THE TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
THE FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY

THE MANAGEMENT INFRASTRUCTURE OF A NETWORK MEASUREMENT SYSTEM

Author: **Alexandru Iosif Bikfalvi**

Supervisors: **Dr. Virgil Dobrota, Dr. Jordi Domingo-Pascual**

2006

ABSTRACT

The goal of this project was to develop a C/C++ management console application for GNU/Linux platforms that enables one to perform QoS measurement sessions by using a set of distributed measurement agents within a SNMP management framework. The console features a graphical user interface and a group of services that handle the management information. There are services used to communicate with the computer's socket interface, to perform SNMP encapsulation and decoding, a measurement session manager that has the intelligence of interpreting the measurement results.

A queuing service solves the issue of asynchronous communication, by implementing a set of eight (half-inbound and half-outbound) message-waiting queues. Four priority levels exist complemented by a Round-Robin servicing policy to ensure that management messages are preferentially handled in the following order: notifications, control messages, results request and results replies.

The advantages of the proposed measurement solution versus the existing tools are the possibility of managing many test scenarios through the control of a large set of agents, no user attendance required during the experiments, customizable sessions and availability of results (numeric or plotted) both during and after the measurement is completed.

CONTENTS

Abstract 1
Contents 2
State of the Art 5
1.1 Overview 5
1.1.1 Why Need for a Network Measurement System? 5
1.2 Testing Network Interface Cards 8
1.2.1 Framework 8
1.2.2 Disadvantage of the Approach 10
Theoretical Fundamentals 15
2.1 Overview of QoS Measurement 15
2.1.1 Principle of QoS 15
2.1.2 Providing Quality-of-Service 16
2.1.3 DiffServ QoS 16
2.1.4 IntServ QoS 19
2.1.5 IP Measurements Framework 20
2.1.6 Connectivity 21
2.1.7 Delay 22
2.1.8 Packet Loss 22
2.1.9 QoS Software Tools 23
2.2 Overview of Network Management 25
2.2.1 What is Network Management? 25
2.2.2 ISO Management Functional Areas 26
2.2.3 Perspectives of Management 28
2.2.3.1 Management in Telecom Industry 29
2.2.3.2 Management in Datacom Industry 29
2.2.3.3 Management in Computer Industry 30
2.2.3.4 Multi-service Management 30
2.2.4 Management Architecture 31
2.2.4.1 Managers 31
2.2.4.2 Agents 32
2.2.4.3 Manager – Agent Communication 32
2.2.5 Management Information 34
2.3 Simple Network Management Protocol 36
2.3.1 Overview 36
2.3.1.1 IETF Standardization 36
2.3.1.2 Versions of SNMP 37
2.3.2 Basic Components 38
2.3.2.1 Community Names 38
2.3.2.2 Manager Configuration 39
2.3.2.3 Agent Configuration 40
2.3.3 Management Information 41
2.3.3.1 Management Information Base 41
2.3.3.2 Managed Objects 42
2.3.4 Structure of Management Information 44
2.3.4.1 SMI Data Types 44
2.3.5 Protocol Specifications 45

2.3.5.1 Protocol Operation 46
2.3.5.2 SNMPv1 Message Formats 47
2.3.5.3 SNMPv2 Message Formats 49
2.3.5.4 Security in SNMPv1 and SNMPv2 50
2.3.5.5 Interoperability SNMPv1 – SNMPv2 51
2.3.5.6 SNMP Version 3 Framework 51
2.3.5.7 SNMP Version 3 Entities 52
2.3.5.8 SNMP Version 3 Message Format 54
2.3.5.9 SNMP Version 3 Security 55
Design and Experimental Results 56
3.1 Network Measurement System Foundation 56
3.1.1 System Architecture 56
3.1.2 Features of the Network Measurement System 58
3.1.3 Other Features of the Management Infrastructure 60
3.1.4 The Administrative Management Console 65
3.1.5 Application Files and Setup Requirements 67
3.2 The Architecture of the Management Infrastructure 71
3.2.1 Generalities 71
3.2.2 Internal Structure 71
3.2.3 The Layered Architecture 75
3.3 The Management Service 78
3.3.1 Service Objectives 78
3.3.2 Service Structure 78
3.3.3 Service Control Thread 79
3.3.4 Socket Control Thread 87
3.3.5 Socket Data Thread 90
3.3.6 Management Service Summary 95
3.4 The SNMP Service 98
3.4.1 Service Objectives 98
3.4.2 Encoding SNMP Messages 98
3.4.3 Encoding SNMP Traps 105
3.4.4 Decoding SNMP Messages 106
3.4.5 Decoding SNMP Traps 111
3.4.6 Managed Objects Data and Usage Requirements 111
3.4.7 SNMP Summary 115
3.5 The Queuing Service 116
3.5.1 Service Objectives 116
3.5.2 Implementing the Message Queue 117
3.5.3 Implementing Message Retransmission 126
3.5.4 Discarding Inbound Duplicate Messages 128
3.5.5 Implementing Priorities 129
3.5.6 Environmental Variables and Summary 130
3.6 The Session Manager 131
3.6.1 Purpose and Objectives 131
3.6.2 Sessions and Session Groups 131
3.6.3 Traffic Generation Sessions 135
3.6.4 Traffic Analysis Sessions 138
3.6.5 Traffic Generation and Analysis Sessions 138
3.6.6 Analyzing Traffic with Endace Cards 140
3.6.7 Implementing Sessions, Groups and Tasks 141
3.6.8 Implementing the Session Manager 144
3.6.8.1 Starting the Session Based Tasks 147

3.6.8.2 Running the Session Based Tasks 149
3.6.8.3 Starting and Running Session Groups 154
3.6.9 The SNMP Wrapper 155
3.6.9.1 The Connection Class 156
3.6.9.2 The System Class 157
3.6.9.3 The Traffic Class 159
3.6.9.4 The Advanced Analysis Class 161
3.6.10 Session Results 163
3.7 Other Services 166
3.7.1 The Local Message Dispatcher 166
3.7.2 The Service Control Manager 167
3.7.3 The Configuration Service 167
3.7.4 The Hardware Manager 168
3.8 Using the Management Console 170
3.8.1 Console Setup 170
3.8.2 Configuring Services 173
3.8.3 Registering Agents 174
3.8.4 Creating Measurement Sessions and Groups 176
3.8.5 Scheduling Tasks and Collecting Results 177
3.8.6 Using the Event Log 181
Conclusions 182
4.1 Measurement Scenario Recreated 182
4.2 Advantages and Disadvantages of the Proposed Solution 187
4.3 Future Work 188
References 189

1

STATE OF THE ART

1.1 OVERVIEW

This chapter describes the reasons that led to the primary objective of this project: developing a management platform that is integrated along with a set of distributed measurement agents within unitary QoS measurement software – the Network Measurement System, or NMS for short. It is presented what is the disadvantage of existent measurement tools, both hardware and software and what is required to improve performance, easiness of use and availability of results.

The goal of this project is to present half of the technologies that make up the Network Measurement System, i.e. the half related to management, processing and presentation of results.

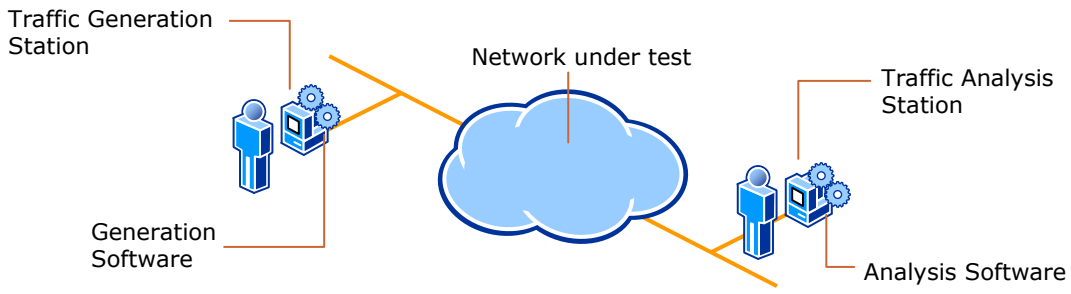
1.1.1 WHY NEED FOR A NETWORK MEASUREMENT SYSTEM?

The main goal of the Network Measurement System software was to overcome some of the limitations of regular measurement applications available today. These limitations refer to the fact that some of the well-known and most used tools do not cope well with complex, repetitive task, especially from the results processing point-of-view.

For example, in the next chapter *MGEN* will be described as one of the most used free traffic generating applications. However, the use of *MGEN* in a real test scenario poses many problems especially when attempting to perform the same operation several times. Even though you can program *MGEN* with the scripting file presented earlier, the problem occurs when you want to collect the results, because the final data usually needs additional, time-consuming post-processing, in order to make it usable.

The figure 1.1 shows a typical scenario that involves the usage of *MGEN*.

Figure 1.1 Scenario of a measurement experiment using *MGEN*



In the scenario presented before in order to perform a measurement task, you had to:

Install *MGEN* on the machine that generated the traffic for the active measurement test.

Install a proper analysis tool, such as *DREC* on the machine analyzing the traffic.

Create a script to instruct *MGEN* what operations to perform.

Running the test and awaiting for results at the analyzing station.

Collecting the results data and processing it according to needs.

Notes

- The *MGEN* traffic generation tool can be used by specifying the traffic options at console as command line parameters, instead of using a script file. Therefore, the step 3 in the above enumeration is optional.

From the steps presented above only one, namely the last one raises difficulties when performing a particular measurement task. Let us say that the objective is to measure the link behavior (in particular the packet one-way delay and packet delay variation) for a set of values for packets per second parameter. The simplest way to do this using *MGEN* is to define a script that successively tells the program to start generating traffic with an increasing number of packets per second. Therefore, after this step is completed the traffic is generated automatically without the user's attention.

At the receiving part a software analysis tool such as *MGEN*, *DREC* or *tcpdump* records the information about the incoming traffic. In theory this can be done, either by creating a dump of the incoming traffic and save it for processing after the capturing session is complete, or computing the required parameters in real time. The first approach has the advantage of having enough data after the dump is completed to compute offline a large number of parameters. Theoretically, every destination-dependent on each packet that successfully arrived is known. The disadvantage of the dump method is that it does not cope well with traffic analysis on high bit-rate flows, since the dump speed is usually limited by the performance of the hard-disk unit.

The alternative is, of course, to compute in real time, as packets arrive, the parameters of interest and to discard all non-essential information. It will take much processing time but it can be successfully used for high-rates experiments, where the dump solution is not a viable choice(in the case of regular IDE hard drives, the access rate rarely exceeds 50 Mbytes per second corresponding to a bit rate of only 400 Mbps).

The tools such as *MGEN* (or *DREC* for older versions) perform logging of incoming traffic allowing both online and offline analysis. They are flow oriented, meaning that you can specify several different flows running simultaneously, and the program is able to make the difference between the packets belonging to different flows.

Tcpdump however does not have almost any measurement capabilities at all. It simply can be used to log or dump all incoming traffic on a given network interface (some filtering options are though available) and later, after the online dumping operation has been completed a third party software can be used to extract the necessary information.

Notes

- As stated before, even though *MGEN* is an application oriented on traffic generation, since version 4.0 it contains traffic analysis functionality as well, by logging incoming traffic and allowing real time analysis and plotting. For *MGEN* version 3.x family, the functionality was split between the traffic generating application, namely *MGEN* and the analysis tool *DREC*.

Nevertheless, the overall scenario has some main disadvantages that cannot be overcome by changing the generating software, the analyzer software or tuning their parameters:

- The most important one is the lack of centralized control of the applications both end-points. You cannot sit at single machine using a single software interface and performing all the measurements you need.
- Second, the data availability for a single flow using available analysis software is very good. Nevertheless, what can be done about combing the data from different successive flows? In a hypothetical experiment where you want to measure the average throughput over a test of 60 seconds length versus the packet rate in series of 10 consecutive tests, the number of tools that can do this job is very scarce. At most, you end up with a pile of raw data needs to be processed manually, consuming time and effort that could be better redirected elsewhere.
- At last, there are possible scenarios in which the usage of the available tools is either impossible or it needs supplementary tricks to make it work. Such a scenario is to be presented in the following paragraph.

1.2 TESTING NETWORK INTERFACE CARDS

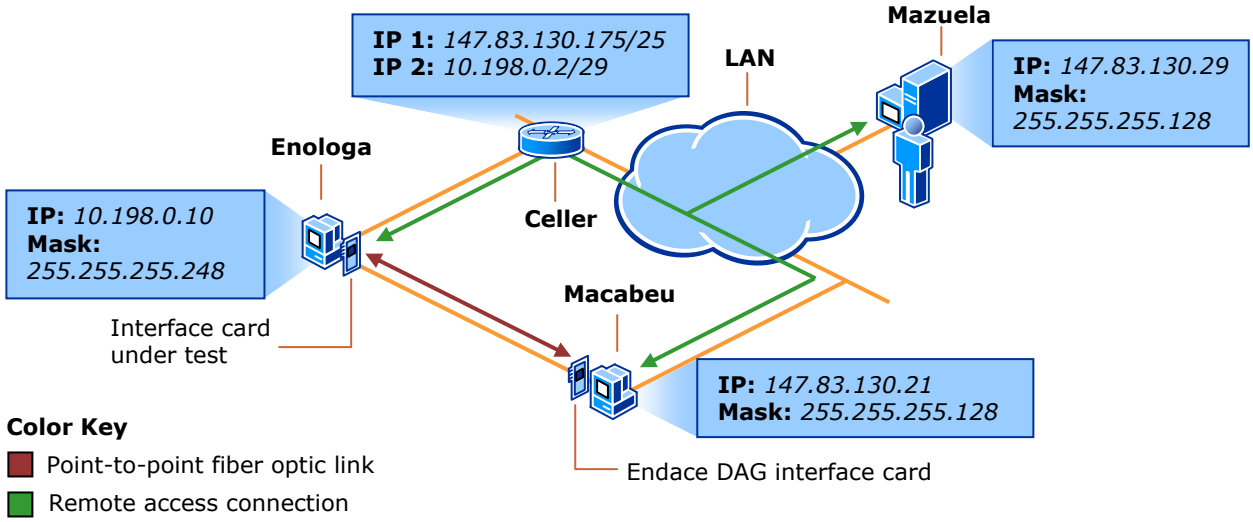
1.2.1 FRAMEWORK

The starting point of the development of the Network Measurement System was a scenario that involved the testing of several ordinary Gigabit Ethernet/IEEE 802.3 network interface cards (NICs) using a dedicated measurement card manufactured by the New Zeeland company, Endace Ltd.

The card and its features were thoroughly presented in the previous chapter. However, as a short reminder, the card is able to generated and analyze in hardware various types of Ethernet/IEEE 802.3 traffic. It has its own specific toolset provided by the manufacturer. The objective of the original experiment was to determine the working-limit and performance for a set of regular Gigabit Ethernet NICs. Since processing of the Endace DAG card is entirely hardware based, the experiment intended to measure various traffic parameters, total throughput, delay, delay variation, packet loss ratio and out of order packets being some of them.

The original experiment setup is presented in figure 1.2.

Figure 1.2 Experiment setup for testing network interface cards



The experiment involved three machines, namely *Mazuela*, *Enologa* and *Macabeu*. A brief description of each one is presented in the following table.

Table 1.1 Devices used in the network interface tests

Device	Interfaces	Description
Macabeu	Endace DAG 4.3 GE Card, used to analyze traffic	This computer was used to analyze the traffic generated by the NIC under test installed on computer *Enologa*.
	100 Base TX Mbps Ethernet NIC (IP Address: 147.83.130.21/25) used for remote connection management	
Enologa	Intel 1000 Base SX Ethernet NIC – the interface card under test	This computer was used to generate the traffic for the active measurement test.
	100 Base TX Ethernet NIC (IP Address: 10.198.0.10/29)	

Mazuela	100 Base TX Ethernet NIC (IP Address: 147.83.130.29/25) used for remote connection management	This computer was used to connect remotely via *ssh* to *Enologa* and *Macabeu* computers.
Celler	100 Base TX Ethernet NIC (IP Address: 147.83.130.175/25)	This computer acted as a NAT server, in order to connect to computer *Enologa* that is installed in the 10.198.0.0 network from computer *Mazuela*.
	100 Base TX Ethernet NIC (IP Address: 10.198.0.2/29)	

Some explanations about the test topology are required:

- The access to computer *Enologa* was made via a NAT server (that also acted as a router). This was due to the constraints to have two machine equipped with a PCI-X bus required by the 1000 Base SX cards used for testing. The first machine was *Macabeu* used as host for the DAG interface. The second machine was *Enologa* that was already installed in a second private network, and therefore the reasons of selecting such a topology depended only on the hardware availability.
- Even it is not shown in the figure; both networks (i.e. 10.198.0.0/29 and 147.83.130.0/25) were constituted as two VLANs, connected physically to a single Cisco switch.
- All involved machines run Debian GNI/Linux, kernel version 2.6.15-1.486 or 2.6.8-1-386. The remote access was made via *ssh*.

The test procedure implied the following steps:

1. Connect from *Mazuela* to *Celler* using a *ssh* remote connection.

```
mazuela:~# ssh 147.83.130.175
login as: root
Using keyboard-interactive authentication.
Password:
Last login: Mon May 15 10:54:40 2006
celler:~#
```

2. From *Celler* console, connect via *ssh* to *Mazuela*.

```
celler:~# ssh 10.198.0.10
Password:
Last login: Sat May 13 17:31:11 2006
Linux enologa 2.6.8-1-386 #1 Thu Nov 11 12:18:43 EST 2004 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

3. On *Enologa*, create the script required by *MGEN* in order to generate traffic.

```
0.0 ON 1 UDP DST 172.16.0.2/5000 PERIODIC [1000.0 512]
15.0 OFF 1
```

Notes

- This example calls for a generation of UDP packets for 15 seconds (from time index 0.0 to time index 15.0) to destination 172.16.0.2 port 5000 following a periodic distribution with 1000 datagrams per second, 512 bytes of payload.

4. Connect from *Mazuela* to *Macabeu* using *ssh*.

```
mazuela:~# ssh 147.83.130.21
login as: root
root@147.83.130.21's password:
Last login: Mon May 15 14:38:48 2006 from dhcp3.ccaba.upc.edu
Linux macabeu 2.6.15-1-486 #1 Tue Feb 21 20:16:13 UTC 2006 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
macabeu:~#
```

5. At the remote connection to *Macabeu*, start a data capturing session using the DAG interface card.

```
dagsnap –v –d dag0 –o capture.erf –s 20
```

6. At the remote connection to *Enologa*, start the traffic generation with *MGEN*.

```
mgen input script.gen
```

Notes

- The previous command means that the card will dump all incoming traffic to the file *capture.erf*. The device to be used is DAG card number 0, the output during the execution of the command is in verbose mode and the capture session duration is 20 seconds (i.e. 5 seconds more than the generation to allow some slack). For more information about the functionality and the commands of the DAG card, review the previous chapter.

1.2.2 DISADVANTAGE OF THE APPROACH

Despite of the logic succession of these experimental steps the measurement session presented has some serious flaws and some overall drawbacks.

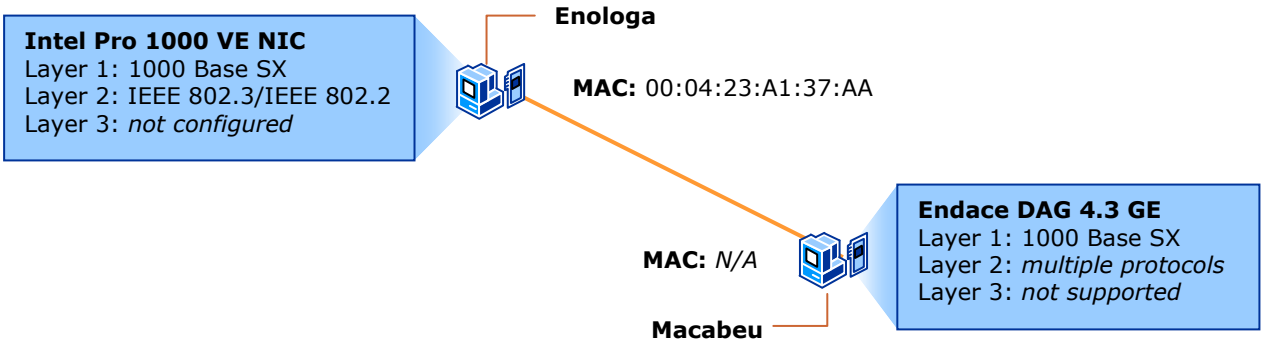
First, one may ask about the destination IP address in the *MGEN* script file (i.e. 172.16.0.2). That is a private address corresponding to the network created by the fiber optic link between the tested NIC and the DAG card. However, no such address was presented on the scenario from the figure 1.2. Second, from the description of the DAG card presented in the previous chapter, on knows the DAG card supports several data-link layer protocols such as Ethernet 2, IEEE

802.3/IEEE 802.2 and ATM. Nevertheless, the card and its software driver have no support for any network layer protocols such as IP.

The explanation is the DAG card actually captures any data-link traffic that is received at one of its ports and only if instructed so specifically by the user. Further, the DAG card does not generate any type of traffic without user's consents, so even if the card had supported a layer three protocol, like IP, it would never reply to an ARP request for example. It turns out that the DAG card does not actually support IP because it does not have to. In addition, because the DAG supports multiple data-link layer protocols it cannot be identified with an address belonging to a specific protocol. Briefly, the DAG card captures any bits it receives on the physical layer, interprets them as Ethernet or ATM traffic, depending on which protocol is configured and saves them into a ERF type file.

On the other hand, the available version of *MGEN* generates only UDP traffic, and therefore requires an IP address, both locally (assigned on the interface where the traffic is being generated) and as a destination address. Even if an IP address would have been assigned to the tested interface card, the implementation of TCP/IP stack in almost every operating system requires the data-link layer address of the next hop device (in this case of the destination device), before the IP packet is actually transmitted. Usually the hardware address resolution would have been the task of ARP, but in this situation, ARP is not available. The following figure depicts the status of the two interface cards involved in the test.

Figure 1.3 Test network interface cards original status

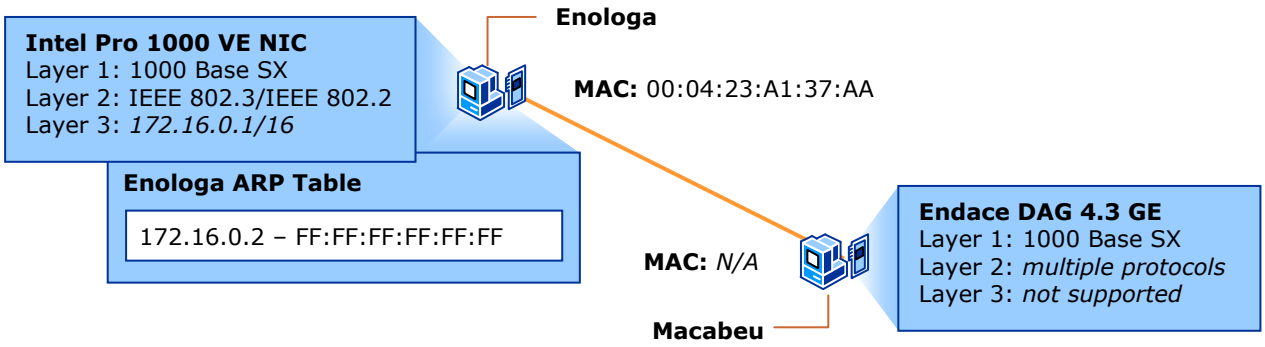


The solution to this problem was:

- To configure the Intel card on *Enologa* with an IP private address. This was required since *MGEN* is based on IP protocols and it cannot use interfaces that do not have IP addresses assigned. Since the first NIC already had in use a class A private IP address, the first IP address from the class B of private addresses was chosen for simplicity, i.e. 172.16.0.1 with the default subnet mask: 255.255.0.0.
- Statically modify the ARP table on *Enologa*, such that no ARP queries would be required by the TCP/IP stack in order to send an IP datagram. The destination of the packet was chosen as a hypothetical computer with an IP address in the same network segment (in this case 172.16.0.2/16) and having any hardware address (for simplicity the broadcast Ethernet address was selected). Remember, that when instructed so by the user, the DAG card would capture any packets that arrive on one of its interfaces, so the trick described here was merely intended to instruct the *Enologa* operating system to send the packets that contain the IP address 172.16.0.2 as destination to the connection of the Intel interface. Since the connection between *Enologa* and *Macabeu* is a point-to-point fiber optic link, the packets sent on the Intel will arrive at their

intended destination, regardless of their MAC and IP destination fields. The figure 1.4 outlines the concept.

Figure 1.4 Configuring IP stack on Enologa for use with MGEN and DAG card



With the changes described here, the experiment worked well, however this example was intended to show the challenges and the extra work required for such a simple measurement scenario. Actually, the problem presented here would never have existed if the generation software could have generated only Ethernet/IEEE 802.3 traffic without any network and transport layer encapsulation. Further more, the extra work performed by the TCP/IP stack when sending an UDP datagram implies more delays due to packet queuing in the software's buffers and processing latency, thus resulting in additional errors, if one wants, for example to test the performance of the NIC at layer 2 level.

One of the key features of the Network Management System is that gives you several options on the type of traffic to generate, thus allowing to test the performance at different layers of the networking implementation. What this means will be further presented on the paragraph entitled "Network Measurement System Features".

The second major drawback that should be outlined using the experimental setup presented here is about data collecting and processing. Since the traffic is generated using *MGEN*, while the captured data is analyzed using the software that comes with the DAG card, the parameter that could be at most measured using the DAG tools is the throughput. The *dagsnap* command presented before creates a dump of the received data, without computing any QoS related parameters. Therefore, additional software tools are required to analyze the available dumps and to extract the parameters of interest.

To make the things even worse, remember that when performing a dump, the speed of the dump is generally limited by the working speed of the storage device, and that for regular hard drives is well under the gigabit limit. The DAG hardware and software contain some features that allow for reasonable high speeds, such as for half a gigabit per second and beyond no loss of incoming data occurs due to the latency of the storage unit. However, the shortcoming of this approach can easily be seen:

- The dumping mechanism introduces additional delays, which are recorded as errors in the one-way delay and packet delay variation. Especially the packet delay variation has variations up to the hundreds of milliseconds due to this effect of saturation of the dump buffers. This is one of the major disadvantages of using the capturing features of the DAG card. A possible solution is to use an option that allows you to record only partial packets, for instance the first 64 bytes. However, with this approach

would not have been possible to make an accurate estimation of the throughput at the packet level.

- The overall effort required in configuring, performing, and collecting the data from such a test is increased, in part due to the addition of several extra steps, such as the set-up of an IP address and ARP table for the interface under test, when such action would not have been required.
- Then is the requirement of an extra piece of software able to analyze the dump created by the DAG card (the ERF file that contains an instance of each packet received with some extra information added about it). With this program (which already has been created) only an offline analysis is possible, thus having no real time data during the test, and extending the test duration up to several minutes after the test, especially at higher data rates for which the size of the dump files reached almost ten gigabytes for a 60 seconds test.
- At last, the test methodology presented here requires complete human attention at all times. Remember that in spite of additional configuration required, which is only a one time operation, at any point one would have had two simultaneously connections to *Enologa* and *Macabeu* computers (not to say to have two different persons handling those computers independently). Test by test you should start a capturing session on *Macabeu*, then starting a generation session on *Enologa*, awaiting at least 1 minute for the test to be completed and then run the custom software to analyze the dump created by the experiment. In the end, the data from the custom analysis software would have been collected by hand, thus that a session group with 15 measurement points, and two operators supervising the experiment, could last up to several hours.

Notes

- In the last item form the previous list, the number of measurement points refers to a group of tests, which have only the value of a parameter as a difference between them. For example, one of the objectives was to determine the QoS parameter for different packet rates. Hence, a session group with 15 measurement points means 15 consecutive experiments, in which the packet rate is modified (by hand) and the result is to obtain the dependency of these parameters versus the packet rate.

The idea of a Network Measurement System came actually from the necessity of improving the analysis application for the DAG card. In its original design, this application was accessible in command line only (to make it accessible over a *ssh* connection) and therefore could display results only in text like format. A regular output of the application is presented below.

```
macabeu:~/tests# ./erf cap.erf verbose noshow
Reading file: cap.erf

Statistics
Total captured packets: 7681

Average delay variation: -0.00000138022005558014 s
Maximum delay variation: 0.01013439893722534180 s
Minimum delay variation: -0.01006840169429779053 s

Average throughput: 94208.4790287216 bps
Maximum throughput: 205800516.2666666667 bps
```

The application, which has been called *erf* by the way, features an additional mode in which per packet data could be obtained. However, since the data collection is done entirely by hand,

would not have been feasible to use this mode to number of packets larger than several tens. One may see in the previous example, the total number of packets received was 7681 for a 60 seconds test at only 94 kbps.

The essence of the Network Measurement System was to smooth the performance of such tests, both from the user operation and data availability point of view. As you will see, measurement experiments using the Network Measurement System are based on the same concepts. However, a regular user essentially will care just about setting up the experimental stage, schedule the experiment to run now or at some time in the future and then, collecting the data which is already available in different formats by the test is done. In the meanwhile, he or she can focus on some other things, such as interpreting the results and drawing conclusions from experiments already finished.

The next chapter will start presenting some theoretical aspects, needed to understand the foundation of this work related to QoS measurements.

The third chapter contains the foundation of the Network Measurement System architecture, how you could setup the experiment described before but this time using NMS. Then, this presentation approaches the management related technologies involved in the operation of NMS, such as the SNMP management, the session and scheduling mechanism, the results collections and much more.

2

THEORETICAL FUNDAMENTALS

2.1 OVERVIEW OF QOS MEASUREMENT

2.1.1 PRINCIPLE OF QOS

According to [3] the denomination of *quality-of-service* or QoS for short means the ability of the network "to provide better service to selected network traffic". For a given network infrastructure establishing and maintaining a quality-of-service policy implies at least some of the following:

- Establishing priorities for dedicated traffic
- Bandwidth provisioning
- Controlled delay and jitter
- Improved loss characteristic

Each of these requirements of QoS comes with a set of prerequisites that shall be explained.

First, establishing priorities for specially selected traffic needs a set of rules according to which one can make distinction between different traffic flows. Imagine that in network with multiple nodes, the essence of having priorities is that the router from each node must know how it should treat the incoming packets on all its interfaces. Which have the highest priority and if priorities are set per flow rather per network basis, how to identify packets belonging to a specific flow?

These prerequisite of QoS implementation seems to be also the toughest to implement, especially when packets travel between networks belonging to different administrative authorities. Later in this introduction, there will be some examples about some solution that implement either local or end-to-end QoS.

Second, in order to ensure some strict characteristics of the network parameters and behavior there is the need of having QoS capable devices. Bandwidth provisioning, controlled delay, low probability of out-of-order packets implies having a network build up with intelligent routers and switches that can be instructed to give priority to the packets that need it at most. That is true not only for packets from a single client that pays for a higher service quality but also for packets coming from different applications of the same client. Between a file transfer and a real time streaming application, the latter should always have a higher priority. In addition to identifying traffic, the devices also must implement priority based service and queuing.

At last, end-to-end QoS is not possible whether the different ISPs do not have common QoS policies. This requires shared management, accounting and administration of QoS issues, otherwise the gold traffic from one ISP will be the bronze traffic of the next one and the overall outcome will be almost bronze as well.

The figure 2.1 suggests the key issues of QoS implementation.

Figure 2.1 Key issues of QoS implementation

2.1.2 PROVIDING QUALITY-OF-SERVICE

In TCP/IP heterogeneous networks there are three basic way for providing QoS. Current developing projects attempt to find new ways in reaching the goal of having end-to-end QoS in networks that do not belong to a single provider, and which are made up of a wide variety of equipments. However, since this introduction is just intended to create a glimpse on QoS only the classical methods shall be discussed.

The first possibility of QoS is not having QoS at all. The lack of QoS in an IP-based network is regarded as the old best-effort service, in which the packets rely on fortune in reaching the destination. Most of the networks today follow this approach, since not implementing a QoS policy is cost-free at the expense of poor performance for some customers and applications.

However, the lack of QoS and connectivity without guarantees may be of good quality if the service assurance can be improved by providing enough resources to meet the peak demands.

One of the most popular QoS implementation methods, which implement some QoS, is *differentiated services*. This policy gives priority to certain traffic, which is classified according to a given criterion.

The last method of ensuring QoS is *guaranteed service* that reserves the requested resources for traffic originating from a specific application.

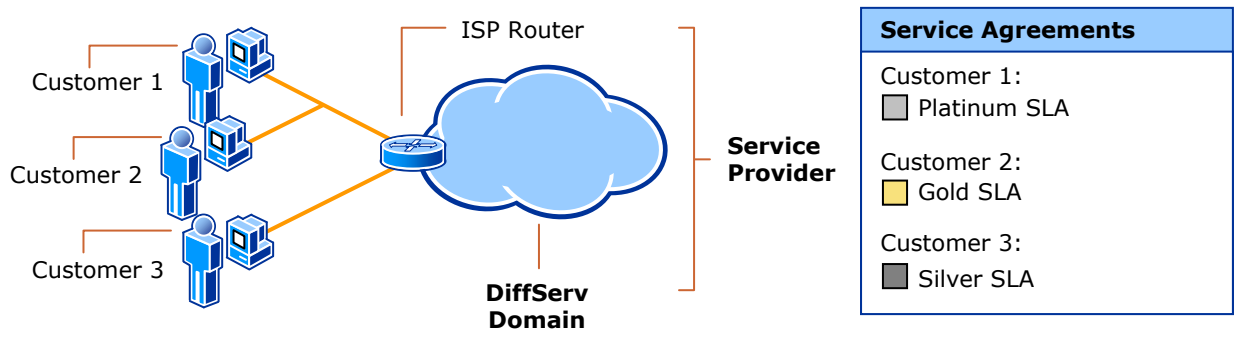
The next topics will a few examples on *DiffServ* and *IntServ* QoS policies, what are their advantages and disadvantages when implementing them in a heterogeneous network.

2.1.3 DIFFSERV QOS

Differentiated service or DiffServ ensures the same service for all packets coming from a given source. DiffServ always treats all packets coming from the same source as having the same QoS policy applied. The details of the QoS policy are usually negotiated as an agreement between the service provider and the customer that requests the QoS. This negotiation is often called a *Service Level Agreement* or *SLA*. The service provider ay have different *SLAs* with different customers of its networking service, meaning the customers that pays the most gets the highest service quality: his data packets are preferred in being served when the network or a node becomes congested.

The figure 2.2 illustrates the DiffServ applied to a small network. The service provider in that case sells three QoS policies, a platinum, gold and silver. It is obvious in this way, which is the best and most expensive service policy.

Figure 2.2 Applying differentiated service



When applying differentiated service the traffic is classified according to its source. For the previous example, the ISP router has a way of identifying which packets are from customer 1, which are from customer 2 and so forth. Because one of the conditions of implementing QoS requires the ISP router to be QoS capable when packets from both customer 1 and customer 2 reach the router at the same time, the first will be preferred. If the router link is congested, only the packets arriving from customers with the best SLA will get through, others will be discarded.

Figure 2.3 Internet Protocol version 4 header

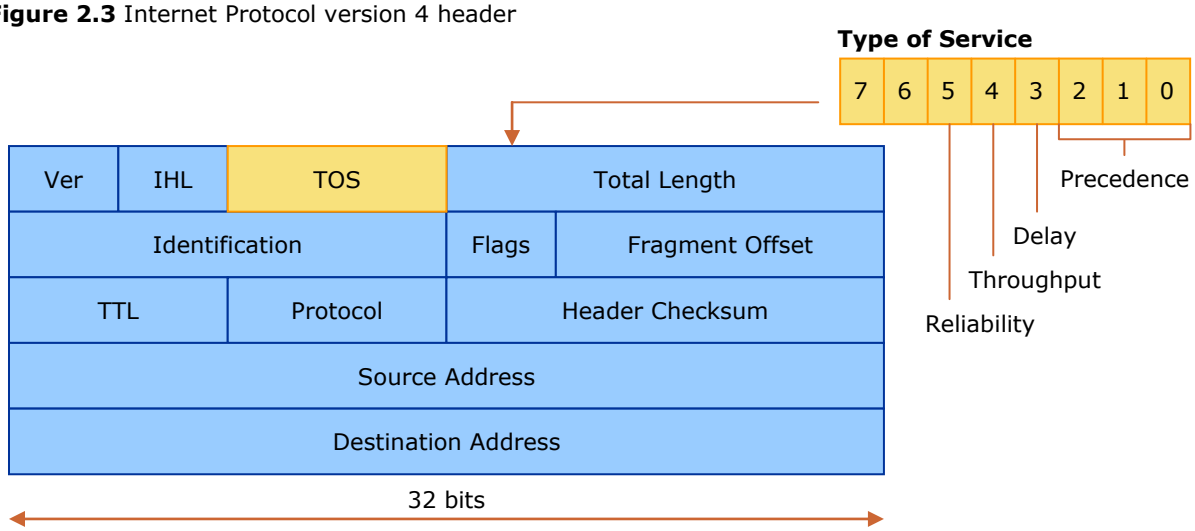
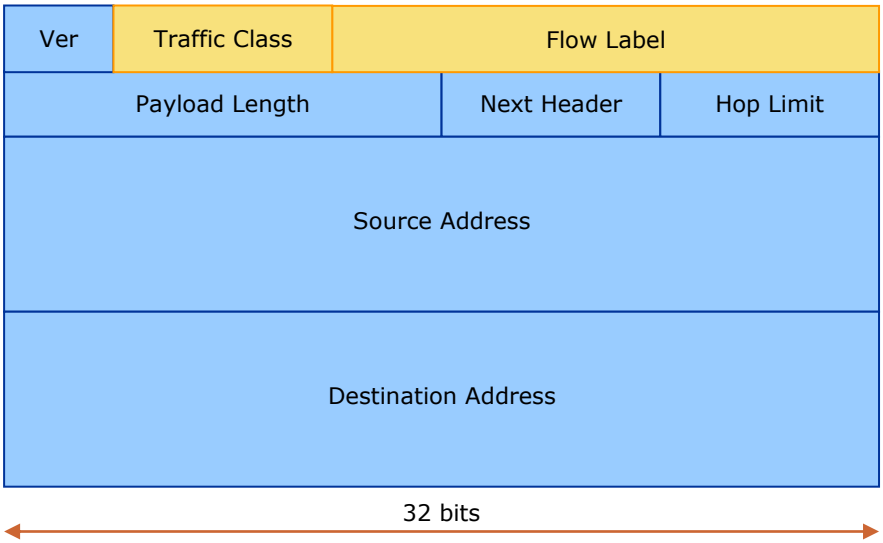


Figure 2.4 Internet Protocol version 6 header



How DiffServ is implemented in IP-based networks? The answer is very simple: using the fields already existing in both IPv4 and IPv6 packet headers. Look at the figures 2.3 and 2.4 to review the structure of IP headers. The IPv4 implementation uses the 8-bit type-of-service (TOS) field. The original intention was to use it to specify the service preference level, as the IP datagram travels through the network.

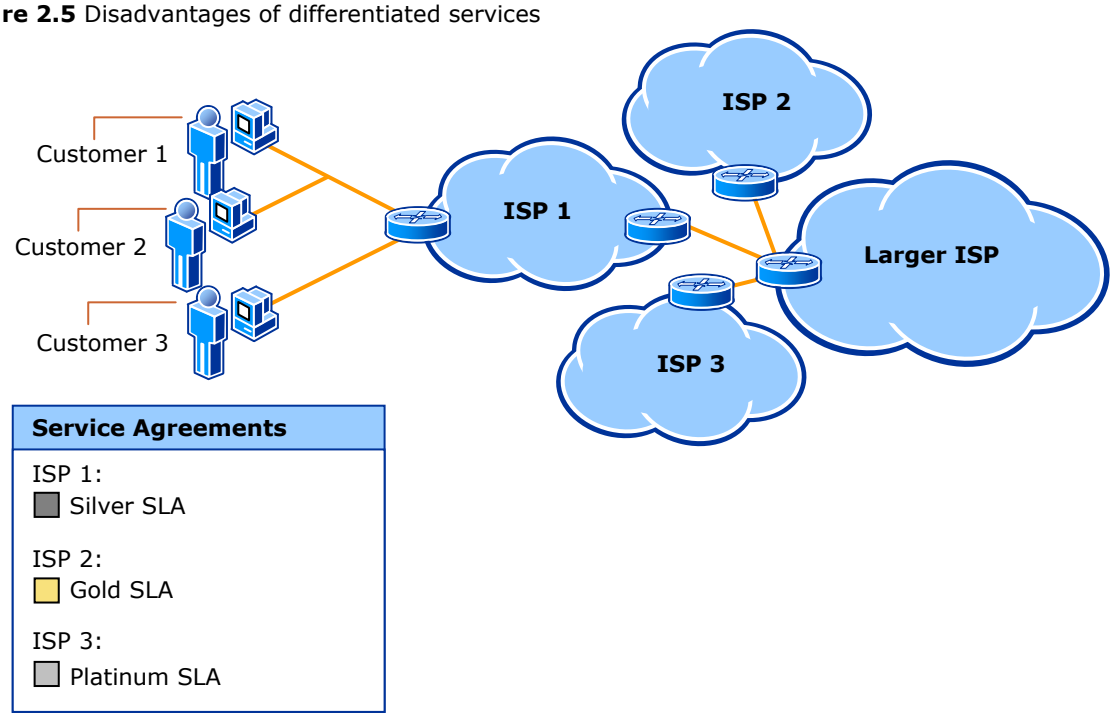
The initial structure of TOS given in [1] was the one from figure 2.3 in which different bits specified whether low delay, high throughput and high reliability are required. Nevertheless, in current implementations, this field is rather used with DiffServ and Explicit Congestion Notification (ECN) defined in [5].

In IPv6, the flow label is used to set-up priorities on a packet-basis while the flow label is intended for distinguishing between packets belonging to different flows (coming from different applications with different QoS needs) in order to implement the next section QoS type, *IntServ*.

Therefore, the structures of both IPv4 and IPv6 at least have support for QoS DiffServ implementation. However, some disadvantages prevented DiffServ of being implemented on a larger scale. The problem with differentiated service is the scalability and the fact it provides no support for end-to-end QoS.

In the previous example imagine the service provide of customers 1, 2 and 3 is also the customer to even a larger ISP. However, our service bought only the silver SLA of the larger ISP to which it is connected, while other ISPs paid for gold and platinum. Figure 2.5 depicts the scenario.

Figure 2.5 Disadvantages of differentiated services



The customer 1 (which is a platinum client for ISP 1) will always have priority in sending traffic through the network cloud (or DiffServ cloud) administered by its service provider. Nevertheless, if the destination of the packets sent by customer 1 is outside the scope of the network of ISP 1, i.e. the packets need to go through the network administered by the *larger ISP* they will have only silver priority and hence QoS level, because the ISP 1 is a silver customer of the larger ISP.

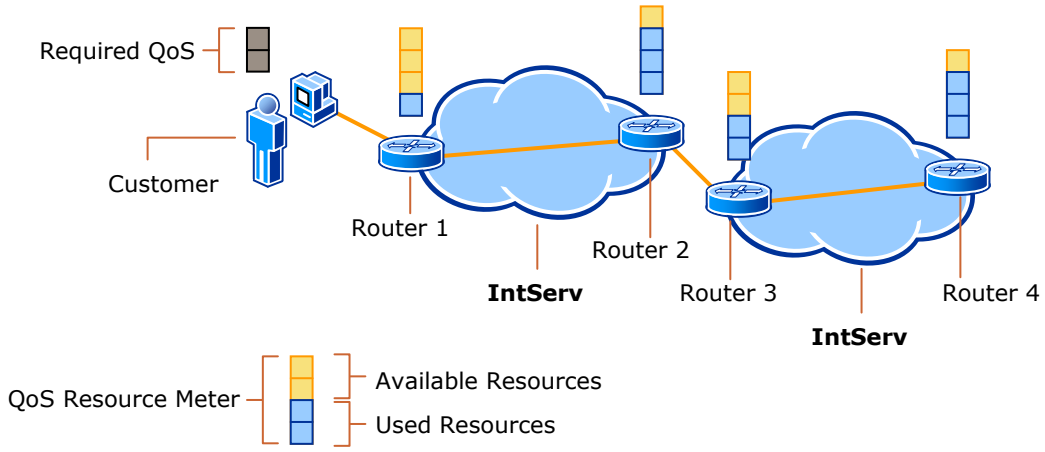
The advantage of using DiffServ is the simplicity in implementation and management since Traffic classification done at boundaries of the DiffServ domains. However, DiffServ cannot solve the issue of having end-to-end QoS. In most situations, overprovisioning along with best effort service is usually a better solution than applying DiffServ. This disadvantage attempts to be solved by the second approach, IntServ.

2.1.4 INTSERV QOS

IntServ implies reservation of resources in each hop for each flow. In this way, end-to-end quality of service can be achieved, since the same packet, when travels through an IntServ network will have the same service priority in each node it passes through.

The resources reservations are usually made using a specific reservation protocol. The Resource Reservation Protocol defined in [6] is one of the most popular. Look to the figure 2.6 for an IntServ scenario.

Figure 2.6 Applying integrated services



Prior the customer's computer sends any messages; resources are reserved in each node until the destination. RSVP or any other similar protocol can be used to achieve this. Each router along the path informs the source host whether the required resources were available. If not, the transmission may be canceled since at least one node in the network is not able to provide the degree of QoS that would have been necessary.

In figure 2.6, a resource meter at the top of each router indicates the percentage of resources that are in use and that are available. Assuming the host of the customer requires a lot of bandwidth for a real time transmission approximately half of the maximum resources from each router need to be reserved for the customer's application. In our example, this is not possible since the router 2 no longer has sufficient resources to cope with the desired QoS level. RSVP will inform the sender that the QoS level requested is not available and the transmission will probably be canceled.

Nevertheless, for most applications this mechanism is successful in ensuring the deemed levels of service. Other advantage of the IntServ scheme is that implements a fine-grained QoS system in which applications can make their own reservations. This means that packets leaving the customer's host will no longer be treated as a whole, but will rather have different QoS priorities depending on application, or on the flow otherwise said.

Unfortunately, flow-based QoS is also one of the major disadvantages of IntServ. Since each flow makes reservations on each hop on its path, the routers must keep a lot of information about each flow and resources reserved. In the case of IPv6, each datagram could be identified as belonging to a flow based on the 20-bit flow label. For very large networks, backbone routers may have difficulties in keeping data on very large number of flows, allocating resources according to each flow and serving the packets (i.e. perform routing) in this way. In conclusion, IntServ does not scale easily to large networks.

Current research in the field attempt to find different hybrid solutions to the end-to-end QoS problem since both of the classical DiffServ and IntServ are only partial successful:

- DiffServ ensures good end-to-end QoS as long as both communicating host are within the same DiffServ domain
- IntServ works across different administrative networks, but imposes problems at implementation on very large networks

Some other QoS related technologies, either open standards or proprietary are:

- Using the IP precedence bits in the way they are presented in figure 2.3 – this is the original usage method of the IPv4 TOS field
- Implementing routing access control lists (ACLs)
- Policy-based routing
- Committed access rate
- Network-based application recognition

Notes

- The last three technologies are implemented on Cisco routers.

This section was intended to provide a general picture about the quality-of-service subject. The next topic will get us closer the objective of this project, i.e. performing QoS-based measurements.

2.1.5 IP MEASUREMENTS FRAMEWORK

The reasons of measuring the QoS parameters is that only after knowledge about the link and service quality is obtained, the QoS-implementing device can take the appropriate traffic shaping actions. For instance, to appreciate the level of delay, packet delay variation, packet loss in order to determine whether the performance of the network is acceptable, there is the need of measuring these values.

For this purpose, IETF defined the *Framework for IP Performance Metrics* in RFC 2330 [7]. The Internet standard contains definitions and equations for determination any QoS related parameter. In additions are presented issues related to measurement methodology, errors and approximations, timing, synchronization and metrics. There are rules in sampling incoming stream of packets, how to determine the goodness of a link, even security considerations related to the interaction with normal Internet traffic.

The measurement framework sets up the basics of IP-based measurement. Since the objective of this project is to build a reliable management infrastructure for a distributed measurement application, rather than implementing the measurement software, the presentation of RFC 2330

will stop here. However, for additional information on this topic you may consult the technical reference of the measurement agent in [2] or check the RFC at the Web link given in [7].

The following paragraphs will explain shortly some QoS parameters that are standardized by IETF.

2.1.6 CONNECTIVITY

The metrics for connectivity are defined in the standard RFC 2678 [8]. This provides a series of definitions regarding the following terminology:

- One-way connectivity
- Two-way connectivity
- Instantaneous one-way connectivity
- Instantaneous two-way connectivity
- Two-way temporal connectivity

The properties of the connectivity metrics are given in table 2.1.

Table 2.1 IPPM connectivity metrics

Metric	Parameters	Unit	Description
One-way connectivity	The IP address of first host The IP address of second host Time Delta time	Boolean	It is true if in the interval of time plus delta time there is a moment in which the source has instantaneous one-way connectivity
Two-way connectivity	The IP address of first host The IP address of second host Time Delta time	Boolean	It is true if in the interval of time plus delta time first host has one-way connectivity with second host and the second host has one-way connectivity with the first
Instantaneous one-way connectivity	The IP address of first host The IP address of second host Time	Boolean	It is true if a packet transmitted at time moment arrives at the second host
Instantaneous two-way connectivity	The IP address of first host The IP address of second host Time	Boolean	It is true if at the time moment the first host has instantaneous one-way connectivity with the second and the second has instantaneous one-way connectivity with the first
Two-way temporal connectivity	The IP address of first host The IP address of second host Time Delta time	Boolean	It is true if within the time plus delta time the first host has instantaneous connectivity with the second and after that the second has instantaneous connectivity with the first

Notes

- IPPM stands for Internet Protocol Performance Metrics as defined in RFC 2330 [7].

2.1.7 DELAY

The metrics for the measurement of delay is specified in:

- RFC 2679 [9]: the one way delay
- RFC 2681 [10]: the round trip delay

The standard defined several delay metrics, from which the following are the most important:

- One-way delay
- One-way delay for Poisson stream
- Round-trip delay
- Round-trip delay for Poisson stream

In addition to these measurement metrics, the RFC documents specify additional statistic parameters for delay, such as the one-way delay percentile, median, minimum and inverse-percentile. Issues about timing, synchronization and calibration are also discussed with respect to their performance impact. The table 2.2 contains detailed information about the most important delay metrics.

Table 2.2 IPPM delay metrics (selection)

Metric	Parameters	Unit	Description
One-way delay	The IP address of first host The IP address of second host Time	Real number - seconds	It is the time interval between the transmission of the first bit at first host and reception of the last bit from the wire on the second host
One-way delay for Poisson stream	The IP address of first host The IP address of second host Time zero Time F Rate	Pair of real numbers	It is the pair time and delay where time is one of the moments of a Poisson process starting at time zero, finishing at time F with given arrival rate, and the delay is the value of the one-way delay obtained at the previous instances
Round-trip delay	The IP address of first host The IP address of second host Time	Real number - seconds	It is the time interval between the transmission of the first bit at first host and reception of the last bit from the wire of the message transmitted by the second host, when it received the last bit from the initial message
Round-trip delay for Poisson stream	The IP address of first host The IP address of second host Time zero Time F Rate	Pair of real numbers	It is the pair time and delay where time is one of the moments of a Poisson process starting at time zero, finishing at time F with given arrival rate, and the delay is the value of the round-trip delay obtained at the previous instances

2.1.8 PACKET LOSS

The one-way packet loss is defined in RFC 2680 [11]. The types of packet loss are:

- One-way packet loss
- One-way packet loss for Poisson stream

Their parameter and description is given in table 2.3.

Table 2.3 IPPM one-way packet loss

Metric	Parameters	Unit	Description
One-way packet loss	The IP address of first host The IP address of second host Time	Binary	It is one if the packet stared to be transmitted at time moment was received at the other host, zero otherwise
One-way packet loss for Poisson stream	The IP address of first host The IP address of second host Time zero Time F Rate	Pair of time and binary	It is the pair time and binary value where time is one of the moments of a Poisson process starting at time zero, finishing at time F with given arrival rate, and the binary number is the value of the one-way packet loss obtained at the previous instances

In addition to the QoS parameters so far, there are other parameters defined in both the RFC standards and other QoS related documents such as:

- Bulk transfer capacity
- Packet delay variation
- Out-of-order delivery
- Congestion avoidance capacity
- Error probability and dropped packets

The study of these parameters is outside the scope of this project. For more details on some of them, consult the documentation of the measurement agent [2], since several are implemented at the measurement agent software. For a complete reference, consult the references from the end of this document.

The following section will present several popular tools for traffic generation and computation of QoS parameters. These tools were also used for performance comparison when determining the accuracy of the NMS software.

2.1.9 QOS SOFTWARE TOOLS

This discussion should start with the presentation of the *MGEN* traffic generation software. Since many details are also given in [2], we shall try to keep the explanations as simple as possible.

MGEN is a versatile multi-purpose traffic generation tool created by the U.S. Naval Research Laboratory. The latest version of the tool is designed to generate UDP datagrams in a various set of distributions. The input data could be given either in a command-like format or as a text-file script, the latter being much easier to use when performing multiple-flow tests. The network layer protocol used can be either IPv4 or IPv6. Currently, a TCP version of the application is under development.

The script file supports the specification of the same parameters and has the advantage of enabling to save it on magnetic media. One example of *MGEN* script file was already presented in the previous chapter when the initial measurement scenario was presented.

The traffic *MGEN* generates is of the following types:

- Periodic, meaning that packets are equally distanced in time
- Poisson distributed, meaning the time interval between the packets obey a Poisson distribution
- Burst, that generates bursts of other *MGEN* pattern types at a specified average interval.

MGEN allows the creation of complex traffic patterns by using a compound of multiple "flows" with the same source/destination and with different pattern types and parameters [12].

The script file for *MGEN* is event-based: you can specify traffic related events to occur at different moments in time, such as starting generating traffic for a flow, ending a flow, capturing a flow and so forth. In addition, at allows setup of various layer 3 protocol fields, supports multicasting and for analysis, it creates a custom log file from which the desired results can be obtained.

Notes

- Since version 4, *MGEN* uses the same executable for both traffic generation and analysis. Either depending on the command line options or on the contents of the script file you can use the same application for both purposes. The latest version of *MGEN* software is 4.2.
- The version 3.x distribution is made up of two different applications for traffic generation and analysis: *MGEN* and *DREC* (dynamic receiver). Versions 4.x and 3.x are not interoperable. Additional modules such as a real-time trace plotter and a GPS utility are available for use with *MGEN*.

This thoroughly description, because as presented in the first chapter, *MGEN* was one of the tools used in the preparation for this project. In reality, the failure to combine smoothly the operation of *MGEN* and the software utilities of the Endace DAG card was one of the reasons for development of our Network Measurement System.

2.2 OVERVIEW OF NETWORK MANAGEMENT

The section will guide you through some of the main concepts behind network management such as what management is, what involves and why it should be implemented. Then the question of what is to be managed will receive an answer in the description of the ISO management areas, which similar to the seven layers of the OSI reference model an networking, act as a guideline in both learning an design.

The introduction would not be complete without a historical perspective of the three main industry approaches that set the foundation for the management technologies of the present and that still have a great influence on the development of the management technologies of the future.

In the end we will try to show you the key components (i.e. managers, agents and structure of management information) of a management infrastructure, that will act that a startup when talking about the management technologies that have already been developed.

2.2.1 WHAT IS NETWORK MANAGEMENT?

The network management is the process of monitoring (supervision) and controlling large complex distributed systems, in which usually the failures are common and resources scarce.

The network management consists of:

- network control
- network supervision

The terms, usually associated with control and supervision (or monitoring), are operation and maintenance.

The network management can also be defined as a service that employs a variety of tools, applications and devices in order to assist human network managers in monitoring and maintaining networks.

The computer and telecommunications networks are playing an increasing role in business world. The general trend towards globalization means larger and more complex networks, required to support many applications and to provide services to large number of users. On the other hand, this means that networked systems are becoming more and more sensitive to malfunctions, making proper management of communications an absolute necessity.

However, such management actions cannot be done by human effort alone due the slow response time, high costs and low efficiency. Automated tools are needed that enable network operators to provide the customers with the services they demand, in a way that creates the greatest possible customer satisfaction, while having these services provided at the lowest possible cost. Efficient network management can give a company a key competitive advantage.

While the benefits of efficient communication management can bring a key competitive advantage, its development is based on the following demands:

- New management tools for automated management. These tools may include any communications protocols, software and hardware components.

- Infrastructures based on equipments from different vendors, using different and sometimes proprietary technologies make more difficult to manage an entire network as whole.
- Further more having a management solution open to new possibilities of extension or upgrade, while on the same time it is required to work on the technology available today might seem even impossible.

Following these demands, it is obvious that for both current communication networks built with different vendor devices, but that up to a point still behave like isolated silos, and for the future multi-service networks, the development of standards widely accepted by the industry is mandatory.

Before implementing a management strategy it is important, what the objectives must be accomplished and what requirements are to be taken into account in order to ensure the viability of the solution chosen.

- Complexity: In this case, the management solution, or more proper said the management standard must be designed with the future in mind. Considering the rapid growth of worldwide telecommunication systems and of the Internet followed by a fast-changing industry is only reasonable to assume that the network management has to be built to change rather than built to last.
- Service: To improve service when the resources of the organization grow and redistribute.
- Dependable: Management must provide a high level of reliability, low downtime, while keeping the required level of performance, availability and security.
- Economics: Management must be focused on obtaining both the required payback from the investment made in the network resources and a high level of customer satisfaction for the offered service.

2.2.2 ISO MANAGEMENT FUNCTIONAL AREAS

When talking about a management strategy it is important to understand what is to be managed, i.e. having a model of the employed operations. One of the most known models is the one created by International Organization for Standardization (ISO). ISO has contributed a great deal to network standardization and in this field, it is best known of creating conceptual models used for both design and learning.

The network management model created by ISO is a framework with the primary means for understanding the major functions of a management system. It consists of five conceptual areas to act as a guideline in practice:

- Fault
- Accounting
- Performance
- Configuration
- Security

Fault management is to detect, log, notify users and up to the possible extent automatically fix the problems that might occur. Faults can cause disruption in service, downtime and performance

degradation so the purpose of fault management is to maximize the reliability and availability as much as possible.

Fault management involves first determining the symptoms and isolating the problem. This may be done either by the end users or management personnel or by the telecommunications network itself. In the last situation the signaling of errors, when they occur, is made either by the elements that experience a problem (event) or by the management center through the interrogation of the network components (polling). Then the problem is fixed and the solution is tested on all the affected components. Finally, the detection and resolution of the problem is recorded.

Accounting management enables operators to set up charges for the use of the telecommunications network resources. It measures network utilization according to some indicators and regulates the service offered to both individual users and groups of users, minimizing possible problems by ensuring the operation at the designed capacity and maximizing the fairness access for all users.

The steps involved in the process are the collection of data from the network itself, analyzing the results obtained and calculating the costs based on the provider policy. This information will finally yield billing information and information that can be used to measure the fair and optimal use of the resources available.

Performance management is monitoring or tracking the network activities and controlling or adjusting its parameters in order to improve performance or to maintain it at an acceptable level.

It requires data collection from the state variables that represent various aspects of the network performance, analyzing this data to determine whether the network functionality is within the allowed predefined limits. Examples of performance variables include network utilization, user response times or line utilization. If the thresholds are exceeded, an alert is generated and sent to the network management system and an action is usually taken to compensate. This action can be either reactive, which means that the system responds automatically or proactive, that means predicting through simulations of the future behavior of the network and finding a solution or workaround before the actually problem occurs.

A subset of performance management is the management of Quality of Service (QoS). This is the key factor of the provider's ability to ensure proper customer satisfaction and to fulfill its Service Level Agreements (SLAs).

Configuration management means to monitor the network and system configuration. It involves activities like initializing a component and shutting it down, adding, maintaining or updating relationships between these components.

Because each component may have a lot of configuration information associated with it, usually this information is organized in databases, inventories or directories, which in case of problems may be searched for clues to determine what has happened.
The management of configuration implies first the collection of already set configuration parameters from the equipments on the network. This can be done either manually (by technical personnel using application specific consoles) or automatically (by an automated management systems). Then the system must accept changes of those parameters. This requires the update of both the component behavior and of the configuration records. This is done locally on that equipment or remote with a management protocol.

Security management means to control the access to the network resources according to a certain policy and in such a way, these resources cannot be accessed without the proper authorization.

The key elements of the security management can be viewed from a physical security and an application security perspective. The first one means having all the equipment locked in secured areas and does not have too much to do with the actual network management itself. The second one involves having an environment security management that will provide information protection and access control. In order for the right information or resources to be accessed only by the right users, concepts such as authorization, authentication, secure data storage and communication were introduced. Because these measures are usually based on encrypted related security, the security management must often take into account the creation and distribution of encryption keys, sometimes with the help of a Certification Authority (CA) that will issue certificates for several purposes.

The general approach of implementing security management is based on the following:

- To define a security policy (having the users classified in groups with different access levels).
- To identify the different areas or access points of the network (such as gateways, consoles, terminals, wireless access points or even local or distributed services like remote control) that need to bee secured and assigning the proper permissions onto these devices.
- The final step is of applying, maintaining these security settings through the use of regular policies updates, encryption keys distribution for both issuance and revocation operations, while monitoring the network with the use of security audits to ensure that no unauthorized attempt occurs.

2.2.3 PERSPECTIVES OF MANAGEMENT

The three perspectives of management are closely related to the beginnings and evolution of the network industries.

The first one was the telecommunications industry, which for a long time was in many countries dominated by a state-owned or semi-public operator that offered telecommunication services within the whole country. The technological development in the telecom industry was driven by goals such as very high reliability, quality of service and within the last few decades, it is concerned with mobile communications. The standardization within the telecom sector is made through the effort of international standardization bodies that will issue standards for the vendors to adhere. The best-known example of such developed standard is Telecommunication Management Network (TMN) developed by International Telecommunications Union (ITU, formerly known as CCITT).

The growth of datacom industry, which has begun in the environment of universities or governmental projects, has been catalyzed by the expansion of the Internet and the wide spread of inexpensive personal computers. Compared with the telecommunications industry, which exists from near a century, the datacom one is young and has its roots in the 70's. The most known stack of protocols that is associated with the datacom networks is TCP/IP. Within this family, the Simple Network Management Protocol (SNMP) was developed for management purposes.

The computer industry focuses on the following aspects of management in computing:

- simplifying the user environment
- streamlining available configurations
- maintaining server operations

2.2.3.1 MANAGEMENT IN TELECOM INDUSTRY

The telecom industry started with widespread of the public switched telephony networks throughout the world. In many countries, the telephone operator was initially a public institution, acting as a national operator, which had the advantage of a physical infrastructure that covered wide areas and that was designed to provide services to large numbers of subscribers.
The infrastructure and the equipment used were costly, but provided high quality service. Because usually the operator within each country was like an isolated island, different types of equipments from different vendors were deployed with less compatibility or interoperability problems.

With the born of multinational companies or of the telecommunication networks that had to concern equipments or technologies from different vendors or of different types it was clear that especially for management there should by a single direction able to accommodate all needs. To ensure multi-vendor management there was the need of an international management standard to which the equipment designers must adhere. The most well known standard that tried to accomplish this goal was Telecommunication Management Network (TMN) developed by International Telecommunications Union – Telecommunications Standardization Sector (ITU-T) in 1989. However, because the standards lacks of several key specifications and because its development (in order to ensure the high reliability specific to telecom networks) took too long there is still a debate regarding its wide acceptance in practice.

2.2.3.2 MANAGEMENT IN DATACOM INDUSTRY

The management in datacom industry came with the growth of local area networks (LANs) and with the expansion of internetworking towards what was to become wide area networks (WANs) the Internet.

By the time automated management of these networks was a need in the beginning of the 80’s, the protocol suite that succeeded in data networks was TCP/IP. The development of a managing standard was greatly influenced on one hand by the fact that at that time computer networks were common only in universities and science laboratories and the expansion towards the consumer market was still at its beginnings. On the other hand, at that time there was no international standardization body that handled the development of standards for TCP/IP networks and most of the TCP/IP technologies were even in the research phase.

The common approach of datacom industry was to have simple, robust standards available when needed. Following this approach the Simple Network Management Protocol was born. Its advantage of the minimal design, that leaved room for vendor specific extensions, encouraged simple implementations and soon this protocol was adopted throughout the industry.

However more than a decade later, the drawbacks of this simple approach and the lack of advanced management concepts, security, efficiency and reliability, tribute to its original design, became problems.

2.2.3.3 MANAGEMENT IN COMPUTER INDUSTRY

The computer management exists since the beginning of computers commercial success. In the beginning were the large corporate mainframes and with the evolution of the computer technologies and the popularity of personal computers it became the management of server software, user environments and applications.

The standards employed were in the beginning hardware vendor specific and nowadays are usually software specific, but in either case, the management platforms are very different and it is difficult to find a common point to all of them. The major manufacturers dictate the criteria for managing their environments and the smaller designers of applications or tools that will have to work to the existing platforms follow the rules that have already been set.

However, in the last 10 years the industry made some major steps towards improving the interoperability in compatibility between their platforms. Starting with the creation of the Distributed Management Task Force – with the major objective to develop standards for desktop, network, enterprise and Internet – and the development of some distributed technologies there are strong signs that at least efforts are being made. From these distributed technologies, we can mention Remote Procedure Call (RPC), Common Object Request Broker Architecture (CORBA), Common Information Model (CIM), Common Object Model (COM) and Directory Enabled Networks, that overall try to smooth the interoperation between different systems.

2.2.3.4 MULTI-SERVICE MANAGEMENT

The movement towards multi-service networks comes from the demand by the market of communications tools and services that enhance performance and minimize costs. Therefore both telecom and datacom service provides have extended their service area toward each other in the attempt to anticipate the customer demand, while increasing the profit, enhancing performance and minimizing the costs both for the service provided and for the customer, which is not required to contact that service from another provider.

The telecom industry begins more and more to include data services in their portfolios, relying not only on their infrastructure used for backbone traffic (like ATM, Frame Relay, PDH, SDH or Sonet) but also expanding it towards the end-user with services like ISDN or different versions of DSL that benefit on both current cabling and the undergo enhancements. Unfortunately, the level of quality and performance regarding data traffic is still low compared to one in a datacom infrastructure (such as Ethernet over a LAN). In mobile communication, the matter is even worse: technologies like GSM (which was not even designed for transport of data) and CDMA2000 being well behind the performance achieved even within a Wireless LAN (WLAN) network.

The datacom providers are taking advantage of the new technologies developed like voice over IP to enable support for voice and video streaming and with these services like telephony, fax and video conferencing. However, these networks are usually TCP/IP based and this stack of protocols still lacks in functionality and quality. Packet switching versus circuit switching is, of course, the major factor in performance penalty, but beyond that, the protocol suite did not offer so far parameters that could control successfully the quality of service or traffic/subscription class. This comes especially when establishing of priorities for packets transported within aggregated flows over a backbone connection is required.

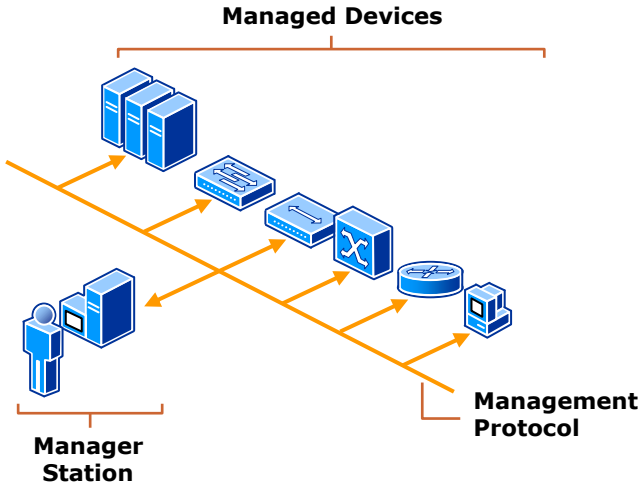
To make things more complex, these facts are related to service provisioning only. When it comes to management, the entire spectrum of applications, devices, vendors and even standards along with the whole set of parameters, states and variables that can be changed, configured or worth monitoring, make it undoubtedly complex.

2.2.4 MANAGEMENT ARCHITECTURE

The management architecture is the collection of managing and managed devices and of the communication of management information between them.

When talking about the management architecture it is important to understand that, it should be related to a specific management technology. For example, there is the so-called Directory Enabled Network (DEN) that keeps track of their resources by using a central repository called directory. The management applications running in such environment and are DEN enabled must identify each resource based on its information from the directory. However, in what follows it will be presented the simple, classical management architecture, which can be found in some of the most used management technologies such as the Simple Network Management Protocol (SNMP) developed initially for IP networks and Telecommunications Management Network (TMN) designed to comprehend a wide spectrum of telecommunication networks.

Figure 2.7 The general architecture of a management system



The information presented here will act as a basis for the future understanding of any other management technology, even if some of the terms described here could be changed or the main components or the interaction between them might by altered.

2.2.4.1 MANAGERS

The managers are the nerve center of a management infrastructure that collect information about the state of the network and send configuration messages to the network elements.

Their key functions are to receive notification data for the managed devices, also known as traps, to request information from these devices or to send parameters. A manager must also allow a configuration to be set up o specify number of retries, timeout duration or polling intervals that will be used as thresholds in order to determine that a device is not responding. The messages transmitted between a managed device or network element, on one side, and a manager on the

other side, is related to any aspect regarding fault, configuration, accounting, performance and security.

The managers must be aware of the structure and format of management information used by the managed devices. This is true regardless of the protocol or the technology involved. To keep their products competitive and to fulfill at their best the objectives of management vendors enhance their products with information and manageable parameters that were not even conceived when the standards used were designed. Therefore, the specifications of the products must contain any new feature added to the management information and the managers must be configured to recognize and interpret this information (if the management protocol is flexible enough to allow that).

The managers should also allow the administrators to define some action templates based on rules to be executed automatically, without user or technical support personnel intervention, in the case of certain events.

2.2.4.2 AGENTS

Agents are pieces of software or hardware implemented functions that run on the managed devices. The first duty of an agent, which is the hardware or software routines within the managed device that are management aware, is responding to the inquiries of the manager.

It should be able, to the extend offered by either the device itself or to any security policy in place, to allow inspection and changing of any device parameters managed at a given time including configuration of the management structure or information. Then it should detect any abnormal condition and report it to the manager without any other intervention.

Examples of agents include a specific service of a Windows® based computer, a daemon running on a Linux machine or software routing from the Cisco's IOS® operating system used on Cisco devices.

The agent requires minimal configuration before use like system information, the manager location (if the agent uses the notification method of communication with the manager) and access rights, such that only management messages received from the trusted managing stations to be allowed.

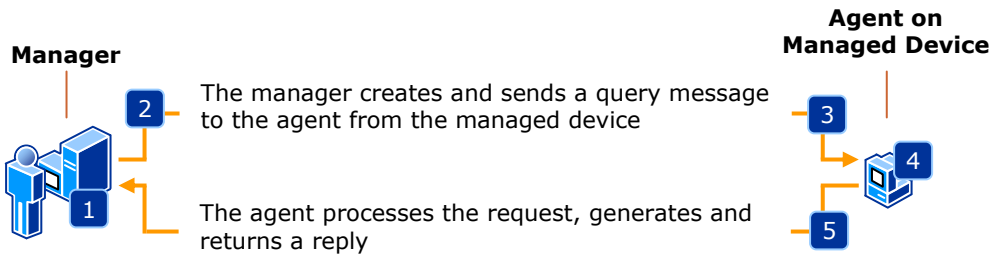
2.2.4.3 MANAGER – AGENT COMMUNICATION

In the previous paragraphs, we have discussed about the main components of a management system, i.e. manager and managed devices (or more appropriate agents). Now is time to talk about the manager-agent communication process. This is the way of the agent informing about the status of the managed device or of the manager requesting some information. The communication is usually based on the transport provided by a management protocol specific to both the platform and managing technologies. Regardless of the implementation, two transported major types of messages carry management information.

The first one is polling based, meaning that managers request information periodically from the network elements. The advantage of this approach is that the complexity of the agents is very low: they will simply have to wait for a request from the manager to be received and then to return the result. A debate is made on what should be an appropriate polling interval. A too small value will result on having the most up-to-date information regarding the network, but wasting

valuable bandwidth. By having a too low value, this results in saving the bandwidth and improving the performance but the probability of missing key events or being aware of them too late increases.

Figure 2.8 Polling based communication mechanism



The communication between the manager and agent of the managed device in the example above proceeds in the following fashion:

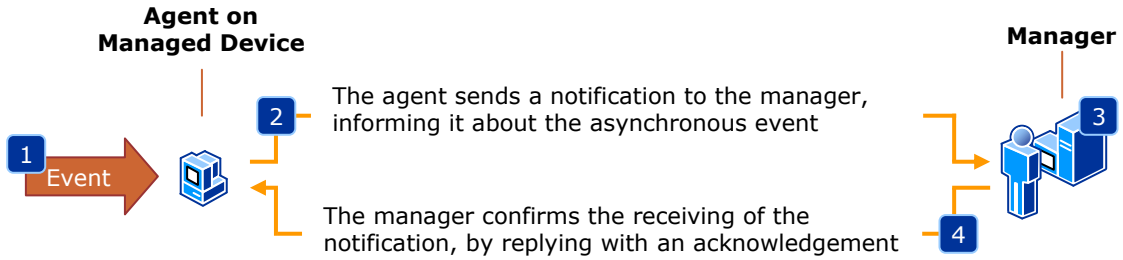
1. The manager forms a query message that contains an information request and the destination of the message — the address of the managed device on the network.
2. The manager sends the information request to the managed device by using the management service running on the physical machine.
3. When the managed device receives the message, it verifies that the current machine is the destination of the query; it checks the authenticity of the message and evaluates the request against the agent's list of access permissions.
4. If the authentication data or access permission is incorrect, the agent might send a notification to a certain management station, if any has been configured.
5. The agent component of the managed device calls the appropriate local service to retrieve the requested information.
6. The managed device sends the response to the manager station.

Notes

- From the list of operations performed by the managed device and the agent at the step 3, the actual activity depends on the management technology involved.

The second method is event-driven, which means that the network elements have the managing intelligence of informing the manager that something has happened. This overcomes the problem of bandwidth from the polling approach but requires the network element to be still operational in order to report a problem and increases the processing and resource usage at the managed device in order to evaluate each operation parameter to determine if a message should be sent to the manager.

Figure 2.9 Event driven communication mechanism



The communication between the manager and agent of the managed device in the example above proceeds in the following fashion:

1. An event – a situation that can be identified through some triggering parameters – occurs on the managed device. The events are usually programmed by the manufacturer of the device, the vendor of the management software or by the management staff responsible for running the network. Examples include a device starting up or shutting down, an interface going online or offline, a malfunction or the exceeding of a system parameter of a preset threshold.
2. The agent from the managed device creates a notification message to be sent to the manager. Message information contains data about the event, security information to enable the manager to receive and process the notification and the address of the manager, previously configured on the agent.
3. The manager receives the notification message, verifies the authentication information and interprets the notification data, taking an appropriate action. This action is programmed into the manager. It could be a visual or audio alarm reported to management staff or by performing a predefined or determined – according to its level of intelligence – operation on the managed device, using a polling-based mechanism.
4. Regardless of the taken action, the manager station can return an acknowledgment message confirming the successful receiving of the notification message.

The most used option is however the mixed one. In this case, event-driven messages could be sent in the case of extraordinary events, while polling can be used at larger intervals, during normal operation.

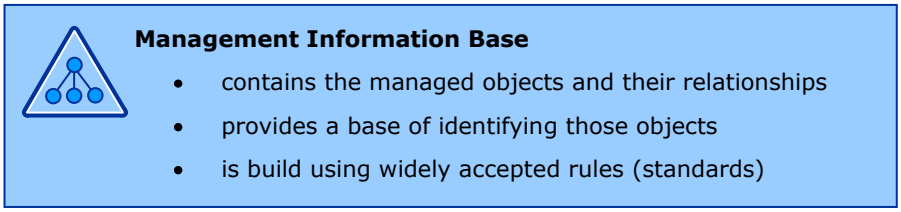
There is even the possibility that a message from the agent triggered by a certain event to generate some polls from the manager in order to obtain more information. Some devices may send a message to the manager to inform about the passing from a bad state to a good one and the resuming of normal operation. There is no restriction on when each type of messages could be sent.

2.2.5 MANAGEMENT INFORMATION

The management information is the information about the managed nodes and devices. It most of the management implementations it is regarded as a collection of physical or logical resources that can be managed, named managed objects. This abstraction is useful in hiding from the management system the specific details of that resource, others than the ones that have management importance, such as implementation, vendor or device specific extensions, access methods. For example, the implementation of counters that keep the amount of information transferred over an interface of a networking device depends on the type of the device, the technology used and vendor. From the management point of view, however, this should be regarded as the same resource despite of the actual physical entity that keeps track of this information.

On the other hand, due to the wide variety of manufacturers, equipments, applications and parameters that could be managed a repository of the possible manageable information is needed, such that both agents and managers understand each other.

Figure 2.10 Management Information Base characteristics



A management information base is a repository of the managed objects. It is defined using some rules, which define its structure and the objects representation such that this information could be exchange between the network elements. An analogy with an information base is a dictionary that both managers and agents use to localize and identify resources based on a code that is transmitted over the network.

2.3 SIMPLE NETWORK MANAGEMENT PROTOCOL

2.3.1 OVERVIEW

The development of SNMP came as a solution to the need of managing and troubleshooting more and more complex TCP/IP based networks. Before SNMP, maintaining these networks was done exclusively by human effort alone with the help of some simplistic tools like ping, traceroute or telnet used for both troubleshooting and configuration purposes.

As the infrastructure grew not only the effort but the costs grew consequently while the efficiency of human intervention became lower and lower. It was obvious that to meet the growing demand of networking by the industry (and nowadays by the consumer market as well) automated management technologies were required to help both in monitoring faults and setting up configurations as inexpensive and well as possible.

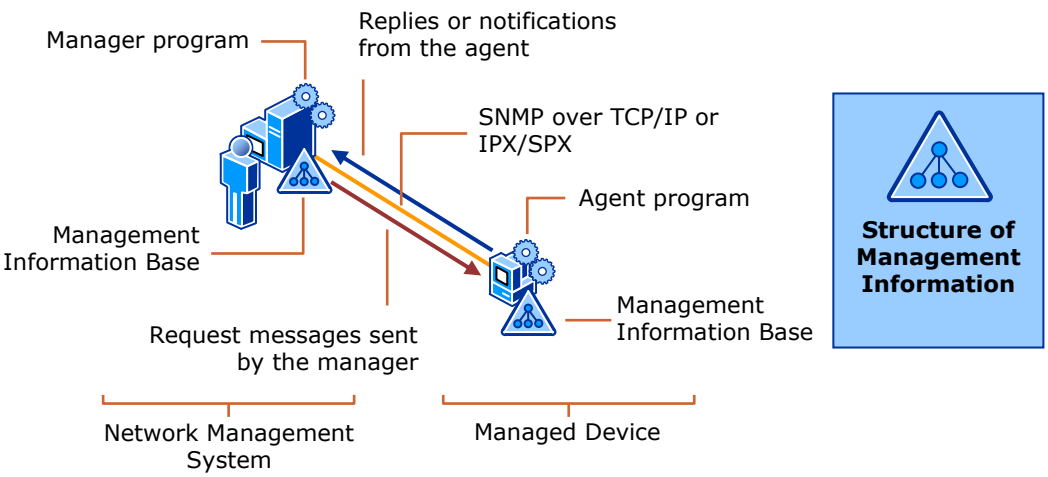
2.3.1.1 IETF STANDARDIZATION

In the field of management, the result of IETF efforts was Internet Standard Management Framework (ISMF), most commonly known as SNMP management, even if SNMP itself is only a part of the framework. The ISMF is composed of several key elements:

- The managed nodes (i.e. the network elements) each of them having a management aware entity usual called an agent, that performs the management specific functions such as monitoring the operation of the managed node and sending that data, or receiving control data and performing configuration related tasks.
- The manager is a station or device that has a SNMP enabled management application that communicates with the agent from the managed devices by receiving status information and sending control messages.
- The management protocol itself: Simple Network Management Protocol. It is responsible for management information transport over the TCP/IP network. (According to RFC 1449, SNMP supports transport not only over TCP/IP based-networks but also over OSI, DDP and IPX/SPX.)
- The management information which is divided into the variables, more commonly referred to as objects, which track the monitored parameters of the managed devices that constitute the Management Information Base (MIB) and the rules for defining these objects and their behavior: the Structure for Management Information (SMI).
- A set of management operations that enable managers and agents to perform some functions on the managed data
- The initial SNMP standard was far from complete when initially released in 1990. It lacked in features like security that were not required or a priority at that time. Since then two more versions of SNMP were released along with other technologies like Remote Monitoring (RMON).

The following figure represents the IETF management framework.

Figure 2.11 Internet Standard Management Framework



2.3.1.2 VERSIONS OF SNMP

The predecessor of SNMP was Simple Gateway Management Protocol (SGMP) developed for managing IP routers. Based on it, IETF created SNMP, which is able to manage any networking device that is SNMP aware, either having a hardware component designed for SNMP management or more commonly running SNMP management software.

SNMP is the central piece of the Internet Standard Management Framework (ISMF) that consists in two types of SNMP entities (a manager and a managed one – referred to as an agent), the protocol itself and the management information.

SNMP consists in a set of a few simple operations along with the management information these operations gather. This allows collection of information from any device that has SNMP support and monitoring its state or enables an administrator to perform remote configuration operations. This is true both for hardware but also for software, i.e. applications such as services and databases.

IETF developed so far three versions of the protocol. They are presented in table 2.4.

Table 2.4 Versions of SNMP

Version	Description
SNMPv1	It is intended as a simple, robust protocol for management of IP-based networks, particularly for fault and configuration. The result was a protocol widely accepted by the industry and therefore, almost any device today has support for SNMP. The disadvantages: has support only for IP (and more recently for IPX) networks, is inefficient in transferring larges amount of management data and virtually no security mechanism.
SNMPv2	This version tried to eliminate these disadvantages. However, it was difficult to agree on the solutions to those problems and several versions of the protocol appeared. Therefore, the industry was reticent in implementing them and SNMPv2 never caught on.
SNMPv2p	SNMPv2p updated the protocol itself: operations, data types and only some security improvements.

SNMPv2c	This version was called community string-based SNMPv2. It uses the same authentication method like in SNMPv1 based on community names.
SNMPv2u	It is the same like v2c but has an improved user-based security system that uses an encrypted authentication mechanism.
SNMPv3	The version 3 brought wide changes not only on the protocol itself but also on the concepts of the management framework. It provides cryptographic based security, allowing authentication and privacy protection.

2.3.2 BASIC COMPONENTS

Managers and managed devices are the two end-points of the management process. While it is obvious that the manager is a SNMP enabled entity, usually even a dedicated machine with management applications that handles the network management process, the tasks of the managed device focus on its main operating functions – that is why the device was built. The fact that is manageable and SNMP aware is an advantage for it, but is not compulsory. That is why the SNMP entity within the managed device, sometimes referred as network elements, must be distinguished somehow and it is called an agent.

Managed devices are networking equipments such as workstations, servers, routers or printers. Agents are software routines, services or applications that reside and run on these equipments and are able to communicate to the manager.

The manager – usually called a Network Management System (NMS) – is the management platform at an operations centers used by network operators to monitor network status. Simple to complex management applications enable some or all of the management functional areas to be performed.

The communication initiative can be held by any of managers and agents. Managers send or request management data to or from agents through queries called polls. Polling is used to check the status of a device or change its operation. On the other hand, agents might need to notify managers of some exceptional events that occurred. Event-driven messages, which in SNMP language are called traps, are sent to the NMS in case of a fault or some other important event filtered on rules that depend on the abilities of each agent instance. Some devices will send a corresponding "all clear" trap when there is a transition from a bad state to a good state. This can be useful for the manager in determining when a problem situation has been resolved.

2.3.2.1 COMMUNITY NAMES

SNMPv1 and SNMPv2 (more precisely SNMPv2c) use the notion of community names to establish trust relationships between managers and agents. These names are used like password. They give to a SNMP entity the ability to access management information on another one.

The main disadvantage of using community names as a method of authentication is that these names are sent through the network in clear text and no encryption is performed on either them or the transported data. The security method developed with SNMPv1 is insufficient in nowadays networks and the next versions of SNMP tried to improve it.

There are three types of community names (see table 2.5). These types define the access level, i.e. a SNMP message carrying an operation to be performed on some variable from a managed

device is restricted by the community name. If the message carries the wrong community name, no access is allowed at all.

Table 2.5 Types of community strings

Community Type	Description
READ-ONLY	It allows reading the parameters from an agent (e.g. reading the number of packets transferred through an interface) but does not allow changes to be performed (e.g. for the same interfaces the counters cannot be reset).
READ-WRITE	It allows both reading and writing operations on the variables held by the agent. This type of community is used for setting up configurations and a manager that uses this community can gain complete control of a managed device in the limits allowed by SNMP.
TRAP	It enables the manager to receive traps – asynchronous notifications from the agent. An agent sending a trap must use this community in order for the manager to accept the message.

Traditional names for READ-ONLY and READ-WRITE community are *public* and *private* respectively. A proper method of securing the network implies changing the community names such that intruders cannot inspect or, worse, take control of it. However, as it was stated before, this security method is only a superficial one. Analyzing management traffic reveals instantly both community names and management data, and this is way community based security virtual lacks in any authentication mechanism.

2.3.2.2 MANAGER CONFIGURATION

The manager is the location from where managed devices are monitored and controlled. Its key functions are:

- To receive and to interpret the traps received from the agents, to take the corresponding action, if necessary
- To upload the management information from the managed device
- To download the parameter values to the managed devices – the last two operations being realized through polls

Managers usually consist in a set of applications that provide a user interface – technical staff can use to configure thoroughly its options, databases that keep the management configuration for the entire network, event logs with various alerting features to notify when something is wrong.

Managers use most of the processing and memory resources between the SNMP entities (for large network or management infrastructures entire computing equipments are assigned only for this purpose) and require detailed configuration before use.

This management-specific configuration can be split as follows:

1. Access Configuration – implies setting up the community names (for SNMPv1 and v2) in order for managers to be able to receive traps and to send requests. (For SNMPv3 things are more complex, depending on the security method used.)
2. Polling Configuration – the manager must be configured with the kind of information it must request from the managed devices, what kind of command to give or what parameters to setup. For frequent operations, such as checking the operational status of a

device or component of it, the polling frequency needs to be specified. In order to execute operations that are more complex, once a week or one time only, advanced managers set up schedules for each activity.

3. SNMP Configuration – involves setting-up the transmission control parameters. SNMP relies on User Datagram Protocol (UDP) from TCP/IP stack at the transport layer, SNMP itself being an application protocol. UDP is connectionless and unreliable, so SNMP must handle reliability issues. This is done through retransmissions after waiting a given duration for a reply. Most managers allow the configuration of this timeout period and the number of retries.

2.3.2.3 AGENT CONFIGURATION

An agent is the process executing management tasks inside the managed device (network element). The actual implementation is vendor specific, but regardless, the major functions that have to be performed are the following:

- To send a reply to a query received from the manager, after it validates the incoming message (using the community name or by some other method for SNMPv3) and processes the request.
- To detect exceptions based on the configured policy and send traps to the NMS.
- The agent controls the access of the manager to each variable from the management information base, usually using the community name and type.

To complete these functions the following minimal SNMP specific setup needs to be performed prior to agent's use:

1. System Information – system and contact information (e.g. the device name, description, location, administrative area) or any other kind that might be useful in troubleshooting. This is required primarily to identify the device if something goes wrong.
2. Trap Destination – the address of the manager station has to be known by the agent in order to send traps.
3. Access Control – implies setting up the community names, specifying their types such that both traps can be sent and polls received.

This minimal configuration is required for any agent regardless its simplicity such as built-in agents for various equipments that already have all other parameters configured by the manufacturer. More complex agents may allow refining management or adding additional features through the configuration of trap policies (i.e. in which cases a trap should be sent and what it should contain as data) or objects access control (i.e. what information from the MIB should be made available).

The configuration requirements described here for both managers and agents is only the prerequisite. Vendors usually add their own options to make their products more flexible and competitive.

However, in this paper we shall resume only to the management specific. SNMP entities must be prior configured to work over a TCP/IP (or IPX/SPX if the implementation supports) network and depending on the situation additional settings are required.

2.3.3 MANAGEMENT INFORMATION

The management information is the information about the managed nodes and devices. This information can be organized in information elements or variables, each tracking a certain aspect or property of the network element.

Each information element is thought to as an abstraction using the notion of managed object. These objects represent physical or logical resources that are being managed and the object's properties represent the actual information. Because these objects are no more than a simple convention used between managers and agents some rules are required.

First, they must have a limited, known number of possible data types. This requirement is obvious: a certain data type implies a known amount of data and some specific processing. Regardless of the object that depends on the device involved, and though there are more to be created, using standardized types means that the entities not only use the same language, i.e. SNMP, but speak in the same terms as well. Having few data types possible simplifies things and implementations, while having more types, it means additional information on what that object is and what to do with it.

Second, each object needs to be identified somehow. The identification solution must be open to new member objects and must be adopted as a standard. Remember that there are a lot of network elements, each with specific parameters that worth monitoring or configured and some are vendor specific – so the solution adopted must allow these parameters to be managed as well, even if they were not invented when the protocol was designed. The actual meaning of each object based on its identifier – or object ID – should be recognized by both manager and agent.

Traditionally the collection of all manageable objects is called by the standard the Management Information Base (MIB). Any sort of status or statistical information that can be accessed by the NMS is defined in a MIB. The rules of defining these objects, their behavior as well as the rule for defining the MIB itself are called the Structure of Management Information (SMI). For every new state variable of configuration parameter of a device, a MIB must be defined containing the definition of an object for that variable or parameter. The MIB and the object definition must be done according to SMI.

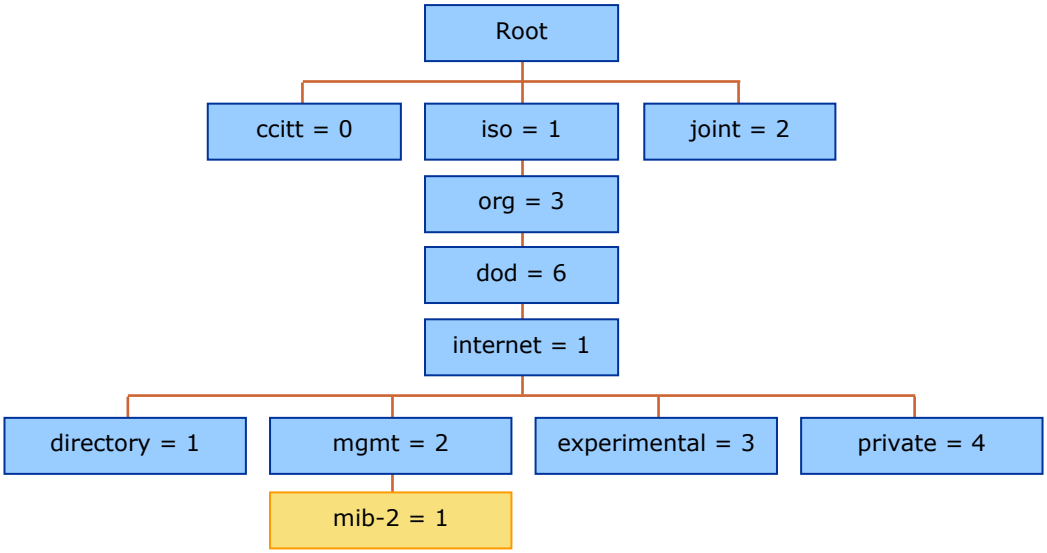
2.3.3.1 MANAGEMENT INFORMATION BASE

The Management Information Base is a virtual store, which comprises all management information from the managed network. The MIB defines the managed objects that an SNMP manager monitors (or sometimes configures) on an SNMP agent. Each system in a network (workstation, server, router, bridge, and so forth) maintains a MIB that reflects the status of the managed resources on that system, such as the version of the software running on the device, the IP address assigned to a port or interface, the amount of free hard drive space, or the number of open files. The MIB does not contain static data, but is instead an object-oriented, dynamic database that provides a logical collection of managed object definitions. The MIB defines the data type of each managed object and describes the object.

The structure of the MIB is of a tree-like hierarchy were the tree nodes representing the managed objects. The structure itself is described by the Structure of Management Information (SMI) that specifies which object is found in some of the root nodes and how the entire tree architecture is organized. The part of the tree that concerns with Internet SNMP management is defined as follows (see figure 2.12): immediately beneath the root of the MIB tree, International

Organization for Standardization (iso) is the Organization (org) branch, followed by Department of Defense (dod), and then Internet (internet). All objects that concern Internet technologies can be found under this node.

Figure 2.12 The path to Internet MIB



The SNMP-related branches of the MIB tree are located in the internet branch, which contains two main types of branches:

- Public branches (*mgmt*=2), which are defined by the Internet Engineering Task Force (IETF) RFCs, are the same for all SNMP-managed devices.
- Private branches (*private*=4), which are assigned by the Internet Assigned Numbers Authority (IANA), are defined by the companies and organizations to which these branches are assigned.

Other branches like *directory*=1 and *experimental*=4 are not used or reserved for testing purposes. There are no limits on the width and depth of the MIB tree.

Management (mgmt), the main public branch, defines network management parameters common to devices from all vendors. Underneath the Management branch is MIB-II (mib-2) – a special MIB that must be implemented by any SNMP-enabled device and beneath this are branches for common management functions such as system management, printers, host resources and interfaces.

The private branch of the MIB tree contains branches for large organizations, organized under the enterprises branch. Each organization has a root branch node under this object. Each organization creates its own subset of MIB branches and objects, which must comply with the Structure of Management Information.

2.3.3.2 MANAGED OBJECTS

At the programmatic level, the definition of each MIB object that an SNMP agent manages includes the following elements:

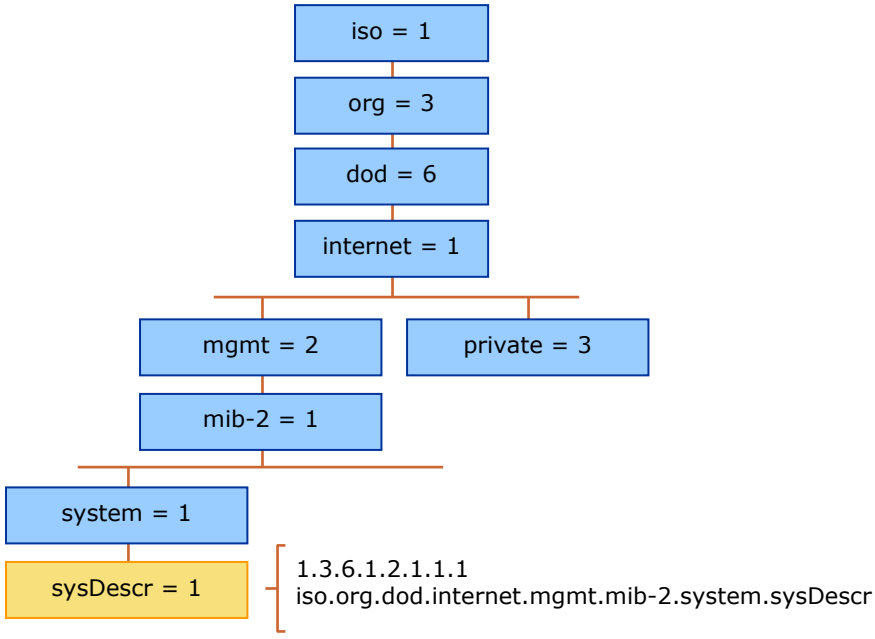
- The object name and object identifier (also known as an OID)
- A text description of the object

- The object's data-type definition (such as counter, string, gauge, or address) The level of access to the object (such as read or read/write) that is allowed
- Size restrictions
- Range information

SNMP references each MIB variable by using its unique object identifier, which identifies the location of a given managed object within the MIB namespace. The object identifier reflects the object's position within the hierarchy of the MIB tree, containing a sequence of subidentifiers that begin at the root of the MIB tree and end at the object (leaf node). Subidentifiers are separated with a period.

Let us look at the situation from figure 2.13. To reference the MIB object at the bottom (showed in yellow), either numeric or text subidentifiers can be used. For example, the following text-based object identifier is interchangeable with its numeric counterpart, shown beneath it. The value of this object identifier, in either format, identifies the current operational state of a network adapter.

Figure 2.13 Using managed objects



The description of this object in the MIB file, following SMI syntax rules, would be:

```
sysDescr OBJECT-TYPE ::= {system 1}
```

Where the parent nodes, i.e. system and so on are specified as follows:

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

Corresponding to each object identifier is a value that represents the current state of the object. SNMP, which accesses only the leaf nodes in the MIB tree, references a MIB variable by the

dotted numeric string that represents its object identifier in order to retrieve the current value of the variable.

2.3.4 STRUCTURE OF MANAGEMENT INFORMATION

Along with the definition of Management Information Base and with the standard that describes the SNMP architecture, the SMI is one of the three pieces that define a simple, workable management framework for TCP/IP networks. The SMI handles with rules and conventions of the management data that is defined in MIBs and transported by the protocol. The first version of SMI (RFC 1155) focused on the following main objectives:

- To specify the path-to-root structure of the MIB tree with precise rules where the management objects are located, and where new or vendor-specific ones can be added, i.e. the MIB extensions. The schema of the MIB has been created using ASN.1 convention.
- Set up the definition format for the managed objects. This involves the syntax, semantics, data types, attributes, encoding and guidelines for definitions to be precisely formulated. The textual definition of each object that is found in a MIB is called a macro. The definition of *sysDescr* object from the example was presented using the macro notation. Because it defines an object and it starts always with the same keyword is referred to as a macro of OBJECT-TYPE.

The SMI version 2 (RFC 2578) brought several enhancements. It increased the number of defined types to better meet developers needs (more data types on 32 and 64 bits, conceptual tables) and the number of object types (tree new more object macros were provided, each having specific functionality and attributes, including trap object that replace the protocol specific messages from SNMPv1 and give more flexibility to trap mechanism implementation).

2.3.4.1 SMI DATA TYPES

Data types are used in the SYNTAX clause of an object definition macro.
SMI version 1 (published in RFC 1155) defines the following data types:

- Primitive types (or non-aggregate types) are subset of ASN.1 in table 2.6.
- Constructor types that can be lists defined with SEQUENCE key word or tables defined with SEQUENCE OF.
- Defined types based on the primitive ones, in table 2.7.

Table 2.6 SMIv1 primitive data types

Data Type	Description
INTEGER	Is a 32-bit number used both for numeric data and enumerated values (such as up, down or testing as a status of an interface)
OCTET STRING	A sequence of zero or more bytes
OBJECT IDENTIFIER	A dotted decimal string represents the managed object within the MIB tree
NULL	It is a legacy type inherited from ASN.1. It is not currently used in SNMP.

Table 2.7 SMIv1 defined data types

Data Type	Description
NetworkAddress	A generic network address from any protocol family
IpAddress	A 4 byte IP version 4 address
Counter	A 32-bit unsigned integer that monotonically increases until it reaches a maximum value, when it wraps around and starts increasing again from zero
Gauge	A 32-bit integer similar to Counter, but unlike it the maximum value cannot be exceeded
TimeTicks	A 32-bit integer that measures time in hundredths of a second
Opaque	Allows any other ASN.1 encoding to be stuffed into an OCTET STRING

SMI version 2 adds these types:

Table 2.8 Data types added by SMIv2

Data Type	Description
Integer32	Same as INTEGER type from SMIv1
Counter32	Same as Counter type from SMIv1
Gauge32	Same as Gauge type from SMIv1
Unsigned32	An unsigned integer value on 32 bits
Counter64	A 64-bit unsigned integer similar to Counter32 type

Notes

- Neither SMIv1 nor SMIv2 have a corresponding type for IP version 6, 128-bit addresses.

2.3.5 PROTOCOL SPECIFICATIONS

The SNMP is an application layer protocol from the TCP/IP stack. SNMP uses the connectionless User Datagram Protocol (UDP) service to transmit SNMP messages. SNMP uses the simple UDP transport service. That does not guarantee either delivery or correct sequencing of delivered packets, so that SNMP can continue functioning after many other network services have failed. By default, UDP port 161 is used for sending and receiving requests and port 162 is used to listen for SNMP traps.

The advantage in using UDP consists in having a low overhead and therefore minimum performance degradation due to management traffic. In the mean time the bandwidth of the network can be more appropriately used for the purpose it was meant.

The main disadvantage is, of course, the unreliable nature of UDP. This means the SNMP itself must handle any lost messages through time-outs and retransmissions. Usually this is not such a big problem since SNMP is expected to encounter networks with problems, otherwise at least fault management would not be necessary.

More recently, Internetwork Packet Exchange (IPX)-based networks have added support for SNMP.

2.3.5.1 PROTOCOL OPERATION

SNMP sends operation requests and responses as SNMP messages. An SNMP message consists of an SNMP protocol data unit (PDU) plus additional message header elements defined by the relevant RFC. An SNMP agent sends information in known two situations:

- When it responds to a request from an SNMP manager
- When a trap event occurs

SNMP version1 specifies the following message types:

Table 2.9 SNMPv1 Operations

Message	From/To	Description
Get	Manager/ Agent	Accesses and retrieves the current value of one or more MIB objects on an SNMP agent.
Get-Next	Manager/ Agent	It browses the entire tree of MIB objects, reading the values of variables in the MIB sequentially. Typically, you use *Get-Next* to obtain information from selected columns from one or more rows of a table. *Get-Next* is especially useful for browsing dynamic tables, such as an internal IP route table or an ARP table, reading the table one row at a time.
Set	Manager/ Agent	It changes the current value of a MIB object. In order to update a MIB value on the SNMP agent, the SNMP manager must have write access to the object. *Set* is used infrequently, because most MIB objects are read-only by default, so that unauthorized changes cannot be made.
Get-Response	Agent/ Manager	It replies to a *Get*, *Get-Next*, or *Set* operation.
Trap	Agent/ Manager	Notifies the specified SNMP manager (the trap destination) when an unexpected event occurs locally on the managed host. You can use traps for limited security checking (such as notifying the trap destination if the agent receives an information request from an SNMP manager that it does not recognize) or for troubleshooting (such as notifying the trap destination if the WINS service fails).

Notes

- The object ID (OID) parameters that are sent through the SNMP messages are referred to as variable bindings. These are pairs (OID, values) meaning that the object specified by the OID has the value next to it. In the case of *get* and *set-next* commands, the value field is left empty.

SNMP version 2 added the following new message types:

Table 2.10 SNMPv2 operations added to the already existing ones in SNMPv1

Message	From/To	Description
Get-Bulk	Manager/ Agent	It retrieves data in units as large as possible within the given constraints on the message size. *Get-Bulk*, which accesses multiple values at one time without using a *Get-Next* message, minimizes the number of protocol exchanges required to retrieve a large amount of information. To avoid fragmentation, restrict the maximum message size to a size smaller than the path maximum transmission unit (MTU), the largest frame size allowed for a single frame on your network. Typically, when it is not known how many rows are in a table, *Get-Bulk* is used (rather than *Get-Next*) to browse all rows in the table.
Notification	Agent/ Manager	The message type is the same with *Get* and *Set* messages; SNMPv2 no more uses a special message type to send trap (notifications) but instead has a new object type that used with ordinary messages.
Inform	Manager/ Manager	It is used for manager-to-manager communication when in a network with many NMS; it can be also used to send traps.
Report	N/A	The operation was never implemented; now is part of SNMPv3 specifications for communication between SNMP engines.

2.3.5.2 SNMPV1 MESSAGE FORMATS

SNMP version 1 messages contain two main parts:

- A message header, which is common to all messages and independent of the message type
- A Protocol Data Unit (PDU) that contains command specific parameters

Figure 2.14 SNMPv1 message format

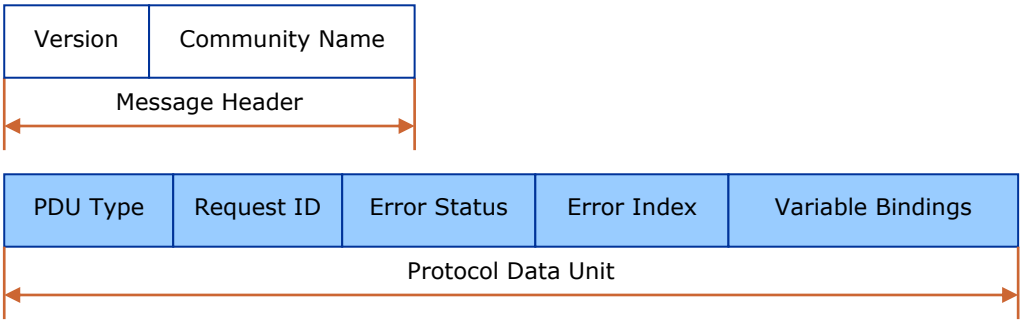


Table 2.11 SNMPv1 message header fields

Field	Description
Version	Specifies the version of SNMP used. For SNMPv1 the value is 0.

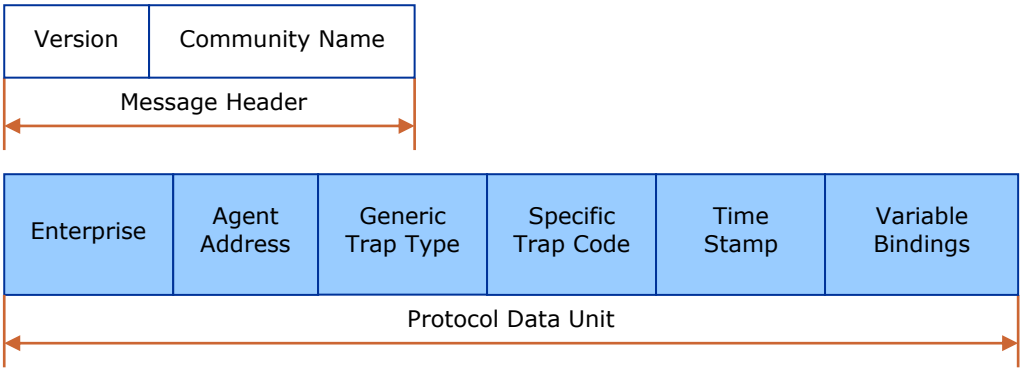
Community Name	Defines the access environment for a group of NMSs. NMSs within the community are said to exist within the same administrative domain.

Table 2.12 SNMPv1 message PDU fields

Field	Description
PDU Type	Specifies the type of PDU the message contains.
Request ID	It is used to distinguish among outstanding requests. This value is used to correlate outgoing requests with incoming replies, and because SNMP is used over UDP, can be used to track duplicate message.
Error Status	It indicates one of a number of errors and error types. Only the *get-response* operation sets this field. The other messages do not use it and set its value in zero.
Error Index	Ti associates an error with a particular object instance. Only the *get-response* operation sets this field. The other messages do not use it and set its value in zero.
Variable Bindings	Represent the data field of the SNMP message and contains the OID, value pairs described in the previous note.

SNMP version 1 makes distinction between manager polling messages and agent traps by providing a unique message format for trap messages. Therefore, the message type presented so far is used for *get*, *get-response*, *set* and *get-next* operations. The message type used for *trap* operations is shown in figure 2.15.

Figure 2.15 SNMPv1 trap format



For trap messages, the PDU contains the following fields:

Table 2.13 SNMPv1 PDU fields for *trap* messages

Field	Description
Enterprise	It identifies the type of managed object generating the trap.
Agent Address	It is the address of the managed object generating the trap.
Generic Trap Type	It indicates one of a number of generic trap types.

Specific Trap Code	It indicates one of a number of specific trap codes.
Time Stamp	It is the amount of time elapsed between the last network initialization and the generation of the trap.
Variable Bindings	Has the same significance as for the other SNMPv1 messages.

2.3.5.3 SNMPV2 MESSAGE FORMATS

The structure of the SNMPv2 message is similar to SNMPv1. In this case, too, the message is composed of two parts: the message header and the PDU.

The SNMPv2 message header has the same two fields Version Number and Community Name with the same role, except the Version Number now contains the value 1 for SNMP v2c and the value 2 for SNMP v2p and v2u.

The SNMPv2 PDU is of two types, depending on the transmitted message.

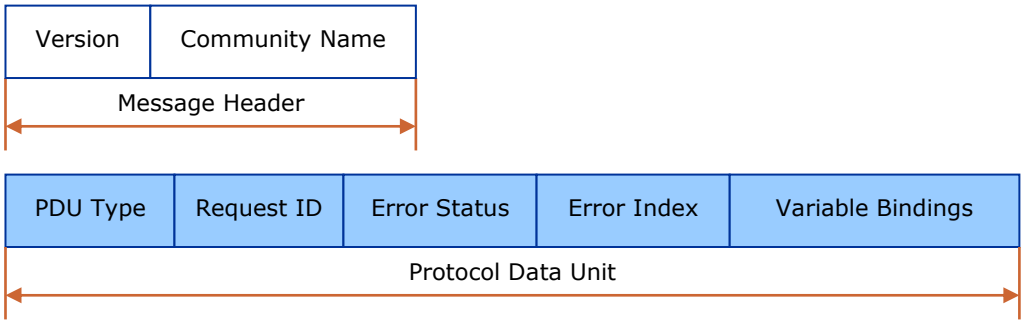
For *get*, *get-next*, *set*, *get-response*, *notification* and *inform* operations we have:

Table 2.14 SNMPv2 PDU fields

Field	Description
PDU Type	Identifies the PDU transmitted.
Request ID	Has the same role as for SNMPv1.
Error Status	Has the same role as for SNMPv1.
Error Index	Has the same role as for SNMPv1.
Variable Bindings	Has the same role as for SNMPv1.

The protocol structure is shown in figure 2.16.

Figure 2.16 SNMPv2 message format



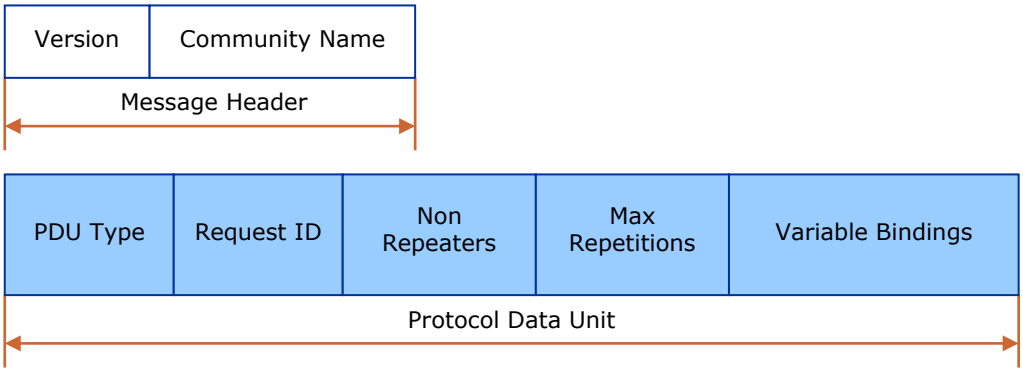
The operation in the second version of SNMP that has a new message format is *get-bulk*. The fields from the PDU are:

Table 2.15 SNMPv2 PDU fields for *get-bulk* message

Field	Description
PDU Type	It identifies the message as a *get-bulk* operation.
Request ID	Has the same role as for SNMPv1.
Non Repeaters	It specifies the number of objects that should not be retrieved more than once from the beginning of the operation.
Max Repetitions	It specifies the number of additional objects to the ones already specified by *Non Repeaters* that should be retrieved.
Variable Bindings	Has the same role as for SNMPv1.

The protocol structure is presented in figure 2.17.

Figure 2.17 SNMPv2 message format for *get-bulk* operation



2.3.5.4 SECURITY IN SNMPV1 AND SNMPV2

In both SNMPv1 and SNMPv2, the (most implemented) security method is the one based on communities. While there is a version of SNMPv2 that brings user-based authentication, there was not widely implemented.

Community names are simply unencrypted text strings used as access mechanism for the SNMP messages and are only used to verify that the agent that is pending the request can be trusted. The disadvantage is that anyone who gets access to the community string can send configuration messages or requests for information, as just the messages would originate from a valid manager. That is why is said that SNMPv1 and v2 offers virtual no authentication methods at all, being vulnerable to a wide variety of security threats.

These threats may include:

- Masquerading – consists in an unauthorized entity attempting to perform management operations by assuming the identity of an authorized manager.

- Modification of information – consists in the attempt of an unauthorized entity of altering a management message sent by an authorized manager in order to perform an unauthorized operation.
- Message and timing modification – occurs whenever an unauthorized entity reorders, delays, or copies and later replies a message sent by an authorized entity.
- Disclosure – is the attempt of an unauthorized entity of extracting values from the managed objects or learning of management events by monitoring the communication.

The greatest weakness of SNMP has long forced vendors to support only monitoring capabilities on their devices, leaving the controlling operations to be performed via direct logon to that particular device. However, solutions can be still found yet. On more advanced and configurable machines such as servers and workstations, to help prevent interception of SNMP messages one may configure the use of Internet Protocol Security (IPSec) policies – doing so will encrypt all SNMP data. The operation must be done on both agents and managers.

The entire security problem has been solved completely by SNMP version 3, which includes support for encryption, advanced authentication, authorization and time stamping in order to prevent capture, reading, forgery and delay of SNMP traffic.

2.3.5.5 INTEROPERABILITY SNMPV1 – SNMPV2

The SNMP interoperability issues were discussed previously in this paper, when the introductory notions about proxy agents were presented. Therefore, this paragraph only tries to emphasize the tasks of a proxy agent in order to solve the incompatibility problems between the two versions of SNMP.

As presently specified, SNMPv2 is incompatible with SNMPv1 in two key areas:

- Message formats – SNMPv2 uses different headers and protocol data units (PDUs).
- Protocol operations – SNMPv2 standard defines four new protocol operations (from which three are implemented and only two are different) that did not existed in SNMPv1.

A common scenario is the one where in the already existent infrastructure with many devices that support only SNMPv1, the network is extended with SNMPv2 devices while the NMSs are upgraded as well. In such a case, the role of the proxy agents is the following:

- To forward get, get-next and set messaged unchanged between the NMS and the SNMPv1 agent.
- The get-bulk messages are converted into a sequence of get-next messages and sent to the SNMPv1 agent.
- SNMPv1 traps are simply mapped to SNMPv2 traps. This is done with the corresponding change in message format and object type.

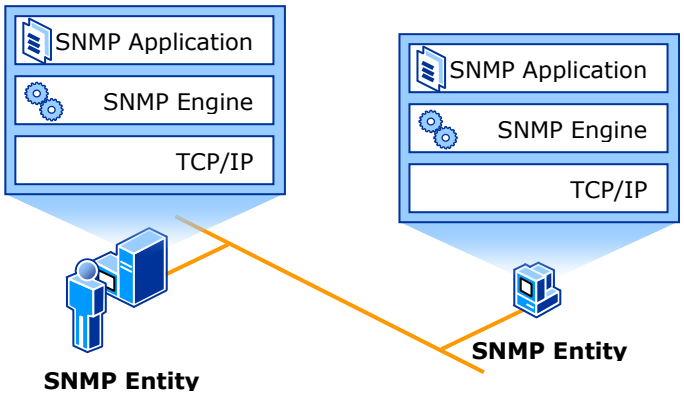
2.3.5.6 SNMP VERSION 3 FRAMEWORK

The SNMPv3 specifications were approved by the Internet Engineering Steering Group (IESG) as full Internet standard in March 2002. The SNMPv3 specifications were previously approved by the IESG as draft standard in March 1999.

The SNMPv3 primarily adds security and remote configuration capabilities to SNMP. However, its developers have managed to make things look much different by introducing new textual conventions, concepts, and terminology. The standard now describes the overall management architecture going in deeper detail on how SNMP-enabled devices should be implemented.

In the new management framework the concepts of agents and managers no longer exists. Instead, the term of SNMP entity is introduced, which could be an agent a manger or both. A SNMP entity consists of a SNMP engine and several SNMP applications. More about the structure of an SNMP entity and the engine and application concepts will be presented in the following section. These concepts are important because they define architecture, rather than simply defining a set of messages. In addition, the architecture helps to separate different pieces of the SNMP system in a way that makes a secure implementation possible.

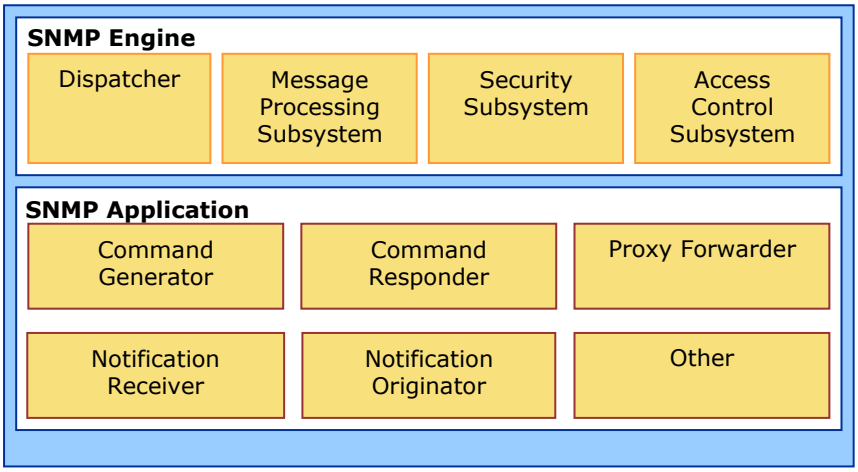
Figure 2.18 SNMPv3 framework



2.3.5.7 SNMP VERSION 3 ENTITIES

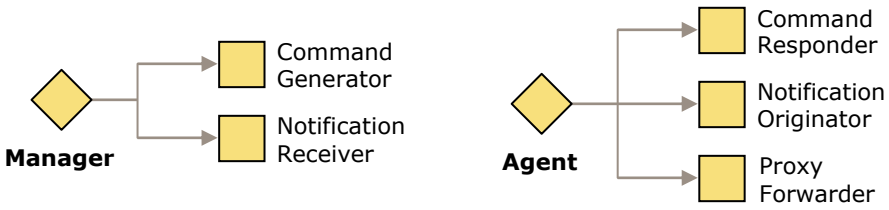
The structure of a SNMPv3 entity is presented in figure 2.19.

Figure 2.19 The structure of a SNMP entity in the SNMP v3 framework



SNMP managers and agents in the traditional way have the same SNMP engine used for SNMP communication, while the applications are specific for the purpose of each of them, like depicted in figure 2.20.

Figure 2.20 SNMPv3 applications for traditional managers and agents



The SNMP engine has the role of sending and receiving SNMP messages, similar to any manager or agent from the old conceptual framework. It provides services like authentication, encryption and access control. It consists of four main components:

Table 2.16 Components of the SNMPv3 engine

Component	Description
Dispatcher	Has the role of sending and receiving SNMP messages. It determines the version of the protocol used, and if the version is supported it sends the message to the Message Processing Subsystem (MPS).
Message Processing Subsystem	Prepares the message in order to be sent or extracts data from the received ones. It consists of several processing modules, one for each supported version of SNMP. This gives flexibility, separates the different methods of solving the same tasks and makes the architecture open to some other modules, yet to be defined.
Security Subsystem	It handles the authentication and privacy of SNMP data. It may be as well of different types, implementation dependant. For example, SNMPv3 specifications already have support for community string-based security (the one from SNMPv1 and v2) but also for the SNMPv3 specific user-based authentication.
Access Control Subsystem	Controls access to the MIB objects. The Access Control Subsystem the authorization method based on communities that already existed. Different permission settings (like read-only, read-write) can be enabled on different objects or parts of the MIB tree.

SNMP applications use the SNMP engine services, and give the already known functionality of SNMP. We have:

Table 2.17 Components of a SNMPv3 application

Component	Description
Command Generator	Generates *get*, *get-next*, *get-bulk*, and *set* requests and processes the responses. Its functionality is manager specific.
Command Responder	It responds to *get*, *get-next*, *get-bulk*, and *set* requests. This application is implemented by the traditional agent.
Notification Originator	It generates SNMP traps and notifications. Its behavior is agent specific.
Notification Receiver	It receives asynchronous messages. It is usually implemented by traditional managers.
Proxy Forwarder	It facilitates message passing between entities and provides functionality similar to the old proxy agents.

2.3.5.8 SNMP VERSION 3 MESSAGE FORMAT

The SNMPv3 message has, like the previous two versions, two parts: a message header and a protocol data unit. The thing that has not changed is the PDU, which is the same like in SNMPv2, but it is subject to the encryption security features of SNMPv3.

However in the message header, SNMPv3 comes with 13 (11 new) fields important for identification, authorization and privacy protection.

Figure 2.21 SNMPv3 message format

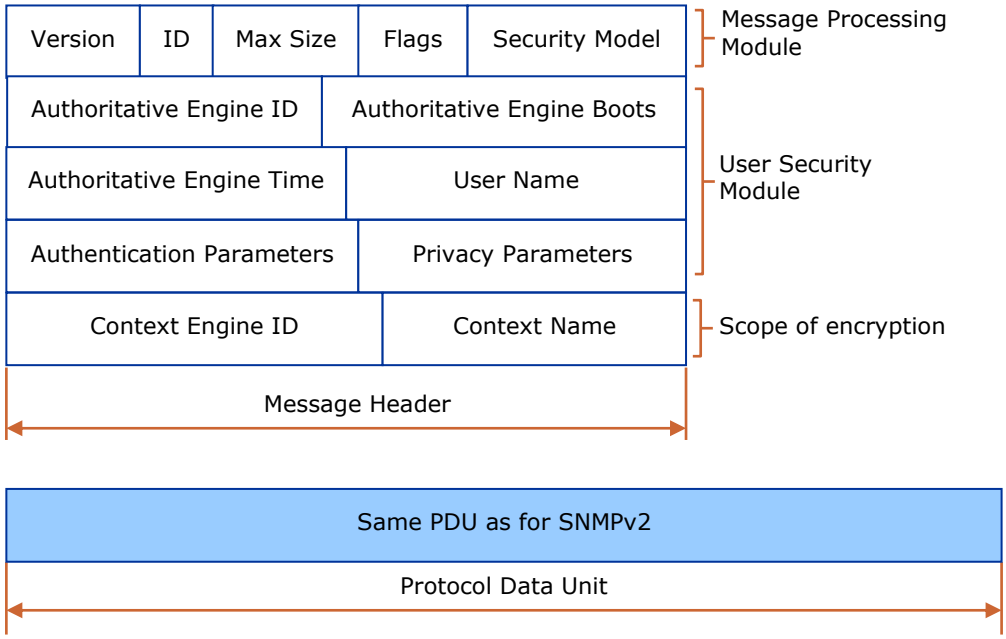


Table 2.18 Fields of the SNMPv3 message header

Field Name	Description
Version	Has the same significance and has the value 3.
ID	It is a unique identifier used between two SNMP entities to coordinate request and response messages (similar to *Request ID* from SNMPv1 and SNMPv2).
Max Size	It is the maximum size of a message in bytes supported by the sender of the message.
Flags	A byte that specifies whether a report PDU is to be sent and whether privacy, authentication or both is used.
Security Model	It specifies the security model to be used. Up to date three security models are supported: SNMPv1 and SNMPv2c, based on community strings and SNMPv3 with the user-based security.
Authoritative Engine ID	It is the ID of the SNMP authoritative engine involved in the exchange of the message. This value refers to the source of *trap*, *inform* and *report* messages and to the destination of *get, get-next, get-bulk, set, or inform*.
Authoritative Engine Boots	The number of reboots the authoritative engine has performed.

Authoritative Engine Time	It is the number of seconds since the entity last reboot.
User Name	The message is being exchanged on behalf of the user.
Authentication Parameters	It indicates which of the authentication method is used.
Privacy Parameters	Depend on the privacy module that has been selected.
Context Engine ID	It is the object ID of the SNMP engine generating the message.
Context Name	The context name in the view-based access method to which the message refers.

The first five parameters are generated by the Message Processing module, the next six by the User Security module, while the last two define the scope of encryption.

2.3.5.9 SNMP VERSION 3 SECURITY

The role of the security framework in the SNMPv3 is meant to meet the following requirements:

- To determine if a message reached the destination unaltered and in a timely fashion way.
- Whether the requested operation can be performed with the user credentials, it came.
- To identify the objects those are accessed by the pending operation.
- To determine the permissions appropriate to the incoming message based on its origin.

The responsibility of the two requirements is taken by the Security module, while the Access Control module handles the last two.

SNMPv3 has support many security models, now available or yet to be developed. Further more the specifications allow the use of several models at one time, since the SNMPv3 messages carry in their header the security model they use. However to maintain interoperability one security model must be implemented by any SNMPv3 compliant device: the User-based Security Model (USM).

It provides support for the following security-related operations to be performed:

1. Authentication - user-based authentication uses MD5 and SHA algorithms to authenticate users without sending a password in the clear.
2. Timeliness – protects against message stream modification or delays by time stamping each SNMP message.
3. Privacy - the privacy service uses the DES algorithm to encrypt and decrypt SNMP messages. Currently, DES is the only algorithm used, though others may be added in the future.

This topic closes our discussion in theoretical fundamentals. The next chapter will explain how some of the basic concepts presented here are used by the software implementation of the Network Measurement System management interface.

3

DESIGN AND EXPERIMENTAL RESULTS

3.1 NETWORK MEASUREMENT SYSTEM FOUNDATION

3.1.1 SYSTEM ARCHITECTURE

The Network Measurement System allows one to perform a various number of QoS measurements while bringing some additional new features to improve the overall efficiency. If referring, to the type of networking tests described in the previous paragraph, one could see that the need for a remote connection to every machine involved in the test is gone. The data is available automatically, online or offline, depending on users' choice, though maximizing either the quality or the number of obtained results.

Several numbers of enhancements of the classical approach makes possible for the network engineer to focus on the core concepts of the measurements he has to perform, rather to find solutions without he could not have made any tests at all.

While the next paragraph will present some of the general features of the software, in the sections that follow, it will be presented what is the concept of the NMS.

The Network Measurement System tries to combine network measurement software and network management software in order to perform network measurement. Though the concept is simple, you will see that allows for much greater flexibility than a regular approach with the tools already available. The Network Measurement System (or NMS in short terms) is made up of two pieces of software:

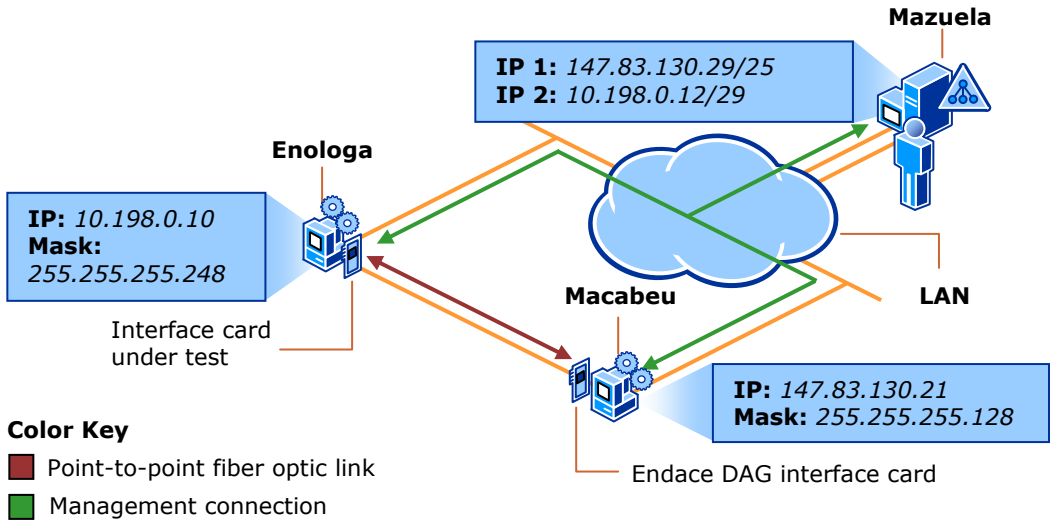
- A software agent - the terms was actually borrowed from the management field to be consistent to the fact that it is not a stand-alone application. The agent is the core part of the measurement infrastructure. All traffic related measurements start and end at the agents.
- Managing software – this would be part on which this document is focusing on. The managing software is its essence the part of NMS that puts the agents on working together. In addition, the software that has the most interaction with the user – using the management software a user can schedule up tests, run them and wait for data to come back. It also features an advance management platform to make any operation performed as smoothly as possible.

The basic topological architecture of NMS implies one or more agents installed on the machines where the measurement tests are being performed, and regular one (but can be many) managers installed on the machine from where the network engineer supervises the entire experiment.

Referring to the experimental setup made up of *Enologa*, *Macabeu* and *Mazuela* from the previous example, the new approach using NMS will look like in figure 3.1.

Looking at the figure one may see two major differences comparing to the figure 1.2. First, there remote connection link has been replaced by a management connection. Nevertheless, as you will see this connection works in a very different way than *ssh* being dedicated only to measurement related tasks, rather to be used for a remote console access. The second difference is that now *Mazuela* has two IP addresses, one in each subnet of *Enologa* and *Macabeu*. No intermediate routing machine is required or allowed for this experimental setup. Remember, that on the previous experiment one would have connected remotely to the router, than called *Celler* and form that remote connection would have made the final connection to *Enologa*.

Figure 3.1 Experiment setup for testing network interface card using NMS



Important

- One should not make the confusion that for a given number of agents installed on several subnets; the management station requires a separate interface card in each subnet. In the example above, this step is necessary since the management station needs to access an agent installed in a private network. If the *Enologa* agent were installed in a subnet with routable IP addresses, the second interface on *Mazuela* would not have been required.

The tests themselves follow the next outline:

Install and configure two NMS agents, one on *Enologa* and one on *Macabeu*.
Install and configure the NMS management software on *Mazuela*.
Enable the management software to use both interfaces to subnetworks 147.83.130.0 and 10.198.0.0.
Enable from the management user interface to use both agents from *Macabeu* and *Enologa*.
Define test sessions on the management application to be performed.
Schedule the sessions and expect the results.

Even though the overall process does not look much simpler that in the previous situation, until the end of this document you will see that using the Network Management System is a more straightforward process. Now, just for the beginning the next topic outlines some of the features of the NMS management application.

3.1.2 FEATURES OF THE NETWORK MEASUREMENT SYSTEM

The Network Measurement System has been designed to provide a user friendly way to perform QoS related measurements. After several agents have been installed in the points of interest and a management station has been designated, the basic setup for get one started is already completed.

The advantages that come with the use of the NMS are presented in the next table.

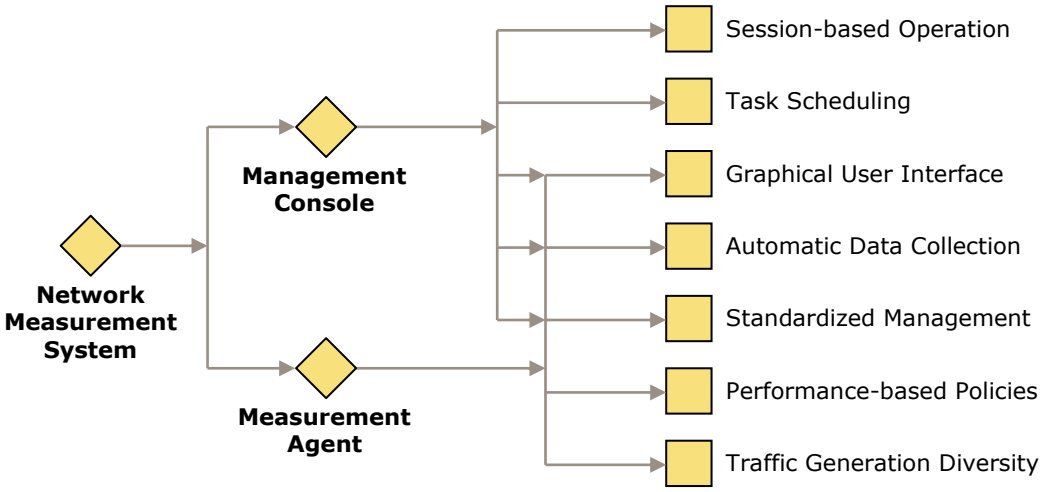
Table 3.1 Features of the Network Measurement System

Feature	Description
User Interface	The Network Measurement System features an accessible user interface on both the manager and the agent part. The manager is built under an administrative console that enables the user to perform almost all operations, either local or remote. Little effort is required to configure each agent alone, and this is done mainly one time only.
Session-based Operation	From the end-user concern, the NMS is session oriented, meaning that the entire experimental activity could be organized as individual sessions. In addition, one may combine several sessions in a session-group, the session-group containing several either independent or related sessions. The session group for independent session was designed for easy management. The IT engineer would no longer care about multiple tests that need to be performed in a specific order. Instead, he can organize the entire testing activity as sessions and group them together into a session group. Creating a group with a series of related sessions can overcome, however, the disadvantages of the existent measurement tools. One could create a group in which the same session is executed several times almost, the same, except for a user specified parameter that receives values in a user-defined range and it updated at each time with a user-specified increment.
Task Scheduling	Task scheduling means greater flexibility in the performed work. Using task scheduling the user could schedule a given session or session group at a specified date and time, without constantly looking after the program. This option can be used especially when measurements need to be performed at specific intervals, without requiring an operator to supervise the program at all times.
Automatic Data Collection	Combined with the previous feature of task scheduling the automatic data collection provides a powerful way in which the NMS software can do its job automatically. Automatic data collection is enabled by default. It means that for every experiment you run, depending on the collection policies you selected, results are retrieved from the agents without user interaction.
Performance-based Policies	When a session is defined, depending on its type (traffic generation, traffic analysis or both), the user may specify the level of detail in the information obtained about the measurement test, during and after the test is completed. This means that for the same session, different data could be available at the end. This approach could cope very well for test, at which the user might need high accuracy, rather than a high level of detail during the test. This is because the measurement agent does not have to perform additional work to provide results that are not required, and instead could be oriented on a higher precision in obtaining the final data.
Traffic Generation Diversity	This feature means that for the experiment from figure 1.1 no ARP tables and no IP addresses must be set-up in order for the experiment to work. The NMS agents provide 3 levels of traffic generation, each being suitable for different kind tests: Ethernet / IEEE 802.3 option allows you to generate data-link layer traffic with a compatible network interface card. For a single generation session, the user has full control over the destination address and protocol/type (depending on whether the Ethernet II or IEEE 802.3 is chosen) fields. This option is

	recommended for tests between equipments in the same network segment. Internet Protocol (IP version 4) traffic enables generation of network layer packets. The following fields are user controlled: the destination IPv4 address, the Time-to-Live (TTL) field, the Type-of-Service (TOS) field and the next layer protocol field. This traffic is recommended for tests across multiple hops. In combination with the layer 2 traffic option, one could also test the performance of hardware devices rather than the performance of the operating system, since the encapsulation of this type of traffic has been optimized and is done entirely by the measurement agent. User Datagram Protocol (UDP) traffic is recommended to be used in tests in which the performance of the TCP/IP stack from the operating system up to the interface between the application and transport layers (considering the TCP/IP networking model) is an issue.
Standardized Management	The Network Management System uses Simple Network Management Protocol (SNMP) for management purposes. This, to a certain extent, allows for interworking with other applications that might follow the NMS operation procedure. In addition, using a standardized protocol means the application is ready and open to any changes, allows easy, quick debugging and provides the foundation for future improvements.

Table 3.1 summarized the overall features of the Network Measurement System. Some of the features presented there hold on the management infrastructure of NMS, others are related to the traffic engine implemented on the NMS agents, while some of them can be found in the implementation of both the management console and the distributed agents. The following figure depicts the distribution of the NMS features throughout its major components.

Figure 3.2 Distribution of major features throughout the NMS components



From the features above, the ones that correspond to the measurement agents will not be covered in this document. For more information regarding those features, including additional attributes and technologies that are specific to NMS agents refer to [2].

The list of features presented before covers the most important aspects of NMS in the light of the final objective of the NMS software, i.e. enabling users performing their network measurement tasks, faster, easier with an improved response time and a lot of burden relief from the post-processing task that would have been normally required. However, in addition to those features there are several other topics, specific to the management infrastructure. These topics refine on one hand the level of detail in understanding what NMS management offers you, and what are the background technologies involved in having the work done.

The next paragraph covers these additional features in more detail.

3.1.3 OTHER FEATURES OF THE MANAGEMENT INFRASTRUCTURE

Although the list of features described in this topic could not be interesting from the end-user point of view, they make a good starting point in understanding how NMS management actually works. Furthermore, starting with the next topic this document will start presenting the internal architecture of the NMS management starting from this list of features. Therefore, discussing them here makes a good opportunity to end this introductory part.
The next table summarized the features of NMS management and afterwards they will be covered in more detail.

Table 3.2 Features specific the NMS management

Feature	Description
Client-Server Architecture	Provides the foundation on which the entire management infrastructure is created, both at the management console and at the agent parts.
Configuration Service	Provides services to store and retrieve user defined configuration data. The service runs at all times and cannot be disabled.
Simple Network Management Protocol version 1	The management service uses SNMPv1. SNMP provides transport of management related data and commands between the management console and the agent applications.
Measurements Specific Management Information Base	The management infrastructure defined a custom management information base (MIB) that provides a larger flexibility in the commands available between the NMS components, reduced overhead and quicker response.
Priority-based Message Queuing	The management service relies on an advanced, double-bounded message queuing mechanism.
User-controlled Services	The NMS management console features a user-controlled Service Control Manager (SCM) able to start, stop, pause or resume during the run-time some of the management related services. This is extremely useful if a configuration change requires a restart of the management service, since it enables the user to keep the console running at all times, thus keeping some data that cannot be saved.
Event Log	The built-in event log is the most useful tool that could be used for troubleshooting and debugging. If something goes wrong, if a test fails or an agents stops responding, check the event log. Several hundred errors, warnings or information messages can help you in most every scenario to determine what has happened and act appropriately.
Plug-and-play Hardware Interface	The PnP interface enables the user to collect as much information as possible on the status of the hardware equipped either locally or on the remote agents. This interface is available for all network interfaces providing you with detailed information about the manufacturer, model, serial number and status data for physical, data-link and network layer protocols.
Identification Protocol	Within the NMS applications, an identification protocol is transmitted over SNMP such that any NMS application that requests a management connection to another NMS application is automatically identified.

Notes

- Since the management infrastructure through its services is implemented both at the management console and at the measurement agent sides, some of the features in table 3.2 can be found on a measurement agent. However, since those features are

always a subset of the features available on the management console, even when discussing about some specific feature as belonging to the manager, one should consider that the same feature could be encountered on the agent application. In this document, it will be specified as much as possible the features that are available on both the management console and the measurement agents.

Regarding the list from the table 3.2, one may find some additional comments useful.

First, regarding the client-server architecture, one should not make the confusion to the networking client-server terminology. The client-server architecture of NMS management simply means that on the same computer where the management service is running, either under the process of the management console or of a measurement agent, the tasks are divided between different independent services. Some of these services provide a service directly to the user, which is always considered the client. Instead, these services could be clients for other services that perform even lower level tasks. All these services are linked between interfaces that enable the exchange of data across them.

The next major topic that will discuss the architecture of the management console should provide a better insight on what the client-server approach of the NMS management means. For now, just as an example, imagine that a user wants to perform a measurement. The following enumeration appoints some of the steps that are performed in the background.

1. Usually measurement related tasks are handled by a service called Session Manager that keeps track of all measurements the user performs in order to dispatch properly the results, when they become available.
2. However, the Session Manager relies on another service, namely the Scheduling service, which takes the user command from the Session Manager, and decides whether the command should be executed.
3. When the Scheduling service decides that it should do so, sends the command to another service called SNMP Wrapper service. According to its name, this service finally interprets the command of the user and generates the appropriate parameters for a SNMP message to be created. The SNMP parameters are then placed in a queue in order to be transmitted on the network interface.
4. The mentioned queue is handled by the obviously called Queuing service. Its main job is to make sure that if two transmission requests come simultaneously (i.e. one from the user and one from some other task that executed in the background; for instance), both of them have the chance to be transmitted. As you will see later in this paper, the queuing service relies on many other components. For example, a priority mechanism ensures that some of the messages are transmitted first even if they arrived last, while others that are not so important are transmitted whenever there is a moment available. Another important issue is the synchronization of queue operations, since you may wonder what if both the message from the user and the message from the background tasks are placed in the queue simultaneously. All these along with other interesting topics shall be discussed later in this text.
5. For now, once the message is inside the queue, we suppose that at some time it should be transmitted. Before the message is encapsulated in datagrams, frames made up of bits sent over the wire, someone must take the SNMP message parameters from the queue and create the SNMP message as defined by the standard. The SNMP service (obvious name again) takes care of that.
6. After the SNMP PDU (protocol data unit) has been created, a networking service takes the message and sends it to the TCP/IP implementation from the operating

system. From now on, it is up to operating system to handle its transmission further on.

The short scenario presented here was intended to explain the concept of client-server used by NMS management.

Important

- In computer-science literature, some other terms can be found that describe the concept presented here, such as skeleton and stub. For additional information that explain the concept but more in detail search for terms related to distributed computing, Local Procedure Call (LPC) and Remote Procedure Call (RPC). Just as a remark, the client-server management architecture uses a modified type of LPC, where the internal transport protocol varies between different types of services.

Caution

- One should never make the confusion between the networking client-server term and the computing client-server terminology used in this document. From the networking point-of-view, even though the two main applications of the Network Measurement System are called manager and agent, both use almost identical networking software. In addition, since the networking transport is provided by the User Datagram Protocol (UDP), both implementations act simultaneously a server and client, and therefore can be considered as peer-to-peer. Manager-to-manager, manager-to-agent and agent-to-agent communication modes are supported, except the first and last one are not regularly used.

The management infrastructure uses the Simple Network Management Protocol version 1. SNMPv1 has been chosen on behalf of several reasons, from which the following were the most important.

- SNMPv1 is considered a relative stable standard compared to SNMPv2 for which several sub-versions exist.
- Even if it lacks the security features of SNMPv3, since most of the usage of NMS is done in a test environment, the security issues were not considered too critical. For more information please read the paragraph about the security issues of SNMPv1 implementation.
- Using SNMPv1 means less overhead in management transmissions over the network. Compared with SNMPv3 it also means less processing time, since no data is ever encrypted. This is extremely important on the agent software, where the traffic generating modules must be scheduled in real time as much as possible.
- The configuration of SNMPv1 is also much easier, a community name (essentially a password) and a community right is all that is required. For additional info, follow the introductory part on SNMP presented earlier in this paper.

The theoretical fundamentals explained that one might not discuss about SNMP without also taking into consideration the Management Information Bases. This is because the most important part of the SNMP management is represented by the management data, data that is a collection of objects, where each object represents a status variable on the managed software that is controlled remotely. Since it was already explained that the RFC standard that define the SNMP and management information structure that comes along (essentially, is about the Management Information Bases and the Structure of Management Information) use ASN.1-like object definition, with managed-objects referred to as nodes of a management tree.

The root of the management tree is defined by the Structure of Management Information (the first version of SMI was published in RFC 1155, SMI version 2 in RFC 2578). For a detailed description of SMI and the root of SNMP management tree, review the first chapter of this document.

Because the managed objects used by the Network Measurement System are used with SNMP, it means the application also relies on the structure given by SMI into defining the objects identifiers for each object used in a measurement operation. Since the current version of the Network Measurement System is under experimental status, it would make sense not to register a private class of managed objects under the *enterprise* node.
The implications of using the *experimental* branch of the SMI tree have two consequences:

- The objects identifiers used by NMS are neither standardized nor unique. This means that the latest version of NMS is intended only for experimental and research reasons, and does not support a production environment deployment.
- If one intends to use NMS in a real environment measurement, it has to register an object identifier class number (see note below) and afterwards upgrade the managing software such that the root of the managed tree used internally by NMS is changed to the private registered class.

Notes

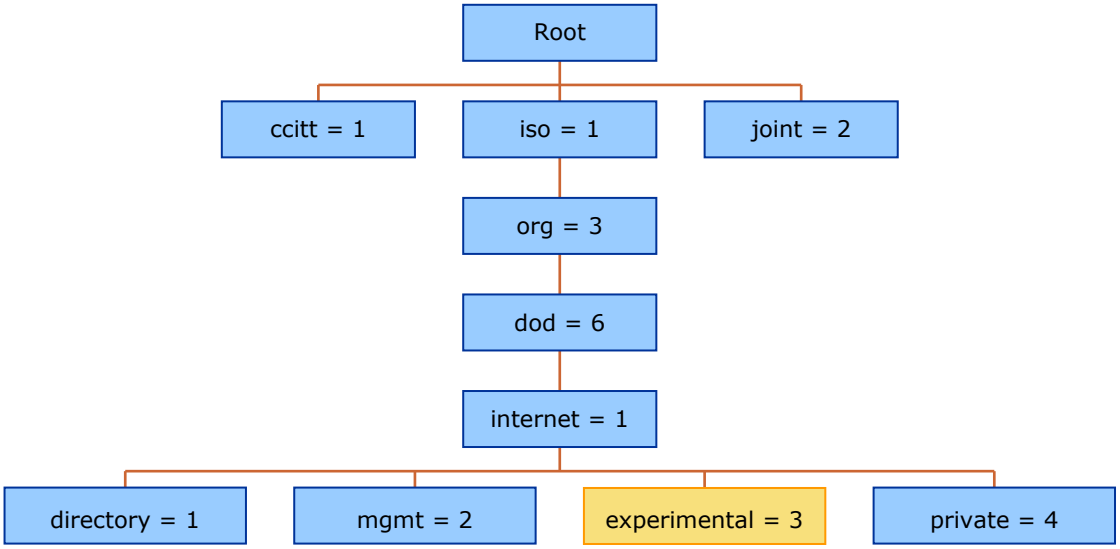
- The enterprise object identifier numbers are currently managed by the Internet Assigned Numbers Authority (IANA). Any private corporation can register free of charge a private OID number. The assignment rule is first come first served. In order to register a private enterprise number one could visit the IANA web site at http://www.iana.org or visit directly the online registration form at http://www.iana.org/cgi-bin/enterprise.pl. The denominated and numbered prefix of the private enterprise numbers is *iso.org.dod.internet.private.enterprise* and *1.3.6.1.4.1* respectively.
- The NMS development project had at no time the goal of registering a special class of managed objects for use with the software. The registration of such a class in order to make NMS usable over large networks such as the Internet is the responsibility of the user. The NMS is designed to support the change of its managed object identifiers with little changes made to the software.

The figure 3.3 depicts the prefix path for the managed objects used by NMS. This path prefix is *iso.org.dod.internet.experimental* or *1.3.6.1.3*.

The managed objects defined for use with NMS were divided in several sub-classes. Briefly, they provide basic management options (such as identification, general information and networking information), traffic control options, traffic engineering settings and results. All classes of managed objects in the *experimental* branch will be covered later. For now, the important issues regarding NMS MIB remains that an experimental set of managed objects is used, for a production environment deployment the assignment of private enterprise prefix number is highly recommended, since the migration is very easy to implement.

The priority-based message queuing service is one of the core pieces of NMS management, not as logical part but rather as an operational part. The basic services are message queuing and multiplexing, message flow control – meaning that simultaneously incoming sending requests are processed one by one. In addition, priority queuing ensures that critical messages are processed first. The basic and advanced concepts on the queuing service will be thoroughly described further on in this paper.

Figure 3.3 Management tree path of the managed objects identifiers used by the experimental version of NMS.



User controlled services ensures that the engineer that handles the NMS management console has control upon the execution of some background services. The most important reference in this case is the management service, which incorporates all management related processing such as sending and receiving SNMP PDUs. When this service is stopped, the management console is restricted in either sending or receiving such messages. The user has the possibility in starting and stopping such service, this being especially useful in either preventing the console to receive SNMP data from unwanted SNMP applications, when the measurement system is not used or to make possible a restart in the situation a configuration that requires a restart is performed, without the need of restarting the entire application. The NMS console does not save all the data regarding the tests it performs so having some of the services restarted while keeping the software running is an obvious advantage.

The event log and PnP interface will be covered in a special paragraph dedicated to these topics. Shortly, the information contained in the event log can be used debugging whenever a problem occurs. The plug-and-play interface is used to provide detailed information about the networking hardware installed both on local and remote machines. Information such as the equipment name, manufacturer, capabilities and configuration is available to the user.

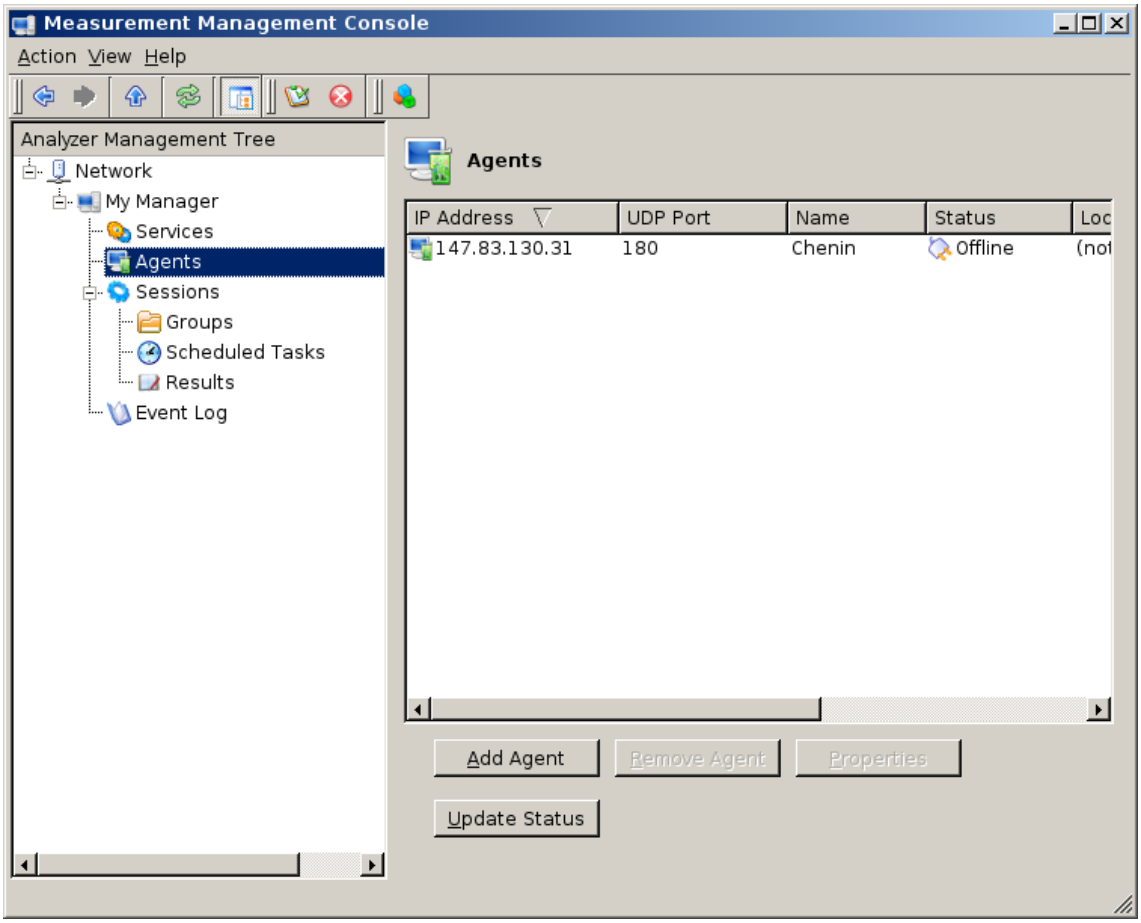
The identification protocol is an extension of the measurement MIB used by NMS management that helps identifying a NMS application as a management console or a measurement agent. Since it was stated earlier that the management infrastructure does not make any difference whether the application is a manager or an agent (the internal processing is identical) the identification protocol uses a set of managed objects that are used by NMS applications to learn more on each other and their capabilities. For example, while using the *Add New Agent Wizard* if the remote application is a NMS manager or no NMS application at all, the wizard will simply ignore it and inform the condition to the user.

The next topic outlines the basic piece of NMS management from the users' perspective, the administrative management console.

3.1.4 THE ADMINISTRATIVE MANAGEMENT CONSOLE

Despite the technologies used, the number of services, queuing mechanisms, task scheduling and management protocols use, the administrative management console or the management console on short, is the central piece of the NMS management. However many agents one will deploy the management console is the software that puts them together. From the users' point of view, their work with NMS will be most of the time focused on the use of the management console. Figure 3.4 depicts a general view of the NMS management console.

Figure 3.4 The management console



The intention of this paragraph is not to be a complete user guide of the management console. A chapter dedicated entirely to the usage of the NMS console and how measurements are performed follows later in this document. For now, it is important to outline the role of the management console and some of its more important features.

The management console is used to manage the set of NMS distributed agents with the objective to perform a set of measurement tests, collect the measured data and display the results back to the user. However, the NMS management console has several additional roles, from which the most important are setting up the configuration of the management service, installing agents, i.e. the console acknowledges that a remote agent exists and it is running and could display to the user information about that agent. In fact, the figure 3.4 shows the agents pane view open, where one could identify an agent (called *Chenin*) in the list by its address, port, name, status and other additional information. The process of installing an agent is called registration and will be discussed later.

Other roles of the management console are, of course, the creation of measurement sessions and session groups as well as using these to create tasks. Tasks can be scheduled and only when a task starts executing the agents involved are contacted by the console and the operation request along with the parameters is transmitted. During the execution of a task and at the end of it, the console may collect online and offline data from the agents that perform it, processing the data into results and make them available to the user.

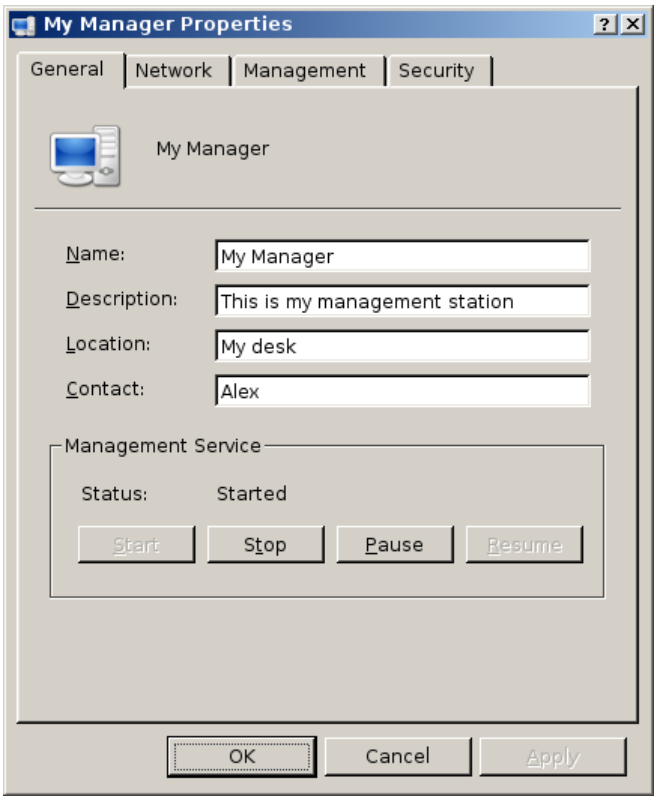
At last, the results availability and mode of presentation is what one appreciates the most. The management console features different ways to present the results of a measurement. Beginning with an inspection of up to date information received from the agents, one may select to view a summary of a session (results displayed here are obviously more reliable only after a test was completed), detailed results in time domain or parametric results for a parametric session group. A specially designed trace plotting window is available to either inspect the date or save it for publication. Along with the option that any data available can always be saved in a *csv* (or *comma separated values*) format, this makes it great for later processing using supplementary tools.

Some key-point features of the management console are:

- Object-oriented user interface
- Linked tree-view pane-view display format
- Wizard guided assistance
- Dynamic shortcuts toolbar

Each of these features will be thoroughly examined in the chapter dedicated to the usage of NMS.

Figure 3.5 The manager's properties dialog



The last three features of the console are intuitive and for a basic understating, do not need additional information. Therefore, only the object-oriented user interface concept will be explained briefly. The term of object-oriented implies that most items in the console's interface are regarded to as objects with which can interact. Such objects are the items from a list or from a tree for example. In the view of the console from the figure 3.4 both the items form the agents list at the right or the contents of the tree from the left are regarded as objects. The advantage of the object-oriented approach comes from the fact that most users are used in manipulating pictograms or items as an abstract representation of a real entity. The most important, most of us associate such user interface objects with a set of parameters that can be displayed or even modified.
The management console follows this belief. The most common thing that comes along with an object, is (what most of us get used to) the object's properties dialog. Because each item from the management console is an object, it means that it always have an associated properties dialog where the user can found more. The figure 3.5 shows the manager's properties dialog. The manager as concept is the entirely management software, the management console included. However, because the manager software has a name, one or more associated IP addresses and other information, it should be treated like an object. Actually, it also has an associated item in the console's tree view from the right: the second item from the top. In the context of the figure 3.4, the name of the manager is displayed, *My Manager*.

One can click the *Properties* option from the *View* menu (or use the toolbar shortcut instead) to open the properties dialog seen in figure 3.5. There much more information about the manager is displayed, usually organized by topic in successive tabs. Since some of the object properties from the properties dialog can be edited, means that the user can change those properties and the new values will be applied as soon as the user hits the *OK* or *Apply* button.

3.1.5 APPLICATION FILES AND SETUP REQUIREMENTS

This is the last introductory topic on the management of Network Measurement System with some information about the binary and configuration files required for NMS management console operation.

The following table summarizes the setup files of NMS management console.

Table 3.3 Files of the management console

File Name	Size	Timestamp	Description
mmc	2,922,158 bytes	Wednesday, May 17, 2006, 3:59:24 PM	The executable of NMS management console
config/global.cfg	299 bytes	Wednesday, May 17, 2006, 3:59:25 PM	Startup configuration of NMS management console

The files in the table 3.3 make up the minimal configuration required to run the administrative console. Any configuration other file that might be used can be created automatically if missing. Those file are presented in table 3.4.

The first file, *mmc* represents the executable binary of the NMS management console. The console GUI and the management related services run under the same user process. By starting *mmc* the console itself along with some basic services are automatically started with other services starting on demand. The default configuration of NMS (i.e. without having NMS

configured at all) does not start the management service. After the manager has been configured, the default policy is to start the management service automatically, but only after the application is restarted.

Notes

- The default configuration settings will be presented in the chapter about using the NMS management console. The services of the management console will be covered later in this chapter, in the topic entitled *The Architecture of the Management Console*.
- The required restart of the application in order to start the management service when the management console is configured is because the startup parameters of each service, including the startup type, which could be automatic, manual or disabled, are applied only at application startup or service startup. Therefore, if you configure the management console, the application needs to be restarted to start the management service, even if the startup type is automatic by default.

Table 3.4 Other configuration files used by the management console

File Purpose	Description
Toolbar Configuration	It contains the user-defined configuration of the console's toolbars. This file is updated whenever the user changes the toolbar settings.
Manager Configuration	It contains the configuration of the manager, such as name, interfaces used, UDP ports assigned, SNMP-related information and security.
Queuing Configuration	It contains the configuration parameters specific to the queuing service.
Management Configuration	It contains the configuration parameters specific to the management service.
SNMP Configuration	It contains the configuration parameters specific to the SNMP service.
Session Manager Configuration	It contains the configuration parameters specific to the session manager.

All configuration files, except the global configuration file, are binary files. The user can make changes directly to those files (i.e. without using the management console) only if he/she has enough knowledge on the structure of each file. The binary structure of the configuration files will be presented in the paragraph dedicated to the *Configuration Manager*.

Caution

- It is highly recommended to use only the management console in the update of the binary configuration files whenever possible. An editing mistake in those files may have serious effects and in some cases could even render your computer useless. Use manual editing of the binary configuration files with appropriate caution, and only when necessary.

The global configuration is the only file required to exist in order to run the NMS console. The default values from all other files are known automatically by the application even though the files may be missing. The contents of the global configuration file and how to appropriate edit its contents can be found in the section devoted to the *Configuration Manager*.

In conclusion, the number of files required to run the NMS management console is extremely low, only two. Some additional folders may be required but that is it. Any other file needed can be created at runtime without adverse effects to the user.

Important

- In the case of missing configuration files, which is the case when the application is first used, or when the configuration files have been deleted the application's event log may report a missing configuration file error. The user should concern these types of errors only if the management console has been already configured. Otherwise, one may simply reconfigure the console, or use it as it is while ignoring the error message.

The following table summarizes other files that can be created at run-time by the management console. Some of these files are temporary and are created for the internal use of the application alone, while others may keep database information, such as the event log.

Table 3.5 Other files used by the management console

File Purpose	Location	Extension	Description
System Event Log	./data	EVT	It records informational, warning and error events when they occur. The data from this file is made available through the Event Log service.
Session Group Data	./data/session	LST	It contains information about the flows that were associated with the sessions from a session group.
Session Flow Data	./data/session/flow	DAT	It contains flow-based session data. This data is used to compute the results for a measurement session.
Session Files	Not specified	SES	Contains session parameters saved by the user. Using session files, one may save a session and reuse it at its convenience.
Spreadsheet	Not specified	CSV	It enables user to save session results in a format that is easy to be imported in other spreadsheet applications, like Microsoft Excel.

The following table summarizes the file paths that are used by the management console. These paths are usually created at installation time.

Table 3.6 Paths used by the management console

Path	Mandatory	Description
./config	Yes	By default, it contains all configuration files. At least the global configuration file must be present at this location.
./data	Yes	By default, it contains the system event log. However, the user may specify other location.
./data/session	Yes	It contains the session group files. This location is always used.
./data/session/flows	Yes	It contains the session flow files. This location is always used.

./sessions	No	It is the default location where users save their session files.

The directory paths that are specified as mandatory should always be created before any attempt to run the management console.

The current version of the management console is intended to run on GNU/Linux based operating systems (the kernel version used was 2.6.15-1-486). However, the application can be compiled for other platforms or versions. In this case, additional libraries may be required. The core of the management console is built by following the POSIX standards. The user interface, which is platform specific, may have difficulties across platforms. The next table contains the environment recommended settings in which the current version of NMS was thoroughly tested.

Table 3.7 Recommended platform settings for NMS management console

Resource	**Recommended Settings**
Operating system	Debian GNU/Linux
Kernel version	2.6.15-1-486
Networking	IP version 4
Graphical user interface	X Window and K Desktop Environment version 3.5

The next section will introduce the architecture of the management infrastructure, emphasizing the extra features that are added by the management console as well as how the management interface is used on the agent software.

3.2 THE ARCHITECTURE OF THE MANAGEMENT INFRASTRUCTURE

3.2.1 GENERALITIES

The management infrastructure of the Network Measurement System provides the entire set of management services in order to allow remote control of the NMS agents from a single management console. Some features of the management infrastructure such as client-server modularity were already discussed in the previous paragraphs. Therefore, this section will focus on presenting the internal structure of the management infrastructure and how different technologies are being put together to yield the final application.

It was already shown that the management infrastructure is placed at the foundation of both the management console and the distributed set of agents. Therefore, first we shall focus on presenting the underlying technologies that are found in both management modules, while in the end emphasizing some of the key differences: what are the features that are found in the management console (which is obviously oriented on management alone) are missing from the agent implementation.

The next paragraph presents the internal structure of the management software and in the sections that follow each module will be presented in detail.

3.2.2 INTERNAL STRUCTURE

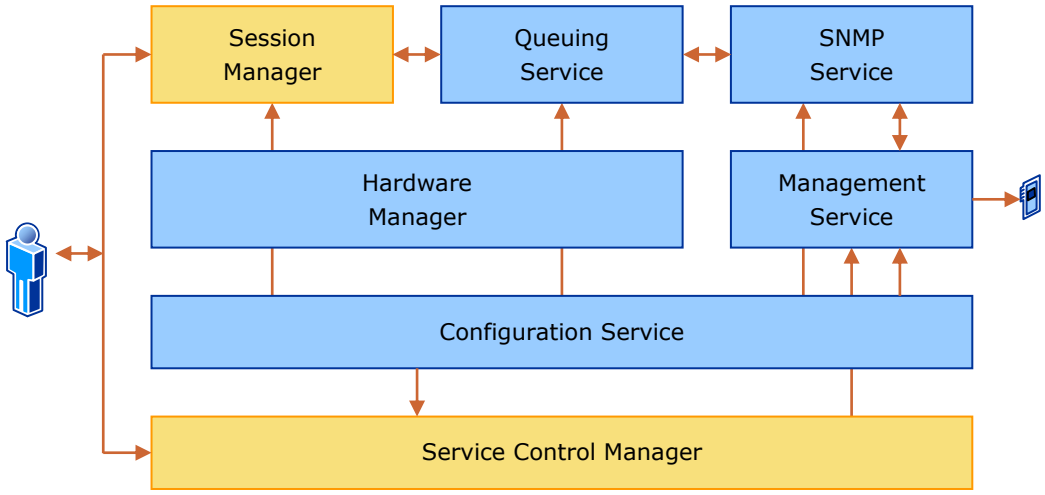
The goal of the management infrastructure is to provide two things: an interpretation of the user's commands and the reliable transport of those commands to the proper agent where they will be interpreted and the appropriate agent software agent will act accordingly.

The interpretation of user's commands is done only at the management console and it means more than just reading the user input and generating immediately a corresponding output. The agent also features two user interfaces, one graphical based and the other command-line base, but both lack the functionality of what the management infrastructure does for the management console. The user-interactive part of the management software is designed to allow the users to setup in minimal steps a measurement test into a session or session group, schedule it, and care nothing more about it. A special service within the management infrastructure will perform the task and even collect the results and make them available for display when the test is over.

Nevertheless, most view as most important objective of the management infrastructure the one of creating management messages and sending them reliably over the network. These messages contain control, status or informational data. A complex mechanism was designed in order to ensure the transmission of those messages with a high degree of reliability. The management message encapsulation and transmission will be covered shortly.

The functionality of the management infrastructure is divided in several operation units called services. As explained, these services interact with each other, hence the client-server architecture. The figure 3.6 shows the coarse splitting of functionality of the management infrastructure between different components. Later, that diagram will be refined, meaning that some components will be analyzed in more detailed, including the algorithms that lay at the foundation.

Figure 3.6 The architecture of NMS management infrastructure



In the figure above, the two components of the management infrastructure that are exclusively found in the management console are depicted yellow. All other components are found on both manager and agent platforms, concluding that the overall management structure is symmetrical, and peer-to-peer from the networking point of view. This means, that at the basic level, it does not matter that functionality the application has (either management purposes or measurement purposes).

From the figure above a key service is also missing: the Event Log service. The Event Log service runs all the time and ensures that error; warning and informational messages are recorded for troubleshooting and recovery procedures. The Event Log is accessible via a series of library functions that allow writing and reading of event messages, plus the inspection of some event log parameters. Since, the Event Log service is outside the scope of the management infrastructure and is provided for other reasons it should be treated separately. A section dedicated to this topic will cover later in more depth the Event Log.

Coming back to the figure, starting from low level of functionality, a short description of each service might be necessary.

The management service handles networking functions, encapsulation of management messages. Although the job of the management service might seem a little simple at the first glimpse, one should take into account that the management service has to provide networking operations for any selected local interface. Usually the management messages are encoded in a single stream up to the management service level and only there the decision upon which message is transmitted on each interface is taken. This decision is based on a destination address and port pair that travels along with message, and which is originally set-up by the session manager (which knows which agents are used with each test). However, since for the blocks in between the destination is rather transparent, i.e. the queuing service does not care which the destination of the message is for example, it could be said that the management service performs message de-multiplexing. The process is also made the other way around and in both directions so one could say that the management service performs bi-directional multiplexing of messages.

The management service also implements security functions by filtering the IP addresses of inbound messages, according to user-defined list. The management service relies on another

service, i.e. the SNMP service in performing message encapsulation. The following list summarizes some of the characteristics of the management service:

- It provides UDP-based networking services.
- The local interfaces and ports to be used are selected via the Configuration Service.
- It relies on the SNMP Service for message encapsulation.
- It uses the Queuing Service in order to send and receive messages to the upper processing layers of the management console.
- It provides security through IP-based filtering and SNMP community checking.
- It is user-controlled via the Service Control Manager (SCM).

The SNMP Service handles the SNMP message capsulation. Its two main functions are to create a brand new SNMP message, given the SNMP parameters, and to identify an incoming message as SNMP extract the SNMP parameters from it. The management service extracts messages from the outbound queue, uses the SNMP service to create the protocol data units and sends them via UDP. On the other hand, the management service listens for incoming UDP data; it sends the data to identify it as SNMP PDUs and if so to return the SNMP message parameters.

The processing algorithm performed by both the management service and the SNMP service will be described in the sections that follow.

By SNMP parameters, one should understand the values of most of the fields from a SNMP message. If you remember the SNMP structure presented in the first chapter, the community name, PDU type, request ID and all other fields are possible candidates as SNMP parameters. The paragraphs that presents the SNMP service in more depth, explain which are the actually parameters returned.

By summarizing, the SNMP Service performs the following.

- It creates SNMPv1 messages, given the data to be put in the packets.
- It verifies that a received data stream is a SNMPv1 PDU.
- If a valid SNMP PDU is received, it parses it and returns the data from the message.
- It performs the ASN.1 encapsulation of each SNMP PDU field.

The job of the Queuing service is to ensure that multiple simultaneous incoming or outgoing management messages can be processed. Actually, the queuing in any system is a way in which the limited resources can be used in order to satisfy all service requests. Since it may happen that at the same time the Session Manager might send two management messages, how the software that implements the lower functions ensures that both messages are transmitted, having a limited bit rate on the physical network link. The answer is to send on of the message first, and put the other one on hold until the first is successfully transmitted. If more messages happen to be sent at the same instance, one is transmitted, while the others wait in the queue.

The Queuing service implements a more robust and reliable method to ensure message queuing. First of all some messages might be more important and should be sent the first, or otherwise said should be placed in the front of the “queuing line”. For this purpose, the queuing service has up to usable queues in each direction, each of them having a different priority level. When the management service waits for messages, it first checks the queue with the highest priority, then the queue with the second priority level and so on. Only after the most important messages were transmitted the not-so-important messages have the chance of being transmitted and only at the end, the non-important messages are transmitted.

Other function of the queuing service is to handle message retransmissions and discarding the duplicated messages. This process is however more complex in order to be described here on short, and will be covered completely in the section devoted to the presentation of the Queuing service.

Briefly, the roles of the queuing service are the following:

- It provides message multiplexing, on both inbound and outbound directions.
- It establishes priorities on the messages placed in the queue.
- It handles retransmission for messages requiring an acknowledgment from the other SNMP entity.
- It ensures that duplicate messages that arrive within a specific interval, called the *duplicate discarding interval* are thrown away.
- It performs queue recycling, meaning that the messages that stay in the queue for a duration larger than the *queue-recycling interval* are removed from the queue, because most probably there is no higher-level software to read and process the messages.

The Session Manager is the highest-level software routine from the management infrastructure. It is implemented only at the management console and it performs measurement related management functions. For the user, the interaction with the software means setting up test by creating sessions, session groups and scheduled tasks. However, the application sees these as creating corresponding management messages and processing the answering management messages that most likely contain the results.

Therefore, the role of the Session Manager is to make this translation from user session view of test to the management view of the tests, i.e. SNMP based PDUs. It has several mechanisms in order to make the proper translation not only for the messages that are sent, but also for results that are received to ensure that the proper results go to the proper session result sheet.

Shortly, the roles of the Session Manager would be:

- To execute the measurement tasks scheduled by the user by translating those tasks into the appropriate management messages.
- To use the services provided by the Queuing service on order to transmit messages to the appropriate agent.
- To expect replies for the agent to which the messages are sent for a specific task. When the first reply from such an agent is received, it is said that a session was created.
- To redirect the replies to the task that awaits for them in order to process the results.
- To save the results for each task, in order to make them available for later inspection.

The operation of the management infrastructure is done in the framework of two other software components, the Configuration Service and the Service Control Manager.

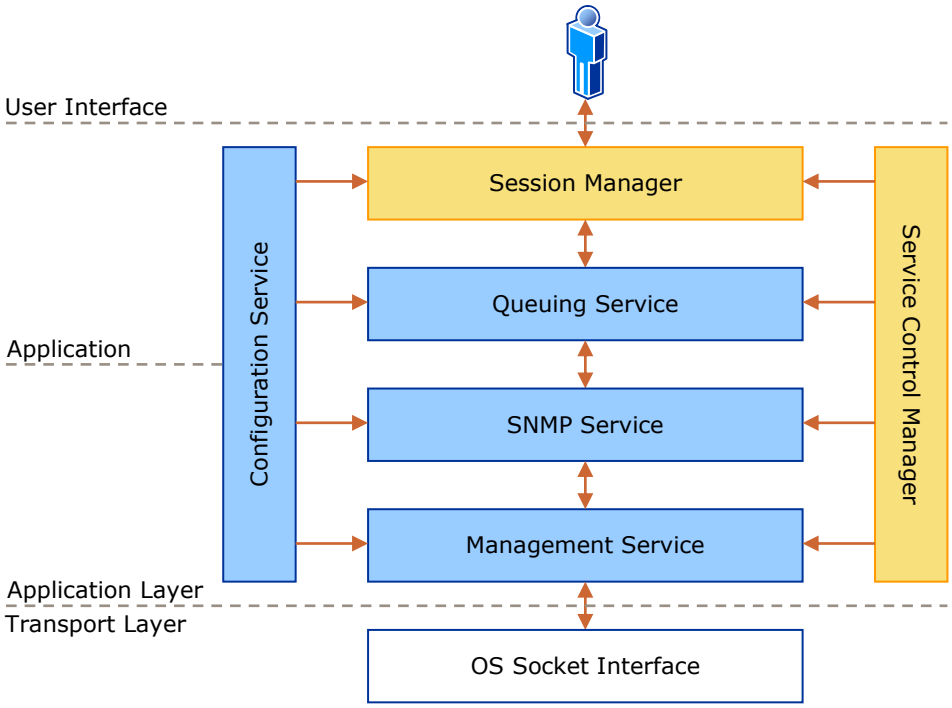
The role of the Configuration Service is to keep the configuration data for all other service components of the management infrastructure. Not only user-configured data but also machine specific data such as the networking devices that are present in the hardware configuration is obtained also via the Configuration Service. Most of the data kept by the Configuration Service is stored onto magnetic media, such that would be available after the console is restarted.

The final architecture block, which is implemented only at the management console, is the Service Control Manager or SCM. The SCM handles all services operation. For each service it maintains the status, such as running or stopped, and for some of them allows the user to change their status, i.e. either stopping or starting a specific service. The SCM also handles service recovery in the situation of a service failure, with several recovery actions being possible to define.

3.2.3 THE LAYERED ARCHITECTURE

This paragraph explains how the services that were briefly presented so far are organized into a layered architecture design. Even from the networking point of view, all services perform application and application layer functions, some of them provide higher-level services, more closely to the user concepts of measurements. Others, on the other hand, have low-level services, such as basic UDP connectivity, encapsulation security. The figure 3.7 organizes the basic services of the NMS management (including those that are found exclusively on the management console) from the user-layer point of view. At the top are found the services that interact more with the user, while at the bottom there are the services more closely related to networking functions. In addition, this layered approach implies the flow of information.

Figure 3.7 The layered structure of NMS management architecture



The agent does not implement the Service Control Manager, meaning that the default behavior of the settings from SCM is used by default. However, without the SCM some features like the service recovery mechanism will not be available. Instead of the Session Manager, the agent software contains other software modules, globally named as measurement engine. The configuration service also exists but it is differently implemented and is provided as a library with a set of global-accessible functions. For detailed information on the architecture of the NMS agent software, you should consult the agent documentation in [2].

The figure 3.8 contains the implementation of the management infrastructure at the agent software. Some details such as the agent's user interface are not covered in that picture.

Important

- You should also keep in mind that the agent's measurement engine implements also lower layers from the OSI networking model, so the drawing the measurement engine block at the top of OSI stack is accurate only from the management point of view.

The two applications, the management console and the measurement agent implement application and application layer functions. The communication at the application layer is realized via SNMP messages and the communication at the application itself is done through session-related information. The figure 3.9 outlines the concept. You can see that the basic informational unit at the application level (and not layer), similar to the concept of PDU, is the measurement session. The section from this document that describes the Session Manager explains thoroughly the sessions, how they are identified based on flow identifiers and how they are correlated to the collected results.

Figure 3.8 The layer structure of the NMS management implemented at the agent

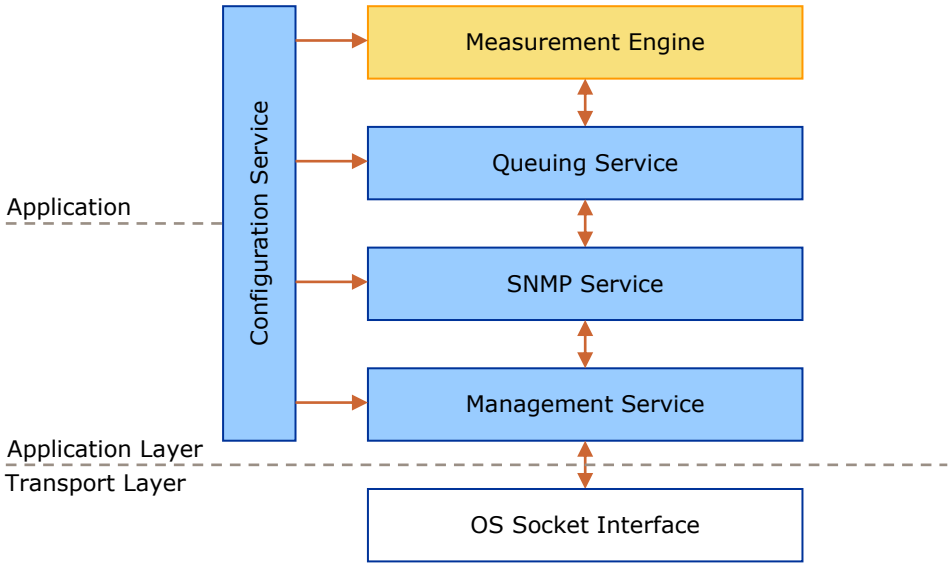
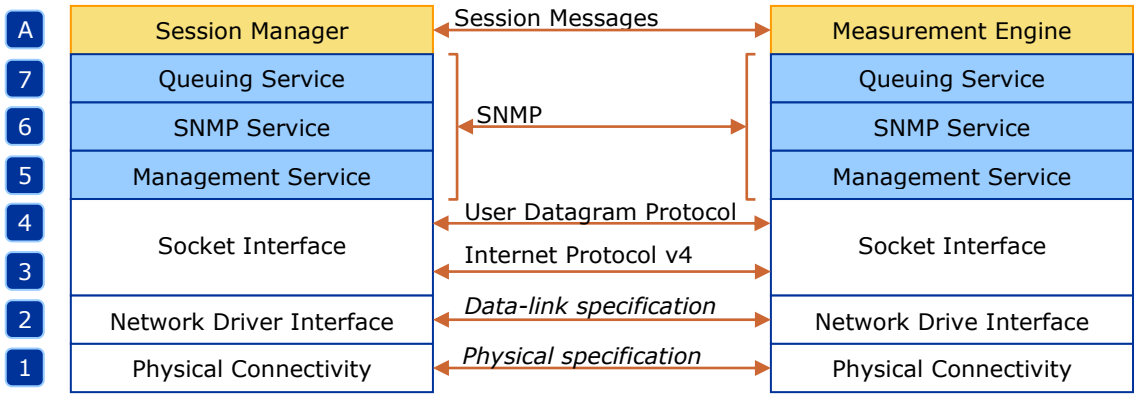


Figure 3.9 Communication layers between the management console and the measurement agent



A few additional remarks about the figure 3.9 are needed. First, the management infrastructure does not specify any physical or data-link specifications, so any layer one and layer two technologies that can be used with IP and UDP is therefore supported (hence, the italic font used

for the layer that do not have specifications). Second, on the left side the figure also contains the distribution of OSI layers over different software interfaces. It can be seen that the management infrastructure implements the layers 5 through 7, i.e. session up to application layer. The digits inside the rectangular blue cases indicate the OSI layer number. The character *A* inside the top case means that level deals with application-to-application communication.

Notes

- If you want to refer to the TCP/IP networking model, instead to the OSI, especially since the SNMP is a TCP/IP hierarchy application protocol, than the changes from figure 3.9 are the following: layers five through seven implemented by the management service, SNMP service and queuing service are defined as the TCP/IP model application layer. Layer 4 could be referred to in this case as transport layer, the layer three as the internetworking layer while the two layers at the bottom are usually considered as the host-to-network layer.

The management infrastructure uses the User Datagram Protocol (UDP) because this is the most used transport option for SNMP. Even though some TCP implementations for SNMP are known to exist, it is not recommended to use TCP for the following reasons:

- The traffic overhead will be too high, especially for a lot of small management messages.
- The SNMP protocol as a management option is usually associated to failures. In networks that work just fine and never fail, there is no need for management. One the other hand, TCP itself cannot overcome network problems like failures, so flooding a link with TCP retransmissions would not be of much help either.

Therefore, UDP usage for SNMP follows the philosophy that if the network works just fine the messages will arrive at the destination anyway. If the network experiences problems to the point where the connectivity between the NMS entities no longer exists, TCP would not be a better choice either.

Using UDP, however, means that the applications must care about messages that could be lost in transit, not just due to a network failure, by means of retransmissions. Nevertheless, if retransmissions are used that implies that it is possible that two identical messages to reach the destination. In this case, the application must properly filter out duplicate messages. In the NMS management infrastructure, the Queuing Service is responsible for message retransmissions and duplicate messages discarding.

As a network layer option, the only one available until this moment is the Internet Protocol version 4 (RFC 791). The IPv6 may be supported on future versions of Network Measurement System.

This paragraph ends the general discussion about the architecture of the management infrastructure. Starting with the next topic, the services of the NMS management console (i.e. including the services of the management infrastructure) shall be examined more thoroughly, including algorithm diagrams, key library functions and configuration parameters. The services will be presented in the logical order, in which they are linked with each other.

3.3 THE MANAGEMENT SERVICE

3.3.1 SERVICE OBJECTIVES

The purpose of the management service is to provide four major things:

- Basic networking connectivity using UDP transport
- Message encapsulation, by relying on the data sent by the SNMP service
- Distribution of messages, by relying on the features of the queuing service
- Security functions, by filtering the incoming management messages using two criteria

From this short enumeration of management service functions one may determine that the layered presentation is not entirely accurate, since not the above layer relies on services provided by the layer below. Actually, in this case, things seem to happen the other way around.

The explanation is that the layered presentation from previous paragraphs was made entirely from the layer functionality point-of-view as defined by OSI. Also to this extent, one may argue that the queuing service is based on services of the management service, since the queuing service waits for management messages to be transmitted and received by the management service, even though, at the programming level the management service calls specific functions of the queuing mechanism. This is also the case with the SNMP service.

Nevertheless, if you look at the sequence in which the data is processed: first, it travels through the queuing mechanism, and then it is encapsulated by the SNMP service and then is transmitted by the management service.

3.3.2 SERVICE STRUCTURE

The implementation of the Management service and by extension the implementation of several other services must take into account the following requirements:

- The execution of the service code must be made independently, meaning that its instructions should execute asynchronously related to other services or user procedures.
- The service flow must be linked with the control procedure of the Service Control Manager, which enables the service to start or stop at the user's discretion.

Notes

- There are services implemented in the management console that do not obey either or both of the two requirements. However, this is because the must perform a synchronous or a requested service, and asynchronous execution it is not possible. Nevertheless, for now, the Management service must obey them, and in what follows we shall describe how the implementation made that possible.

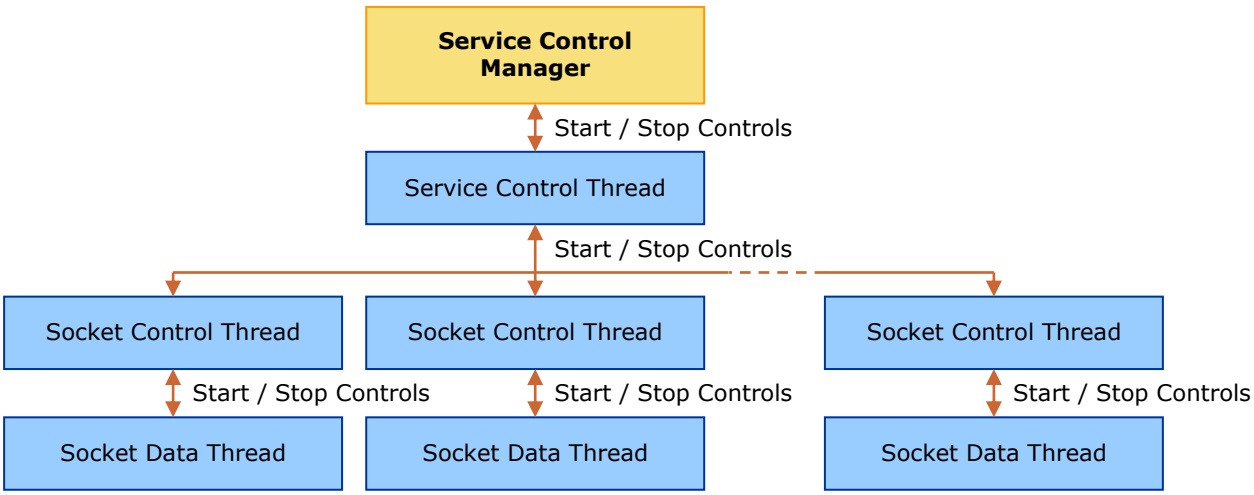
The core of the Management Service is made up of three functions that run independently in three execution threads:

- A control thread that supervises the execution of the whole service, creates and closes the other threads
- A socket control thread that performs basic socket services such as socket opening and closing

- A socket data thread that performs reading and writing of networking data

The figure 3.10 suggests the communication process between the three types of threads.

Figure 3.10 Basic communications between the Management service threads



The Management service features a single Service Control thread but multiple Socket Control and Socket Data threads. This is because the service requires an additional set of threads for each local interface on which the management service is used. After the setup of the management console, at the first configuration the user usually specifies the local interfaces to be used with the management service. In addition, at the measurement agent, the user may choose to use the management over multiple network interfaces, to cope with connectivity with multiple managers in different networks.

At the Service Control thread startup, the service reads the current configuration and creates a number of Socket Control threads to match the number of local interfaces used. Each thread will bind only to the interface to which it is assigned, so management traffic can be sent and received only to and from that interface.

Important

- Because the interface assignment to different threads is done only at the Management service startup, when changing the local interface setting you must restart the Management service for the changes to take effect.

3.3.3 SERVICE CONTROL THREAD

The Management service starts by receiving a start control from the Service Control Manager or SCM. The start control means that the start function of the Management service is called by the SCM and the service control thread is created.

However, prior to the service creation several control variables are set to indicate the execution status in both directions. What this means, it will be explained shortly. After these variables have been properly set, the service control thread can be created. Then it is up to the code inside the thread to indicate that the service has successfully started.

Understating now how the services control is achieved is very important, because on all service that receive controls from the SCM use the same mechanism. This is also true for secondary threads, such as in this situation of the Socket Data threads, that are started in their turn by the Socket Control threads.

Any service thread software uses at least two variables with global access to indicate the execution status and to control the execution flow:

- The control variable is simply used to indicate whether the service thread is allowed to run or it should stop. This variable is always set by the superior instance (or the parent) that controls the thread execution. An example will follow to explain the concept.
- The status or flag variable indicates to parent the status of the service thread, i.e. whether it is running or it is stopped.

For the Management Service two additional variable types are used:

- A global status variable, which indicates more precisely the condition of the service, as a whole entity
- A thread-associated variable that indicates whether the service is in paused state

The algorithm on which the execution flow is based is explained in what follows. The logic diagram is also depicted in figure 3.12. Before continuing, the table 3.8 summarizes the control and status variables used by the Management service.

Table 3.8 Control and status variables used by the Management service

Variable Name	Description
Thread control variable	Indicates from the parent whether the service thread should run or should stop
Thread status variable	Indicates to the parent the current running state of the service thread
Thread extended-status variable	Indicates from the parent whether the service thread should pause its execution (but without stopping)
Global status thread	Indicates to the parent the global status of the service (not of individual threads)

When the SCM sends the start control, by calling the service function mentioned before it first must check the global status thread variable to see whether the service supports the start command. The table 3.9 contains the possible Management service global states.

Table 3.9 Global states of the Management service

Status	Value	Description
Stopped	0	The service threads are not running and all control thread variables indicate the treads as stopped.
Started	1	The service threads are running (all or some of them) and the main control thread is in running state.
Paused	2	The service threads are in the running state, but are placed into a pausing loop in which no operations are performed.

Stopping	3	The service already received a stop control, but the main control thread is still running and is waiting for all child threads, i.e. the socket control and socket data threads, to stop. At the end of the stopping state, the service enters automatically in the stopped state (0) and the control thread exits.
Starting	4	The service received a start control, the main control thread already started executing but it is in the process of initialization and starting the descendent threads, i.e. the socket control and socket data threads. If no errors occur, at the end of the starting state, the service enters automatically in the started (or running) state.

The service accepts the start control only if it is currently in the *stopped* (0) state. In addition, the startup type should not be *disabled* (2).

Notes

- Since the Service Control Manager is not implemented at the agent, the default startup type will be used in that case, which is always *automatic* (0).
- The following startup types are implemented by the Service Control Manager: *automatic* (code 0), *manual* (code 1) and *disabled* (code 2). More information about the service startup types will be found in the paragraphs about the Service Control Manager.

If the service does not accept the start control due to either of the mentioned reasons, it ignores the control and returns an error. At the management console, the SCM uses this error to report an event in the event log.

If the service accepts the start control, it changes the global state to *starting* (code 4). The startup procedure it then changes the control thread status and control variables:

- The running control variable is set to *true* meaning the control thread is allowed to run.
- The status variable is set to *false* meaning the control thread was not yet started.

After this final variable, the service control thread is created. The startup procedure verifies that the thread was created without any errors, and at the end, it waits for the control thread to indicate that it successfully started. The waiting procedure is actually a loop in which the startup procedure holds until the global control thread variable is set for the running (or started – code 1) condition.

The startup procedure performs a synchronous start of the Management service threads. This means that the start function could not block indefinitely the caller thread, if the service threads do not indicate that they started within a limited amount of time. An operation timeout check is used in this case to prevent the caller to hang if something goes wrong. The timeout check code simply adds a countdown counter to the final waiting loop (the one in which the startup procedure expects the service control thread to indicate that it started). The counter is initialized to a value, called *thread operation check countdown* and is decreased by one at any iteration. The iteration duration is not less a parameter called *thread operation check timeout* expressed in microsecond. The status of the service is checked each iteration, to see whether it started.

The conclusions of what was presented so means that the after the control thread was created the startup procedure waits until indicates that it started but no longer than the time of *thread operation check countdown* multiplied by *thread operation check timeout*. The default values are given in table 3.10 and can be changed by the user. In the table, other variables are mentioned, whose role will be explained a little further.

Table 3.10 Default values for environmental variables of Management Service

Variable Name	Default Value	Unit	Description
Thread Operation Check Timeout	10000	microseconds	The duration of loop iteration, that waits for a thread operation to complete
Thread Operation Check Countdown	255	iterations	The maximum number of loop iterations, in which a caller waits for a thread operation to complete
Thread Operation Finish Timeout	500000	microseconds	The interval duration in which the descended threads are expected to stop, when a service stop control was received
Thread Sleep Timeout	10000	microseconds	The minimum duration of an execution loop for a control thread
Thread Read Sleep Timeout	5000	microseconds	The minimum duration of an execution loop for a data thread

Therefore, the default maximum waiting time for the startup procedure to end is 10000 microseconds multiplied by 255, yielding 2550000 microseconds or 2.55 seconds. Usually, if the service startup procedure exits, without having the service in the started state, it is said that the service hanged, i.e. it remained suspended into an intermediary state and it is not known if a stable state should ever be reached.

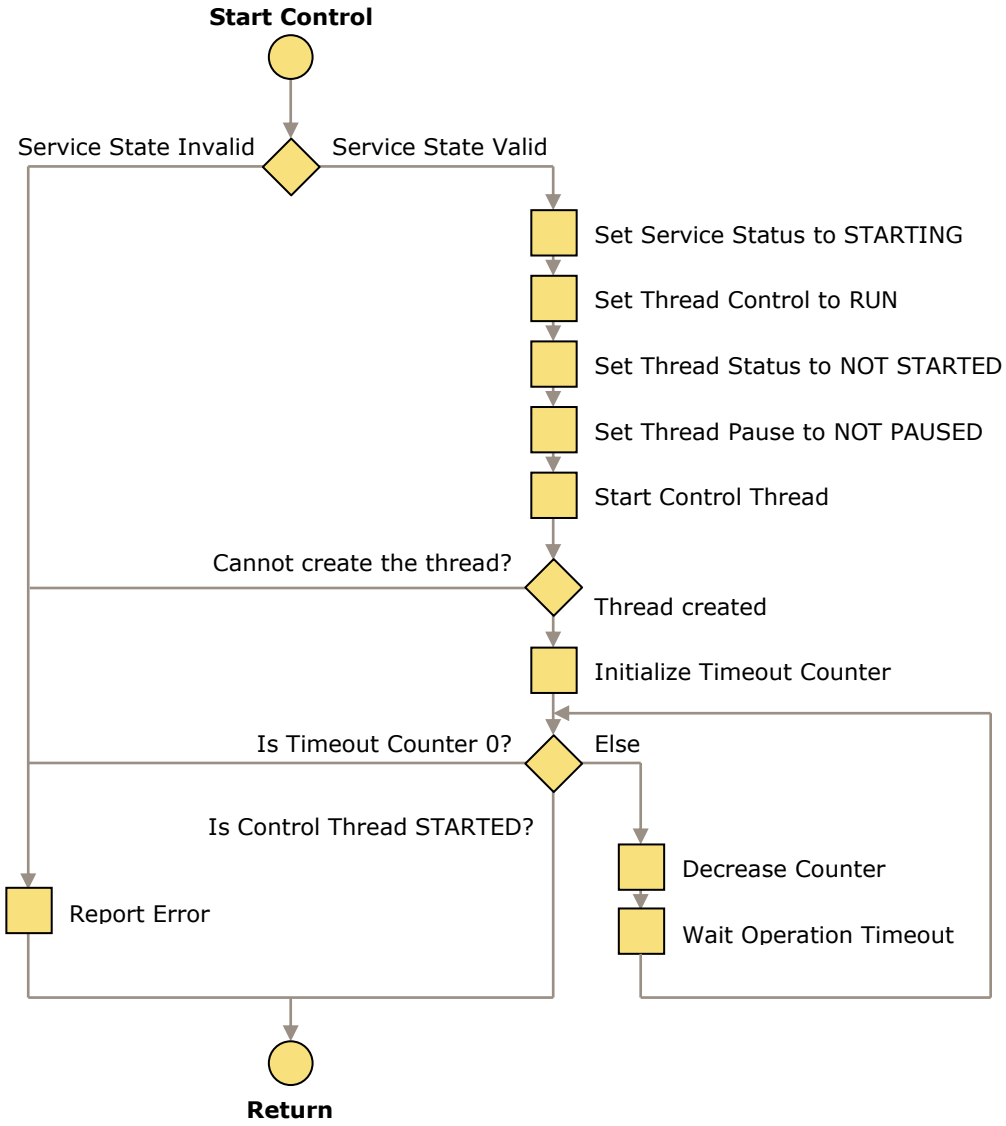
Referring to the startup procedure discussed here, if the service control thread takes more than 2.55 (or other value if configured by the user) seconds to start (i.e. to execute up to the point where the service status variable is set to started) the startup procedure exits leaving the service in the *starting* state. Usually if this happens an event is logged to warn the user, but there is no recovery procedure available, since the service threads can no longer be controlled (because they refuse to change their status).

It is a rare occurrence for something like this to happen due to lock of the service threads to some point, but it cannot be excluded. However, in most cases the service could not start within the timeout duration (of 2.55 seconds, default) because the time duration from its creation until it changing the status, announcing that the initialization completed and it is started, took slightly longer. In such situations, the service will eventually start and all subsequent operation will work just fine. The figure 3.11 outlines the logic of the startup process.

Now, let us look at the execution of service control thread. On one hand, the control thread is responsible of performing some initialization routines prior of setting up its status as running. As promised, the figure 3.12 it is presented the logic diagram of the service threads operation. Nevertheless, before a few explanations are necessary.

After the control thread start it immediately puts its associated status flag to *started*, so any other function it is aware that the thread started execution.

Figure 3.11 The management service startup procedure



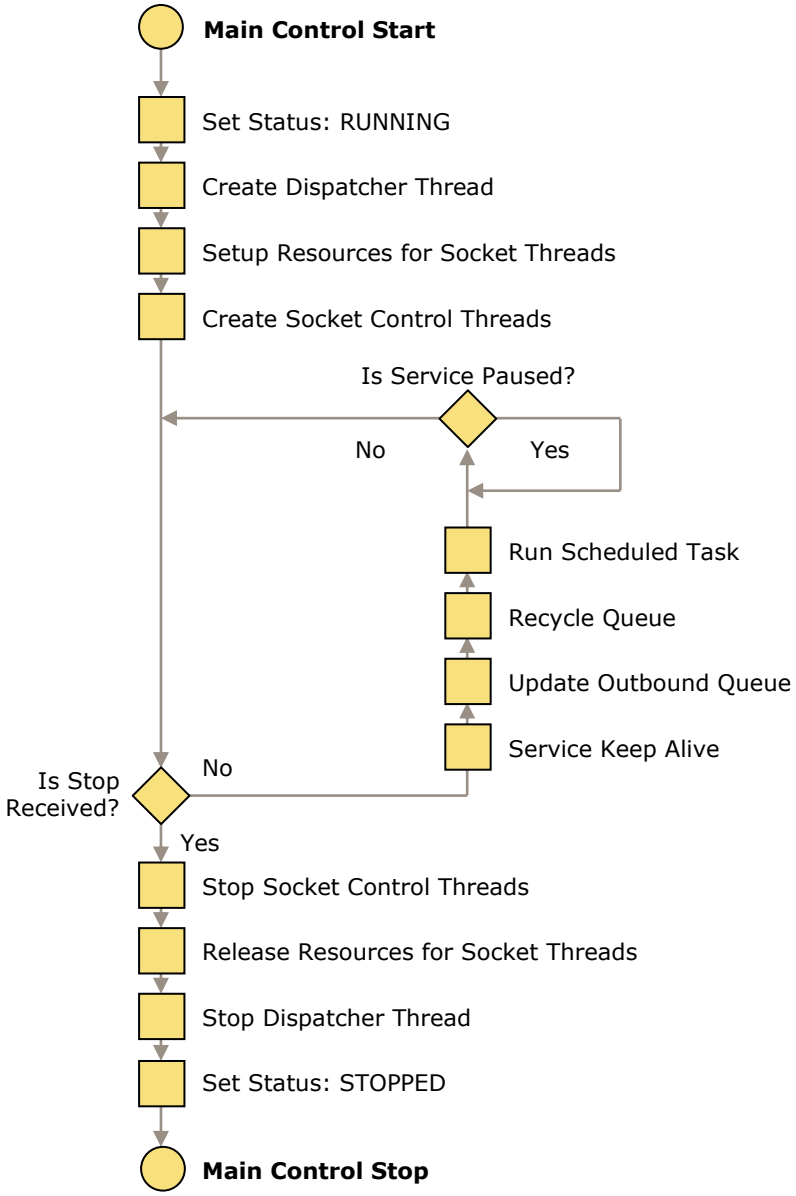
The thread initialization process has several stages:

- To start the dispatcher service thread – this thread will process the management specific requests that are addressed to the management console.
- To allocate the resources required by the socket threads.
- For each local network interface that is selected by the user, to create and start socket control thread.

The dispatcher service thread processes SNMP messages that are received by the management service. However, the dispatcher performs message the operations indirectly, using the queuing service and SNMP service. To make a good picture of what is actually happening, you need to remember that: the management control thread starts a separate thread for each local interface. The role of these threads is to bind to the local interface in order to intercept the incoming management messages. In turn, each also has to create a secondary thread that will perform reading and writing socket operations, hence the socket data thread name.

These final threads will get the data from the network, and interpret it using the SNMP service, to ensure that the received stream of bytes is a valid SNMP message. If the checking is okay, the message parameters will be placed in the queue waiting for processing.

Figure 3.12 The logic diagram of the Management Service



The above figure illustrates in a detail what is the execution flow of the Management service main control thread. One may identify three different stages:

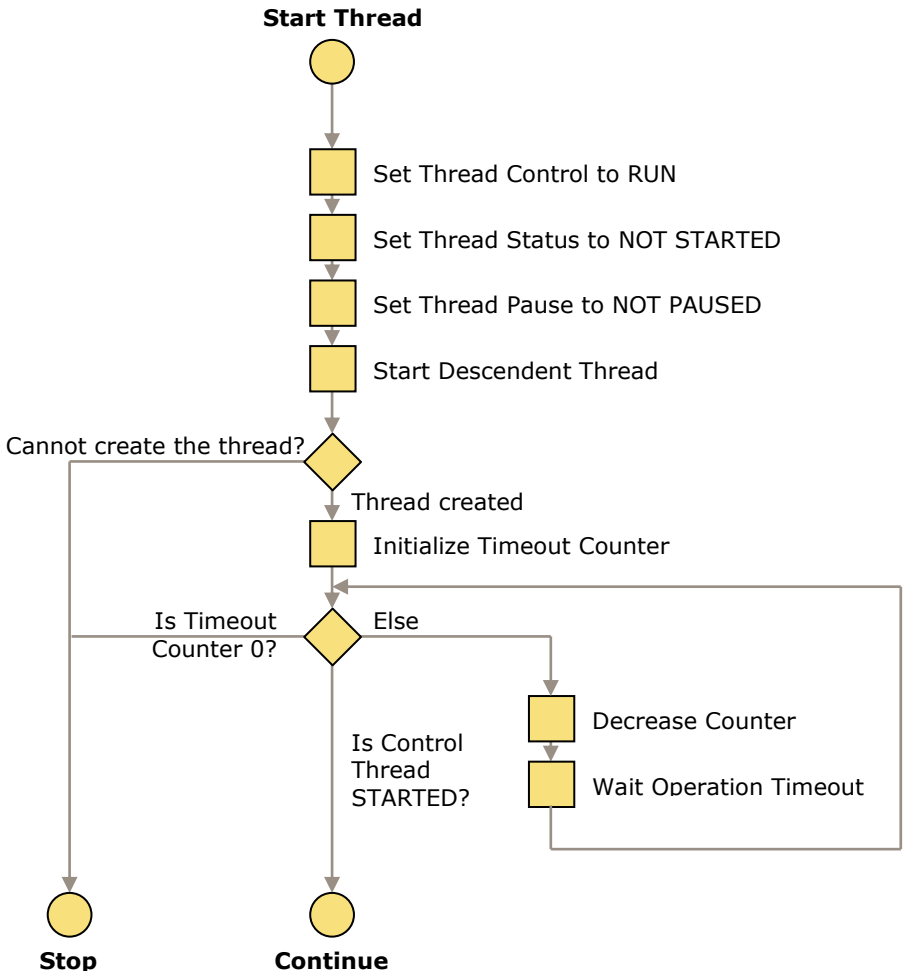
- The initialization stage, in which several set of resources are allocated and descendent threads are created and started.
- After the initialization is complete, follows the second stage or the stationary stage, in which the control thread performs cyclic operations in a thread loop.
- The loop finishes when a stop control is received (i.e. the thread control variable is set to *stopped*). After the thread loop has stopped, the descended threads are stopped and the allocated resources are released.

Some additional information about these execution stages of the main control thread might be needed. First, the creation and stopping of descendent threads is done in the same way the SCM performs the start and stop of the main control thread. I.e. any thread is started, by setting up a thread's control variable to *running* and a status variable to *stopped*, after which the thread is created. When the descendent thread starts, it sets the associated status variable to *started* and it runs as long as the control variable is set to *running*.

The parent thread waits for the newly created thread to start until the status variable is *started* or until a timeout period expires. The timeout is measured in the same way it was explained before, see figure 3.13 for more information. If a timeout occurs, the control thread decides to abort.

Each descendent thread has a loop similar to the one of the main thread, in which it performs its tasks. The execution flow stays in this loop while the control variable is *running*. If during the execution the *pause* control variable is set, the thread enters into a secondary pause loop, without exiting the primary *running* loop. The execution exits the *pause* loop either when the *pause* control is set to *resume* or when the thread control variable is set *stopped*.

Figure 3.13 Thread start control procedure



The stop procedure is done in a similar way. First, the thread control variable is set to *stop* value. After the stop control has been send, the caller thread (or parent thread) expects in a waiting loop for the descendent thread to finish. This is done by checking during the iterations whether the status variable is set *stopped* or the timeout counter reached zero. In the first case, the parent thread continues normally, while in the second a failure policy is used.

The figure 3.14 contains the algorithm diagram used for the stop procedure.

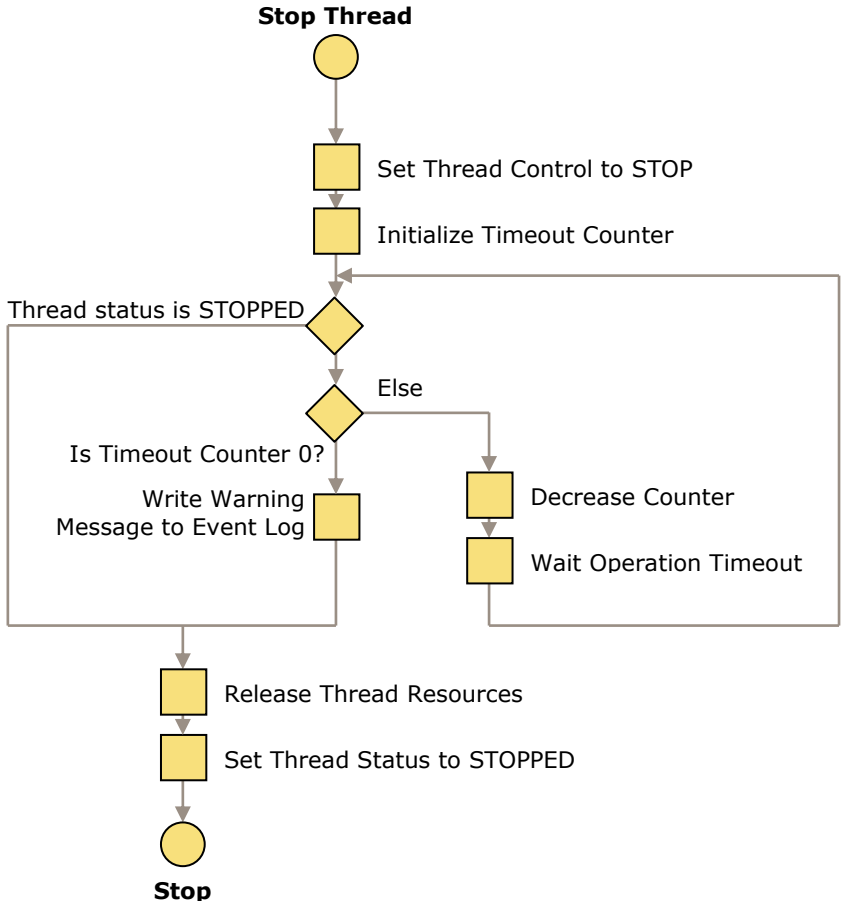
Notes

- The failure policy, in the case a created thread cannot be stopped, mostly because a blocking event occurred consists in logging an event in which the user is informed upon the situation and an automatic release of resources.

Caution

- In the situation a descendent thread hangs, and you are warned that the "*resources are released automatically and an undefined behavior may occur*" you might want to restart the application because data or application memory corruption may happen in the future. Although it is extremely unlikely for a service thread to hang, if this happens, save your data and restart the application. The control threads and especially the Service Control Manager are required to release any resources when a stop control has been issued, even if the service or part of the service stopped responding. Therefore, having threads still in execution, which did not respond to the stop control and maybe are still using resources that have already been released, eventually will lead to an undefined behavior.

Figure 3.14 Thread stop control procedure



One may observe from the previous algorithm that the descendent thread or threads status is not affected even if they fail to stop within the allowed timeout interval. Actually, when discussing the thread implementation in the management infrastructure the golden rules are:

- Only the parent thread is allowed to change the control variable for a descendent thread
- Only the descendent thread is allowed to change its corresponding status variable, the parent only inspects the value of these variables in order to determine the status of the descendent threads.

Notes

- However, the parent thread is allowed to initialize the status variable of a descendent thread prior to its startup. This exception is presented in figure 3.13 where the thread status is set to *not started* after the control has been set to *run*.

3.3.4 SOCKET CONTROL THREAD

The socket control concept refers to the operations that make possible the transmission and reception of network data using the operating system's socket interface. The socket control operations contain procedures such as socket initialization and release.

In addition, because the management service supports connection through multiple network interfaces or through a multihomed interface (a network interface having multiple IP addresses assigned), the socket control ensures that a socket reference is opened on all interfaces that have been previously selected by the user.

After the socket procedures are completed, the socket control thread is also responsible of starting the socket data threads that will handle the data transfer socket operations. There are two reasons for which the socket operations i.e. socket open/close and socket read/write have been separated in two threads:

- A failure or a block is more probable to occur during data transfer procedures, and therefore if such an event happens the control of the thread is still maintained. Shortly, if a read/write procedure blocks the socket data thread it would never affect the socket control thread. The advantages of this approach will be explained immediately.
- The second issue is robustness. By keeping socket and data operations running in two functions and having two separate stacks prevents even the smallest corruption event such as the case of buffer overflow and the socket operation can be recovered. Furthermore, a prolonged data operation cannot affect the ability of the threads in receiving controls from parent threads and finally, it isolated user management data from other internal data structures.

Notes

- In implementation, the probability that a socket operation to fail or block is prevented through several methods such as data availability checking and received data size checking. Even so, because the implementation of the service relies on operating system code, such events were not excluded and the occurrence was taken into account.

In understanding the reasons why the socket control should be implemented separately from the socket data operations, try to imagine what if a socket operations takes too long or simply blocks the thread operation. The most basic operation that would block the thread would be a read operation from an empty buffer or a write operation onto a full buffer. In the first situation, the operation would block indefinitely if management data would never arrive on the computer.

The management service has been designed to prevent recovery from such events, in the case that their occurrence cannot be prevented. This is done by using a thread in which the socket control operations are performed. If the socket data operation blocks for instance, it would only block the socket data thread.

Now imagine that the SCM sends a stop control to the management service. The stop control is received by the main service control thread in the form of a control variable change from *running* to *stop*. As it was explained in the previous section, the main control thread would exit the thread loop and start sending stop control to its descendent threads, i.e. the socket control threads. The socket control threads also have a thread loop in which they wait for a stop control to arrive.

When the socket control thread receives the stop control it exits the loop immediately and like its parent sends stop control to its only descendant, the socket data thread – see figure 3.10 to see the threads dependencies. Nevertheless, at this point, the socket data thread is blocked in a socket data operation and it cannot stop, meaning that associated status variable does not change – remember that only the thread could modify its status variable.

The socket control thread implement the stop procedure described before, meaning that it waits a finite amount for the descendant thread to modify its status. If this does not happen, it would immediately attempt to release the resources allocated for the data thread, in this case by close the socket. Usually in most socket interface implementations when a blocking socket operation is performed onto an invalid socket, the calling functions exits immediately by returning an error code.

This is what really is performed when the data thread becomes locked on a socket operation. If the parent thread, i.e. the socket control thread does not receive the status change within the timeout it closes the socket anyway causing any blocking the function to return immediately with an error. You will see that the socket data thread is created in such a way that these types of error are recognized and the thread is stopped.

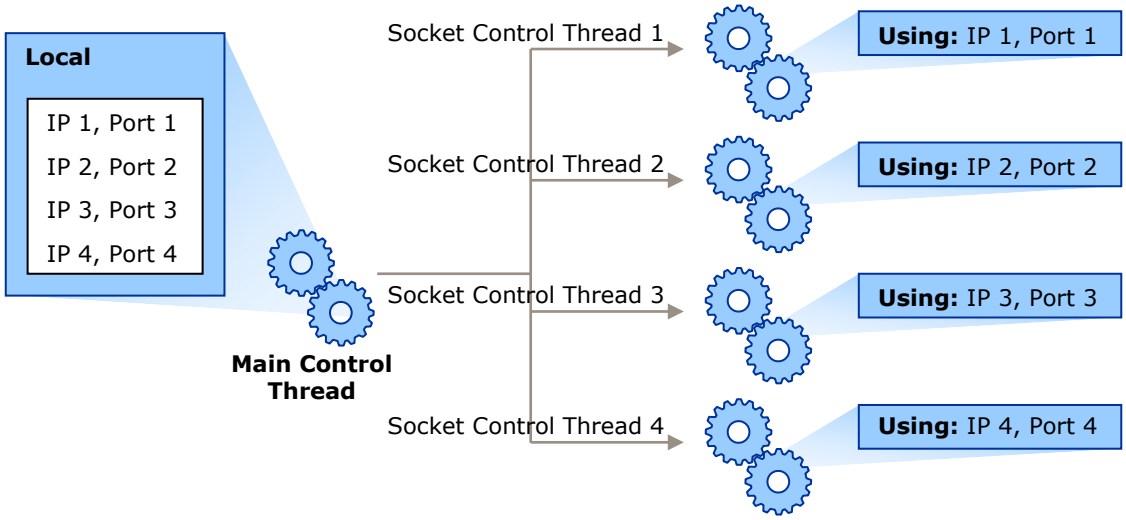
As mentioned, there are two main objectives for the socket control:

- To perform socket data operations to terminate, by closing the working socket regardless on whether the data thread stopped or not
- To handle socket opening and closing

Since the first objective was already discussed, here is some info on the last one.

When the socket control thread is started, the first operations are to gather the socket specific information from the parent and to open a new socket. The socket is opened on the local IP address received as parameter from the main control thread, and if successful, a valid socket handle is obtained.

Figure 3.15 Receiving local IP information by the socket control threads



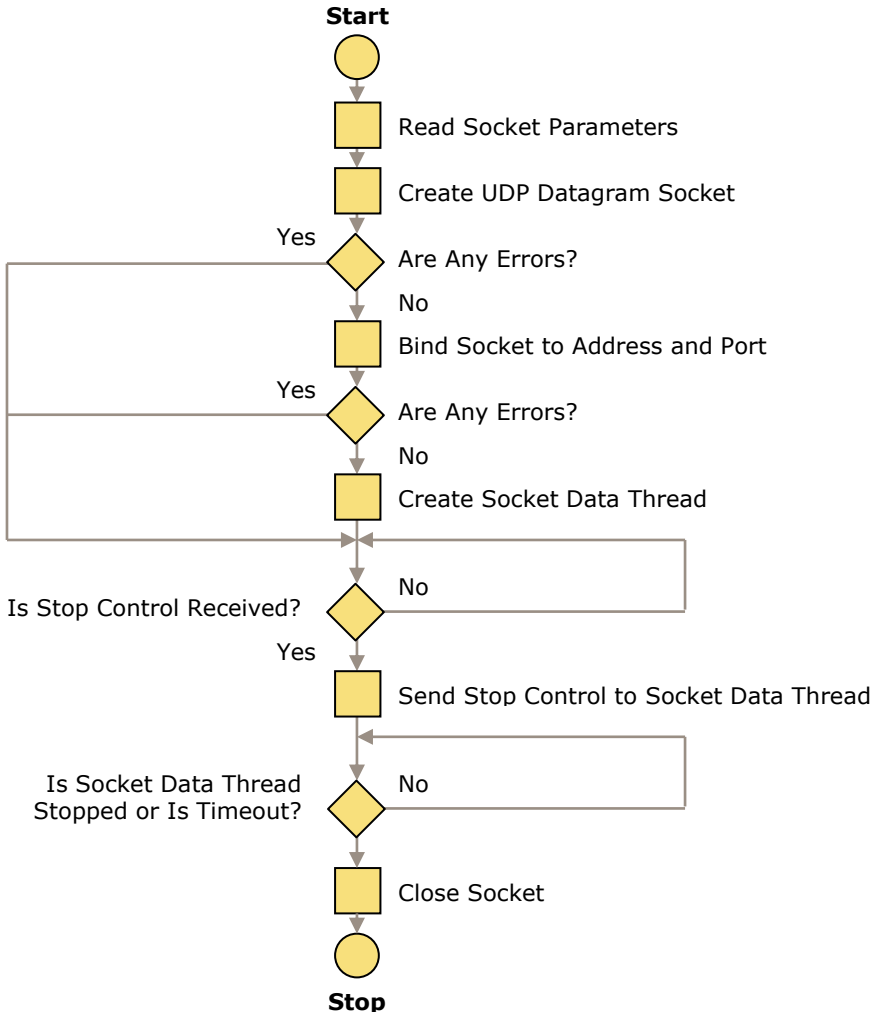
The main control thread is responsible for starting a socket control thread for each IP address selected by the user, so by the time the socket control thread function starts, the local IP information is already available as parameters. The figure 3.15 illustrates the concept.

The steps performed by the socket control thread in order to create a socket viable for SNMP communication are the following:

- Create a new socket, referenced by a socket handle. Since SNMP uses UDP as transport protocol, the type of socket is of UDP type.
- If the socket is created without errors, the socket is bind to the local IP address and port received as parameters. Usually this IP address corresponds to a specific local network interface but could you may also have two IP addresses assigned to the same interface. The default port for SNMP is 161, but it can be changed by the user either from the configuration wizard, manager properties dialog (for the management console) or from management tab (for the measurement agent).
- If the bind yields no errors, the socket control thread starts the socket data thread and then enters in the thread loop waiting for the stop control.
- When a stop was received, it sends a stop control to the data socket, it waits to finish a timeout duration after which the socket is closed, and its status is set *stopped*.

If an error occurs during either operation the socket control thread enters in suspend state, i.e. it enters directly into the thread loop without starting the socket data thread. The algorithm flow is presented in figure 3.16.

Figure 3.16 Socket control thread flow control



The algorithm schematic from figure 3.16 contains only the important steps performed. In addition, there are of course the usual thread control procedures such as setting up at the beginning and at the end the thread status variables, which are no longer presented in the figure.

A small difference regarding the socket control thread from the thread control point of view it is the lack of activity while the service is running. Otherwise said, the socket control thread is responsible only of closing its associated socket when the service stops, in the meanwhile it does not do anything more. The main thread loop, from the socket control thread, checks if a stop control is received, in which case it exits. Since there is no activity during the execution of the loop this thread is not concerned whether the service is paused or not.

This paragraph ends the discussion on the socket control in the management service. The next section presents the last topic of this service, i.e. the socket data thread that actually handles the transmission and reception of management messages, cares of security issues and performs application layer encapsulation upon the management data.

3.3.5 SOCKET DATA THREAD

The socket data thread is almost as every other service thread presented in this section. That means that it contains a thread control part, in which the thread must check for controls received from his parent (the socket control thread) and a notification part in which the thread informs its parent whenever a change in its status (i.e. running, stopped or paused) occurs.

What makes this service thread important is that at its core in the service loop, this thread performs all management related operations that are essential for the main goal of the management console and the measurement agent. The architecture presented so far was indented to increase the reliability and availability of the service provided by the software, but one must take into account that to some implementations this could be expendable. On the other hand, what the socket data thread does needs to be done.

The aim of this thread is very simple, the simplicity coming from the modular structure of the management service. So far, the main service thread was responsible of creating a new socket control thread for each local network interface selected by the user. The socket control threads just open and close the socket; creates the socket data thread and that is it. Finally, the socket data thread is responsible of reading and writing from and onto the socket and delivering data to the upper layer from the networking point of view.

I.e. the socket data thread does not deliver the data to its parent (the socket control thread) but rather to the upstream entity in the flow of management information. According to the figures 3.7 and 3.8, these are the SNMP and the queuing service.

In addition to reading and writing streams of bytes, the socket data thread has several other features; all of them are enumerated on the following list:

- Reading bytes from the network interface, whenever an UDP/IP packet having the destination address the address assigned to the network interface and a port number equal to the one configured, is received.
- Applying both types of security policy on the incoming data: the IP filtering policy allows or discards incoming packets, based on whether their source IP address is found either in allow or deny list. The SNMP security policy is based on the standardized community names (this is because the NMS uses SNMP version 1).

- If the packet is not discarded, the thread passes the data to the SNMP service in order to verify if the stream is a valid SNMP message and if so to extract the management information (i.e. the values of the managed objects) from it.
- It performs the role of an intermediary in passing the data returned by the SNMP service to the queuing service, which carries it to the user interface.
- It ensures through various mechanisms that no receiving buffer overflows and any error that might occur is handled appropriately, usually by discarding the data that caused the error.
- When sending data, it receives the values of the managed objects (including with additional information such as the agent to which the data is send, the community name to be used, and the PDU type and so on) and sends it to the SNMP service.
- Receives encapsulated data from the SNMP service for which it allocates a buffer, and afterwards it writes the data to the socket.

As one may remark, the role of the management service is in fact of basic raw UDP data transmission, and does not care of the meaning of data. Regarding the encapsulation, it relies on the SNMP service. When it comes to flow and error control, the queuing service is the one dealing with that matter. Finally, the meaning of the management data is known only by the Session Manager.

This however, does not mean that the job of the data socket thread is easy. Here are some of the factors that care the most when it comes to reliability and performance. Also for performance reasons alone, the management service also cares about security. The idea is there makes no sense on interpreting, queuing and allocating resources for a message that will obviously be discarded from one reasons or another. This implies that at least for some of the performed tasks the socket data thread of the management service looks to some parameters of the incoming and outgoing data. The following paragraphs will explain each topic in detail.

As usual, figure 3.17 shows the logic schematic of the algorithm employed. Already known pieces such as the thread control and status are obviously missing. What is important is the flow of execution inside the thread loop, where the bulk of the work is performed.

What is obviously from the beginning is that both the reading and writing operations are executed in same thread and in the same loop. The decision in doing so was based on the fact, that if the management service would not care of synchronizing duplex data, the job would have been carried out by the operating system. Therefore, instead of using two threads, one for reading and one for writing, only one is use at the expense of a little more complicated implementation.

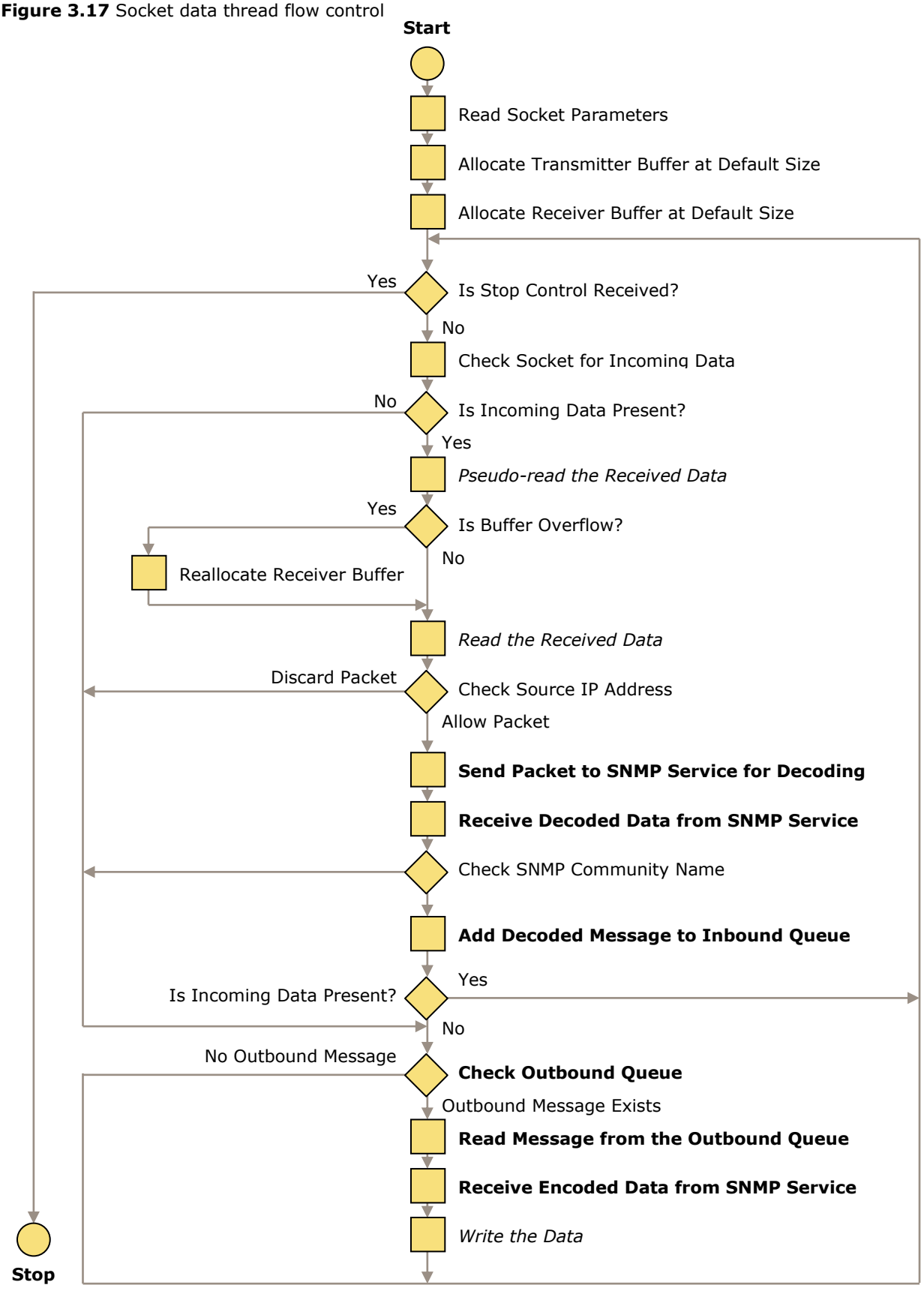
The basic idea of the thread loop is the following. At each cycle, the thread verifies if a reading or writing can be performed. The socket data thread makes first the check for reading and then for writing, but the selection to do so was purely arbitrary, so it could have been the other way around.

The check for inbound data (i.e. data coming from the network) is made using a socket specific function. If data exists, the thread prepares an incoming buffer in which the data to be put. The initial size of the buffer is of *recv buffer size* and the default value is 4096 bytes or four Kbytes.

The data is then inspected to see whether it would fit into the allocated buffer. The test is made by performing a *peek* reading into the buffer. Since the read function of the socket interface, (i.e. *recvfrom*) receives the size of the buffer as parameter ensures that no buffer overrun occurs.

Instead, the function returns an error if the buffer was too small and so the program knows that the data was only partly read and the buffer needs some adjustments. In addition to make the implementation even easier, a flag can be specified to the read function in order to return the actual number of bytes available for reading.

Figure 3.17 Socket data thread flow control



In this way, the socket data thread is able to adjust the size of the buffer on the fly in order to avoid the loss of any packets. Even so, the initial large size of the buffer (4096 bytes) prevents this process of occurring to often, such that the performance degradation seldom ever happens.

Of what the implementation is concerned there is no such big difference in inspecting the data and actually reading it, the following piece of code shows how this is done in reality.

```
iRet = recvfrom   (
                  hSocket,
                  lpRecvBuffer,
                  dwRecvBufferSize,
                  MSG_PEEK | MSG_TRUNC,
                  (struct sockaddr*)&saiRemoteAddress,
                  &iRemoteAddressLength
                  );
```

If the buffer is of appropriate dimensions, the reading is performed and the incoming data is removed from the socket buffer calling the same function but without the two-bolded flags.

```
iRet = recvfrom   (
                  hSocket,
                  lpRecvBuffer,
                  dwRecvBufferSize,
                  0,
                  (struct sockaddr*)&saiRemoteAddress,
                  &iRemoteAddressLength
                  );
```

The significance of the two flags, which are used when inspecting the data, is presented in the following table.

Table 3.11 Flags used when inspecting the socket for input data

Flag Name	Meaning
MSG_PEEK	Peek at the incoming data. The data is copied into the buffer but is not removed from the socket input queue.
MSG_TRUNC	The function returns the size of the data waiting in the socket input queue rather than the size of the data copied into the allocated buffer.

In addition, in the figure above the interaction of the management service with the upper layer services is showed with bold characters. From the entire management service, only the socket data thread interacts directly with the queuing service by calling functions specific to each of them. Later in this document, a flow chart will explain the servicing layer of each part of the management service.

For now, if the data was successfully read, it is the time to check whether it is of any good. This is done at the beginning by checking if the source IP address of the datagram that just arrived is allowed to pass. You will see in the next chapter that the IP-based security configuration can be done by either creating a list of denied IP addresses or a list of allowed IP addresses. The thread simply looks what kind of list is it about and takes the appropriate action: i.e. allowing the packet of the source address is *not* found in a deny list or discarding the packet if the source address is *not* found in an allow list.

If the message passes the IP filter check, the data is passed to the SNMP service. In the next major section, it will be presented how the SNMP service works. For now, you might keep in mind that the socket data thread sends the buffer with the received bytes to the SNMP service, and the service returns a structure that contains the data from all the fields of the SNMPv1 message, as presented in the theoretical introduction.

The SNMP service also reports any errors. Remember that the encoding of the SNMP PDUs is rather complex, following the ASN.1 rules each filed having three parts that indicate its data type, length and value. If one of these is wrong, the SNMP service tells the socket data thread what happened. If this would be the case, any future reference to that data is simply ignored and the message is lost.

Nevertheless, if the message is correctly decoded the socket data thread will receive at the end the values of all the parameters from the message. However, most of them make no sense for the management service, but some can be used for a second verification. This is, as expected the community name and the PDU type.

Remember that there are several types of SNMP messages; each of them indicate the operation that is performed. For example, the field called *PDU type* from the message tells whether the purpose of the message is to read the value of a managed object or to set its value. For a review of the SNMP notions, review the previous chapter. Now, since each community name – the security mechanism of SNMP has some associated rights that come hand in hand to the type of the message and the type of the object that is handled by the message.

In the figure 3.6, 3.7 and 3.8, one may see that almost all of the management services rely on a *configuration service* that provide the necessary user configured data. In this case, the *configuration service* provides access to the user configured community names and their set of permissions.

If the message passes the community check, it will be transmitted to the upper layers by placing it into the appropriate inbound queue. From now on, it is the responsibility of the *queuing service* of delivering the message to the application – remember that the SNMP and the management service implement the application layer of the OSI stack and not the application itself.

After the received message was sent to the queuing service, the data socket thread will handle the transmission part, again if a message is waiting for transmission.

Important

- Before the outbound queue is checked for messages to be transmitted, the software always verifies if any other messages are waiting to be read. The next section on management service conclusions will explain the reasons of this approach.

The transmission of a management messages starts, obviously with a check of the outbound queues. If the queues are empty, the transmission is canceled and the thread loop is repeated. If the queues are not empty, the queue check function provided by the *queuing service* will return the index of the queue in which the next message is found, and locks the queue for further operations.

Notes

- The previous paragraph may not say too much for the moment, since the *queuing service* was not yet explained. Later in this chapter, an analysis of the *queuing service* will explain how the priority based mechanism is implemented using several different

queues for inbound and outbound messages, and what are the queuing lock fundamentals. For now, just keep in mind that several queues exits, each having a different priority, and always the queue with the highest priority is checked first. The queue lock means that a queue operation has begun or it is in progress and the queue is currently inaccessible. However, the services that rely on the *queuing service* functions do not care on these details, which are performed automatically when a *queuing service* function is called. Nevertheless, in this case the management service requires the index of the queue in which the message was found for transmission.

Next, the socket data thread reads the message from the known queue, the message remaining in the queue.

Important

- No mechanism exits to delete an outbound message from a queue. Instead, the *queuing service* is responsible of doing this using a principle that will be explained later in this document.

The read message – or more appropriately the parameters of the read messages – is afterwards passed to the SNMP service that handles the encapsulation into a brand new SNMP PDU. The PDU is saved in the transmission buffer that is usually of the same size as the receiving buffer, 4096 bytes by default. If the size of the transmission buffer is exceeded, an error is returned by the SNMP service and the size of the buffer is reallocated. The steps involved in the transmission buffer adjustment are not presented in figure 3.17 to keep the figure as simple as possible, but you should remember that the management service always checks and increase if required the size of both transmission and reception buffers.

Notes

- The default size of the buffer cannot by changed by the user in the current version of the Network Measurement System.

3.3.6 MANAGEMENT SERVICE SUMMARY

The management service implements the low-level functions of SNMP communication. However, this is only partial true since the *management service* takes care only of the SNMP polling mechanism. The NMS console and agent provide support at the application on the SNMP trap messages but they are not implemented at the lower layers because are not used.

Regarding priorities, the reality is that the management service does not care at all of them, i.e. the management service does not implement priority-based service. You will see later in this document that this approach is taken care of in the implementation of the queuing service. The management service makes however, a distinction between incoming and outgoing messages in the way they are treated if more are present at a moment.

If simultaneously, several messages are present in the queue for transmission and several are waiting in the buffer of the socket interface for reception, the latter always have priority. This is to ensure that the limited buffer size of the socket interface that is dependent on the operating system implementation does not become full causing loss of packets. Instead, it is much more preferred to let the outbound queue fill up, since each queue is able to hold by default up 8192 management messages, totaling up 32768 messages (not SNMP PDUs) in each direction.

The last issue of the management service would be the one of duplicate messages and retransmissions. Again, this issue will be solved by the upper layers.

As a final remark, the implementation of the management service resides in the *scm.cpp* and *scm.h* files.

Table 3.12 Source files of the management service implementation (version 1.0.0.605)

File Name	Size (bytes)	Timestamp	Comments
scm.cpp	38001	05/10/2006 01:58	C++ source code file
scm.h	1522	06/04/2006 16:42	C++ header file

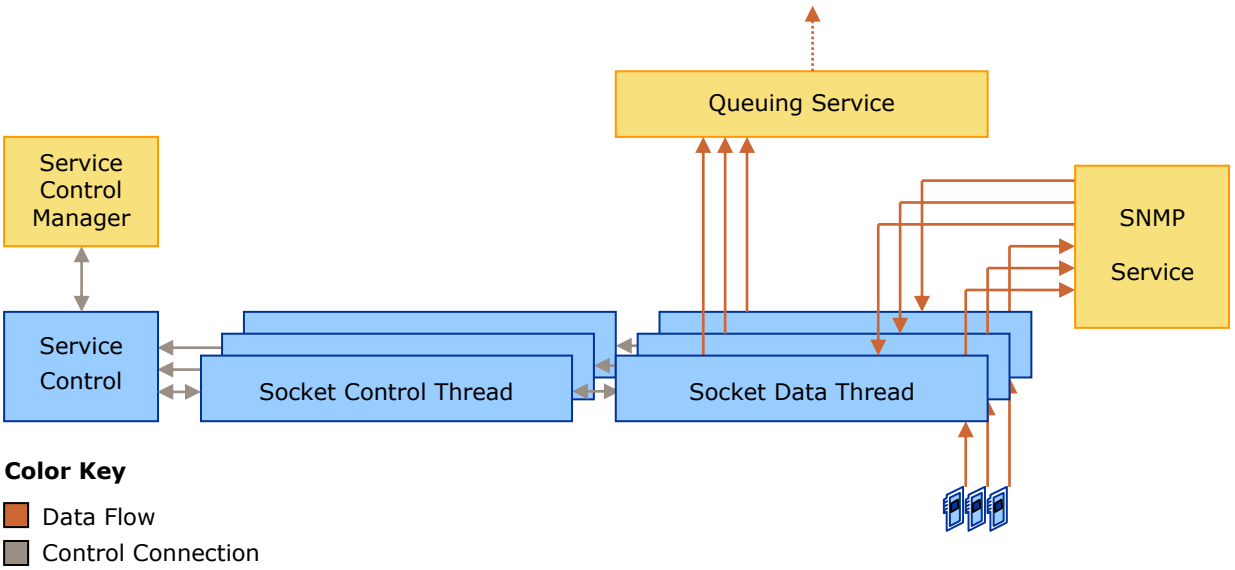
Notes

- In the table 3.12, and wherever file dates are to be expressed, they are presented in the *month*/*day*/*year* format.

The content of those files is found in appendix A.

The figures 3.18 and 3.19 show a summary of the flow of data inside the management service.

Figure 3.18 The flow of inbound data inside the management service

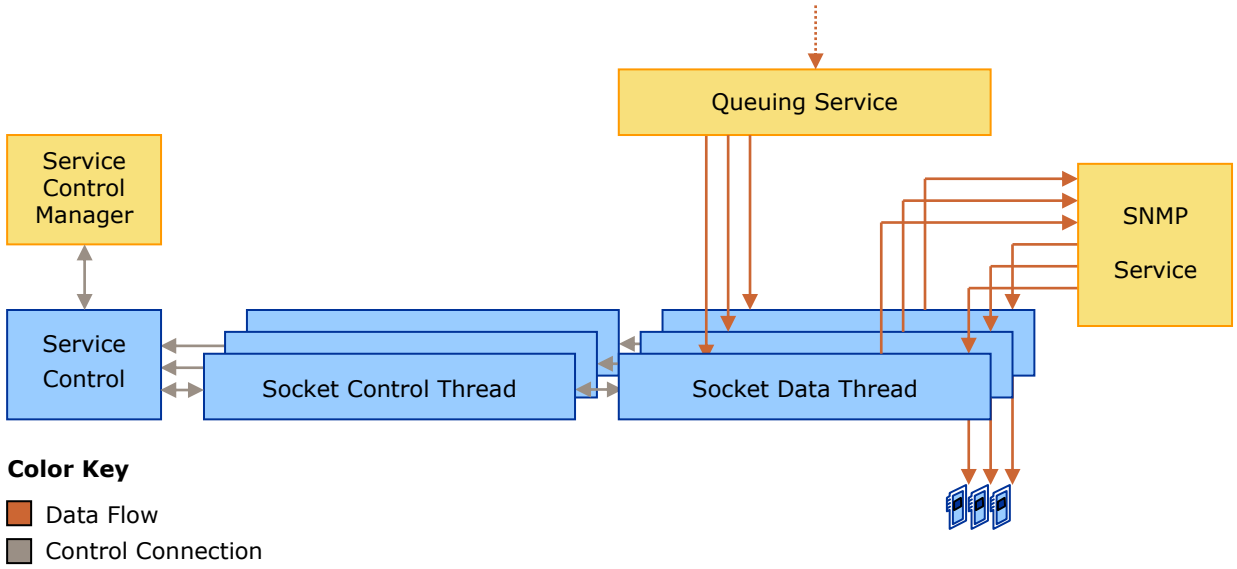


In the previous figure, you see that the flow of data and control links are exactly as explained so far. The user controls the whole service by using the Service Control Manager, which starts and stops the main service thread of the service. The main thread creates a new socket thread for each configured and selected IP address responsible of opening a socket handle and which in turn creates a secondary data thread. This thread handles the reading and writing (see next figure), sends the read data to the SNMP service for identification and decoding, and takes the result and place in into one of the inbound queues.

When sending messages the outbound queues are checked, if any messages are available it is taken from the queue (only read, not removed – the queuing service handles that), send to the SNMP service for encapsulation and then is written to the socket.

See the next figure for the data flow of outbound messages.

Figure 3.19 The flow of outbound data inside the management service



This is on short, the description of the management service. Next, we shall focus on the presentation of its closest "relative", i.e. the SNMP service.

3.4 THE SNMP SERVICE

3.4.1 SERVICE OBJECTIVES

The SNMP service is used only by the management service (the socket data thread, if considering the previous paragraphs) and provides:

- SNMP message encapsulation services
- SNMP/ASN.1 related data encoding and decoding
- Message verification

By comparison with the management service, the SNMP service does not run in its own threads, but rather is a set of functions that are executed in the context of the calling threads, in this case the socket data thread of the management service.

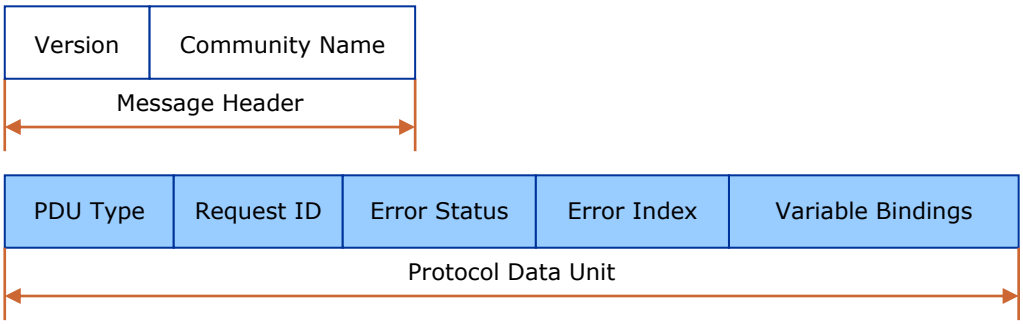
Two major functions are exported by the SNMP service as a provided service: a function that performs message encoding and one that handles message verification and decoding. In addition to these, there is also a large number of small functions that handle the encoding and decoding of almost each ASN.1 type used with SNMP.

The next paragraphs will explain shortly how the SNMP encoding and decoding is performed. Keep in mind that the functions of the SNMP service are very simple, and therefore the idea presented here is not so complicated.

3.4.2 ENCODING SNMP MESSAGES

To start, one should remember the structure of SNMP version 1 messages, already presented in first chapter of this document.

Figure 3.20 Review of the SNMPv1 message format



Even if the management service does not use SNMP traps, the SNMP service features functions for trap message encoding and decoding for use with future needs and upgrades of the NMS software. Trap handling will be presented later in this sub-chapter.

If one remembers the theoretical aspects regarding SNMP encoding from the first chapter, should know the binary representation does not resemble with the one presented in the figure above. It is not like in an Ethernet frame or IP datagram where a field in a schematic representation contains a fixed or variable number of bits that contain the binary information of that field.

In case of SNMP, each field is encoded following the ASN.1 rules. It contains three parts:

- The type of the field
- The length of the field
- The data of the field

Notes

- The encoding rules of ASN.1 data (including the encoding of the type and length) are found in the ITU-T Recommendation X.690 [13]. In the following, some basic rules about ASN.1 encoding will be explained as we look on the SNMP service implementation.

The encoding of any SNMP message starts by calculating the total length (in bytes) of the final header and PDU. To calculate the length it is important to present now the encoding of type and length fields, considering that the length of data to be encoded is already known.

The encoding function receives as parameters:

- The buffer in which to place the SNMP data as a pointer to bytes
- The original buffer size as an unsigned number
- The community name as a string
- The PDU type, request ID, error status and error index as numbers
- The managed objects as an array of structures where each structure contains the object ID (OID) and data

The encoding function returns the data into the passed buffer and the size of the buffer if it was increased.

The computation of the final message size starts by assuming a start value of zero. Then the function follows the algorithm in the figure 3.22, by adding the size of each type, length and data component of each of the fields from figure 3.20. In the figure 3.21, a refined structure of a SNMP message is provided for reference.

Figure 3.21 The refined SNMPv1 message format

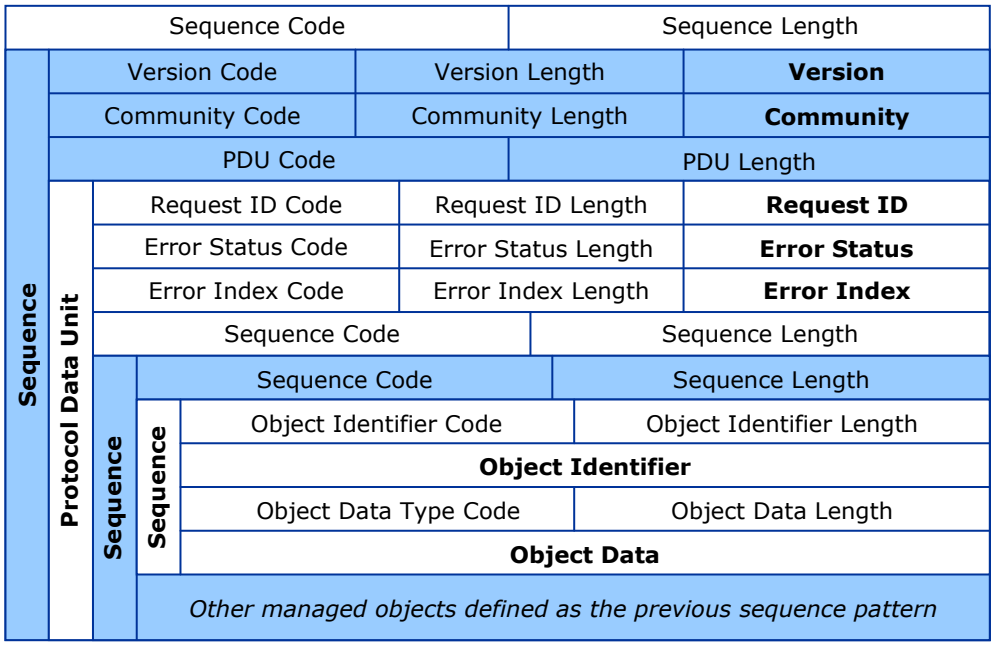
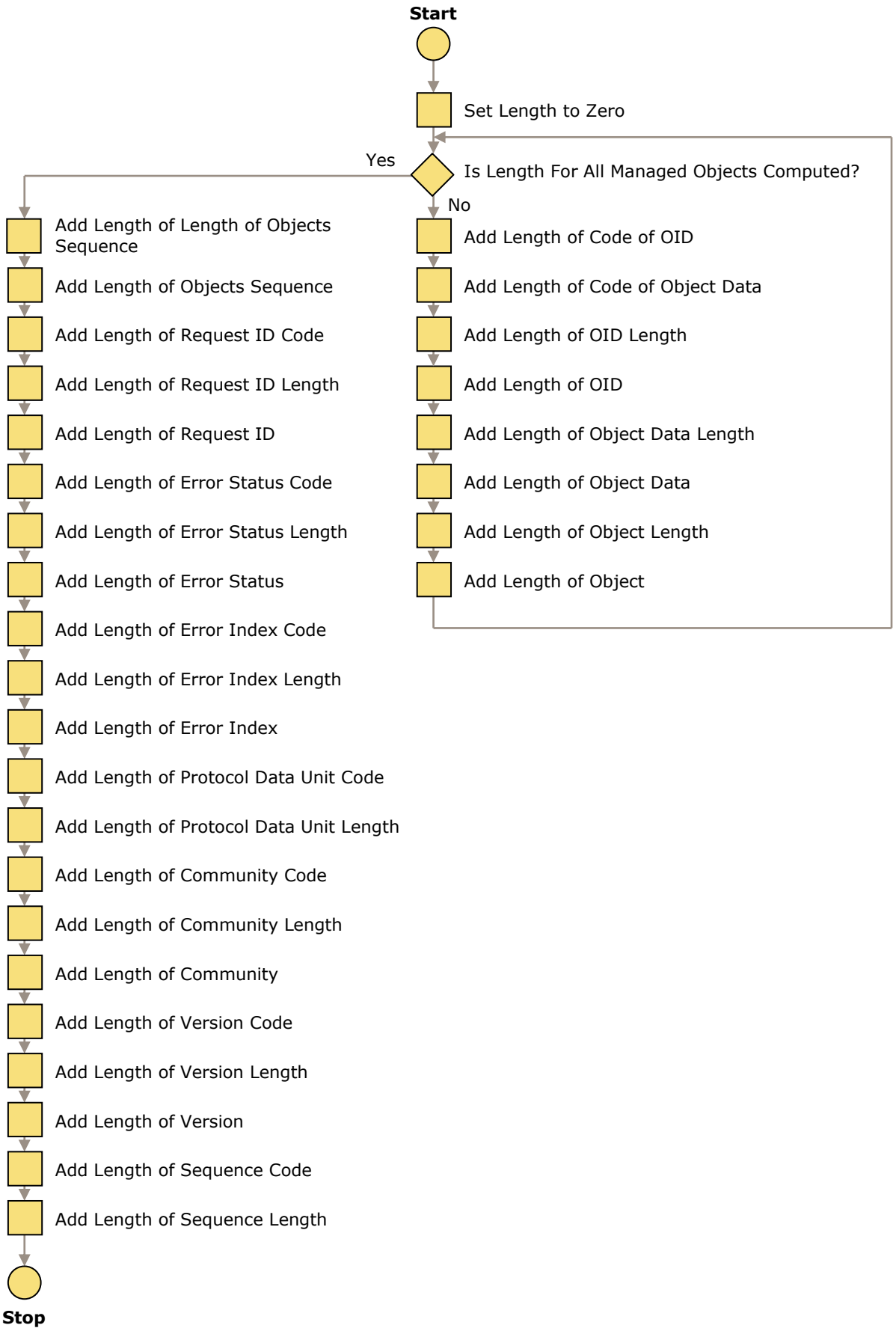


Figure 3.22 Computing the SNMP message size



One may remark that the computation of the message length starts from the end of the message. This is because, if you look at the figure 3.21, there are some *length* fields that contain the length of the data that follows them. Therefore, the length of those *length* fields is not known prior to the computation of length of the data that is in their continuation.

The first length field is the *sequence length* placed in the top row of the message format. This field contains the value of the next number of bytes that follow. In order to compute the content of its field, such that its length could be determined, one should know the number of bytes that follow, or otherwise said the length of the following data. By continuing this judgment, you may found that for the computation of the length of the *sequence length* field from the ninth row you need to know the size of all managed objects.

At the very bottom, to know the length of the message means to know the length of each managed objects such that the length of the *variable bindings sequence* is computed. Then the *sequence length* parameter from the eighth row is known so its length can be determined and so on. In the end, all lengths will be known and the length in bytes of the entire message will be the available.

Regarding the pictures from figures 3.21 and 3.22, some remarks may be required. First, there is no *version code* or *community code* or anything like similar. Those denominations are use just to understand the principle. For each field inside the SNMP there is an associated standard ASN.1 type, and its code is actually used. For example, the *version* is of integer type and therefore in the place of the *version code* the code of the integer ASN.1 type shall be used. The same is the case of the community name, which is an octet string, and the code of the *octet string* type is used instead of *community code*.

Second, the computation of the length is not done exactly like the algorithm flow chart. That should be used again only to understand the basics. In reality, intermediary results such as the lengths of each managed object and of each sequence are important because in addition the function does not only computes the length of the message but it also needs these results to assemble the packet.

The length of each field is computed by respecting the encoding ASN.1 rules as they are presented in [13]. The SNMP service features functions to compute the length of standard numeric types, of length values and of object identifiers, since the latter follow rules that are more complex. Table 3.13 summarizes the lengths SNMP service computes.

Table 3.13 Lengths computed by the SNMP service

Type	Expressed In	How Is It Done?
Numeric	Bytes	By counting all bytes starting from the least significant one, until the most significant bytes encountered are all zero
Numeric	Bits	By counting all bits starting from the least significant one, until the most significant bits encountered are all zero
Length	Bytes	By computing the length of a length field takes into account that if the number from the field is represented on more that 7 bits that the length filed contains a first byte containing the number of bytes that follow and contain the binary representation of the length number
Object Identifier	Bytes	By computing the length of each OID number taking into account the flowing exceptions:

		The first two OID numbers are encoded following the rules from the theoretical introduction If the size of one OID number is larger that 7 bits, then 1 bit is added to each group of 7 bits

The following pieces of code are more suggestive in explaining how the computation for each different data type is performed.

Computing the length in bytes for a numeric type:

```
BYTE GetNumberBytes(DWORD dwNumber)
{
	BYTE bBytes = 0;
	do
	{
		bBytes += 1;
		dwNumber = dwNumber >> 8;
	}
	while(dwNumber);
	return bBytes;
}
```

Computing the length in bits for a numeric type:

```
WORD GetNumberBits(DWORD dwNumber)
{
	WORD wBits = 0;
	do
	{
		wBits += 1;
		dwNumber = dwNumber >> 1;
	}
	while(dwNumber);
	return wBits;
}
```

Computing the length in bytes for a length field:

```
BYTE GetLengthNumberBytes(DWORD dwLength)
{
	WORD wBits = GetNumberBits(dwLength);
	BYTE bBytes = ((wBits / 8) + ((wBits % 8 == 0)?0:1));

	if(wBits > 7)
		return bBytes + 1;
	return 1;
}
```

Computing the length for an object identifier:

```
DWORD GetOidNumberBytes(LPSNMPOBJECT lpObject)
{
	DWORD dwBytes = 0;
	DWORD dwCnt = 0;
	WORD wBits;

	if(lpObject->dwObjectIdLength >= 2)
	{
		wBits = GetNumberBits(40*lpObject->lpObjectId[0] + lpObject->
				lpObjectId[1]);
		dwBytes += ((wBits / 7) + ((wBits % 7 == 0)?0:1));
```

```
            dwCnt = 2;
      }

      for(; dwCnt<lpObject->dwObjectIdLength; dwCnt++)
      {
            wBits = GetNumberBits(lpObject->lpObjectId[dwCnt]);
            dwBytes += ((wBits / 7) + ((wBits % 7 == 0)?0:1));
      }

      return dwBytes;
}
```

The last function may look strange because the type used for the objects is undefined yet. That type will be explained shortly. For now you just need to remember that it defines a single managed object along with its object identifiers, data, object identifiers length – the number of digits in the OID, data length (in bytes) and data type. In the code above, the OID length is used to determine how many digits the OID has and whether the special encoding of the two first digits is applied or not.

The length of the packet must be known before the packet encapsulation begins, since the buffer length must be checked to see whether the buffer size needs to be adjusted. If this is the case the SNMP service functions returns with an error code that tells the caller, in this case the management service, that the buffer should be increased and to try again. The previous sub-chapter explained that this is actually the case for the management service.

If the buffer size checks just fine, the message encapsulation should begin. The process is very straightforward since functions for encoding each data type are available. The difference, by comparison with the calculation of the length, is the fields are written to the buffer sequentially, starting with the first.

The prototypes of the functions used to write data to the message buffer are given here.

```
void WriteLengthToBuffer(LPBYTE lpBuffer, DWORD dwLength,
                         LPDWORD lpdwIndex);

void WriteByteToBuffer(LPBYTE lpBuffer, BYTE bNumber, LPDWORD lpdwIndex);

void WriteNumberToBuffer(LPBYTE lpBuffer, DWORD dwNumber,
                         LPDWORD lpdwIndex);

void WriteStringToBuffer(LPBYTE lpBuffer, LPCTSTR szString,
                         LPDWORD lpdwIndex);

void WriteIpAddressToBuffer(LPBYTE lpBuffer, DWORD dwIpAddress,
                            LPDWORD lpdwIndex);

void WriteObjectIdentifierToBuffer(LPBYTE lpBuffer, LPSNMPOBJECT lpObject,
                                   LPDWORD lpdwIndex);

void WriteObjectDataToBuffer(LPBYTE lpBuffer, LPSNMPOBJECT lpObject,
                             LPDWORD lpdwIndex);

void WriteTimeticksToBuffer(LPBYTE lpBuffer, DWORD dwNumber,
                            LPDWORD lpdwIndex);
```

The following table contains a small description of each function used to write data to a simple SNMP message buffer.

Table 3.14 Functions used to write data to SNMP message buffer

Function Name	Description
WriteLengthToBuffer	It writes to the message buffer a number taking into account the ASN.1 rules of writing lengths, i.e. if the length number is represented on more than 7 bits a prefix with MSB in 1 is used to indicate its length in bytes.
WriteByteToBuffer	It writes to the message buffer a number represented on 1 byte.
WriteNumberToBuffer	It writes to the message buffer a number represented on up to 4 bytes.
WriteStringToBuffer	It writes to the message buffer a null-ended character string.
WriteIpAddressToBuffer	It writes to the message buffer an IPv4 address in network byte-order.
WriteObjectIdentifierToBuffer	It writes to the message buffer the value of an object identifier.
WriteObjectDataToBuffer	It writes to the message buffer user defined data. The type and length of the data must be supplied. If the length of the data is zero, the type of the object is set to NULL.
WriteTimeticksToBuffer	It writes to the message buffer an extended time tick value represented on up to 8 bytes.

Notes

- The SNMP service does not need to compute the length of the data for composed or structured data types such as character strings or user passed data. In the case of character strings the C/C++ convention of defining the length of the string is applied, i.e. the end of the string is marked by the null character. The data passed by the user, is usually accompanied by a numeric variable that specify its size.
- Each function from table 3.14 also writes to the message buffer the type and length of the parameter that follows, implying that the calling procedure does not have to perform these operations.
- The complete list of SMIv2 data types is found in tables 2.6, 2.7 and 2.8.

The flow chart of creating the message is presented in figure 3.23.

Figure 3.23 Creating the SNMP message

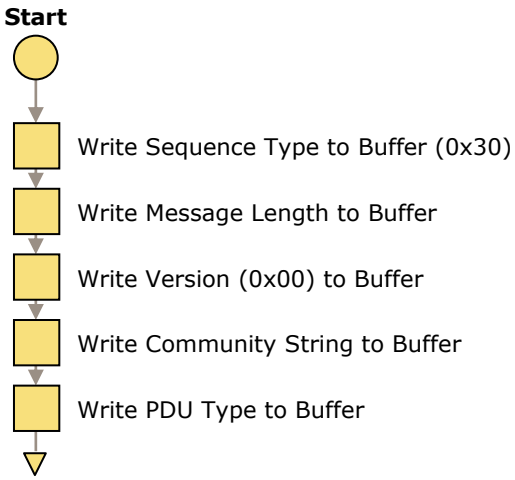
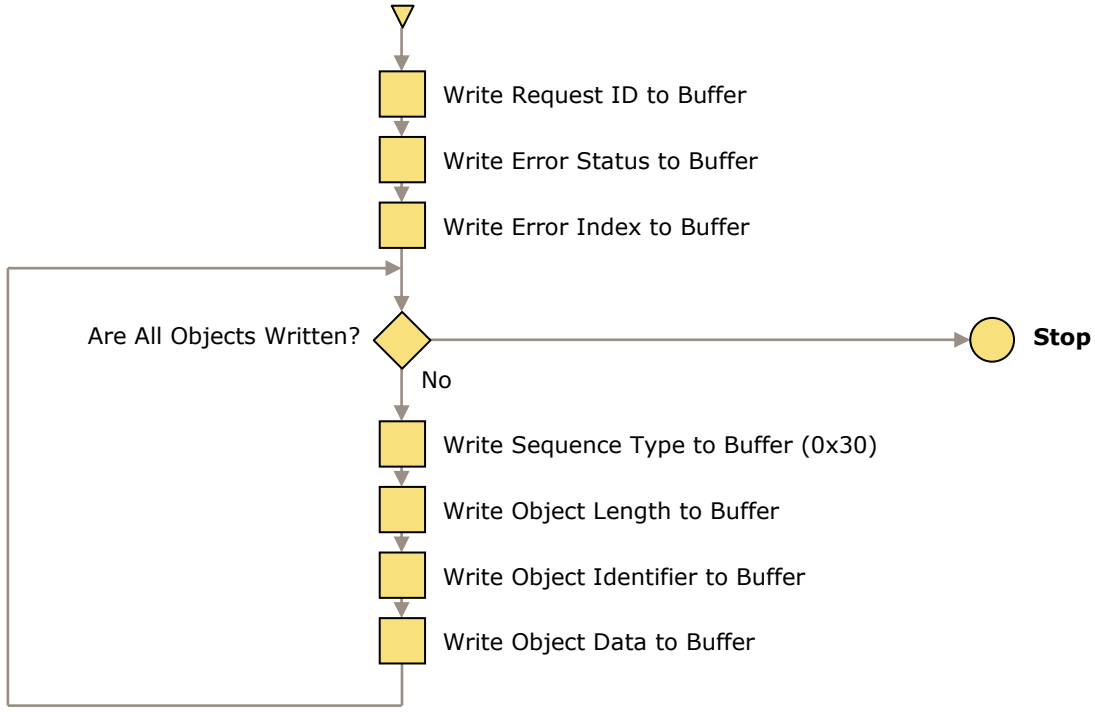


Figure 3.23 Creating the SNMP message (continued)



Each SNMP message field uses the encoding specific to its type, and hence the specific function of writing data. The only exception is the *version* field that uses the *write byte* function since it is always represented on one byte – all other numbers use the up to 4 bytes number implementation. For a review of the types of each SNMP v1 field, consult the tables 2.11 and 2.12 in the theoretical introduction.

3.4.3 ENCODING SNMP TRAPS

The structure of a SNMP version 1 trap was already presented in figure 2.15, so it will not presented once more time as in the previous case.

The current version of Network Measurement System management does not implement traps. However, the SNMP service contains the corresponding functions in order to cope well with possible future upgrades. For this reason, the SNMP service features full SNMP version 1 support even if it is not used entirely.

The encoding process of each field of the SNMP trap resembles so well with encoding of the normal polling message that many of the implementation aspects that were discussed will be skipped in this case. In this category can be included the length determination process, and the functions that write the message data to the passed buffer.

The flow charts are as well very similar, the only difference being at the points were different data is written. In this situation the *PDU Type*, *Request ID*, *Error Status* and *Error Index* fields are replaced by *Enterprise*, *Agent Address*, *Generic Trap Type*, *Specific Trap Code* and *Time Stamp*.

The following implementation aspects change:

- The parameters the trap generation function receives
- The length determination of the SNMP PDU

- The message assembly

In the figure 3.22, the additions of the length of the four replaced objects are changed by their substitute fields. This is also true in the case of the figure 3.23. Besides that, there is no such big difference between composing SNMP polling messages and SNMP traps. On what this project is concerned the SNMP trap features are fairly used, so this topic will end here. For more information about creating SNMP traps, consult either the theoretical introduction, reference paper or the source code of the SNMP service from appendix B.

3.4.4 DECODING SNMP MESSAGES

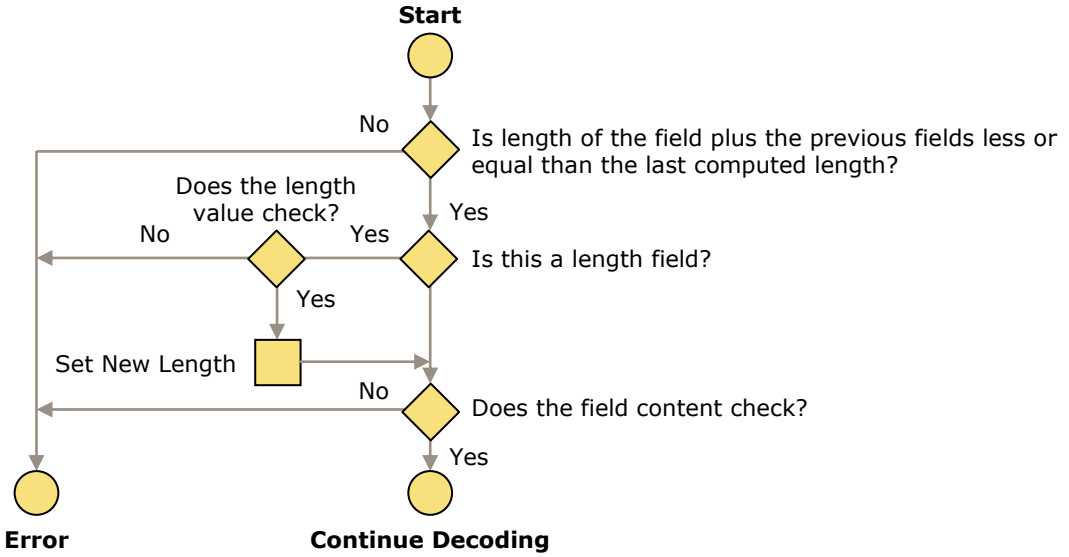
When decoding the SNMPv1 messages the key issue is to check at each field if the following are true:

- The length of the parameter field in addition to the fields already decoded does not exceed the number specified in the last length field and of the number from the last length sequence, if it exists.
- The type of parameter is the correct one, if the field is of fixed type.
- The field data fulfills the ASN.1 encoding rules.

The first criterion means that at any decoded parameter the length of the message or the length of the last sequence is not exceeded. To understand the principle, look at the figure 3.21. You can see that the SNMP message starts with a *sequence code*, a *sequence length* and so on. The length verification implies prior to check the value of the *sequence code* the software should check if the size of that field in addition to the size of fields decoded so far is less than the last length field. In this case, we have no fields decoded so far, their number is zero. There is again no previous length field, so the length of the entire message is assumed.

In the case of the second field, which is a length filed two checks are performed: too see whether the field is not outside of the received buffer, i.e. the *sequence field* length added to the *sequence code* length is less than the last available length – the size of the buffer. The second one is to see whether the value stored in *sequence field* is accurate, i.e. the binary number represented in that field is equal to the length of sequence structure that follows, which in this case is the size of the buffer minus the length of the two fields we already talked about.

Figure 3.24 Principle of checking the length when decoding SNMP messages



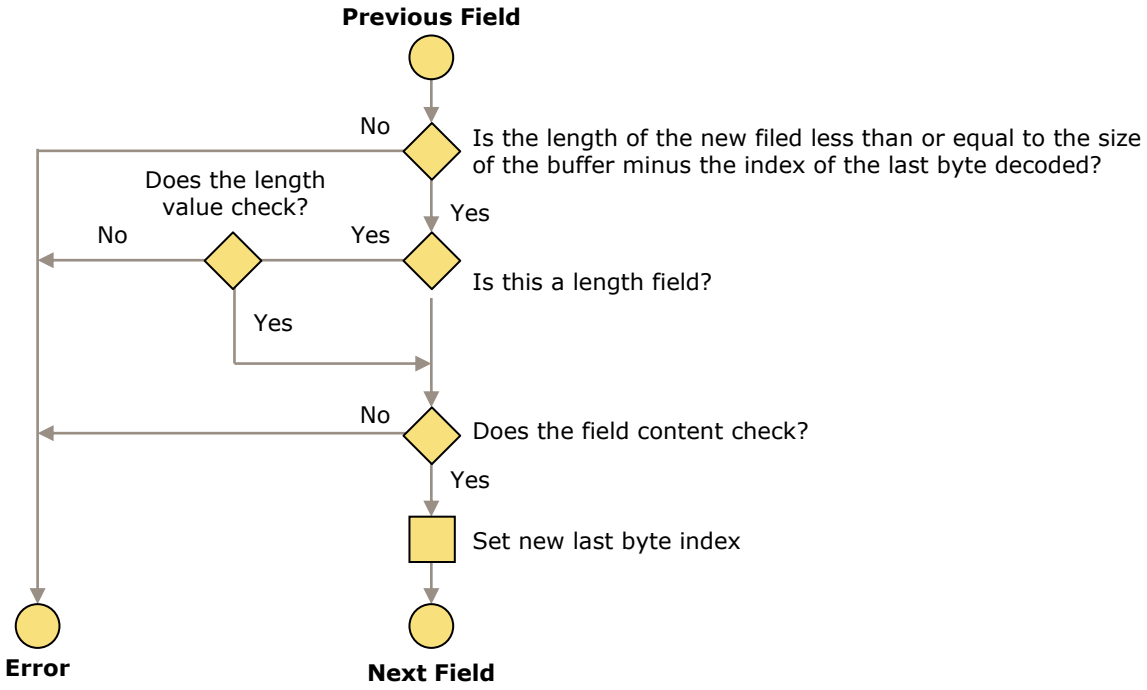
If no errors are encountered so far, in the next steps the last length value becomes the one from *sequence length*. Of course, one may wonder how this is implemented since it seems to be a lot of variable to be kept in mind when checking the length. The NMS implementation of the SNMP service does exactly as described in the previous steps except that it uses a shortcut method in doing it.

The shortcut is that the program does not computes that last lengths, storing them in some variables and then at each new filed its length is added to the previous ones and then compared to those lengths. You have already seen that the encoding of the SNMP message is done in a hierarchical manner, so especially when it comes to encoding the objects there will be too many lengths to be taken into account. Instead, the following axioms are applied to simplify the problem:

- Once the length of a field has been verified to be correct, it is no longer required to check the length of the data that follows against the value of that field.
- Nevertheless, at each new field it is necessary to check its length versus the end of buffer, to see whether the length (either specified in a length field or determined for a standard field) is not to large.

The problem is therefore reduced from checking many lengths with the current SNMP field to checking only against the end of the buffer. This is done by keeping the index of the last byte from the buffer successfully decoded. If a new field follows, it is checked only if the length of the new filed is less or equal to the size of the buffer minus the index of the last byte successfully decoded. The figure 3.25 shows the algorithm.

Figure 3.25 Implementing length checking for new decoded fields



In implementation, a series of functions perform the length and contents check for any field from the message. The prototypes of these functions are presented in the next paragraph.

```
int ReadLengthFromBuffer(LPBYTE lpBuffer, LPDWORD lpdwLength,
                         LPDWORD lpdwIndex, DWORD dwMessageSize);

int ReadNumberFromBuffer(LPBYTE lpBuffer, LPDWORD lpdwNumber,
                         LPDWORD lpdwIndex, DWORD dwMessageSize);

int ReadIpAddressFromBuffer(LPBYTE lpBuffer, LPDWORD lpdwNumber,
                            LPDWORD lpdwIndex, DWORD dwMessageSize);

int ReadStringFromBuffer(LPBYTE lpBuffer, LPCTSTR *lpszString,
                         LPDWORD lpdwIndex, DWORD dwMessageSize);

int ReadOidFromBuffer(LPBYTE lpBuffer, LPSNMPOBJECT lpObject,
                      LPDWORD lpdwIndex, DWORD dwMessageSize,
                      DWORD dwObjectSize);

int ReadTimeticksFromBuffer(LPBYTE lpBuffer, LPDWORD lpdwNumber,
                            LPDWORD lpdwIndex, DWORD dwMessageSize);

int ReadObjectDataFromBuffer(LPBYTE lpBuffer, LPSNMPOBJECT lpObject,
                             LPDWORD lpdwIndex, DWORD dwMessageSize);
```

The next table contains the description of each function used in decoding SNMP message fields.

Table 3.15 Functions used in decoding of SNMP messages

Function Name	Description
ReadLengthFromBuffer	It is used to read the length field, which is the prefix of any data type represented in ASN.1 format. This function is usually called by the following ones since they perform both length and type reading and verification.
ReadNumberFromBuffer	It is used to read a numeric value from the SNMP buffer at the specified index.
ReadIpAddressFromBuffer	It is used to read an IPv4 address value from the buffer at the specified index.
ReadStringFromBuffer	It is used to read an octet string from the buffer at the specified index.
ReadOidFromBuffer	It is used to read an object identifier from the buffer at the specified index.
ReadTimeticksFromBuffer	It is used to read an extended 8-byte time tick from the buffer at the specified index.
ReadObjectDataFromBuffer	It is used to read any type of data from the passed buffer at the specified index.

Notes

- The parameters required to all functions are the message buffer, in which the received buffer is stored, a pointer to a memory area where the results shall be stored, the current index in the buffer, and the message size.
- All functions perform automatic detection and check of length and type fields so no additional processing regarding those fields is required. If the check fails, because the length is incorrect or the value of the type code is invalid for that specific function

(e.g. an octet string variable with type 4 is found when reading a number), an error code is returned and the reading is aborted. The result returned in these cases is undefined.

- All functions – except the ones for reading managed object values – receive a pointer to an allocated variable in which the results are returned. The functions that perform reading of objects, i.e. *ReadOidFromBuffer* and *ReadObjectDataFromBuffer* receive a pointer to a managed object structure, and the necessary memory is allocated by the function. In this case will be the job of the receiver of the message to free the allocated memory. See the important notice that follows.

Figure 3.26 Reading the SNMP message

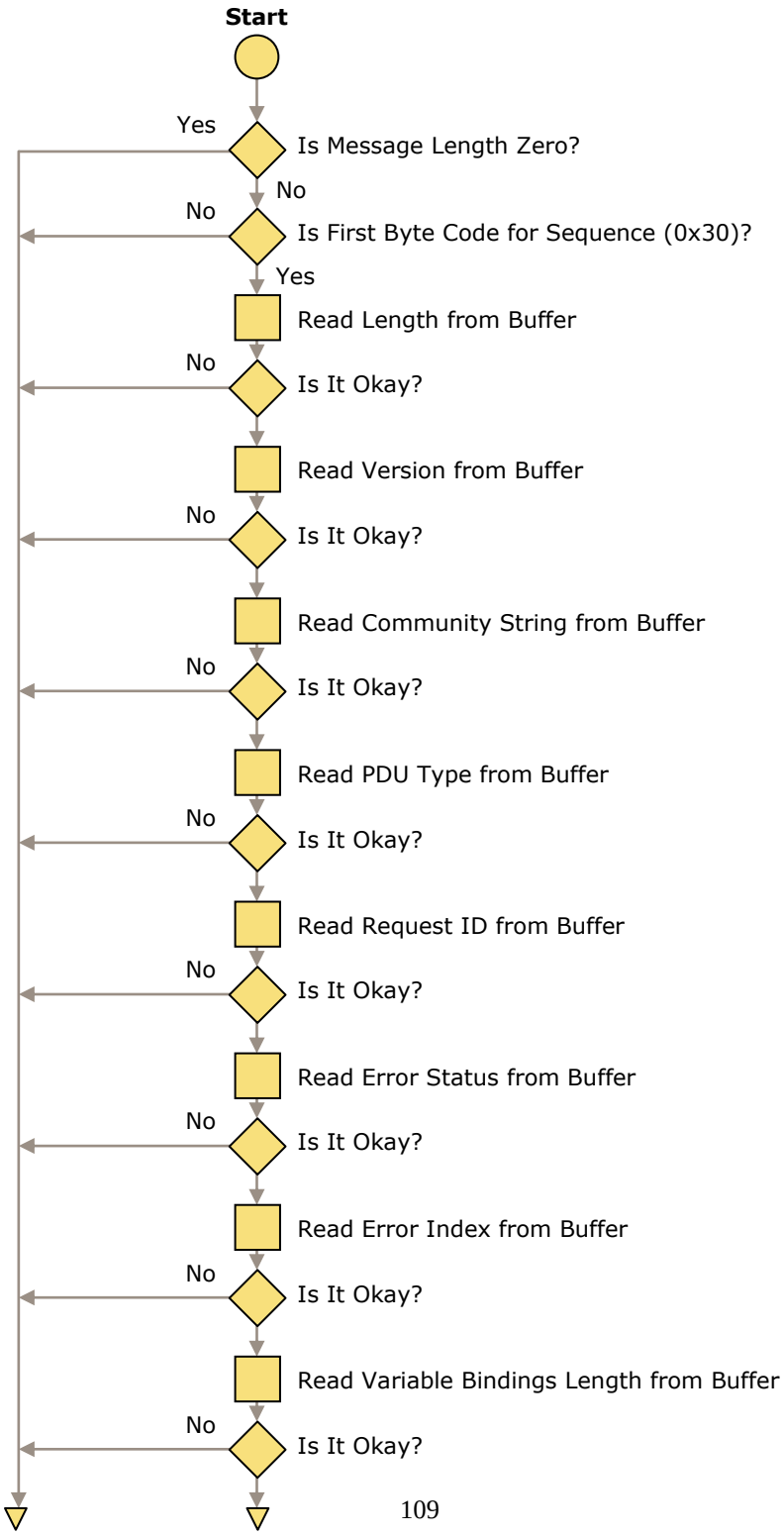
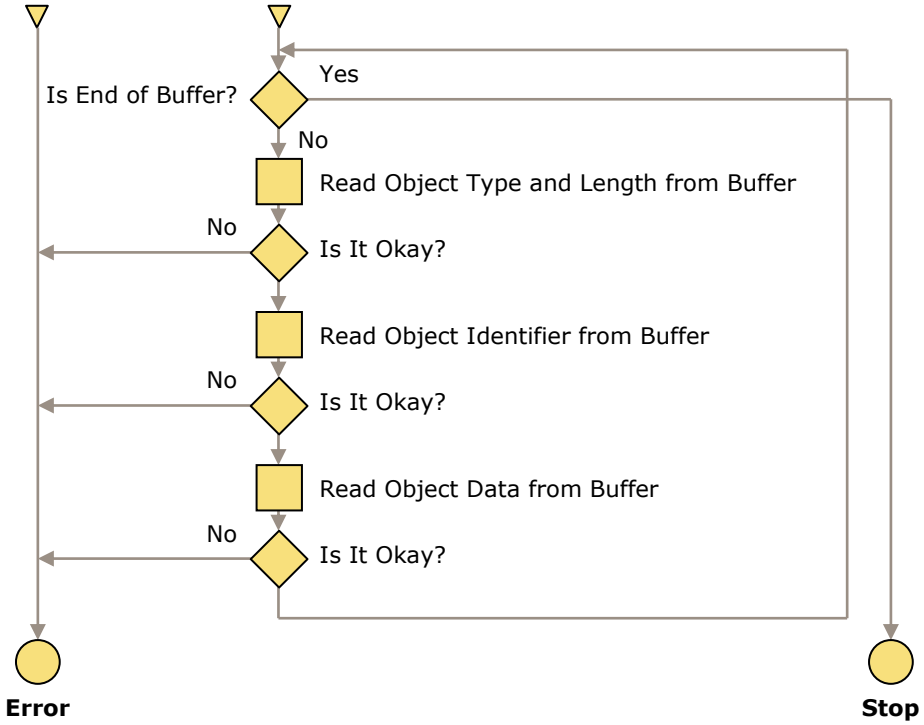


Figure 3.26 Reading the SNMP message (continued)



Important

- This notice is about memory management across the services of the management infrastructure of the Network Measurement System. There are situations, like the functions from the last note, in which a function receives a pointer to a variable and the function allocates the necessary memory space. This is true for functions that create managed objects structures. These structures will be covered shortly. In this case, it is the duty of the functions from the other point of the data flow to release that memory space. Therefore, for functions that read SNMP messages from the network and create the SNMP message structure (a structure that contains in a set of variables all the fields of the message) the destination, which is usually either the *Session Manager* or the *Message Dispatcher,* releases the necessary memory. For messages that were created by the *Message Dispatcher* or the *Session Manager*, will be written to socket by the SNMP PDU creation function from the previous sections, the memory release is usually performed by the management service. The previous subchapter about the management service did not mention this detail, because would have been hard to understand at that time. For now, you should take into account that the management service is responsible to free any memory that was allocated by other service related to SNMP messages data structures. When the SNMP message structures will be presented later in this section, the fields that need to be released when the message is no longer used shall be pointed out.

Figure 3.26 contains the algorithm used in reading of SNMP polling messages. Because all the functions already exist, each step is performed by calling the functions for the type of each message parameter. Objects data are read as unknown types and stored into memory areas of the appropriate sizes. Pointers to these areas are regularly returned as results.

After each filed read operation the reading function verifies whether the last operation completed successfully, i.e. the portion of the SNMP PDU already read is free of errors. Otherwise, the reading halts and an error is returned, informing the calling thread, usually the socket data

thread, that either the SNMP message is corrupted or it is not a SNMP message – the message will be discarded.

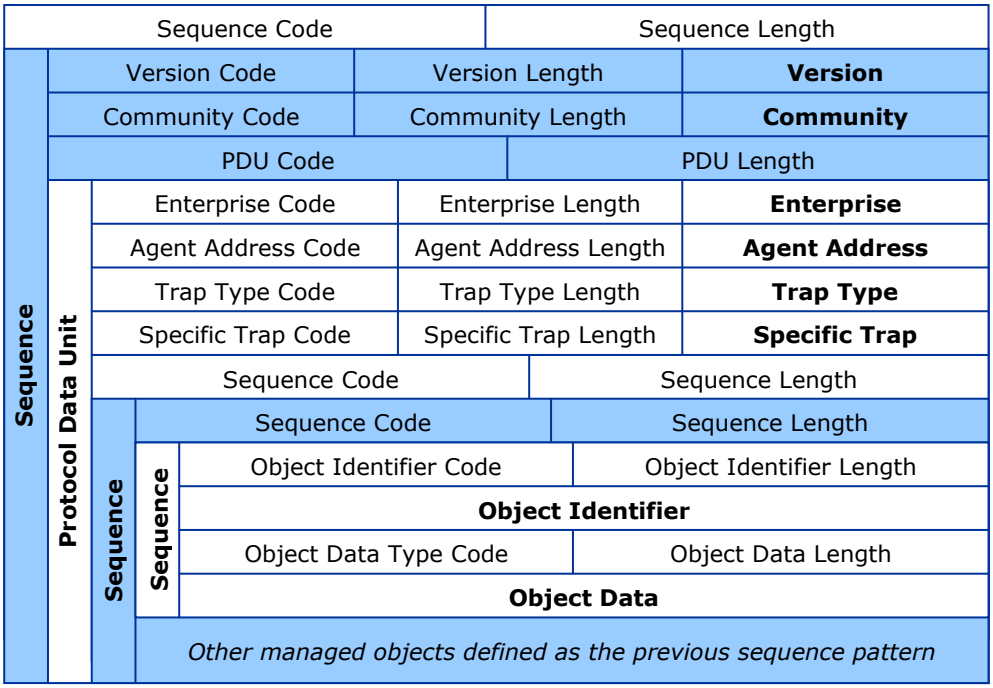
3.4.5 DECODING SNMP TRAPS

The discussion about decoding the SNMP traps will be very short for the following reasons: first, the difference between a polling message and a trap for SNMPv1 means that four of the fields are replaced with five. In implementation, it implies the substitution of four of reading operations with other five – remember that the functions that implement the reading of each type of file already exists so no additional code is necessary.

Second, the SNMPv1 trap messages are not important from the point of view of our project because they are not used. The reasons, why the SNMP service however implements them were already presented.

The figure 3.21 contained the detailed structure of a SNMP polling message. For your reference alone, the figure 3.27 contains the detailed structure of a SNMP trap message. For additional information, refer to the introduction or to the list of references from the end of this paper.

Figure 3.27 The detailed structure of a SNMP trap message



3.4.6 MANAGED OBJECTS DATA AND USAGE REQUIREMENTS

Even the encoding and decoding functions of the SNMP service require each message filed to be separately specified the service library features several data types to help handle the SNMP information more easily. These is also one type which is required as parameter for the SNMP service encoding/decoding functions, the array of managed objects to be filled in the *variable bindings*.

Table 3.16 contains the types defined by the SNMP service library. The first one contains the structure of a SNMP managed object, while the latter has the structure of a SNMP managed objects array.

Table 3.16 SNMP data structures available in the SNMP service library

Type Name	Description
SNMPOBJECT	It is a record-like type, which contains data regarding a single managed object.
LPSNMPOBJECT	It is a pointer to the previous type.
SNMPOBJECTLIST	It is a record-like type, which contains data regarding a set of managed objects.
LPSNMPOBJECTLIST	It is a pointer to the previous type.

The C/C++ definition of the first two types is:

```
typedef struct {
      LPWORD lpObjectId;
      DWORD dwObjectIdLength;
      LPBYTE lpObjectData;
      SNMP_DATA_TYPE sdtDataType;
      DWORD dwObjectDataLength;
} SNMPOBJECT, *LPSNMPOBJECT;
```

The C/C++ definition of the last two types is:

```
typedef struct {
      DWORD dwObjectCount;
      LPSNMPOBJECT lpObjects;
} SNMPOBJECTLIST, *LPSNMPOBJECTLIST;
```

Some instructions about the two data types and their usage in conjunction with the SNMP service are required. The SNMP object type is used to define a single managed object, and therefore it contains the records needed for such a definition:

- A pointer to a 16-bit numeric array to store the object identifier (OID)
- A numeric variable to store the length of the OID
- A pointer to a byte array to store the object data – the data for each object is saved to such a memory area, regardless of its original format (e.g. for a numeric variable represented on 3 bytes this pointer will reference a 3-byte array that will contain in network order the value of the number)
- A variable that indicated the type of object data – this is useful to interpret the bytes stored in the object data byte array
- The object data length in number of bytes (e.g. in the previous example the value of this number is 3)

Notes

- The type used to define SNMP data types, i.e. *SNMP_DATA_TYPE* encountered in the definition of SNMP object type is presented in the source code of the SNMP service from appendix B.

The SNMP object list type features the following properties:

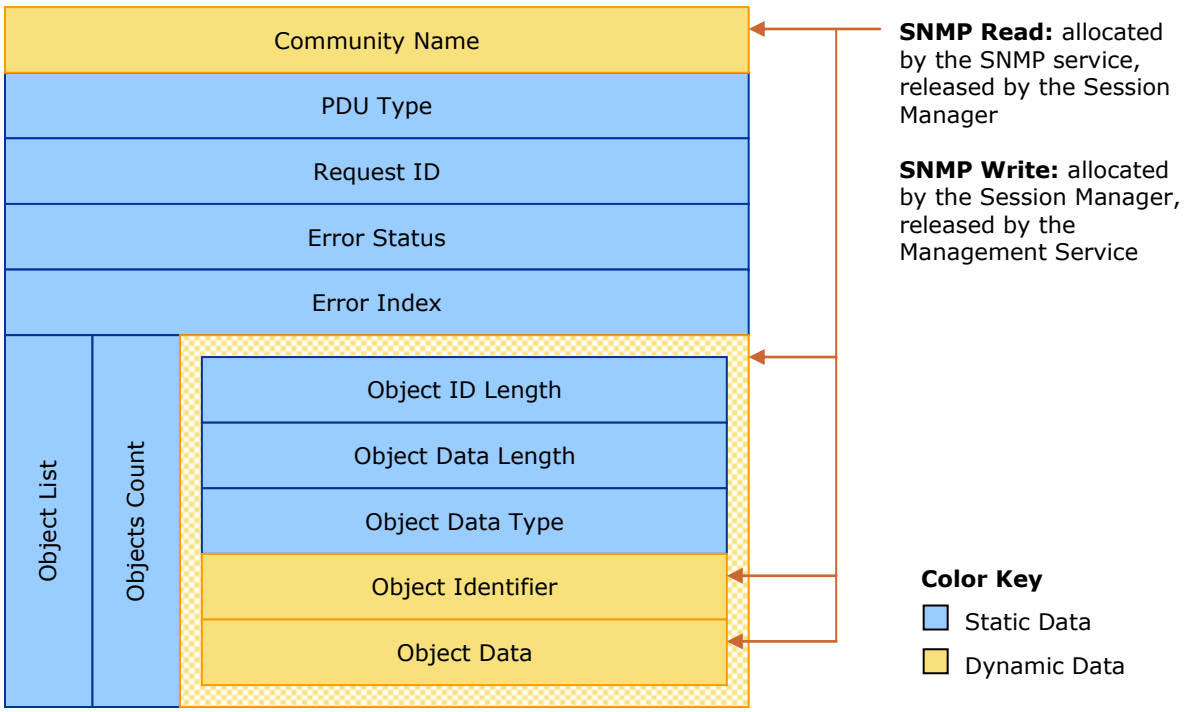
- A 4-byte value to indicate the number of managed objects, hence up to 4,294,967,296 objects can be stored into a single SNMP message. In reality, this number is limited by the computing system available resources.
- A variable of type pointer to a SNMP object type. This will reference a memory area that will contain consecutive SNMP object records. The size of the allocated memory space equals the value of the object count field multiplied by the size of a single object (i.e. 26 bytes: 4 bytes for both pointers to OID and object data, 4 bytes for both their lengths and 2 bytes for the data type).

When using these data types with the SNMP service the following considerations need to be taken into account.

- For data arrays which have unknown size prior to the SNMP read operation, the required memory will be allocated by the SNMP service function. Consequently, it is the duty of the application to release that memory. For the management console, the *Session Manager* is responsible for that. In the case of the measurement agent, the release of resources is done by the message dispatcher if it is the case.
- In the case of SNMP encapsulation operation, which receives pointers to allocated memory, the SNMP service does not release the memory. The management service handles that release if the message is not to be retransmitted. Since the queuing service handles the retransmissions, a status variable tells whether the data from the SNMP message is no longer used and the memory could be free.

The figure 3.28 indicates which fields from a SNMP message are dynamically allocated, and require a release when they are no longer in use.

Figure 3.28 SNMP message resources



The algorithms for allocating and releasing memory resources for SNMP messages are presented in figures 3.29 and 3.30.

Important

- If you plan to implement the NMS management infrastructure in your own application, keep in mind that the memory allocated to an incoming message is always done for you by the SNMP message, and therefore is your responsibility of releasing the memory only *after* the message is *read* and *deleted* from the message queue.
- In the case of outbound messages (messages you send using the management service) you must always allocate the memory required for SNMP data, following the algorithm below, and the management service will automatically free that memory when the message is deleted from the message queue.

Caution

- Failure to follow the recommendation from the previous important notice may other cause an application failure (due to access in invalid memory) if the required memory space was not properly allocated or going out of system resources if the allocated memory is systematically not released.

Figure 3.29 Algorithm of memory allocation for SNMP messages

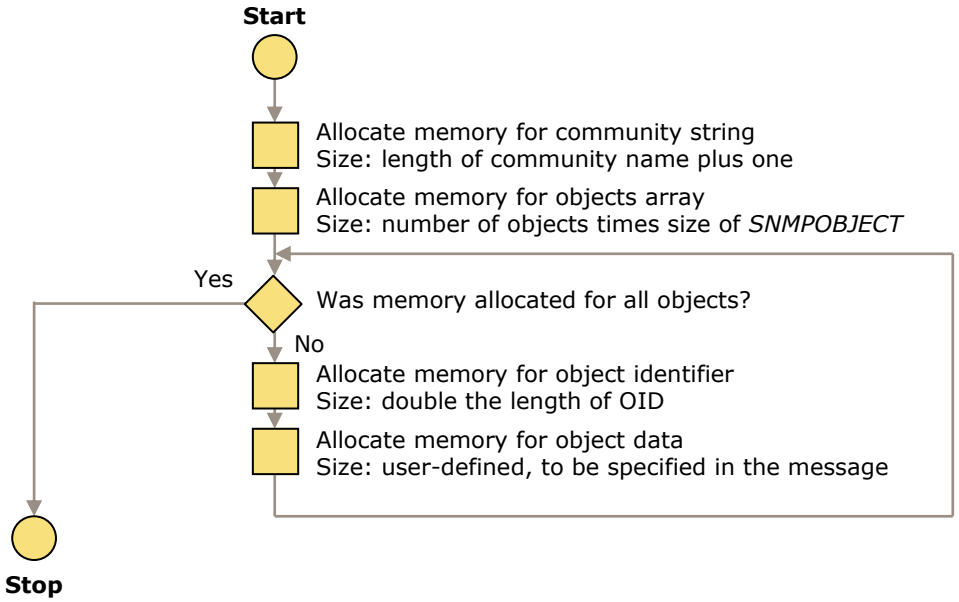
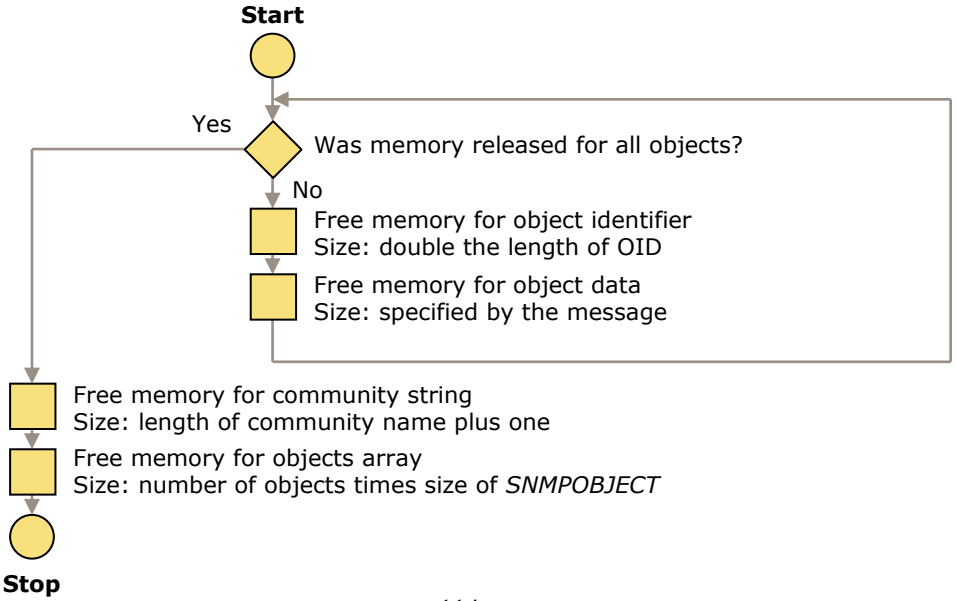


Figure 3.30 Algorithm of memory release for SNMP messages



3.4.7 SNMP SUMMARY

This section makes also the greatest opportunity in discussing why SNMP version 1 has been selected as management protocol. When decision to implement SNMP version 1 was made the next aspects were considered:

- SNMP is an IETF standard, so the platform, code and protocols are opened to other implementations extension or future upgrades. If one tries to use some of existing management services in their own application, the management notions can be learned not only from this documentation but also from other and more detailed sources.
- Version 1 was chosen since it still widely implemented in many applications: you may download free management software from the Web with SNMPv1 support.
- By comparison, version 2 of SNMP is represented by a couple of different standards and it is not as popular.
- Version 1 also has the advantage of easy configuration process; no advanced knowledge is required.
- The major disadvantage of SNMPv1 is the lack of strong authentication and security. Nevertheless, several security measures like the community-name defaults that come with SNMPv1 along with IP-based filtering implemented at the core of NMS management are though to be robust enough for the purposes of this software. Especially, since the machines on which the agents are installed are usually under control of the person that handles the measurements, the simple IP filtering could mitigate almost all threat factors. For the remaining ones, such as denial-of-service attacks or IP-spoofing the possible damage level is very small. Even if such events should occur, one should remember, that the from the management console software is always under control of the user tasks. Unwanted incoming SNMP data will be ignored by the application. Advanced operations parameters such as inbound queue size and inbound queue recycling help preventing such unwanted messages to accumulate and are automatically discarded.
- The final reason of selecting the SNMPv1 for the management of the Network Measurement System was the limited available time for developing of the management infrastructure. The delivery of management messages is at the boundary of the primary scope of NMS – the main goal is to perform quickly reliable measurements and to have the results available as soon as possible. Therefore, the selection of SNMP was made just to provide a standardized framework.

Future work and upgrades to the Network Measurement System may include also an upgrade to the SNMP implementation possible of using SNMPv3.

The last topic of this sub-chapter is the SNMP service environmental variables or environmental variable since only one exists. Table 3.17 contains this variable, description and its default value.

Table 3.17 Environmental variables of the SNMP service

Variable Name	Default Value	Unit	Description
Maximum Object Identifier Length	512	numbers	It represents the maximum numbers that can be used in the dotted ASN.1 format of object identifiers (OID).

The value of SNMP service environmental variables can be changed by the user via the SNMP service properties dialog box in the Services view of the management console. On the agent, this value is fixed and cannot be modified.

3.5 THE QUEUING SERVICE

3.5.1 SERVICE OBJECTIVES

Imagine that you have a management console installed on a computer with two network interfaces, each interface connected to a different IP network. In each network, you have several measurement agents, and you are performing a test with two agents, one in each network. This simple scenario is enough to point out that there is the possibility for the management console to receive two management messages at the same time instance or at least very close to one another.

When the architecture of the management console was presented, it was said that the final destination of incoming management messages is the Session Manager, the only software entity that is aware of the meaning of management data from each message. However, you will see in the next chapter that the Session Manager not only processes sequentially the management messages for a single measurement test, but rather for all tests. This means if you have ten test sessions running, end you receive simultaneously management messages for each, the Session Manager processes the first received message, after it finishes the second and so forth.

In addition, if SNMP messages cannot arrive simultaneously on one network interface, for sure they can arrive simultaneously on two. The solution seems very simple: to send the first received message to the Session Manager for processing (if they arrive at the same time, one can be selected arbitrarily) and save the second until the first one is processed. If more messages arrive at once, send one and save all the other. After each message is processed, send another one the Session Manager until all messages are processed.

The things happen in the same way if more messages arrive, when one message is already in processing and others are already waiting.

The issue presented here can also be thought of in reverse. If two or more test sessions are executed, and many management messages need to be send at very close intervals (closer than the duration of sending a single message) a mechanism should ensure that the first message is sent to the management service for encapsulation and transmission while others are waiting. If more messages are coming in the meanwhile they will be added to the messages waiting in the *message queue*.

The little story presented above means only one thing. Between the processing and transmission of messages, a queue-like memory buffer should exist in which the extra management messages could wait until the server – either the Session Manager or the management service for the management console and either the message dispatcher or the management service for the measurement agent – is available to receive another message.

The role of the management service is to implement a set of queues in which the messages could wait their processing turn. In addition to this primary goal, some additional features of the queuing service bring management transmission reliability and performance:

- Implementing message retransmission for acknowledgeable messages

- Implementing a discarding mechanism for duplicate messages
- Implementing queue recycling to eliminate messages that stayed in the queue too long
- Establishing priorities for different classes of messages: the messages with the highest priority always will be processed first
- Implementing a queue lock mechanism which prevents for two different threads accessing simultaneously the queuing system: the thread that began the operation with the queuing service always gets to finish its task, and only afterwards other threads may be granted access

The following sections present how each of the objectives presented before is implemented by the software.

3.5.2 IMPLEMENTING THE MESSAGE QUEUE

The queuing service features different queues for inbound and outbound messages. Actually, there are four queues in each direction in order to implement priorities, but for now, the operation of a single queue is enough to understand the principle.

Each queue is created as a dynamically allocated double-linked list. Each node in the list contains information about the management message (i.e. the values of each parameter from the SNMP PDU including the managed object list), the source and destination IP addresses, the remote host port. Other parameters include flags on whether the transmission of the message requires acknowledgement, whether a message has been received a timestamp indicating how many times a message should cycle through the queue before is being retransmitted and a retransmission counter.

The C/C++ definition of a queue node is given next:

```
typedef struct __SNMPMESSAGE{
      __SNMPMESSAGE *lpNextMessage;
      __SNMPMESSAGE *lpPrevMessage;

      DWORD dwLocalIpAddress;
      DWORD dwRemoteIpAddress;
      WORD wRemotePort;

      LPCTSTR szCommunity;
      SNMP_OPERATION_TYPE sotPduType;
      DWORD dwRequestId;
      SNMP_ERROR_CODE secErrorStatus;
      DWORD dwErrorIndex;
      SNMPOBJECTLIST solObjects;

      BOOL bAckReply;
      BOOL bAck;
      BYTE bTimeoutRetry;
      WORD wTimeoutCount;
} MESSAGEQUEUE, *LPMESSAGEQUEUE;
```

Table 3.18 Node parameters of the message queue

Parameter Name	Description
lpNextMessage	It contains a pointer to the next node in the queue list. If this is the last node in the queue, the value of this parameter is null.

lpPrevMessage	It contains a pointer to the previous node in the queue list. If this is the first node in the queue, the value of this parameter is null.
dwLocalIpAddress	This parameter represents the IP address assigned to the interface to which the message is transmitted or on which the message was received. This value is the source destination IP address for outgoing messages and the destination IP address for incoming messages (from the management console point of view).
dwRemoteIpAddress	This parameter represents the IP address of the measurement agent. For outgoing messages, this value will be the destination IP address, for incoming messages it is the source IP address (from the management console point of view).
wRemotePort	It is the value of the UDP port used on the peer station. This value is important only for outgoing messages because the management service requires the remote UDP port in order to encapsulate the SNMP data.
szCommunity	It contains the SNMP community string.
sotPduType	It contains the type of the SNMP PDU. The types used by NMS are *get*, *set* and *response*. Check table 2.9 for additional SNMPv1 PDU types.
secErrorStatus	It contains the value of the error status.
dwErrorIndex	It contains the index of the error, if any.
solObjects	It is a structure of object list type, as previously presented in the sub-chapter about the SNMP service.
bAckReply	It is a flag set to true if acknowledge has been received for the current message.
bAck	This flag specifies whether this message is retransmitted until an acknowledge message is received.
bTimeoutRetry	It specifies the number of times a message is to be retransmitted if acknowledge is not received.
wTimeoutCount	It specifies the start value of a countdown counter that controls the message retransmission.

Some meaning of the parameters may still be confusing, but as the presentation of the queuing service goes along, they will become more meaningful.

The operation of the queuing service starts by initializing all eight queues (four in each direction). Initializing a queue is to empty it. Therefore, at the starting point all queues are empty. The software representation of each queue is made through two pointers that keep the beginning and the end of each queue. This means that when the queues are empty both pointers have null values. Figure 3.31 depicts the initial status of a queue.

Figure 3.31 The initial queue

When a message is added to a queue, either inbound or outbound the following set of steps are performed:

- If the queue is empty, a new node is created (by allocating the required memory for it) and the input and output will reference the same node. For this node, there are no next or previous nodes, these values being null.
- Otherwise, if the queue is not empty a new node will be created and the input pointer will reference it. Since this is the input node in the queue now the input pointer will reference it. The node's previous pointer is set to null, while the node's next node pointer takes the address of the old input node.

Figure 3.32 show how the queue looks like if a node is added to an empty queue, while figure 3.33 presents the same process with a non-empty queue.

Figure 3.32 Adding a node to an empty queue

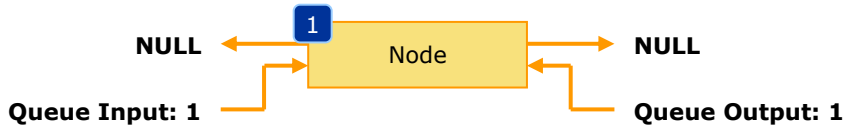
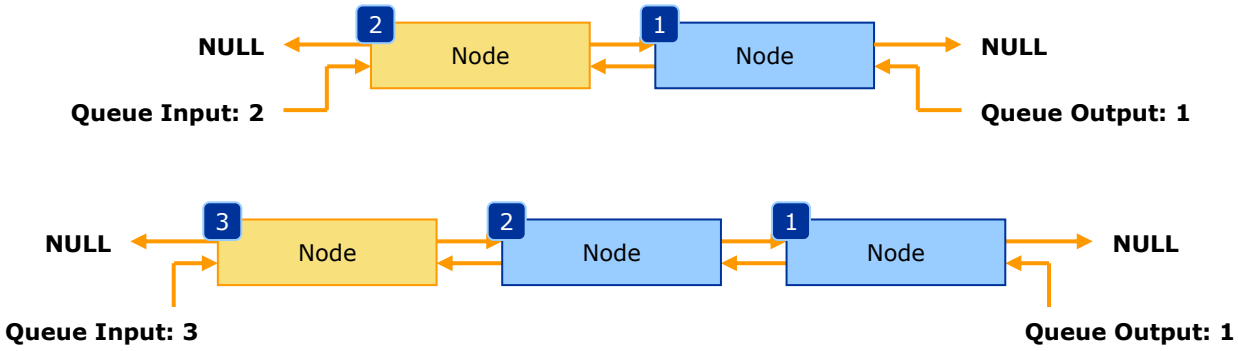
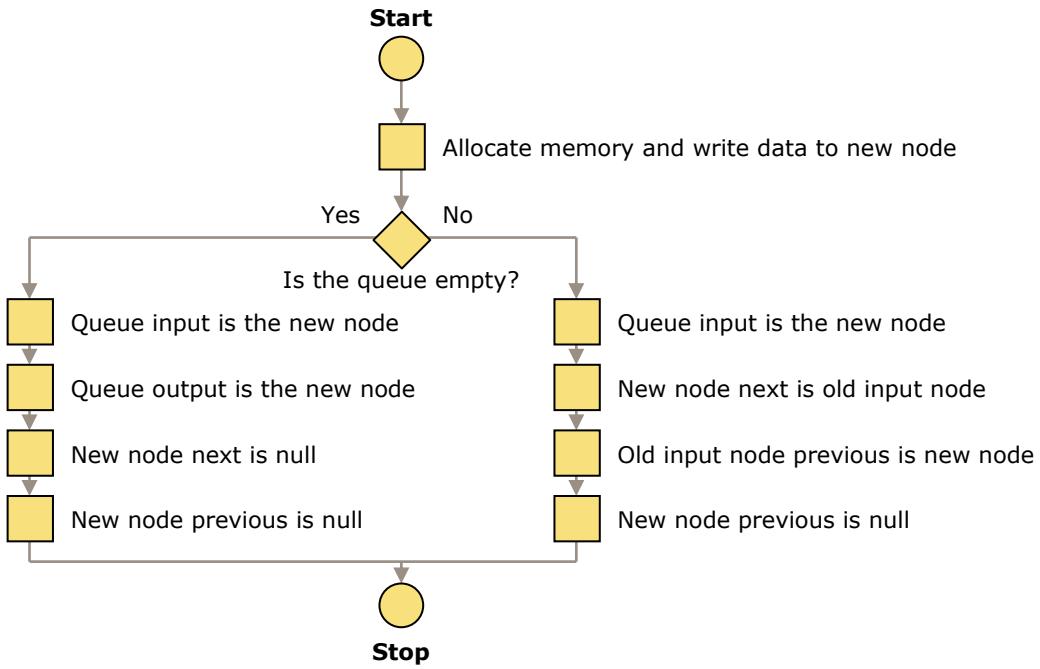


Figure 3.33 Adding consecutively two nodes to the queue from figure 3.31



The algorithm used when adding a new node to the queue is presented in figure 3.34.

Figure 3.34 Algorithm for adding a new node to a queue



Retrieving a message from the queue implies two stages:

- A search in the queue to find the message most closely to the output node that fulfills a set of criteria.
- The extraction of the message from the queue

Practically, the implementation of the management infrastructure from the Network Measurement System uses two different approaches, depending on whether the queue is inbound or outbound.

For inbound queues that hold the messages that came from the network and were placed in the queue by the management service the application needs to perform a test on the queue to find a message according to a set of filtering parameters. At the maximum extent, no filter can be specified and the message closest to the output of the queue with the highest priority is returned. If a filter is used the table 3.19 contains the possible filtering parameters and the filtering type.

Table 3.19 Filtering parameters when searching a message in the inbound queue

Filter	Type	Description
SNMP Request ID	Operation/Flow	The function should return the queue with the message that has the SNMP Request ID equal to the one specified.
Local IP Address	Operation/Flow	The function should return the queue with the message that has the local IP address equal to the one specified.
Remote IP Address	Operation/Flow	The function should return the queue with the message that has the remote IP address equal to the one specified.
Remote UDP Port	Operation/Flow	The function should return the queue with the message that has the remote UDP port equal to the one specified.
Operation Type	Operation/Flow	The function should return the queue with the message that has the SNMP operation type equal to the one specified.
Flow ID	Flow	The function should return the queue with the message that has the flow ID equal to the one specified.

The table 3.19 pointed out that two filtering types are available: namely *operation* filtering and *flow* filtering. The first filtering method can be applied to regular management messages that do not contain any measurement data. In this case, the message can be filter according to IP (source and destination address), UDP (remote port) or SNMP (request ID filed and operation type).

The *flow*-based filtering comes into use when the management messages containing measurement commands or results. The purpose of this filtering is to select between messages that are simultaneously found in the message queue. If several measurement tests are performed on the same agent either the Session Manager or the message dispatcher on the agent need to differentiate between them on a larger basis than just IP addresses, ports and/or request IDs. The next sub-chapter that presents the Session Manager will explain that the basic unit of a traffic generation or analysis is a flow identified through the suggestive name of *flow ID*.

The flow IDs are unique to a management console and a set of agents and are used to identify more precisely each management message. However, if the IP addresses, UDP port, request IDs or operation types are fields in IP, UDP headers or SNMP PDUs, what about the flow ID, where

is it stored? The flow ID it is represented as a managed object inside the SNMP PDU. The current version of NMS assigned the 1.3.6.1.3.3.14.0 OID to the flow ID.

To summarize the inbound queue message filtering keep in mind that:

- Two filtering types are possible: for messages not involved with measurement tests, and messages carrying measurement data
- The first type of messages must *not* contain the flow ID (1.3.6.1.3.3.14.0) managed object and can be filtered according IP addresses, remote port, request ID or operation type
- The messages that contain the flow ID object (1.3.6.1.3.3.14.0) are always measurement related messages and in addition *must* be filtered according to the flow ID value

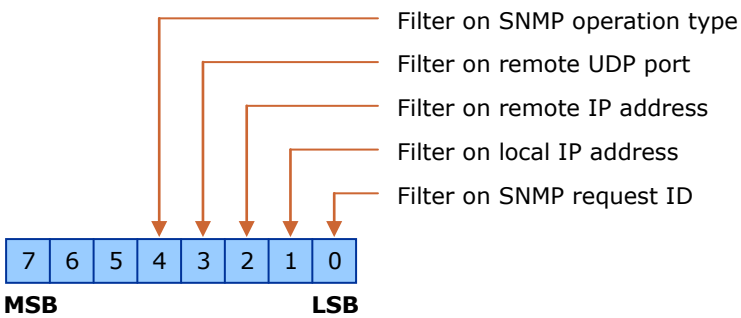
The functions that perform the filtering in the queuing service are the following:

```
BOOL TestOperationInboundMessageQueue(
            LPBYTE lpbPriority,
            BYTE bFilter,
            DWORD dwRequestId,
            DWORD dwLocalIpAddress,
            DWORD dwRemoteIpAddress,
            WORD wRemotePort,
            SNMP_OPERATION_TYPE sotType
            );

BOOL TestFlowInboundMessageQueue(
            LPBYTE lpbPriority,
            BYTE bFilter,
            DWORD dwRequestId,
            DWORD dwLocalIpAddress,
            DWORD dwRemoteIpAddress,
            WORD wRemotePort,
            SNMP_OPERATION_TYPE sotType,
            WORD wFlowId
            );
```

Obviously, the first function performs operation-based filtering while the latter is in charge of flow-based filtering. Both functions return *true* if at least one message matching the given parameters is found. There is no need in explaining the role of each parameter, except the filter parameter, *bFilter*. The filter parameter is a binary mask that specifies according to which values is the search to be filtered. Only the five least significant bits are used (the parameter is byte, hence in 8 bits) and their meaning is given in the next figure.

Figure 3.35 The structure of the inbound queue filter mask



Notes

- If the message contains the flow ID managed object and the first function is used, the message will always be filtered. If the second function is used the message will be filtered according to the specified flow ID value regardless of the value of the filter parameter mask.
- The usage of the filter mask is done by putting a one on the positions of the parameters that will be filtered and a zero on the other ones. The three most significant bits are always ignored. For example, if you want to search the inbound queue on the operation type and SNMP request ID you have to set the 5 LSB to 10001 in binary or 17 in decimal.

The filtered search operation when calling either of the function is done by checking the message node from the output position. If the filter checks out the function returns *true*. Otherwise, if the queue has at least one more node, it moves the message from the output position to the input position and checks again the message from the output position. The messages are being cycled when either a match is found or the original message is on the output position.

If the inbound queue is empty, the search function always returns *false*.

The following three figures illustrate the concept.

Figure 3.36 Searching an empty inbound message queue

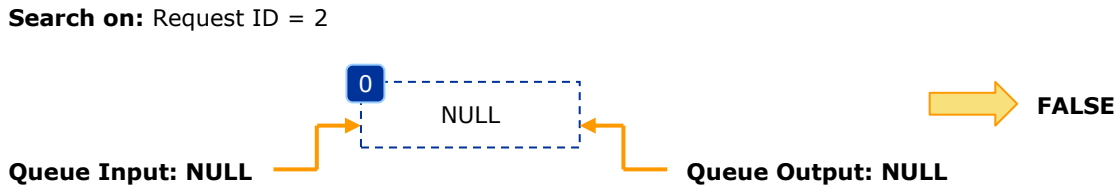


Figure 3.37 Searching a three nodes message queue, searched message exists

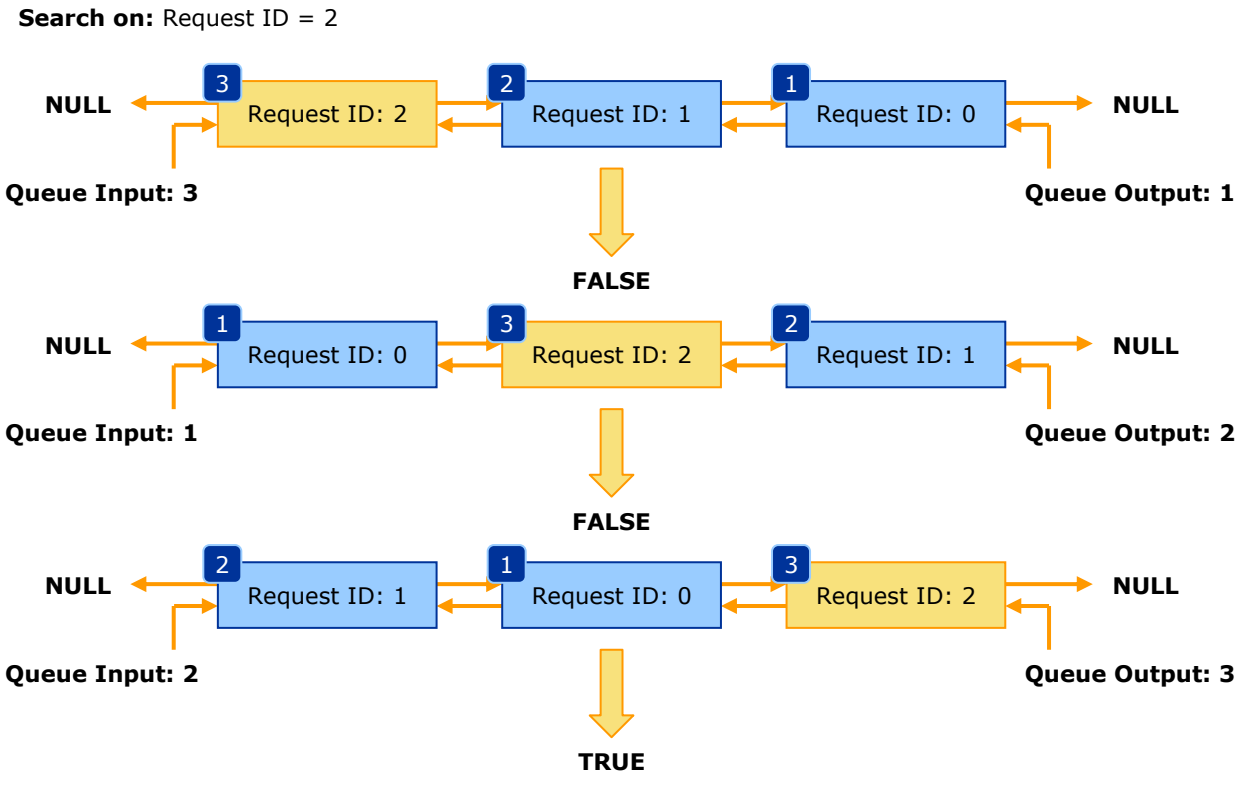
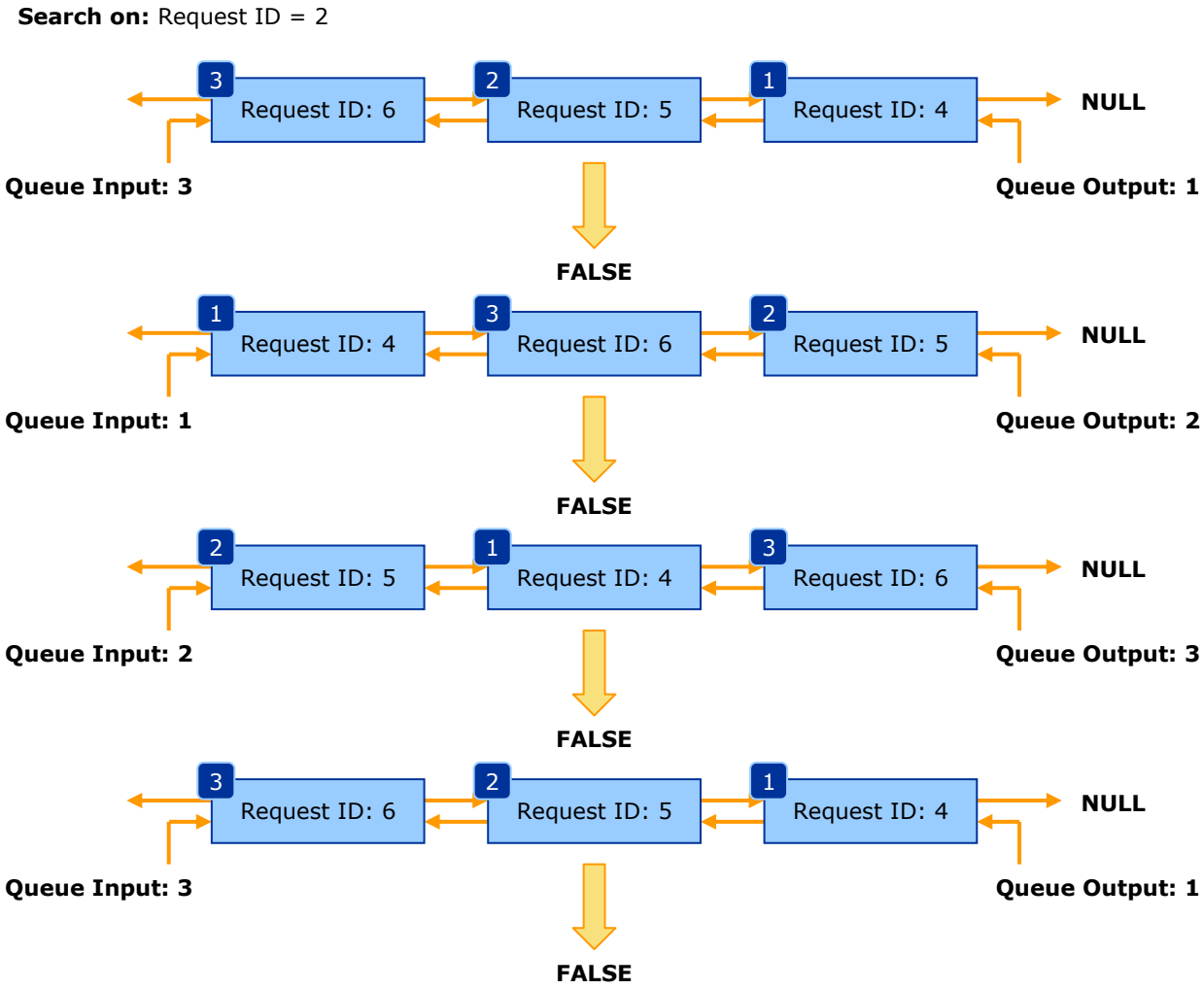


Figure 3.38 Searching a three nodes message queue, searched message does not exist



You can see in from the previous figures that if the searched message is not found in the output position, the message from the output is moved on the input position until a match is found. The search stops when on the output position is the original message (the one with request ID 4 in figure 3.38) and the result of the search is *false*.

The search mechanism presented do far is available in the case of inbound messages. The search is usually performed by the Session Manager but it is not always the case – nevertheless the management console upper application is taking care of inbound messages.

After an inbound message search has been performed if the result of the search is *true*, the queue is automatically locked, meaning that no other queue operations are possible. The queue needs to be locked in order to prevent other threads that might be using the queue of modifying the order of the messages. Imagine, for example, that another thread might perform a search on the inbound queues with another filter parameter. If at least one message exists that fulfills the search conditions, that message will be placed at the end of queue, when the search function returns *true*.

The locking of the queue prevents the change in position of the search result message. This is because the next function that should be called is the function of reading a message from the inbound queue. The reading of an inbound message is the only operation allowed after the inbound queue has been locked, and after the message has been read the queue will be automatically unlocked.

Notes

- Whenever a queue operation is performed, the queue is automatically locked to prevent other conflicting operations. If another thread starts a new queue operation, that thread will be forced to wait until the first queue operation is finished and the queue unlocked. From this perspective, the queuing service means not only the waiting of the management messages in the queuing buffer, but also the waiting of the application threads before they are granted access to the message queuing system.
- The message queues are in general of FIFO type. However, exception occurs, one of them being the search in the list. For this reason, priorities are implemented.
- The access to the queuing service is done on the first-come exclusive basis, meaning that the first thread that begins the operation and locks the queue will be granted exclusive access. When the operation is finished the thread that happens to lock the queue next will also be granted exclusive access and so forth.

Caution

- If you use the existing implementation of the queuing service, when searching for inbound messages, if the search function returns *true* you must always read the message next. Failure to do so will keep the queuing system locked indefinitely.

The search queue not only returns *true* or *false* if the searched message was found but also the number of the queue in which it was found. Remember, that there are four inbound queues with different priorities, and the search function looks in all of them, starting of course with the queue having the highest priority. The read operation that *must* follow, always retrieve the message from the output position: this is why the queue number needs to be specified and the queuing system locked until the searched message is read (otherwise, the message could have been moved from the output position).

The read function of inbound messages is very simple:

```
int RetreiveInboundMessageFromQueue(
            BYTE bPri,
            LPSNMPQUEUEMESSAGE lpsqmMessage
            );
```

It receives a parameter having the number of the queue (the first argument) and returns in the second parameter a pointer to a SNMP message structure, which contains the information of the message found on the output position of the specified queue. The SNMP queue message type is very similar to the structure of a SNMP message having in addition just the local and remote IP addresses and the remote UDP port. The complete structure definition can be found in the source codes from appendix C or table 3.18.

A particular property of the function that reads messages from the inbound queues is that messages are automatically deleted once they are read, because it is assumed that the message reached their destination.

The outbound queues are managed in simpler manner. The messages from outbound queues are read usually only by the management service which does not care about the content of the message. Either message that is found on the output position (i.e. the outbound queue is not empty) can be read and transmitted. However, by comparison with the inbound queue reading mechanism, the messages are not automatically deleted if they require an acknowledgment.

It was presented earlier that four of the flags of each message node inside the queue hold whether the message requires an acknowledgment, the number of retransmission and the waiting period between retransmissions expressed as an integer number of cycles. There are the following cases when a message is deleted from an outbound queue:

- The message does not require an acknowledgement; in this case, it is deleted at the first read operation.
- The message requires an acknowledgment, an acknowledgment has been received in a form of an inbound message and the upper layer application (usually the Session Manager) deleted the message from the outbound queue (i.e. the message is deleted by the same entity that originally created the message and which received the acknowledgment).
- The message requires an acknowledgment and the number of retransmissions left is zero; this means that a timeout occurred and the peer application did not respond. In this situation, a function is available to inform the upper layer application in order to notify the user.

The cases of deleting messages from the outbound queue are illustrated in figure 3.39, 3.40 and 3.41.

Figure 3.39 Deleting outbound messages when acknowledgment is not required

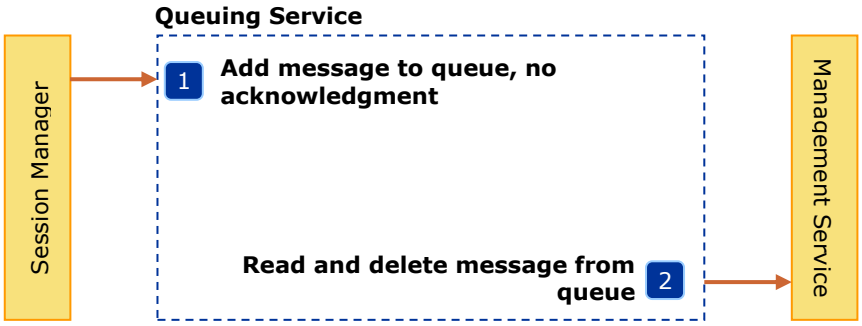


Figure 3.40 Deleting outbound messages when acknowledgement is required and received

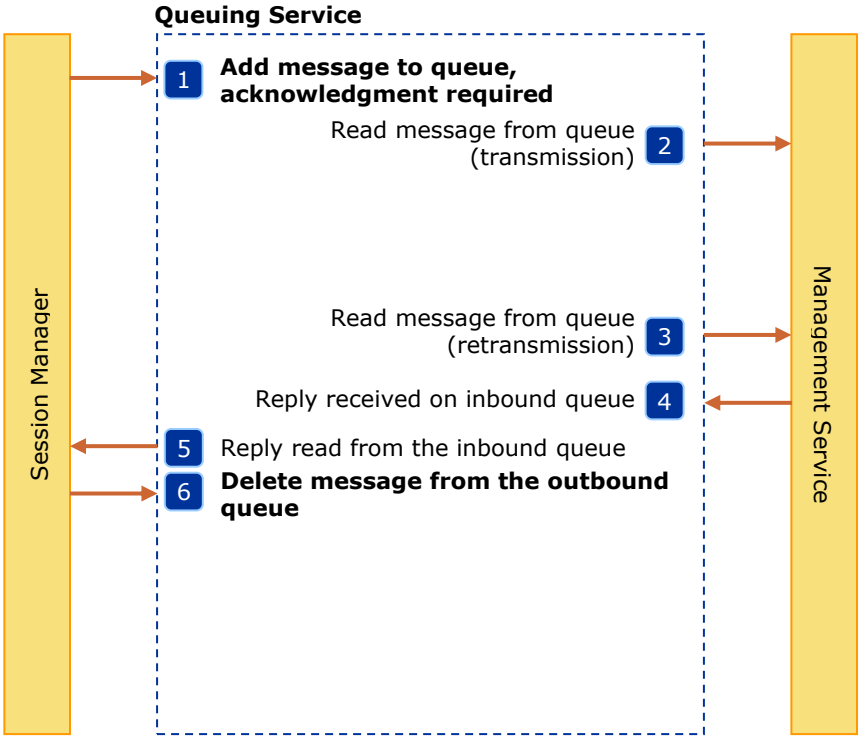
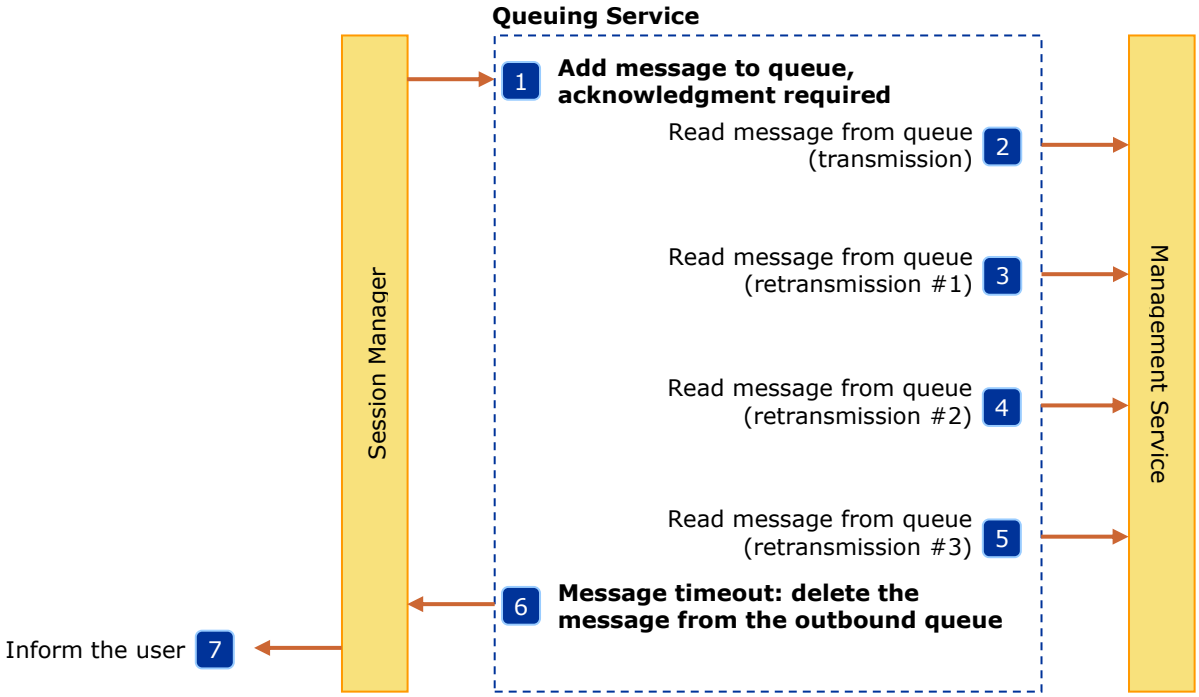


Figure 3.41 Deleting outbound messages when acknowledgment is required but not received



The queuing service notifies the upper application layer whenever a message timeout occurs through a function that should be regularly called.

```
int GetTimeoutMessage(DWORD dwRequestId, LPBOOL lpbFound);
```

3.5.3 IMPLEMENTING MESSAGE RETRANSMISSION

If outbound messages have the acknowledgement required attribute set, the messages are not deleted from the queue as they are read for the first time. Instead, they will be kept for a future retransmission, unless a reply is received in the meanwhile and are deleted from the queue by the upper layer – the Session Manager for the management console.

Two parameters are important when regarding message retransmissions: the number of retransmissions and the time duration between two of them, hence the two parameters that are used for this purpose. This section tries to explain how message retransmission is accomplished.

Notes

- There is no retransmission mechanism for the inbound queues.

Suppose that a message was queued to for transmission. If the queue is not empty after the messages from the output side of the queue are transmitted, our message is finally read by the management service from the queue (remember that the output queue is of FIFO type). Because the message has the acknowledgement attribute set, the queuing service checks the number of retransmissions left and the timeout counter. By default, the initial value of these parameters when the message is added to the queue is five retransmissions and zero for the counter. If the number of retransmissions left is not zero, the service does not delete the message; instead, it

moves it to the input of the queue. In addition, it resets the timeout counter (the default value is 100).

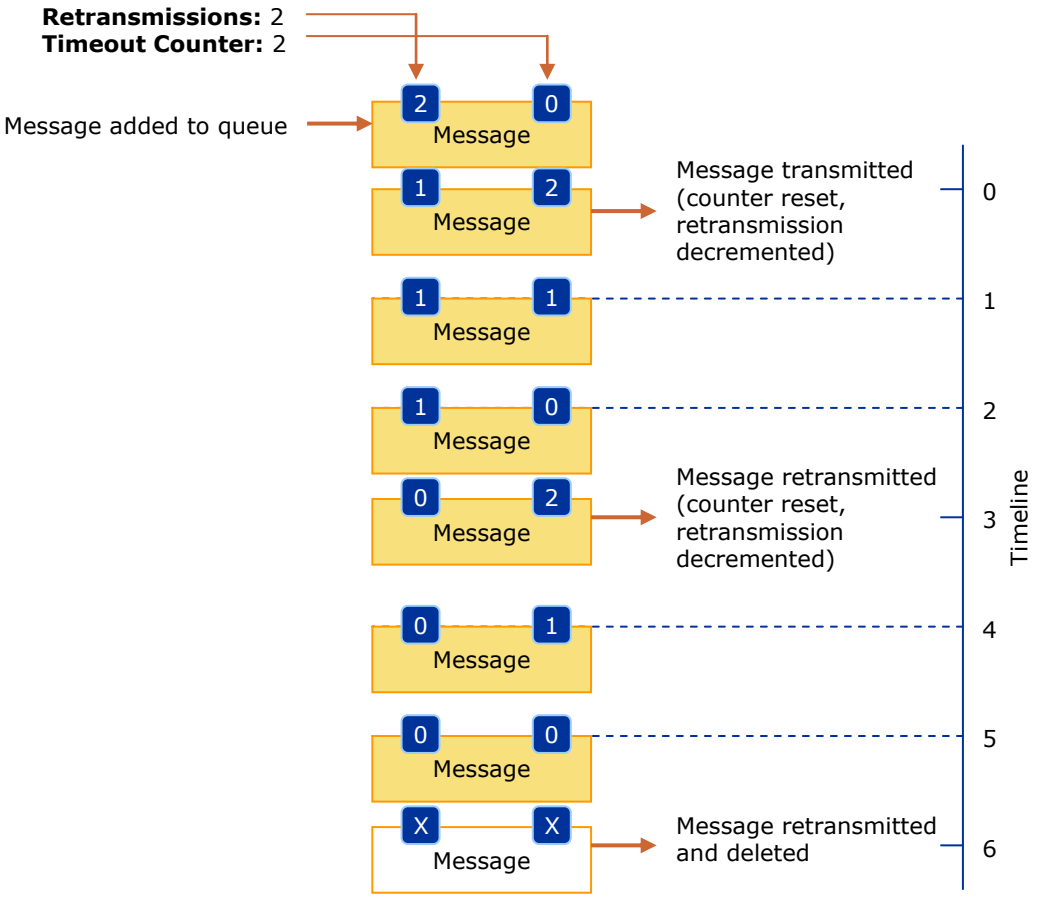
After a while our messages reaches again the output of the outbound queue. However, the message is no longer read this time by the management service to be transmitted because the timeout counter is not zero as in the first case, so the message it would just be moved to the input of the queue again. Relying on what it was explained so far it seems that our message will never have the chance of being transmitted again, unless the value of the timeout counter is decreased.

This is done by an update function of the outbound queues that is executed in the context of the main service thread of the management service. Remember that each thread of the management service has a thread loop. One of the things the main service loop of the management service does in its thread loop is to execute at every cycle the update procedure of the outbound queues. This update function decreases the timeout counter of all messages from the outbound queues until zero is reached.

A cycle of the main service thread last by default approximately 10 milliseconds. Because at each cycle the timeout counter is decreased by one, it takes approximately 1 second until the counter is zero. Because from the outbound queue only messages having a zero timeout counter can be read, means that the next time the messages reaches the output of the queue it will be transmitted again and its retransmission left counter decremented.

The following figure illustrates the retransmission principle, however to keep the figure as simple as possible, was assume that only our message is inside the queue, and a value of 2 for retransmissions and 2 for timeout counter.

Figure 3.42 Messages retransmission principle for a single message in the queue



In the example above it is assumed that no reply is received. In this case:

- The message is deleted from the queue by the queuing service
- The management service is informed that the message was deleted and the message dynamic allocated data can be released
- The upper layer is informed that the message was deleted due to a retransmission timeout

If an acknowledgment reply had received, then the message would have been deleted from the outbound queue by the upper layer at some point between the zero and six time instances on the timeline.

It is not very hard to imagine the retransmission algorithm applied when more messages exist in the queue. What is added is that at any moment the message reaches the output of the queue it is automatically moved at the input unless is deleted and regardless on whether it was retransmitted or not.

3.5.4 DISCARDING INBOUND DUPLICATE MESSAGES

When a retransmission mechanism exits the chances of duplicate messages to occur is very high. Imagine that the time duration between sending a message for the first time and sending s retransmission of it is smaller than the duration between the first transmission and the receiving of the reply. In this case, a reply will finally be received but at least two identical messages were already sent.

The other software on the other machine needs to identify the second message as a duplicate of the one received first and to ignore it. Since the queuing service implements the retransmission mechanism, it is also responsible of detecting and discarding duplicate messages.

From the beginning, before passing to implementation issues it must to be decided what a duplicate message means. In the context of the management infrastructure of the Network Measurement System a duplicate message is a message having the same IP source address, UDP source port and SNMP request ID, and that arrives within a fixed interval after the previous identical message was received.

This interval is usually selected to be higher than the retransmission interval in order to be sure that a retransmission is identified as a duplicate. It also should not be very large for two reasons: first, because some data about each received message is kept until this interval expires (in order to identify duplicates received during this period) resources will be allocated. Second, because after a while numbers such as the SNMP request ID start repeating so non-duplicate messages could be discarding by confusing those messages to the ones received previously. Since the request ID is a 4-byte number, means that messages received from the same IP address and UDP port will start repeating the request ID after 4,294,967,296 messages. Therefore, the duplicate message timeout interval should be less than the time required to transmit the previous number of messages. The same could also happen if one application is restarted and the request ID start repeating some of the old values.

The default value for duplicate message identification interval is around 30 seconds. This interval is made up of two counters and the duration of a cycle of the main service thread of the management service, since the recycle procedure of the duplicate messages it is also executed

within its frame. The first counter is 30 and the second is 100 and by multiplying those two values with 10 milliseconds 30 seconds is obtained.
When a message is received by the management received and it is written to the inbound queue, a record with the information needed to identify duplicate messages is automatically created into additional list containing only information to identify duplicates. Each newly added node in this list receives a timestamp similar to the one used for retransmissions. A recycling procedure is executed within the main service thread once at every 100 (by default) cycles – by default meaning once at every second. Within this recycling procedure, the timestamp of each duplicate record from the list is decreased from the default value of 30.

When the timestamp of a record reaches zero, the duplicate record is removed from the list. Since the timestamp is decremented by default at every second after 30 seconds the record about each received message is deleted and a new message having the same identification parameters (i.e. source IP address, UDP port and SNMP request ID) can be received.

3.5.5 IMPLEMENTING PRIORITIES

The implementation of queuing priorities is very simple and it does not affect what it was presented so far about the other queuing service features. This is done by having multiple inbound and outbound queues, each corresponding to a different priority level. The messages are placed in either queue based on a priority policy. When extracting messages from the queue, the queue with the highest priority is checked first and so on. If a message is found in a queue with higher priority that fulfills the criteria, the less priority queues are not check.

The queuing service has four inbound and four outbound queues. For outbound messages, the decision in which queue to place the message is taken by the upper layers of the application. The default policy for inbound messages in selecting the priority for each message is made depending on the message PDU type. The next table contains the default mapping of each PDU type to each priority level.

Table 3.20 Mapping of inbound PDU types to priority levels

PDU Type	Priority Level	Description
TRAP	0	This is the message with the highest priority level. Trap messages are not used within the Network Measurement System.
SET	1	The *set* messages are the messages with the highest priority that are used by the NMS management.
GET	2	These are medium-priority messages.
GET-NEXT	2	These are medium-priority messages.
RESPONSE	3	These are the lowest priority messages.
Any Other	3	These are the lowest priority messages.

According to their implicit priority, inbound messages are automatically added to the proper queue, when the management service performs the queue operation. For outbound messages, the decision on the priority level is done according to the same basis; however, this could not always be the case.

3.5.6 ENVIRONMENTAL VARIABLES AND SUMMARY

The queuing service features also a set of environmental variables accessible via the *services* view in the management console. At the measurement agent, these parameters keep their default values, which have been chosen to be appropriate to the most encountered situations. Table 3.21 contains the environmental variables of the queuing service and a short description of each of them.

Table 3.21 Default values for environmental variables of the Queuing Service

Variable Name	Default Value	Unit	Description
Message Retransmission Retry	5	retries	The number of times an acknowledgeable message is retransmitted if not deleted from the queue
Message Timeout Counter	100	times	The message retransmission is done once at every *message timeout counter* cycles of the main service thread of the management service
Maximum Queue Size	8192	messages	The maximum size of each queue in number of messages (in total there are up to 65536 messages into the 8 queues)
Message Duplicate Counter	30	times	The record of received messages used to detect duplicates is deleted after *message duplicate counter* recycle intervals
Duplicate Queue Recycle Counter	100	times	This parameter defines the duration of a recycle interval which is *duplicate queue recycle counter* times the duration of cycle of the main service thread of the management service (10 milliseconds by default)

This ends our discussion about the queuing service. As a conclusion, one should remember why the queuing service is necessary: to cope with incoming asynchronous management messages – even messages that arrive at the same moment – and to transmit them synchronously to the server service. The server service is either the management service for outbound messages or the upper layer application for inbound messages and can process only one message at the time. Hence, if more messages need to be processed, there is a single lucky one, while the others wait in the queue for their turn.

In addition to this major role, the queuing service also provides the excellent background in delivering other services such as reliability through retransmission, duplicate message elimination, priorities.

The next sub-chapter presents the last major service of the management console, the Session Manager.

3.6 THE SESSION MANAGER

3.6.1 PURPOSE AND OBJECTIVES

The Session Manager it is only implemented by the management console. At the remote measurement agent, there is another software entity in its place, called the message dispatcher and processing unit but this is described in [2].

The Session Manager it is the point where all management messages start and end. This manager provides services that directly accessible by the user via the graphical user interface of the management console. The basic of the Session Manager starts in the following way. Suppose that the user wants to perform a measurement test. The user must first create something that will be an abstract representation of that test, and which will contain the parameters of the measurement.

Let us call the abstract representation of the measurement test a *measurement session* or simply a *session.* After a session is created the test could either start right a way or could be delayed and start to a specified time moment into the future. To offer the user more flexibility the management console allows the scheduling of each session by creating a *session task* or simply *task*. When a task is created, means that the measurement session is scheduled for execution.

At the right moment, the task we talked about in the previous paragraph starts executing. This implies that a proper set of management messages – according to the variables that define the session – is sent to the measurement agent or agents and replies are expected. Usually each test contain two parts: first in which the parameters of the session are sent and acknowledgments are expected, called *session negotiation* and second in which the management console polls for data from the agents in order to feedback results to the user.

Considering the example discussed the objective of the session manager becomes very clear: to execute *session tasks*. I.e. when the time is right, the Session Manager looks at the task to see which session is used, after that, it reads the parameters of the session and creates successively in state-machine manner management messages (which are sent to the involved agents).

3.6.2 SESSIONS AND SESSION GROUPS

When a measurement is performed, a session and a task are created. The session contains the parameters that describe the measurement test, such as the agents involved, the network interfaces used on each agent, and variables that describe what kind of test is performed. The Network Measurement System supports three different session types, which are presented in table 3.22.

Table 3.22 Session types

Session Type	Agents Used	Description
Traffic Generation	1	This type of session uses just one agent to generate network traffic.
Traffic Analysis	1	This session type uses one agent to analyze the incoming network traffic.

Traffic Generation and Analysis	2	This type uses two agents, one for generation and one for analysis the traffic generated by the first one. This type of session is flow-based, meaning that only the traffic generated by the first agent will be analyzed by the second one.

The following sections will explain in more detail the parameters specific to each session type. For now, keep in mind that these three types exist and when you want to perform a measurement, you have to choose one among them. How each section is created will be presented in the next chapter that explains the user interface and some results that can by obtained using NMS.

After a session is created, you have to create a task based on that session in order to inform the Session Manager that the session can be executed. This approach is also useful in having the session created just once and then executed several times without the need of specifying the session parameters again. Therefore, scheduling multiple sessions can be done by starting a session now within a task, another in thirty minutes, and the third one in two hours, for example. This is achieved only by creating different tasks based on same session and specifying the previous start times.

Nevertheless, there might be situation in which you may want to run the same session several times just by changing something to the original. To anticipate the discussion about the session types, imagine that you create a traffic generating session in which you specify a packet rate of 100 packets per second. If for example, you want to analyze the network behavior for generating traffic having the packet rate between 100 and 1000 packets per second in increments of 100 packets per seconds, the task scheduling does not help too much. Finally you will be constrained of creating 10 session with 100, 200, 300 packets per second and so forth.

The management console features a way that could greatly reduce for you the headache in such a situation. The solution is called *session group* and it is what its name says: a group of sessions. However, it is not only a collection of groups, but also rather a set of measurement session and a relationship between them. Two types of session groups exist – table 3.23 comes with them.

Table 3.23 Type of session groups

Session Group Type	Sessions Used	Relationship
Group of Independent Sessions	Multiple	The relationship is based on time
Group of Parameter Dependent Session	Single	The relationship is based on a user-selected parameter

The first type of session group means that several sessions are represented by a single entity – the group and within the group, the relationship between the sessions is the time. For the sessions that were added the user interface enables to specify the start moment of the session with respect to the beginning of the task, which at this moment is unknown. For each new added session, four options allow you to determine the start moment, see next table.

Table 3.24 Options for selecting the start moment for a session a type I group

Startup Option	Parameter	Description
On Start	No	This session will start at the scheduled moment inside the task. This moment is the reference moment for all the

		session in the group of this type.
With Previous	No	This session will start at the same instance with the previous session in the group. The order of the session in the group is the order of their addition in the group.
After Previous	No	This session will start after the duration of the previous session in the group elapses. Since a session might actually last longer that their specified duration, means that the previous session could not be finished until the next one starts for this startup option.
Within	Number of seconds	The user must specify a number of seconds in which the current session will start from the moment of beginning of the previous one.

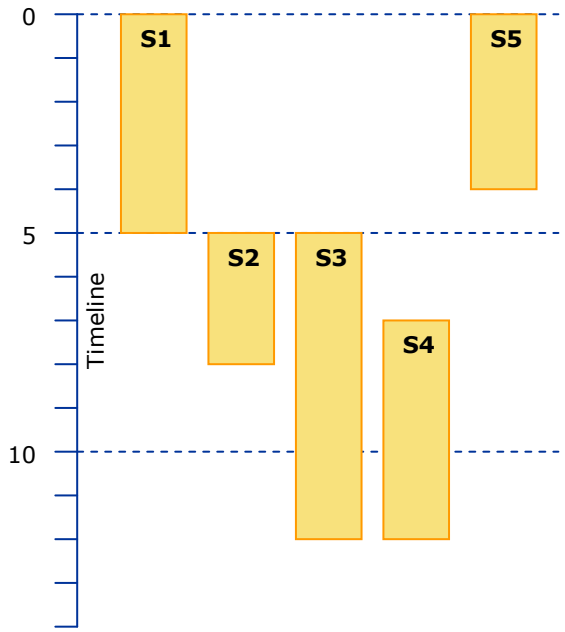
Notes

- The first session in the group is always set as *on start* and the startup option of that session cannot be changed.
- The startup option can be defined for each session in the group.

The following figure illustrates how sessions are executed for a group of independent sessions.

Figure 3.43 Executing a group of independent sessions

Session	Startup	Duration
S1	*On Start*	5 seconds
S2	After Previous	3 seconds
S3	With Previous	7 seconds
S4	Within 2 seconds	5 seconds
S5	On Start	4 seconds



The most used group type is, of course the second one, which allows using just a single session and then to run that session multiple times by changing the value of a single parameter. The parameters as well as the range and increment are selected by the user. Table 3.25 lists the parameters that are available for selection (not all parameters may be available, depending on the session type), and then figure 3.44 depicts how scheduled parameter dependent session groups are executed.

Table 3.25 Parameters available for selection for a type II session group

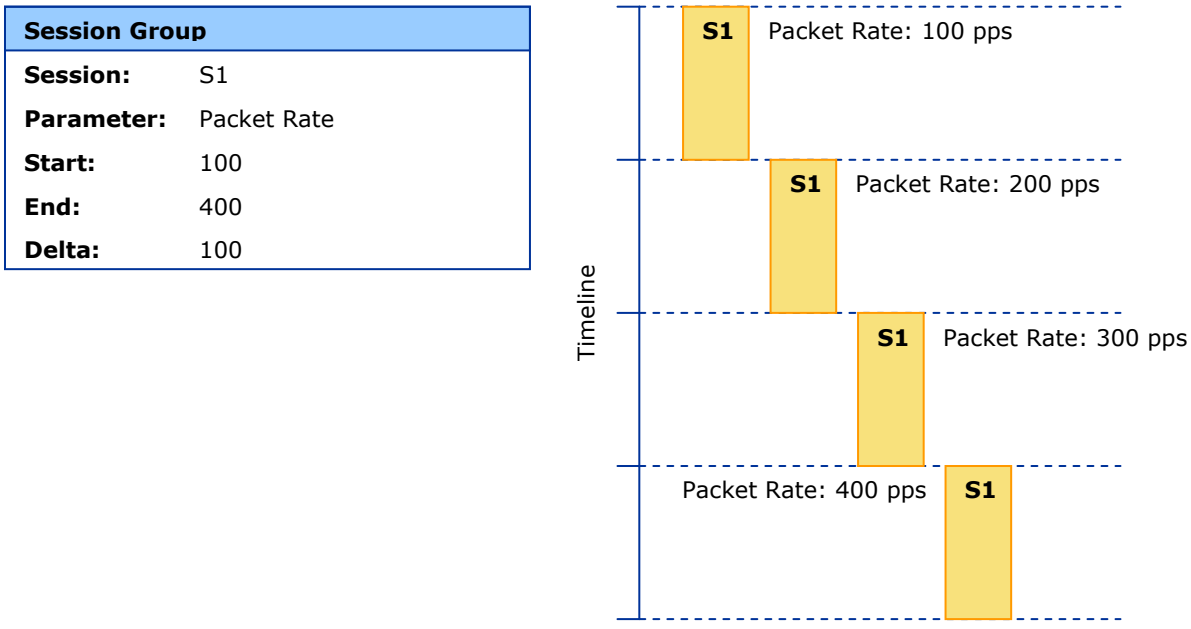
Parameter	Range	Description
Packet Rate	1 ÷ 4,294,967,296	This parameter means either frames per seconds, IP datagrams per second or transport PDUs per second depending on selected traffic
Packet Size	For Ethernet, IP: 64 ÷ MTU For UDP: 1 ÷ 65,536	This parameter specifies the used protocol header and PDU size
Test Duration	1 ÷ 4,294,967,296	This is the duration of the measurement test, not of the session which is longer due to the parameters negotiation
Protocol/Length	For Ethernet: 0 ÷ 65,536 For IP: 0 ÷ 255	This is the value of the protocol/length field from Ethernet/IEEE 802.3 or IP headers
Time-to-Live	For IP: 0 ÷ 255	This is the value of the TTL field from IP header
Type-of-Service	For IP: 0 ÷ 255	This is the value of the TOS field from IP header

Notes

- For a group created with a specific session not all parameters may be available. Later in this sub-chapter when the types of sessions will be presented, the available group parameters will be specified.
- The parameter range used in the experiment is set by a start and an end plus a delta. It is not necessary the start parameter is less than the end value, but in this case, the values are accepted only if the delta is negative.

Suppose that you create a session group based on a parameter, let us choose the packet rate, for example, with a range between 100 and 400 packets per second with an increment of 100 packets per second. The next figure illustrates the group execution.

Figure 3.44 Executing a group of parameter dependent session



3.6.3 TRAFFIC GENERATION SESSIONS

These types of sessions perform traffic generation. The first question should be what kind of traffic is it generated? The answer is that the Network Measurement System features three options for traffic generation, depending on the layer up to which the traffic is encapsulated:

- Ethernet/IEEE 802.3 or layer two traffic, meaning that raw frames are sent through the network
- Internet Protocol version 4 (IPv4) or layer three traffic, in which case IPv4 packets are sent through the network
- User Datagram Protocol (UDP) or layer four traffic, by using UDP over IPv4

Why are different types of traffic necessary? Well, because if you want, for example, to test the link between two machines connected directly (they are within the same network segment) there is no need of using a network layer protocol (IP in this case), unless you want to test the performance of the IP implementation software from the operating system as well.

If at least one router separates the two machines involved in the test you must go with a layer three protocol that supports routing, IP in this case. Finally, if the network load is made with many different flows, in the way the destination could confuse the generated packets with other packets, you may go for a transport layer protocol, in which the multiplexing mechanism ensured by the UDP port number allows the destination application to filter the packets destined to it. The table 3.26 contains the possible encapsulations available for traffic generation.

Table 3.26 Encapsulation options for traffic generation

Traffic Type	Layer	Description
Ethernet / IEEE 802.3	Data-link	Generates Ethernet / IEEE 802.3 depending on the value from protocol/length field from the header
Internet Protocol version 4	Network	Generates IPv4 datagrams
User Datagram Protocol	Transport	Generates UDP segments

The parameters for a traffic generation session are divided in two parts:

- Common parameters, that are used regardless of the session type
- Specific parameters that might be available or not depending on which session type is selected

The following tables contain these parameters, starting with the common ones and then specific for each session in part. There is also mentioned whether a parameter is editable or not (for example parameters such as source addresses always exist but hey cannot be modified).

Table 3.27 Common parameters for generating sessions

Parameter	Editable	Description
Test Duration	Yes	It expresses in seconds the duration of a measurement test (not of a session).
Packet Distribution	Yes	It selects the packet distribution used when packets are generated. Three types of packet distributions are available:

		Periodic distribution, in which the time spacing between the packets is approximately constant and given by the reciprocal of the packet rate Poisson distribution, in which the time distance between the packets follow the Poisson probability law Link flooding, in which the packets are generated respecting no distribution alone but sending them up to the available capacity of the link
Packet Rate	Yes	It is used in computing the packet spacing for the periodic distribution, and this value is considered the average packet rate in the case of Poisson distribution.
Poisson Distribution Approximation	Yes	It is available only if the Poisson distribution has been selected, and represents the method used in minimizing the error between the used packet distribution and the Poisson packet distribution. For more information on Poisson approximation, methods consult the appropriate topic from [2].
Real-time Failure Policy	Yes	It represents the action taken by the traffic generation software from the agent in correcting the moment of packet transmissions, when the packet distribution can no longer be respected due to a system bottleneck or because the link capacity has been reached.

For Ethernet/IEEE 802.3 traffic, the next parameters are added.

Table 3.28 Specific parameters used for Ethernet/IEEE 802.3 traffic

Parameter	Editable	Description
Source Physical Address	No	It represents the MAC address of the source (in this case of the network interface card used on the transmitting agent).
Destination Physical Address	Yes	It represents the MAC address of the NIC from the destination host or it can be broadcast.
Frame Size	Yes	It is the size of the Ethernet / IEEE 802.3 frame. It must be larger than 64 bytes and it cannot exceed the value of the maximum transmission unit.
Maximum Transmission Unit	No	It is the maximum size of a frame, which could be transmitted by the networking hardware.
Protocol/Length	Yes	It represents the contents of the 16-bit protocol/type field from the frame header.

When using IPv4 instead of the parameters from table 3.28 the following are available.

Table 3.29 Specific parameters used for IPv4 traffic

Parameter	Editable	Description
Source IP Address	No	It represents the IP address assigned to the network interface where traffic is being generated. If an IP address is not assigned the IP traffic generation option will not be available.
Destination IP Address	Yes	It represents the IP address assigned to the NIC of the remote host, where the packets are sent or it can be broadcast/multicast.

Datagram Size	Yes	It is the size of the IPv4 datagram, header included.
Protocol Field	Yes	It represents the value of the 8-bit protocol field from the IPv4 header.
Time-to-Live	Yes	It represents the value of the 8-bit TTL field from the IPv4 header.
Type-of-Service	Yes	It represents the value of the 8-bit TOS field from the IPv4 header.
Do Not Fragment	No	The *do not fragment* flag in the IPv4 header is always set.
Identification	No	This field represents the identification number of the IP fragment. Since the datagram cannot be fragmented, this value cannot be modified and its value is undefined.

For transport layer traffic, the options are in table 3.30.

Table 3.30 Specific parameters used for UDP traffic

Parameter	**Editable**	**Description**
Source IP Address	No	It represents the IP address assigned to the network interface where traffic is being generated. If an IP address is not assigned the IP traffic generation option will not be available.
Destination IP Address	Yes	It represents the IP address assigned to the NIC of the remote host, where the packets are sent or it can be broadcast/multicast.
Source UDP Port	Yes	It represents the value of the UDP port number within the generated UDP segment.
Destination UDP Port	Yes	It is the value of the destination port number within the UDP segment.
Segment Size	Yes	It is the size of the UDP datagram, header included.

When you begin creating a session, first the session type must be selected (i.e. generation, analysis, or both) and then you have to choose the layer type on the case of traffic generation or generation plus analysis. After which this information was provided the parameters specific to each traffic type could be introduced. At the end, the common parameters should be provided and afterwards the generation session setup is completed.

The results available when a traffic generation session is executed are:

- The number of packets that were sent from the beginning of the session
- The number of real time failures, i.e. for how many packets the time distribution could not be respected due to insufficient link capacity or processing power

The next section will describe the second session type, for traffic analysis measurements.

3.6.4 TRAFFIC ANALYSIS SESSIONS

These types of sessions are very simple in the sense they require less input data at the setup and provide at least the same information as for traffic generation session. Traffic analysis sessions are used to perform passive measurement, i.e. measuring the throughput of existing incoming traffic in the network.

The background and implementation of the analysis methods will not be discussed here; therefore, for a through description consult the appropriate topic in [2]. However, it should be only mentioned that two types of analysis sessions exist:

- Traffic analysis sessions performed using regular Ethernet/IEEE 802.3 cards
- Traffic analysis sessions that used the Endace® DAG card

The parameters required by these analysis sessions are presented in table 3.31, and the result available for both types of cards is always only the throughput.

Table 3.31 Parameters of traffic analysis sessions

Parameter	Editable	Description
Test Duration	Yes	It expresses in seconds the duration of a measurement test (not of a session).

Since incoming messages are captured and analyzed regardless of other parameters, no additional parameters are required.

Notes

- For traffic generation sessions the hardware supported is only the regular network interface cards (Ethernet/IEEE 802.3)

3.6.5 TRAFFIC GENERATION AND ANALYSIS SESSIONS

These types of sessions are the most complex one, because they involve two agents working together, one generating the traffic, the other capturing the traffic.

The session parameters that need to be provided are the parameters from the traffic generation sessions and in addition, one may specify the results that should be available after the measurement test is complete. These sessions support the three types of traffic generation mentioned before, while the analysis can be done with either regular NICs or the Endace card. However, for the user the selection of the card is very straightforward – both cards are selected from the same list, and then the agent application cares in using the appropriate traffic capturing software.

Two analysis methods are available when double generation and analysis sessions are used:

- A real-time analysis in which parameters are received in real time from both agents – this implies a time domain description of the QoS parameters
- A dump analysis that saves the data on the hard drive and performs an offline analysis upon it after the test is completed.

For the real-time method, the user may also specify whether the session is a simple one or an advanced one, meaning that supplementary QoS traffic parameters are computed.

These results are available from a simple, real-time session:

- The transmitted number of packets from the beginning of the session
- The real time failure at the transmission
- The received number of packets from the beginning of the session
- The incoming throughput of the analyzed flow between two consecutive readings
- The average one-way delay
- The average packet delay variation

In addition, if advanced real-time session is specified the following results are available:

- The minimum and maximum of the one-way delay
- The minimum and maximum of the packet delay variation
- The number of out-of-order packets

Table 3.32 shows which of the parameters are available when either simple, advanced or offline analysis method is selected.

Table 3.32 Availability of results with respect to selected analysis method

Parameter	Simple Online	Advanced Online	Offline
Average Throughput	Yes	Yes	Yes
Received Number of Packets	Yes	Yes	Yes
Average One-way Delay	Yes	Yes	Yes
Minimum One-way Delay	No	Yes	Yes
Maximum One-way Delay	No	Yes	Yes
Average Packet Delay Variation	Yes	Yes	Yes
Minimum Packet Delay Variation	No	Yes	Yes
Maximum Packet Delay Variation	No	Yes	Yes
Out-of-order Packets	No	Yes	Yes

Notes

- Between the two types of online analysis at least one is always performed. However, when selecting a simple method of analysis not all parameters from table 3.32 are computed, increasing the results accuracy since the agent software allocates the processing resources towards the increasing of analysis precision.
- The offline analysis can be either selected or not. Using offline analysis permits the computation of the rest of parameters when it is used in combination with a simple online analysis.

- The selection of parameters is available only at the receiving agent, i.e. the one that captures and analyzes the active traffic. At the generating agent, the both parameters (the number of transmitted packets and the number of real-time failures) are always available. Note, that since data about the transmitted and received number of packets is always collected in real-time from both agents, the management console can compute parameters such as the packet loss and the packet loss ratio.

Important

- Keep in mind that performing an offline analysis gives some processing slack to the analyzing agent; it implies dumping information about each incoming packet on the hard drive. When the incoming data rate exceeds the data rates of writing to the magnetic medium (or other type storage), information about the packets will be lost and error will occur in the processing of the information. Even the agent alerts the user in such situations, the test is nevertheless compromised.

3.6.6 ANALYZING TRAFFIC WITH ENDACE CARDS

From the management console point of view, the Endace DAG interface cards are not very special. This is one of the reasons this project neither emphasize their features nor provides additional information such as how these cards are used with the tools provided by the manufacturer. In the first chapter, the example scenario was enough to understand the basics.

Of course, for the measurement agent using a regular NIC or a DAG means completely different implementation, algorithms and data processing. But the role of the distributed architecture of the Network Measurement System is to separate the functions that interact with the hardware, generate packets, analyze flows and so on, by the management part that contains the intelligence of gathering QoS results, putting them together and presenting them into an accessible manner to the user.

The management console does not make any difference whether the agent measures the throughput with the expensive DAG or with the cheapest NIC on the market. The difference lies only in the quality of the result and obviously additional function at the agent. If you are interested to find out more on the DAG card, how it is used and some details about the implementation of the software that uses it, consult [2].

Nevertheless, for purposes of user interface response, the management console is aware to some extent that a DAG is used. This is related to the fact that when using an Endace card the following are true:

- The traffic generation option cannot be selected for an Endace DAG interface card
- The method of online analysis at the receiving agent is always advanced (i.e. when capturing traffic with the DAG the minimum/maximum one-way delay, packet delay variation and out-of-order packets are automatically computed).
- The control and return parameters used for the DAG are assigned a different set of managed objects. Hence, the Session Manager will rely on different functions when assembling messages that are sent to a DAG than when using a regular NIC.

Except these exceptions no knowledge of the fact that a QoS parameter was measured with a DAG rather than an ordinary card is ever kept.

3.6.7 IMPLEMENTING SESSIONS, GROUPS AND TASKS

When the user schedules a command, the Session Manager must verify, at first, which is the moment when the task must start. The task is based either on a session or on a session group by keeping a reference to the already loaded sessions or groups. The reference is:

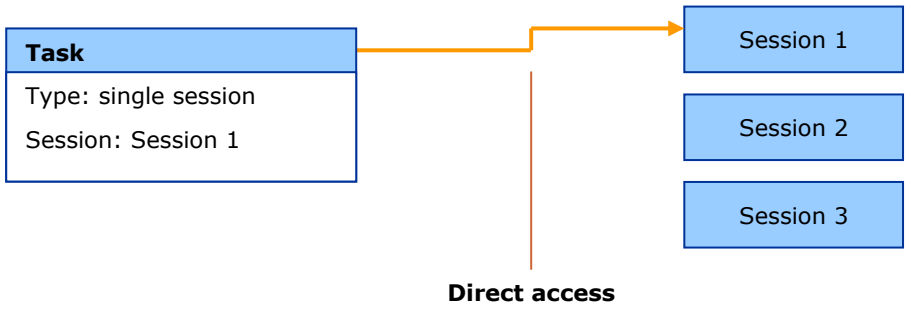
- If the task is based on session, it is the index of the session. The session index is the session ID displayed by default in the first column of the management console.
- If the task is based on a session group and the group is made up of individual sessions, it is the group ID. By accessing the group via the group ID, the task can obtain the list of sessions and finally the properties of each test.
- If the task is based on a group of a parameter dependent session, the implementation becomes more complex.

In the first two situations, the things are simple because no parameters of either session are changed. Keep in mind that the session is the basic unit of a test, since the session keeps the data according to which the experiment can be executed: used agents and interfaces, type of tests, type of traffic, traffic parameters. The groups are merely a way of simplifying the way of doing tests from the user point of view.

When the user schedules a group made up of parameter dependent session it means that the session must change the parameter that was selected within the group and successively take the values within the specified group range in increments or decrements of delta. However, the task cannot modify directly the data within the session due to two reasons: if the data would be modified, the original version of the session will be compromised and second, other tasks may be using the same session.

Bottom line, tasks cannot modify either the sessions or the session groups. Instead, for situations like the one of a parameter-dependent group, they must make a copy of the data they partially intend to modify. The copy of the original session is used only by the task that created it, and will not be available either in the session list, or to the other tasks. Because of this, this type of session is called a *virtual session*. The following figures show how sessions are accessed by the tasks.

Figure 3.45 Accessing sessions: task of single session



The figure 3.45 illustrates the situation of the single session task in which the properties of the session are accessed directly, since they are not modifies. This is also the situation of a group made up of independent sessions, because the properties of the sessions do not chance in that case too. Figure 3.46 suggests that.

Figure 3.46 Accessing sessions: task of independent sessions' group

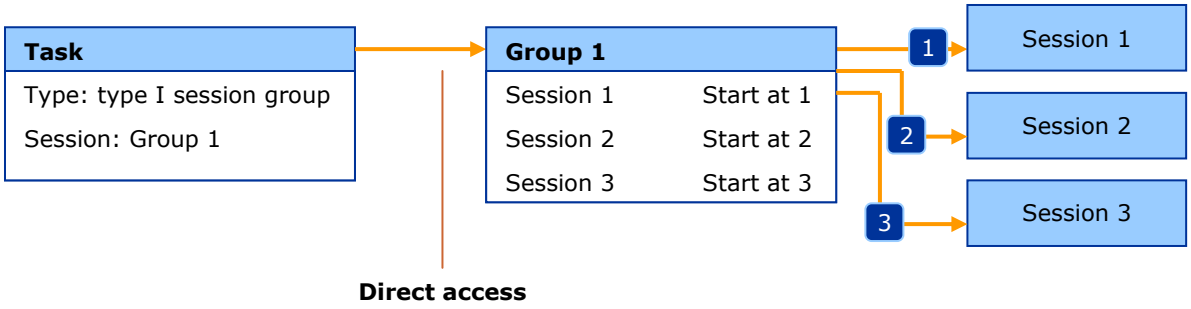
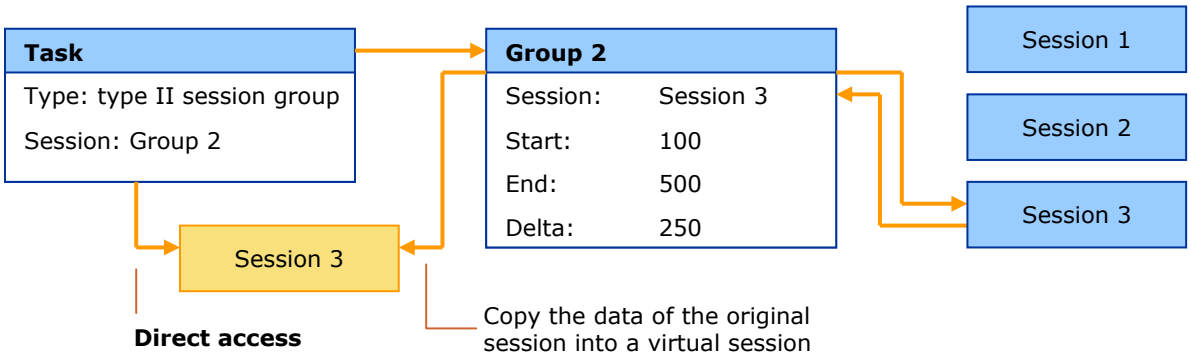


Figure 3.47 Accessing sessions: task of a parameter dependent session group



The last example expresses the idea already explained: when the data from a session needs to be modified the tasks copies the whole session into a new one. However, because the new session is not available to other tasks or to new groups, it is said to be virtual.

The Session Manager has several exportable functions, which other modules, like the user interface can use to create sessions, groups and tasks. The headers of these functions are given below, while the complete implementation can be found in appendix D.

```
void CreateSession(LPSESSIONRECORD lpRecord);
void SaveSession(LPVOID hwndMainWnd, LPVOID lpSelected);
void LoadSession(LPVOID hwndMainWnd);
WORD GetSessionCount(void);
SESSIONRECORD GetSession(WORD wIndex);
```

Table 3.33 Functions of the Session Manager for managing sessions

Function Name	Description
Create Session	It is used to create a new session. The parameter is a pointer to a session record that contains the session information. The definition of the session record is specified in appendix D. After this function is executed, the session will be automatically added to the session list in the console interface.
Save Session	It is used to save the selected session from the user interface to a file.
Load Session	It is used to load a session from file. After this function is executed is automatically added to the session list, in the user interface.
Get Session Count	It is used to return the number of existing sessions.
Get Session	It returns the data of a session as a session record, when given the index of a session. The index must be lower than the current number of sessions.

```
void CreateSessionGroup(LPSESSIONGROUP lpGroup);
WORD GetSessionGroupCount(void);
SESSIONGROUP GetSessionGroup(WORD wCnt);
```

Table 3.34 Functions of the Session Manager for managing session groups

Function Name	Description
Create Session Group	It is used to create a new session group. The parameters of the group, such as the type and component sessions, are specified in the session group parameter. The structure of the session group type are specified in appendix D.
Get Session Group Count	It returns the number of existing sessions.
Get Session Group	It is used to obtain the parameters of the session group, given the group index parameter. The index must be lower than the maximum number of session groups.

```
void CreateScheduledTask(LPTASKRECORD lpTask);
WORD GetScheduledTaskCount(void);
TASKRECORD GetScheduledTask(WORD wCnt);
```

Table 3.35 Functions of the Session Manager for managing scheduled tasks

Function Name	Description
Create Scheduled Task	It is used to create a new scheduled task. The parameters of the task, such as the type and reference to the session or session group, are specified in the task record parameter. The structure of the task record type are specified in the source code from appendix D.
Get Scheduled Task Count	It returns the number of existing scheduled tasks. This number includes the tasks that were executed in the past and are now in finished state.
Get Scheduled Task	It is used to obtain the parameters of a scheduled task, given the task index parameter. The index must be lower than the maximum number of scheduled tasks.

In addition to the functions above, functions providing the following services are also available:

- Obtaining the status of a service or service group, which could be: not scheduled or scheduled, in the last situation the task ID to which the session/session group is assigned can be obtained as well
- Deleting a scheduled task, prior to its start

Caution

- The management console allows you to delete a task as long as it is not started. After deletion, the task will be marked as deleted but will not be removed from the task list. Nevertheless, neither you can delete a task after it started nor you can stop it. In the situation, an unwanted scheduled task started, and you want to cancel it from some reason without closing the management console, you should stop the management services in the Services view.

3.6.8 IMPLEMENTING THE SESSION MANAGER

The goal of the implementation of the Session Manager is to execute scheduled tasks. The Session Manager does not feature its own execution flow in order to run; rather is based on another service: the management service. The reason for this is that by comparison with other services, the Session Manager has always the execution initiative – this is the software component that handles all the measurement.

Because the Session Manager is most of the time user interactive, it does not require a fined-grained temporal flow in the multitasking environment. Instead, it suffices to execute the required code from time to time and then using the processing power for other objectives rather than keeping a single thread in idle.

The Session Manager has a single function that must be executed periodically in the context of other services that implements stand-alone threads. For the management console, this is the management service. This approach has to implications, which are neither bad:

- When the management service is stopped or pause the Session Manager is also stopped or paused. This condition is nevertheless necessary since if one endpoint of the queuing service is stopped (the management service) the other one – the Session Manager – could not be up and sending messages to the queue. This is also available in the reverse way.
- Second, because the job of the Session Manger is done on behalf of processing time of another service, it must execute at large intervals, in such a way the host thread (and service) is not affected. The Session Manager is intended to create management messages whenever the user gives a command and during a task is run: to collect real time data from the agents. Since neither collecting data from the agents nor user commands must be executed within milliseconds, running the Session Manager allows for a much coarse time spreading.

Notes

- You should remember that the queuing service featured also several maintenance functions that were executed within the main context of the management service, thus underlining the approach that when the management service is stopped (or paused) the entire management data halts.

The function used to take control from the main service thread of the management service is given below:

```
void ScheduleTask(void);
```

This function is executed by the main service thread of the management service at every cycle, in order to yield control for a brief moment to the Session Manager. Because the cycle duration of a control thread of the management service is too small (10 milliseconds, by default) and because the Session Manager does dot require such fined-grained execution, only once at a given number of cycles will the schedule task function do its job; otherwise will just exit.

The session tasks are scheduled with second accuracy. In addition, it is believed that for running measurement tests no higher real time resolution than one second is necessary, for the following reasons:

- In order not to flood the network with management messages, especially when the number of tasks is very large

- The update of the real time data computed by the measurement agent, is done most frequently once at every second; hence, requesting data often that a second interval would yield the same information
- Computing and transmitting real time information often than once per second implies higher processing load, for the agent, decreased performance and poor accuracy
- The retransmission interval for an acknowledgeable management message is one second by default

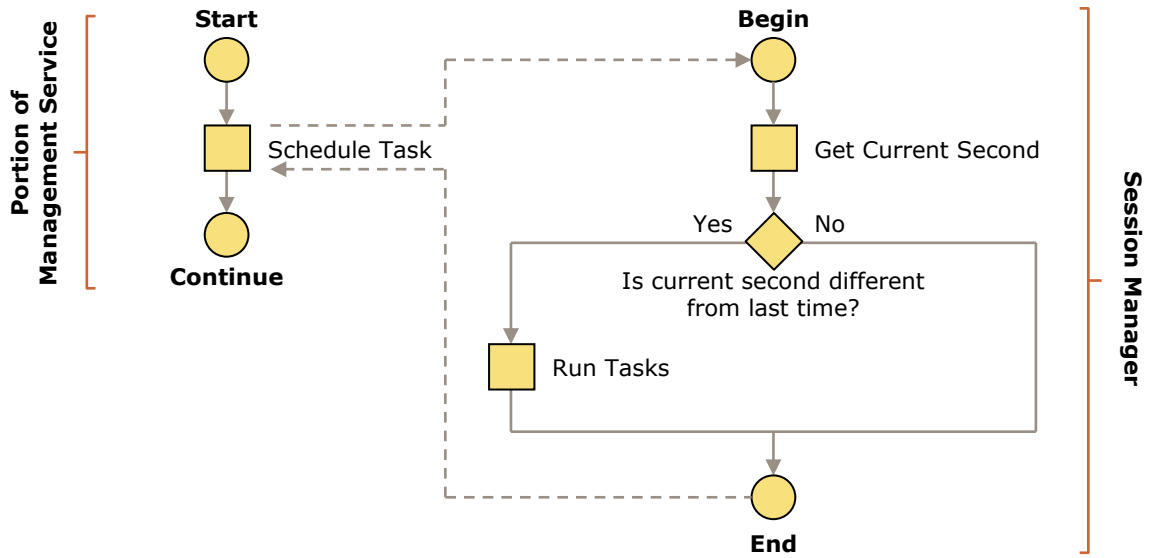
Notes

- The agent documentation [2] shows that the implementation of the agent supports the computation of real time QoS parameters at a much higher rate that once per second. The decision not to use this feature was to prevent the performance degradation on one hand and to synchronize with the retransmission mechanism.

When the management requests for online data it sends a request message and waits for a reply. The request messages are acknowledgeable. Final argument, since the Session Manager will not send the next packet until an answer for the first one is received and because at least one second is waited by default to receive an answer, there is no need of even trying in sending request messages more often.

However, because the cycle of the service control thread is much less than a second, implies the schedule task function is executed at a much higher rate. Some algorithm is required to prevent the function doing its main job unless more than a second elapsed from the last time. The algorithm is presented in figure 3.48 along with the presentation of the portion of the management service that releases control for a small period to the Session Manager.

Figure 3.48 Scheduling task timing and control



Now let us pass to the code, which implements the main objectives of the Session Manager: to execute session or session groups tasks. The first aspect to be considered comes from the name of what the Session Manager is executing *scheduled tasks*. Because the tasks are scheduled, means that the user might setup a task right now, but will be executed tomorrow. Each task can be scheduled to a specific moment in time, either in the past, present or future.

The timestamp identifies when a task should start executing. It is kept in the task record along with the task type (of session or session group), session or group reference, reference to a virtual session if the group is of type II, and some other parameters among which one is the task status that is to be discussed in a few moments. The timestamp is expressed in multiples of seconds (actually, it is the time in Epoch format – the number of seconds that passed since January 1, 1970, 00:00:00 hours), this being one of the reasons why the session manager should not execute more often than one second, at least from the starting tasks point of view.

Another parameter of a scheduled task is its status. Three status values have been defined and are presented in table 3.38. The status tells the Session Manager in which condition the task is, whether it should start it by performing some initialization (remember that before a measurement test is started some negotiation with the agents is required), to run it or ignore it.

Table 3.36 Scheduled tasks status values

Status Name	Value	Session Manager Action
Pending	0	At each execution cycle the Session Manager checks to see whether the start time of the task is less or equal with the current time. If so, the task is started and its status set to *running*.
Running	1	The task initialization already occurred, at each cycle the Session Manager performs a task specific operation. The tasks keep an identification of their execution flow, and the Session Manager use this information to execute them.
Finished	2	After a task is finished, wither normally or with an error, thy pass to *finished* state. The Session Manager ignores any tasks that are in finished state.
Deleted	3	The user can delete a task while it still is in the pending state. When the task is deleted, it is not removed from the task list, but it is set to the *deleted* state. The Session Manager ignores any tasks that are in deleted state.

Once at each second the scheduling task function runs the task, i.e. in the figure 3.48 the execution flow goes through the left branch of the conditional block. In the run task procedure, each existing task is checked for its status. If the task is in pending status, the Session Manager checks its start time against the local time of the system. The task is started when the local time is greater or at least equal with the startup time of the task. Otherwise, the task is ignored for now.

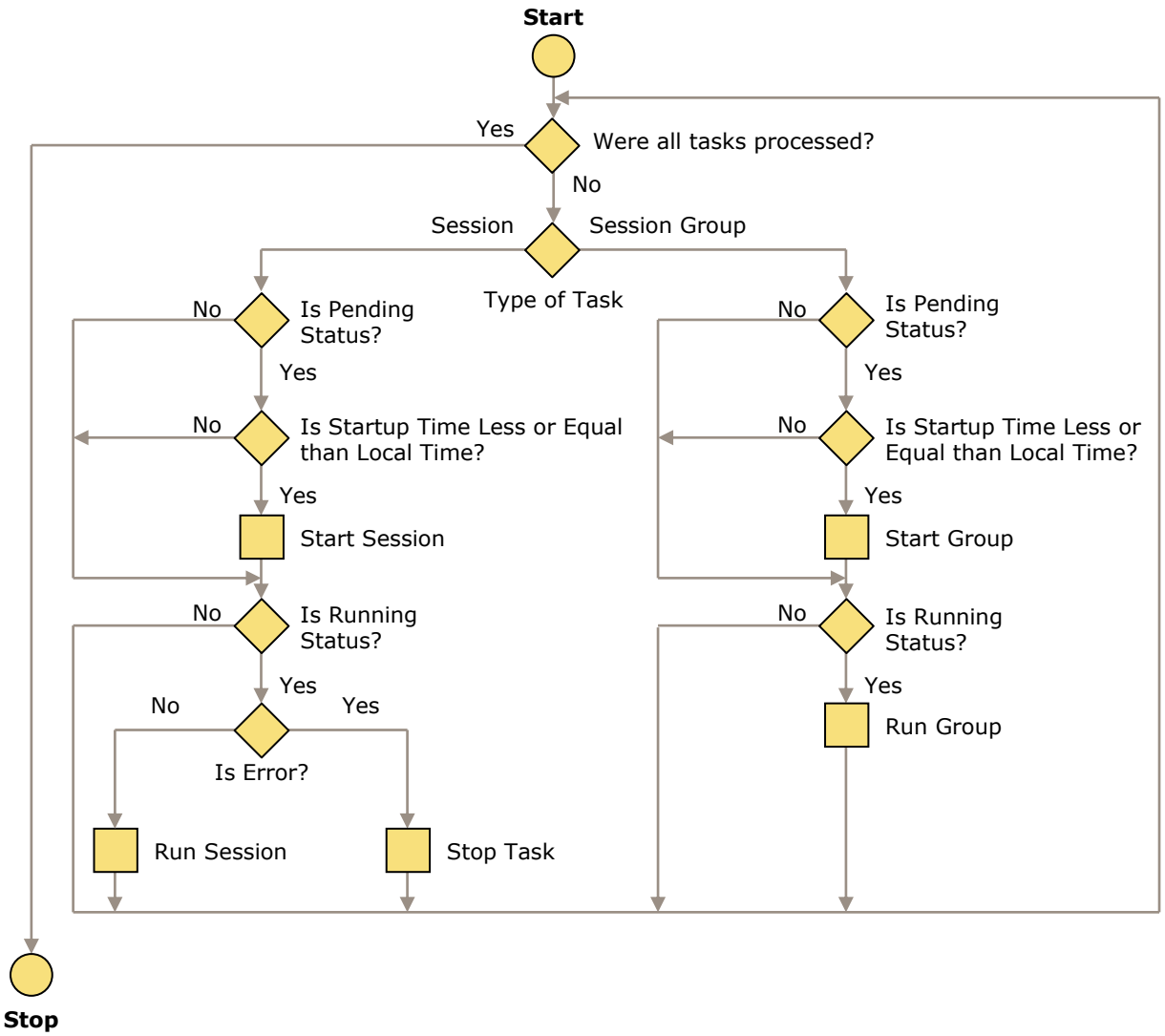
For tasks in the running condition, the Session Manager runs each task, by calling a running procedure. This procedure will be analyzed a little later (it receives the task identifier, collects information about the task, checks the last execution point and runs the next associated function).

At this point, the Session Manger makes a difference between single session tasks and session groups meaning the two different types of tasks receive different processing. To understand better the explanations, follow the algorithm diagram from figure 3.49.

If you look at the figure, the major difference, in addition of being completely different procedures for session and for session groups is that the running task procedure must check the error condition for session based tasks. If an error occurs for such a task, the task automatically receives the *finished* status and it will no longer be executed.

In the case of group based task, the group running procedure that at its turn dispatches the execution to the component session verifies for each session that an error occurred. In that case, the decision is made not only to cancel the session but also the entire group.

Figure 3.49 The run task procedure of the management service



3.6.8.1 STARTING THE SESSION BASED TASKS

The role of the start session functions is to set the scheduled task to *running* state and to begin the negotiations with the agents involved in performing the measurement. The function implementing this is:

```
void StartSingleSession(WORD wTaskId);
```

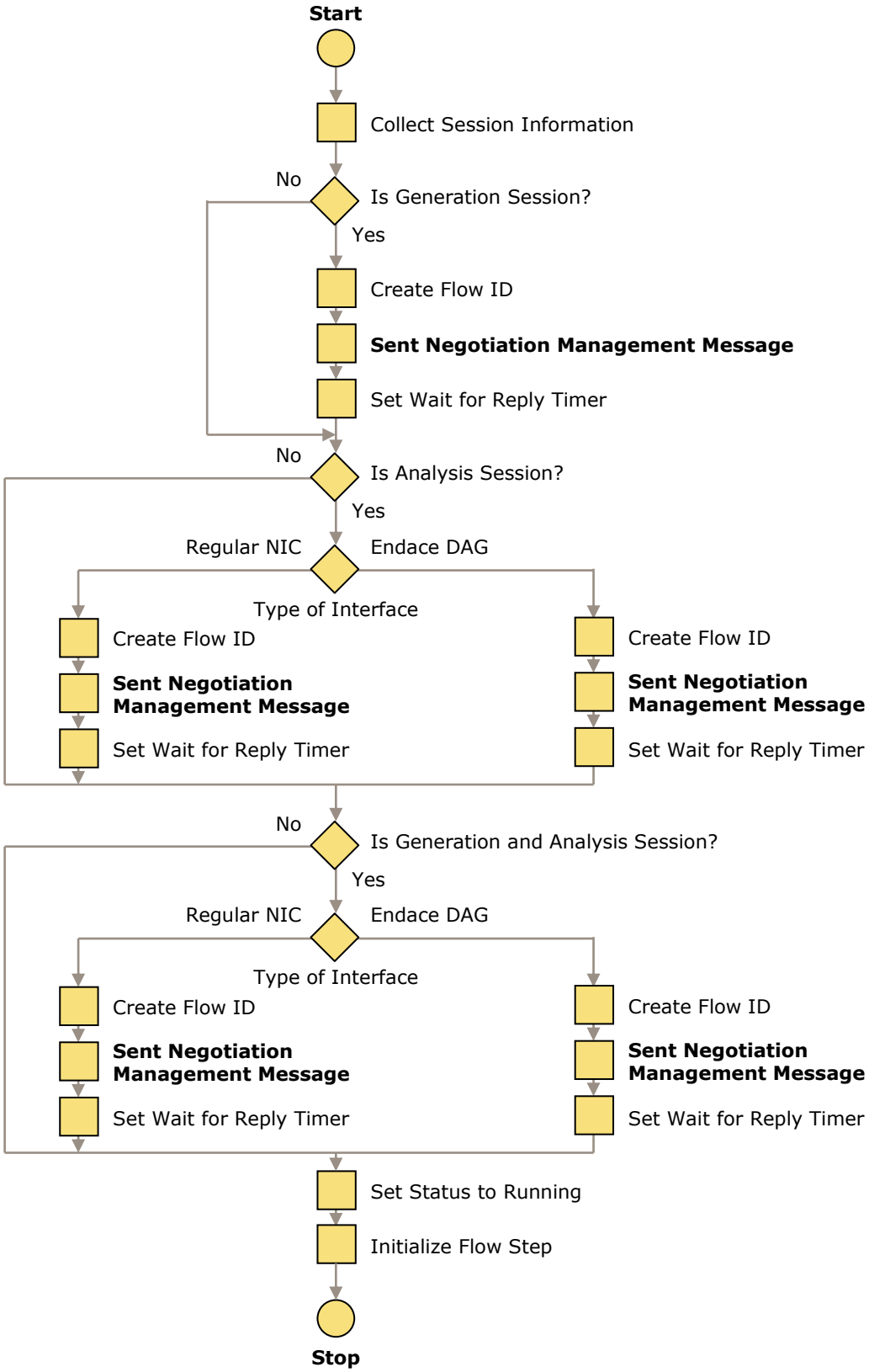
The function receives the task identifier, this being enough in order to have access to all task related data, including session information. The first thing is to collect the session parameters by requesting the session record (the session identifier is provided, since it is available with the task information). After the session information is collected, the session starting function can send the first messages to the involved agent or agents.

The golden rule here, which makes the distinction, is the session type. Depending on whether there is a generation, analysis or combined session, the function sends a negotiation message to one or both of them. If the session either is of analysis or combined, the distinction is also made

according the used hardware: a regular network interface card or the DAG card. This is necessary, since for the DAG other managed objects are used for session parameters compared to sessions that use ordinary NICs.

At the end of the start session function, if no error occurred, the status of the task is set to *running*. The next figure contains the graphical representation of the execution flow.

Figure 3.50 The start session algorithm



The most interesting aspect of running the sessions by the Session Manager is the way in which each task keeps its execution flow. The simplest analogy to understand it is the one of the state machine. Each task keeps an execution or flow step, which is initialized when the task is started. Later when the session running procedures comes, it knows according to the flow step which part of the session must be executed.

Each step of execution comes with some specific operations that must be performed. Depending on the results of those operations, the decision is taken to remain at the same step, i.e. the next time the procedure runs the same operation are undertaken, or to move on to another.

In the case of the start procedure, it initializes the flow step; usually the value *zero* is used to represent the task flow initialization. Because the status is set to running, at the next second the running task procedure takes its turn. Look at the figure 3.49 that illustrates how a task is executed. The running procedure knows that according to the flow step, that only negotiation of the session has been performed (meaning management messages have been sent to agents) and the corresponding action is to wait for the replies.

The management messages that are sent to the agents are specific for each type of operation. In the figure 3.50, different messages are used for different session types and even for different hardware. These functions are available in a library called SNMP wrapper, since it performs the top-level function in handling SNMP data: it receives session data, such as information about agents, measurement parameters and converts them into an appropriate SNMP message structure that will be sent to one of the queues of the queuing service. More details about the SNMP wrapper are found later in this document.

3.6.8.2 RUNNING THE SESSION BASED TASKS

The purpose of this topic is to explain the principles that stand at the basis of executing simple measurement tasks, and by extension, of the sessions and virtual sessions for the group based scheduled tasks.

Running a task means that for each task in the *running* state the Session Manager should execute the following function:

```
void RunSingleSession(WORD wTaskId);
```

As in the previous situation, the function receives the task identifier, which is enough in order to obtain all other information. Along with the task identifier comes the possibility of getting the task information, including the session identifier (to know the session data), and the task flow step.

What the running session function does is very simple:

- First, it gathers information about the session to see the session type and the hardware used by the agents. Remember that a session could be either of traffic generation, analysis or both involving one or two agents, while the hardware is important since different managed objects are used for the Endace DAG card than for regular Ethernet/IEEE 802.3 cards. The running session function relies on the same SNMP wrapper to generate SNMP information, and involves using different wrapping functions – that make the implementation easier. Different function for different session types and different hardware means to know which type and hardware is in use.

- Next, according to the previous parameters the execution jumps to the code section that involves only the current session. If, for example the current session is a session of traffic analysis using the DAG, the session running function, after determining these parameters, is focused only on executing the pieces of program dedicated to that scenario.

Notes

- The Session Manager uses a series of codes for describing each type of session. There are seven codes in total describing five possible session scenarios. Each scenario is implemented separately on both the start and running function. In figure 3.50, you can identify each scenario by looking at the bolded text, which means different negotiation management message for each of the five.
- Table 3.37 contains a description of each session type mapped to each session scenario.

Table 3.37 Mapping session types and hardware to session scenarios

Session Type	Code	Hardware Used	Session Scenario
Generating Layer 2 (Ethernet)	0	Regular NIC	Traffic Generation
Generating Layer 3 (IP)	1	Regular NIC	
Generating Layer 4 (UDP)	2	Regular NIC	
Analysis	3	Regular NIC	Traffic Analysis using Ethernet
		Endace DAG	Traffic Analysis using DAG
Generating and Analysis Layer 2	4	Regular NIC	Traffic Generation and Analysis using Ethernet
Generating and Analysis Layer 3	5		
Generating and Analysis Layer 4	6	Endace DAG	Traffic Generation and Analysis using DAG

After the determination of the session scenario, the program checks the flow step of the task. According to the numeric value of this parameter, the execution jumps to a particular sub-procedure that is executed. During this procedure, the decision is also taken if at the next moment the step remains the same, it is changed, or even the task is finished.

It is not the intention of this document to explain the execution flow for each session scenario. You must understand only the principle of execution, since the only difference between the scenarios is only that at the same step indexes other operations are performed, operations that imply:

- Creating management messages using the SNMP wrapper (the messages are automatically added to the outbound queue)
- Testing the inbound queue for replies
- Retrieving the replies from the inbound queue
- Using the SNMP wrapper to extract QoS data from the SNMP variables

- Using the Session Manager functions that handle result processing, store the QoS data and make it available for the user

Notes

- In accordance with the requirements of the queuing service the inbound testing of the queue and the extractions of the message from the queue; both operations must be performed at once, if the queue test yields true. Remember from the chapter describing the queuing service that queuing operations are locked between the testing of the inbound queue and reading the message. Check the topic on implementing the queuing service from the sub-chapter about the queuing service.
- The Session Manager uses flow-based filtering of the inbound queue in order to seek for agent replies. This is necessary, because when having multiple sessions running, need to ensure that each session gets only the messages that are addressed to it. The flow number is generated when the session is created (see figure 3.50) and it is unique per management console. The algorithm of generation the flow number is given below. In addition to the flow ID, the following variables are used to filter the inbound queue: the agent's IP address, the agent's port and the message type.

The flow identifier is a unique number that is used to refer to a measurement session. In the communication between the management console and the measurement agent, this number ensures that messages can be identified at the destination, when messages for multiple measurements have been received. At the agent software, the same flow identifier is also used when generating and analyzing traffic by stamping the packets with this value at a specific offset inside the packet. Therefore, if one agent generates the traffic, it puts the flow ID inside the packets (along with other information) and at the destination of the generated traffic, the analyzing agent can distinguish between all received packets.

The flow ID is a 16-bit number meaning that at one moment the management console may have up to 65536 running tasks. There are two situations in computing the flow ID:

- When there is only one agent involved in the session
- When there are two agents used

The following parameters are used in computing the flow ID:

- The IP address of traffic-generation agent (if it is the case)
- The IP address of traffic-analysis agent (if it is the case)
- An incremental number managed by the Session Manager

Figure 3.51 Computing the flow ID with only one agent

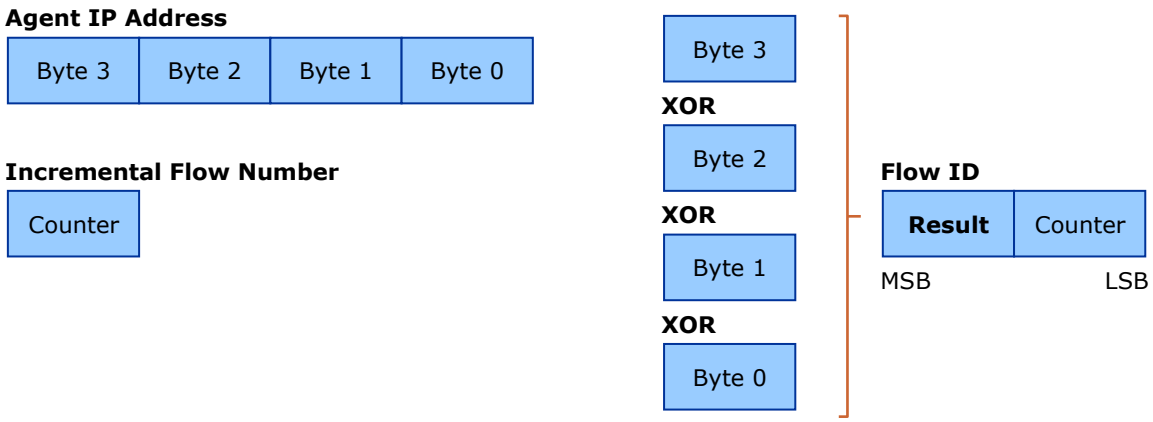
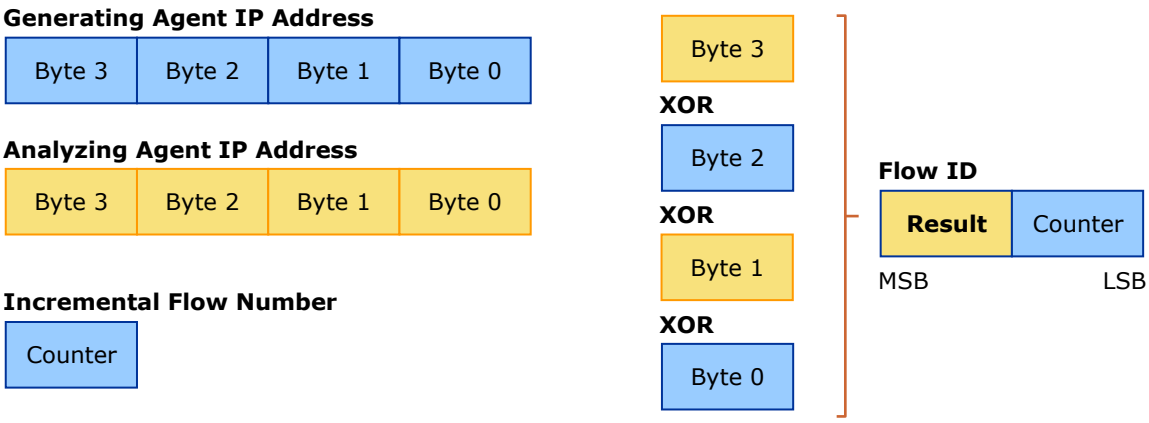
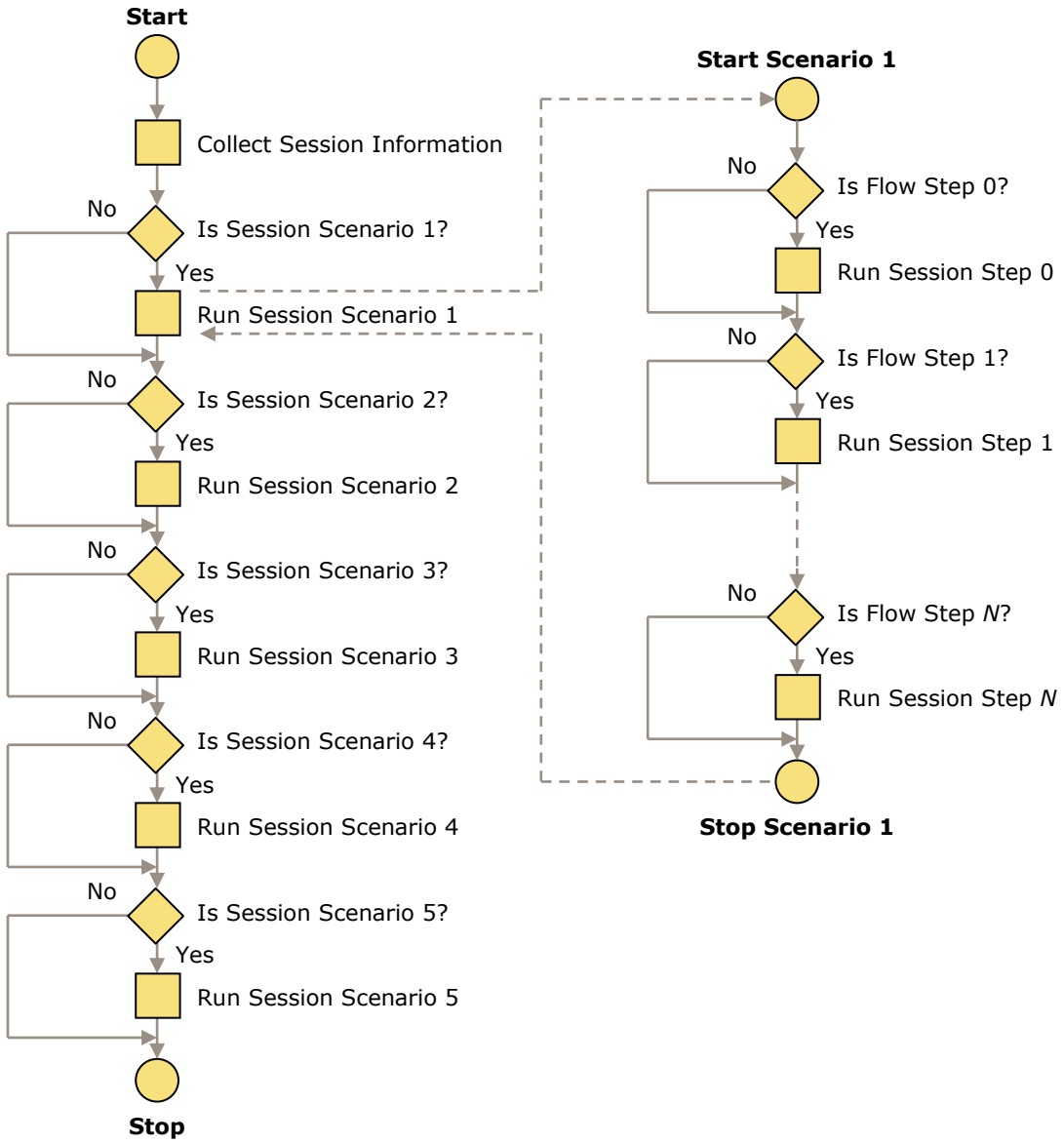


Figure 3.52 Computing the flow ID with two agents



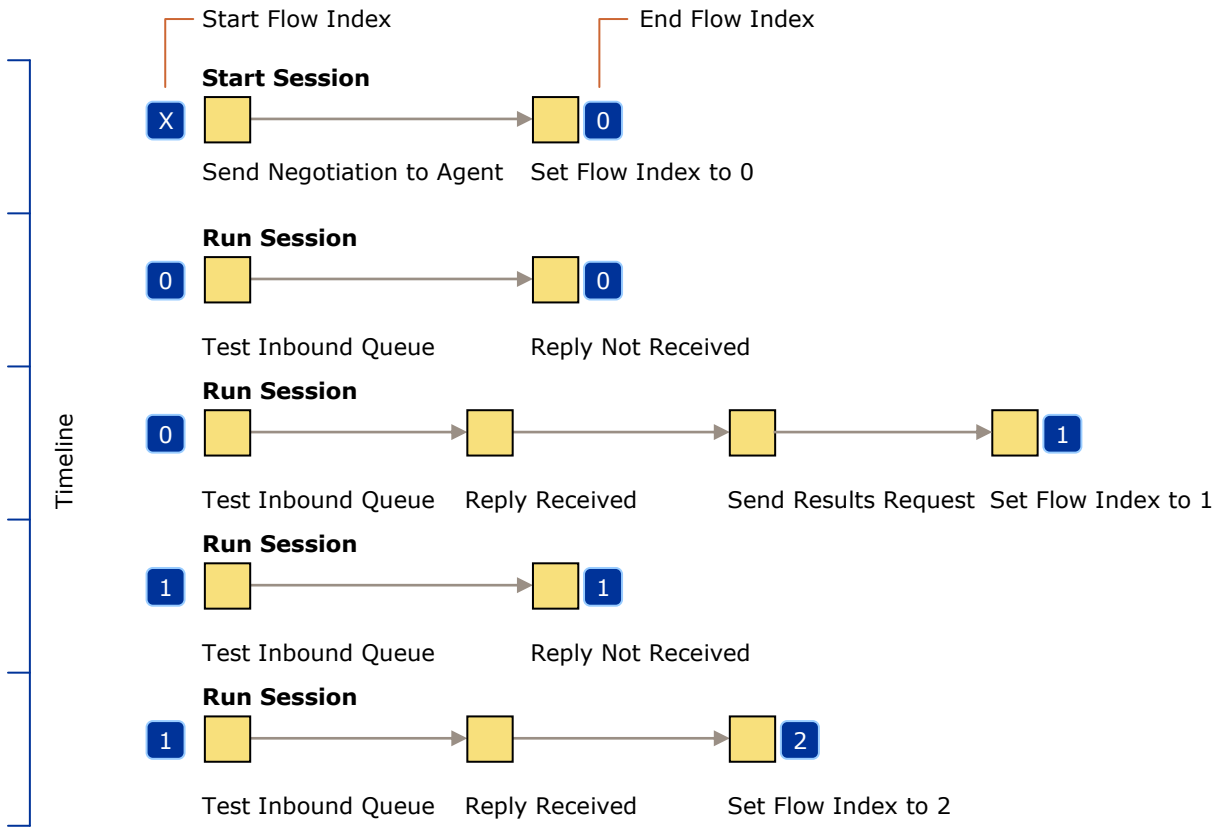
The next figure illustrates the algorithms inside the running session procedure, while the second figures exemplifies for an imaginary situation.

Figure 3.53 Execution algorithms inside the running session procedure



In figure 3.53, the execution algorithm is exemplified only for the first scenario. However, you may see that diagram presented is a general one and does not contain session specific blocks. In figure 3.54, there is an example for a traffic generating session. However, the diagram from this drawing does not depict the implementation algorithm but the execution flow. The flow is split up in several cycles, where each cycle is processed at every second.

Figure 3.54 Execution flow for a single session task – traffic generating session



The above figure is an easy example in order to understand what is happening:

- At start, the session starts procedure starts, by sending a negotiation message to the agent. The road of message has been explained, the procedure calls for a function from the SNMP wrapper library that determines the value for the SNMP fields and puts the message to the queue. The message is handled by the queuing service from now on; it will be transmitted by the management service and eventually it should reach the agent. When this step is completed, the flow step (or index) is initialized.
- At the first step in the running procedure, the algorithm will test to see either a reply was received. If no reply was received, the step will not be changed, and after a second, the same thing will be executed. A counter, which is not shown in the figure, will prevent the Session Manager of looping indefinitely with a given task, if a reply is not received. When a message is sent to the agent, the counter is reset and decremented at each execution of the running routine, i.e. every second. When the timer reaches zero, means the remote agent did not respond and the session is usually aborted.
- Assuming that a reply is finally received the results are being processed, in our situation it will be checked whether the agent responded in favor of session negotiation. If yes, means that on the agent, the traffic generation already started and the management console could request for results: a new management is sent to the

agent requesting real time data. The execution step is changed to one, because next time, the Session Manager expects for a reply to a data request rather than a negotiation request.

- At step one, the same verification on inbound messages is performed. If no messages are received this cycle, the flow step is not changed until either a reply is received or the timeout counter (that is decremented at each second) reaches zero.
- When a reply is received, the data will follow another processing path, because it is no longer a negotiation reply but rather a set of measurement results.
- The figure diagram stops at this point, but the next a new request message should be sent to the agent requesting newer results. After the new request is sent, the execution step should change back to one, since this is the step in which the Session Manager expects to receive results.

It is obvious now, the behavior of the task execution flow within the Session Manager is very similar to a state machine, where at each step, based on the received messages, and the decision is taken whether to stay at the same step, to move forward or to go back.

3.6.8.3 STARTING AND RUNNING SESSION GROUPS

Handling session groups is not very different in starting and executing simple session. The difference consists in an intermediary additional step required between the task scheduling and the execution of a specific session.

The following list contains similarities with the execution mode of the simple session tasks:

- When a session group either is started or executed, the task ID is passed as parameter. This allows the determination of the parameters of the task, including the group ID. If the group ID is known, the list of the sessions that make up the group is also available right away.
- For each executed session, the flow of execution and the algorithms used are identical with the single session case. Remember that the last two topics were referred to sessions alone. There was no connection with tasks; each function received only the task ID in order to determine easily the session parameters. For the session group start and running functions, the mechanism is the same; however, the session information is no longer passed as a flow ID from which the session is determined but is rather available as a pointer to a virtual session, where the virtual session is prepared in advanced.

Regarding the differences between running sessions and session groups:

- The first difference is related to the virtual sessions. When a simple session is started, the start procedure usually executes the first step of the state machine algorithm of the measurement session. The session data was available directly via the task ID and the corresponding session ID. However, in the case of groups is no longer the case, because for each group you can have a list of sessions or a single session with a list of parameters. To minimize the implementation differences as much as possible with respect of the single-session approach, the Session Manager creates a set of virtual sessions for each session group.
- The idea of virtual sessions is that each session is independently treated and executed. There are start and running procedures for virtual sessions similar to the ones for single sessions. Their content is astonishing alike. Furthermore, in the situation you

have parameter dependent sessions, each new session is a copy of the original in which the parameter has already been changed. In this way, the execution of each virtual session is done like for single session. The procedures do not know and care that the sessions they are executing are part of a group.

- In the start procedure of a group, the virtual sessions are created: a list of virtual sessions or a single virtual session, depending on the group type. In the latter case, the selected parameter is changed to the start value from the group.
- In the running procedure of the group the following actions are performed: verify if not all the sessions from the group are in the stopped state, in which case stop also the group. For type parameter dependent groups, there is only one virtual session and the group stop condition is the parameter reached its end value. Also, in the case of these groups, if the group is not stopped but the virtual session is, perform an update on the session parameter, by adding the delta value to it.

These are the functions used to run session groups and virtual sessions. You can find their implementation in appendix D.

```
void StartSessionGroup(WORD wTaskId);
void RunSessionGroup(WORD wTaskId);

void StartVirtualSession(LPTASKRECORD lpTaskRecord, WORD wTaskId);
void RunVirtualSession(LPTASKRECORD lpTaskRecord, WORD wTaskId);
```

3.6.9 THE SNMP WRAPPER

The SNMP wrapper is a collection of functions that perform the mapping between a call of a specific function and a management messages. Its role is to provide the Session Manager the functions required to send each management message. In this way to the Session Manager, sending a message is as simple as calling a function, while the SNMP wrapper creates the required SNMP data and places it into the message queue.

The work performed by the SNMP wrapper is of two types:

- Receiving through a function call the parameters of a management command, and the wrapper created the SNMP data o put it in the queue.
- The wrapper receives SNMP data as arguments, for data read directly from the queue by the Session Manager, and is required to return a specific value from that data.

The most difficult part in handling mapping between program variables and SNMP values is related to the creation and reading of managed objects. In addition to the community name, PDU type, request ID, error status and error index, SNMP features only the filed containing managed objects that could carry management information. Therefore, the parameters of every control message and any QoS result must have a managed object correspondent according to the SMI rules.

The Network Measurement System implements a set of managed objects specific for NMS measurements. For the current implementation the set of objects are descendants of the *experimental* class. The path from the MIB root to the experimental class is given in figure 3.3. The *experimental* class has bee divided into four classes, depicted in figure 3.55. Each class contains either objects or subclasses for different operations. In table 3.38, you may find each class along with a short description and the equivalent OID of the class.

Notes

- The class OID cannot be used within the management messages.

Figure 3.55 MIB classes implemented by NMS management

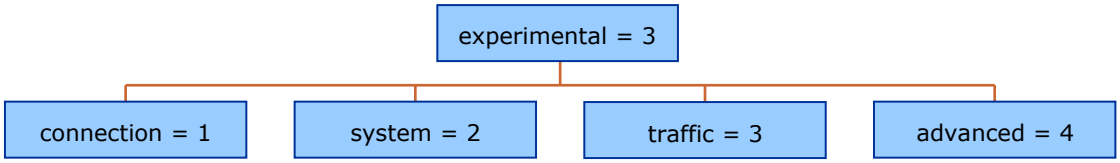


Table 3.38 Root MIB classes of NMS management

Class Name	OID	Description
Connection	1.3.6.1.3.1	This class contains objects used in connection negotiations between NMS entities, either managers or agents
System	1.3.6.1.3.2	This class contains objects used in for system parameters such as name, network interfaces
Traffic	1.3.6.1.3.3	This class contains objects used in traffic generating and analysis sessions
Advanced	1.3.6.1.3.4	This class contains objects used in advanced traffic analysis sessions

3.6.9.1 THE CONNECTION CLASS

The Connection class is used to implement the Identification Protocol. This protocol is not a networking protocol, but rather an agreement method between a NMS agent and the management console.

The SNMP relies on UDP and is connectionless, meaning no connection is established prior of management data transmission, through some negotiation messages. However, in the case of the Network Measurement System, both the management and the console require prior knowledge of each other before the exchange of management information related to measurements begins.

Table 3.39 Managed objects in *connection* class

Object Name	OID	Description
SNMP AGENT IDENTIFICATION	1.3.6.1.3.1.1	It is used to identify the agent software. When the agent receives a SNMP message having this object with value null, it replies with value 0x0F0F0F0F (using the same object). NMS software, which is not a measurement agent replies with another value.
SNMP MANAGER IDENTIFICATION	1.3.6.1.3.1.2	It is used to identify the manager software. When the agent receives a SNMP message having this object with value null, it replies with value 0x0F0F0F0F (using the same object). NMS software, which is not a management console replies with another value.

SNMP AGENT DISMISS	1.3.6.1.3.1.3	It is used to inform the agent that it is no longer used by the management console.

This is implementing using also SNMP but having as “payload” several special managed objects. These are the objects from the *connection* class. They are described in table 3.39.

Notes

- When the agent receives the *agent identification* object, a dual trust relationship is set up between the management console and the measurement agent. In this way, further management messages are automatically accepted without the need of additional connectivity negotiation. This process is called registration.
- The registration can be cancelled by sending a message with *agent dismiss* object with the value 0x0F0F0F0F.

3.6.9.2 THE SYSTEM CLASS

The *system* class is used to exchange descriptive information between the NMS entities, either managers or agents. This class contains objects for either software and hardware settings such as the NMS entity name, description, contact, networking data (number of network interfaces, vendor, name, IP configuration).

The table 3.40 contains the objects from this class and a brief description of each of them.

Table 3.40 Managed objects in the *system* class

Object Name	OID	Description
SNMP SYSTEM NAME	1.3.6.1.3.2.1	It returns the administrative name of the NMS entity.
SNMP SYSTEM DESCRIPTION	1.3.6.1.3.2.2	It returns the administrative description of the NMS entity.
SNMP SYSTEM LOCATION	1.3.6.1.3.2.3	It returns the administrative location of the NMS entity.
SNMP SYSTEM CONTACT	1.3.6.1.3.2.4	It returns the administrative contact of the NMS entity.
SNMP SYSTEM INTERFACES	1.3.6.1.3.2.5	It is class containing managed objects for network interface properties.

Notes

- The first four objects from this class refer to administrative properties of the NMS entity. These properties are specific to both agents and managers and are set by the network administrator or by the supervising user, hence the description of administrative.

The *system interfaces* class contains two descendants, given in table 3.41. One of the objects from this class, namely *interfaces table* is also a class containing objects for generic network interfaces. The objects from the *interafaces table* class are given in table 3.42.

Table 3.41 Managed objects in *interafaces* class

Object Name	OID	Description
SNMP SYSTEM IF COUNT	1.3.6.1.3.2.5.1	It contains the number of network interfaces available on the system.
SNMP SYSTEM IF TABLE	1.3.6.1.3.2.5.2	It is a class containing objects for each network interface in the system

Table 3.42 Managed objects in *if table* class

Object Name	OID	Description
SNMP SYSTEM IF INDEX	1.3.6.1.3.2.5.2.1	It is a number set by the remote peer, which indicates the interface data that is available in the other objects.
SNMP SYSTEM IF NAME	1.3.6.1.3.2.5.2.2	It contains the name of the selected network interface.
SNMP SYSTEM IF VENDOR	1.3.6.1.3.2.5.2.3	It contains the name of the manufacturer of the selected network interface.
SNMP SYSTEM IF SYSID	1.3.6.1.3.2.5.2.4	It contains the name of the interface from the operating system reference (e.g. *eth0*)
SNMP SYSTEM IF PHYSADDR	1.3.6.1.3.2.5.2.5	It contains an octet string value with the physical (or MAC) address of the selected network interface
SNMP SYSTEM IF MTU	1.3.6.1.3.2.5.2.6	It contains a numeric value, which is the maximum transmission unit of the selected network interface
SNMP SYSTEM IF CLASS	1.3.6.1.3.2.5.2.7	It contains the name of the type of network interface (either *Ethernet controller* or *network controller*)
SNMP SYSTEM IF IPADDR	1.3.6.1.3.2.5.2.8	It contains the IPv4 address assigned to the selected network interface
SNMP SYSTEM IF IPMASK	1.3.6.1.3.2.5.2.9	It contains the Ipv4 subnet mask assigned to the selected network interface
SNMP SYSTEM IF ASSGN	1.3.6.1.3.2.5.2.10	It contains a number representing the interface assigned roles: Interface used for management (0) Interface used for traffic generation (1) Interface used for traffic analysis (2)

Notes

- In order to access the information about a specific network interface when more than one is available on a given system one should first set the index of the chosen interface in the *if index* object and then perform a get operation on the other objects.
- The interface index value must be less that the interface count parameter (1.3.6.1.3.2.5.1)

3.6.9.3 THE TRAFFIC CLASS

The *traffic* class contains the bulk of objects used for QoS measurements. It has 35 managed objects for operation setup, control and QoS parameters. The table 3.43 listing contains these objects and a brief description of each of them.

Table 3.43 Managed objects in the *traffic* class

Object Name	OID	Description
SNMP TRAFFIC OPERATION	1.3.6.1.3.3.1	It is used to specify the measurement operation: Traffic generation (0) Traffic analysis (1)
SNMP TRAFFIC PROTOCOL	1.3.6.1.3.3.2	It is used to specify the protocol used for traffic generation and analysis in the case of double sessions: Ethernet II/IEEE 802.3 (0) IPv4 (1) UDP (2)
SNMP TRAFFIC TX PHYSADDR	1.3.6.1.3.3.3	It is used to specify the source MAC in case of Ethernet traffic generation
SNMP TRAFFIC RX PHYSADDR	1.3.6.1.3.3.4	It is used to specify the destination MAC in case of Ethernet traffic generation
SNMP TRAFFIC TX IP	1.3.6.1.3.3.5	It is used to specify the source IP address in case of IPv4 or UDP traffic generation
SNMP TRAFFIC RX IP	1.3.6.1.3.3.6	It is used to specify the destination IP address in case of IPv4 or UDP traffic generation
SNMP TRAFFIC TX UDP	1.3.6.1.3.3.7	It is used to specify the source UDP port in case of UDP traffic generation
SNMP TRAFFIC RX UDP	1.3.6.1.3.3.8	It is used to specify the destination UDP port in case of UDP traffic generation
SNMP TRAFFIC RATE	1.3.6.1.3.3.9	It is used to specify the traffic rate: Frame rate for Ethernet II/IEEE 802.3 Datagram rate for IPv4 Segment rate for UDP
SNMP TRAFFIC PACKET SIZE	1.3.6.1.3.3.10	It specifies the packet size: Frame size for Ethernet II/IEEE 802.3 Datagram size for IPv4 Segment size for UDP
SNMP TRAFFIC DISTRIBUTION	1.3.6.1.3.3.11	It specifies the generated traffic distribution: Periodic (0) Poisson distributed (1) Link flooding (2)

SNMP TRAFFIC POISSON STEP	1.3.6.1.3.3.12	It specifies the value used for approximation of traffic distribution when using Poisson: it can be one percent of the mean value, square root of the mean or other value (see note below)
SNMP TRAFFIC TEST DURATION	1.3.6.1.3.3.13	It specifies the test duration in seconds
SNMP TRAFFIC FLOW ID	1.3.6.1.3.3.14	It specifies the session flow ID
SNMP TRAFFIC RT CORRECTION	1.3.6.1.3.3.15	It specifies the method of correction applied in case of real time failure: Increase test length (0) Increase packet density (1)
SNMP TRAFFIC PROT FIELD	1.3.6.1.3.3.16	It is the value of the protocol field (applies only for Ethernet and IP traffic)
SNMP TRAFFIC IP TTL	1.3.6.1.3.3.17	It is the value of the TTL field (applies only for IP traffic)
SNMP TRAFFIC IP TOS	1.3.6.1.3.3.18	It is the value of the TOS field (applies only for IP traffic)
SNMP TRAFFIC UPDATE	1.3.6.1.3.3.19	It is the time interval (in seconds) at which the measurement agent performs an update of online QoS parameters
SNMP TRAFFIC DETAILED	1.3.6.1.3.3.20	It specifies whether the online method of analysis is simple (0) or advanced (1)
SNMP TRAFFIC DUMP	1.3.6.1.3.3.21	It specifies whether a dump is performed by the agent for offline analysis
SNMP TRAFFIC START TIME	1.3.6.1.3.3.22	It contains an 8-byte value which is the Epoch representation of the session start time
SNMP TRAFFIC END TIME	1.3.6.1.3.3.23	It contains an 8-byte value which is the Epoch representation of the last computed data
SNMP TRAFFIC PACKET COUNT	1.3.6.1.3.3.24	It specifies the number of packets transmitted or received
SNMP TRAFFIC THROUGHPUT	1.3.6.1.3.3.25	It specifies the analyzed through
SNMP TRAFFIC AVG OWD	1.3.6.1.3.3.26	It specifies the analyzed average one way delay in the last observation
SNMP TRAFFIC MIN OWD	1.3.6.1.3.3.27	It specifies the analyzed minimum one way delay over the all observation period
SNMP TRAFFIC MAX OWD	1.3.6.1.3.3.28	It specifies the analyzed maximum one way delay over the all observation period
SNMP TRAFFIC AVG JITTER	1.3.6.1.3.3.29	It specifies the average packet delay variation over the last observation period
SNMP TRAFFIC MIN JITTER	1.3.6.1.3.3.30	It specifies the minimum packet delay variation over the all observation period

SNMP TRAFFIC MAX JITTER	1.3.6.1.3.3.31	It specifies the maximum packet delay variation over the all observation period
SNMP TRAFFIC OOO PACKETS	1.3.6.1.3.3.32	It specifies the number of out-of-order packets received
SNMP TRAFFIC RT FAILURE	1.3.6.1.3.3.33	It specifies for how many packets real time failure occurred at the transmission agent
SNMP TRAFFIC BUFFER OVERFLOW	1.3.6.1.3.3.34	It specifies whether buffer overflow occurred during the saving of the data dump, if the dump is performed
SNMP TRAFFIC DUMP READY	1.3.6.1.3.3.35	It specifies whether the dump analysis is completed (1) at the end of a session

Notes

- The Poisson step refers to a unit of approximation when computing the distance between two packets, such as the distribution is followed with higher accuracy. The NMS management console allows selecting this parameter either one percent of the average time distance between packets or square root. One option copes with packets at higher rates, while others work better at lower rates. The lowest possible value of the parameter is one millisecond. If this parameter is zero, the distribution is periodic. For more information, please consult the agent documentation in [2].
- Real-time failure occurs whenever the software no longer can send packets at the specified moments in time due to either limited processing power of insufficient available capacity on the link. For each packet a real time failure occurs, the agent that generates the traffic increases a counter. A real time correction policy could, in case of a real time failure, to align all new packets to the current time (value 0 for *rt correction* parameter), or to send as much packets as possible in order to reach the old time references (value 1).

3.6.9.4 THE ADVANCED ANALYSIS CLASS

This class contains managed objects to be used when using the following analysis methods:

- Full link analysis using the PCAP system library
- Traffic analysis using the Endace DAG network controller

It contains two classes, one for each type of analysis. The objects from each of these classes are given in tables 3.46 and 3.47.

Table 3.44 Classes in the *advanced* class

Class Name	OID	Description
SNMP TRAFFIC PCAP	1.3.6.1.3.4.1	It contains objects to be used with PCAP-based analysis
SNMP TRAFFIC DAG	1.3.6.1.3.4.2	It contains objects to be used with DAG card based analysis

Table 3.45 Managed objects in the *PCAP* class

Object Name	OID	Description
SNMP TRAFFIC PCAP IF	1.3.6.1.3.4.1.1	It is used to specify the interface with which the PCAP analysis is performed (the values is the MAC address of the interface)
SNMP TRAFFIC PCAP THROUGHPUT	1.3.6.1.3.4.1.2	It contains the throughput obtained on the interface on which the PCAP analysis is performed
SNMP TRAFFIC PCAP TIME	1.3.6.1.3.4.1.3	It contains an 8-byte timestamp with the time in Epoch format at which the last information update was made

Table 3.46 Managed objects in the *DAG* class

Object Name	OID	Description
SNMP TRAFFIC DAG NAME	1.3.6.1.3.4.2.1	It specifies the device name of the DAG card to be used with the measurement
SNMP TRAFFIC DAG TEST DURATION	1.3.6.1.3.4.2.2	It specifies in seconds the length of the measurement session
SNMP TRAFFIC DAG ADVANCED	1.3.6.1.3.4.2.3	This managed object is obsolete. Its value should always be one.
SNMP TRAFFIC DAG PROTOCOL	1.3.6.1.3.4.2.4	It specifies the type of traffic analyzed. It can be: Ethernet II/IEEE 802.3 (0) IPv4 (1) UDP (2)
SNMP TRAFFIC DAG SOURCE ADDR	1.3.6.1.3.4.2.5	It specifies the source address for incoming packets in an octet string format. Options: 6-byte hardware address for Ethernet 4-byte IP address for Ipv4 and UDP
SNMP TRAFFIC DAG TIME	1.3.6.1.3.4.2.6	It contains the timestamp in Epoch format of the last parameters update
SNMP TRAFFIC DAG DUMP	1.3.6.1.3.4.2.7	It specifies whether a dump is performed for offline analysis
SNMP TRAFFIC DAG THROUGHPUT	1.3.6.1.3.4.2.8	It contains the last value of the analyzed inbound throughput of the specified flow
SNMP TRAFFIC DAG AVG OWD	1.3.6.1.3.4.2.9	It represents the average one way delay during the last observation period
SNMP TRAFFIC DAG MIN OWD	1.3.6.1.3.4.2.10	It represents the minimum one way delay during the all observation period
SNMP TRAFFIC DAG MAX OWD	1.3.6.1.3.4.2.11	It represents the maximum one way delay during the all observation period
SNMP TRAFFIC DAG AVG JITTER	1.3.6.1.3.4.2.12	It represents the average packet delay variation during the last observation period

SNMP TRAFFIC DAG MIN JITTER	1.3.6.1.3.4.2.13	It represents the minimum packet delay variation during the all observation period
SNMP TRAFFIC DAG MAX JITTER	1.3.6.1.3.4.2.14	It represents the maximum packet delay variation during the all observation period
SNMP TRAFFIC DAG OOO PAKCETS	1.3.6.1.3.4.2.15	It represents the number of out-of-order packets received during the analysis period
SNMP TRAFFIC DAG PACKET COUNT	1.3.6.1.3.4.2.16	It represents the number of packets received during the analysis period
SNMP TRAFFIC DAG DUMP READY	1.3.6.1.3.4.2.17	It specifies whether the dump performed at the end of the analysis session (if any), is completed and the QoS parameters are available

The information contained in these tables is a brief description of the 74 managed objects that are used by the Network Measurement System management infrastructure. The information contained here is, of course, not enough to cover all aspects of using these set of objects correctly with either the manager or the agent software.

However, for additional information of using the NMS MIB, you may want to consult the documentation of the agent, since the agent software is the one creating the data for most of them. You may also consult in the appendix E the source code of the SNMP wrapper that gives valuable information on how toe use this set of objects when communicating with the agents.

3.6.10 SESSION RESULTS

During the execution of each session when results are received, they are automatically saved and are available to the user. In order to prevent memory issues, the management console saves the incoming results to disk. In this way, the program is scalable to a large number of tasks and long session, without the worry that it will be out of system physical memory.

Each new set of results is referenced by an entry into a result list. The mapping is usually made according to flow ID (in the case of session group global results the mapping is made according to the task ID, rather than the flow ID).

The following functions implement results management:

```
void CreateDatasetSingleSession(WORD wTaskId);
void SaveDatasetSingleSession(
            WORD wTaskId,
            WORD wFlowId,

            BYTE bType,
            BOOL bDump,

            struct timeval tvTxStartTime,
            struct timeval tvTxEndTime,
            DWORD dwTxPacketCount,
            DWORD dwTxRtFailure,

            struct timeval tvRxStartTime,
```

```
            struct timeval tvRxEndTime,
            DWORD dwRxPacketCount,
            DWORD dwRxThroughput,
            unsigned long long lwRxAvgOwd,
            unsigned long long lwRxMinOwd,
            unsigned long long lwRxMaxOwd,
            long long liRxAvgOwd,
            long long liRxMinOwd,
            long long liRxMaxOwd,
            DWORD dwRxOoo,
            DWORD dwRxOverflow
            );

void CreateDatasetSessionGroup(WORD wTaskId);
void AddDatasetToSessionGroup(WORD wTaskId, WORD wFlowId,
                              WORD wSessionId);
void CreateDatasetSession(WORD wTaskId, WORD wFlowId);
```

Before some explanations about the previous functions, there must be specified the method of saving the results. Two folders are available by default for keeping the session results. See table 3.47.

Table 3.47 Folders for session results

Folder Name	Description
./session	This folder keeps the session group related data files
./session/flows	This folder keeps the flow data files, containing the QoS parameters

The files that store the results are either files that keep session group information or files that keep session flow information. For example, if you run a single session, the resulting data will be stored in a flow file in the second directory. The file name contains the flow ID, and since the flow ID has one-to-one correspondence with the task ID, is easy to collect the results.

However, in the case of group the mapping flow ID – task ID is no longer saved. Having only the task information you can determine at most the group ID, the sessions ID from within the group but not the actual flow IDs that were used for each session in its turn. Because of this, in the case of session, the flow ID correspondence must be saved to a group results file saved in the first folder.

We have:

- The first function is used to create a flow ID file for use with a single session task.
- The second used to save flow real time information in the flow file, regardless it is for a single session task or for a session group task
- The last three functions are used exclusively for group: they allow to create an empty group file when a group is started; to add a new flow to the group as more sessions from the group start executing and finally to create new flow files in the *flows* folder – the files will be populated with data by using the same function like for single session case

In addition, the Session Manager contains functions for the following:

- Accessing the results in both the saved groups and saved flow files
- Check the availability of results during a session or a session group
- To export the results in a *comma separated values* format

```
WORD GetTaskResultCount(void);
WORD GetTaskResultTaskId(WORD wIdx);
WORD GetTaskResultFlowId(WORD wIdx);
BOOL GetTaskResultAvailable(WORD wTaskId, WORD wFlowId);
DWORD GetTaskResultDatasetCount(WORD wFlowId);

int SaveListViewAsSpreadsheet(LPVOID ptrListView, LPCTSTR szFileName,
                              BYTE bColumns);
```

3.7 OTHER SERVICES

The role of this subchapter is to present briefly some additional technologies of the NMS management console, in order to be better acquainted with the terminology, the options and possible settings.

3.7.1 THE LOCAL MESSAGE DISPATCHER

The messages from the queue are regularly read by the Session Manager, if the message is related to a measurement session. However, the received SNMP messages could also be management messages unrelated to any measurement process. In this situation, it is the job of the message dispatcher thread to handle the processing of each message. Such messages could be requests of information such as whether the software is an agent or a manager, the name and so on.

This type of descriptive information that ensures distribution of identification data between NMS managers and agents is called the Identification Protocol and it was already presented as a feature of the management infrastructure. If you need more information, review the section that presents the extended features of the management infrastructure, and especially the table 3.2.

The implementation of the message dispatcher is based on a single function running under the context of the management service.

```
void DispatchMessage(void);
```

The purpose of the message dispatcher is to reply with the values of some local parameters when interrogated by another SNMP capable entity, either a manager or an agent. The objects implemented by the dispatcher of the management console are listed in table 3.48.

Table 3.48 Managed objects handled by the message dispatcher

Object Name	OID	Description
Agent Identification	1.3.6.1.3.1.1	The management console always replies with the value 0xF0F0F0F0, hence different from 0x0F0F0F0F. The management console does not identify as a measurement agent.
Manager Identification	1.3.6.1.3.1.2	The management console always replies with the value 0x0F0F0F0F, hence it identifies itself as a management console.
System Name	1.3.6.1.3.2.1	The management console replies with the administratively set name.
System Description	1.3.6.1.3.2.2	The management console replies with the administratively set description.
System Location	1.3.6.1.3.2.3	The management console replies with the administratively set location.
System Contact	1.3.6.1.3.2.4	The management console replies with the administratively set contact information.

3.7.2 THE SERVICE CONTROL MANAGER

The role of the Service Control Manager is to allow the user to control the status of the management (and dependent) services, while ensuring at the same time recoverability in the case of failures.

The user can control the status of the management service, and due to dependency reasons of the following services:

- The queuing service
- The SNMP service
- The Session Manager
- The local message dispatcher

The control is achieved by allowing the external change of the management service's control variables that could place it in either of the following states: stopped, started and paused. Intermediary states such as starting, stopping, pausing and resuming exist but they are not interesting from the user's perspective.

The control of the management services is useful since allows changes the configuration to be performed without requiring an application restart. It also brings reliability since the SCM supervises the execution of the management service. This is achieved using a keep-alive status variable that is periodically checked to see whether the management service terminated its execution (most probably due to a hang). In such a situation, the SCM can do nothing or take one of the following corrective actions:

- Restart the service
- Restart the management console application

The corrective actions in case of failures can be setup via the Services view.

The implementation of the SCM uses a single procedure within its own thread. The Service Control Manager starts executing before any other service at the application startup and terminates the last.

3.7.3 THE CONFIGURATION SERVICE

The configuration service features a set of functions that control the configuration of all other services and of the user interface. Five global parameters specify the files in which specific service configuration can be kept. Table 3.4 already presented the configuration descriptions; table 3.49 contains the default file names for each configuration.

Table 3.49 Default names for the configuration files

File Purpose	Default File
Toolbar Configuration	./config/toolbar.cfg
Manager Configuration	./config/mgr.cfg

Queuing Configuration	./config/queuesrv.cfg
Management Configuration	./config/mgmtsrv.cfg
SNMP Configuration	./config/snmpsrv.cfg
Session Manager Configuration	./config/sessmgr.cfg

Because the configuration parameters of the configuration service are quite critical (without the correct file names, the proper configuration cannot be loaded), the filenames from table 3.49 can also be edited offline. They are saved in a text file called *global.cfg* and it is the single configuration file that is in text format (all others are binary).

The default contents of the *global.cfg* file:

```
[CONFIGURATION FILES]
ToolbarConfiguration=./config/toolbar.cfg
ManagerConfiguration=./config/mgr.cfg
QueuingServiceConfiguration=./config/queuesrv.cfg
ManagementServiceConfiguration=./config/mgmtsrv.cfg
SnmpServiceConfiguration=./config/snmpsrv.cfg
SessionManagerConfiguration=./config/sessmgr.cfg
```

3.7.4 THE HARDWARE MANAGER

The Hardware Manager collects from the system information about the local devices and makes them available to the management console. A copy of the Hardware Manager is also implemented at the measurement agent.

The device data that is available from the hardware manager is:

- Data about the PCI installed devices in the system
- Data about the networking configuration

Notes

- The implementation of the Hardware Manager was based on the shared code provided by *lspci* and *ifconfig* GNU applications. For more details about their implementation, you may consult the source code of the files from table 3.50.

Table 3.50 Source code files of the Hardware Manager

File Name	Size (bytes)	Version	Comments
pciapi.cpp	52,444	1.0.0.510	C++ source code file
pcilst.cpp	1,235,312	1.0.0.510	C++ source code file
pciapi.h	1,563	1.0.0.510	C++ header file
pciapt_config.h	347	1.0.0.510	C++ header file

pciapt_header.h	47,419	1.0.0.510	C++ header file
pciapt_sysdep.h	1,658	1.0.0.510	C++ header file
netapi.cpp	13,234	1.0.0.510	C++ source code file
netapi.h	981	1.0.0.510	C++ header file

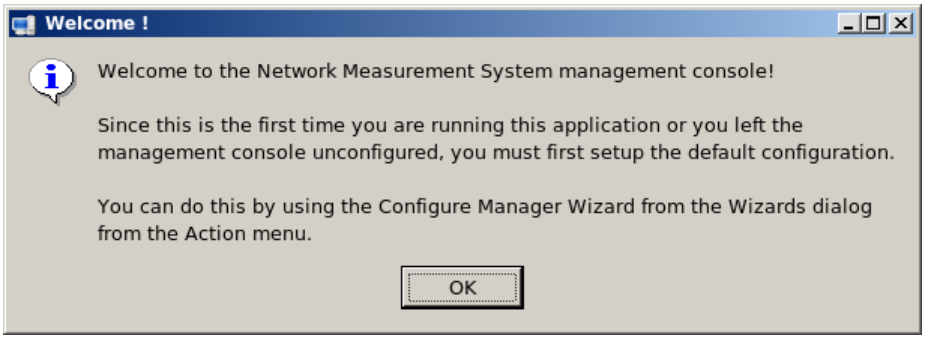
3.8 USING THE MANAGEMENT CONSOLE

This sub-chapter is intended to be a brief user guide into using the Network Measurement System management console. So far, the project focused on the implementation alone, with the most important aspects discussed in detail. Now, it is time to turn our attention on part that is the most from the user point of view: the user interface.

3.8.1 CONSOLE SETUP

When you first start the management console, may be when only the default files are present and the console is not configured. If the manager configuration files are missing, you are notified by a welcome message inviting you to configure the management console.

Figure 3.56 The welcome dialog



Important

- There may be situations in which the configuration file is not missing but the manager configuration became corrupted. In other cases, some parameters may be incorrect. Such situations will also prevent the management console of starting correctly or even loading the configuration. Such errors are usually notified to the user with specific error dialog boxes, in which detailed error information shall be available.

While the management console is not configured, the options of using it are very limited. Actually the most important component, the management service, is not running and at this time, the manager properties dialog requires a configuration – see figure 3.57 for details. To help the user in performing the configuration with as little knowledge as possible, various elements starting from dialog boxes, the main console view and the properties will indicate which are the steps required to follow. You may even have configuration buttons just one click away, similar to the configure manager button from the manager's properties dialog.

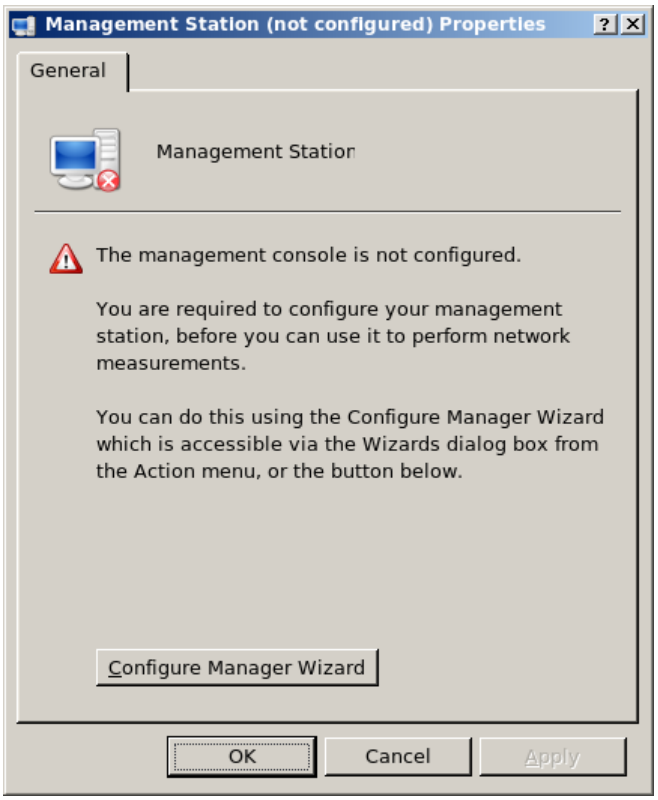
The management console features a series of wizards to guide you through complex procedures. The configuration process is one of them, and for this reason, you will have to use the **Configure Manager Wizard** to have the console configured.

You may start the Configure Manager Wizard in the following ways:

- Click on the **Management Station** item in tree view and use the **Configure Management Station** button.

- An alternative method is to open the properties dialog of the management console, by selecting the **Management Station** in the left tree view and then by choosing **View** > **Properties**. In the properties dialog, you must click on **Configure Manager Wizard**.
- The last method of opening the wizard is by opening the **Open Wizard** dialog from **Action** menu > **Wizards**. In the dialog select **Configure Manager Wizard** in the list and click **Start**.

Figure 3.57 Not configured manager's properties dialog



The wizard contains a set of steps what will guide the user in introducing a set of information, some mandatory, and other optional. In the order, the configuration data that one needs to input is:

- The management console name (mandatory), description (optional), location (optional) and contact (optional). This is the so-called administrative description information of a NMS entity. It must be configured on the management console and is recommended to configure it on the agents as well. You will see later, that this information is used to identify at the console between different NMS applications (especially agents) installed at different measurement points.
- You have to select a primary network interface to use it with the management service. The management service supports multiple network interfaces, but these could be configured only after the manager is configured. At this time, it allows you to select only one. When you choose the network interface, you may also inspect its properties, provided by the Hardware Manager to be sure of the selection. Figure 3.58 shows the type of data available in the wizard's window; while the figure 3.59 shows the standard network interface properties dialog (it contains hardware information, such as device identifier, name, manufacturer, MAC and other hardware related properties and software configuration parameters like the IP configuration).
- At the next step, one should set up the SNMP community name along with their permissions: none, notify, read-only, read-write and read-create.

- The last settings are the one related to security. The management supports two types of management packet security: based on SNMP communities and based on source IP filtering. The IP filtering information can be introduced in two ways: specifying either a list of addresses that will be filtered (the default) or to create a list and only management messages coming from those sources will be allowed.
- In the end, you may review your selection and entered data to verify if the management console got it correctly and finishing will configure the management console and save the configuration.

Figure 3.58 Selecting a network interface in the Configure Manager Wizard

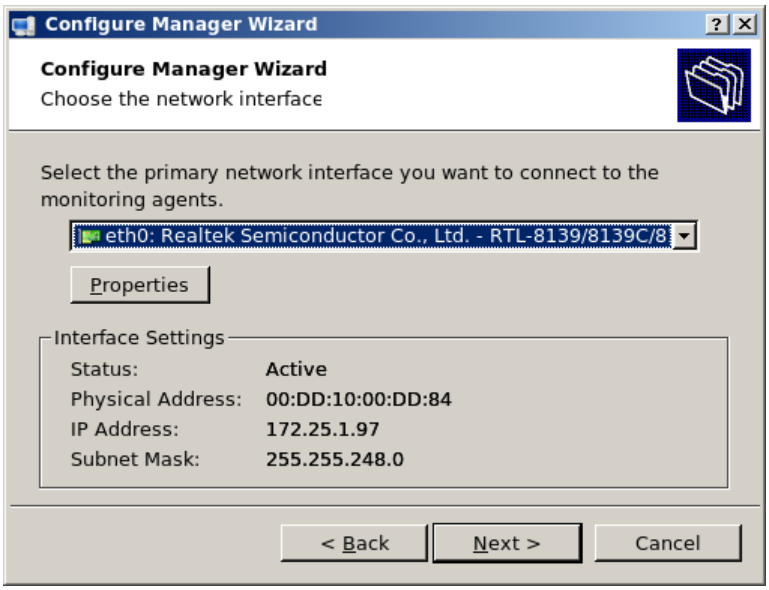
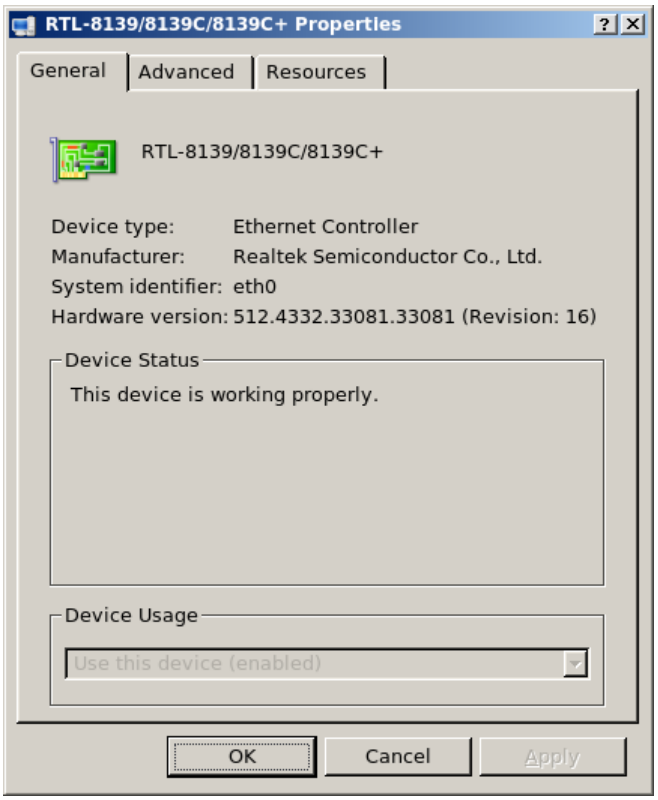


Figure 3.59 Network interface properties dialog



Important

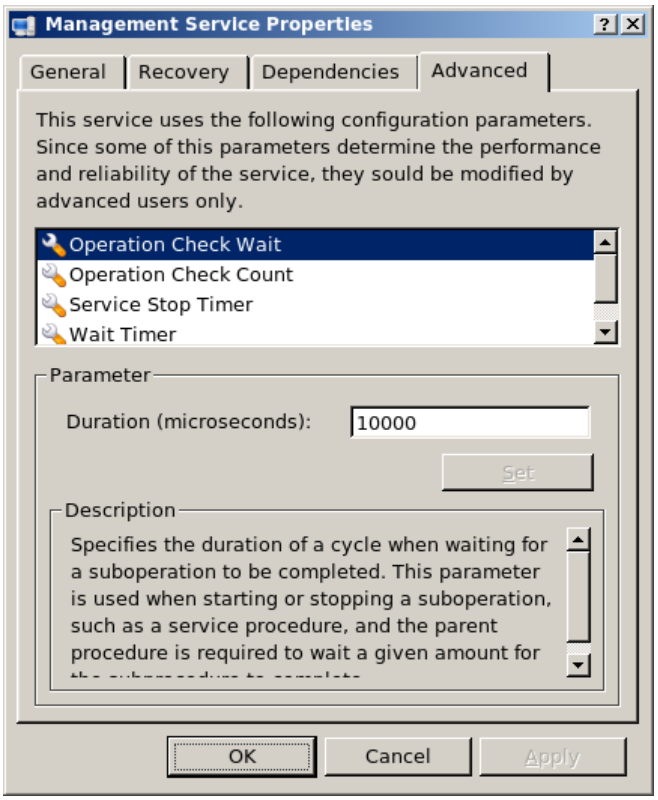
- After the management console has been configured for the first time, the management service is not started by default. To start the management service you have the following option: to open the manager's properties dialog (selecting the manager from the tree list, and then click on **View** > **Properties**) or to open the properties dialog of the management service by selecting it the **Services** view and following the same procedure. In both dialogs, you will find a control area from which you can control the management service.
- You must start the management service in order to pass to the next steps that involve communicating with the agents. Remember that as far as the management service is stopped the following service are also stopped: the queuing service, the Session Manager and the SNMP service.

3.8.2 CONFIGURING SERVICES

Before using the management console, you may also want to refine the configuration information for various services. If you have read the documentation so far, you noticed that there many configuration parameters could be set up, such as operation timeouts, queue size, maximum OID length etc. The role and usage of those parameters has been previously explained.

Now, in order to access the configuration of a given service you must select the service, open its properties dialog and go to the **Advanced** tab.

Figure 3.60 The advanced tab of the management service



You may see that various parameters come with a small description, however it is recommended to understand them well prior to make and save any changes.

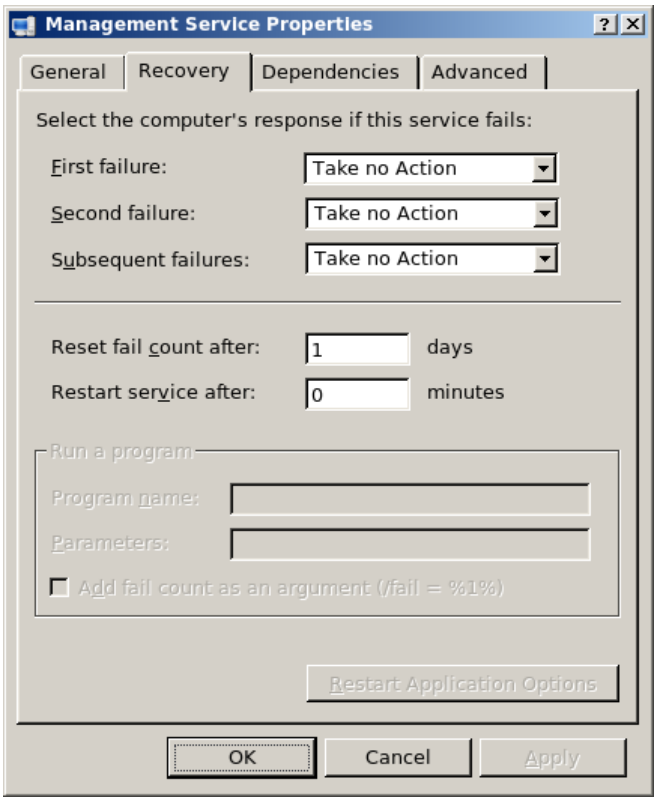
Caution

- The changes to the services parameters take effect when you click the **OK** or **Apply** button. In order to prevent any problems it is recommended that changes on the advanced configuration setting is made by advanced users only.

Not only the management service has configuration parameters, but also other services that were previously presented, such as the queuing service, SNMP service, configuration service.

The services properties dialog may also allow you to change the recovery settings of a service. Remember the role of the Service Control Manager was to provide reliability in the case a service failed. However, with the default setting no such feature is provided and must be set up manually. Figure 3.61 suggests how this can be achieved using the service properties dialog.

Figure 3.61 The recovery tab of the management service



You may select a different recovery policy for the first, second or subsequent failures. Other options such as the *reset fail count* specify after how many days the failures counter (that indicates whether the failure was the first, second and so on) is reset. If you select the *restart the service* recovery policy the *restart service after* parameter specifies the time interval in which the service will be restarted.

Finally, if you choose the *run a program* recovery policy you need to specify a program name and optional some parameters (the failure count can be added as a parameter as well).

3.8.3 REGISTERING AGENTS

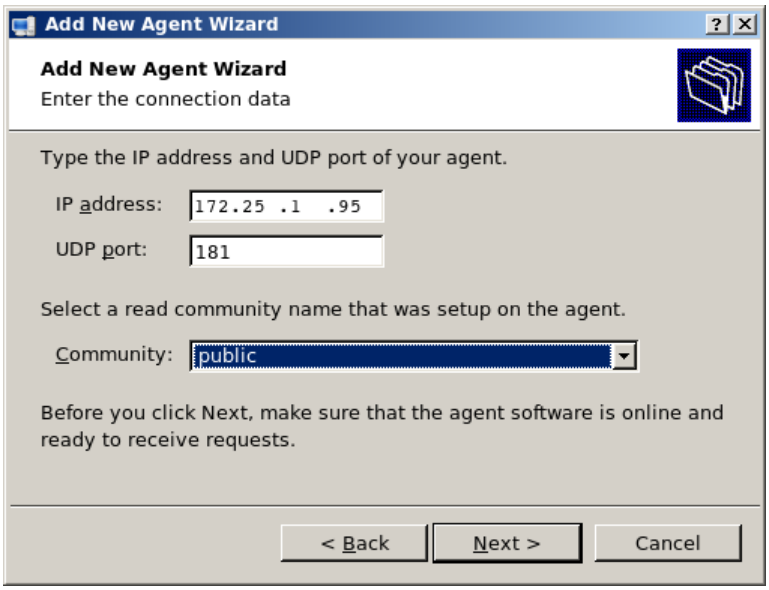
If the manager has been configured successfully, and the configuration of the console's services has been tuned, you might start registering agents as the first step in performing network

measurements. You can do this by clicking on the **Add Agent** button in the **Agents** view. This will start the **Add New Agent Wizard** that will guide you through the registration process.

To add a new agent the following information is required:

- The IP address of the agent
- The UDP port which is used by the management service on the agent
- The read-only and read-write communities configured (at least one read-write community is required)

Figure 3.62 Agent registration process



If the agent exists, it should reply with the identification information (name, description, location and contact). You may review this information and if it is correct, you may finish the registration process. If the agent was successfully added, it will appear in the agents list.

There are two most often situations that will not allow a registration process to complete:

- The agent is not responding, in which case the wizard will inform the user with a specific message
- The remote SNMP entity is not a NMS agent, meaning that it does not reply to the identification request correctly.

When an agent is no longer in use, you may choose to remove its registration on order to delete it from the agents list. The agent registration removal is also performed in agreement with the agent software; hence, you will need to have the agent up and running in order to succeed.

The first window in the registration removal wizard will attempt to connect to the agent and if, successful it will display the agent identification information. If the information checks out you may confirm the registration removal process and the agent is also automatically removed from the list.

When an agent is registered, you may collect additional information from it such as:

- The number of the interface cards installed on the system where the agent is running
- The name, vendor and type of those interfaces

- For regular Ethernet cards additional information such as the hardware physical address, IP configuration, the maximum transmission unit and the roles for which that interface is used is also available.

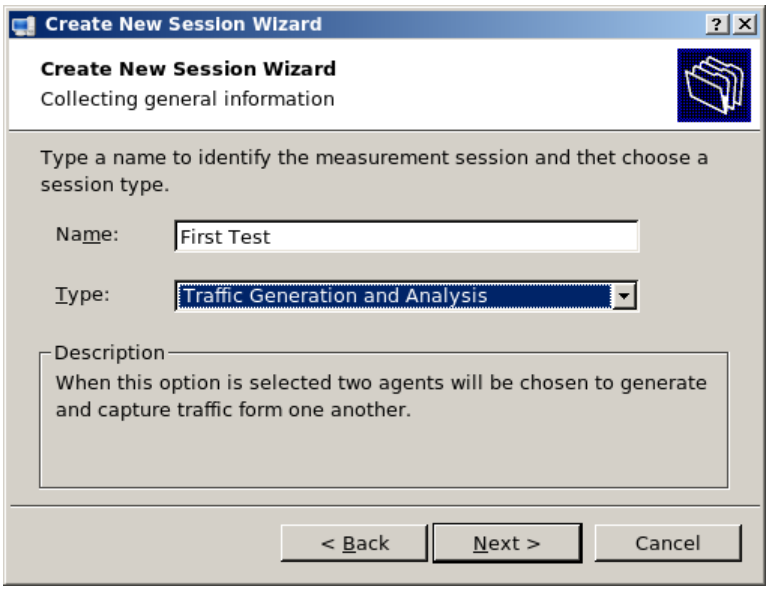
3.8.4 CREATING MEASUREMENT SESSIONS AND GROUPS

The next step in using the management console is creating measurement sessions. There was already explained in the previous sub-chapter that three session types are available:

- Traffic generating sessions
- Traffic analysis sessions
- Double traffic generating and analysis sessions

Due to the complex nature of the session creation process a wizard is also available to guide you through it. Figure 3.63 displays a snapshot of it, when you are required to select the session type.

Figure 3.63 The Create New Session wizard



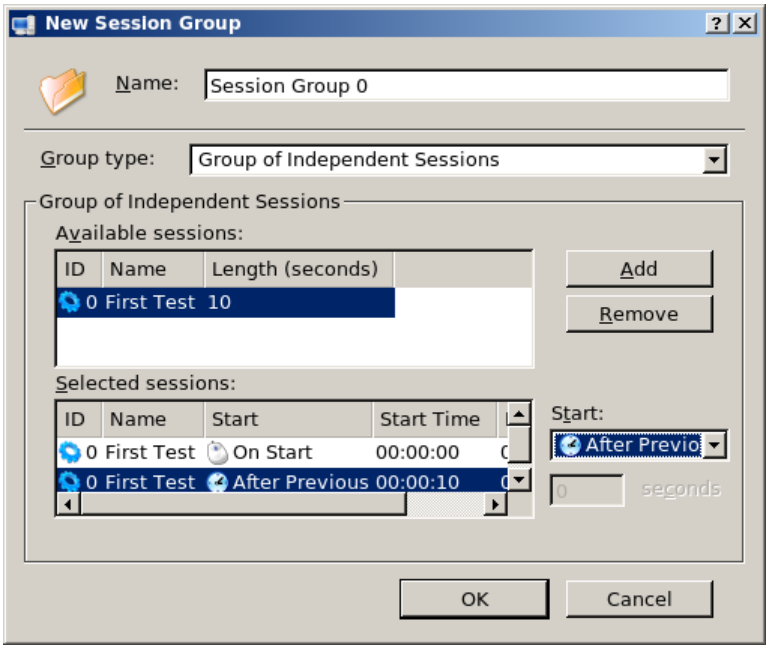
Depending on which session type you choose, you need to select one or two agents, and the configuration parameters for the tests depending on both the session and traffic type. The parameters required were already presented in the description of the Session Manager earlier in this document.

After the session has been created, it will be automatically added to the session list of the management console and it will be available for creating session groups and tasks. You may also save an existing session to a file, or load previously created session from file, in order to ease the session handling.

If you want to run multiple sessions at once, or single session with a variable parameter you must also create a session group. Since the group creation is very straightforward, a single dialog is available to do this in one-step. In figure 3.64 there is the dialog used for creating session groups, in the group of independent session mode. You may change the mode by selecting an alternate option in the top drop-down list.

After the group is created, it will be automatically added to the session group list and is made available for creating scheduled tasks.

Figure 3.64 Creating a session group

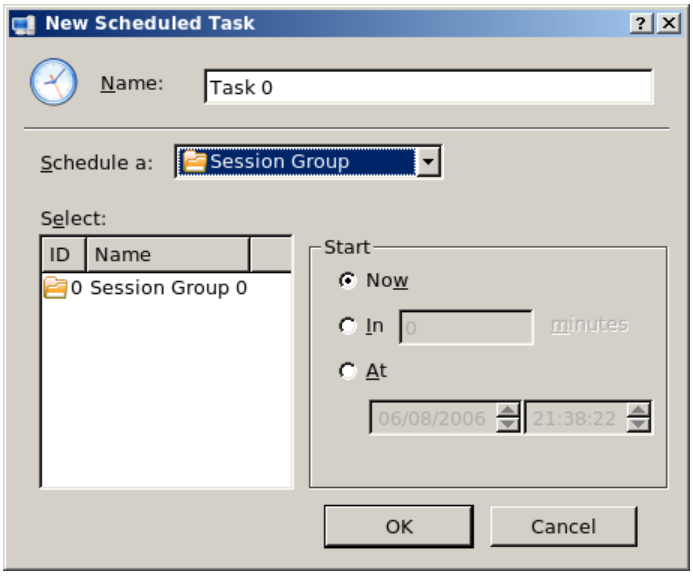


3.8.5 SCHEDULING TASKS AND COLLECTING RESULTS

To create a scheduled task you need to have at least one session or session group loaded (either a new created one or an existing one that has been loaded). To create the task use the **New Task** button on the scheduled tasks view. Similar to creating session groups a single dialog will appear in you need to specify the following:

- The type of objects that is scheduled: a session or a session group
- A session or a session group depending on the first selection
- A start moment which could be now, within a given number of minutes or at a given time moment

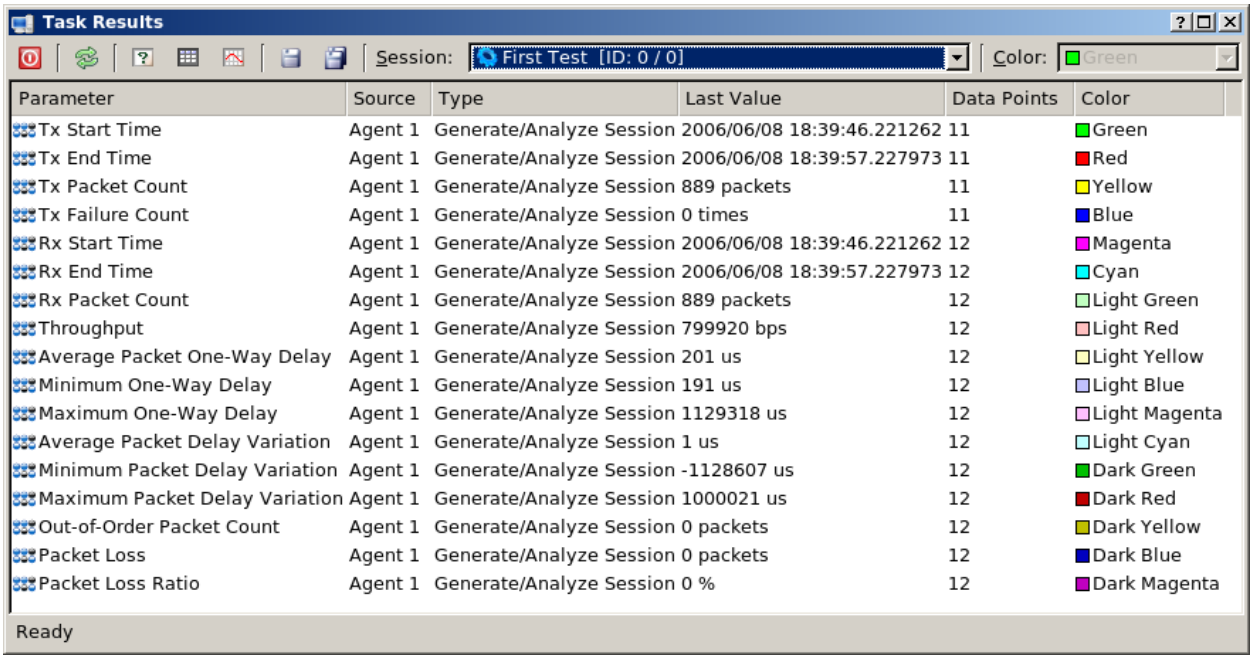
Figure 3.65 Creating a scheduled task



Remember that the session will start if the start up time of the session is less or equal with the current time, so specifying a past time moment will result in the task starting immediately. After the task has been created it will results a new task line being added to the management console's task list. For each task, the status of pending, running, finished or deleted is specified.

The results of a task are available both during the execution of the task, as QoS data is received from the agents and of course, at the end of the test. To access the results you must use to results view. Here, for each task that has at least one set of stored data, a new item will appear. Opening the result set will imply that a window similar to the one from figure 3.66 is created.

Figure 3.66 Results window



Parameter	Source	Type	Last Value	Data Points	Color
Tx Start Time	Agent 1	Generate/Analyze Session	2006/06/08 18:39:46.221262	11	Green
Tx End Time	Agent 1	Generate/Analyze Session	2006/06/08 18:39:57.227973	11	Red
Tx Packet Count	Agent 1	Generate/Analyze Session	889 packets	11	Yellow
Tx Failure Count	Agent 1	Generate/Analyze Session	0 times	11	Blue
Rx Start Time	Agent 1	Generate/Analyze Session	2006/06/08 18:39:46.221262	12	Magenta
Rx End Time	Agent 1	Generate/Analyze Session	2006/06/08 18:39:57.227973	12	Cyan
Rx Packet Count	Agent 1	Generate/Analyze Session	889 packets	12	Light Green
Throughput	Agent 1	Generate/Analyze Session	799920 bps	12	Light Red
Average Packet One-Way Delay	Agent 1	Generate/Analyze Session	201 us	12	Light Yellow
Minimum One-Way Delay	Agent 1	Generate/Analyze Session	191 us	12	Light Blue
Maximum One-Way Delay	Agent 1	Generate/Analyze Session	1129318 us	12	Light Magenta
Average Packet Delay Variation	Agent 1	Generate/Analyze Session	1 us	12	Light Cyan
Minimum Packet Delay Variation	Agent 1	Generate/Analyze Session	-1128607 us	12	Dark Green
Maximum Packet Delay Variation	Agent 1	Generate/Analyze Session	1000021 us	12	Dark Red
Out-of-Order Packet Count	Agent 1	Generate/Analyze Session	0 packets	12	Dark Yellow
Packet Loss	Agent 1	Generate/Analyze Session	0 packets	12	Dark Blue
Packet Loss Ratio	Agent 1	Generate/Analyze Session	0 %	12	Dark Magenta

In the first results window a list of all parameters that are available is presented along with the source for that parameter (the name of the agent), the session type - this is useful when you have a group with multiple sessions and from the top drop down list you select the session you want to visualize. Other information is the last value received for each parameter, the number of data points available so far and a color used to identify each parameter a little easier.

This window offers only generic information about the task results but not so much measurement data. For this reason, three other dialogs are available to provide data that are more detailed:

- A session summary containing global session information such as the start time, end time, total number of packets average values, minimum and maximum values over the entire session (see figure 3.67). If a dump for offline analysis is performed, a dump data summary is also provided.
- A time or parameter domain list, in which data entries are available for each session and globally for the group (see figure 3.69)
- The parameters displayed can be selected independently based on a source-basis, using the dialog from figure 3.68.
- Finally, a trace plot is available for the same parameters displayed in table format.

Figure 3.67 The session summary

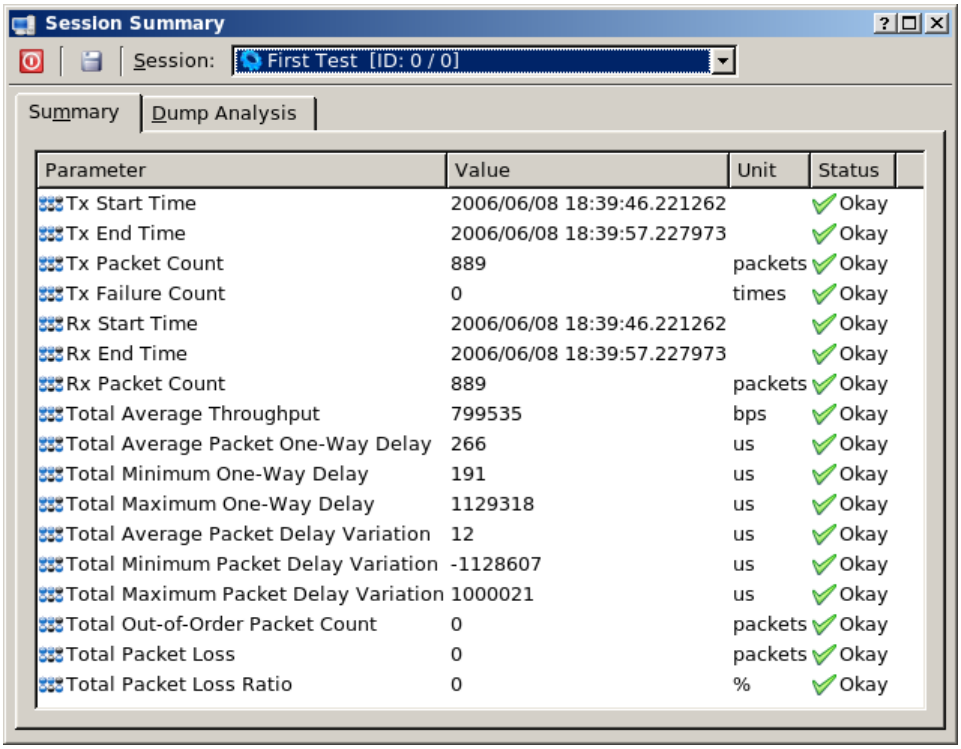
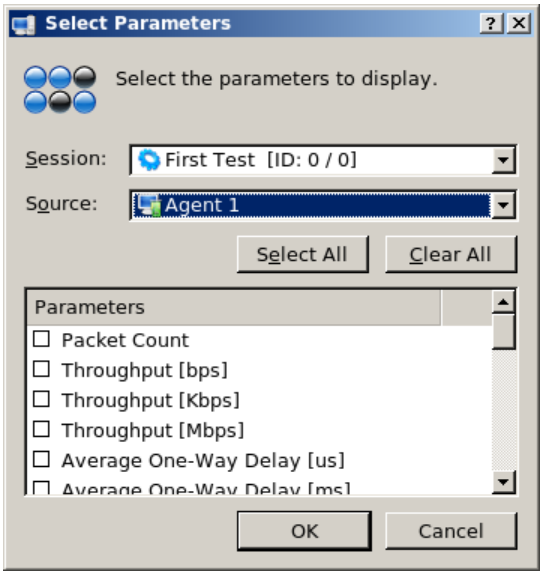


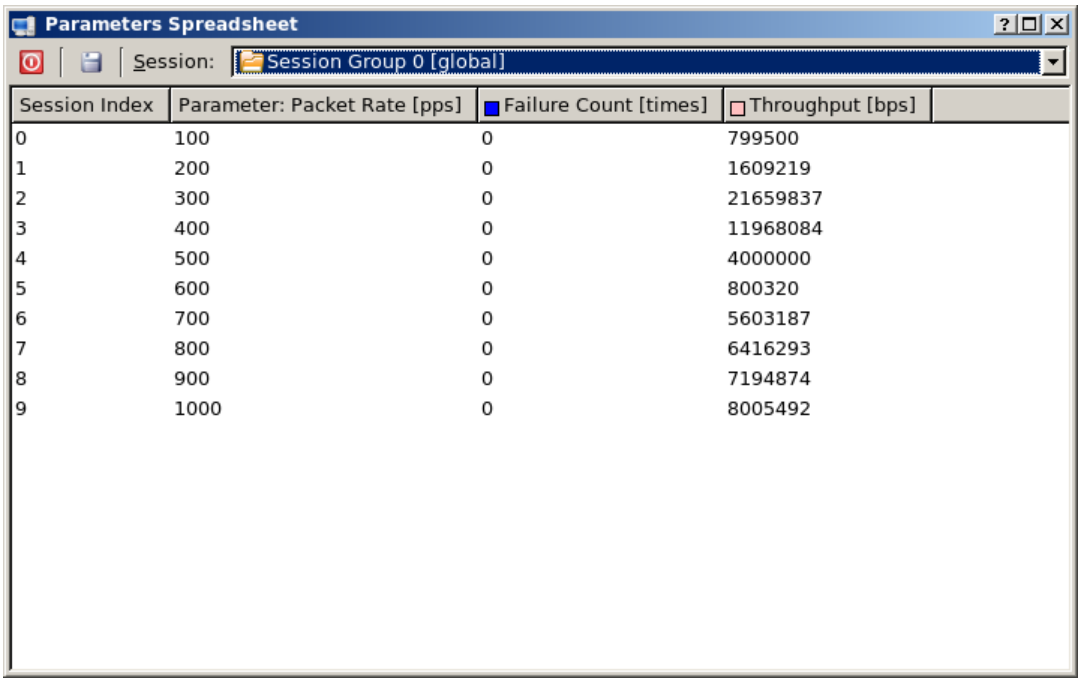
Figure 3.68 Selecting parameters for table-format display



The following figure displays the parameters for a session group, in the table format. This dialog and method of inspecting the results is also available for the sessions (either single or group members), the only exception being that for sessions the first two columns display time domain information rather than parameter domain.

In the next figure, the data was obtained from a group of parameter-dependent session having a packet rate between 100 and 1000 packets per second in increments of 100. The data in this format can be also saved into a spreadsheet CSV format, allowing analysis with third party applications.

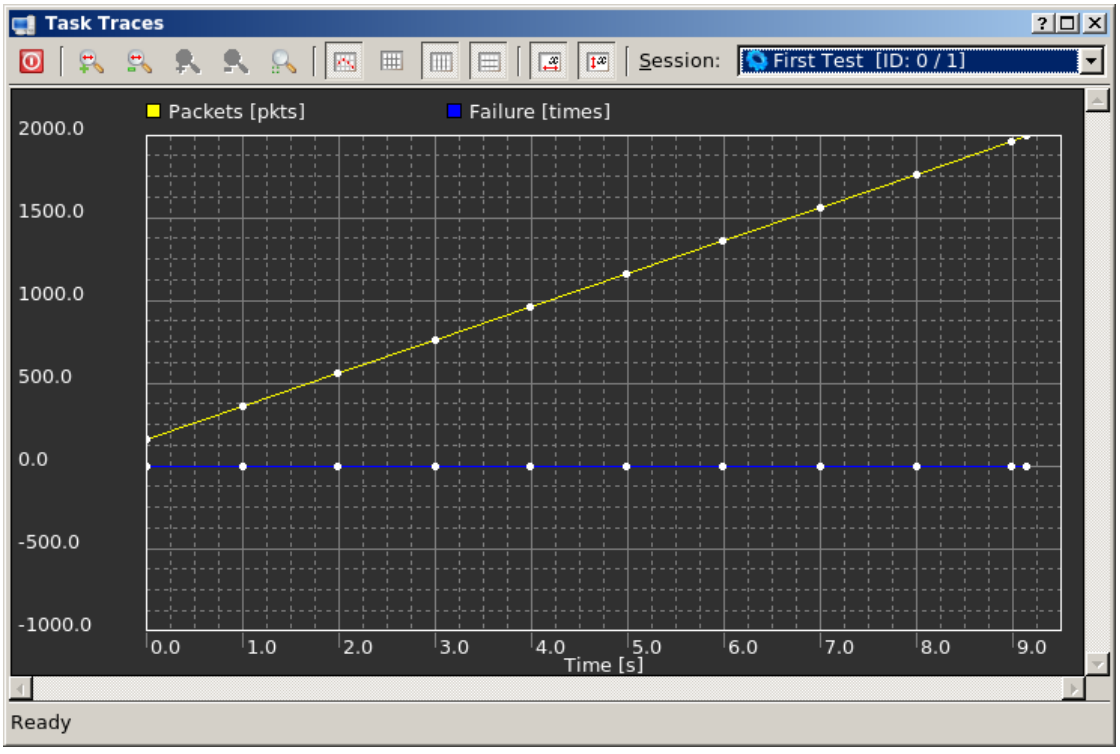
Figure 3.69 Displaying session (group) parameters in a table format



Session Index	Parameter: Packet Rate [pps]	Failure Count [times]	Throughput [bps]
0	100	0	799500
1	200	0	1609219
2	300	0	21659837
3	400	0	11968084
4	500	0	4000000
5	600	0	800320
6	700	0	5603187
7	800	0	6416293
8	900	0	7194874
9	1000	0	8005492

The last method of result analysis is the trace plotting. The selection of the data to be displayed is done using a dialog similar to the one from figure 3.68. The trace output is presented in figure 3.70. Trace handling controls are available to allow zooming, displaying of data points, grid and axis customization.

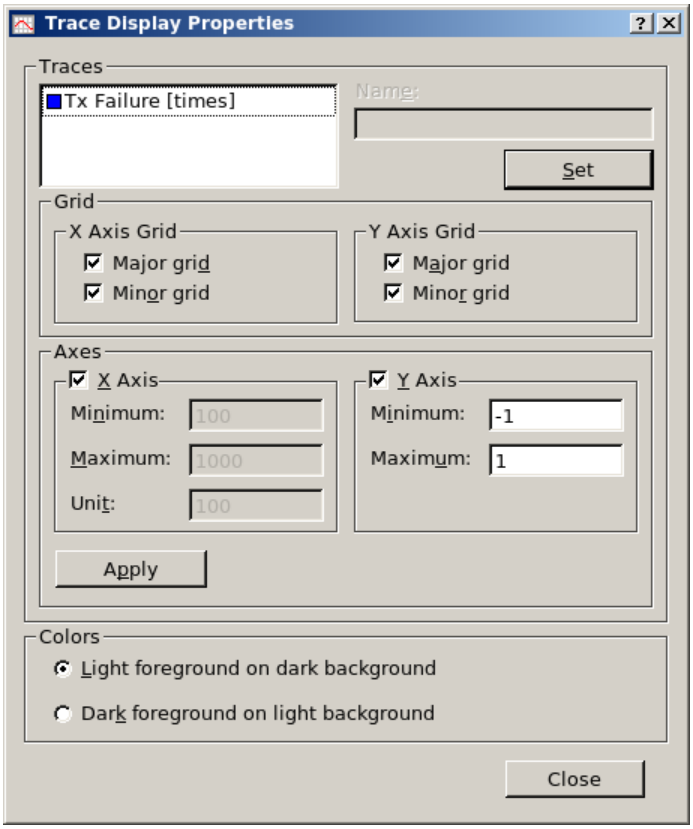
Figure 3.70 Visualizing data in trace format



An additional dialog can be used to refine the axis range, to change the colors of the background when indenting to use screenshots for publishing or to select titles for each trace, other than the

default one. The figure 3.71 presents the trace display properties dialog, which can be opened using the trace properties tool button.

Figure 3.71 The trace properties dialog



3.8.6 USING THE EVENT LOG

The event log is an embedded service in the NMS management console, that provides troubleshooting, debugging and notification information whenever a special event is occurring.

The event log service starts always before any other service (except the Service Control Manager) and it is ready of receiving event data from the application components. In the event log pane, this data is made available to the user.

For the event log, the user may specify the following:

- The log file name
- The maximum log file size
- The event log full action, that could be either of overwriting events automatically, overwrite events older that a specific number of days or to clear the event log manually using a clear log button.

For each event, the following information is available: the event type (information, warning or error), event code, source, description and parameters, if any.

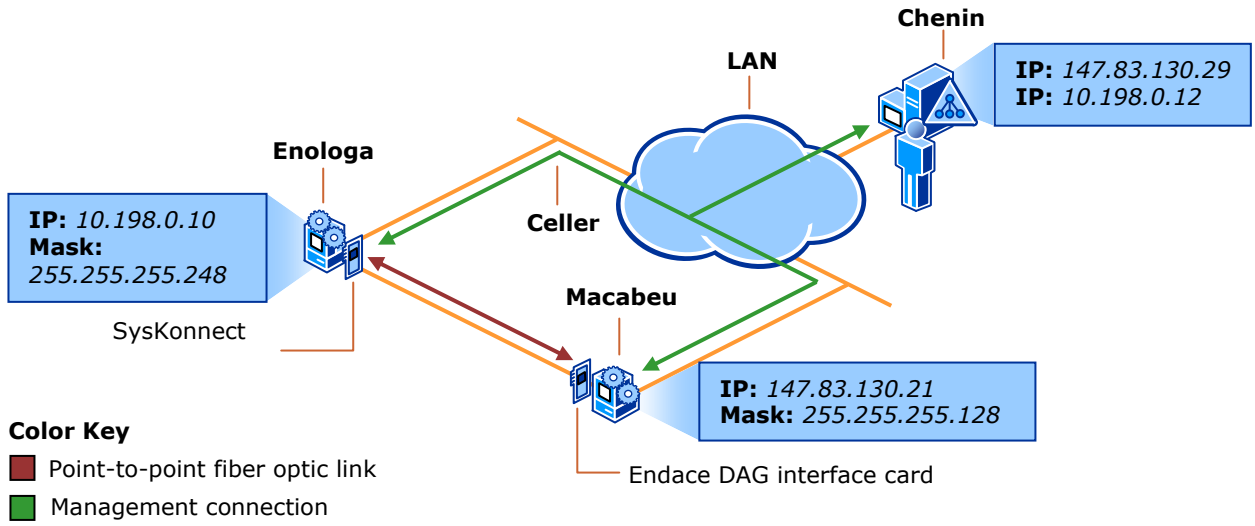
4

CONCLUSIONS

4.1 MEASUREMENT SCENARIO RECREATED

This last topic is intended to redo the initial experiment of testing Gigabit network interface cards, to see how NMS can be used to accomplish the job instead of the classical approach and what kind of results are available at the end.

Figure 4.1 Testing NICs with NMS scenario



In the new scenario, we have replaced the *Mazuela* computer with *Chenin* that has a NMS management console installed. On both *Enologa* and *Macabeu* NMS agents have been deployed. On *Enologa* there is a SysKonnect SK9843 NIC connected with 1000BaseSX to the Endace DAG card installed on *Macabeu*. The network traffic was generated with *Enologa* agent while the analysis is done, of course with the Endace card on *Macabeu*. Next, it is assumed that the following operations were completed successfully:

- The configuration of the management console
- The configuration of the measurement agents
- The agents registration process

The type of session used is flow-based using Ethernet encapsulation for data-link testing, both traffic generation and analysis. There were created three types of scenarios using periodic, Poisson and link flooding traffic. For each type of traffic distribution, were selected a set of packet sizes: 256, 512, 768, 1024 and 1500 bytes. The maximum transmission unit was 1500 bytes.

For each packet size, a session group was created to run a series of tests with packet rates going up to a gigabit for the selected packet size. In the following will be presented the results obtained for 256 and 1500 byte frames in the case of the periodic distribution.

The interesting parameters are:

- The average throughput
- The packet delay variation
- The packet loss ration

Figure 4.2 Average throughputs for 256-byte packets

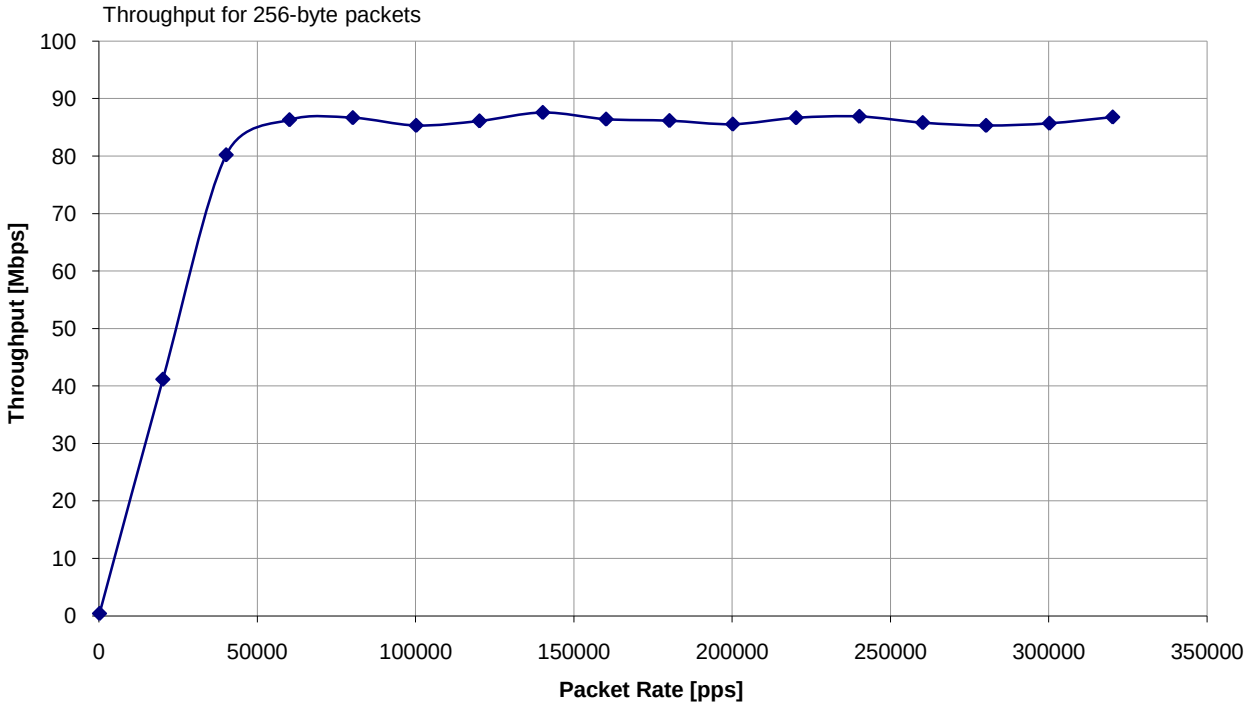
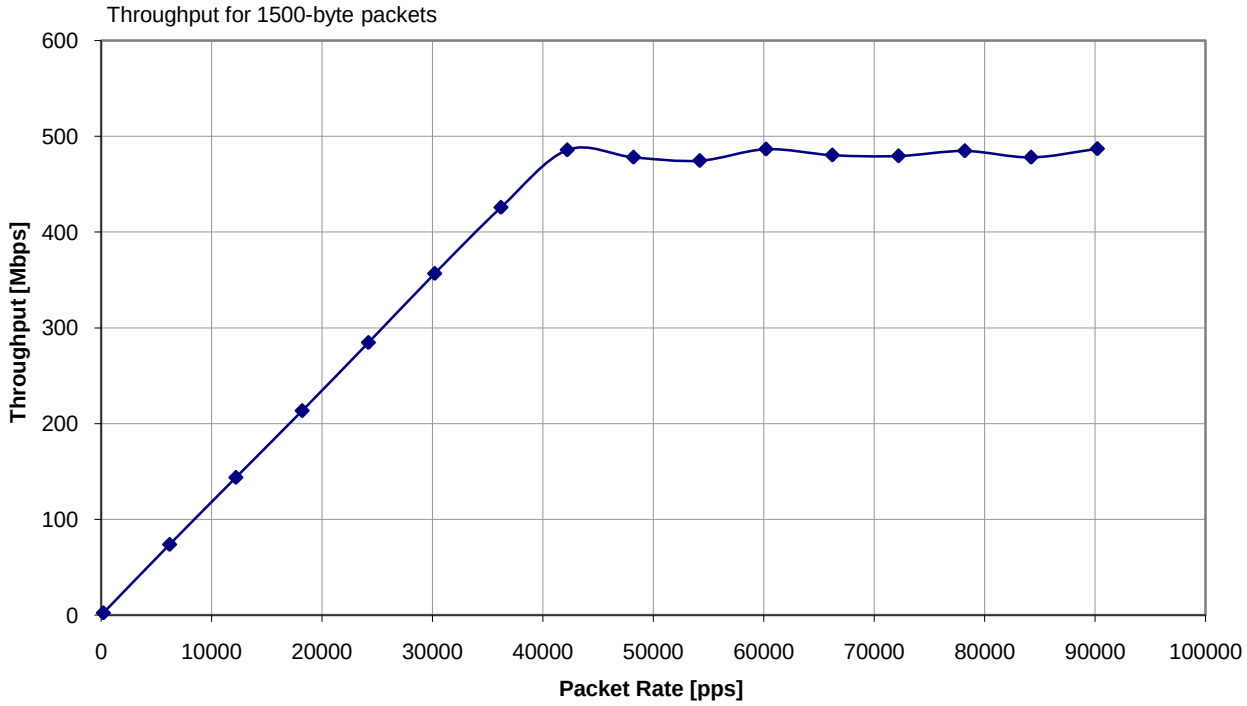


Figure 4.3 Average throughputs for 1500-byte packets



You may see that the obtained throughput depends largely on the packet size, while the software limitations in this case depend rather on the packet rate, in both situations at around 50000 packets per second.

In figures 4.3 and 4.4, are represented the number of packets transmitted and received for each of the previous two situations:

Figure 4.3 Transmitted and received number of packets for 256-byte frame

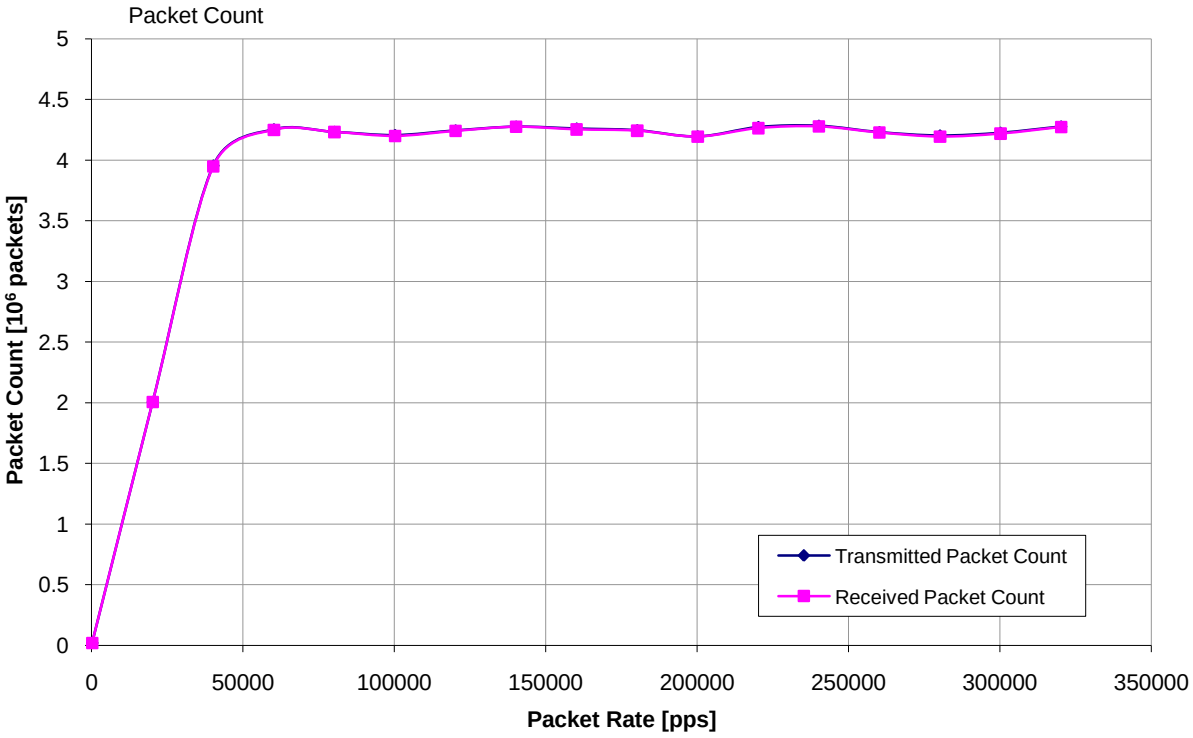
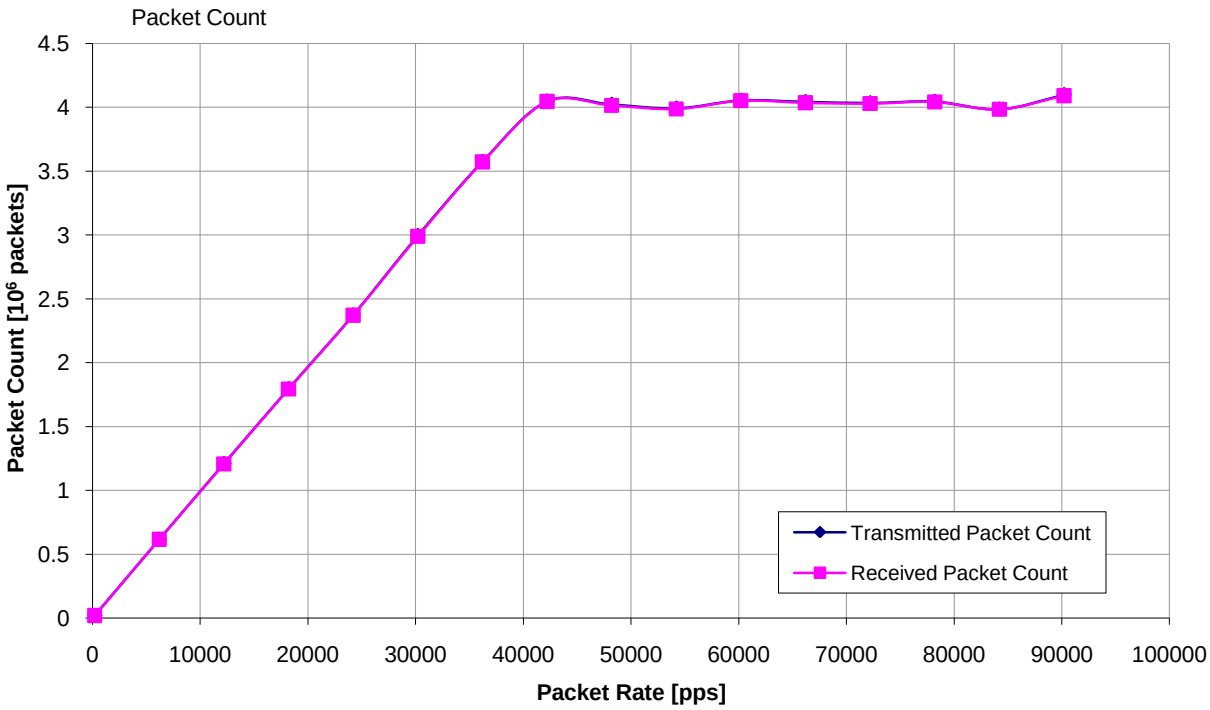


Figure 4.4 Transmitted and received number of packets for 1500-byte frame



The figures above are intended to show the kind of results you can obtain. The actual plots were made in Microsoft® Excel based on the data exported from the management console. All four figures contain the data from the session group with the parameter the packet rate.

In addition to the group data, session data is also available. Figures 4.5 to 4.8 show the same parameters as in the previous figures, i.e. the throughput for 256 and 1500-byte packets and the packet count for the same packet size, but this time versus time (i.e. during one session). The selected session was chosen to be in the middle of the group, to be representative for the average packet rate.

The duration of one session was of 100 seconds.

Figure 4.5 Transmitted packet count (packet size: 256 bytes, packet rate: 160200 pps)

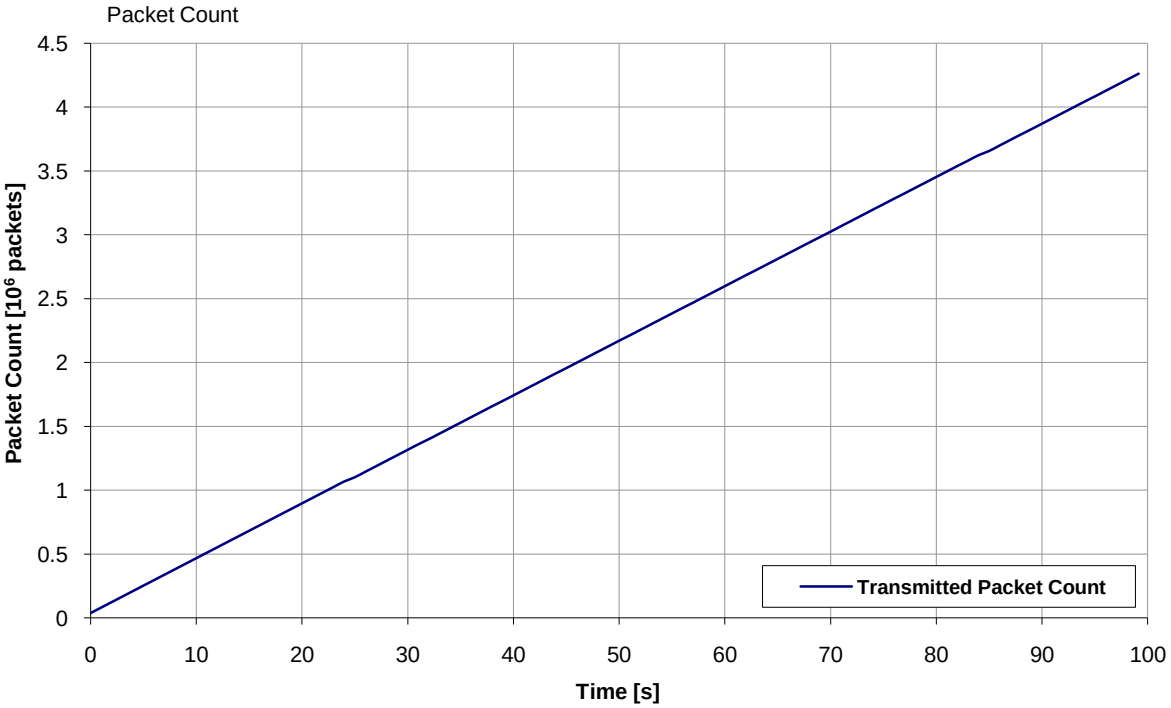


Figure 4.6 Transmitted packet count (packet size: 256 bytes, packet rate: 160200 pps)

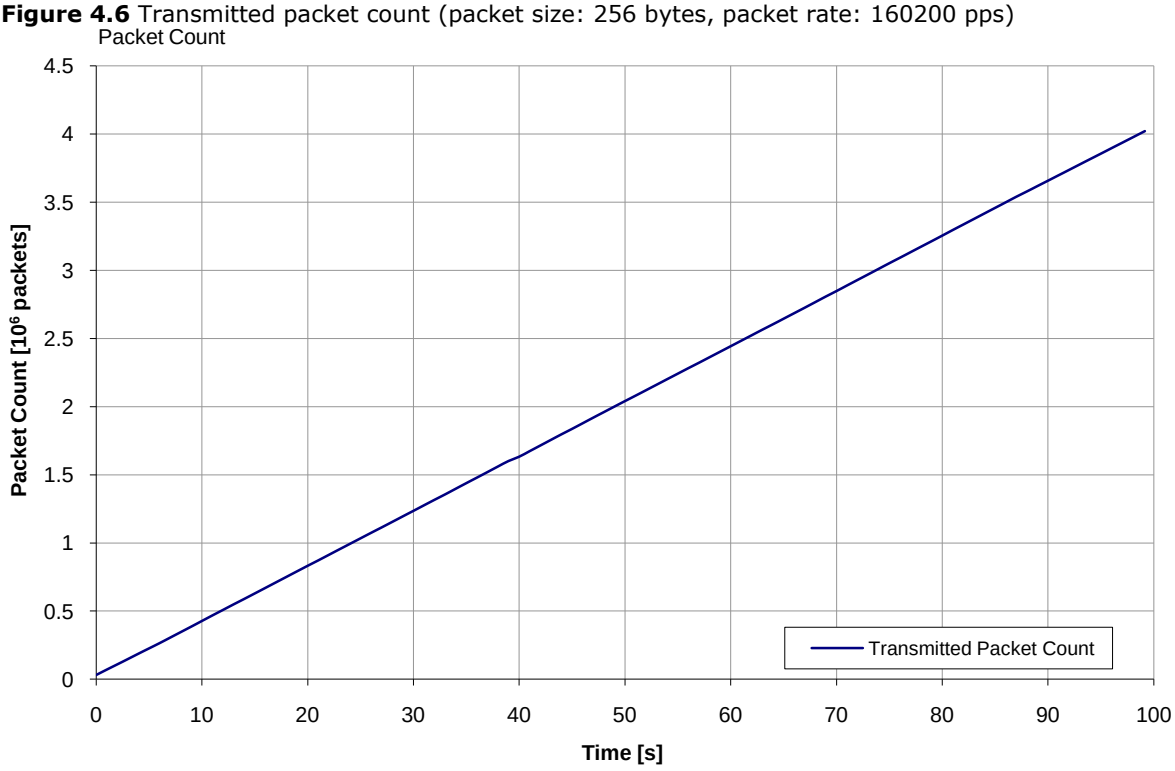


Figure 4.7 Throughput for 256-byte packets at 160200 pps

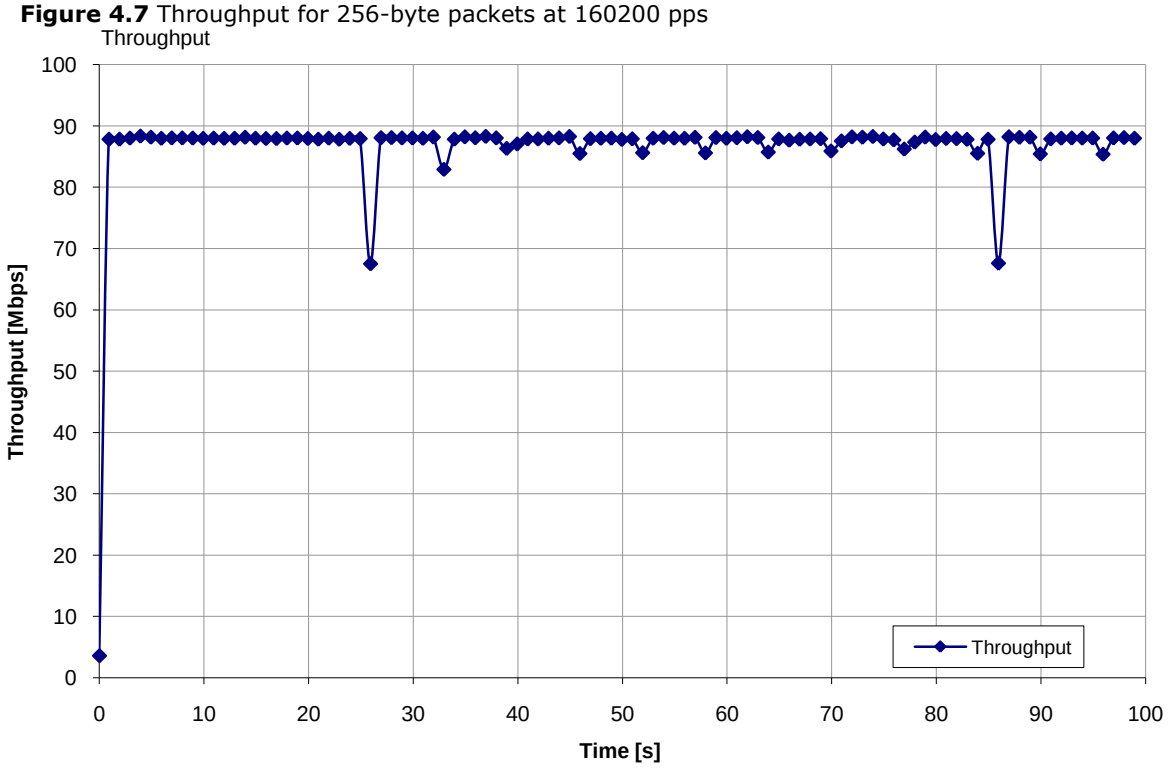
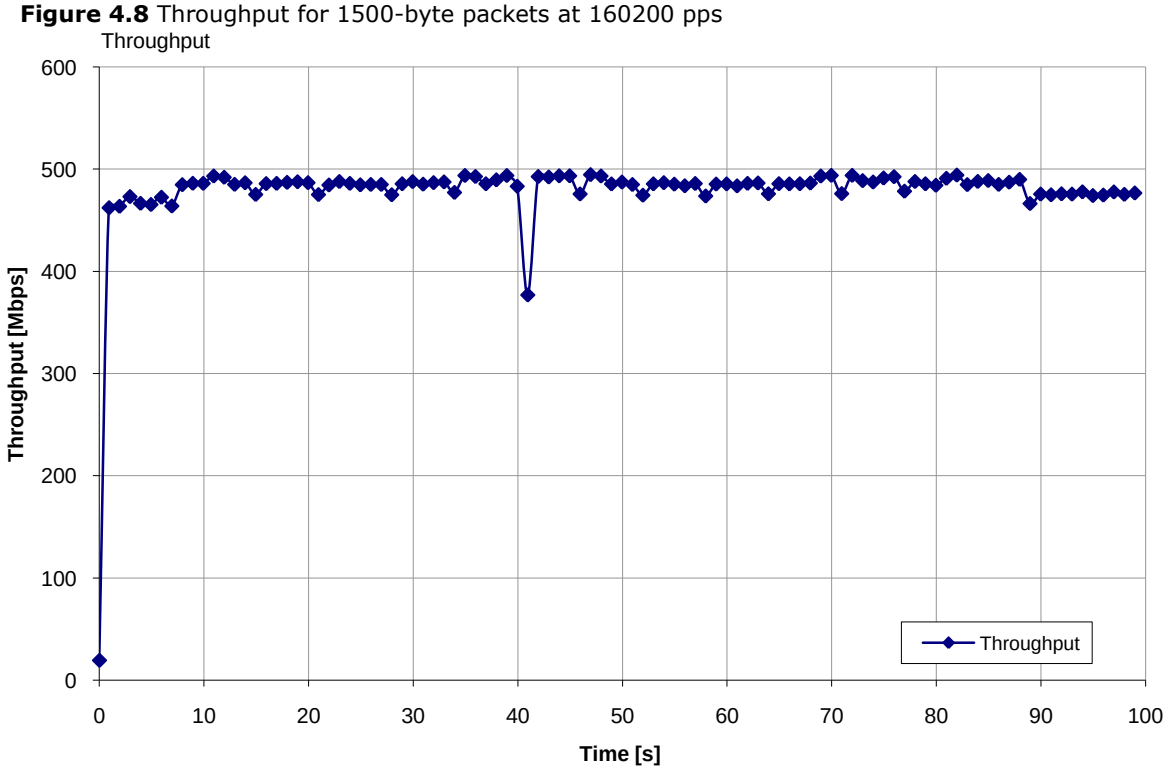


Figure 4.8 Throughput for 1500-byte packets at 160200 pps



You can see from previous figures that data availability is quite high, since the entire measurement session was done with very little efforts compared to the classical approach.

4.2 ADVANTAGES AND DISADVANTAGES OF THE PROPOSED SOLUTION

The Network Measurement System is not the perfect application. It has its inherent drawbacks like any beginning. Nevertheless, the following advantages are gain whey you choose NMS in favor of the existing measurement solutions:

- You can implement very quickly a wide palette of network measurements, with just several clicks and without moving from your computer. In a large test environment, once the machines are in placed, the software is installed all operations can be performed only from the management console. Since the start of a session takes just a few clicks in a wizard, it greatly saves a lot of time, when doing even complex measurements.
- NMS does not require user attendance during the execution of the tests. Maybe it is hard to believe but some of the results from the previous paragraph were performed at night. We just created the corresponding sessions, groups, scheduled some tasks and went home. The next morning, the management console displayed the list of results ready for exporting and interpretation.
- If you want to use *MGEN*, the first thing to do is to obtain the *MGEN* documentation and to study either the command line method (only the command syntax is almost half of page) or the script file format. To make the things even worse if you created a *MGEN* script and forgot to kick an enter after the last script line, you will probably spent half a day trying to understand why the last command from the script is not executed.
- It offers better performance for traffic generation.
- The user can focus on the objectives of the test, rather on how to implement them.
- It offers a high level of functionality, in most of the situations there is no need of using additional third party software.
- Finally, the programming model is open source. Therefore, if you have some problems or you found a better implementation feel free to put it in practice.

Among the disadvantages, it can be mentioned:

- The NMS is not completely portable across all GNU/Linux platforms. Nevertheless, since the source code is available with slight changes it can be ported to almost all major distributions.
- It is not optimized for local resource usage.
- It does not implement a fined-grained analysis system in order to save the limited bandwidth of the management link.
- The latest version of the Network Measurement System requires additional testing.

4.3 FUTURE WORK

The Network Measurement System is far from being a complete flaw-free set of applications. The previous list of disadvantages is clearly a thing that needs improvement. The following topics can therefore be considered as a future project in order to enhance the NMS capabilities:

- To eliminate the existing disadvantages
- To ensure support for other versions of SNMP
- To use managed objects set in the *private* class rather in the *experimental* class.
- Better communication control between the management console and the agent; in the exiting implementation there are situation in which an agent may begin an operation before the confirmation is received from the management console
- To extend the support for other networking technologies, hardware, protocols and QoS parameters

REFERENCES

[1] Alexandru Bikfalvi, *Network Measurement in Telecommunications*, draft paper, Technical University of Cluj-Napoca, 2005

[2] Paul Pătraş, *Distributed Agents for a Network Measurement System*, B.S. thesis, Technical University of Cluj-Napoca, 2006

[3] ●●●, *Internetworking Technologies Handbook*, Cisco Systems Inc, 2002

[4] ●●●, *Request for Comments 791*, Internet Engineering Task Force, 1981

[5] ●●●, *Request for Comments 3168*, Internet Engineering Task Force, 2001

[6] ●●●, *Request for Comments 2205*, Internet Engineering Task Force, 1997

[7] ●●●, *Request for Comments 2330*, Internet Engineering Task Force, 1998

[8] ●●●, *Request for Comments 2678*, Internet Engineering Task Force, 1999

[9] ●●●, *Request for Comments 2679*, Internet Engineering Task Force, 1999

[10] ●●●, *Request for Comments 2681*, Internet Engineering Task Force, 1999

[11] ●●●, *Request for Comments 2680*, Internet Engineering Task Force, 1981

[12] ●●●, *MGEN User's and Reference Guide*, U.S. Naval Research Laboratory, 2006

[13] ●●●, *Recommendation X.690*, International Telecommunications Union, 2002

[14] ●●●, *DAG 4.3 GE Card User Manual*, Endace Measurement Systems, 2005

[15] ●●●, *Endace Linux and Free BSD Software Installation*, Endace Measurement System, 2005