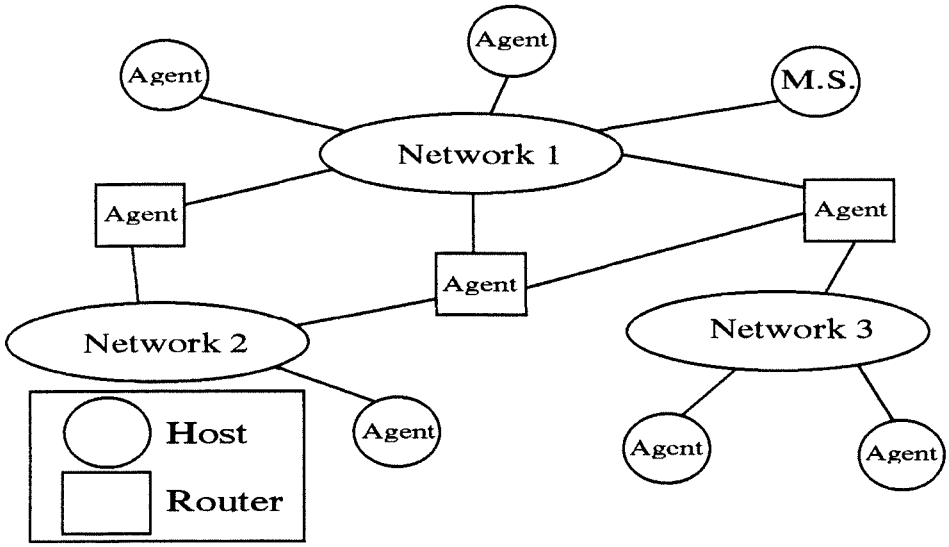


Fig. 3.1: Example of management architecture

The major concern that prevailed during the specification of SNMP and the MIB is the ability to realise, as fast as possible, working implementations of the concepts. For this, simplicity was the keyword. The agents had to be as small as possible in order to have rapid implementations. Therefore, the amount of information contained in the MIB is restricted and the design of SNMP is very simple.

As already said before, in TCP/IP, to manage devices, the management station retrieves or sets the value of variables in the agents. These variables are chosen in order that the management station is able to infer the state of the devices. The rules for constructing the variables and for defining the names to give them are explained in the *first* section of the chapter. The *second* section will deal with the management protocol, SNMP. The *last* section will review all the variables that are part of the MIB. The other management protocol, CMOT, will be presented in the next chapter.

3.1. Structure of Management Information

At the application level, not only the way by which information is exchanged is of importance, but also is the information. In the network management context, this information concerns the state of network devices. As the devices are heterogeneous in nature (different utilities, different makes, etc), they do not use the same internal representation of information. That is why a formalism must be used to work with a common way for representing the information. And this is made by a definition language, the Abstract Syntax Notation 1 (ASN.1) [ISO8824] which is an ISO standard. This will be the subject of the *first* point of this section.

This information must be expressed in the same formalism, but it must also be defined by the same rules. And this is the role of the Structure of Management Information (SMI) that will be analysed in the *second* point.

3.1.1. ASN.1

The problem to solve was to allow a set of heterogeneous machines to talk to each other, in order to exchange information. And these heterogeneous machines use different formats to represent their data. A solution to solve this problem is to use a common external data format. The format chosen for representing TCP/IP network management information is the Abstract Syntax Notation 1 (ASN.1) developed by ISO and the CCITT.

The ASN.1 specification is described in International Standard 8824 [ISO8824]. The rules for encoding ASN.1 data structures to a bit stream for transmission are given in International Standard 8825 [ISO8825]. In TCP/IP network management, ASN.1 is used for defining the formats of the PDUs and also for defining the managed information.

"ASN.1 helps keep the standards documents unambiguous, helps ease the implementation of network management protocols and guarantees interoperability." [COME91]

The basic concept of ASN.1 is the *module*. A module packages together all data structure types relating to a common theme. A module is expressed by:

```
module DEFINITIONS ::= BEGIN
...
...
END
```

ASN.1 defines 3 different objects (uppercase and lowercase letters are of importance):

- *types*: which are used to define new data structures (e.g. Gauge),
- *values*: which are instances of a type (e.g. internet) and
- *macros*: which are used to change the grammar of the language (e.g. OBJECT-TYPE).

A type is defined by:

```
NameOfType ::=
    TYPE
```

While a value is defined by:

```
nameOfValue NameOfType ::=
    VALUE
```

The types can be separated into 4 categories: primitive types, constructor types, tagged types and subtypes.

A. Primitive Types

ASN.1 defines 7 primitive types: INTEGER, BOOLEAN, BIT STRING, OCTET STRING, ANY, NULL and OBJECT IDENTIFIER from which only 4 are allowed to be used for network management: INTEGER, OCTET STRING, NULL and OBJECT IDENTIFIER.

Integer

An Integer is a cardinal number used for counting. In addition, it is possible to associate names with values that might be taken by instances of the data type. For example:

```
Day ::=
  INTEGER {sunday (1), monday (2), tuesday (3), ...)}

today Day ::= monday          (or today Day ::= 1)
```

Octet String

An octet string is a list of octets that can take a value between 0 and 255. An octet string can be used to represent characters or byte oriented data.

Null

The NULL type has only one value, also called NULL. When a field is assigned the value NULL, it means that this field has no type at all. The NULL type is currently not used in network management.

Object Identifier[1]

Object identifiers are used to name objects whatever the semantics associated with them may be. An object identifier is a specially encoded sequence of integers which uniquely identifies a node in a unique naming tree defined by ISO.

An object identifier is represented by a sequence of non-negative integers (and/or a brief textual description). The integers are seen as forming the nodes or the leaves of a tree. The administrative control of the meanings assigned to the nodes may be delegated as one traverses the tree.

The format of an object identifier can be one of the following ones:

- integers separated by dots. For instance, 1.0.8571.5.1 represents an identifier formed by the root then the node with label 1, then the node with label 0, then the node with label 8571, etc.
- small text. For example iso.standard.ftam
- a combination of the two. For example iso (1) standard (0).8751.5.1

An example of the use of object identifiers follows:

```
MgntInfo ::= OBJECT IDENTIFIER

mib MgntInfo ::=
        {1 3 6 1 2 1}
     or {iso org dod 1 2 1}
     or {iso(1) org(3) 6 1 2 1}
```

1. see the structure in pt 3.1.2 (SMI)

B. Constructors

ASN.1 defines 5 constructors, SEQUENCE, SEQUENCE OF, SET, SET OF and CHOICE from which only 2 are allowed to be used in network management SEQUENCE and SEQUENCE OF.

Sequence

A sequence is an ordered list of 0 or more elements of various types. It can be seen as a record in Pascal. For example:

```
person ::= SEQUENCE {
    name OCTET STRING,
    age  INTEGER
    }
```

Sequence of

A sequence of is a list of 0 or more elements of the same single type. It corresponds to an array in some programming languages. For example,

```
groupOfPersons ::= SEQUENCE OF person
```

C. Tagged Types

A tag is used to identify a data type of a field within a particular environment. Four types of tags are defined in ASN.1. They differ by their scope. The classes are Universal, Application-wide, Context-specific and Private-use.

A tag is defined by the following syntax: class + non-negative integer. For example, [APPLICATION 4] represents the fifth data type (the numbers begin at 0) which is meaningful within a particular application. Universal data types are meaningful throughout all ASN.1 definitions.

D. Subtypes

In ASN.1, subtyping is an important means for defining new types. Methods for defining subtypes are numerous. For TCP/IP network management, only two of these methods are allowed.

The first method is to fix the length of, for example, a string.

```
IpAddress ::=
    [APPLICATION 1]
        OCTET STRING (SIZE (4))
```

This example fixes the length of the octet string to 4. This permits to represent address like 128.141.200.10. This example also shows that this is the second tag which is defined in the application.

The second method for defining subtypes is to fix the values that can be taken by the variables. For example,

```
Counter ::=
    [APPLICATION 2]
        INTEGER (0..4294967295)
```

3.1.2. SMI

The Structure of Management Information defines the general framework within which an MIB can be defined.[STAL89] The MIB can be seen as a database, as a collection of managed objects, which specifies network management variables and their meanings, while the SMI defines the schema for the database. If an MIB is defined according to the SMI guidelines, it can be used with either SNMP or CMOT. The complete definition of SMI is given in Appendix A.

The SMI defines 3 important concepts. The *first* one is the specification of the rules for naming the variables, the objects. The *second* one is the definition of the data types that can be used in the MIB. It gives the rules for defining variable types and makes restrictions on the types of variables allowed in the MIB. The *third* one gives the format to use for defining objects.

A. Names

Names are used to identify managed objects. The objects, in the management context, are hierarchical. The OBJECT IDENTIFIER type has been chosen to represent these objects. The semantic of the OBJECT IDENTIFIER has been explained in the preceding point.

The root of the tree is not labelled but it has 3 subordinates:

- ccitt (0) which is administered by the CCITT.
- iso (1) which is managed by the ISO.
- joint-iso-ccitt (2) which is administered by both.

For network management, only the iso subtree is of interest. This node has four subordinates:

- standard (0) which contains all international standards. For example, 1.0.8571 is the identifier of the FTAM standard.

- registration-authority (1) which is reserved for use by OSI registration authorities.

- member-body (2) which has a subordinate assigned to each member body of ISO. Each node receives a label corresponding to the Decimal Country Code (DCC) of the member body.

- identified-organization (3) which has a subordinate assigned to any organization that ISO wishes to favour.

One of the children node of the org(3) node has been assigned to the Department of Defense (DoD) which, in turn, has decided to allocate a node to the Internet community. This node will be administered by the Internet Activities Board. This is formalised by the following definition.

```
internet OBJECT IDENTIFIER ::=
        { iso org(3) dod(6) 1 }
```

The IAB has decided to give 4 children nodes to the internet node. They choose directory, mgmt, experimental and private.

```
directory    OBJECT IDENTIFIER ::= { internet 1 }
mgmt         OBJECT IDENTIFIER ::= { internet 2 }
experimental OBJECT IDENTIFIER ::= { internet 3 }
private      OBJECT IDENTIFIER ::= { internet 4 }
```

The directory(1) subtree is reserved for using the OSI directory in the internet.

The mgmt(2) subtree is used to identify objects which are defined in IAB documents. The administration of this subtree is given to the Internet Assigned Numbers authority. For the moment, only one subtree of mgmt(2) is defined, mib. This contains objects described in the MIB specification.

```
mib   OBJECT IDENTIFIER  ::= { mgmt 1 }   or   1.3.6.1.2.1
```

The experimental(3) subtree identifies objects used in internet experiments. It is administered by the same authority as the mgmt subtree. It is used to test new experimental mib objects.

The last subtree, private(4), is administered by the Assigned Numbers authority. Now, only one subtree is defined, enterprises(1). It allows to register vendor-specific objects. It is defined by:

```
enterprises   OBJECT IDENTIFIER ::= { private 1 }
```

To summarise the structure, the tree is described in figure 3.2.

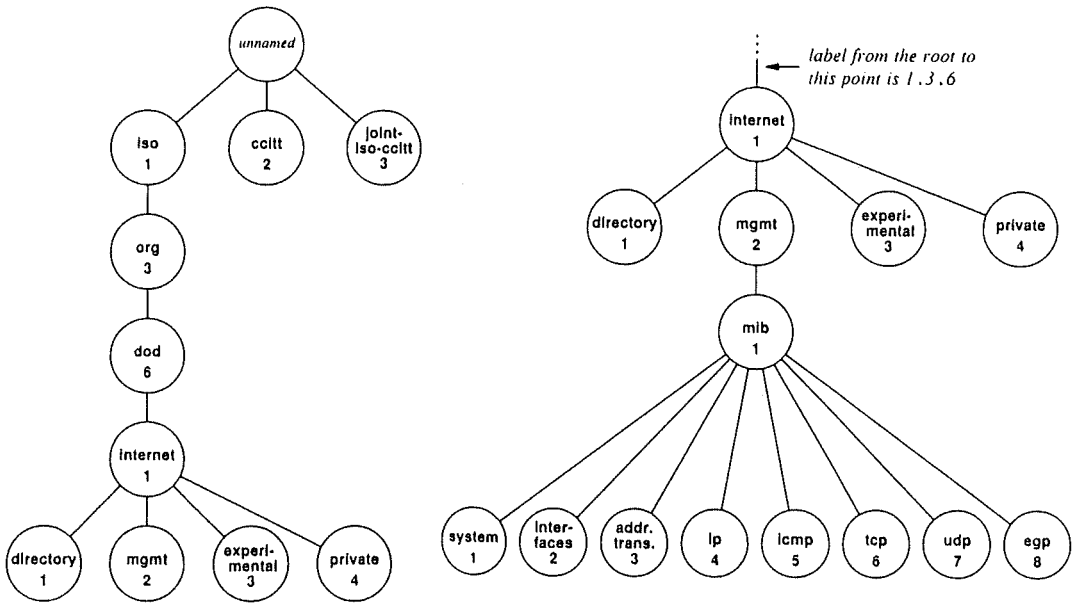


Fig. 3.2: Naming tree

B. Data Types

This point will review all the data types that can be used in an MIB definition. As already seen in the presentation of ASN.1, not all the ASN.1 types are available for constructing new object types or for defining MIB variables. The first point will describe the primitive types. The second one will review the two allowed constructor types. And the third point will show new special data types defined by the SMI.

Primitive Types

Only the INTEGER, OCTET STRING, OBJECT IDENTIFIER and NULL primitive types are allowed in the management framework.

Constructor Types

The SMI allows to use two kinds of constructor types. This gives the possibility to build lists or tables.

For lists, the syntax is:

```
<list> ::=
    SEQUENCE {
        <type1>,
        ...
        <typeN>
    }
```

In this definition, each <type> corresponds to a primitive type or a new defined type.

For tables, the syntax is:

```
<table> ::=
    SEQUENCE OF <list>
```

These definitions lead to the observation that all tables defined for network management are two-dimensional.

Defined Types

New application-wide types are defined in the SMI. The new defined types are six in number.

NetworkAddress

The NetworkAddress is a data type which represents an address from one of several protocol families. Currently, only one CHOICE is present.

```
NetworkAddress ::=
    CHOICE {
        internet
            IpAddress
    }
```

IpAddress

The IpAddress type represents a 32-bit internet address. This is an OCTET STRING, in network byte-order.

```
IpAddress ::=
    [APPLICATION 0]
        IMPLICIT OCTET STRING (SIZE (4))
```

Counter

The Counter type represents a non-negative integer, which monotonously increases until it reaches a maximum value. Then it rolls over to zero and it starts increasing again from zero. The maximum value is 2^{32}-1.

```
Counter ::=
    [APPLICATION 1]
        IMPLICIT INTEGER (0..4294967295)
```

Gauge

A Gauge represents a non-negative integer, which may increase or decrease but which stops at a maximum value. Gauge are used to measure levels, such as the current number of packets stored in a queue.

```
Gauge ::=
    [APPLICATION 2]
        IMPLICIT INTEGER (0..4294967295)
```

TimeTicks

A TimeTicks represents a non-negative integer which counts the time in hundredths of a second since a given epoch. This type is used for timestamps and clock values.

```
TimeTicks ::=
    [APPLICATION 3]
        IMPLICIT INTEGER (0..4294967295)
```

Opaque

"The Opaque type represents an arbitrary encoding. It is used as an escape mechanism, to bypass the limitations of the restrictive data typing used by the SMI."[ROSE91]

```
Opaque ::=
    [APPLICATION 4]
        IMPLICIT OCTET STRING
```

C. Managed Objects

The SMI also defines the format to use to define objects in the MIB. An object can be defined using four fields. The first one is an OBJECT DESCRIPTOR with its corresponding OBJECT IDENTIFIER. The second one, the SYNTAX, is the abstract syntax for the object type. The third one represents the ACCESS and the last one the STATUS of the defined type. This is formalised by the following macro:

```
OBJECT-MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)
    Access ::= "read-only"
                     | "read-write"
                     | "write-only"
                     | "not-accessible"
    Status ::= "mandatory"
                     | "optional"
                     | "obsolete"
END
```

A simple example describing the use of the macro is given below.

```
sysDescr OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-only
    STATUS mandatory
    ::= { system 1 }
```

The ***syntax*** field represents the data type which models the object. This is one of the allowed data types described in the preceding point. In the example, the type of the object is an OCTET STRING.

The ***access*** field represents the level of access to the managed object. the value of the object can be *not-accessible*, *read-only*, *read-write* or *write-only*. In the example, the access is read-only. This means that the value can be read but cannot be modified.

The ***status*** field represents the implementation requirements for the managed objects. If the status is *mandatory* then the managed nodes must implement this object. If the status is *optional*, this object may be implemented and if it is *obsolete*, the object needs no longer to be implemented. In the example, the object is mandatory. This means that it must be implemented in every system.

The ***value*** field represents the name of the object. The type of the object name is an OBJECT IDENTIFIER. In the example, the object name is { system 1 }.

3.2. SNMP

The Simple Network Management Protocol specifies the communication between a management station and an agent executing on a host or a router. It defines the form and the meaning of messages exchanged and the representation of names and values in those messages. SNMP also defines administrative relationships among routers being managed. For example, it provides a means for authenticating the managers.[COME91]

SNMP has been designed simple. This is for several reasons. The first one is to allow network management programs to be implemented quickly to meet the immediate needs of the Internet. The second one is to allow implementations in agents to be small and efficient in order that the agents spend the majority of their time on performing their primary functions. Any significant management processing needs are performed in management stations for which network management is the primary function.[STAL89]

An other design goal was to make SNMP robust under adverse network conditions (for fault management). Therefore, SNMP is datagram-oriented. This means that no connections must be established between two entities and that SNMP has the whole control on the retransmissions of datagrams. As it is datagram-oriented, the protocol messages must be sent wholly contained within a single datagram.[STAL89] SNMP also specifies that operations must be *atomic*, meaning that if a single SNMP message contains operations on more than one variable, the agent either performs all operations or none of them.[COME91]

To manage the networks, SNMP uses a method called a "trap-directed polling"[STAL89]. This means that the agents are only able to send a limited number of traps to the management station. And in case of a problem, when it is warned, the management station has the responsibility to further investigate.

For the transmission of messages on the network, SNMP uses the services of the User Datagram Protocol. The UDP ports used by SNMP are port 161 for the agents and port 162 for management stations.

In this section, several points will be studied. The *first* one reviews the different interactions that can be performed between a management station and an agent. The *second* one gives the format of the SNMP PDUs. The *third* point explains some administrative concepts. The *fourth* point shows how instances of objects are

identified while the *fifth* one explains the mechanism used to retrieve a whole table from the MIB. The *last* one gives an example of the encoding of the PDUs for transmission in the data field of a UDP datagram. The complete specification of SNMP is given in Appendix C.

3.2.1. Protocol Interaction

Unlike the other protocols of the suite described in the previous chapter, SNMP is an application layer protocol. The working of the protocol will rather be described in terms of protocol interactions that can be found during the communications than in terms of service primitives.

Four protocol interactions can be shown off. The first one is the get interaction.(figure 3.3)

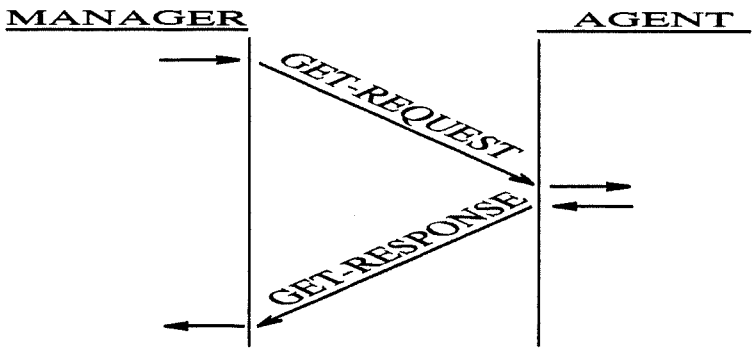


Fig. 3.3: Get interaction

The get interaction allows the manager to retrieve management information from the agent with a **get-request**. The agent will respond with a **get-response** message.

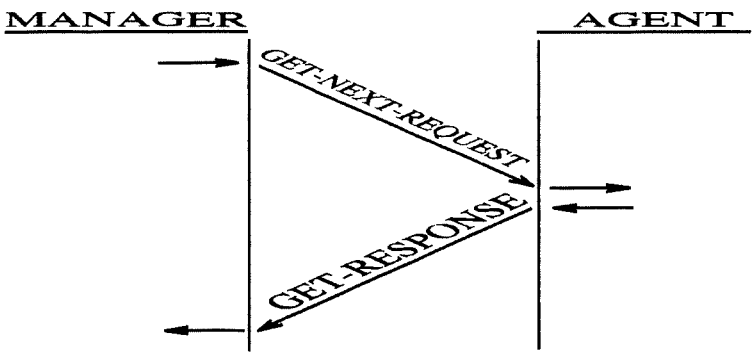


Fig. 3.4: Get-next interaction

The second means for retrieving management information is the get-next interaction.(figure 3.4)

The get-next interaction allows the manager to traverse a portion of the MIB. This is because the **get-next-request** operation asks the agent to return the value of the object following the one specified in the **get-next-request** message. The agent will also respond with a **get-response** message.

The third interaction, the set, is shown in figure 3.5.

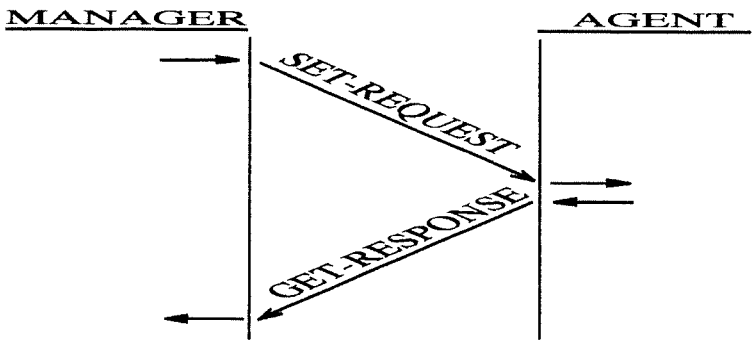


Fig. 3.5: Set interaction

The set interaction permits the manager to store information in the agent MIB. The manager requests the agent to change the value of MIB variables as specified in the **set-request** message. The agent responds with a **get-response** message which informs the management station on how the operation took place.

And the last interaction is the trap.(figure 3.6)

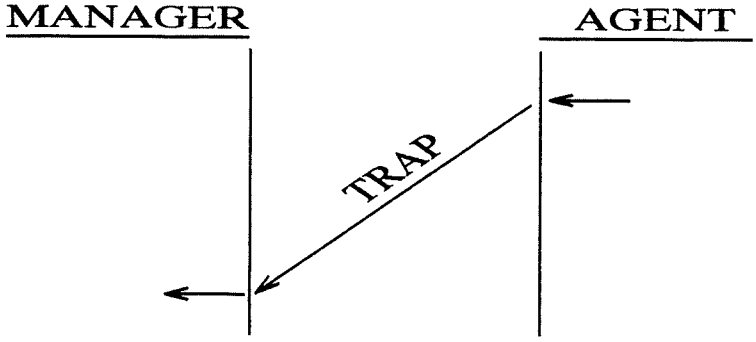


Fig. 3.6: Trap interaction

The trap interaction allows the agent to report an event to the manager. For this, it sends a **trap** message which contains the description of the event to the

management station. This interaction requires no message from the manager to the agent.

These protocol interactions permit to distinguish 5 protocol data units which are exchanged between the managers and the agents. These PDUs are get-request, get-next-request, set-request, get-response and trap.

Hereunder is described more deeply how they are working.

A. Get-Request

The **get-request** operation names a set of variables and requests that the agent generates a **get-response** containing their values. For each variable in the request, the named instance is retrieved if it matches exactly with an object instance available in the agent MIB. Otherwise, if the instance does not exist, a **get-response** is returned with error *noSuchName*.[ROSE91] & [STAL89]

B. Get-Next-Request

The **get-next-request** operation also requests that the agent generates a **get-response** containing the values of a set of objects. But in this case, for each of the object names in the variable list, it is the name and the value of the next object which is returned in the **get-response**.[STAL89] If the end of the MIB is reached, a **get-response** is returned with error *noSuchName*.[ROSE91] This mechanism is discussed hereunder in point 3.2.5.

C. Set-Request

The **set-request** operation requests that each of the objects named in the variable list be set to the values specified. All of the variables are updated simultaneously and a **get-response** identical to the request is returned. Simultaneously means that either all variables must be updated or none of them. If an instance does not exist, a **get-response** is returned with error *noSuchName*. If the instance exists but does not permit writing, a **get-response** is returned with error *readOnly*. If the instance exists and permits writing, but the value supplied has a wrong syntax or a range error, a **get-response** is returned with error *badValue*.[STAL89] & [ROSE91]

D. Get-Response

The **get-response** is sent back by an agent to return the result of an operation requested by a management station with a **get-request**, a **get-next-request** or a **set-request** PDU. When receiving a **get-response**, the manager checks its list of previously sent requests to locate the one which matches this response. If no record is found, the response is discarded. Otherwise, the manager handles the response in an appropriate fashion.[ROSE91]

E. Trap

The **trap** is generated by an agent for sending a notification to a management station of the occurrence of some significant event. The message identifies which agent generated the trap and when, and what type of event occurred. A variable list is also present. In this case, the objects and their values are used to supply additional information about the event.[STAL89]

3.2.2. SNMP PDUs

SNMP messages do not have fixed fields. For this, they are expressed using the ASN.1 notation. An SNMP message consists of three parts: a *version*, a *community name* and a *data* field. This is shown below:

```
Message ::=
        SEQUENCE {
            version
                INTEGER {
                    version-1(0)
                },
            community
                OCTET STRING,
            data
                ANY
        }
```

→ *version:* The protocol version number.

→ *community:* The community name of the requester.

→ *data:* The data field contains the protocol data units (PDUs). The protocol data units are two in number. As can be seen in the following definitions, the get-request, the get-next-request, the get-response and the set-request PDUs have the same format, whilst the trap PDU has a specific format.

```
PDUs ::=
        CHOICE {
                get-request
                    GetRequest-PDU,
```

```
                get-next-request
                    GetNextRequest-PDU,
                get-response
                    GetResponse-PDU,
                set-request
                    SetRequest-PDU,
                trap
                    Trap-PDU
              }

GetRequest-PDU ::=    [0]
        IMPLICIT PDU
GetNextRequest-PDU ::=    [1]
        IMPLICIT PDU
GetResponse-PDU ::=    [2]
        IMPLICIT PDU
SetRequest-PDU ::=    [3]
        IMPLICIT PDU
```

The first one is the "normal" PDU. It is the same for the Get, Get-Next, Response and Set messages. It contains four fields: request-id, error-status, error-index and variable-bindings.

```
PDU ::=
        SEQUENCE {
           request-id
                INTEGER,

            error-status
                INTEGER {
                    noError(0),
                    tooBig(1),
                    noSuchName(2),
                    badValue(3),
                    readOnly(4),
                    genErr(5)
                },

            error-index
               INTEGER,

            variable-bindings
                VarBindList
        }

VarBind ::= SEQUENCE {
              name
                  ObjectName,
              value
                  ObjectSyntax
              }

VarBindList ::=
         SEQUENCE OF VarBind
```

→ *request-id:* The request-id field is an integer which is used to distinguish the requests in progress or for a response which is the same as the one in the corresponding request.

→ *error-status:* The error-status field indicates if an error occurred when processing the request. An error occurs when (1) the result of an operation cannot fit into a single SNMP message, (2) an unknown variable has been requested, (3) an incorrect syntax or value was given for modifying a variable, (4) a read-only variable was tried to be modified and the value (5) is given for unforeseen errors.

→ *error-index:* The error-index indicates which variable in the request was in error. It is only used with noSuchName, badValue and ReadOnly errors. (pointer to the variable-bindings list)

→ *variable-bindings*: The variable-bindings field contains a list of variables that are concerned with the current operation. The variables contain a name and a value. The value part is not significant for request PDUs.

The second PDU is only used for the Trap messages. Its structure is the following one:

```
Trap-PDU ::=
    [4]
        IMPLICIT SEQUENCE {
            enterprise
                OBJECT IDENTIFIER,
            agent-addr
                NetworkAddress,
            generic-trap
                INTEGER {
                    coldStart(0),
                    warmStart(1),
                    linkDown(2),
                    linkUp(3),
                    authenticationFailure(4),
                    egpNeighborLoss(5),
                    enterpriseSpecific(6)
                },
            specific-trap
                INTEGER,
            time-stamp
                TimeTicks,
             variable-bindings
                VarBindList
        }
```

→ *enterprise:* The agent's sysObjectID (MIB).

→ *agent-addr:* The agent's network address.

→ *generic-trap:* A generic trap is generated if (0) the agent is reinitializing itself and objects may be altered, (1) the agent is reinitializing itself and the objects will not be altered, (2) an interface went to the down state, (3) an interface went to the up state, (4) an SNMP entity claimed to be in a community but was not, (5) a neighbour went to the down state and (6) another event occurs.

→ *specific-trap:* The specific-trap field identifies an enterprise specific trap in case the generic-trap field takes the value (6) enterpriseSpecific.

→ *time-stamp:* The value of the agent sysUpTime (see MIB).

→ *variable-bindings:* A list of variables containing information about the trap.

3.2.3. Administrative Concepts

A *community* is a set of SNMP entities. An agent belongs to a set of communities. A community is represented by a string of octet, a *community name*. An SNMP message always contains a community name, sent in the clear. The community name is used for three purposes: authentication, access control and proxy identification.

A. Authentication

Now, only a trivial scheme for authentication is used. "If the community name corresponds to a community known to the receiving SNMP entity, the sending SNMP entity is considered to be authenticated as a member of that community."[ROSE91] Any SNMP message with a valid community name is authentic.

B. Access Control

"The purpose of access control is to provide different management capabilities to different management stations."[STAL89] Once the sending SNMP entity is authenticated, the managed node must determine what level of access is allowed. The community defines the subset of the MIB (a *view*) to which requests can have access and the access mode allowed (read-only, read-write, write-only and not-accessible).

C. Proxy

As network managers want to manage all the devices in their networks (e.g. bridges, modems and not only hosts and routers), SNMP uses the proxy management to satisfy this need. A special agent (*proxy agent*) is running on another machine implementing the protocol suite. When such a device is to be managed, the management station contacts the proxy agent. This agent contacts the device to perform the desired operation using the device's protocol. Then, the results are returned to the proxy agent which sends them to the management station (using

SNMP). The proxy agent acts as an intermediate between the management station and the device.

"A proxy agent has a view of managed objects corresponding to its foreign devices. Since all of the objects contained by an agent need not be visible to a community, a proxy community has a view containing exactly those objects corresponding to a particular foreign device."[ROSE91]

3.2.4. Instance Identification and Lexicographic Order

Object identifiers are used to identify the types of the MIB objects. SNMP uses another scheme for identifying the instances of the objects.

If the object is a column of a table, to identify instances of the columns or rows, SNMP uses a value composed by the object identifier plus a suffix which is the set of columns necessary to make the suffix unique. If more than one column is necessary, then the suffix is constructed by concatenating the columns.

"For example, instances of the columns of the *ifTable*[2] are identified by using the value of the *ifIndex* column. So, the instance of *ifDescr* associated with the first interface is: [ROSE91]

```
ifDescr.1 or 1.3.6.1.2.1.2.2.1.2.1
```

If the object is not a column, the suffix is 0. For example, the identifier of an instance of sysDescr is:

```
sysDescr.0  or  1.3.6.1.2.1.1.1.0
```

With such instance identifiers, a lexicographic ordering is created over all objects instances. A lexicographic ordering means that "for instances names a and b, one of the three conditions consistently holds: either a<b, a=b or a>b".[ROSE91]

By having a lexicographic ordering of all object instances in the agent's MIB, the management station can supply an object identifier and ask for the object instance which occurs next in the ordering.[STAL89] This will allow, as explained in the next point, to easily retrieve all the variables composing a table in the MIB.

---

[2]. see the description of the MIB in point 3.3.

3.2.5. Searching Tables with the Get-Next-Request[3]

It was already seen that it is possible to identify individual elements of a table by appending a suffix to the object identifier. But a management station may wish to examine entries in a table for which it does not know all valid suffixes. The **get-next-request** operation allows this. It also allows to iterate through a table without knowing how many items the table contains.

When a management station sends a **get-next-request**, it supplies a prefix of a valid object identifier. The agent responds by sending a **get-response** command for the variable that has an object identifier lexicographically greater than the one in the request. The call **get-next-request (sysDescr.0)** returns the name and the value of the next instance in the tree which is *sysObjectID.0*. But the operand need not identify an instance, it can be any object identifier. For example, the call **get-next-request (sysDescr)** returns the name and the value of the next instance in the tree which is *sysDescr.0*. Thus, the **get-next-request** operator can be used to see if an object is supported by an agent. The only thing to do is to specify the name of the object rather than the desired instance of that object.

The MIB uses suffixes to index tables. Thus, a management station can send the prefix of a table object identifier and receive the first element of the table. Then, it can send the name of the first element of the table and receive the second and so on.[COME91] Because of the names used in the MIB, when traversing a table, each instance of the first column is retrieved, then each instance of the second column is retrieved, and so on, until the end of the table is reached.

Tables can be quickly retrieved by using the fact that the **get-next-request** operator can be given multiple operands. "For example, the call **get-next-request (ipRouteDest, ipRouteIfIndex, ipRouteNextHop)** returns the name and value of these three columns in the first row of the IP routing table. To find the next row in the table, the returned names are used as operands to another call to the **get-next-request** operator. This process may be continued until the entire table is traversed.

The end of the table is detected when the next object instance returned has a different prefix than the one given in the **get-next-request**. An error is returned only

---

[3]. Readers should have a look at the MIB structure before reading the two following points.

if an operand given to the get-next operator is lexicographically greater than or equal to the instance identifier with the lexicographically largest value. For example:

```
get-next-request (ipRouteDest)              -> ipRouteDest.0.0.0.0
get-next-request (ipRouteDest.0.0.0.0)      -> ipRouteDest.192.33.4.0
get-next-request (ipRouteDest.192.33.4.0)   -> ipRouteIfIndex.0.0.0.0
```

The third call to the get-next operator returned an instance with a different prefix than the supplied operator. Thus, the manager knows it has reached the end of that column in the table.[ROSE91]

3.2.6. Example of Encoding

Beside ASN.1 specification, there is another standard, the Basic Encoding Rules (BER) [ISO8825] which is a transfer syntax to serialize instances of ASN.1 data types into strings of octets.

The ASN.1 types are encoded with 3 fields:

- tag: field which indicates the ASN.1 type,
- length: field which indicates the size of the ASN.1 encoded value and
- value: field which contains the encoded value.

All the encoding rules will not be described in this point. Only an example of encoding will be given. For further analyses, see [ROSE91] or [ISO8825].

A **get-response** PDU returning the value of the sysDecr variable will have the following ASN.1 representation:

```
snmpMessage Message ::=
    {
     version version-1,
     community "public"
     data {
           get-response {
              request-id 17,
              error-status noError,
              error-index 0,
              variable-bindings {
                 {
                  name 1.3.6.1.2.1.1.1.0,
                  value {
                         simple { string "unix" }
                        }
                 }
                }
```

```
            }
        }
    }
```

And the BER encoding for this message would be:

30	2A	02	01	00
SEQUENCE	len=42	INTEGER	len=1	vers=0

04	06	70	75	62	6C	69	63
STRING	len=6	"p"	"u"	"b"	"l"	"i"	"c"

A2	1D	02	01	11
get-response	len=29	INTEGER	len=1	req. id=17

02	01	00	02	01	00
INTEGER	len=1	status	INTEGER	len=1	error index

30	12	30	10	06	08
SEQUENCE	len=18	SEQUENCE	len=16	objectid	len=8

2B	06	01	02	01	01	01	00
1.3	. 6	. 1	. 2	. 1	. 1	. 1	. 0

04	04	75	6E	69	78
object value	len=4	"u"	"n"	"i"	"x"

The message is, thus, a sequence of 42 octets in length and it contains 3 fields. The first one is an integer of 1 octet in length which contains the version number (0). The second one is a string of length 6 which represents the community name (public). And the third field is the encoding of the get-response which is 29 octets in length and contains 4 fields. The first three fields are all 1-octet integers which represent the request identification (17), the status field (0) and error index (0). The fourth field is the variable bindings-list. Therefore, it is represented by a sequence. This sequence contains only one item, the sequence representing the instance identification (1.3.6.1.2.1.1.1.0) and its value (unix).

3.3. MIB

The MIB describes the objects which are expected to be implemented by managed nodes running the Internet protocol suite. It describes the variables needed for monitoring and controlling the various components of the Internet.

The first version of the MIB, MIB-I [RFC1156], contains objects which are considered to be essential for management. It was designed to include the minimal number of managed objects thought to be useful for internet management. Thus, an object is included in the MIB if it is considered to be essential to management. The criteria that an object has to meet to be considered as essential are listed in the MIB-I specification. In theory, all the objects defined in the MIB are mandatory. But, in some cases, it is allowed to "forget" some of them. For example, if a router does not implement the TCP protocol, the TCP-related MIB variables may not be included in the MIB.

The first MIB specification was considered as the first step. But now, another MIB, MIB-II, has been specified [RFC1158]. It is currently a proposed internet standard. The emphasis of MIB-II is to create new objects whilst maintaining compatibility with the SMI and MIB-I. The entire MIB-II specification is given in Appendix B.

The MIB has been divided into 10 groups: System, Interfaces, Address Translation, IP, ICMP, TCP, UDP, EGP, Transmission and SNMP. These groups have been defined to provide a means of assigning object identifiers and to provide a method for implementations of agents to know which objects they must implement. In the MIB, the format of the description is given using the OBJECT-TYPE definition. In several groups, variables are listed in the same line. This is only for compactness reasons.

3.3.1. System Group

The system group is mandatory for all managed nodes. It contains general configuration information, in particular, information about the system's manufacturer, software revision and how long the system has been up.

```
system OBJECT IDENTIFIER ::= { mib 2 }
```

→ *sysDescr*: Textual description of the entity.

→ *sysObjectID*: Identification of the agent. The value is allocated within the SMI enterprises subtree.

→ *sysUpTime*: The time since the agent was started.

In MIB-II were added the following objects:

→ *sysContact*: The name of person to contact concerning this device.

→ *sysName*: The device name.

→ *sysLocation*: The device physical location.

→ *sysServices*: The services offered by the device.

3.3.2. Interfaces Group

The Interfaces group is mandatory for all managed nodes. It contains information on the entities at the interface layer. It presents information about each network interface in the system.

```
interfaces OBJECT IDENTIFIER ::= { mib 2 }
```

The interface group contains two objects:

→ *ifNumber*: The number of network interfaces.

→ *ifTable*: A table which contains information about the interfaces. There is one row for each interface.

Each row of the table contains several columns:

→ *ifIndex*, *ifDescr*, *ifType*: A unique value for each interface (identifier), a text describing the interface and the type of the interface (e.g. ethernet-csmacd, fddi).

→ *ifMtu*, *ifSpeed*, *ifPhysicalAddress*: The maximum transmission unit, the transmission rate in bits/second and the media-specific address of the interface.

→ *ifAdminStatus*, *ifOperStatus*, *ifLastChange*: The desired state of the interface, its current operational state and how long ago its state changed.

→ *ifInOctets*, *ifInUcastPkts*, *ifInNUcastPkts*, *ifInDiscards*, *ifInErrors*, *ifInUnknownProtos*: The total number of octets received on the interface, the number of unicast packets, of broadcast or multicast packets delivered to the upper layer and the number of inbound packets discarded due to resource limitations, due to format errors and due to unknown protocol.

→ *ifOutOctets*, *ifOutUcastPkts*, *ifOutNUcastPkts*, *ifOutDiscards*, *ifOutErrors*, *ifOutQLen*: The total number of octets transmitted on the interface, the number of unicast packets, of broadcast or multicast packets from the upper layer, the number

of outbound packets discarded due to resource limitations or due to format errors and the size of the output packet queue.

An object has been added in MIB-II:

→ *ifSpecific*: A reference to MIB definitions specific to the particular media being used to realize the interface.

3.3.3. Address Translation Group

The address translation group contains the mappings between IP addresses and subnetwork-specific addresses that all IP systems must support. It contains a table which is the union across all interfaces of the translation tables for converting an IP address into a subnetwork-specific address. It must be implemented by all systems.

```
at OBJECT IDENTIFIER ::= { mib 3 }
```

Each row of the table contains three columns:

→ *atIfIndex*: The number of the interface.

→ *atPhysAddress*: The media-dependent physical address.

→ *atNetAddress*: The IP address corresponding to the physical address.

Note that in MIB-II, the information on address resolution has been moved to the network protocol group (IP group).

3.3.4. IP Group

The IP group contains information about the IP layer. The IP group is mandatory for all systems. It contains several simple type variables and three tables.

```
ip OBJECT IDENTIFIER ::= { mib 4 }
```

The simple type variables are:

→ *ipForwarding*: It indicates whether the entity is acting as gateway or as host.

→ *ipDefaultTTL*: The default Time To Live for IP packets.

→ *ipInReceives*, *ipInHdrErrors*, *ipInAddrErrors*, *ipForwDatagrams*, *ipUnknownProtos*, *ipInDiscards*, *ipInDelivers*: The total number of input datagrams, of input datagrams discarded due to errors in IP header, due to addressing errors, the number of datagrams forwarded, the number of datagrams

sent to unknown protocols, the number of datagrams discarded due to resource limitations and the number of datagrams successfully delivered to the upper layer.

→ *ipOutRequests*, *ipOutDiscards*, *ipOutNoRoutes*: The number of datagrams received from the upper layer, discarded due to resource limitation and discarded due to no route.

→ *ipReasmTimeout*, *ipReasmReqds*, *ipReasmOKs*, *ipReasmFails*: The timeout value for reassembly, the number of received IP fragments needing reassembly, successfully reassembled and unsuccessfully reassembled.

→ *ipFragOKs*, *ipFragFails*, *ipFragCreates*: The number of IP datagrams successfully fragmented, discarded because they had to be fragmented but could not due to the DF flag and the number of IP fragments generated.

The **IP address table** contains IP addressing information concerning this entity.

→ *ipAddrTable*: The table of addressing information.

Each row of the table contains:

→ *ipAdEntAddr*, *ipAdEntIfIndex*:, *ipAdEntNetMask*, *ipAdEntBcastAddr*: The IP address of this entry, the number of the interface, the subnet mask of this IP address and the least-significant bit of the IP broadcast address.

Another column was added in MIB-II:

→ *ipAdEntReasmMaxSize*: The size of the largest IP datagram that this entity is able to reassemble.

The **IP routing table** contains an entry for each known route.

→ *ipRoutingTable*: The routing table of this entity.

Each row contains:

→ *ipRouteDest*, *ipRouteIfIndex*: The destination IP address and the number of the interface.

→ *ipRouteMetric1*, *ipRouteMetric2*, *ipRouteMetric3*, *ipRouteMetric4*: The primary and alternate routing metrics for this route.

→ *ipRouteNextHop*, *ipRouteType*, *ipRouteProto*, *ipRouteAge*: The IP address of the next hop, the type of the route (invalid, direct, remote, other), the mechanism used to determine the route (local, netmgmt, icmp, egp, ggp, ...) and the age of the route in seconds.

MIB-II added one column:

→ *ipRouteMask*: The subnet mask of the destination address.

Another table was added in MIB-II, the **IP address translation table**, which replaces the tables in the address translation group.

→ *ipNetToMediaTable*: The IP address translation table.

Each row of the table contains:

→ *ipNetToMediaIfIndex*, *ipNetToMediaPhysAddress*, *ipNetToMediaNetAddress*, *ipNetToMediaType*: The interface number, the media physical address, the IP address and how the mapping was determined (static, dynamic, invalid, other).

3.3.5. ICMP Group

The ICMP group contains information about the ICMP protocol. As ICMP is mandatory for all devices implementing IP, it must be implemented in all systems.

```
icmp OBJECT IDENTIFIER ::= { mib 5 }
```

This group contains 26 variables. As seen in the previous chapter, ICMP possesses several different messages. The ICMP group contains two variables for each ICMP message, one for counting the number of generated messages and one for the number of received messages of this type. For example:

→ *icmpInDestUnreachs*, *icmpOutDestUnreachs*: The number of "destination unreachable" messages sent and received.

The four additional variables are:

→ *icmpInMsgs*, *icmpOutMsgs*, *icmpInErrors*, *icmpOutErrors*: The number of messages received and sent and the number of messages received in errors or not sent due to errors.

3.3.6. TCP Group

The TCP group contains information about the TCP protocol. It must be implemented in all systems implementing TCP.

```
tcp OBJECT IDENTIFIER ::= { mib 6 }
```

The TCP group contains:

→ *tcpRtoAlgorithm*, *tcpRtoMin*, *tcpRtoMax*: The algorithm used to determine the retransmission timeout and the minimum and maximum values permitted for the retransmission timeout.

→ *tcpMaxConn*, *tcpActiveOpens*, *tcpPassiveOpens*, *tcpAttemptFails*, *tcpEstabResets*, *tcpCurrEstab*: The maximum number of connections allowed, the number of active and passive opens, of connection attempts which failed, of connection resets and of connections currently established.

→ *tcpInSegs*, *tcpOutSegs*, *tcpRetransSegs*: The number of segments received, sent and retransmitted.

Two variables were added in MIB-II:

→ *tcpInErrs*, *tcpOutRsts*: The number of received segments discarded due to format errors and the number of resets sent.

The **TCP connection table** contains TCP connection-specific information.

→ *tcpConnTable*: TCP connection table

Each row of the table contains:

→ *tcpConnState*, *tcpConnLocalAddress*, *tcpConnLocalPort*, *tcpConnRemAddress*, *tcpConnRemPort*: The state of the connection (listen, sys sent, estab,...) and the local and remote addresses and ports of the connected entities.

3.3.7. UDP Group

The UDP group contains information about the UDP protocol. It must be implemented in all systems implementing UDP.

```
udp OBJECT IDENTIFIER ::= { mib 7 }
```

The UDP group contains:

→ *udpInDatagrams*, *udpNoPorts*, *udpInErrors*, *udpOutDatagrams*: The number of received datagrams delivered to users, sent to unknown ports, discarded due to format errors and the number of sent datagrams.

In MIB-II, a new table, which contains information about the application entities which are using UDP, has been added:

→ *udpTable*: UDP listener table.

Each row of the table contains:

→ *udpLocalAddress, udpLocalPort*: The local IP address and port of the UDP user.

3.3.8. EGP Group

The EGP group is required only for the systems which support the Exterior Gateway Protocol.

```
egp OBJECT IDENTIFIER ::= { mib 8 }
```

It contains:

→ *egpInMsgs*, *egpInErrors*, *egpOutMsgs*, *egpOutErrors*: The number of sent and received EGP messages and the number of EGP messages received with errors and not sent due to resource limitations.

It also contains a table with information about the neighbours.

→ *egpNeighTable*: EGP neighbour table.

Each row of the table contains:

→ *egpNeighState*, *egpNeighAddr*: The state and the address of the neighbour.

MIB-II added many other entries to this tables (see Appendix B).

3.3.9. Transmission Group

The transmission group is defined in MIB-II only. It should contain media-specific MIB variables. But the variables to be included are still tested in the experimental subtree.

```
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
```

3.3.10. SNMP Group

The SNMP group is only defined in MIB-II. It contains SNMP-related information. Some of the variables will be zero-valued for implementations which act only as agent or as management stations.

```
snmp OBJECT IDENTIFIER ::= { mib-2 11 }
```

→ *snmpInPkts*: The number of SNMP PDUs received.

→ *snmpInBadVersions*, *snmpInBadCommunityNames*: The number of received PDUs with an unsupported version number and with an unknown community name.

→ *snmpInBadCommunityUses*: The number of received PDUs which contain an operation that was not allowed by the community name.

→ *snmpInASNParseErrs*, *snmpInBadTypes*: The number of received PDUs containing an ASN.1 parsing error and which had an unknown PDU type.

→ *snmpInTooBigs*, *snmpInNoSuchNames*, *snmpInBadValues*, *snmpInReadOnlys*, *snmpInGenErrs*: The number of received PDUs for which the value of the "ErrorStatus" field is tooBig, noSuchName, badValue, readOnly and genErr.

→ *snmpInTotalReqVars*, *snmpInTotalSetVars*: The number of MIB objects successfully retrieved as the result of either a Get-Request or a Get-Next PDU and the number of MIB objects successfully set as the result of a Set-Request PDU.

→ *snmpInGetRequests*, *snmpInGetNexts*, *snmpInSetRequests*, *snmpInGetResponses*, *snmpInTraps*: The number of Get-Request, Get-Next, Set-Request, Get-Response and Trap PDUs which have been processed by the SNMP entity.

→ *snmpOutPkts*: The number of SNMP PDUs sent.

→ *snmpOutTooBigs*, *snmpOutNoSuchNames*, *snmpOutBadValues*, *snmpOutReadOnlys*, *snmpOutGenErrs*: The number of generated PDUs for which the value of the "ErrorStatus" field is tooBig, noSuchName, badValue, readOnly and genErr.

→ *snmpOutGetRequests*, *snmpOutGetNexts*, *snmpOutSetRequests*, *snmpOutGetResponses*, *snmpOutTraps*: The number of Get-Request, Get-Next, Set-Request, Get-Response and Trap PDUs which have been generated by the SNMP entity.

→ *snmpEnableAuthTraps*: It indicates if the agent is configured to generate authentication-failure traps.

CHAPTER 4 : CMIP

In this chapter, the OSI structure for network management, the Common Management Information Protocol (CMIP) will be briefly described. Common means that CMIP is a general purposes network management protocol. It is not only aimed at OSI networks. Therefore, it has been applied to the management of TCP/IP networks. This has been named CMOT, the Common Management information protocol Over TCP/IP. CMOT will be examined in the second section.

4.1. CMIP

Like SNMP, CMIP is designed to transfer network management information from one place to another. The review of CMIP will be made in four points. The first one will have a look at the management information. The second one will examine the services provided to the user and the third one will analyse the PDUs used to transfer the information. The last point will give rapid explanations on the working of CMIP. Many ideas to realize this chapter have been found in [RFC1095].

4.1.1. Management Information

As in SNMP, management information is stored in a Management Information Base (MIB) defined by a Structure of Management Information [ISO10165-1]. However, the organisation of the MIB is not the same as in SNMP. In CMIP, the base concepts are the abstraction of a managed object and the various kinds of relationships that objects can be involved in. Hierarchies formed by the relations between the objects are also of importance. Three hierarchies are defined: the registration, the containment and the inheritance hierarchies.

Managed Objects and Attributes

Management information is modelled using object-oriented techniques. Everything to be managed in the network is represented by managed objects. Examples of managed objects are protocol entities, modems and connections.

An object class represents a collection of managed objects with the same or similar properties. "A managed object is defined as an instance of the object class. Each object class is defined as having (among other things) a set of attributes."[STAL89] An example of an object class is *transport connection*. There are a number of managed objects (specific transport connections) that are instances of this class.

Managed object classes are defined by:

- the *attributes* or properties the object has,
- the CMIS *operations* that can be performed on the object,
- the *actions* that can be performed on the object,
- the *events* that the object can generate and
- *information* about various relationships the object may be involved in.

"CMIP objects are represented using the ISO defined ASN.1 [ISO8824] and are encoded using the ISO Basic Encoding Rules (BER) [ISO8825]. For SNMP only a subset is used."[WARR90]

The Registration Hierarchy

The registration hierarchy is determined by the ASN.1 registration tree for assigning OBJECT IDENTIFIERs. In the context of management, these OBJECT IDENTIFIERs are used for identifying object classes and attributes. Its purpose is simply to generate universal unique identifiers.

The Containment Hierarchy

The containment hierarchy is constructed by applying the relationship "is contained in" to objects and attributes. Objects of one class may contain other objects. Objects may also contain attributes. The containment hierarchy is important because it can be used for identifying instances of a managed object.

One or more attributes *(distinguished attributes)* are chosen so that specifying their values uniquely identifies which managed object is being referenced.[STAL89] A distinguished attribute is composed of an OBJECT IDENTIFIER naming the attribute and the value of the attribute. For each object class, the distinguished attributes that differentiate instances of that class are called the *relative distinguished*

name. A sequence of relative distinguished names is the *distinguished name* of the managed object. The containment hierarchy is sometimes referred to as the *naming tree* because it is used to name a particular instance of a managed object. For example, if the highest object class represented in the hierarchy is a network and nodes are contained in a network, then if an object class *(transport entity)* that is contained in an node, contains an object class *transport connection*, an instance of a *transport connection* can be identified by the concatenation of "instance information" for each object class.

The Inheritance Hierarchy

The inheritance hierarchy is constructed by applying the relationship "inherits properties of" to object classes. An object class may inherit properties of another object class. Refinement is obtained by adding additional properties. In this relationship, the parent class is called the *superclass* and the inheriting class the *subclass*. The inheritance hierarchy has no relevance to the naming of object instances.

4.1.2. Management Services

CMIS [ISO9595] is an Application Service Element which is used by an application process to exchange information and commands. The following 10 CMIS service primitives form the basis for all OSI management activities:

→ M-EVENT-REPORT: It provides a means to report events to the management station.

→ M-CONFIRMED-EVENT-REPORT: It is the same as M-EVENT-REPORT but it requires an acknowledgment from the management station.

→ M-CONFIRMED-GET: It allows to retrieve management information from the peer MIB.

→ M-SET: It allows to modify information in the peer MIB.

→ M-CONFIRMED-SET: It is the same as M-SET but it requires an acknowledgement.

→ M-ACTION: It requests to perform some action.

→ M-CONFIRMED-ACTION: It requests to perform some action but it requires a confirmation by the managed device.

→ M-LINKED-REPLY: It is used to link different replies in response to multiple requests.

→ M-CONFIRMED-CREATE: It requests to create a managed object instance.

→ M-CONFIRMED-DELETE: It requests to delete a managed object instance.

→ M-INITIALIZE: It is used to initialize an association.

→ M-TERMINATE: It is used to terminate an association.

→ M-UABORT: It is used to abruptly terminate an association.

4.1.3. Management Protocols

"OSI application layer protocols are built using Application Service Elements (ASEs). CMIP [ISO9596] uses three ASEs: The Association Control Service Element (ACSE), the Remote Operations Service Element (ROSE) and the Common Management Information Service Element (CMISE)."[STAL89]

CMIP uses the Association Control Service Element [ISO8649] & [ISO8650] to establish and release associations between application entities. Before any management operations can be performed using CMIP, it is necessary for the two application entities involved to form an association. The five ACSE PDUs that are used by CMIP to manage its associations are: the AARQ (Association request) PDU to request the establishment of an association, the AARE (Association Response) PDU to confirm the establishment, the RLRQ (Release Request) PDU to request the graceful termination of an association, the RLRE (Release Response) PDU to confirm the graceful termination and the ABRT (Abrupt) PDU to abruptly close the association.

CMIP uses the Remote Operations Service Elements (ROSE) [ISO9072-1]&[ISO9072-2] to provide the transaction oriented services required by the systems management application entity. The ROSE are supported by four PDUs: the ROIV (Invoke) PDU to invoke to perform an operation, the RORS (Return Result) PDU to report the successful completion of an operation, the RORE (Return Error) PDU to report the unsuccessful completion of an operation and the RORJ (Reject) PDU to reject an operation request due to a problem."[KLER88]

CMISE provides the network management applications with the Common Management Information Services (CMIS). These services are listed in the point hereabove. The last three ones (M-INITIALIZE, M-TERMINATE and M-UABORT) are performed by using the ACSE while the other ones are performed by using the ROSE.

4.1.4. Working

Here are described , in bulk, some of the important characteristics of CMIP.

CMIP is an association oriented protocol. Thus, it requires a reliable transport layer such as TP-4 or TCP/IP. An association oriented protocol involves more processing for communications because an association must be established before sending data. However the sender of a CMIP message is sure the message has reached its destination which is not the case with SNMP for which the applications have to guarantee delivery by themselves. Therefore, CMIP is better for retrieving large amounts of data. But a problem with association oriented protocols is that the network is harder to manage when trouble occurs because of the connection establishment phase. For example, if the network is saturated by a device, it could be difficult to send more than one message to stop this device.

CMIP provides for the implementation of sophisticated conditional commands based on object type, value and relative location in the managed network. For example, a CMIP-based management system could directly request port information for all gateways for which ifNumber $>= 5$. SNMP would request the information from all routers and check the ifNumber value to see if the router is of interest.[WARR90]

CMIP requests can be either atomic or carried out on a best effort basis. If an error occurs during the processing of the request, if the request is atomic, no result is returned and if it is carried out on a best effort basis, all the results which cause no errors are returned. Furthermore, CMIP supports linked replies. If the response is too large to fit in a single packet, several packets can be linked together to return the response. SNMP is only able to return a reply packet per request.[WARR90]

The data query mechanism makes that CMIP is oriented more toward retrieving aggregate information than individual items of information as with SNMP.[FISH91]

SNMP actions can only be performed as side effects of variables setting, whereas CMIP provides its user with the definition and execution of object specific imperative commands.[WARR90]

Both SNMP and CMIP agents are able to send event messages to their managers. But SNMP events are always unconfirmed while CMIP events either confirmed or unconfirmed.

SNMP uses a polling-based management. This means that the manager regularly asks each device for its status. "CMIP uses an event-based management. The managed devices asynchronously send pre-configured information of interest to the manager. The device informs the manager of its status when it changes."[FISH91]

The event-based management has the advantage that if a large number of devices is to be managed, it will consume less network bandwidth than the polling based management. However some stupid devices may be unable to tell the manager that they have problems and, in this case, polling is more appropriate.

And finally, CMIP is significantly more complex to design and implement than SNMP. It also occupies more code-space (3 times more than SNMP) which is important in some devices with limited resources.

4.2. CMOT

CMOT differs from CMIP for at least two points. The first one is the information which can be managed and the second one is the structure of the protocol layers.

4.2.1. Management Information

As CMOT uses the standard Internet MIB, the Internet SMI does not use the notions of *object class* and *attribute*. Only the concepts of *object type* and *object instance* are used. In order to use CMIP to convey information defined in terms of the Internet SMI, it is necessary to show how object instances are specified and to provide the necessary structure for differentiating object class and attributes. These objectives are both met by separating the containment hierarchy used for naming objects from the registration hierarchy and by imposing an *object class* structure on the Internet SMI.[RFC1095]

The mapping between the Internet SMI and the containment hierarchy is achieved by mapping those object types defined in the Internet MIB as leaf nodes to attributes and non-leaf node object types to object classes. The mapping with the MIB is shown in figure 4.1. [STAL89] For example, the attributes of the *system* class are *sysDescr*, *sysObjectID* and *sysUpTime*.

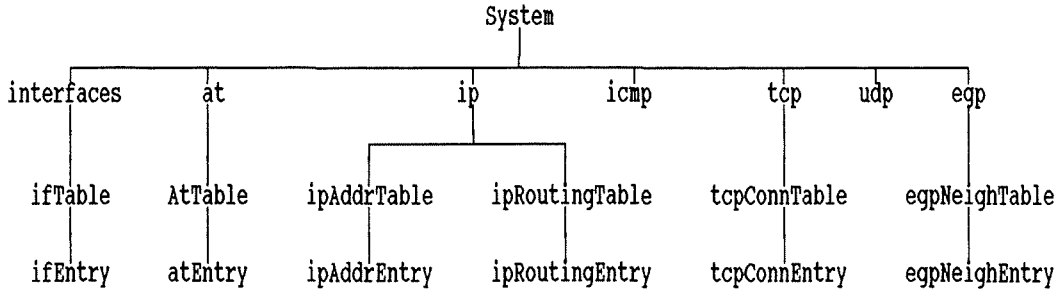


Fig. 4.1: Mapping of MIB to CMOT containment hierarchy

The OBJECT IDENTIFIER naming a distinguished attribute together with its value is called an attribute value assertion. A set of attribute value assertions is the relative distinguished name associated with that object class. The sequence of relative distinguished names for each of the object classes in the containment hierarchy to which a managed object belongs is the distinguished name of the object.

The Internet SMI does not use the inheritance hierarchy.

4.2.2. Protocol

The following figure (4.2) summarizes the CMOT protocol suite. It contains the ISO ACSE protocol, the ISO ROSE protocol, the ISO CMIP protocol, the Lightweight presentation protocol (LPP), UDP, TCP and IP.

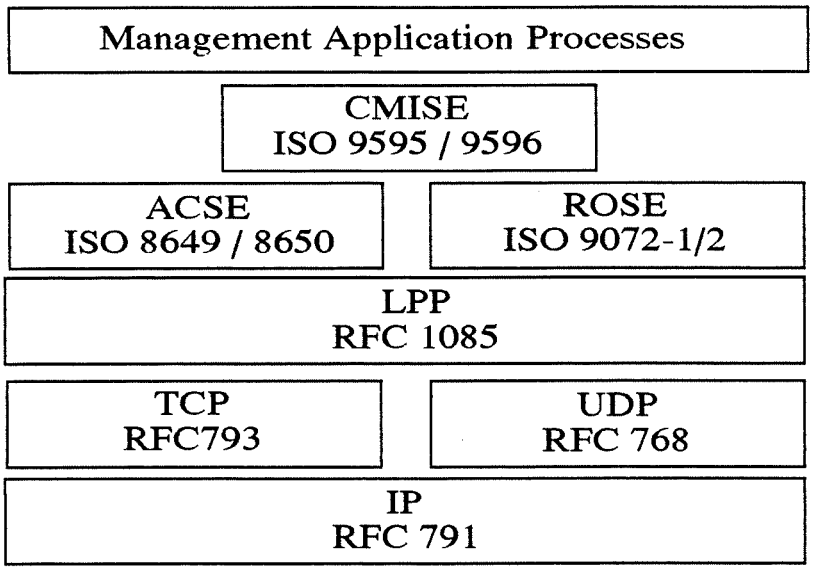


Fig. 4.2: CMOT protocol suite

The difference between OSI CMIP and CMOT is the LPP layer.

The problem was to put ISO application protocols on top of TCP/IP. But the gap between the ISO protocols (ACSE and ROSE) and the Internet protocols (UDP and TCP) must be filled. The approach is presented in [RFC1085]. Since the service elements required for network management do not require the use of full ISO presentation layer services, it is possible to define a simple presentation layer that provides only the services required. This lightweight presentation protocol allows the use of ISO presentation services over both TCP and UDP.

4.2.3. Opinion

The opinion of David Mahler (vice-president of marketing for Remedy, a company in Palo Alto, CA, developing protocol-independent network management products) about CMOT will conclude this chapter. Mahler said in [FISH91]: "My particular opinion, and I think you'll find it to be the general consensus, is that CMOT is dead. It lost its market window, and SNMP has very well filled the role of

management protocol for TCP/IP. The SNMP community delivered more functionality, faster, to the market place. (...) The CMOT community was working on the problem of managing TCP/IP devices. That's the same thing that SNMP was doing. The OSI/Network Management Forum worked on a very different problem. [Its members] didn't care what network you were trying to manage. They said the management systems had to talk to each other and were largely independent of the kind of network out there."

CHAPTER 5 : HP OPENVIEW NETWORK NODE MANAGER

Openview is termed as a node manager. Therefore management operations concentrate on the nodes and the management function on the network is less developed. For example, it is not possible to find a monitoring function that detects broadcast storms. Every action made by the program consists of operations on the individual nodes and not on the network.

Openview has been tested on a HP-9000 workstation under the Unix (HP-UX) operating system. It uses the services of X-window to make the displays on the screen. Openview is made to manage nodes implementing at least the Internet Protocol. Some functions are available for Ethernet LANs but Openview is also convenient for WANs.

In the *first* section, the main functions of HP Openview are covered. And in the *second* section, a conclusion resumes the positive and negative points of this program.

5.1. Presentation and Comments

Below are summarized the major possibilities of Openview.

5.1.1. Graphical User Interface

A. Presentation

Openview possesses a database containing a representation of the network. This database is automatically updated with information coming from the nodes (by polling the nodes, listening to broadcast traffic (ARP), etc). Openview draws a view of the network using this database. An example of a general map is displayed in figure 5.1.

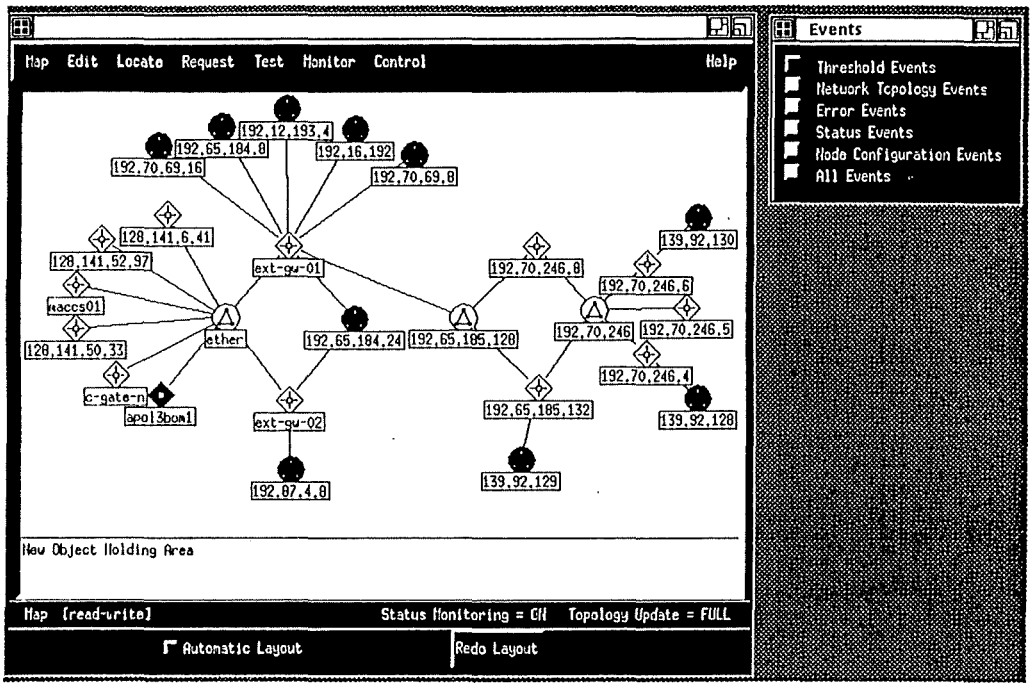


Fig. 5.1: Example of map

The map of the network is represented in three different levels.

Internet Level.

It shows a logical view of the entire network (internal and external). Here only the networks and the gateways are displayed. (As in figure 5.1)

Physical Level.

It shows a view of a network with all the segments, the bridges, etc. It is accessed by clicking on a network in the Internet level.

Segment Level.

It shows a view of a segment with all the nodes connected on this segment. It is accessed by clicking on a segment in the Physical level. An example of a three level map is given in figure 5.2.

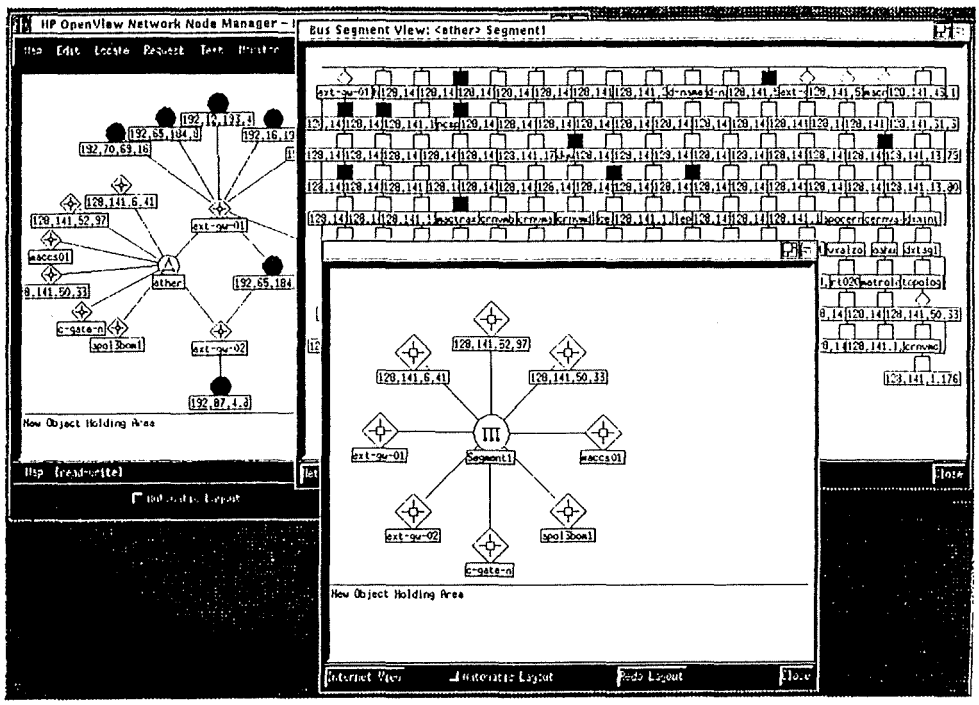


Fig. 5.2: Three level map

As Openview possesses a topology builder, it automatically draws a view of an entire internetwork with the routers, the networks and the hosts that it can detect. This view is not complete because only the nodes implementing the IP protocol can be detected with the topology builder. Therefore, if the network contains bridges, repeaters, terminal servers, they will not be automatically detected and, thus, they will not be present on the screen. The user has to add bridges, new segments, repeaters to the map built by the topology builder. By this means, the map will correspond to the real (complete) physical topology of the network. These changes are saved in the database.

To help update the network, Openview disposes of tools to edit the views. These tools allow users to add and to remove objects (computers, bridges,...) or connections, and to move an object from one segment or one network to another, or to change the type of an object. An object can be moved to another part of the view only by using the mouse.

A snapshot of the network can be taken. The snapshot is a representation of the network at a certain moment. A snapshot is a means for saving a network map. It is possible to take a snapshot, change the network topology and, if wanted, restore the old version of the network. A snapshot can be recovered to work with a previous view of the network.

Every object can be either managed or unmanaged. If it is managed, then it will be polled at fixed intervals to learn information on its status, its performances, etc. If it is unmanaged, no information about it will be available. When managed, a node sends information to the node manager.

The user can locate an object by using many parameters. An object can be found by its name, its IP address, its link address. One or more objects can be located by using the type or the comment field. Every object corresponding to the selected type or to the chosen comment is highlighted. All SNMP or non-SNMP objects can also be found. The window containing the highlighted objects is created and/or popped up if it is already present on the screen.

It is possible to find the route between two objects. The only thing to do is to give the addresses or select the objects. And the program highlights the route and gives the address of every gateway that is on the route.

By clicking with the right button on an object, a description of it is displayed. If the object is a node, the description contains the name, the type, some comments and the SNMP or non-SNMP character of the host. It also contains the IP address, the link address, the type of every interface. If the object is a network, the description contains the name of the network, the network number, the number of segments, the number of nodes and some comments about the network.

B. Comments

The presentation of the program is very good. It looks very professional. The windows are well designed, with shadows, etc. Openview is very easy to use because the drawing is made automatically. But the only way to stop the extension of the map is to unmanage the end objects. This is not very convenient. It would have been better to add a learn on/off function to begin or stop the extension of the map.

But the drawing is more difficult if made manually. The display is divided into three levels and these levels cannot be changed. It is not flexible. It is impossible to move an object from one level to another. It is, for example, impossible to make a host become a gateway because they do not belong to the same level. To make this sort of change, the user must delete the host and create a gateway with the same name and the same address. The user must be rigourous when adding new objects. It is not possible to add an object without giving its name or its address. It seems constraining but it is better than having an object with no name on the map.

The user can choose to place the objects wherever he wants. But he can use a facility to redraw the map where the placement is calculated by the program. This function works perfectly well but only if the map is not overloaded. If there are too many objects on the map, they begin to be superposed.

A bad thing to notice is that, in the internet level, all the networks are represented by the same picture. It would have been better to use different pictures for Ethernet, point to point links, etc.

The facility to search the route between two nodes works well but there are two imperfections. The first one is that in the answer table, sometimes the name of the gateway is given and sometimes the IP address is given. It is not very easy to follow the trace. The second imperfection appears when the two nodes are on the same network but on different segments. On the network level, there is no highlight between the two segments.

In conclusion, the graphical representation of the network is not perfect but is very good. The possibility to stop the extension of the map would have been very useful. All useful tools to draw the map and manage objects (add, delete,...) are present. And once the basic concepts of the different levels are understood, it is very easy to work with the graphical facilities.

5.1.2. Alarm Manager

As can be seen in the graphical user interface, an alarm manager is always present on the screen (see figure 5.1). By clicking on the different buttons, it gives the possibility to display the different events gathered. The events are divided into five categories. The first category is related to thresholds trespassing, the second one to network topology changes, the third one to errors, the fourth to status events and the last one to node configuration changes. The sixth button displays the events from the five categories. As can be seen in figure 5.3, an event is characterized by a timestamp, the node concerned, the source of the information (Node or Manager) and the description of the event.

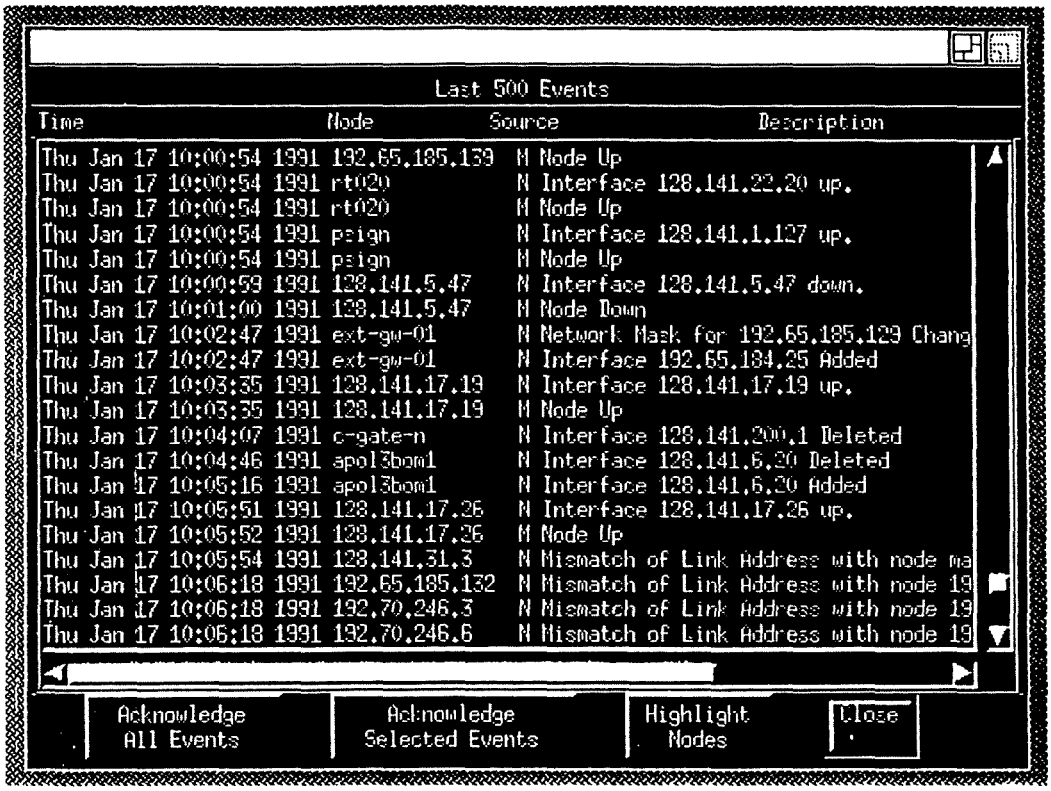


Fig. 5.3: All Events display

5.1.3. Devices Polling

It is possible to enable or disable the polling of the nodes and to fix the intervals between two pollings for all the nodes. Different polling intervals can be selected, each corresponding to a particular function. These are: discovery of new nodes, status of one node, thresholds polling (system load, diskspace, interface).

5.1.4. Traffic Monitor

A. Presentation

It is possible to fix thresholds for every machine in particular. It is only needed to select a node, choose a type of polling and fix the threshold. The types of polling are *interface percent deferred*, *interface percent collisions*, *interface CRC errors*, *interface percent input errors*, *interface percent output errors*, *CPU load* and *percentage of used disk space* .

The program is able to produce on-line graphs using some parameters of one node. It can make graphs representing the CPU load of a node (only available for specific stations), the interface traffic, the link transmissions and link receptions

(available for the SNMP hosts). An example of a graph on the interface traffic is shown in figure 5.4. It shows the out packets on the "ethernet 0" interface of gateway "ext-gw-01".

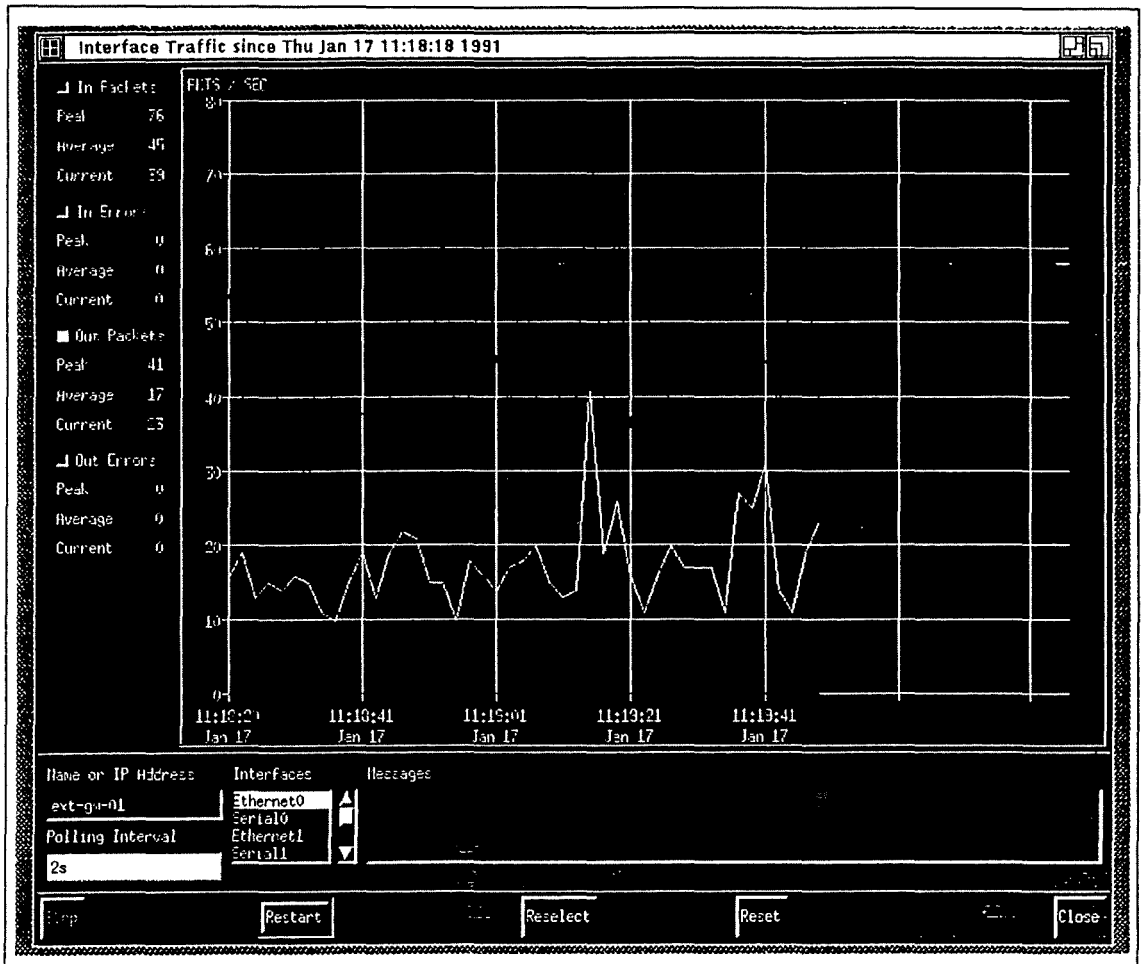


Fig. 5.4: Interface Traffic

B. Comments

The threshold function is indispensable in a management system. But here, We think it is too limited. Only 6 variables are available, and amongst these 6 variables, only 4 can be used normally and only concern interface dependent parameters. The last two ones are HP dependent. It should have been possible to fix thresholds on a wider choice of variables.

5.1.5. Reachability Tests

A. Presentation

It is possible to test if a node is working or not (up, down). This test is made by the ping procedure. The program sends ICMP echo packets and wait for answers from the tested machine. The time between the sending of every packet and the response is shown. The program also calculates the maximum, minimum and average time between the sending and the reception.

Another interesting test to make is the remote ping test. The program forces a remote node to send packets to another one. The remote node listens to an answer and sends the results to the program.

The program is able, for every node, to make an immediate demand poll. The polling consists for a normal non-SNMP node of a ping test, a demand of description and a verification of the nodename. The SNMP nodes are also requested to send their internal tables like routing table, ARP cache, services, etc.

B. Comments

These functions are useful in a management program. They represent the minimum facilities. All of them work well.

There is also a monitoring program called netmon which collects information like node up/down from the managed objects and sends it to the graphical representation of the network in order to update the colour of the nodes. But this program dies too frequently. This problem was investigated but not yet resolved when this test was performed.

5.1.6. Data Query

A. Presentation

For the nodes supporting the SNMP protocol, more information can be retrieved by using the Request menu. It is possible to obtain immediately all the events that occurred on this node, the interfaces, the addresses, the routing table, the ARP (Address Resolution Protocol) cache memory and the services provided by the node. The result of an interface query is given in figure 5.5.

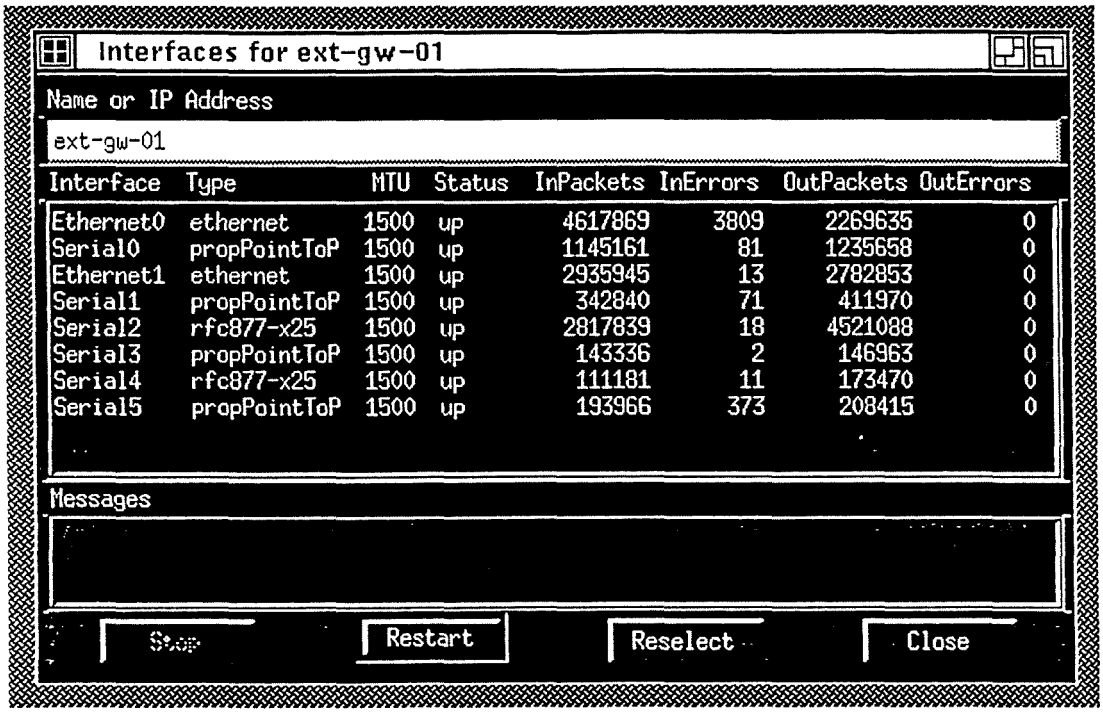


Fig 5.5: Interface Display

For devices connected to an Ethernet network, data-link counters (CRC-, send-errors,...) are also available.

The TCP connections option displays the TCP connection table that can be found in the MIB TCP group. It contains, for each connection, its state and the local and remote addresses and ports which identify the connection.

The disk space option says that no file system information is returned by the tested node (except for the station itself). This last option is HP dependent.

The program is able to test the protocol supported by every node (ICMP, TCP, SNMP). It can test if the three protocols are implemented on the machine, at the same time or one at a time.

B. Comments

These functions give only global information about an object. They give routing tables, interfaces, etc, but they do not give more detailed information like the value of a particular MIB variable or the error rate on a specific interface of an SNMP object. Information given by these functions cannot be used efficiently to detect, for example, troubles on the network.

5.1.7. Others

It is also possible to connect directly to a computer simply by selecting this computer on the segment view and using the connect option of the last menu. The same mechanism is used to connect to a gateway. It is done by using the Telnet Protocol.

The last possibility offered by this program is to start the System Administration Manager (SAM) on a remote node (HP-UX) in order to reduce the management related CPU load on this station.

5.2. Conclusion

In conclusion, Openview has two advantages and five disadvantages.

Openview is attractive because it is a well finished product. It looks very professional and is, in fact, rather easy to use with the mouse. For example, the menus and the windows are designed in a perfect way. The common operations are working satisfyingly. It is very pleasant to work with a program possessing a graphical interface like the one possessed by Openview.

Furthermore, the graphs that can be displayed are very pleasant and very readable.

But its disadvantages are important.

The first is that it is only able to manage the nodes. No operations can be made on the network itself. For example, it is impossible to listen to the broadcast traffic. The only way to have an idea of the traffic is to examine the number of packets passing through gateways, which is not representative at all.

The second severe problem is that it is never possible to save anything on files. All information gathered by the program (tables, rates,...) can only be displayed on the screen. When it disappears from the screen, it is lost. Long tables are easier to read on a paper copy than on the screen. And if data are stored on a file, they can be used by other applications, for example, by a statistics database.

The following problem is the poor capacity of gathering information. The program is too limited. It is only able to retrieve a few tables such as services table,

routing table, interfaces table and values of some variables such as crc-errors, number of packets sent, number of collisions from a remote object. It is not sufficient to manage a network. For example, the possibility to retrieve MIB variables from a remote object would have been very useful.

Furthermore, it is impossible to act directly on remote devices. For example, it is not possible to change the value of an MIB variable from the management station.

Another problem is that Openview is unable to manage the bridges. Nothing is available to get information from the bridges. Therefore, the program is less useful for some internal networks (bridged Ethernets) than for external networks.

To end the conclusion, We would say that Openview is designed for the network manager and not for operators. Operators need to be warned as quickly as possible if a problem occurs. And this is impossible with Openview which only displays a small error box. Openview is not a satisfying product for the operators. It can be satisfying for the manager to have a global view of the network and to perform the first tracking operations when a problem occurs. But in this eventuality, the manager needs in complement more specific tools, to investigate more deeply.

CHAPTER 6 : DIGITAL NETWORK TOOLS

This chapter describes an experience with the new Digital product for managing networks, the *Digital Network Tools* (DNT).

All the possibilities offered by this program will be reviewed in the first section. And then, a conclusion about this product will end the chapter.

6.1. Possibilities of DNT

The DNT program uses SNMP, Decnet or CMOT to manage a network. It is running on a Decstation with the Ultrix (Unix from Digital) operating system. A colour screen is mandatory to display the information. This is made using the Decwindows graphical interface.

6.1.1. Graphical User Interface

A. Presentation

As presented in figure 6.1, the graphical user interface is composed of four windows.

→ The first one is the general window, called **Network Map Application**. It is used to draw the map and support the menus.

→ The second window is the **Palette** window. It is divided into three parts. The first part gives a legend of the different colours used to display the elements of the map. The second part contains symbols which represent objects available to design the map. And the third part contains symbols which represent functions the user can perform. It is possible to hide the palette window if the map does not need to be changed anymore, in order to enlarge the other windows.

→ The third window is the **Navigation** window. It is used, if the map is too big, to navigate from one part to another. In the small square, the part of the map currently displayed in the network map application window is shown off.

→ The fourth one is the **Alarm Management** window. It is used to display alarms. This will be explained hereunder.

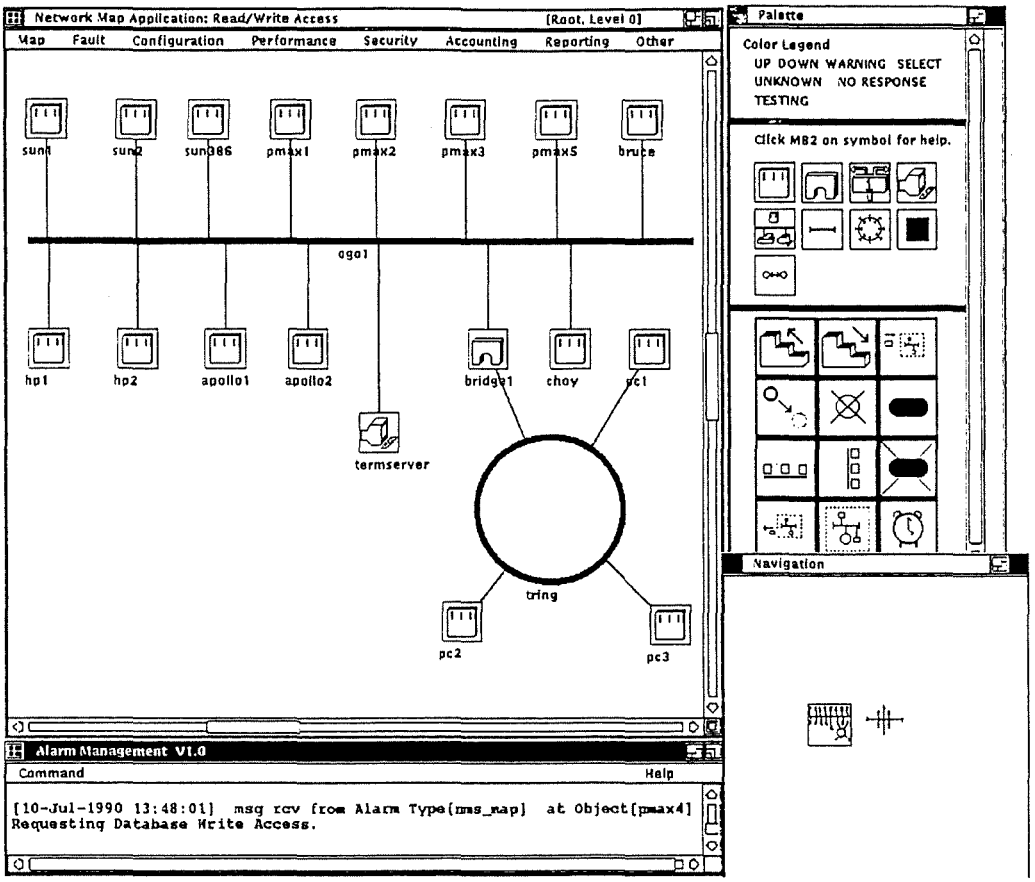


Fig. 6.1: DNT Main Screen

The network map application window does not only display the map, but it is also used to support the menus. Eight menus are visible. Five of them correspond to the five management categories defined in the OSI management framework [ISO7498-4]: configuration, fault, performance, security and accounting management. The other three menus are more specific and are used for performing map, reporting and other (remote connections) operations.

When drawing the map, the user places objects wherever he wants only by clicking on the corresponding symbols of the palette. To make a connection between two objects, the user can use the corresponding symbol on the palette or a connection function on the map menu. This function is mainly used to make a connection between two objects which does not appear in the network map application window at the same time because they are placed too far from one another.

To map can also be built with several levels. The user must choose the view symbol on the palette to represent a sub-map. Then, when clicking on this symbol on

the map, the user enters in a new view and he has the possibility to build a new sub-map.

If the map if too large, it may not fit into the map application window. Only a part of it is displayed on the main window. And it is not easy to search the entire map in order to find an object. The user is allowed to find an object by its name. If the object is found, the view to which the object belongs is displayed on the window.

The user can save the map. The map is not automatically saved when a change is made. Furthermore, the user can rebuild a previously saved map. This permits to recover the last saved map, providing the latest changes have not been saved.

A function to print the map is also available. This function is very useful because, in general in the map application window, only a part of the whole map is displayed.

B. Comments

It is easy to draw the map manually. All the drawing functions are available on the screen (palette). But it becomes more complex to work with different views because some objects are represented on both the upper and the lower views. For example, if an Ethernet segment is to be represented by a view, the gateway between the Ethernet segment and the rest of the network will be shown on both views. But its place is the same on the two views. Therefore, this object must be placed in order that two objects are not superposed.

6.1.2. Database

Six different databases can be distinguished in DNT: a service database, an object database, an interface database, a group database, a topology database and a statistics database.

Presentation

In the ***Service Database***, it is possible to save detailed information concerning the vendors, the contracts, the location of devices and the contacts which are important for network management. The user can add, update, query and delete

details for any service. A facility lets the user copy information from or to files. An example of vendor information is given in figure 6.2.

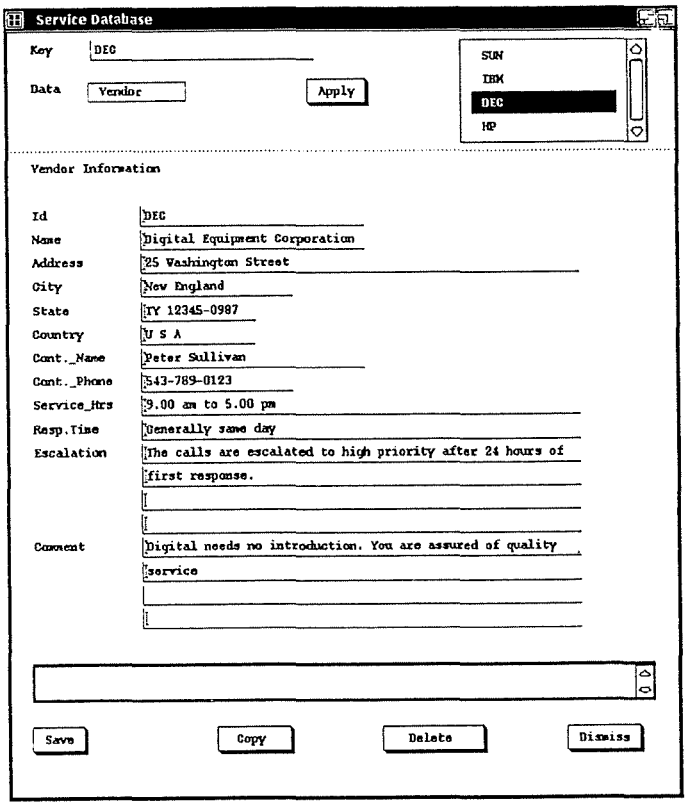


Fig. 6.2: Vendor information

The ***Object Database*** allows the user to manage information about network objects. The information is categorized into four classes which correspond to the service classes. An object is associated with an occurrence of each class. The user can add, update or query all these classes. If some service information has not been defined, it can be added from the object database. Thus, it is possible to update the service database directly from the object database and associate the services with the object.

The ***Interface Database*** allows the user to enter information about an object and its interfaces. The object information is the name, the community name, the group name, the protocol used to discover the object status and the polling interval. The interface information is the connection type, the name and the address of the interfaces, the protocol used for polling and the priority for polling the interfaces. When the user enters a new interface, he must type the name of the interface and if it is possible to resolve it using the Domain Name Server protocol or some other means, the address of the interface can be displayed by clicking on the address field. An example of the interface database screen is shown in figure 6.3.

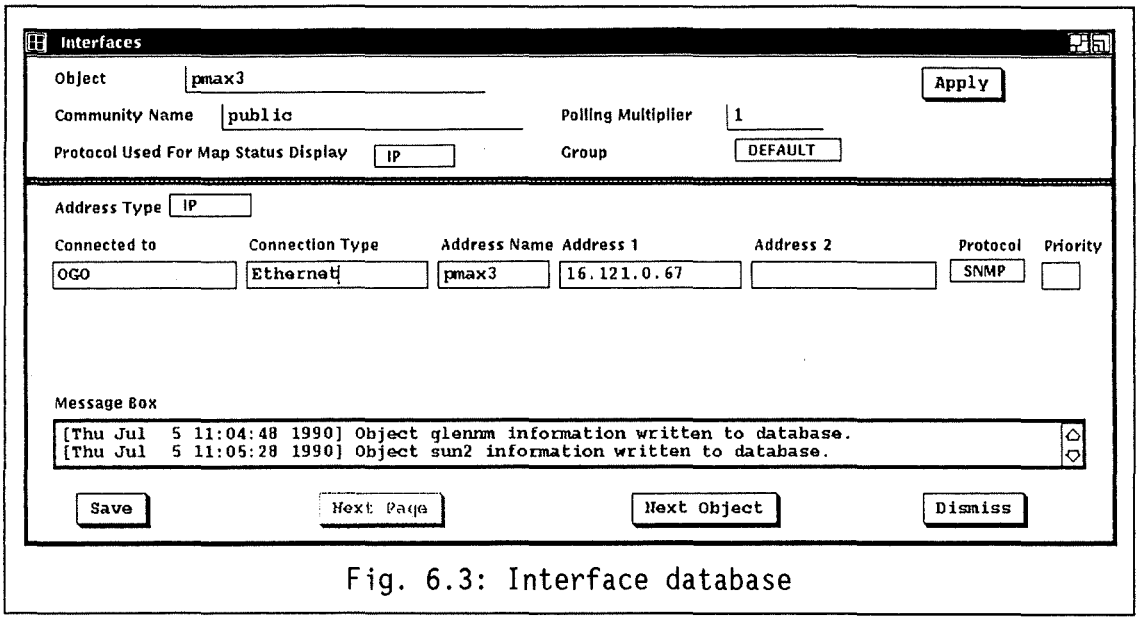


Fig. 6.3: Interface database

The ***Group Database*** allows the user to manage the groups. A group is a gathering of one or more objects which possess a common characteristic. An object can belong to more than one group. For example, it is possible to define an SNMP group which contains all objects that can be polled using the SNMP protocol. This permits to divide the consumption of resources between different computers and to reduce the global load of the network. Each group of objects can be polled by a different agent. And if the groups are designed taking into account the location of the managed objects, the polling agents can be installed in the same areas as the managed objects. It minimizes the traffic between the objects and their polling agent.

The ***Topology Database*** contains all the information necessary to display the map of the network. The information is the place of the objects on the map, their status, etc. This database is directly queried in order to display, on-line, the state of the network on the graphical user interface.

The ***Statistics Database***, an INGRES database, is used to store and retrieve reporting information. Objects can be polled at regular intervals in order to store the value of variables in the database. The database can also receive information from incoming traps. With the data contained in the statistics database, reports can be generated. These reports are built by a report generator. But reports can also be created by a user. They can be made using an SQL interface, an INGRES Report Writer or embedded SQL commands through a C program.

Comments

In the *service database*, information can be copied to a simple file in order to use it to add new objects which have nearly the same characteristics. It is, for example, possible to save the information about contract X and then use this information in contract Y. But it is also allowed to copy information concerning a service to a file and then load this information within another service. No error message is displayed but the window is filled with rubbish. The program does not check if the information belongs to the same service.

As has been said in the first chapter, it is vital to have an inventory of objects connected to the network. But the service and object databases do not fulfil this requirement as they should. First, they only contain what could be called administrative information. That is location, contract, etc. Administrative information is indispensable but not sufficient. Network information such as IP or Decnet address, physical address, is also needed. If, for example, a problem is detected coming for object with physical address X, it is important to know to which object it corresponds and then, when the object is found, information is needed such as vendor, contract and location to resolve this problem. Second, this function does not perform any network query or verification. That is the result of the first problem. The databases have no relation with the network. There should exist a link between the objects in the databases and the packets passing on the network. And, for example, this link could be made by an address field in the object database or by a relation with the interface database.

In the *Interface Database*, only two connection types are available, Ethernet (or token ring) and point to point serial. In fact, more types are needed in a real network. For example, if an X25 connection is present in a gateway, it is not possible to display it. The program should, at least, for the gateways, examine the interface ifTable_ifEntry_ifType MIB variables to see the number and the type of the interfaces on one object.

6.1.3. Topology DB builder

An autotopology function to draw automatically different maps (for Decnet nodes, SNMP nodes, etc) is in plan. But it was not yet available for this field test, so it could not be tested.

6.1.4. Alarm manager

There is only an alarm window which displays the events. Every message contains a time stamp, the object name, the type of the alarm and an alarm message. It is possible to make searches on this window using different arguments. But this is not very friendly. It would have been better to class the messages, for example, according to their type.

6.1.5. Device polling

Device polling is made through programs which send the information they gather to a topology database. And the map is directly updated using this database. The colour of an object is determined by its state. The real state is displayed only if this object can be examined using SNMP or Decnet. A non-SNMP or non-Decnet object is marked as unknown. But for the gateways supporting SNMP, DNT displays the state of each interface and the global state of the object.

6.1.6. Traffic monitor

The traffic monitoring function can be divided into two parts. The first one is an Ethernet traffic monitoring function. The second one is rather a traffic monitor for device interfaces.

Presentation

The ***Ethernet Packet Filter*** function provides the user with a display of the load (percentage of the total bandwidth in use during a particular period) on the Ethernet. The user can choose between displaying the aggregate load and displaying a trend graph. He can also choose to display a separate graph for each packet type or for each source or destination address. Many other display options are available.

The ***Monitor Interfaces*** function monitors all interfaces of an SNMP object. It counts the number of input and/or output packets on each interface and displays one or two graphs for each interface. The user can choose to display the number of input and/or output bytes per interface. He can also choose between displaying input information, output information or both. An example is given in figure 6.4.

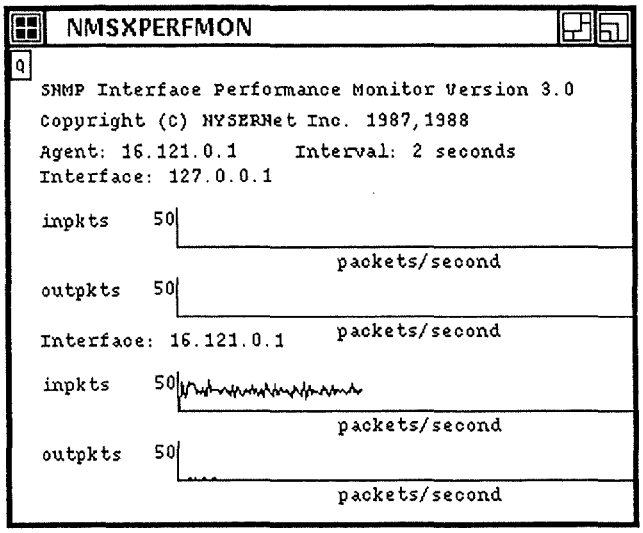


Fig. 6.4: Monitor Interfaces

Comments

A monitoring function of the Ethernet is essential for the management of Ethernet networks. When problems occur, it is vital to react as quickly as possible. But, the problem is that with DNT it is only possible to monitor Ethernet traffic. It would have been better to have the possibility to monitor traffic on other networks, for example, on Token Rings. It is also impossible to fix a threshold on the network load. The only possibility given is to display a graph of the load. But in case of a problem, the program is unable to detect it and therefore no alarm can be generated.

When monitoring the interfaces, the displayed graphs are too small. It would have been better to display only one interface at the same time but to have made the graph bigger. It is not easy to have an idea of the height and of the scale of the graphs.

6.1.7. Reachability Tests

Presentation

The ***Ping*** reachability test allows to check if a host responds to an ICMP echo message. If no object is selected before invoking the function, a terminal window is created and the user must type the name of the object to ping. If an object is selected, the ping test begins immediately. It displays the ping time for every packet and when the user stops it, it computes the minimum, maximum and average ping times.

The ***traceroute Ping*** prints the route taken by an IP packet to the destination host. It uses the same display as the ping test. The same remarks are available for this function. It displays the names of the different hosts on the route and the time to reach them.

Comments

For those two tests, a terminal window is created. But the user must click in the window before beginning any action in this window. A non-terminal window would have made the use of these functions more friendly .

6.1.8. Data Query and Values Setting

Six functions are available to query and set management variables. Three of them are aimed at SNMP agents and the other ones at DECNET objects.

Presentation

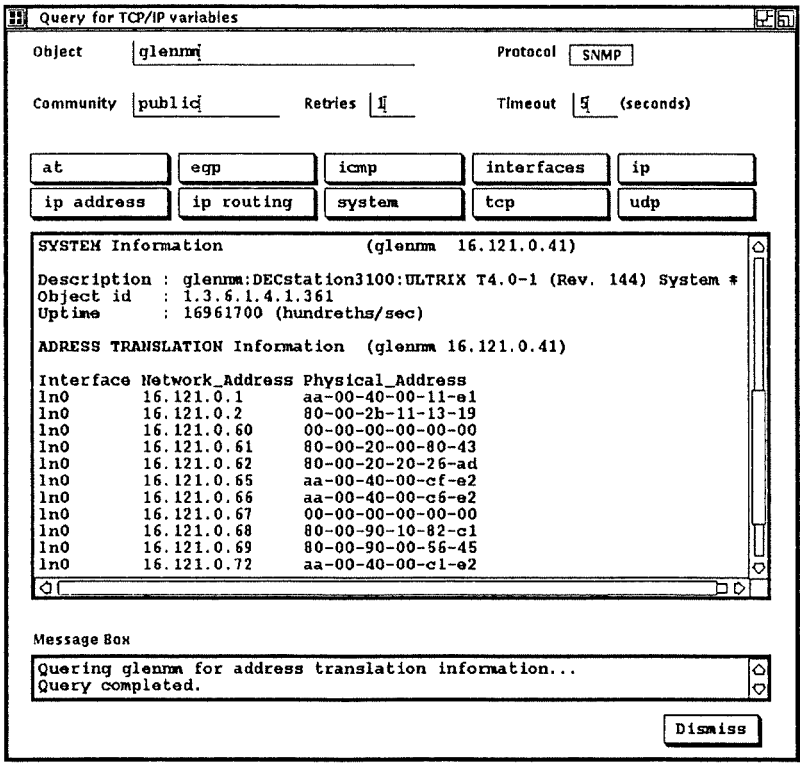


Fig. 6.5: SNMP object Query

The user can **query the SNMP objects**. He can extract information about various MIB variables from this object. It is possible to retrieve the Address translation table, IP address table, IP routing table and the values of EGP, ICMP,

Interfaces, IP, TCP, UDP and system parameters. This function is shown in figure 6.5.

Furthermore, it is also possible to automatically send SNMP requests to retrieve the value of MIB variables. The user must create a file to be used by the query function. In the file, the address of the queried object (and some details like timeout, number of retries,...), the base interval between two queries and the names of variables to query must be given. The results are saved in a file.

For Decnet objects, two query functions are available. The ***Query Decnet Objects*** function is used to query specific Decnet variables such as nodes, circuits, lines, executor, objects, node counters, circuit counters, line counters and exec counters. It uses nearly the same display as the "Query SNMP objects". There is also a possibility to save the outputs in a file.

The second one, the ***Query object using Data Dictionary***, has been implemented to support private MIB variables. The Data Dictionary contains variables which are not part of the MIB specification but which possess the same representation. This function allows the user to query a class of MIB variables. In addition, some vendor specific variables are added to the set of MIB variables, for example, Cisco variables which are specific to Cisco routers. Others can be added by the user regarding his needs. The user also has the possibility to use files. A file menu can be displayed in which he can choose between saving or logging output to a file. The save option prints in a file everything that has been done previously since the call of this function. The log option prints in a file everything that is done by the user until this option is stopped or until the function is dismissed.

For setting the value of parameters, two functions are available, one for SNMP objects and the other one for Decnet objects. The ***Set SNMP Object*** function allows the user to choose a MIB variable and to set its value (if it is possible). When this function is invoked, the object is queried for the existing value of the MIB variable that has been specified. Then the user can decide to change this value. The program displays a message if it is successful in performing the change.

The ***Set Decnet object*** function allows the user to interactively issue all NCP commands to any Decnet objects. It also gives the possibility to use an input file containing a list of NCP commands if, from time to time, the same actions have to be performed. The commands can be executed on more than one node at the same time. A menu enables to use files. It is possible to write the output window in a file, to log

output in a file or to save the commands performed in a file. This function is illustrated in figure 6.6.

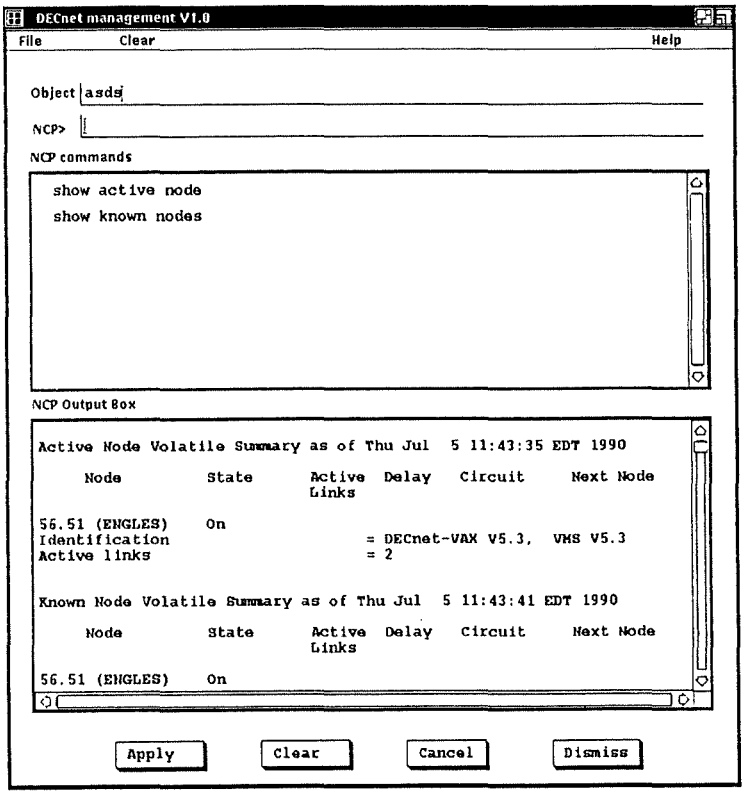


Fig. 6.6: Set Decnet Object

Comments

The **query SNMP objects** function is useful because it rapidly gives general information about a specific object. But there are some problems. First, when the user asks to print a table, if the table is too long, it is not possible to stop the display. The user must wait until the end of the query. The second problem is related to the message box. If the same error message occurs more than once, the user cannot see if more than one message was printed. There is no difference on the screen. The only way to see the difference is to count the number of messages using the scroll bar.

The ***Query SNMP object using config file*** function is useful because it allows to make off-line observations and over a long period of time. It can be used, for example, to gather statistics or to monitor a specific MIB variable. But it is impossible to stop the query function within the program. The user is forced to kill the process if he wants to stop the query.

The ***Query Decnet Objects*** function is useful because it rapidly gives general information about a specific object. The information is grouped in categories which are labelled in the right way. But some problems are present. There is a help menu but when clicking on this menu, nothing works, no help appears.

The Data Dictionary has been made to allow the user to deal with non-standard variables and vendor-specific variables. The manager can add these variables to the Data Dictionary and retrieve their values by using this function. This is very interesting because all cases of manageable objects are not foreseen in the MIB. In the function, there is a problem with the cancel button which does not work correctly or does not work at all. When querying a large table (IP routing table for example), it is not possible to stop the display. The only solution is to stop the query function with a kill command.

The ***Set Decnet object*** function is very powerful. It allows to manage all Decnet objects from the management program. But the problem is that the user must know all NCP commands to use it. It would have been better if the program could help the user in issuing these commands with, for example, a menu-based interface.

6.1.9 Monitoring and Statistics Computing

Presentation

The **Plot MIB Variables** and **Bargraph MIB Variables** functions allow to display the value of an MIB variable and to display a graph of the value of a MIB variable over time (only for the numeric variables). This function consists of three steps. (1)The user must give the object name and choose a variable class. Then a new window is created, it shows all the variables of the chosen class. (2)Then the user selects one or more variables to print their value at the present moment. (3)After that, the user can select one variable for which the value is printed, and use it to plot a graph or a bar graph. An option of the plotting window exists to modify the interval between two plots during the display and without stopping it.

Another function gives the possibility to gather SNMP information from various objects and store the information in the statistics database. For specifying the objects to query and the variables to retrieve, a screen like the one shown in figure 6.7 is used.♠☺

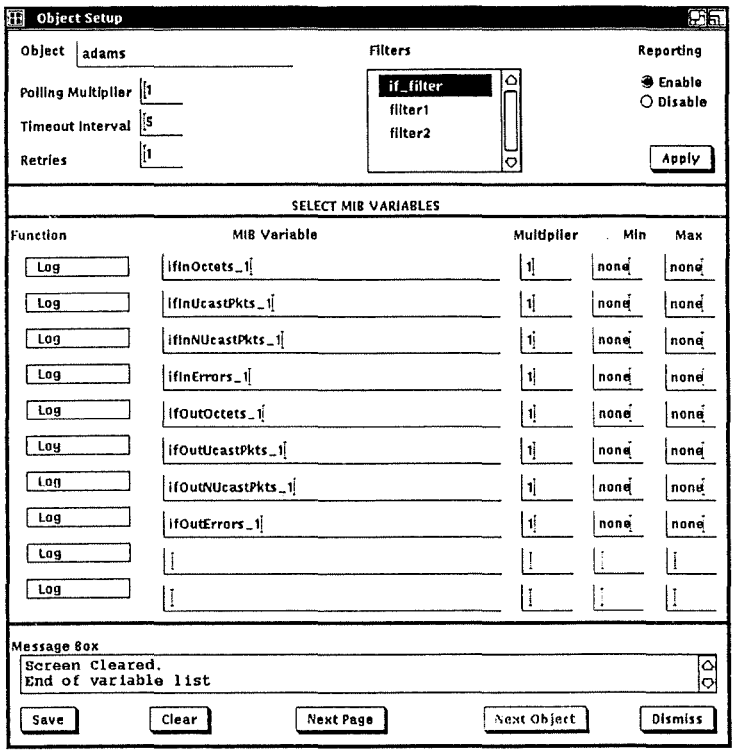


Fig. 6.7: Reporting object setup

Furthermore, this function allows to fix minimum and maximum thresholds on MIB variables. And if one of these thresholds is bypassed, then an alarm is automatically generated.

Comments

Various problems can be found when using the plot and bargraph functions. The first one is that in the Plot MIB variables, there is no time scale on the graphs. In the Bargraph MIB Variables function, when the user wants to change the display interval, the display of the interval becomes unreadable and then all the display inside the window vanishes.

These two functions could have been grouped. The only thing that changes is the last window. An option in one of the first windows could have let the user choose between the drawing of a normal graph or a bar graph.

It should be noted that for thresholds fixing, only the value of MIB variables can be used to fix thresholds. This seems sufficient but, in fact, MIB variables are, in majority, values such as counters, integer, gauge, etc. It has little sense to fix thresholds on such values. For example, nobody cares to know that 540 errors occur

since the last reboot of the device. Rates are more important, for example, 10 errors per second.

6.2. Conclusion

In conclusion, DNT is a rather good product. Many elements listed in the first chapter are present. But it is not perfect because some useful functions are not implemented and others need to be improved. However the program that has been tested is still on field test and improvements are foreseen for the next versions.

Below are listed the important features of the program. Only a few of them are negative. But many of the positive points must be moderated by negative aspects.

The graphical representation of the topology of the network is pleasant. The display is good and obvious. The tools to draw the map are always present on the screen and are easy to use. But the drawing of the whole map is rather painful. It is difficult to remember to enter the name and the interfaces every time a new object is added. But this problem should be solved in the next version of the program. In this version, the autotopology function should be present. This should well improve the friendliness of the program for the design of the map.

A big advantage of DNT is that it will give the possibility to manage objects using different protocols (SNMP, CMOT, DECNET). It is useful because the management is not restricted to a single protocol. For the moment, CMOT management is not yet available.

A good idea of DNT is the implementation of databases. They are used to give information on the services, the objects and their interfaces. It is an indispensable function in a network management system. But this function is not sufficient. Alone, it is only a gathering of information. Such a database needs to be dynamic. For example, the program should have functions to interact with the network, functions to verify that the databases and the network are coherent. And nothing like that is available in DNT.

An advantage of DNT is the possibility to print reports using statistics about the objects. The information is stored in an INGRES database, therefore it is possible to retrieve this information with the built-in tools (print different types of reports) or with the special tools made by the user. But these statistics are only concerning MIB variables which consist in gauge, counters, integers, etc. That is not very interesting

in this state. What would have been much more interesting is information about rates, for example *error rates, traffic, throughput rates (packets/sec, bytes/sec), etc.*

A problem of the program is that it is impossible to send useful alarms when thresholds are exceeded. It is possible to print alarms if thresholds are exceeded but the thresholds can only be defined on MIB variables. For example, it is not interesting to know the number of errors on an interface but what is interesting is to know the error rate on this interface. It is more interesting to know that there have been 100 errors/second than to know that there have been 1280 errors since the last reboot of the machine. And it is not possible to define thresholds on such variables.

Another interesting feature is the possibility to print graphs of the load on the Ethernet. It is important to have the possibility to fix thresholds on this value. And only a display on the screen is made. It would also have been interesting to use this data in the reports.

The alarms are not very easy to handle. To be useful to the operators, the alarm display must be improved. All alarms are coming in the same small window, without filtering nor classification. This alarm handling system is useful, if a problem is detected, to react, to understand what happened or to see when it occurs but it is not useful to help in the detection of the problem.

A bad point is that the program does not provide tools to manage the bridges. The management of bridges is not useful if the program is used for WAN management but if the program must be used as a tool to manage, for example, Ethernet LANs, that is a very important feature. Fortunately, this function is foreseen in a following version.

To end the conclusion, DNT can be considered as a rather good product. However it could become a better product once it is finished, for example when important features like the autotopology function and the management of bridges have been included.

CHAPTER 7 : XGMON

The analyse of this product will begin with an overview of the working of the program. Then, detailed information will be given about the different commands that are available and about the problems discovered with these commands. Finally, a global opinion concerning this program will be given in the conclusion.

7.1. Overview

XGMON is a network monitoring program developed by IBM. It is still on field test. It runs on an IBM PC/RT computer with the AIX (Unix from IBM) operating system. It uses X-window to make the display on the screen. Note that some limited facilities are available with a simple non-graphical terminal.

XGMON provides an environment that includes a command language, simultaneous execution of multiple, independent queries and an X11-based graphical presentation of the network. XGMON has been designed to manage TCP/IP networks. Only the nodes running IP, and better SNMP, will be taken into account. Some limited T1 modems management facilities are also available but have not been tested.

The major function of the XGMON core is to oversee the operations of the virtual G-machines. These are imaginary machines that execute G-code object code. The XGMON core contains a compiler that translates programs written in G (Gateway language "similar" to C) into G-code. Each virtual machine works on its own program and, because XGMON is able to control more than one virtual G-machine, more than one program can be executing at the same time. Each machine is essentially independent of all the others (it is possible to implement some inter-machine communication, if needed).

This makes that the XGMON system is extendible. New commands can be added in much the same way new commands are added to the UNIX operating system. One simply writes an appropriate program in the G programming language,

and it can be made available as a new command. New commands can be added at any time and used without having to terminate the current XGMON program. Similarly, commands that have bugs can be corrected without having to stop everything. This means that algorithms can be corrected without having to lose global state information that has already been collected. The aspect of the screen is presented in figure 7.1.

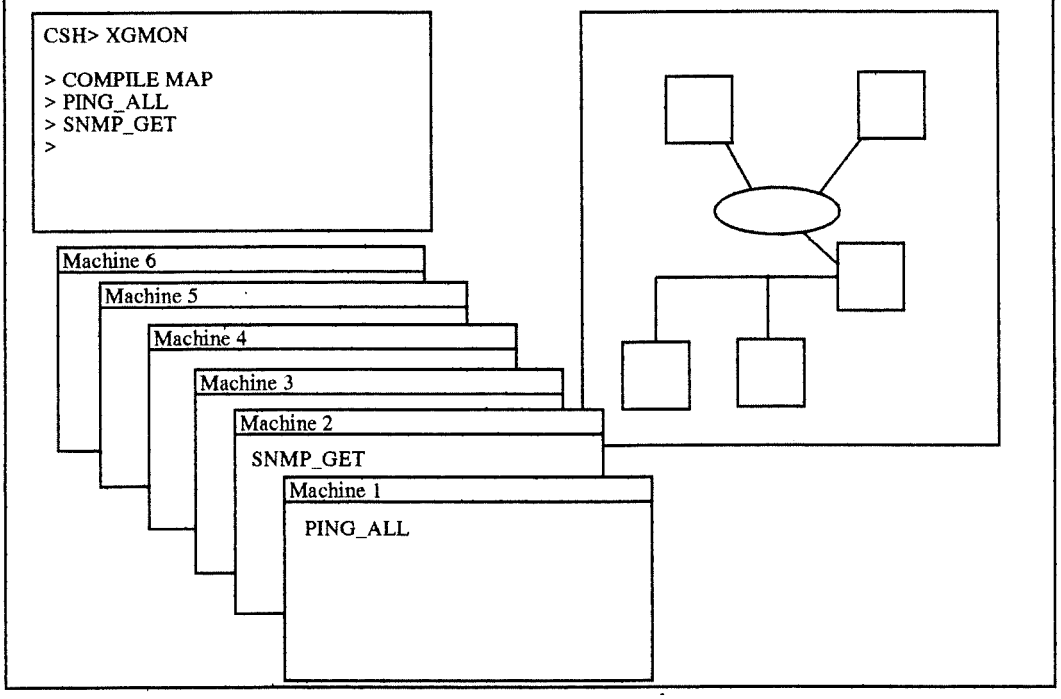


Fig. 7.1: XGMON screen

When running with X11, each virtual G machine has a window associated with it. A program (library command) running in the virtual G-machine may write output to this window. It is also possible for graphics to be drawn in this window. It is not a text-only "output device".

Each virtual G machine has a standard output device. Normally this is associated with the above mentioned window. However it is possible to redirect the output from a virtual G machine into a file instead of to the window. This makes it possible to save information that will be processed off-line.

The XGMON core provides a set of general purpose, intrinsic functions which are used by library commands to perform work. All active processing is performed by library commands which are programs external to the XGMON core. Any local customization of algorithms is embedded in library commands. The core is

normally not altered. The intrinsic functions attend with the database operations, the string manipulations, the file I/O, the graphics, etc.

XGMON possesses 2 types of commands. A small number of commands which are built into the XGMON core are classified as system commands. In general, system commands are used to control virtual G-machines. The majority of commands are classified as library commands. These are G-programs stored as source code which are compiled into G-code and executed when required by virtual G-machines.

Two mechanisms are used to reduce the load of the SNMP agents and the traffic created by the monitoring functions. The first is the use of a cache database and the second is the possibility to bind several XGMON and/or other systems (for example Netview) to an SNMP query engine which will attend with the query to SNMP agents.

Another interesting feature is the possibility to make automation. G-programs can be set up so that they will automatically get started when a certain variable changes or a user-defined event occurs.

7.2. Details

7.2.1. Graphical User Interface

A. Presentation

The user can draw a graph to represent the network. For this, he must build a file in which he gives the names of the objects and where to place them, their interfaces and the links between the objects. Then, this file must be compiled by the XGMON compiler and a window is created to display the topology. The nodes are coloured regarding their status which is found in the cache database. An object can be represented by a user-defined X-window bitmap. If any of the links or hosts defined by the current topology description are down, a signal can be played. Traps or SNMP requests are used to fix the status of SNMP objects and ping is used to fix the status of other TCP/IP objects. The width of the link between two objects can be automatically adjusted regarding the value of different MIB interface variables.

Library commands are provided to help with the display of the network.

The *display, dont_display* commands are used to control which types of display elements can be drawn on the topology display. It is possible to display or to hide the hosts, the links, the nodes, etc.

The *move* command allows to change the position of a display element within a topology display window. The user can choose between entering the name of the node and its new position or changing its position using the mouse. The *rename* command allows to change the name of a display element. The user must type the old name and the new one. The name of the object is changed on the topology map. The *add_new* command allows the user to add new objects on the topology display. He must only enter the name of the object and then place it on the map with the mouse. To make the change permanent for all these functions, the map must be saved.

The *save_win* command can be used to save the current topology information stored in the XGMON core as a G topology description file. It is used to save the topology when new nodes have been added or when the display has been changed.

The *traffic* command sets up the necessary environment to drive the links widths on the topology display based on counters such as packets in or out, octets in or out, errors in or out. This command does not acquire the needed data by itself. Another program (*snmp_p_all*) must be running to perform this task.

B. Comments

This way to build the topology map of a network is very tedious. The interactive way to enter and to place the objects is much easier. With an interactive program, the only thing to do is to choose the type of object to display, to give its name and to draw the lines representing the connections with other objects. With XGMON, a file must be used to enter the name of the object, its place, the IP addresses of its interfaces and the links existing between the objects. Then the file must be compiled and the result is displayed on the screen. If an error occurs, the file must be re-edited, modified and re-compiled.

The library commands provided to help with the drawing are not very useful. The *move* command could help to move objects but it does not work very well with the mouse. This makes that the place of the object on the grid has to be calculated. The *add_new* command allows to place an object with the mouse. But this modifies only the display, no information (for example interface information) must be entered. Therefore, the topology file must be edited to add this information. And the third

command (*save_win*) allows the user to save the topology display in a topology file. But unfortunately, a lot of information is lost during the save. Only the essential information is kept. For example, the bitmaps, the physical links, the inactive objects are not saved on the topology file. This option is only useful when the topology map is drawn for the first time. The user places the objects wherever he wants and then after saving the map, he edits the file to make further customization.

Furthermore, the topology window is too small. The coordinates are relative to a 100 by 100 grid. Then, in general, no more than 5 objects can be drawn on the same line. If the map is too big, it is difficult to draw. It is also possible to use groups of objects. If a group is defined, a new window is created when clicking on this group object. But it is not sufficient if the network is big. A bigger window should have been very useful.

The *traffic* command is useful. It gives rapid graphical view of the load on a line or of the numbers of errors on the line. All these values are calculated on a per second basis. But it is rather difficult to set it up. The user must for example adjust the scale of the line manually. This is not made automatically by the program. The user must also choose on which side of the line the measure will be made. Unfortunately, this function is not available for connections other than serial lines. It is not, for example, possible to use it for the interface connected to an Ethernet.

7.2.2. Database

XGMON disposes of only one database, a *cache database*. This gathers a lot of information and makes it available to all programs running within the XGMON environment. Whenever SNMP data or ping information is received in response to G-program initiated requests, XGMON stores that information in an internal database. The information is then made available not only to the requesting G-program, but also to any other G-program that wants that same information. The data is kept in the cache for a user-specified time. If unsolicited traps come in, that information is also stored in the database and made available to G-programs.

7.2.3. Topology DB Builder

A. Presentation

An easy way to draw the map of the topology of the network is to use programs that are able to find all SNMP nodes present on the network. XGMON integrates such a program. It is called *explore*.

This command is used to explore all the network, searching for the SNMP agents. From a beginning agent, the program searches for the new SNMP agents by looking at every interface. Once it finds a new agent, it stores the address and it will use it later to search for other agents.

B. Comments

The *explore* command is useful to find all SNMP nodes with a given community name. By looking to the result file, it is possible to see all the nodes explored by the program. Unfortunately, this command does not display graphically the nodes that it found. Neither does it build a topology file that could be compiled by XGMON to have the topology display.

7.2.4. Devices Polling

A. Presentation

The detection of problems on the network is an important goal for a network management system. For this, the user disposes of 3 mechanisms: the *snmp polling* and *trap management* for SNMP objects and the *iterative ping test* for SNMP object or only TCP/IP objects. In XGMON, the 3 mechanisms are performed by library commands.

The *ping_all* command is used to ping, at regular intervals, all IP-based hosts defined by the topology description.

The *snmp_p_all* command is used to poll all SNMP-based agents for the status of their interfaces. The agents to be polled are defined by the topology description.

The *trap* command accepts and handles SNMP traps received by XGMON. The traps are also recorded in a file in the current directory.

Multiple copies of XGMON can be used in concert when joined with an SNMP Query Engine. A single copy of XGMON can drive the topology displays of the other instances, as can any other network management system connected to the SNMP Query Engine. If the SNMP query engine has been started on one machine, XGMON can be told to obtain all data via that query engine. The database is then maintained by the query engine. Multiple XGMON instances can thus share the same data, so only one instance has to run the G-programs that do all the polling and all instances of XGMON get an indication of the status of the network. This will have as effect to reduce the network traffic due to network management and also to reduce the time spent by managed devices in responding to management requests.

B. Comments

In XGMON, this testing function is satisfying. In general, the 3 commands are run at the beginning of the working session. But the user is not obliged to run the 3 at the same time. If there are no objects other than SNMP objects, then the ping test can be ignored.

In the *snmp_p_all* command, a big problem occurs when several SNMP objects must be managed at the same time. Every time the program wants to poll the SNMP objects, either a memory fault or a bus fault occurs. This problem has been reported to IBM.

7.2.5. Traffic Monitor

A. Presentation

In a network, the relevant problems are not only problems of node failures. For example, some problems concern traffic becoming too important on one interface. Therefore, it is important to have a display of the load on all the interfaces of a host. It can also be important to give an idea of the repartition of the load during one day. In XGMON, the *perfmon* command can be used for this purpose.

The *perfmon* command displays graphically counts of incoming and outgoing packets on a specific host. The program makes on-line graphs of all interfaces of the given host.

B. Comments

Perfmon is not sufficient for traffic monitoring. Other functions should have been implemented such as functions to fix thresholds on MIB or other variables and to send alarms if one of these thresholds is exceeded.

Another problem is that it is only possible to get graphs on in/out packets. It is, for example, impossible to get graphs concerning error-rates on the interfaces of an object.

Some bugs are present within *perfmon*. The first is at the initialization. Parts of the screen background are used as the window background. Furthermore, when the scale of the graphs changes, the graphs are reinitialized. And the graphs are too small. It is not possible to have an idea of the values of the displayed counters.

7.2.6. Reachability Tests

The *ping* command pings one specific host. If the hosts responds, the time to make the trip is displayed, otherwise a message is displayed announcing that there was no response.

The *trace_path* command determines the route between two hosts. It queries SNMP agents defined by the topology description for routing information.

7.2.7. Data Query and Values Setting

A. Presentation

All these functions allow to get much information about the objects. For the SNMP objects, XGMON possesses a panel of commands which are designed to access or to set the value of all MIB variables. It is also possible to get the ARP cache of an object. And a command allows to know the way followed by the packets to go from one host to another one.

The *snmp_get*, *snmp_g_next* and *snmp_set* commands are used to request an SNMP agent to return or to modify the value of MIB variables using, respectively, the SNMP get-request, get-next-request and set-request. The *snmp_p_list* command acquires the status of one or more hosts that have an SNMP agent running. Each host must be defined by the current topology description.

The *snmp_dump* command is used to dump a table. It queries the agent as long as the returned MIB variable object IDs have the same prefix as the given variable.

Other commands are available for retrieving table information. For example, it is possible to retrieve the IP address table, the IP routing table, information associated with the interfaces of a host or router, the IP address translation tables (ARP cache), the table of EGP neighbours and the table containing TCP connection-specific information of an SNMP agent. It is also possible to get values of other MIB variables, for example, ICMP counters, IP-related counters, TCP counters and UDP counters.

B. Comments

All these functions give interesting information about all SNMP objects connected to the network. All MIB variables can be known, either using group commands or using single SNMP get requests. To a certain extent, SNMP objects can be controlled by setting SNMP MIB variables. All these commands work well. However, the number of such functions is too important. Because it is not possible to remember the name of each variable, a better way of doing such requests would have been to use one or two functions integrating menus. Furthermore, the use of these functions is not easy. The user is forced to type every time the name of the command, the name of the object to work with and sometimes the name of the requested MIB variable and some options. A better solution, at least, would have been to give the possibility to select the objects on the map in order to avoid typing their name. The best solution to perform these functions is to use menus to select the functions and the names of the variables and to give the possibility to select objects directly on the user interface.

7.2.8. Automation support

A. Presentation

An important aspect of a network management system is the possibility to perform some actions automatically when a given event occurs. For example, it is interesting to have a program that detects if a host has problems and to send a command to reboot it. XGMON possesses 3 library commands to help in the support of automation.

The *add_filter, del_filter* commands are used to add or delete automation filters. The automation filters detect when the value of a given MIB variable changes, and in this case, start library commands specified by the user. These commands could permit the creation of a program which would be able to detect when a variable trespasses a given threshold. The problem is that the number of allowed filters is restricted. This would work only for a limited set of variables.

When invoked, the *display_var* command obtains the desired data from the internal cache and displays it. It is, in general, used for debugging. For example, if this command is given as an argument to the *add_filter*, in association with an MIB variable, every time the value of the given variable changes, its new value is displayed by the *display_var* command.

B. Comments

It is very interesting to have the possibility to automate certain tasks. With XGMON, it should be rather easy to do nearly everything. But this automation is only possible within a certain range. Commands can only be run if the value of a given MIB variable changes. It is not possible to make automation with other arguments.

However the automated task is not defined. The user can do everything he wants. He can build a program which will serve as an argument to the *add_filter* command.

7.3. Conclusion

XGMON possesses 3 characteristics that make it attractive. They allow to reduce the network and the hosts load due to management operations, to make automation and to write one's own applications.

The first characteristic is that XGMON possesses an SNMP Query Engine and a cache database. These two particularities are aimed at reducing the traffic and the load of devices created by the network management systems. Now, as the number of network management systems is growing, it is important to load the hosts and the network by traffic and not by management requests. However, the Query Engine is an IBM product and only Netview and XGMON are able to share the data collected by it.

The second characteristic is that XGMON is able to do automation. It is possible to start some tasks when a special event occurs. This possibility can be useful in an operational environment. For example, before warning the operators that a problem occurs, a number of operations can be performed to resolve the problem automatically.

And the third characteristic which is the most interesting one is the possibility to write one's own programs. As XGMON possesses a compiler, it is rather easy to write new commands that can be added to the set of already available library commands and that can be used as any other one.

But on the other side, XGMON does not possess interesting functions which are nearly indispensable in the normal management of a network.

The first one is that it does not perform management functions at a layer below the IP layer. For example, it is not able to manage Mac-level bridges. This makes it unable to help in the management of LANs composed of, for example, several Ethernet segments connected by Mac-level bridges. XGMON is only able to manage SNMP or TCP/IP hosts. And, for the moment, in LAN management, SNMP is rarely used. However, for WAN management or the management of other LANs, this characteristic is not important at all.

The second problem is that XGMON does not give the ability to fix thresholds on different variables without writing the necessary programs. This is a big problem because this type of failure cannot be detected. The only detectable failures are the failures concerning a node or a line going down.

Furthermore, there is, in XGMON, no obvious display of the alarms. No library command is available to gather and to display all the alarms in an obvious way. To make the program operational, such a function must be written by the user.

Another negative point is that XGMON is only a host management program. It is not a product for managing a network. It is, for example, interesting in a LAN to know the number of broadcast and multicast packets loading the network. This is not possible with XGMON but for WAN management, it is not important.

And the last negative point is that XGMON does not use menus. For some commands, it is not important, but for others it is nearly indispensable. For example, in the *snmp_get* command, the user must enter the name of the MIB variable to query

and I think it is not easy to remember the right syntax for each variable. The use of menus would have been useful.

In conclusion, XGMON is not a product that can be used by a network manager without a consistent programming work. It does neither possess easy-to-use query functions nor below-IP management functions and the alarms are not handled.

However, it can be used by the operators. If an alarm management function is written, it is possible to display a view of the network and of the errors which can occur on it. But work must be done to make this product usable.

One of the goals of this program is to give tools to a programmer to write the commands that he really needs to manage his network. In this manner, XGMON fulfils its task and it can be seen as a good development tool.

CONCLUSION

For people developing and maintaining networks, the problem of network management becomes more and more important.

This is due to at least two reasons.

The first one is that the needs for computers and communications between computers are increasing. So the size of the networks increases.

The second reason is that, in general, as the offer of network devices is more varied, people do not buy all the devices from a single vendor. This makes that the networks are more heterogeneous than before.

The need for powerful tools for managing bigger networks begins to be faced by the apparition of integrated network management systems like the one described in chapter 1 and like the ones tested in chapters 5, 6 and 7.

In what concerns the heterogeneity of the networks, ISO has begun to answer the needs for common management protocols by specifying a set of network management standards with CMIP as the centre. But OSI networks are not very widespread yet and, even if CMIP is a common management protocol and if it is not only intended for OSI networks, CMIP implementations are rare.

On the other hand, TCP/IP networks are developing very rapidly. And the Internet Activity Board, the TCP/IP authority, managed to respond to the needs for management systems. This has been made in two steps.

The first step was to develop, as fast as possible, working systems, even simple, to respond to urgent needs. This led to the specification of SNMP. SNMP is now in tune. Many implementations can be found on the market. (3 of them have been tested). Furthermore, many vendors propose an SNMP agent with their network devices.

The second step is the specification of CMOT, derived from CMIP. CMOT is aimed at making easier the transition from TCP/IP to OSI. But CMOT has few chances of emerging because it focuses on the place caught by SNMP, which SNMP fills satisfyingly.

Specialists predict that both SNMP and CMIP could be used in the future for managing networks. SNMP would be used for the communications between management stations and agents and CMIP would be used for the communications between management stations.

This work was particularly based on TCP/IP and on the management of TCP/IP networks with SNMP. But further analysis of CMIP and of CMOT could help to see whether CMOT is definitely dead and whether CMIP and SNMP are "compatible" and could be used in concert to manage networks.

APPENDIX A : SMI

```
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

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

OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
                    | "read-write"
                    | "write-only"
                    | "not-accessible"
    Status ::= "mandatory"
                    | "optional"
                    | "obsolete"
END

-- names of objects in the MIB

   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
```

APPENDIX B : MIB-II

```
RFC1158-MIB

DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks
            FROM RFC1155-SMI;

mib-2       OBJECT IDENTIFIER ::= { mgmt 1 }    -- MIB-II
            -- (same prefix as MIB-I)

system      OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces  OBJECT IDENTIFIER ::= { mib-2 2 }
at          OBJECT IDENTIFIER ::= { mib-2 3 }
ip          OBJECT IDENTIFIER ::= { mib-2 4 }
icmp        OBJECT IDENTIFIER ::= { mib-2 5 }
tcp         OBJECT IDENTIFIER ::= { mib-2 6 }
udp         OBJECT IDENTIFIER ::= { mib-2 7 }
egp         OBJECT IDENTIFIER ::= { mib-2 8 }
-- cmot     OBJECT IDENTIFIER ::= { mib-2 9 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp        OBJECT IDENTIFIER ::= { mib-2 11 }


-- object types

-- the System group

sysDescr OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        ::= { system 1 }

sysObjectID OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-only
        STATUS  mandatory
        ::= { system 2 }

sysUpTime OBJECT-TYPE
        SYNTAX  TimeTicks
        ACCESS  read-only
        STATUS  mandatory
        ::= { system 3 }

sysContact OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-write
        STATUS  mandatory
        ::= { system 4 }

sysName OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-write
        STATUS  mandatory
        ::= { system 5 }

sysLocation OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        ::= { system 6 }

sysServices OBJECT-TYPE
        SYNTAX  INTEGER (0..127)
        ACCESS  read-only
        STATUS  mandatory
        ::= { system 7 }


-- the Interfaces group

ifNumber OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { interfaces 1 }

-- the Interfaces table

ifTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IfEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { interfaces 2 }

ifEntry OBJECT-TYPE
        SYNTAX  IfEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifTable 1 }

IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
```

```
        DisplayString,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        OCTET STRING,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge,
    ifSpecific
        OBJECT IDENTIFIER
}

ifIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 2 }

ifType OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),           -- none of the
                                        -- following
                    regular1822(2),
                    hdh1822(3),
                    ddn-x25(4),
                    rfc877-x25(5),
                    ethernet-csmacd(6),
                    iso88023-csmacd(7),
                    iso88024-tokenBus(8),
                    iso88025-tokenRing(9),
                    iso88026-man(10),
                    starLan(11),
                    proteon-10Mbit(12),
                    proteon-80Mbit(13),
                    hyperchannel(14),
                    fddi(15),
                    lapb(16),
                    sdlc(17),
                    t1-carrier(18),
                    cept(19),       -- european
                                    -- equivalent of T-1
                    basicISDN(20),
                    primaryISDN(21),
                                    -- proprietary
                                    -- serial
                    propPointToPointSerial(22),
                    terminalServer-asyncPort(23),
                    softwareLoopback(24),
                    eon(25),            -- CLNP over IP
                    ethernet-3Mbit(26),
                    nsip(27),           -- XNS over IP
                    slip(28)            -- generic SLIP
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 3 }

ifMtu OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 4 }

ifSpeed OBJECT-TYPE
        SYNTAX  Gauge
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 5 }

ifPhysAddress OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
        SYNTAX  INTEGER {
                    up(1),      -- ready to pass packets
                    down(2),
```

```
                testing(3) -- in some test mode
            }
        ACCESS  read-write
        STATUS  mandatory
        ::= { ifEntry 7 }

ifOperStatus OBJECT-TYPE
        SYNTAX  INTEGER {
                up(1),      -- ready to pass packets
                down(2),
                testing(3) -- in some test mode
            }
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 8 }

ifLastChange OBJECT-TYPE
        SYNTAX  TimeTicks
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 9 }

ifInOctets OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 10 }

ifInUcastPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::=  { ifEntry 11 }

ifInNUcastPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 12 }

ifInDiscards OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 15 }

ifOutOctets OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 16 }

ifOutUcastPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 17 }

ifOutNUcastPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 18 }

ifOutDiscards OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 19 }

ifOutErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 20 }

ifOutQLen OBJECT-TYPE
        SYNTAX  Gauge
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 21 }

ifSpecific OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ifEntry 22 }

nullSpecific OBJECT IDENTIFIER ::= { 0 0 }

-- the Address Translation group (deprecated)

atTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF AtEntry
        ACCESS  read-write
        STATUS  deprecated
        ::= { at 1 }

atEntry OBJECT-TYPE
        SYNTAX  AtEntry
        ACCESS  read-write
        STATUS  deprecated
        ::= { atTable 1 }
```

```
AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        OCTET STRING,
    atNetAddress
        NetworkAddress
}

atIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  deprecated
        ::= { atEntry 1 }

atPhysAddress OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  deprecated
        ::= { atEntry 2 }

atNetAddress OBJECT-TYPE
        SYNTAX  NetworkAddress
        ACCESS  read-write
        STATUS  deprecated
        ::= { atEntry 3 }


-- the IP group

ipForwarding OBJECT-TYPE
        SYNTAX  INTEGER {
                    gateway(1), -- entity forwards
                                -- datagrams
                    host(2)     -- entity does NOT
                                -- forward datagrams
                }
        ACCESS  read-write
        STATUS  mandatory
        ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ip 2 }

ipInReceives OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 7 }

ipInDiscards OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 8 }

ipInDelivers OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 9 }

ipOutRequests OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 10 }

ipOutDiscards OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 12 }

ipReasmTimeout OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 13 }

ipReasmReqds OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
```

```
        STATUS  mandatory
        ::= { ip 14 }

ipReasmOKs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 15 }

ipReasmFails OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 16 }

ipFragOKs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 17 }

ipFragFails OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 18 }

ipFragCreates OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 19 }

-- the IP Interface table

ipAddrTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IpAddrEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
        SYNTAX  IpAddrEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrTable 1 }

IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        IpAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        IpAddress,
    ipAdEntBcastAddr
        INTEGER,
    ipAdEntReasmMaxSize
        INTEGER (0..65535)
}

ipAdEntAddr OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 1 }

ipAdEntIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 2 }

ipAdEntNetMask OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 3 }

ipAdEntBcastAddr OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 4 }

ipAdEntReasmMaxSiz OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 5 }

-- the IP Routing table

ipRoutingTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IpRouteEntry
        ACCESS  read-write
        STATUS  mandatory
        ::= { ip 21 }

ipRouteEntry OBJECT-TYPE
        SYNTAX  IpRouteEntry
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRoutingTable 1 }

IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
        INTEGER,
    ipRouteMetric2
        INTEGER,
    ipRouteMetric3
        INTEGER,
    ipRouteMetric4
```

```
        INTEGER,
    ipRouteNextHop
        IpAddress,
    ipRouteType
        INTEGER,
    ipRouteProto
        INTEGER,
    ipRouteAge
        INTEGER,
    ipRouteMask
        IpAddress
}

ipRouteDest OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 1 }

ipRouteIfIndex  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 2 }

ipRouteMetric1 OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 3 }

ipRouteMetric2 OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 4 }

ipRouteMetric3 OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 5 }

ipRouteMetric4 OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 6 }

ipRouteNextHop OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 7 }

ipRouteType OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),    -- none of the following
                    invalid(2),  -- an invalidated route

                                 -- route to directly
                    direct(3),   -- connected
                                 -- (sub-)network

                                 -- route to a non-local
                    remote(4)    -- host/network/
                                 -- sub-network
                }
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 8 }

ipRouteProto OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),    -- none of the following

                                 -- non-protocol
                                 -- information

                                 --   e.g., manually
                    local(2),    --   configured entries

                                 -- set via a network
                    netmgmt(3),  --   management protocol

                                 -- obtained via ICMP,
                    icmp(4),     --   e.g., Redirect

                                 -- the following are
                                 -- gateway routing
                                 -- protocols
                    egp(5),
                    ggp(6),
                    hello(7),
                    rip(8),
                    is-is(9),
                    es-is(10),
                    ciscoIgrp(11),
                    bbnSpfIgp(12),
                    ospf(13)
                    bgp(14)
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipRouteEntry 9 }

ipRouteAge OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipRouteEntry 10 }

ipRouteMask OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-write
```

```
        STATUS  mandatory
        ::= { ipRouteEntry 11 }

-- the IP Address Translation tables

ipNetToMediaTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IpNetToMediaEntry
        ACCESS  read-write
        STATUS  mandatory
        ::= { ip 22 }

ipNetToMediaEntry OBJECT-TYPE
        SYNTAX  IpNetToMediaEntry
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipNetToMediaTable 1 }

IpNetToMediaEntry ::= SEQUENCE {
    ipNetToMediaIfIndex
        INTEGER,
    ipNetToMediaPhysAddress
        OCTET STRING,
    ipNetToMediaNetAddress
        IpAddress,
    ipNetoToMediaType
        INTEGER
}

ipNetToMediaIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),   -- none of the following

                    invalid(2), -- an invalidated mapping
                    dynamic(3), -- connected (sub-)network

                    static(4)
                }
        ACCESS  read-write
        STATUS  mandatory
        ::= { ipNetToMediaEntry 4 }
```

```
-- the ICMP group

icmpInMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 4 }

icmpInParmProbs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 9 }
```

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

icmpInTimestampReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 23 }

icmpOutTimestampReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 24 }

icmpOutAddrMasks OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 25 }

icmpOutAddrMaskReps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { icmp 26 }


-- the TCP group

tcpRtoAlgorithm OBJECT-TYPE
        SYNTAX  INTEGER {
                    other(1),    -- none of the following
                    constant(2), -- a constant rto
                    rsre(3),     -- MIL-STD-1778,
                    vanj(4)      -- Van Jacobson's
                                 -- algorithm
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 1 }
```

```
tcpRtoMin OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 2 }

tcpRtoMax OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 3 }

tcpMaxConn OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 4 }

tcpActiveOpens OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 6 }

tcpAttemptFails OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
        SYNTAX  Gauge
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 9 }

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

tcpOutSegs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 12 }

-- the TCP connections table

tcpConnTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF TcpConnEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
        SYNTAX  TcpConnEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcpConnTable 1 }

TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}

tcpConnState OBJECT-TYPE
        SYNTAX  INTEGER {
                    closed(1),
                    listen(2),
                    synSent(3),
                    synReceived(4),
                    established(5),
                    finWait1(6),
                    finWait2(7),
                    closeWait(8),
                    lastAck(9),
                    closing(10),
                    timeWait(11)
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
```

```
        ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcpConnEntry 5 }

-- additional TCP variables

tcpInErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 14 }

tcpOutRsts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { tcp 15 }


-- the UDP group

udpInDatagrams OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { udp 1 }

udpNoPorts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { udp 2 }

udpInErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { udp 3 }

udpOutDatagrams OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { udp 4 }

-- the UDP listener table

udpTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF UdpEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { udp 5 }

udpEntry OBJECT-TYPE
        SYNTAX  UdpEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { udpTable 1 }

UdpEntry ::= SEQUENCE {
    udpLocalAddress
        IpAddress,
    udpLocalPort
        INTEGER (0..65535)
}

udpLocalAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { udpEntry 1 }

udpLocalPort OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        ::= { udpEntry 2 }


-- the EGP group

egpInMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 1 }

egpInErrors OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 2 }

egpOutMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 3 }

egpOutErrors OBJECT-TYPE
```

```
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 4 }

-- the EGP Neighbor table

egpNeighTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF EgpNeighEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 5 }

egpNeighEntry OBJECT-TYPE
        SYNTAX  EgpNeighEntry
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighTable 1 }

EgpNeighEntry ::= SEQUENCE {
    egpNeighState
        INTEGER,
    egpNeighAddr
        IpAddress,
    egpNeighAs
        INTEGER,
    egpNeighInMsgs
        Counter,
    egpNeighInErrs
        Counter,
    egpNeighOutMsgs
        Counter,
    egpNeighOutErrs
        Counter,
    egpNeighInErrMsgs
        Counter,
    egpNeighOutErrMsgs
        Counter,
    egpNeighStateUps
        Counter,
    egpNeighStateDowns
        Counter,
    egpNeighIntervalHello
        INTEGER,
    egpNeighIntervalPoll
        INTEGER,
    egpNeighMode
        INTEGER,
    egpNeighEventTrigger
        INTEGER
}

egpNeighState OBJECT-TYPE
        SYNTAX  INTEGER {
                    idle(1),
                    acquisition(2),
                    down(3),
                    up(4),
                    cease(5)
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 1 }

egpNeighAddr OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 2 }

egpNeighAs OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 3 }

egpNeighInMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 4 }

egpNeighInErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 5 }

egpNeighOutMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 6 }

egpNeighOutErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 7 }

egpNeighInErrMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 8 }

egpNeighOutErrMsgs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 9 }

egpNeighStateUps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
```

```
        ::= { egpNeighEntry 10 }

egpNeighStateDowns OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 11 }

egpNeighIntervalHello OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 12 }

egpNeighIntervalPoll OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 13 }

egpNeighMode OBJECT-TYPE
        SYNTAX  INTEGER {
                    active(1),
                    passive(2)
                }
        ACCESS  read-only
        STATUS  mandatory
        ::= { egpNeighEntry 14 }

egpNeighEventTrigger OBJECT-TYPE
        SYNTAX  INTEGER {
                    start(1),
                    stop(2)
                }
        ACCESS  read-write
        STATUS  mandatory
        ::= { egpNeighEntry 15 }

-- additional EGP variables

egpAs OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { egp 6 }


-- the Transmission group (empty at present)

-- the SNMP group

snmpInPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 1 }

snmpOutPkts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 2 }

snmpInBadVersions OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 3 }

snmpInBadCommunityNames OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 4 }

snmpInBadCommunityUses OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 5 }

snmpInASNParseErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 6 }

snmpInBadTypes OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 7 }

snmpInTooBigs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 8 }

snmpInNoSuchNames OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 9 }

snmpInBadValues OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 10 }

snmpInReadOnlys OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 11 }
```

```
snmpInGenErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 12 }

snmpInTotalReqVars OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 13 }

snmpInTotalSetVars OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 14 }

snmpInGetRequests OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 15 }

snmpInGetNexts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 16 }

snmpInSetRequests OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 17 }

snmpInGetResponses OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 18 }

snmpInTraps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 19 }

snmpOutTooBigs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 20 }

snmpOutNoSuchNames OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 21 }

snmpOutBadValues OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 22 }

snmpOutReadOnlys OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 23 }

snmpOutGenErrs OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 24 }

snmpOutGetRequests OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 25 }

snmpOutGetNexts OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 26 }

snmpOutSetRequests OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 27 }

snmpOutGetResponses OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 28 }

snmpOutTraps OBJECT-TYPE
        SYNTAX  Counter
        ACCESS  read-only
        STATUS  mandatory
        ::= { snmp 29 }

snmpEnableAuthTraps OBJECT-TYPE
        SYNTAX  INTEGER {
                    enabled(1),
                    disabled(2)
                }
        ACCESS  read-write
        STATUS  mandatory
```

```
        ::= { snmp 30 }

END
```

APPENDIX C : 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 sometimes ignored
             VarBindList
       }

Trap-PDU ::=
   [4]
      IMPLICIT SEQUENCE {
          enterprise     -- type of object generating
                         -- trap, see sysObjectID

             OBJECT IDENTIFIER,

          agent-addr           -- address of object
             NetworkAddress,   -- generating 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
```

BIBLIOGRAPHY

[COME91] Douglas E. COMER, INTERNETWORKING WITH TCP/IP: VOLUME 1; PRINCIPLES, PROTOCOLS, AND ARCHITECTURE, second edition, Prentice Hall, Englewood Cliffs, New Jersey, 1991

[DAIS88] Jean-Paul DAISOMONT, REFLEXION SUR LE MODELE TCP/IP, COMPARAISON ET ANALYSE D'IMPLEMENTATION, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, NAMUR, 1987-1988

[DALL88] I.N. DALLAS, "Operational Experiences in Managing a LAN/WAN Gateway", ISSUES IN LAN MANAGEMENT, Dallas/Spratt editors, IFIP, 1988, pp.95-113

[FISH91] Sharon FISHER, "Dueling Protocols, Will SNMP win out over CMIP, of vice versa?", BYTE, March 1991, pp.183-190.

[GAMB90] Dr John N. GAMBLE, Experience at CERN in the Management of Large-Scale Multivendor LANs, CERN Computing and Networks Division, CERN/CN/90, June 5, 1990

[HALL88] Jane HALL, Robbert van RENESSE, Hans van STAVEREN, "Gateways and Management in an Internet Environment", ISSUES IN LAN MANAGEMENT, Dallas/Spratt editors, IFIP, 1988, pp.77-94

[ISO7498-4] ISO/IEC 7498-4, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework, International Standards Organization, November 15, 1989

[ISO8649] ISO/IEC 8649, Information Processing Systems - Open Systems Interconnection, Service Definition for Association Control Service Element, International Standards Organization

[ISO8650] ISO/IEC 8650, Information Processing Systems - Open Systems Interconnection, Protocol Specification for Association Control Service Element, International Standards Organization

[ISO8824] ISO/IEC 8824, Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Standards Organization, May, 1987

[ISO8825] ISO/IEC 8825, Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), International Standards Organization, May, 1987

[ISO9072-1] ISO/IEC 9072-1, Information Processing Systems - Open Systems Interconnection, Text Communication - Message-Oriented Test Interchange System (MOTIS) - Remote Operations, Part 1: Model, Notation and Service Definition, International Standards Organization, 1988

[ISO9072-2] ISO/IEC 9072-2, Information Processing Systems - Open Systems Interconnection, Text Communication - Message-Oriented Test Interchange System (MOTIS) - Remote Operations, Part 2: Protocol Specification, International Standards Organization, 1988

[ISO9595] ISO/IEC 9595, Information Processing Systems - Open Systems Interconnection, Management Information Service Definition - Part 2: Common Management Information Service, International Standards Organization, December 22, 1988

[ISO9596] ISO/IEC 9596, Information Processing Systems - Open Systems Interconnection, Management Information Protocol Definition - Part 2: Common Management Information Protocol, International Standards Organization, December 22, 1988

[ISO10165-1] ISO/IEC 10165-1, Information Processing Systems - Open Systems Interconnection, Structure of Management Information, Part 1: Management Information Model, International Standards Organization

[JOSE88] Celia JOSEPH, Kurudi H. MURALIDHAR, "Network Management: A Manager's Perspective", ENTERPRISE Conference Proceedings, Society of Manufacturing Engineers, 1988

[KLER88] S. Mark KLERER, "The OSI Management Architecture: an Overview", IEEE NETWORK, Vol.2 N°2, March, 1988, pp.20-29

[MEUN88] J.M. MEUNIER, "An Interactive Network Display System for Network Management", IEEE 1988 Network Operations and Management Symposium, New Orleans, LA, February 28-March 2, 1988, pp.1.18-18.18

[MINO89a] Daniel MINOLI, "Evolving Security Management Standards", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, 1989, pp.NM20_500_101-107

[MINO89b] Daniel MINOLI, "Managing Local Area Networks : Fault and Configuration Management", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, 1989, pp.NM50_300_401-411

[MINO89c] Daniel MINOLI, "Network Management Functions: Telecommunications Hardware", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, October, 1989, pp.NM20_100_101-105

[MINO89d] Daniel MINOLI, "Managing Local Area Networks: Accounting, Performance and Security Management", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, June 1989, pp.NM50_300_501-508

[MORR89] Wayne MORRISON, "Ethernet LAN Management NMCC/VAX ETHERnim, A Case Study", Proceedings of the IFIP TC6/WG6.6 Symposium on Integrated Network Management, Boston, ELSEVIER SCIENCE PUBLISHERS, May 15-17, 1989

[NM20-300-1] "Inventory and Configuration Management", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, June 1989, pp.NM20_300_101-104

[NM50-600-1] "Modem/Multiplexer-Based Network Management", DATAPRO NETWORK MANAGEMENT/DATAPRO RESEARCH, 1990 McGRAW-HILL, June 1989, pp.NM50_600_101-106

[RFC768] User Datagram Protocol, Request For Comments 768, J. Postel, DDN Network Information Center, SRI International, August 28, 1980

[RFC791] Internet Protocol, Request For Comments 791, DDN Network Information Center, SRI International, September, 1981

[RFC792] Internet Control Message Protocol, Request For Comments 792, DDN Network Information Center, SRI International, 1981

[RFC793] Transmission Control Protocol, Request For Comments 793, DDN Network Information Center, SRI International, September 1981

[RFC1085] ISO Presentation Services on top of TCP/IP-based internets, Marshall T. ROSE, Request For Comments 1085, DDN Network Information Center, SRI International, December, 1988

[RFC1095] The Common Management Information Services and Protocol over TCP/IP (CMOT), Request for Comments 1095, Unnikrishnan S. WARRIER, Larry BESAW, DDN Network Information Center, SRI International, April, 1989

[RFC1155] Structure and Identification of Management Information for TCP/IP based internets, Request for Comments 1155, Marshall T. ROSE, Keith McCloghrie, DDN Network Information Center, SRI International, May, 1990

[RFC1156] Management Information Base for Network Management of TCP/IP based internets, Request for Comments 1156, Marshall T. ROSE, Keith McCloghrie, DDN Network Information Center, SRI International, May, 1990

[RFC1157] A simple Network Management Protocol, Request for Comments 1157, Jeffrey D. CASE, Mark S. FEDOR, Martin L. SCHOFFSTALL, James R. DAVIN, DDN Network Information Center, SRI International, May 1990

[RFC1158] Management Information Base for Network Management of TCP/IP based internets: MIB-II, Request for Comments 1158, Marshall T. ROSE, DDN Network Information Center, SRI International, May, 1990

[ROSE91] Marshall T. ROSE, THE SIMPLE BOOK: AN INTRODUCTION TO MANAGEMENT OF TCP/IP-BASED INTERNETS, Prentice Hall, Englewood Cliffs, New Jersey, 1991

[SLOM88] Morris SLOMAN, "Distributed Systems Management", ISSUES IN LAN MANAGEMENT, Dallas/Spratt editors, IFIP, 1988, pp. 15-46

[STAL89] William STALLINGS, HANDBOOK OF COMPUTER-COMMUNICATIONS STANDARDS : THE TCP/IP PROTOCOL SUITE (VOLUME 3), Second Edition, Howard W. Sams & Company, 1989

[TANE88] Adrew S. TANENBAUM, Computer Networks, Prentice Hall Software Series, Prentice-Hall, Englewood Cliffs, New Jersey, 1988

[WARR90] Unni WARRIER, Amatzia BEN-ARTZI, Asheem CHANDNA, "Network Management of TCP/IP Networks: Present and Future", IEEE NETWORK MAGAZINE, July, 1990, pp. 35-42

GLOSSARY

ACK	:	Acknowledgement segment
ARP	:	Address Resolution Protocol
ASN.1	:	Abstract Syntax Notation 1
BER	:	Basic Encoding Rules
CCITT	:	Comité Consultatif International de la Téléphonie et de la Télégraphie
CMIP	:	Common Management Information Protocol
CMIS	:	Common Management Information Service
CMOT	:	CMIP over TCP/IP
CPU	:	Central Processing Unit
DBMS	:	DataBase Management System
DEC	:	Digital Equipment Company
DF	:	Don't Fragment flag
DNS	:	Domain Name Server
DoD	:	Department of Defense
EGP	:	Exterior Gateway Protocol
FIN	:	Finish segment
FTAM	:	File Transfer, Access and Management
FTP	:	File Transfer Protocol
GGP	:	Gateway-to-Gateway Protocol
IAB	:	Internet Activities Board
ICMP	:	Internet Control Message Protocol
IHL	:	Internet Header Length
INMS	:	Integrated Network Management System
IP	:	Internet Protocol
ISO	:	International Standards Organisation
LAN	:	Local Area Network

GLOSSARY (END)

MF	: More Fragment flag
MIB	: Management Information Base
MTU	: Maximum Transmission Unit
NFS	: Network File System
NSAP	: Network Service Access Point
OSI	: Open System Interconnection
OTDR	: Optical Time Domain Reflectometer
PDU	: Protocol Data Unit
PING	: Packet INternet Groper
RARP	: Reverse Address Resolution Protocol
RFC	: Requests For Comments
RST	: Reset
SAP	: Service Access Point
SMI	: Structure of Management Information
SMTP	: Simple Mail Transfer Protocol
SNA	: Systems Network Architecture (IBM)
SNMP	: Simple Network Management Protocol
SQL	: Structured Query Language
SYN	: Synchronize segment
TCB	: Transmission Control Block
TCP	: Transmission Control Protocol
TDR	: Time Domain Reflectometer
TFTP	: Trivial File Transfer Protocol
TSAP	: Transport Service Access Point
TTL	: Time To Live field
UDP	: User Datagram Protocol
WAN	: Wide Area Network

Réalisation, impression et reliure : SPRL Centre de Reprographie - Tél. : 081/22.87.72