



**TECHNICAL UNIVERSITY OF CLUJ-NAPOCA**  
**Faculty of Electronics, Telecommunications and Information Technology**

Diploma Project

**DISTRIBUTED AGENTS FOR A NETWORK MEASUREMENT SYSTEM**

Author: **Paul-Horațiu Pătraș**

Supervisors:

**Prof. Virgil Dobrotă, PhD**  
Technical University of Cluj-Napoca

**Prof Jordi Domingo-Pascual, PhD**  
Polytechnic University of Catalonia

Cluj-Napoca, July 2006

# **ABSTRACT**

The goal of this project is to develop a remotely controlled multithreaded C/C++ software application for GNU/Linux platforms, capable of evaluating the QoS performances of network links based on active and passive measurement techniques.

The network measurement agent is able to simultaneously generate and analyse multiple flows of periodic and Poisson distributed traffic using Data Link layer (Ethernet frames), Network layer (IPv4 packets) and Transport layer (UDP datagrams) protocols. The packet generation is optimized for high-speed link testing by applying real-time scheduling policies to dedicated routines and network performance is evaluated using parallel threads specialised in QoS parameter computation.

The software also contains a module that enables real-time measurement using the Endace DAG card, which is initially provided by the manufacturer with tools that perform only hardware based capture at 100% line rate. The advantage of the proposed solution is that it replaces off-line processing, which is limited by the time required for hard-disk writing operations, with real-time processing and determines the following QoS parameters : one-way delay, packet delay variation, throughput and out-of-order packets (for a specified time interval, e.g. one second).

# TABLE OF CONTENTS

Abstract .....	1
Table of Contents .....	2
List of Figures .....	4
<b>State of The Art .....</b>	<b>5</b>
<b>Theoretical Fundamentals .....</b>	<b>7</b>
2.1. Quality of Service .....	7
2.1.1. The QoS Concept .....	7
2.1.2. QoS Metrics .....	9
2.1.2.1. Delay .....	9
2.1.2.2. Jitter .....	10
2.1.2.3. Packet Loss .....	11
2.1.2.4. Out-of-Order Packets .....	11
2.1.2.5. Throughput .....	11
2.1.2.6. Connectivity .....	11
2.1.3. QoS Mechanisms Implemented in Networking Technologies .....	12
2.1.3.1. Traffic Handling Mechanisms .....	12
2.1.3.2. QoS Implementation in IPv4 .....	12
2.1.3.3. QoS Implementation in IPv6 .....	13
2.1.3.4. IEEE 802.1p .....	13
2.1.3.5. Other Technologies Implementing QoS .....	14
2.2. Network Measurement And Monitoring .....	14
2.2.1. The Purpose of Internet Measurements .....	14
2.2.2. Architecture of Measurement Systems .....	15
2.2.3. Network Measurement Techniques .....	16
2.2.3.1. Passive Measurements .....	16
2.2.3.1. Active Measurements .....	17
2.3. Synchronisation Techniques .....	18
2.3.1. Network Time Protocol .....	18
2.3.2. Global Positioning System .....	19
2.3.2. Radio Signals .....	20
<b>Design And Experimental Results .....</b>	<b>21</b>
3.1. Motivation .....	21
3.2. Application Design .....	22
3.2.1. Software Architecture of the Network Measurement Agent .....	23
3.2.2. The Graphical User Interface .....	25
3.2.3. The Command Line Interface .....	26
3.2.4. Application Initialisation Steps .....	28
3.2.5. The Traffic Control Service .....	29
3.2.5.1. Main Control Thread and Socket Threads .....	29
3.2.5.2. IEEE 802.3 / Ethernet Frame Generation .....	32
3.2.5.3. IPv4 Packet Generation .....	36
3.2.5.4. UDP Traffic Generation .....	38
3.2.5.5. IEEE 802.3 / Ethernet Frames Analysis .....	39
3.2.5.6. IPv4 Packet Analysis .....	42
3.2.5.7. UDP Datagram Analysis .....	43
3.2.6. The PCAP Service .....	44

---

3.2.7. The Endace Card Module.....	46
3.2.7.1. Overview of the Endace DAG 4.3 GE Card .....	46
3.2.7.2. DAG Card Configuration .....	47
3.2.7.3. DAG Module Implementation .....	48
3.2.8. The Management Service .....	50
3.2.9. The Message Dispatcher .....	51
3.2.10. Measurement Agent Configuration.....	54
3.3. Experimental Results.....	64
3.3.1. Evaluating the Performances of Some Gigabit Ethernet Cards .....	64
3.3.2. Comparing the Network Measurement Agent with other Tools .....	71
<b>Conclusions .....</b>	<b>73</b>
References .....	74
Appendix A .....	75

## LIST OF FIGURES

<b>Figure 2.1.</b> QoS facilities in IPv4 .....	12
<b>Figure 2.2.</b> QoS implementation in IPv6.....	13
<b>Figure 2.3.</b> Layer 2 QoS implementation using IEEE 802.1p.....	13
<b>Figure 2.4.</b> Basic architecture for a Measurement and Monitoring System.....	15
<b>Figure 3.1.</b> Basic scenario of using the Network Measurement System.....	22
<b>Figure 3.2.</b> The architecture of the Measurement Agent application.....	24
<b>Figure 3.3.</b> A snapshot of the Network Measurement Agent GUI.....	25
<b>Figure 3.4.</b> Example of remote start-up of the Network Measurement Agent.....	26
<b>Figure 3.5.</b> Probability distribution function of a Poisson process .....	32
<b>Figure 3.6.</b> Real-time failure correction methods.....	33
<b>Figure 3.7.</b> The Ethernet / IEEE 802.3 frame structure.....	34
<b>Figure 3.8.</b> The IP header structure .....	37
<b>Figure 3.9.</b> Data dumping mechanism.....	41
<b>Figure 3.10.</b> QoS parameters calculation procedure .....	42
<b>Figure 3.11.</b> The architecture of the Endace DAG 4.3 GE card.....	46
<b>Figure 3.12.</b> The Ethernet variable length record.....	48
<b>Figure 3.13.</b> The management MIB .....	52
<b>Figure 3.14.</b> The management MIB (continued).....	53
<b>Figure 3.15.</b> Configuration of the machines running the measurement agent .....	54
<b>Figure 3.16.</b> First run of a Network Measurement Agent in GUI mode .....	55
<b>Figure 3.17.</b> Network interfaces assignments .....	55
<b>Figure 3.18.</b> Agent details set up.....	56
<b>Figure 3.19.</b> The <i>Managers</i> tab .....	57
<b>Figure 3.20.</b> Ports configuration for management interfaces .....	57
<b>Figure 3.21.</b> SNMP community names list .....	58
<b>Figure 3.22.</b> The IP filter .....	59
<b>Figure 3.23.</b> The error log .....	59
<b>Figure 3.24.</b> First experimental set-up for testing the Intel card .....	64
<b>Figure 3.25.</b> Packet count for 256-byte periodic frames .....	65
<b>Figure 3.26.</b> Packet loss ratio for 256-byte periodic frames .....	66
<b>Figure 3.27.</b> Real-time failures for 256-byte periodic frames.....	66
<b>Figure 3.28.</b> Packet delay variation for 256-byte periodic frames .....	67
<b>Figure 3.29.</b> Throughput for 256-byte periodic frames.....	67
<b>Figure 3.30.</b> Packet loss ratio for 1500-byte Poisson distributed frames.....	68
<b>Figure 3.31.</b> Real-time failures for 1500-byte Poisson distributed frames .....	68
<b>Figure 3.32.</b> Throughput for 1500-byte Poisson distributed frames .....	68
<b>Figure 3.33.</b> Packet delay variation for 1500-byte Poisson distributed frames.....	69
<b>Figure 3.34.</b> Throughput for link flood with 1500-byte frames .....	69
<b>Figure 3.35.</b> Second experimental set-up for testing the Intel card.....	70
<b>Figure 3.36.</b> Throughput for 1500-byte periodic frames.....	70
<b>Figure 3.37.</b> Out-of-order packets for 1500-byte periodic frames .....	71
<b>Figure 3.38.</b> Packet delay variation for 1500-byte periodic frames .....	71
<b>Figure 3.39.</b> NMA vs MGEN performances .....	72

# 1

## STATE OF THE ART

When the Internet was initially developed, there was no perceived need for guaranteeing a certain quality level for network based application, all data being exchanged based on a 'best effort' approach. Meanwhile, it has evolved so fast that at present it is possible to view the infrastructure of Internet as a cybernetic ecosystem. Multiple data and telecommunication technologies have been interconnected and the Internet is shifting towards a next generation network, which already started to have a great impact over the social and economical life and consequently needs to be operated in a reliable and efficient manner [1]. Nowadays, end users and small to large enterprises demand real-time network quality and most of the internet service providers (ISP) are constrained to provide Quality of Service (QoS). QoS refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET and IP-routed networks that may use any or all of these underlying technologies. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled jitter and latency (required by some real-time and interactive traffic) and improved loss characteristics. It is also important to make sure that providing priority for one or more flows does not make other flows fail. QoS technologies provide the elemental building blocks that will be used for future business applications in campus, wide area networks (WAN), and service provider networks [4].

In order to provide QoS, first it is required to be able to properly characterise traffic, intelligently provision network resources and to tune the employed traffic engineering mechanisms. Despite of the significant work performed recently in this area, the Internet traffic is not yet well understood and considered as difficult to be described by mathematical formulas. Consequently, most of the research on improving heterogeneous IP QoS networks is based on well-suited measurements [3].

The fast development of QoS demanding application (inelastic services) determined various networking technologies to implement mechanisms that meet the demand for guaranteed values of some parameters. For example, multi-media streaming requires guaranteed throughput, IP telephony quality is affected by delay and jitter and safety critical applications such as remote surgery demand guaranteed availability. Traffic measurement technologies have become vital for network design, service level agreement (SLA) validation, traffic engineering applications, developing next-generation network devices, protocols, applications, and services supplied by ISPs.

A large number of tools have been developed by companies and research groups in order to simplify the measurement and monitoring process and to provide a reliable image of a network segment that could help choosing the appropriate techniques to full-fill the requirements of the user connected to the network. Despite this, improvement of the existing tools are still necessary, since none of them is able to provide multiple functionalities in order to perform different type of tests, with as little assistance as possible and present the results in such manner that they are easy

to interpret. When performing Internet measurements telecommunication engineers need to be able to operate multiple software and find a way to correlate all the gathered results, such that appropriate conclusions can be drawn.

Our project offers a solid alternative to previous measurement and monitoring tools by trying to combine the features of several simple tools into an integrated solution. We aimed to transform the measurement process into an easier task. It permits the control of the distributed measurement system from a single node, tests can be automated and scheduled, results can be collected in real-time or performed upon test completion and the obtained QoS parameters can be exported as tables or plotted. The measurement probes can be configured locally through a graphical user interfaces or through a remote connection. Multi-layer tests can be performed in order to asses the performance of a LAN link, a multi-hop link or the quality of an end-to-end connection between two dedicated applications.

The latest developments of hardware dedicated NICs led to the demand of software capable of acquiring the data from such device and extract the essential QoS. Nevertheless, software should provide a way of efficiently use such hardware in measurement systems that mostly rely on conventional network controller. Our project contains a module that makes possible QoS parameters evaluation using a dedicated network adapter (Endace DAG card), improving the rudimentary tools offered by the manufacturer.

The increasing use of Gigabit and 10 Gigabit Ethernet in WAN raises additional problems for QoS evaluation. Powerful computers are required, real-time operating systems become essential and the performance of Gigabit Ethernet controllers need to be evaluated prior to Internet measurements. The project presented in this paper also aimed to optimise the measurement software in order to permit the evaluation of various Gigabit cards and finally determine the bottleneck of some measurement stations and a set of NICs.

Nevertheless, we have tried to analyse a variety of passive and active network measurement tools that have been already deployed and are currently used by engineers (CoralReef, NeTraMet, OreNETa, etc.). Existing software can be downloaded from [8] together with additional documentation. We have compared the performances of the developed tool with previous works such as NetMeter [9] and MGEN [10].

# 2

## THEORETICAL FUNDAMENTALS

### 2.1. QUALITY OF SERVICE

#### 2.1.1. The QoS Concept

Quality of Service is a traffic engineering term which refers to the probability of a certain network to meet a given traffic contract i.e. guarantee a level of performance, throughput, latency usually by prioritizing traffic. QoS defines a set of standards and mechanisms to ensure high-quality performance for critical applications. By using QoS mechanisms, network administrators can use existing resources efficiently and ensure the required level of service without reactively expanding or over-provisioning their networks. Traditionally, the concept of quality in networks meant that all network traffic was treated equally. The result was that all network traffic received the network's best effort, with no guarantees for reliability, delay, variation in delay or other performance characteristics. With best-effort delivery service, however, a single bandwidth-intensive application can result in poor or unacceptable performance for all applications. The QoS concept of quality is one in which the requirements of some applications and users are more critical than others, which means that some types of traffic need preferential treatment [7]. The goal of QoS is to provide preferential delivery service for the applications that need it by ensuring sufficient bandwidth, controlling latency and jitter and reducing data loss. QoS also can be used to improve the throughput of traffic that crosses a slow link, such as a dial-up connection.

When two end hosts transfer packets over a network, a lot of unexpected events can occur and affect the quality of the communication process. Routers might drop packets if they arrive when their buffers are already full and the receiving application must ask for this information to be retransmitted, possibly causing severe delays in the overall transmission. Queuing mechanisms, multiple path routing and congestion avoidance mechanisms introduce unpredictable delays or out-of-order packet delivery. Nevertheless, packet can get corrupted along the route to the destination. When implementing QoS the following benefits will be provided [12]:

- control and management of network resources,
- increased priority for time-sensitive and mission-critical applications while allowing other applications access to the network,
- improved end-user experience,
- cost reduction by using existing resources efficiently.

All the network elements along the path that prioritized traffic takes must support QoS. Such network elements include the sending and receiving hosts, Data Link layer network devices (bridges and switches) and Network layer devices (routers). If a network device along this path

does not support QoS, the traffic flow receives the standard first-come, first-served treatment on that network segment.

The basic QoS architecture defines three fundamental pieces for implementation:

- QoS identification and marking techniques for coordinating QoS from end to end between network elements,
- QoS within a single network element (queuing, scheduling, and traffic-shaping tools),
- QoS policy, management, and accounting functions to control end-to-end traffic across a network.

To provide preferential service to a type of traffic, it must first be identified. Common methods of identifying flows include access control lists (ACLs), policy-based routing, committed access rate (CAR) and network-based application recognition (NBAR). Second, the packet may or may not be marked. These two tasks make up classification. When the packet is identified but not marked, classification is said to be on a per-hop basis. This is when the classification pertains only to the device that it is on, not passed to the next router. This happens with priority queuing (PQ) and custom queuing (CQ).

Congestion management, queue management, link efficiency and shaping/policing tools provide QoS within a single network element. Because of the bursty nature of traffic, sometimes the amount of traffic exceeds the speed of a link. Congestion-management tools will buffer traffic and implement priority queuing, custom queuing, weighted fair queuing (WFQ) and class-based weighted fair queuing (CBWFQ). Because queues are not of infinite size, they can fill and overflow. When a queue is full, any additional packets cannot get into the queue and will be dropped. To avoid tail drop a mechanism will try to make sure that the queue does not fill up and allow some sort of criteria for dropping packets that are of lower priority before dropping higher-priority packets.

Link efficiency involves the elimination of too many overhead bits. For example, RTP headers have a 40-byte header. With a payload of as little as 20 bytes, the overhead can be twice that of the payload in some cases. RTP header compression (also known as Compressed Real-Time Protocol header) reduces the header to a more manageable size.

Shaping is used to limit the full bandwidth potential of the flows. This is often used to prevent the overflow remote low bandwidth links connected to high speed links. Traffic above the configured rate is buffered for transmission later to maintain the rate configured. Policing is similar to shaping, but it differs in one very important way: traffic that exceeds the configured rate is not buffered (and normally is discarded).

QoS management helps to set and evaluate QoS policies and goals. Typically, the traffic characteristics of the network are determined through measurements, QoS techniques are deployed and the results are evaluated by testing the response of the targeted applications to see whether the QoS goals have been reached. In an ever-changing network environment QoS is not a one-time deployment, but an ongoing, essential part of network design.

Applications requiring QoS are also called real-time applications or inelastic services, i.e. they require a certain level of some metrics to function. If the values of the metrics exceed the minimum level, the additional resources will not be used but values below the limits will render the service non-functioning. IP telephony applications impose strict limit on delay and jitter, multimedia applications demand guaranteed throughput and safety-critical applications will

require a guaranteed level of availability (hard QoS). In the next paragraph we will focus on defining the quality of service parameters and explain how they can influence the performance of some applications present in every day life.

## 2.1.2. QoS Metrics

When referring to validating or providing QoS it means that some specific performance attributes associated with a service need to be measured or guaranteed. In the IP network environment QoS can be characterised by a small set of measurable parameters.

IETF has defined a set of metrics together with measurement methodology in [20]. IP Performance Metric Group (IPPM) continued the work begun by Benchmarking Methodology Working Group (BMWG) in order to develop a set of standard metrics that can be performed by network operators, end user or independent testing groups to measure the quality, reliability and performance of real Internet services. The adopted metrics are the following:

- Connectivity
- One-way delay and loss
- Round-trip delay and loss
- Delay variation
- Loss patterns
- Packet reordering
- Bulk transport capacity
- Link bandwidth capacity

### 2.1.2.1. Delay

Due to waiting queues, packet routing and congestion avoidance mechanisms a certain *latency* appears which can be described by one way delay and round trip delay. One-way Delay (OWD) is the time in seconds that a packet spends in travelling across the IP network from a host to another [13]. The transmitted packets need to be identified at source and destination in order to avoid packet loss or packet reordering.

$$OWD = t_{dest} - t_{src} > 0 [s] \quad (2.1)$$

In order to accurately measure OWD the source and the destination of packets must be closely synchronized. GPS systems afford one way to achieve synchronization to within several microseconds. Ordinary application of NTP may allow synchronization to within several milliseconds, but this depends on the stability and symmetry of delay properties among those NTP agents used, and this delay is what we are trying to measure. A combination of some GPS-based NTP servers and a conservatively designed and deployed set of other NTP servers should yield good results.

The developed Network Measurement Agent (NMA) is able to compute the average, minimum and maximum values of one-way delay. The average will be computed for  $n$  number of packets and the minimum and maximum values for the overall length of a test ( $N$  packets).

$$OWD_{avg} = \frac{\sum_{i=1}^n OWD_i}{n} [\mu s] \quad (2.2)$$

$$OWD_{min} = \min_{i=1}^N \{OWD_i\} [\mu s] \quad (2.3)$$

$$OWD_{max} = \max_{i=1}^N \{OWD_i\} [\mu s] \quad (2.4)$$

Another delay metric is the Round-trip Time (RTT) and is defined as the time elapsed from the moment a packet has been sent by a source to a destination until the moment a reply has been received, considering that the destination will immediately send a response upon the reception of the packet [14].

$$RTT = \Delta t_{src \rightarrow dest} + \Delta t_{dst \rightarrow src} [s] \quad (2.5)$$

The measurement of round-trip delay instead of one-way delay has several weaknesses. The Internet path from a source to a destination may differ from the path from the destination back to the source ("asymmetric paths"), such that different sequences of routers are used for the forward and reverse paths. Therefore round-trip measurements actually measure the performance of two distinct paths together. Even when the two paths are symmetric, they may have radically different performance characteristics due to asymmetric queuing. On the other hand, the measurement of round-trip has the advantage that it is easy to be deployed. Unlike in one-way measurement, it is often possible to perform some form of round-trip delay measurement without installing measurement-specific software at the intended destination. A variety of approaches are well-known, including use of ICMP Echo or of TCP-based methodologies.

### 2.1.2.2. Jitter

Packets may reach the destination with different delays due to unpredictable behaviour of the network. The delay variation is known as jitter and affects the quality of streaming applications. IETF has defined in [15] the IP Packet Delay Variation as the difference between the one-way-delay of two consecutive packets.

$$PDV_i = OWD_i - OWD_{i-1} [s], i > 1 \quad (2.6)$$

The agents will be able to measure the average maximum and minimum values of packet delay variation similar to one-way-delay. The relations used for their computation are presented below.

$$PDV_{avg} = \frac{\sum_{i=2}^n PDV_i}{n} [\mu s] \quad (2.7)$$

$$PDV_{min} = \min_{i=2}^N \{PDV_i\} [\mu s] \quad (2.8)$$

$$PDV_{max} = \max_{i=1}^N \{PDV_i\} [\mu s] \quad (2.9)$$

### 2.1.2.3. Packet Loss

It is possible to evaluate the reliability of a link by counting the number of lost packets. Packets are discarded during transfer through a network, typically due to congestion. The rate at which packets are dropped is measured by the developed software as packet loss ratio (PLR) (2.10), the management console receiving the number of transmitted and received packets from both agents involved in the measurement process.

$$PLR = \frac{Tx\_packets - Rx\_packets}{Tx\_packets} \cdot 100[\%] \quad (2.10)$$

### 2.1.2.4. Out-of-Order Packets

Packets may take different routes and arrive with different delays at the destination, resulting in an *out-of-order delivery*. This problem requires additional protocols responsible for rearranging out-of-order packets once they reach their destination. In order to be able to determine if the packets arrived at the destination in a different order from the one they have been transmitted, it is necessary that they have a sequence number attached. If the packet arrives, but is corrupted, then it is counted as lost [16]. Our approach will add a stamp to the generated packets that will include a sequence number. At reception, if two consecutively received packets do not have consecutive sequence numbers it will be considered that they had been delivered out of order. A counter will be incremented for each out-of-order received packet during the runtime of a test.

### 2.1.2.5. Throughput

Basically, the throughput represents the rate at which data is transmitted over the network. A more precise definition would be the amount of data that can be transferred through a digital connection in a given time period (in other words, the connection's bit rate). It is also called channel capacity in telecommunications contexts, and is usually measured in bits or bytes per second. The Network Measurement System evaluates the throughput as the amount of data received during a certain time interval (2.11).

$$\theta = \frac{8 \times total\_number\_of\_received\_bytes}{t_2 - t_1} [bps] \quad (2.11)$$

### 2.1.2.6. Connectivity

Packet-switched networks do not offer a permanent connection between two parties, but there are applications which require instantaneous connectivity at a certain moment when some critical data is to be transmitted. A source has instantaneous connectivity to a destination at a moment  $T$  if a packet transmitted by the source at time  $T$  will arrive at the destination [17]. Most application will demand bidirectional connectivity (TCP applications) but for other such as security related application unidirectional connectivity may be of interest.

## 2.1.3. QoS Mechanisms Implemented in Networking Technologies

### 2.1.3.1. Traffic Handling Mechanisms

There are two essential ways of providing QoS: Over-provisioning and resource reservation. The first approach is very simple but inefficient. The idea is to provide lots of resources to meet an expected demand with a large safety margin. The second method requires that a certain application makes a reservation which is accepted only if the router can provide reliable service. Consequently, the users are charged according to their needs. Resource reservation can be implemented based on two mechanisms, IntServ and DiffServ [6].

The Integrated Services architecture applications uses the Resource Reservation Protocol (RSVP) to request and reserve resources through a network. IntServ mechanisms involve a “per-flow” basis and it was realized that in large networks routers would be required to accept, maintain, and tear down a very big number of reservations. It was believed that this approach would not scale with the growth of the Internet, and in any event was antithetical to the notion of designing networks so that Core routers do little more than simply switch packets at the highest possible rates.

Differentiated Services is a Network Layer QoS mechanism that has been in limited use for many years, although there has been little effort to standardize it until very recently. DiffServ defines a field in the layer 3 header of IP packets, called the differentiated services code point (DSCP)<sup>1</sup>. Typically, hosts or routers sending traffic into a DiffServ network will mark each transmitted packet with the appropriate DSCP. Routers within the DiffServ network use the DSCP to classify packets and apply specific queuing or scheduling behaviour (known as a *per-hop behaviour* or *PHB*) based on the results of the classification. PHBs are individual behaviours applied at each router. They make no guarantees of end-to-end QoS, but this can be constructed by concatenating routers with the same PHBs.

### 2.1.3.2. QoS Implementation in IPv4

When Internet Protocol version 4 was first developed, an 8-bit field of the IP packet header was allocated to Type of Service (ToS). The original intention was for a sending host to specify a preference for how the datagram should be handled as it made its way through a network.

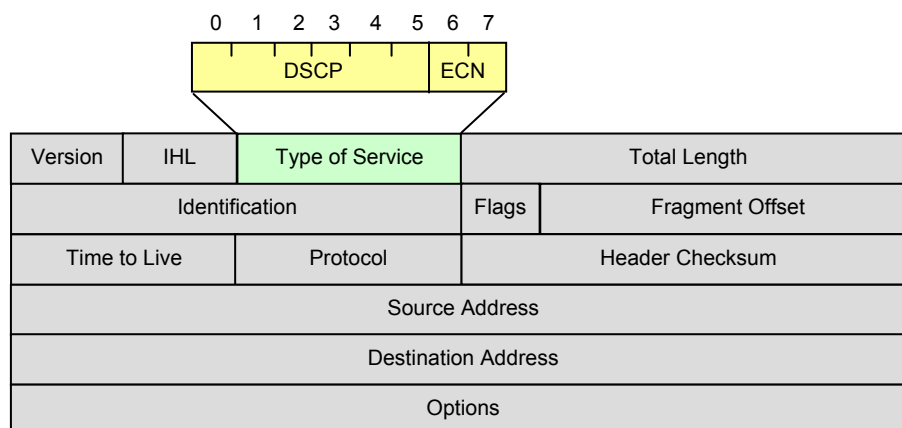


Figure 2.1. QoS facilities in IPv4

<sup>1</sup> Initially used as Type of Service (ToS) and IP precedence.

For instance, one host could set its IPv4 datagrams' ToS field value to prefer low delay, while another might prefer high reliability. In practice, the ToS field has not been widely implemented. However, a great deal of experimental, research and deployment work has focused on how to make use of these eight bits. Recently ToS has been redefined through DiffServ and Explicit Congestion Notification (ECN) (figure 2.1). The first 3 bits of the ToS field were formerly used for IP precedence, which indicated the importance of a packet. DSCP values are backward-compatible with IP precedence values, which means that legacy routers that support only IP precedence can interpret DSCP values.

### 2.1.3.3. QoS Implementation in IPv6

When Internet Protocol version 6 was initially developed, a 4-bit field called *Priority* was used for flow control. The priority values were divided into two ranges. Values 0 to 7 were used to specify the priority of traffic for which the source is providing congestion control. Values 8 to 15 were used for constant rate real-time traffic [6]. In the current standard [18] the *Traffic Class* field replaces the obsolete priority field and is extended to 8 bits. The first intention was to include DSCP in this field but the final document states that upper layers can set the values of this field and it can be modified by intermediary nodes. Also, the *Flow Label* field is used for QoS management in order to identify the packets of a flow that will receive special treatment in some hops (figure 2.2).

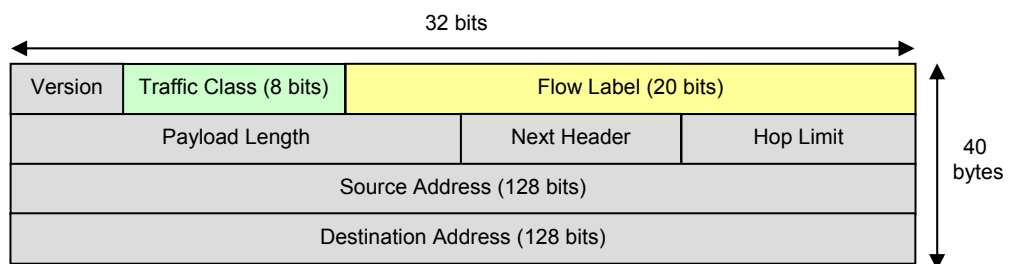


Figure 2.2. QoS implementation in IPv6

### 2.1.3.4. IEEE 802.1p

Traffic prioritization is also possible at Layer-2 but it is limited to the Local Area Network. IEEE 802.1p specification enables Layer 2 switches to prioritize traffic and perform dynamic multicast filtering. The prioritization specification works at the media access control (MAC) framing layer (OSI model layer 2). The 802.1p standard also offers provisions to filter multicast traffic to ensure it does not proliferate over layer 2-switched networks.

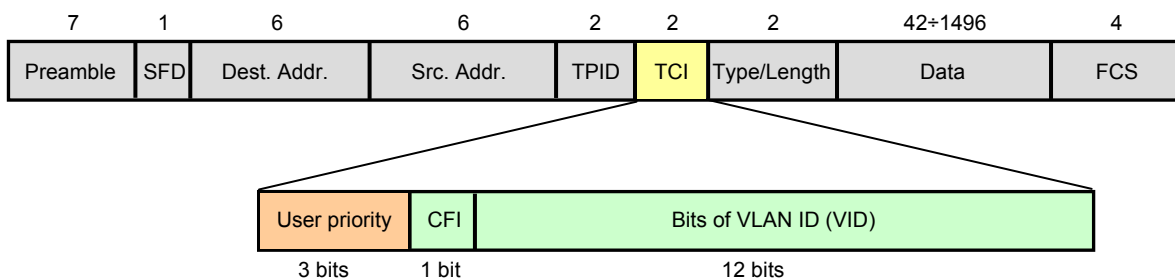


Figure 2.3. Layer 2 QoS implementation using IEEE 802.1p

The 802.1p header includes a three-bit field for prioritization, which allows packets to be grouped into various traffic classes (eight priority levels). The IEEE has made broad recommendations concerning how network managers can implement these traffic classes, but it stops short of mandating the use of its recommended traffic class definitions. It can also be defined as best-effort QoS (Quality of Service) or CoS (Class of Service) at Layer 2 and is implemented in network adapters and switches without involving any reservation setup. 802.1p traffic is simply classified and sent to the destination; no bandwidth reservations are established. The IEEE 802.1p is an extension of the IEEE 802.1q (VLAN tagging) standard and they work in tandem. The 802.1q standard specifies a tag that appends to an Ethernet MAC frame. The VLAN tag has two parts: The VLAN ID (12-bit) and Prioritization (3-bit). The prioritization field was not defined and used in the 802.1q VLAN standard. The 802.1p defines this prioritization field (figure 2.3).

### **2.1.3.5. Other Technologies Implementing QoS**

There are also a lot of other technologies that implement Quality of Service (ATM, ISSLOW, MPLS, Frame Relay, etc), some of them being actually designed for such purposes, but as the most deployed networks nowadays rely on Ethernet and IP, we have focused on explaining in detailed QoS mechanisms used with these network standards.

It is worth mentioning that technologies like Asynchronous Transfer Mode (ATM) guarantee QoS. ATM allows the accuracy and speed of data transfer to be specified by the client. This feature distinguishes ATM from other high-speed LAN technologies such as gigabit Ethernet. The QoS feature of ATM also supports time dependent (or isochronous) traffic. Traffic management at the hardware level ensures that quality service exists end-to-end. Each virtual circuit in an ATM network is unaffected by traffic on other virtual circuits. Small packet size and a simple header structure ensure that switching is done quickly and that delays due to high traffic are minimised.

## **2.2. NETWORK MEASUREMENT AND MONITORING**

### **2.2.1. The Purpose of Internet Measurements**

Network traffic measurement and monitoring has an essential role in next generation networks development. Multiple measurement technologies are currently under testing and new ones are being developed. Monitoring procedures are applied in routers, switches and other networking devices in order to compute statistics, characterize real-time flows or manage traffic. The researches in the field follow three main directions:

- New application development
- Network services and new protocols development
- Heterogeneous Internet network interconnection

The next generation networks are introducing adaptive applications, multimedia services, and intelligent distributed mobile applications. Consequently, there is a demand for efficient ISP capacity planning, security and inter-domain traffic engineering.

Internet measurement plays a key role in QoS provisioning and resource reservation, QoS based routing and network friendly multicast communication. New QoS service are developed in order to efficiently use TCP over high-speed links, support real-time and safety-critical applications or provide seamless mobile network interconnection.

Since packet data, mobile telecommunication, PSTN and wireless networks are merging, measurement helps enhancing the architecture of hardware and software of exist equipment, network dimensioning and solving transmission issues between different interface types.

### 2.2.2. Architecture of Measurement Systems

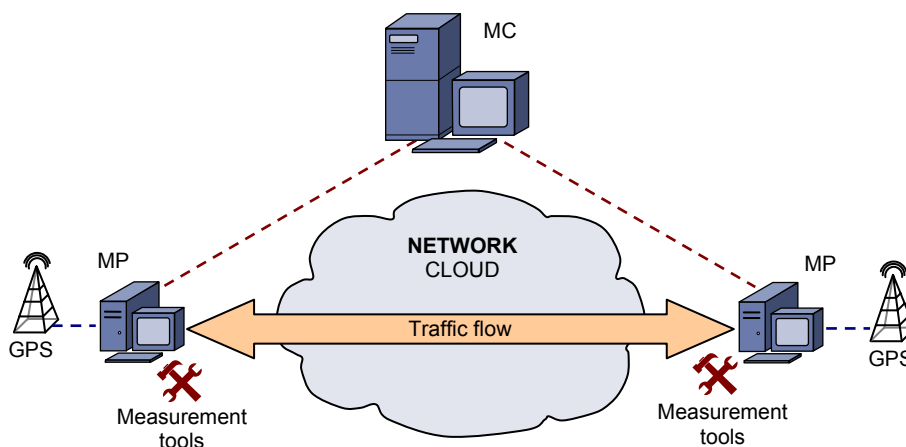
The basic architecture of a measurement and monitoring system is composed of three entities [19]:

1. The measurement points (MP)
2. The measurement tools
3. The measurement controller (MC)

The measurement points are the nodes in the network where the measurements are performed. When the networks tests involve the use of monitoring tools in different points of the network, it is essential that the measurement stations are closely synchronized in order to obtain a set of accurate results. In these nodes the QoS parameters of the network will be evaluated (OWD, packet delay variation, packet loss, etc.).

Most of the time for a complete evaluation of the performance of a link multiple tools need to be used, each specialized for different types of measurement. The project we have developed tries to combine the features of several such tools into a single application capable of computing numerous Internet metrics.

Another important element of a monitoring system is the measurement controller. This has the role of starting and stopping the tools installed on remote machines and afterwards collects the results.



**Figure 2.4.** Basic architecture of a Measurement and Monitoring System

Regarding the measurement requirements the distributed measurement architecture is modular designed to reach the following goals:

- to emulate end-user applications and to characterise the mapping of their end-to-end QoS requirements into network services in a quantitative way, i.e. measurement of performance metrics for generated single and aggregated flows.
- to maintain and store measurement information of different types (end-to-end performance, path metrics, network traffic, etc.) in order to be used for a mapping of the user end-to-end QoS to the network service requirements.
- to offer online access to measurement information, e.g. to enhance the network operation by the use of path performance metrics or packet drops in network elements.
- to provide time continuous performance analysis of measured end-to-end QoS of the network services with their mappings to traffic classes and resource reservations in order to enhance SLAs and to predict the performance of the network services.

### 2.2.3. Network Measurement Techniques

Quality of Service monitoring is usually divided in inter-domain and intra-domain. Network measurement can be performed in an active, passive or hybrid mode. Basically, passive measurements rely only on traffic already existing in the network while active measurements can be performed only when test traffic is generated. The two techniques will be explained in what follows.

#### 2.2.3.1. Passive Measurements

Passive measurements provide a statement about the treatment of the current traffic at the moment in the observed network section. Since no test traffic is generated, passive measurements can only be applied in cases where the kind of traffic we are interested in is already present in the network. This is the case for most applications where statements about the actual situation in the network is required (like SLA validation, traffic engineering).

Passive measurements mainly have been used for simple tasks like packet counting and associated metrics, like volume, so far. The area of packet filtering and classification is an active research field that permanently comes up with new fast algorithms and methods to improve the filtering performance. Passively measuring round-trip metrics is possible at a single measurement point by using packet pair matching techniques that rely on existing packet pairs like TCP-SYN/SYN-ACK, DNS request/response, etc. Nevertheless this method does only work for protocols based on packet pairs. It requires classification based on higher layer information and only measures round-trip metrics.

The advantages of this approach are the following:

- They do not perturb the network traffic
- They do not require cooperation or coordination from end hosts
- They are relatively simple to be implemented

Passive measurement techniques also have some drawbacks:

- Privacy issues when capturing packets from clients
- Difficulties in packet event correlation if packets are lost or duplicated
- The full load on the link is processed which can be problematic for high speed links

Among popular tools that perform passive measurements are CoralReef and NeTraMet. The first one is a software suite which can collect and analyse data from passive Internet traffic monitors in real time or from trace files. It is known to operate under most Unix-like OS. The package includes programming APIs for C and Perl and also applications for capturing, analysis and web report generation. The second one is a Network Traffic Flow Measurement Tools and is based on the Real Time Flow Measurement architecture, which is a general architecture for measuring traffic flows, developed by IETF. It builds up packet and byte counts for traffic flows, which are defined by their end-point addresses. These can be Ethernet addresses, protocol addresses (IP, IPX), port numbers. The traffic flows to be observed are specified by a set of rules, which are downloaded to NeTraMet by a 'manager' program. Traffic flow data is collected via SNMP from NeTraMet by a 'collector' program.

### 2.2.3.1. Active Measurements

Active measurements inject test traffic into the network in order to measure network characteristics. The development of active measurement methods to survey large parts of the Internet with a high precision is an active field of research. Active measurements give a prediction on how traffic would be treated in the observed part of the network. They are controllable experiments that can be performed at any time and with any kind of traffic that is of interest for the specific measurement objective. Nevertheless active measurements are based on the sending of additional test traffic through the network under test. This test traffic always generates additional load on network links and routers and can significantly influence the measurement results. It may lead to additional costs if accounting is usage-based and it might bother intermediate providers. Especially if test traffic is not recognizable as such, providers may suspect an attack.

Active measurements can provide all the means for determining all the QoS parameters of a link. Despite that some problems may be encountered. For example round-trip time is typically computed by generating ICMP echo request messages. Some nodes may implement ICMP packet filtering for security reasons which will make the delay estimation impossible.

When measuring one-way delay two measurement points are involved. In order to obtain accurate results the two nodes involved in the measurement process need to be synchronized. Synchronisation can be obtained using global positioning system, network time protocol or radio signals (e.g. DCF77). Synchronisation issues will be discussed in the next paragraph.

The software we have developed implements active measurement techniques. The QoS parameters are obtained by adding a stamp at transmission that will be used by the receiving end to calculate the one way delay, packet delay variation, throughput and number of out of order packets.

Some of the most popular active measurement tools are *ping* and NetMeter. Ping relies on ICMP echo request and echo reply messages in order to compute the round trip time. The approach is not very precise as there is no guarantee that the other end will reply immediately after receiving the request. In case the remote host is behind a firewall it is possible that the echo request will not reach the destination and the measurement cannot be performed.

NetMeter relies on another tool called MGEN which is able to generate UDP datagrams and different rates, with different sizes and different distributions (periodic or Poisson distributed).

NetMeter will analyze the packets generated by MGEN and will calculate the one-way delay, instantaneous packet delay variation, throughput and packet losses. The UDP datagrams can be encapsulated in IPv4 or IPv6 packets and the obtained results can be easily plotted.

## 2.3. SYNCHRONISATION TECHNIQUES

### 2.3.1. Network Time Protocol

The Network Time Protocol (NTP) is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks. NTP uses UDP port 123 as its transport layer. It is designed particularly to resist the effects of variable latency. NTP is specifically designed to maintain accuracy and robustness, even when used over typical Internet paths involving multiple gateways, highly dispersive delays and unreliable nets. Current NTP Version 3 has been in use since 1992, with nominal accuracy in the low milliseconds [21].

NTP is organized in a hierarchical client-server model. In the top of this hierarchy there are a small number of machines known as *reference clocks*. A reference clock is known as *stratum 0* and is typically a caesium clock or a Global Positioning System (GPS) that receives time from satellites. To these machines are attached the so-called *stratum 1* servers (that is, stratum 0 clients), which are the top level time servers available to the Internet, meaning that they are the best NTP servers available.

Following this hierarchy, the next level in the structure are the *stratum 2* servers which in turn are the clients for *stratum 1* servers. The lowest level of the hierarchy is made up by *stratum 16* servers. Generally speaking, every server synchronized with a *stratum n* server is termed as being at *stratum n+1* level. A server may broadcast time to a broadcast or multicast IP addresses and clients may be configured to synchronize to these broadcast time signals.

NTP is designed to produce clock offset, round trip delay and dispersion, all relative to a selected reference clock. Clock offset represents the amount to adjust the local clock to bring it into correspondence with the reference clock. Roundtrip delay provides the capability to launch a message to arrive at the reference clock at a specified time. Dispersion represents the maximum error of the local clock relative to the reference clock. Since most host time servers will synchronize via another peer time server, there are two components in each of these three products, those determined by the peer relative to the primary reference source of standard time and those measured by the host relative to the peer. Each of these components is maintained separately in the protocol in order to facilitate error control and management of the subnet itself. They provide not only precision measurements of offset and delay, but also definitive maximum error bounds, so that the user interface can determine not only the time, but the quality of the time as well.

The NTP Version 4 architecture, protocol and algorithms have been evolved to achieve accuracy of the order of microseconds. The new version of the protocol provides:

- improved clock models which accurately predict the time and frequency adjustment for each synchronization source and network path.
- engineered algorithms which reduce the impact of network jitter and oscillator wander while speeding up initial convergence.

- redesigned clock discipline algorithm which operates in frequency-lock, phase-lock and hybrid modes.

The new modifications, confirmed by simulation, improve accuracy by about a factor of ten, while allowing operation at much longer poll intervals without significant reduction in accuracy.

On UNIX-like systems a NTP daemon has been implemented which will operate by exchanging packets (time and sanity check exchanges) with its configured servers at poll intervals and its behaviour will depend on the delay between the local time and its reference servers. Basically, the process starts when the NTP client sends a packet containing its timestamp to a server. When the server receives such a packet, it will in turn store its own timestamp and a transmit timestamp into the packet and send it back to the client. When the client receives the packet it will log its receipt time in order to estimate the travelling time of the packet.

The packet exchange takes place until a NTP server is accepted as a synchronization source, which take about five minutes. The NTP daemon tries to adjust the clock in small steps and will continue until the client gets the accurate time. If the delay between both the server and client is big enough, the daemon will terminate and you will need to adjust the time manually and start the daemon again.

### **2.3.2. Global Positioning System**

The Global Positioning System (GPS) is the only fully-functional satellite navigation system. A constellation of more than two dozen GPS satellites broadcasts precise timing signals by radio to GPS receivers, allowing them to accurately determine their location (longitude, latitude, and altitude) in any weather, day or night, anywhere on Earth. GPS also provides an extremely precise time reference, required for telecommunications and some scientific research.

Many systems that must be closely synchronized use GPS as a source of accurate time. For instance, the GPS can be used as a reference clock for time code generators or NTP clocks. Also, when deploying sensors (for seismology or other monitoring application), GPS may be used to provide each recording apparatus with a precise time source, so that the time of events may be recorded accurately. Communications networks often rely on this precise timing to synchronise RF generating equipment, network equipment and multiplexers. Without this precise timing, synchronisation timing errors would be substantial causing a high BER (bit error rate) and or unstable TX/RX frequencies resulting in sporadic signal loss which is known as "timing jitter". Multiplexers might loose whole frames of data which could cause dropped calls.

The atomic clocks on the satellites are set to "GPS time". GPS time is counted in days, hours, minutes and seconds, in the manner that is conventional for time standards based on Earth rotation. GPS time is not connected to the rotation of the Earth at all. A GPS day is, as close as can be achieved, 86400 SI seconds of International Atomic Time (TAI), the principal realisation of Terrestrial Time (TT). GPS time was set to read the same as Coordinated Universal Time (UTC) in 1980, but because GPS time does not include leap seconds it has since diverged from UTC. GPS time thus maintains a constant offset but not perfectly. The GPS clocks are steered to keep as close as possible (within 20 ns).

The GPS day is identified in the GPS signals using a week number along with a day-of-week number. GPS week zero started at 00:00:00 UTC (00:00:19 TAI) on January 6, 1980. The week number is transmitted in a ten-bit field, and so wraps round every 1024 weeks (7168 days). The transmitted week number returned to zero at 00:00:19 TAI on August 22, 1999 (23:59:47 UTC on August 21, 1999). GPS receivers thus need to know the time to within 3584 days in order to correctly interpret the GPS time signal. A new field is being added to the GPS navigation message that supplies the calendar year number in a sixteen-bit field, thus performing this disambiguation for any receivers that know about the new field.

The main advantage of the global positioning system is that it is free for use and provides nanosecond accuracy synchronisation signal. The drawback on the other hand consists of the fact that the equipment required for synchronising telecommunication devices is relatively expensive.

### 2.3.2. Radio Signals

Though not very used, another solution for close synchronization of equipment relies on radio signals. The most popular is DCF77 which is available in Europe. DCF77 is a long-wave time signal and standard-frequency radio station. Its primary and backup transmitters are located in Mainflingen, about 25 km south-east of Frankfurt, Germany. It is operated by *T-Systems Media Broadcast*, a subsidiary of Deutsche Telekom AG, on behalf of the Physikalisch-Technische Bundesanstalt, Germany's national physics laboratory. DCF77 has been in service as a standard frequency station since 1959, date and time information being added in 1973.

The 77.5 kHz carrier signal is generated from local atomic clocks that are linked with the German master clocks in Braunschweig. The signal from the 50 kW transmitter can be received in large parts of Europe, as far as 2000 km from Frankfurt. Its signal carries an amplitude-modulated, pulse-width coded 1 bit/s data signal. The same data signal is also phase modulated onto the carrier using a 511-bit long pseudorandom sequence (direct-sequence spread spectrum modulation). The transmitted data repeats each minute.

# 3

## DESIGN AND EXPERIMENTAL RESULTS

### 3.1. MOTIVATION

There have been a relatively large number of projects and research papers on Internet measurement and monitoring in the past years. The main reason is that nobody expected Internet to grow so fast and by the time the first networks were deployed, there was no concern about guaranteeing a minimum level of certain parameters in order to ensure optimal performances of network applications.

There are a variety of tools available at present which could help a network traffic engineer to perform measurements, validate some parameters and perform changes when required. The reason why we decided to develop another network measurement system is that most of the tools we have tested so far seem to lack a complete functionality. Some of them rely perform analysis by relying on traffic generators developed by different research groups, which limits the overall control of the measurement process (e.g. NetMeter works with MGEN or NetPerf). The most used generators need a script in order to perform the required task. This is not very convenient as the user needs to familiarize first with the scripting language used and then needs to deploy the script file on each machine at the end of the links that need to be tested. Other tools rely on protocols that sometimes are filtered because of security reasons (e.g. Windows PING uses ICMP<sup>2</sup> messages). In other cases the results are hard to be interpreted as the tools lack a graphical user interface. Or, on the contrary, they only provide graphical interfaces, thus leading to difficulties in the remote configuration process. Another lack in some measurement tools is the option of selecting the certain layer from whose point of view the tests should be performed. This is useful as it is known that the implementation of TCP/IP on computer operating system adds certain delays. Another feature missing from the majority of the tools developed so far is the capability of processing information from high performance hardware monitoring cards, such as the Endace DAG cards. These cards allow full link monitoring for Gigabit and even 10 Gigabit connections. The manufacturer provides tools for capturing that but those tools actually dump the capture data on the drive and this needs to be inspected offline. The disadvantage of this approach is that for longer tests it is impossible to save all the data in real time, as the hard-drive writing speed is bellow the link speed. Nevertheless, what most of the network measurement systems fail to provide is a standard communication protocol between the central node that collects the data and the probes that perform traffic generation or capturing.

By developing this new Network Measurement System we tried to overcome the disadvantages presented above and to make the measuring process as easy as possible for an engineer. From the

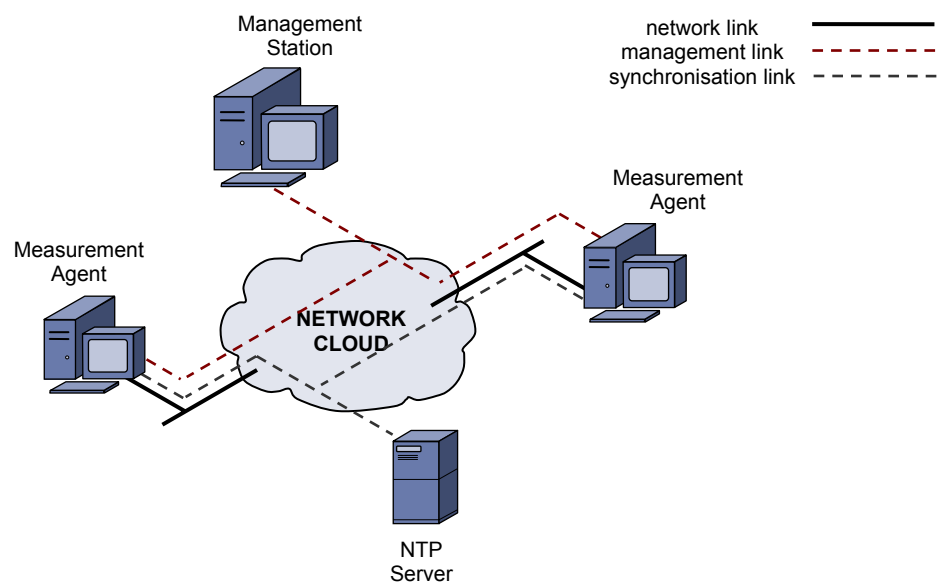
---

<sup>2</sup> ICMP stands for Internet Control Message Protocol. It is part of the Internet protocol suite and it is used for diagnostic or routing purposes. More details can be found in RFC 792.

point of view of the distributed Measurement Agents described in this paper, the main concerns were to provide powerful traffic generation and capturing capabilities, to permit running tests at different layers of the OSI<sup>3</sup> reference model, to allow saving data for offline analysis and to optimize the capabilities of an Endace card software tool for performing high speed detailed measurements. The agents also permit local configuration and status inspection through a friendly user interfaces, but at the same time a user can remotely connect to a machine that needs to run tests, start an agent in text console mode, configure and inspect the settings through a set of defined commands. Nevertheless, the agents are remotely controlled in the measurement process by a Measurement Manager which can start multiple measurement sessions on different machine and collect data in real-time or after the test has been completed. The communication between all entities of the Network Measurement System is done obeying a standard management protocol, SNMPv1<sup>4</sup>, using a predefined set of management objects. Simultaneously, this approach offers a certain level of security when transferring the results or commands through the network and also high reliability degree, through the use of the message retransmission mechanism in case of response time-outs.

### 3.2. APPLICATION DESIGN

In what follows the paper will try to present in depth the features of the Distributed Agents for the Network Measurement System, the system requirements for running such an application and the main concerns and problems encountered in the implementation process. It will be explained how the software can be configured and used for accurate measurements, what are the key services running in background to provide reliable traffic generation and analysis, their detailed functioning and nevertheless what are the difficulties and important matters to be taken into account when designing a measurement agent.



**Figure 3.1.** Basic scenario for using the Network Measurement System

<sup>3</sup> Open System Interconnection is the standard established by the International Standardisation Organisation which defines a common network standard that ensures interoperability between different manufacturer devices.

<sup>4</sup> Simple Network Management Protocol is an application layer protocol used by network management systems for monitoring and control network-attached devices.

In order to perform some measurements of the QoS parameters inside a network, the minimum requirements that need to be fulfilled when using the Network Measurement System is to have a manager running on one machine, and at least to measurement agents running on two computers, preferable placed at the ends of the link that needs to be tested. Of course, in order to ensure high precision of the measurement results, the internal clocks of the measurement stations need to be synchronised to a common clock provided by a dedicated synchronisation system (GPS or NTP server). A basic scenario of using this measurement system is shown in figure 3.1.

### 3.2.1. Software Architecture of the Network Measurement Agent

The Distributed Agents for a Network Measurement Systems were designed to operate on computers running a GNU/Linux type operation system. The Network Measurement Agent (NMA) has been developed on Debian GNU/Linux kernel 2.6.15-1-486 and has been deployed and tested on machines running Debian and Kubuntu distributions of Linux. There are several reasons for choosing GNU/Linux as development platform: it is an open-source operating system, it allows secure connections from remote machine through SSH<sup>5</sup> and is considered to be safer and more reliable from the networking point of view. The programming language used to implement the software was C/C++.

The minimal hardware configuration of a machine running a measurement agent depends on the type of link one wants to monitor. In case of links up to Fast Ethernet speed the requirements are not very big. The hardware configuration plays a key role when high speed links are to be analysed. So far, we have performed measurements on Gigabit Ethernet links and it turns out that the better the performances of the machine running the software, the more precise the results will be. Actually, there is a set of hardware factors that influence the performance of an agent, but they will be depicted later on.

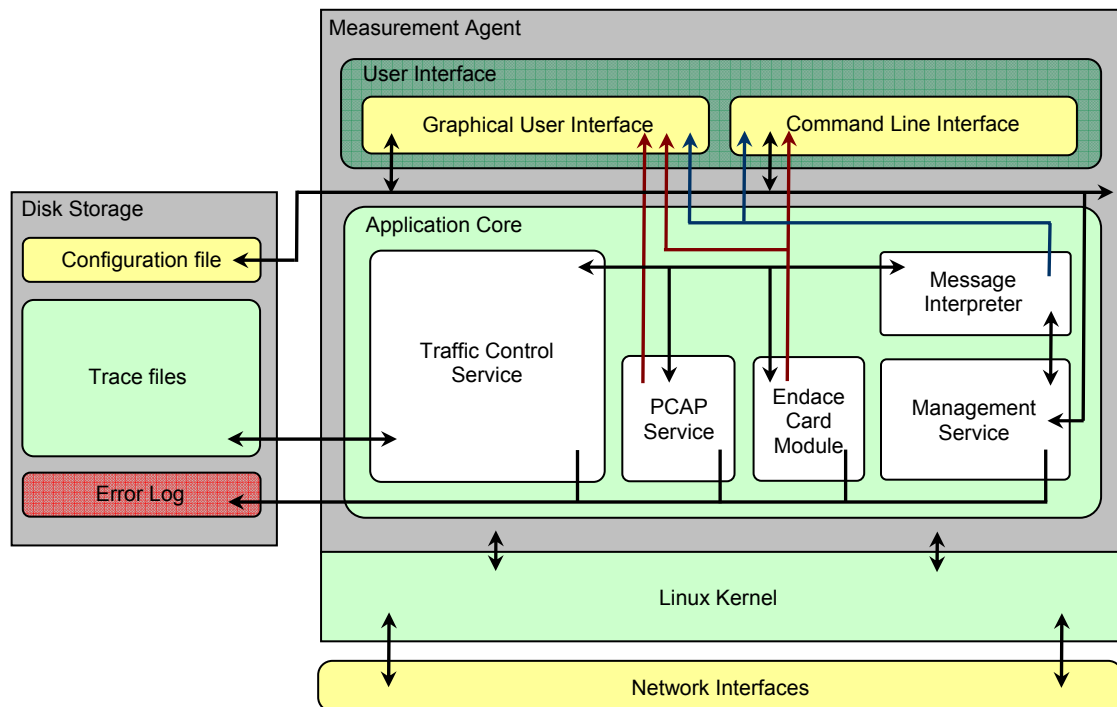
The application consists of a two option user interface and a set of services. Those services control the traffic generation and capturing processes, ensure the communication with the managers, inform the user about the state of the network and of the processes running and can offer interoperability with a high performance network monitoring card (Endace DAG card). A block diagram of the network measurement agent software is presented in figure 3.2.

The user is able to choose between starting up the application in graphical mode or in text-console mode. The first option makes the configuration process easier and faster and also enables the inspection of the network activity on the links seen by the available Ethernet controllers. But in some cases it is rather important to be able to remotely connect to a machine through a secure shell, start and configure the agent in text mode and ensure the same functionality from the measurement process point of view.

Essential information is stored in a configuration file, which helps saving the last used settings. This is important in order to ensure that certain vital parameters are maintained even in case of a power failure or a simple application restart. In the same time, this file helps detecting if the system configuration has changed since the last utilisation of the application. Consequently, the user will be required to reconfigure the software.

---

<sup>5</sup> SSH (Secure Shell) represents a set of standards that allow establishing a secure connection between two remote computers. It uses encryption to ensure confidentiality and integrity of the data transferred over the network.



**Figure 3.2.** The architecture of the Measurement Agent application

There is also other data stored on the disk, as the error log, which is used by all the services for error notification and which can help one finding the cause of a malfunctioning in order to re-establish the optimal working parameters. Besides those, when a manager requests this, the agent can store trace file on the hard drive and later perform an offline analysis on them.

The application core is a multithreaded piece of software that provides higher priority to real-time demanding tasks such as traffic generation sessions, notifications to corresponding modules when appropriated messages are received from a manager, efficient use of memory resources and communication with the user interface. There are four main services running behind a measurement agent. The most important is the traffic control thread which is responsible for initializing, maintaining and finishing measurement sessions. The second one is the PCAP<sup>6</sup> service that enables full link monitoring on an Ethernet link. Another feature is the tool used for performing measurement with the Endace DAG card. Nevertheless, an essential part of the application is the management service that offers control of the agent to the manager through the message interpreter module. The message interpreter actually commands the measurement sessions and gathers up results and agent related information whenever a manager requests it through the management service. In what follows all of those blocks that build up the measurement agent software will be presented in detail.

It is important to have in mind that the functioning of the application presented in this paper strictly depends on the work presented in [5]. Without a network manager, an engineer willing to perform Internet measurements will not be able to control traffic generation/capturing processes and to collect the results only by using distributed agents.

<sup>6</sup> Packet Capture Library (PCAP) is an application programming interface for advanced packet capturing that is currently used by a lot of network tools, such as Ethereal, tcpdump, Snort etc.

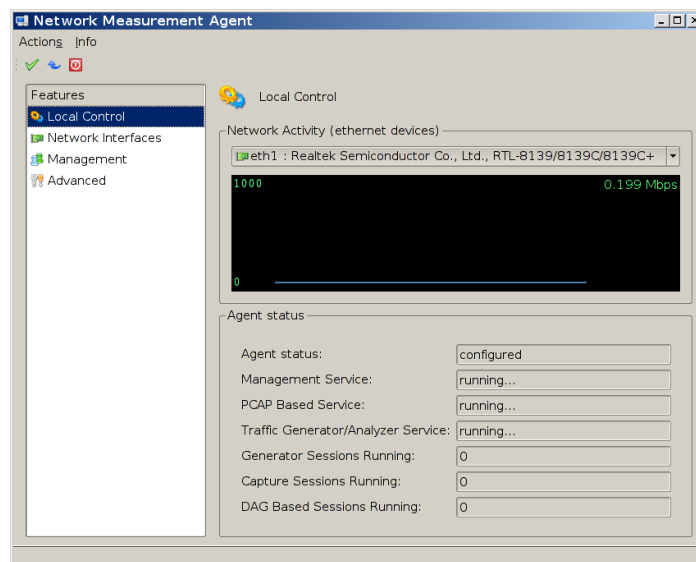
### 3.2.2. The Graphical User Interface

The graphical user interface of the application was designed using the open source edition of the Qt3 integrated development environment (IDE). Qt is a multiplatform C++ toolkit created by the Norwegian company Trolltech and is available for Unix/X11<sup>7</sup>, Macintosh and Embedded Linux. The essential benefit of using this technology is that it makes the application easy to use even for engineers that are not familiar with Linux-based operating systems. We will focus on presenting the features offered by the GUI and their full features will be described when showing how application can be used and configured.

There are four essential things that can be accessed through the graphical Interface:

- The status of the agent and the running services,
- The information and behaviour of the available network interfaces,
- Management related information,
- The error log.

As mentioned before, the user interfaces have direct access to a set of global variables, so that one can see at any moment the number of traffic generation/capturing sessions, the bandwidth of a link and the state of the background services. In case some modifications occur, the graphical interfaces will be appropriately updated. There has been defined a special function, `updateGUI()`, that performs all the necessary modification when invoked. Information about the installed networking devices is collected at application start-up, consequently the properties of each NIC can be viewed and the user can select for what purpose(s) that particular card should be used. The most consistent part of the interfaces is related to management issues. One can configure the description of the measurement station, the port settings for devices used for management, the list of community names and eventually a list of allowed/banned IP addresses.



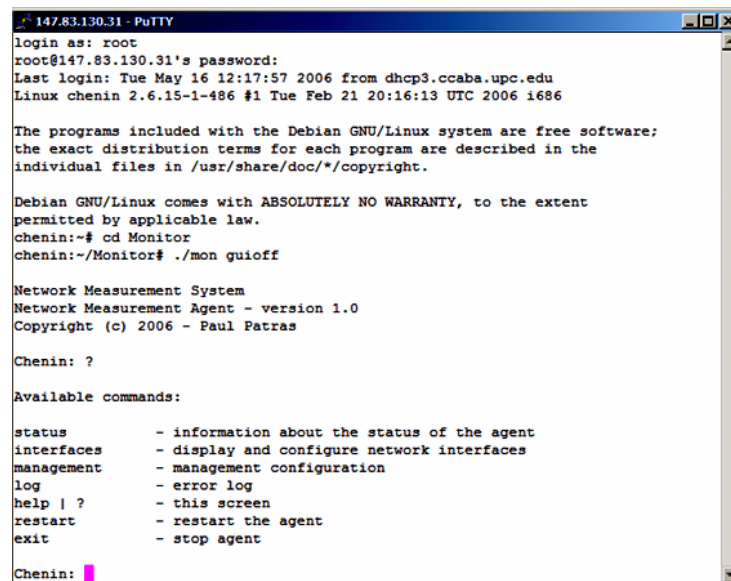
**Figure 3.3.** A snapshot of the Network Measurement Agent GUI

Once a registration/detachment process has been performed, a list of available managers is also updated. When modifying settings, it is possible to restore the initial configuration or save the new options in the configuration file.

<sup>7</sup> X11 is the abbreviation used for X Window System, which provides the standard framework for developing graphical applications for Unix-like systems.

### 3.2.3. The Command Line Interface

As mentioned before, the purpose of the CLI is to enable remote connection to a measurement station, measurement software start-up, configuration and monitoring. In order to start the application in this mode, one needs to pass `guiouff` as a command line argument when executing the application. In case a valid configuration file is found, a prompter containing the name of the measurement agent will be displayed and the user can start using a set of predefined commands. By typing `'help'` or `'?'` the list of available options will be displayed. In figure 3.4 it is shown an example of how one can remotely connect from a machine running Windows OS to a Linux machine, start the application in text-mode and find about the main list of available choices.



```

147.83.130.31 - PuTTY
login as: root
root@147.83.130.31's password:
Last login: Tue May 16 12:17:57 2006 from dhcp3.ccaba.upc.edu
Linux chenin 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.
chenin:~# cd Monitor
chenin:~/Monitor# ./mon guiouff

Network Measurement System
Network Measurement Agent - version 1.0
Copyright (c) 2006 - Paul Patras

Chenin: ?

Available commands:
status          - information about the status of the agent
interfaces      - display and configure network interfaces
management     - management configuration
log            - error log
help | ?       - this screen
restart        - restart the agent
exit          - stop agent

Chenin: █

```

Figure 3.4. Example of remote start-up of the Network Measurement Agent

From this moment on the agent is available for measurement sessions. There are certain features that may not be available if the application is executed by a user who has no administrative permissions on the machine running the application. Such features are the real-time scheduling policy for traffic generation/capturing threads. Actually this issue also holds if the application is started under the same circumstances in graphical mode. If certain operations that need privileged rights are to be executed, appropriate error messages will be appended to the error log. For best performances it is recommended to run the application in super-user mode.

By typing the `status` command it is possible to find out whether the measurement agent is appropriately configured, what is the state of the main services and what kind of sessions are running. When choosing the `interfaces` option, the user will be provided with another set of commands that will enable him to inspect the features of the NIC available on the system and configure their behaviour, i.e. specify whether they should be used or not for management, traffic generation or traffic capturing purposes. For devices such as Endace cards, those options are limited because these types of interfaces cannot be assigned logical addresses, and traffic generation is not available for most of the models. Later details about the use of Endace DAG cards will be explained in a future paragraph. Another important thing to have in mind is that when modifying the behaviour of an interface from the point of view of the management service, the application needs to be restarted, in order for the changes to be applied.

The `management` section offers a group of sub-menus and a variety of commands. It is possible to view and modify information related to the description of the agent (name, location,

description, contact). This is used by managers to identify each management station. Once a manager successfully adds a new agent to its list, it will send similar information to the agent. The received data will be stored locally and will be available through the `managers` command. The ports (polling and trap port) used for communication with managers can be set up for each management enabled interface after accessing the `net` menu. It is also possible to check for each of those interfaces the IP address and the already assigned port values.

In order to perform modifications upon the list of community names and their rights (add/edit/remove) it is necessary to access the `snmp` menu. Having an appropriate community list is essential for the communication process. One manager cannot control the agent if it doesn't include the appropriate name inside the management message. A manager sending message with invalid community rights will be replied appropriately. Moreover, different community names could be used for communicating with different managers. This also ensures security, as unauthorized network user will not be able to ask an agent to perform measurements, or to request results and configuration information. When adding a new community name to the list, first the software verifies whether this is already known, and in case of a matching, the engineer performing the configuration will be notified and asked whether he wants to cancel or modify the existing entry. Every operation is followed by an update of the configuration file.

The last option of the management configuration/inspection feature refers to a list of allowed/restricted IP addresses (`filter`). This is another security option that ensures that only messages coming from known locations are to be interpreted. The behaviour can be set to allow messages from all the logical addresses in the list or to allow all messages except the ones from the addresses specified in the list, using the `mode` command. As for all the other management related features, to guarantee that the changes made take effect, it is suggested to restart the application. For convenience, the main menu offers the `restart` command, which automatically closes all the services and restarts the application with all the changes enabled.

For debugging, an error log is created and updated every time a service fails to perform an operation. Possible errors could appear when asking a measurement agent to start a new UDP session using the same IP address and the same port (bind error), failing to deliver packets at the established moments (real-time failures), failing to load the configuration file due to hardware changes, missing file or file corruption, running out of virtual memory, trying to open a busy Endace card, etc. The error log can be inspected and eventually cleared by accessing the `log` menu. All the entries in the log keep the date and time of the encountered error, the error code (the set of error codes was defined by the developer), the process that caused the error and also a brief description which helps the debugger to understand faster what could have been the problem.

When navigating through a certain section of the measurement agent console, at any point, the list of valid commands applicable to that menu can be viewed by typing `?`. Also, commands that require specific parameters are provided with a `usage` model, i.e. in case it is incorrectly invoked, the user will be notified about the correct syntax, the meaning and appropriate values of each parameter. It is always possible to return to the previous menu using the `back` command. An example of using all the available commands will be presented in section 3.2.10.

### 3.2.4. Application Initialisation Steps

When the Network Measurement Agent application is started there are several important instructions that are executed. The first thing is to check whether a command line argument has been passed or not. The only accepted argument is `guioff`. In case something else is given as parameter the application will fail to start. The previously mentioned option indicates that the agent should run in text mode. If no arguments are passed (default) the software will run in graphical mode. The operating mode is kept inside a global variable and will be used by other processes in order to know whether an update of the GUI is necessary after some changes.

Then the `main` function invokes a function named `StartUp()`. Initially, this function will inspect the system and gather information about all the enabled Ethernet controllers. It is important to know that a NIC may be correctly installed from the hardware point of view but unavailable, since it was not activated by the operating system. This kind of situation may occur when the `/etc/network/interfaces` file of the Linux system is not correctly configured, the interface has not been assigned a valid IP address or it is simply down. The information related to Ethernet interfaces is obtained using a set of functions included in the `netapi.h` and `pciapi.h`. These libraries are modified versions of open-source Linux system utilities that allow requesting low level information from the kernel about the installed devices. More details can be found in [5]. To be more precise, first we obtain the total number of interfaces and then select the ones that are not loop-back, are up and running and are of Ethernet type. For those we store their identifiers, hardware (MAC) addresses, IP addresses, network masks, MTU<sup>8</sup>s, base addresses and IRQs. Then a scan is performed through all the PCI devices until the device that matches the base address and IRQ with a certain interface. When this is found, information related to vendor, model and type is acquired and stored in the list of available network interfaces. Also, by default each device is supposed to have the values 161 and 162 for polling and trap ports. The process is somehow similarly performed for detecting Endace cards. A correctly installed DAG card is identified by the system through a string of the form `/dev/dagx` where `x` is usually a value between 0 and 3. A function named `check_dag()` is used to scan for this type of network devices. If the hardware can be opened and the device class and vendor ID match the ones used by Endace, the complete specifications of the card will be stored. As it will be explained later, a DAG card does not have a MAC address and cannot be assigned any logical addresses. The main purpose of those cards is full link monitoring at high speeds. Some of them also support high speed traffic generation for testing purposes, but this feature will not be presented in this paper, as the card we have used for measurements was not tested by the manufacturer for traffic generation. Consequently, the Network Measurement Agent will never use an Endace card for management or traffic generation.

Once the software knows everything about the system resources from the networking point of view, it will pass to loading the configuration file (`config.bin`). This is handled by a function named `LoadConfiguration()`. If any error is encountered while processing this file the function will return a negative value, and in case the GUI is enabled, the user will be noticed by a dialog window. Suppose the file is found, the function will compare the information related to NICs. If the current configuration does not match the one stored in the file, it is considered that either hardware changes have been made or the computer administrator performed some configuration changes since the last time the application was executed. In this case a reconfiguration of the networking devices is required. If this stage was successfully passed the software tries to load information related to agent description, community names and list of allowed/restricted addresses. If the configuration is successfully loaded, a global variable will be modified in order

---

<sup>8</sup> Maximum Transmission Unit refers to the maximum number of bytes that a NIC can pass on a link. The typical value for an Ethernet Controller is 1500 Bytes.

to ensure that the other services are aware of the status of the agent (configured or not). The next step is to start-up the management and the traffic control services. An in depth presentation of the management service can be found in [5] and the traffic control service will be explained later in this document.

The last step that is performed before launching the graphical interface or the text console is related to the PCAP service. This service will try to run regardless the state of the agent (configured or not) so that in case the GUI was started, one can inspect network activity on the links connected to the Ethernet controllers. First the initialisation function tries to allocate some memory for keeping a history of the throughput on each link for 20 seconds. These values will be used for sending reports to a manager and plotting network history. Then a separate thread will be created for handling all data acquired from Ethernet cards through the packet capture library API. The thesis presents in detail this service in 3.2.6.

If the application was started in graphical mode, there are also a few things to be performed before launching the window. The class that the main window is about to instantiate has itself a method named `startup()`. This method first calls several functions in external libraries in order to pass a pointer to the main form to all the processes that would need to update the GUI. Then the list view of the main features of the graphical interface is dynamically build, a *TabWidget* containing NIC related information created and if something went wrong during the loading of the configuration, an error window will be displayed. Assuming no errors occurred so far, whether the agent has been started in graphical or text mode, it should be fully available for communicating with remote managers and ready to start traffic generation/capture sessions.

## 3.2.5. The Traffic Control Service

### 3.2.5.1. Main Control Thread and Socket Threads

In this section the paper will focus on presenting the most important part of the Network Measurement Agent – the traffic control service. In essence, this service is an independent thread that when started prepares all the necessary data structures for future measurement sessions and enters a loop which it will exit only upon closing or restarting the application. While inside its main loop, it will check every 200 milliseconds to see whether a manager has requested starting up a new traffic generation/capturing process. This is signalled through a global variable which the message dispatcher modifies after having all the necessary parameters for the session to be created. To ensure synchronization between the traffic control thread and the network management thread, a *lock* variable has been defined so that none of them processes information related to traffic parameters while the other one uses this data, but waits for that process to finish its job.

There is a set of arrays that store information about all the traffic threads running, as a separate thread will be created for each new flow. When the traffic control thread is notified about a new session, it will find the first index for which the arrays that mark the execution state of traffic threads (running or not) are zero, i.e. there is no thread that currently uses that index for control. Another condition that leads to deciding whether a certain index could be used or not for identifying the new thread is to have a zero value at that position in the 'lpRead' array. This marks whether the manager has finished collecting results after the execution of a thread has ended. The first entry found in those three vectors (`lpThreadsRunning`, `lpThreadsFlag`, and `lpRead`) will be considered the ID of the new traffic thread and shall be used for later addressing

of the information related to that particular traffic flow. After copying the parameters of the new traffic flow and marking the newly assigned indexes as 'busy', the main control thread will create a new thread for the new process, clear up the memory where the later will store its data and wait for a new command.

Assuming a traffic thread has been created as a result to a request from a manager, its associated routine (`SocketThreadFunction`) will receive a pointer to a memory location where a structure that holds the parameters of the new sessions is stored and begin the execution. This routine performs a few essential operations:

- Creates a child thread that handle the generation/capture process
- Sets up a real-time scheduling attribute to the child thread
- Opens an appropriate socket depending on the protocol involved in measurement
- Passes appropriate session requirements
- 'Unlocks' the memory zone where parameters for new sessions are placed
- Ensures that the new session terminates as scheduled and closes sockets

The main concern of the measurement agent is to perform traffic analysis and generation as accurate as possible, trying to obey as much as possible a given time distribution of the packets. Hence, the attributes of the thread routine performing these tasks will need to have a very high execution priority. The POSIX<sup>9</sup> thread libraries offer a set of facilities than allow the programmer to set a real-time scheduling policy to certain threads, assuming the parent process runs in super-user mode. There were several policies defined, the last mentioned one having the highest priority. The code sequence that sets up the attributes of a thread is shown below. In this example a new thread is created with real-time attributes, having as start routine the function that will generate Ethernet frame according to the parameters stored in the structure passed as argument (`llstTraf`).

```
THREAD hComThread;
pthread_attr_t attr;

pthread_attr_init (&attr);
pthread_attr_setschedpolicy(&attr, SCHED_RR);
.....
iRet = pthread_create(&hComThread, &attr, L2SendFunction, (void *) &llstTraf);
```

For each new session an end point for communication is created (socket). This allows sending/receiving data on a network interface. When manager calls for a new session, a custom message containing the operation type (send/receive) and the protocol (Ethernet/Internet Protocol version 4/UDP) to be used will be passed. The type of socket to be opened highly depends of those two choices. Linux provides the `socket` function that takes as parameter, the communications domain in which a socket is to be created, the type of socket and a particular protocol to be used and returns a descriptor which will be further used for sending/receiving data. For exchanging Ethernet frames the packet interfaces was used (`PF_PACKET`) and the socket type is defined as `SOCK_RAW`, which means direct access to the device driver is ensured (OSI Layer 2). For IP packets `SOCK_RAW` is used but the family is set as `AF_INET`, the protocol type is set as required by the manager in order to appropriately inform the operating system what value should be encapsulated in the protocol field of the IP header. An example of preparing a descriptor for an IP send session is presented below. Errors are also treated.

<sup>9</sup> POSIX (Portable Operating System Interfaces for uniX) is a set o standards that define the API for software compatible with variants of the Unix OS. The standard threading library defined by POSIX is supported by most of the modern operating systems.

```

sSock = socket (AF_INET, SOCK_RAW, ipstTraf.bProtocol);
if (sSock == -1)
{
    switch(errno) {
        case EPROTONOSUPPORT:    ReportError(0x00020001, 0x0001); break;
        case EAFNOSUPPORT:      ReportError(0x00020002, 0x0001); break;
        case ENFILE:            ReportError(0x00020003, 0x0001); break;
        case EMFILE:            ReportError(0x00020004, 0x0001); break;
        case EACCES:            ReportError(0x00020005, 0x0001); break;
        case ENOBUFS:           ReportError(0x00020006, 0x0001); break;
        case ENOMEM:            ReportError(0x00020006, 0x0001); break;
        case EINVAL:            ReportError(0x00020007, 0x0001); break;
        default :                ReportError(0x00020008, 0x0001);
    }
}

```

When dealing with UDP sessions a high level approach is used, relying on the TCP/IP protocol stack implemented by the operating system. The reason for choosing this option rather than manually encapsulate all the payload of a datagram and pass it directly to the interfaces is that we wanted to test the Transport Layer from the application point of view, i.e. analyze how much delay and jitter does the protocol stack add prior to transmission. In this situation the communication domain is `PF_INET` (IPv4 Internet Protocols), the type of socket is `SOCK_DGRAM` (connectionless, unreliable message delivery) and the protocol parameter is set as `IPPROTO_UDP`.

When generating traffic, the *socket thread* will be suspended until the child thread handling the generation process terminates. Afterwards, it will wait for maximum 3 seconds for the manager to collect results and then terminate:

```

pthread_join(hComThread, NULL);
while(lpRead[ipstTraf.bThreadId] && (iCnt > 0))
{
    iCnt--;
    sleep(1);
}

```

The things are a little bit more delicate for capture sessions, as the function that dispatches data from a network interface suspends the execution until a new PDU is received. This problem could be overcome in two manners: either by checking the communication descriptor before trying to receive the data or by forcedly closing the socket when the test duration is considered to be over. The first one may seem more elegant at a first look, but in fact it has two disadvantages. If constantly checking the state of a descriptor without idle period, the processor will become overloaded and other threads will be influenced, while it is possible to have a small amount of data to be received (an agent will not know at what rate the data to be capture shall arrive). One may think that if there is not constantly data to be received, the thread could be suspended for short periods of time so that the processor can allocate appropriate time to other parallel processes. This is true, but in this situation, for higher rates, data will be read in bursts, which means that additional packet delay will be considered. So we chose too let the *socket thread* monitor its child process, meaning it will check every 100 milliseconds to see whether the session needs to be terminated (runtime is exceeded). In that case packet interfaces sockets will be closed; consequently the reception will be interrupted. This approach fails for datagram sockets, which have great latency in returning from a receive function when the socket has been closed. This is solved by sending a loop-back packet on the receiving interfaces in order to cause return from the `recvfrom` function. After this the child thread will get the system time and will know that it should terminate. Similar to generation sessions, a safety time interval is left for allowing result collection before freeing resources. In what follows the paper will gradually explain how traffic generation and capturing is performed for each protocol.

### 3.2.5.2. IEEE 802.3 / Ethernet Fame Generation

There are certain steps that are common to all types of traffic generation sessions. We will focus on presenting them in this section but it is important to have in mind that they operate exactly the same for IP and UDP traffic. For each new thread that will generate traffic, the manager will specify the time distribution of the packets and what should the process do in case it fails to deliver data at the expected moments.

There are three types of distributions considered: periodic, Poisson and link flood. The last one is not really considered to be a distribution. What the software will do if link flood was chosen is to try to generate as many packets as possible, the rate highly depending on the processor's clock frequency. In this case it is not guaranteed that they will be delivered periodically, but rather in bursts, as now time control is performed. This option may be useful for testing the overall performance of a system. In case it was demanded to deliver PDUs according to a periodic or Poisson distribution each transmitting function will calculate offline the exact moments inside a second at which a packet should be sent. This helps reducing execution time, letting the send loop focus only on data delivery.

For periodic distributions, the agent receives from the manager the value for the transmission rate (in frames per seconds/packets per second/ datagrams per seconds). A table of 'rate' elements will be dynamically allocated and filled with values that represent the difference in microseconds between two consecutive packets inside a second (3.1). The next time for delivery will be the last value plus  $t_i$ .

$$t_i = i \cdot \frac{10^6}{rate} [\mu s], \quad i = \overline{0, rate - 1} \tag{3.1}$$

Assuming a Poisson distribution of packets, the manager shall provide the average number of packets per second. This will be converted to the average time distance between two consecutive packets, and the array of difference between two consecutive delivery moments will contain time intervals that are Poisson distributed with respect to the mean. The manager will also provide the value of the step to be used for modifying an interval. According to Poisson's law, the probability of having an interval of duration  $k$  between two packets, when the average packet distance is  $\lambda$  is given by (3.2).

$$p(k) = \frac{e^{-\lambda} \lambda^k}{k!}, \quad k = 0, 1, 2, \dots \tag{3.2}$$

The probability distribution function looks similar to figure 3.5.

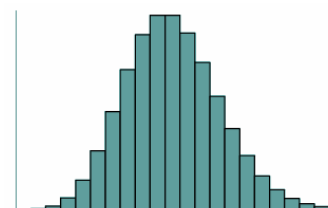


Figure 3.5. Probability distribution function of a Poisson process

As the Poisson is defined for an infinite number of events, what it is actually needed is to have a number of time intervals whose distribution approximate a Poisson process and whose sum equals one second. So starting from the mean value the algorithm computes the probability of occurrence of an interval that is smaller/larger than the mean with  $j$  steps. The probability is converted to an integer number of repetitions of that interval inside a second. The calculations are carried on since the sum of the intervals multiplied by the number they appear inside a second equals one second. So we will select only limited number of intervals around the mean, as they have the higher probability. Also, the probabilities will not be computed in discrete points but as the area of the trapezoid formed by the two neighbouring points on the probability distribution function and the abscissa. Finally, the intervals are randomly spread across the second (random permutations) using the procedure known as *Knuth shuffle*. Basically, the array is scanned and every element is interchanged with a randomly chosen one. The algorithm is presented below.

```

for(dwCnt1 = 0; dwCnt1 < dwCnt; dwCnt1++)
{
    dwCnt2 = dwCnt1 +
        (int) ((float) (dwCnt-dwCnt1-1)*rand() / (float)RAND_MAX);
    dwStep = lpTime[dwCnt1];
    lpTime[dwCnt1] = lpTime[dwCnt2];
    lpTime[dwCnt2] = dwStep;
}

```

Another issue was encountered when implementing the Poisson probability function. It is impossible to represent on predefined C data types the factorial of a number and repetitive multiplication require a long execution time, so we performed a mathematical calculus in order to transform the expression of the probability distribution function into a sum and rescale the values of the term to be easier to fit in predefined data types. The idea is presented in (3.3).

$$p(k) = \frac{e^{-\lambda} \lambda^k}{k!} = e^{-\lambda} \cdot \prod_{i=0}^k \frac{\lambda}{i} = \exp\left(\log\left(e^{-\lambda} \cdot \prod_{i=0}^k \frac{\lambda}{i}\right)\right) = \exp\left(\sum_{i=0}^k \log\left(\frac{\lambda}{i}\right) - \lambda\right) \quad (3.3)$$

Another common issue for all types of traffic generation sessions is real-time failure correction. Real-time failure means the software fails to send a packet at the expected time. This situation typically occurs at high transmission rates, when the time distance between two consecutive packets is smaller than the execution time of one cycle of the loop that encapsulates the data and sends it on the interfaces. This error may also occur when the rate to be transmitted exceeds the link's maximum throughput. The software will not be able to send new packets when the transmission buffer is full so the send function will wait, and consequently the next moment of scheduled delivery will be over passed.

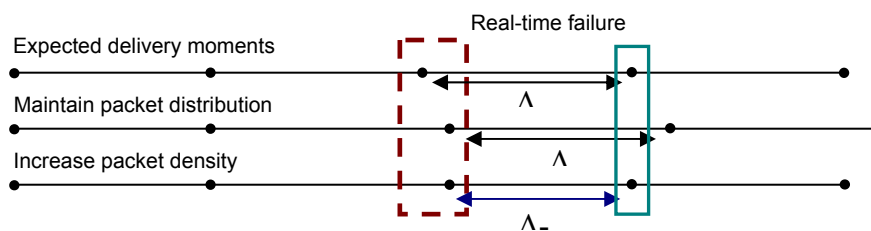


Figure 3.6. Real-time failure correction methods

The manager will specify prior to session start what an agent should do in case real-time failure occurs. There two choices available. The first one will focus on maintaining the time distribution

(periodic/Poisson), with the cost of virtually increasing the test length. The test duration will be actually the one established at start time, but not the entire number of frames/datagrams will be sent as expected. On the contrary, the second approach will try to send more packets in case of a real time failure, until a new packet is delivered on the expected time. A simplified example of how the two methods work is presented in figure 3.6. The matters described so far in this chapter apply also to other layer traffic generation processes (IP packets and UDP datagrams).

Now we will focus on some aspects that are particular to the Ethernet / IEEE 802.3 traffic generation. Prior to any transmission process, a standard layer 2 address structure needs to be initialized. The `sockaddr_ll` is presented below.

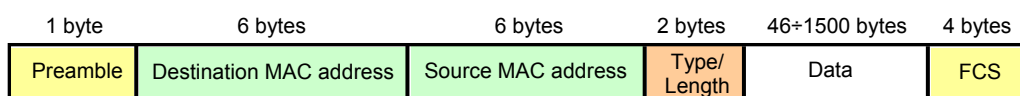
```

struct sockaddr_ll {
    unsigned short  sll_family;    /* AF_PACKET */
    unsigned short  sll_protocol; /* Ethernet protocol */
    int             sll_ifindex;   /* Interface number */
    unsigned short  sll_hatype;   /* Header type */
    unsigned char   sll_pkttype;  /* Packet type */
    unsigned char   sll_halen;    /* Length of address */
    unsigned char   sll_addr[8];  /* MAC address */
};
    
```

When using *raw* sockets, the domain is always the low packet interface and `sll_protocol` specifies the type of data encapsulated (`ETH_P_802_3` or `ETH_P_ALL` can be used for Ethernet frames). `sll_ifindex` represents the index of the interface from the operating system point of view. This value is obtained when the manager requests a low level session using one of the agent’s interfaces. It sends up the appropriate MAC address of the desired NIC and the message dispatcher will open a communication stream with the kernel, pass it as argument to an `ioctl`<sup>10</sup> call and obtains the interface index which is transmitted as parameter to the traffic generation routine. The header type is ARP as defined in the `linux/if_arp.h` include file. The packet type indicates whether the packet is addressed to another host, to local host, to a broadcast or multicast address. Address length is considered 6 bytes for Ethernet frames and the last record of the structure represents the destination MAC address.

The next step is to allocate memory for the new frame and prepare its header (a pointer to an `ethhdr` – Ethernet header structure is assigned the value of memory address where the frame content is build). For a better understanding, the structure of the Ethernet frame is presented in figure 3.7. The routine fills up the destination hardware address, source hardware address and type/length field with the values indicated by the manager. As for the payload, each frame will contain at least a stamp which stores the following information:

- Transmission time of the packet (in Epoch<sup>11</sup> format, with microsecond precision)
- Flow ID
- Sequence number



**Figure 3.7.** The Ethernet / IEEE 802.3 frame structure

<sup>10</sup> System call that allows an application to communicate with a device driver. It takes as parameters a socket descriptor, a request code and a pointer to a memory location where the result will be returned.

<sup>11</sup> Time since 00:00:00 UTC, January 1, 1970

These values will be used at reception for computing QoS parameters. The rest of the payload, up to the required sized is filled with random data. The preamble and the frame check sequence (FCS) are added by the device driver. Having the frame headers prepared, the routine passes to examining what kind of distribution has been chosen, an except for link flood, to see which mechanism of real-time correction shall be used if necessary. Those issues have been explained previously in this paragraph. Now the code will enter a `while` loop which will be executed until the programmed runtime is exceeded, the application is stopped or a send error occurs. Because most of the conventional operating systems, including most of the Linux distributions, are not real-time systems, i.e. a software is not able to have full control of execution times, a rather resource consuming approach has been implemented. To be more precise, every system implementing the POSIX standard needs to provide functions for suspending the execution of threads (`sleep`, `usleep`, `nanosleep`). But in case of non real-time kernels, the functions that suspend the thread for sub second periods of time do not guarantee that the process will restart execution after the desired time interval. Which means trying to suspend a process for several microseconds will lead to a pause in execution of around one millisecond. In fact the minimum value guaranteed for thread suspension is 1 millisecond. This is rather inconvenient when packet rates higher than 1000 packets/second are required. Consequently our approach will follow the below mentioned steps at each cycle of the loop:

4. Send the prepared frame
5. Relinquish the use of the processor
6. Update the stamp for the next frame
7. Fetch the next time instance for transmission
8. Check for real-time failures
9. Wait in a loop for the next moment of transmission

There are a few things to be explained about the above mentioned steps. The frame is sent using the `send` function, taking as arguments the socket assigned by the parent thread, a pointer to the location of the frame data, the size of the frame, the address of the `sockaddr_ll` structure and the size of the later. The use of the processor is abandoned by calling the `pthread_yield` function, which puts the thread to wait in the processor's queue until it is scheduled again. If the run queue is empty when the `pthread_yield` subroutine is called, the thread is immediately rescheduled. On the other hand, if multiple sessions are running, as they are scheduled with real-time policy, they will be treated first and will not be competing with lower priority processes. Despite that, a traffic generation session may be considered a relatively processor consuming task.

For each frame sent, the total number of delivered frames is updated, so that the manager can inspect the total frame count up to the request moment. Another value that is updated whenever necessary in order to make it available for managers is the number of real time-failures. Those values, together with the timestamp of the last sent packet are stored in corresponding vectors, at the index given by the assigned `ThreadId`.

When the test runtime expires the socket is closed, the resources are set free (memory allocated for storing frames and time intervals between consecutive frames) and the routine clears the running flags so that the main control thread knows that it has finished his job and its index could be assigned to a new thread whenever required. This mechanism ensures that up to 255 consecutive traffic threads could run simultaneously. Of course, for precise measurements it is recommended to make prior estimations when running several parallel sessions on one agent, especially when demanding real-time operation at high data rates. In those cases, capturing threads could be asked to perform only offline analysis in order to spare additional resources.

Traffic capturing session will be explained in a later paragraph. For status reporting consideration, each generating/capturing process updates some global variables at the beginning and the end of the execution in order to give the possibility to the user of inspecting the total number of sessions running at a certain time.

The advantage of data link layer traffic generation is that it eliminates all the delays added by upper layer protocols during encapsulation and queuing processes and the analysis directly reflects the quality of the link. Also it is highly useful for determining the overall performance of a NIC. The disadvantage is that layer 2 tests can only be performed inside LAN domains as layer 3 devices (routers) strip the Ethernet header of a frame when forwarding packets to destinations outside the local network.

### 3.2.5.3. IPv4 Packet Generation

When layer 3 traffic generation is requested by network managers, a separate thread will handle the task, executing a routine named `IPSendFunction`. A pointer to a data structure containing all the characteristics of the new flow will be passed as parameter. The data structure is presented below.

```
typedef struct {
    unsigned char  lpRemote[8];
    unsigned char  lpLocal[8];
    unsigned char  bInterfIdx;
    unsigned long  dwPps;
    unsigned short int wPacketSize;
    unsigned char  bDistrib;
    unsigned short int wRuntime;
    unsigned short int wFlowId;
    unsigned long  dwStep;
    unsigned char  bRTFailMeth;
    unsigned char  bThreadId;
    unsigned int   sSock;
    unsigned long  dwRemoteIP;
    unsigned long  dwLocalIP;
    unsigned char  bTTL;
    unsigned char  bToS;
    unsigned char  bProtocol;
} IP_SEND_TRAF;
```

Similar structures are used to defined all other types of traffic, regardless the protocol involved or the type of operation (send/receive). We will take a closer look at the purpose of each record of the structure, as they work quite the same for all the sessions. Before starting a session, such a structure is filled with data received from the manager, but also assigned by the main control thread and parent threads. A manager gathers information about the agent's configuration after the registration process, consequently it will have available all the details of any network interface card (MAC address, configured IP address and network mask, MTU, etc.). When asking for generating a flow of IP packets it will send the MAC address of the device on the agent that should perform the task, the remote MAC (in case of tests between computers placed in different networks this will be stripped but the packet will not be lost as the router relies on the IP header of the datagram), the IP assigned to that particular interface, the remote IP address, the number of packets per seconds (for periodic flows) or the average number of packets per second (for Poisson traffic), the packet size, the time distribution type (0 – periodic, 1 – Poisson, 2 – link flood), the test's duration in seconds, a *Flow ID* that will be used at reception for parameter calculation, the size of the steps used to modify time intervals in case of Poisson distribution, the

correction method to be used in case of real-time failures, a value for the TTL<sup>12</sup> field, Type of Service and the value to be placed in the Protocol field of the IP header. The software identifies the interface index, assigns a thread ID for the new process and associates a new socket for the task.

One of the first steps to be performed before sending IP packets is to instruct the kernel to set the desired values for the TTL and ToS fields of the header. This is done by setting additional options to the layer 3 socket as shown below.

```
bVal = ipstTraf.bTTL;
setsockopt(ipstTraf.sSock, SOL_IP, IP_TTL, &bVal, 1);
bVal = ipstTraf.bToS;
setsockopt(ipstTraf.sSock, SOL_IP, IP_TOS, &bVal, 1);
```

Initially we have tried to encapsulate directly the IP packet into the Ethernet frame and manually build the IP header (figure 3.8). But this approach requires the routine to know in advance the MAC address of the next hop. So we rely on the kernel to build the Ethernet and IPv4 header, rather than manually build them, but we managed to specify the fields highlighted in green and orange as imposed by the management console.

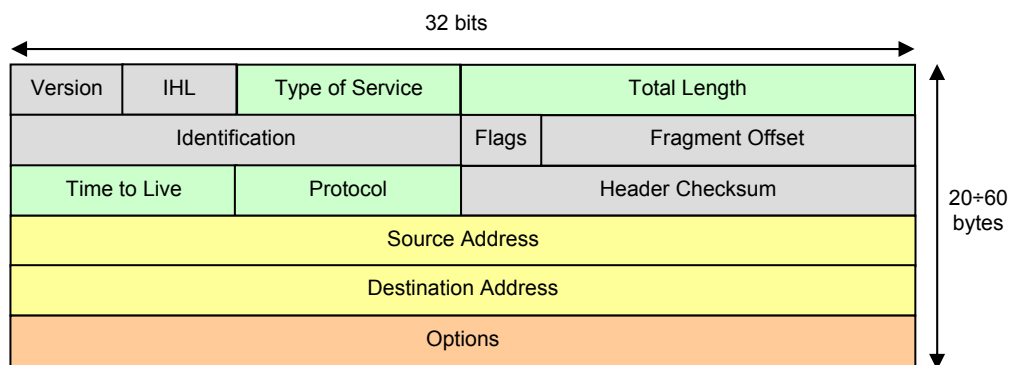


Figure 3.8. The IP header structure

The fields highlighted in grey will be automatically set by the OS and the *options* field will not be used. The value for the *version* field will be 4, which represents IPv4, IHL (Internet Header Length) will be set to 5, meaning the size of the header is five 32-bit words (20 bytes) and the identification field is set to zero. Also the DF (don't fragment) flag will be set, so the router will be instructed not to fragment the packet. Hence, the identification and fragment offset fields have no use. The ToS field can be set by the manager. Recently those bits have been defined through *DiffServ* and *ECN*, as presented in Chapter 2. The reason why we decided to let the user specify this field is that one may want to investigate the behaviour of a layer three device performing Quality of Service based routing. The *total length* field is also established by the manager, representing the total size of the packet.

Another feature that managers can set when requested by their users is the value of the *TTL* field. This is not actually useful from the QoS parameter measurement point of view but one may combine the use of a Network Measurement Analyzer with some other tool. For example, it is possible to run a group of sessions that only generate traffic to a given destination and choose TTL as the parameter to be changed. This way, by either using a network analyzer like Ethereal

<sup>12</sup> Time To Live - field of the IPv4 header, which is decremented by each hop.

or tcpdump, it is possible to filter ICMP messages and look for *Time Exceeded*<sup>13</sup> replies. This is a possible method of finding out the number of hops to the destination in cases when it is known that packet filtering is performed at the other end or by intermediary nodes and *echo requests* are ignored. More details about creating parameter based sessions using the Network Management Console can be found in [5].

The manager is also able to set a value for the *Protocol* field. This feature could be helpful when dealing with Cisco routers that have policy-enabled routing. One possible application of policy routing is to provide protocol-sensitive routing, hence offering different treatment to different types of protocols and applications.

The source address will be the one assigned to the interface that was requested to send data and the destination address is received from the Network Measurement Analyzer. It is assumed that the engineer leading the tests from the management station has good knowledge about the topology of the network under testing and he will provide valid addresses to agents. Assumed an invalid IP address was provided as destination for a generator, this will only load the link without obtaining expected results. Of course it is possible to select a simple generation session, when one only wants to test the available bandwidth or the system's performances.

The payload will contain the stamp that shall be used at the other end for various parameters computations and dummy data to complete the required packet size. As for Ethernet traffic, it is possible to generate periodic packets, Poisson distributed packets or simply flood the link. Details about those approaches were presented in paragraph 3.2.5.2.

### 3.2.5.4. UDP Traffic Generation

User Datagram Traffic generation relies on operating system's implementation of the TCP/IP protocol suite. As explained before, a higher level socket will be used and the software will not have to take care about building up the structure of the datagram prior to transmission as the kernel will take care of it. Of course that it would have been possible to manually perform the encapsulation and forward the data directly to the device driver, but it is also desirable to know how the operating system influences QoS parameters.

In case of datagram sockets the steps required before transmission are slightly different from the ones involved in raw socket based communication. First two IP socket addresses have to be defined, one for each end of the communication flow. The data structured used for this purpose is defined in `netinet/in.h` and presented below.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    u_int16_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    u_int32_t      s_addr;     /* address in network byte order */
};
```

---

<sup>13</sup> Time exceeded is an ICMP message generated by a layer three device to inform the source that the packet has been discarded because TTL reached the zero value.

The UDP sending routine fills up two `sockaddr_in` structures, one with the IP address and port used by the local interface and the corresponding values for the other peer. The opened socket is *bind* to the local address and in case no errors occur (invalid address, bad descriptor, address in use, etc.) the transmission can start. One of the most frequent mistakes is trying to bind a socket to an address and port already in use. This could occur when running multiple UDP generating flows on the same agent and the engineer operating the Network Measurement Analyzer misses to change the port settings for a new session. If this step fails, the transmission will never start and the other end might collect erroneous results. Of course, in such situations the error log is appropriately update, hence if unexpected results appear one may easily find out the cause of the problem. The `bind` procedure is presented below.

```

if (bind(ustTraf.sSock,
        (struct sockaddr *)&saLocal, sizeof(struct sockaddr)) == -1)
{
    switch (errno){
    case EBADF:           ReportError(0x00040001, 0x0006); break;
    case EINVAL:         ReportError(0x00040002, 0x0006); break;
    case EACCES:         ReportError(0x00040003, 0x0006); break;
    case ENOTSOCK:       ReportError(0x00040004, 0x0006); break;
    case EFAULT:         ReportError(0x00040005, 0x0006); break;
    case ENOENT:         ReportError(0x00040006, 0x0006); break;
    case ENOMEM:         ReportError(0x00040007, 0x0006); break;
    case EADDRINUSE:     ReportError(0x00040008, 0x0006); break;
    case EADDRNOTAVAIL: ReportError(0x00040009, 0x0006); break;
    default :            ReportError(0x0004000A, 0x0006); break;
    }
}

```

Next the packets will be delivered to the other end, according to the required time distribution and using the desired real-time failure method if necessary. The routine updates periodically a set of parameters that can be requested by managers: the transmitted packet count, the time instant of the last transmitted packet and the total number of real-time failures. Centralized results gathered from transmitted agent are available in the experimental related section of this paper.

### 3.2.5.5. IEEE 802.3 / Ethernet Fames Analysis

Another essential feature of the Network Measurement Agent is the capturing and analysis capability. The traffic control service take care of double session based analysis, i.e. it is able to capture Ethernet frames, IP packets and UDP datagrams from similar agents and compute a set of QoS parameters. Simple capture sessions that reflect only the link usage are available through PCAP service and the Endace card tool. When starting a new capture session, the manager can specify whether it demands a detailed or simple analysis, meaning what parameters should be computed. The differences between simple and detailed analysis are shown in table 3.1.

QoS parameter	Unit	Simple analysis	Detailed analysis
average throughput	bps	yes	yes
average OWD	µs	yes	yes
minimum OWD	µs	no	yes
maximum OWD	µs	no	yes
average PDV	µs	yes	yes
minimum PDV	µs	no	yes
maximum PDV	µs	no	yes
OoO packets	packets	no	yes

**Table 3.1.** Parameters computed by simple/detailed analysis

Those parameters are computed in real-time and periodically sent to the management station. An agent also updates the total number of received packets, regardless the type of analysis in order to permit the manager to determine the packet loss ratio. Another option that could be selected is to perform an offline analysis based on a dump file. The operations involved by all types of analysis will be explained in this paragraph and they work exactly the same, regardless the protocol used between generator and capture agents.

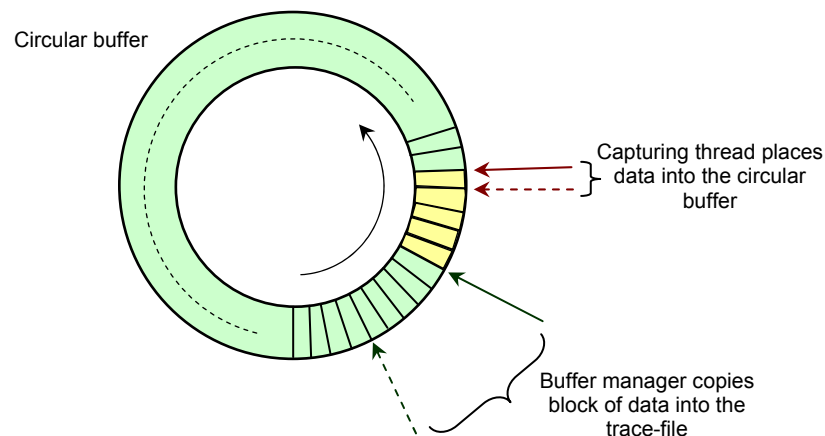
The QoS parameter calculation process relies on the stamp added to each frame/packet/datagram by the transmitter. The receiver agent extracts the stamp from the PDU (transmission timestamp, flow identifier, sequence number) and adds to it the value of the time moment it was received with microsecond precision. An essential role for the accuracy of the measurement is played by the synchronisation between the agents. In case they are not well synchronized to a common clock provided by a GPS system or an NTP server, the results will be altered, especially the values for the one way delay. Some of the parameters are calculated for a time interval that is specified by the manager and other are computed for the entire length of the test.

For each packet the one way delay is computed as the absolute value of the difference between the reception and transmission timestamps. It is computed as an absolute value because for non-perfectly synchronised systems this value could be negative, which may seem as the packet was received before transmission. The packet delay variation is determined as the difference between the one way delays of two consecutive frames. The first packet is considered to have zero PDV. The minima and maxima of those parameters are update if necessary at each received frame when a detailed analysis has been requested. Based on the sequence number, if two consecutively received frames will not have consecutive sequence numbers, it is considered that they are out-of-order.

It was explained that the manager is supplied with average values of the OWD, PDV and throughput computed on a predefined time interval. In order not to consume from the processing time requested by the capture process, a separate thread is created that calculates those average values every  $k$  microseconds ( $k$  represents the parameter update time as passed by a Network Measurement Analyzer). The thread calculating the parameters and the thread handling the reception need to be synchronized as they both use several common variables (packet count, delay sum, delay variation sum). This is realized by the use of a flag. When one of the threads wants to use/update the variables it checks the *lock* flag, waits if necessary, then modifies the flag in order to inform the other one that he is performing some operation and unlocks the data when its task is finished. The update of the average values is typically performed every second so the parameter computation thread will not seriously influence the execution of the receiver, while this one will not have to spend additional time for mathematical operations.

There are situations when an engineer would desire high-rate tests and is not interested in a detailed analysis, but rather an overall image of the parameters of the network. For those types of situations it is possible to select a simple on-line analysis and perform a detailed analysis based on a dump file. This may seem a relatively simple task but in fact it is quite complicated. First of all it is not efficient to dump all the receive data on a disk because this could become full very fast at high rates and for long test duration. Also, writing to disk is a slow operation especially with regular hard-disks but even high performance SCSI drives are not fast enough compared to Gigabit networks. Consequently, we decided to store only important information. A data structure was defined whose records store the transmission and reception timestamp, flow identifier, sequence number and the size of the frame. Also, it is not really efficient to write small amount of data. It is preferable to transfer data on the disk in bigger chunks. In order to save time, the receiving thread will place this *capture header* into a circular memory buffer and from where the data will be written into a trace-file by another thread. The 2048 element buffer

is associated to each running session that has dump enabled. The position of the last element written in the buffer and the last copied in the file need to be known by both the capturing thread and the one that empties the buffer in order to make sure that no data is overwritten before it is saved into the trace-file and no data is read until present in the buffer. As the buffer length is  $2^{11}$  elements, 11 bits will be need for indexing, so an 11-bit mask has to be applied on the 16-bit data storing the positions. The thread that writes into the file will perform operations every 200 milliseconds ensuring it is not resource consuming and also copying bigger data blocks. The name of the trace file contains the `trace` prefix followed by the flow id and the `.cap` extension. This way it will be easily recognized when performing offline analysis. In case to much data is written into the buffer and it becomes overflowed, the manager will be noticed, though this is quite unlikely to happen on fast machines. The mechanism of writing frame information into the circular buffer and copying into a file is presented in figure 3.9.



**Figure 3.9.** Data dumping mechanism

At the end of the test, a specialized function opens the trace file and computes all the parameters listed in table 3.1 plus the packet count and makes them available for managers. The message dispatcher will inform a Network Measurement Analyzer whether the offline analysis is ready or not, and after completion it will mark the thread id as read and available for new processes. The features presented so far are also applicable for IP and UDP captures and will not be mentioned in the sections focusing on those types of tasks.

When capturing Ethernet frames raw sockets are used similar to the generation process. A data link layer address structure needs to be filled with device specific information (socket family, protocol, interface index, expected packet type, hardware address of the interface and address length). For receiving data on a socket, the `bind` operation is always mandatory. When capturing layer 2 data, the routine will not know in advance what type of data to expect, so a `recvfrom` call will get all the frames from the interface's Rx buffer. First of all it is desired to have enough memory available for receiving frame in order to avoid errors, so the routine allocates a MTU size buffer (1500 bytes). Upon a '`receive from`' call, a `sockaddr_ll` structure will be filled with information related to the sender. This is extremely useful especially when a network adapter is connected to a hub and frames that are not address to it can be intercepted. Actually there are two conditions that need to be fulfilled in order for a capturing thread to take a frame into consideration. Those requirements are listed bellow:

- The remote MAC address must match the one previously specified by a manager
- The data corresponding to the *flow id* inside the stamp of the packet must match the flow id prior communicated by the manager.

If these two items fail to match, the frame will be discarded. Otherwise, QoS parameters will be calculated according to the requirements. The capture process will be stopped by the parent thread in case the receiving loop blocks inside a `recvfrom` call waiting for further frames. In case an offline analysis has been also selected, supposing the file buffer overflows, certain received packets will not be transferred into the trace file so the overall results might look slightly different from the real-time ones. It is not recommended to perform simultaneously online and offline analysis at high packet rates because they will require a lot of processor time and the data may be received in burst, hence the values of the delays may be affected. A possible solution for combining the two options at high speeds is selecting a large parameter update interval.

### 3.2.5.6. IPv4 Packet Analysis

IP packet capturing is also based on raw sockets. The manager specifies which interfaces should be bind for reception. Compared to Ethernet traffic analysis, this routine will be instructed first of all to consider only IPv4 packets.

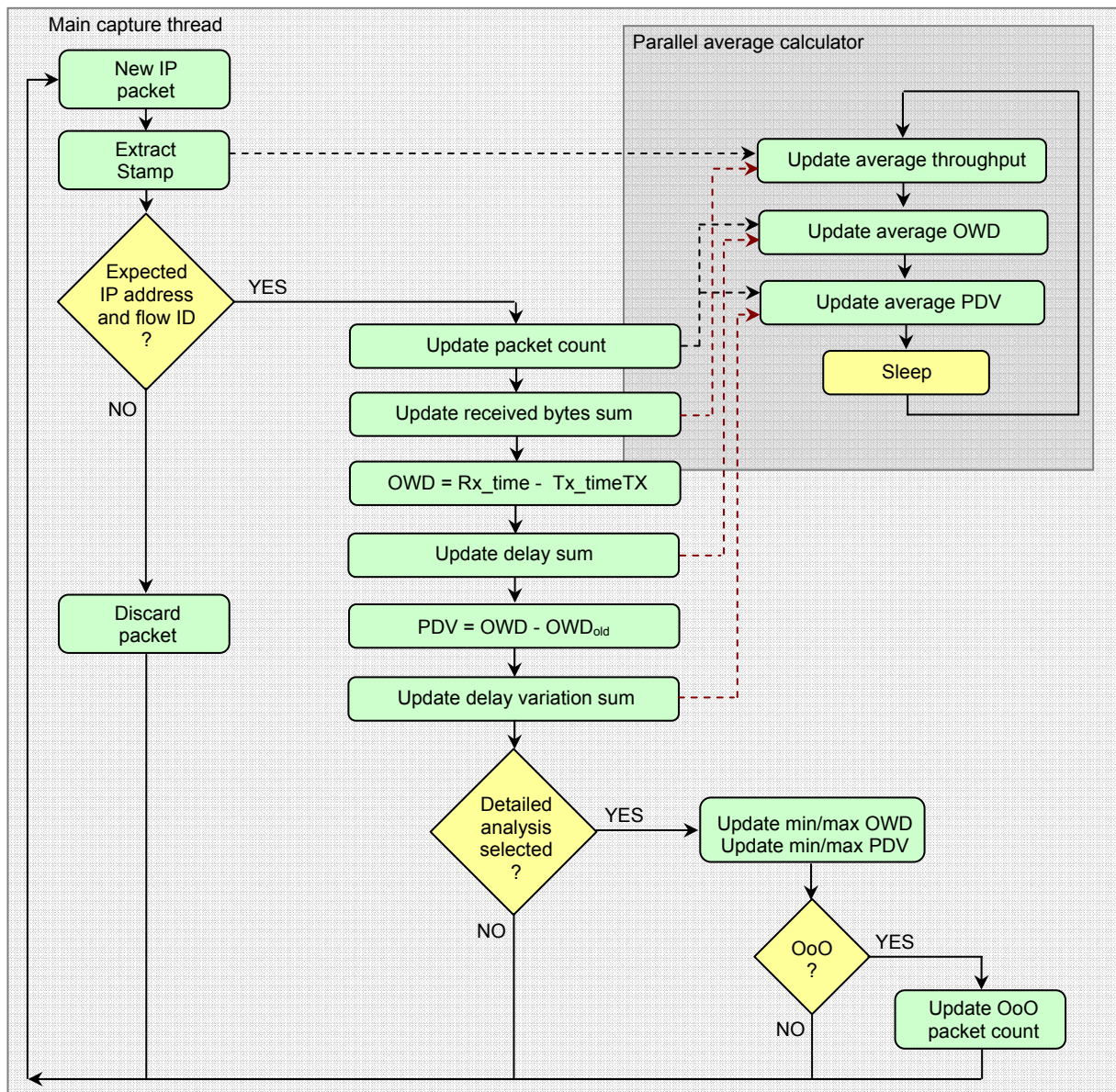


Figure 3.10. QoS parameters calculation procedure

The protocol record of the data link layer address structure will be set to `ETH_P_IP`, i.e. only frames whose type/length field match the defined value for IP (`0x800`) will be captured.

Broadcast and multicast frames will be ignored, by setting the packet type record to `PACKET_HOST`. Of course that common traffic between two computers contains various IP packets exchanged by different applications.

Nevertheless, multiple hosts are sending packets to an agent connected to a network. Consequently, the routine must select out of all the IP packets received on an interface only those that are significant from the test's point of view. A Network Measurement Analyzer will provide sufficient information to capturing agents before starting the other end, in order to make sure the computed results are valid.

Upon the reception of an IP packet, the analysis routine will compare the source address from the IP header with the remote IP address already known from the manager and the expected flow identifier with the 16-bit word from the payload that should eventually store a flow id. If these two items match it is considered that a test packet has been received and the number of received bytes, the transmission and reception timestamps and the sequence number are memorized for QoS parameters calculation. As explained before some metrics are computed by a parallel thread and when required extreme values are update for the whole test length. The steps involved by these operations are presented in figure 3.10.

The IP capturing routine could be improved in future work in order to count the number of hops a packet has passed through from the source to the destination. The disadvantage of the implemented method for IP packet generation and capturing consists of the fact that relies on raw sockets, and a packet of maximum 1500 bytes is placed inside the Ethernet frame payload. One possible improvement could be to generate bigger packets but fragment them prior to transmission in order to fit the layer 2 MTU and reassemble them at reception. But this approach requires a more thorough investigation of the headers and will not be covered in this paper.

### 3.2.5.7. UDP Datagram Analysis

Measurements based on User Datagram Protocol flows rely on high level datagram sockets. As explained in paragraph 3.2.5.4 the routine will not have to extract IP packet out of the Ethernet frame, then the UDP segment and finally the payload because the operation system will take care of these steps. Instead, it will prepare a socket address structure with the IP address of the receiving interface and the port to be used for reception. The purpose of UDP based sessions is to provide a complete characterisation of an end-to-end connection, thus simulating a connection between connectionless user applications such as VoIP or video broadcasting.

The analysis routine will capture only datagrams received on the specified port and local IP address. It is obligatory to specify the local IP address used for testing as an agent may have several network interface cards that can be simultaneously used for different purposes, testing different networks or links and using different protocols. Upon reception of a user datagram the `recvfrom` function will return into an address structure the remote IP address and transport port and the agent will compute traffic parameters only for the data received from the expected node. The description of the remote station is provided by the management console before the start of the test.

UDP-based link testing between a conventional NIC and an Endace DAG card will require some additional configuration. As the DAG card cannot be assigned a logical address, the traffic generating machine needs to be added a static ARP entry with a 'dummy' hardware address and

an IP address that is assumed to be assigned to the DAG card. Further details about data capturing and analysis using the DAG card will be explained in 3.2.7.

The approach used for QoS parameter calculation is similar to the one implemented for Ethernet and IP traffic analysis. Simple or detailed analysis is similarly available and for resource saving off-line analysis based on a dump file could be chosen. The mechanism is exactly the same as presented before, relying on parallel processing threads that either computed an average of OWD, throughput and PDV at certain time intervals or transfer the essential part of the payload into a file.

For offline parameter calculation a specialized function will open the appropriate trace file (the one that contains inside its name the flow id) and read all the data. The information inside the trace file consist of multiple predefined structures whose records store transmission and reception time, packet size, sequence number and flow id. The function returns the overall statistics of the test: average throughput, average OWD, minimum OWD, maximum OWD, average PDV, minimum PDV, maximum PDV and the total number of out-of-order packets. When dump analysis is selected and the test duration is relatively long, a flag is used to signal the requesting management station whether the offline analysis has been completed or not, as this make take few seconds to finish the computations. Its advantage consists of the fact that not a full dump is performed, but only measurement essential data is stored.

Special attention had to be given to the data types used for computing the parameters, because at high rates, summing up the size of the packets for computing throughput and working with microseconds for delay and jitter values require large representation. Consequently, 64-bit representations have been used when computing delay and delay variation with microsecond precision. Another issue is memory usage, as for high rates arrays that store traffic engineering related items (e.g. delivery time instants for periodic/Poisson distributions of packets) will consume considerable amount of memory. To optimize the resource utilisation, such variables are dynamically allocated just before they need to be used and the resources are manually made free after the process requiring them is finished.

### 3.2.6. The PCAP Service

The PCAP service is based on the Packet Capture Library which offers a specialized interface for low-level network monitoring. The system needs to have installed the *libpcap* kernel module and the agent should be run in super-user mode. The advantage of using this library is that it permits full-link monitoring by setting a device in *promiscuous mode*. A network interface set in promiscuous mode passes all the received traffic to the CPU rather than only the packets that are addressed to it. When a computer receives a particular packet, it checks the hardware address in it to see if the packet is addressed to it. If not, then the network card drops the packet. When in promiscuous mode, the network card doesn't drop the packet, thereby enabling it to read all packets. This option is typically used for network connectivity diagnosis and packet sniffing. The reason why we have chosen to add this facility to the measurement agent is because an engineer might be interested in monitoring a link of a LAN that uses a hub, to have an image of the overall bandwidth usage. In other cases, high-speed link monitoring may be done using optical splitters, consequently the monitoring interfaces needs to be able to track packets that are not address to it. But the use of a device in this mode has two drawbacks:

1. Slightly increases the CPU load
2. Some software might send responses even though packets were not addressed to them

At start-up the agent will try to set all the available networking devices in promiscuous mode and separate threads will determine the link throughput and save it in 20-seconds history vectors. These vectors are mainly used for displaying the network activity in the status section of an agent that has been started with the graphical interface enabled. Also, upon a management station requirement, the link throughput is sent.

The steps used for capturing all the packets from an interface are described below:

1. obtain a packet capture descriptor to examine packets on the network
2. periodically collect and process packets
3. calculate parameters
4. close the descriptor

The first steps involves opening a network device with a given identifier (eth0, eth1...), specifying the maximum capture length (the routine is able to capture packets up to  $2^{16}$  bytes long), setting it in promiscuous mode, passing an infinite timeout and assigning an error buffer. In order not to overload the processor the obtained descriptor is set as *blocking*, which means that when calling for packet dispatching, the execution will be suspended when no packets are to be read. The associated source code is presented below.

```
pcap_t *pcDescriptor;
WORD wMaxSize = 65535;
char errbuf[PCAP_ERRBUF_SIZE];

.....
pcDescriptor = pcap_open_live(lpDev, wMaxSize, 1, -1, errbuf);
if(pcDescriptor == NULL) return NULL;

pcap_setnonblock(pcDescriptor, 0, errbuf);
```

Packets are collected using a `pcap_dispatch()` call, which has been set to process all available packets during a call. For each capture packet it calls another routine that will receive the parameters of the packet and update the byte count. Every 700 milliseconds, the results are updated and the user can inspect the link history using the GUI and simultaneously a management console can gain access to the last calculated value of the throughput. The packet capture descriptor is closed when the measurement agent is shutdown. In order to make distinction between the different threads running PCAP based analysis, each thread receives the index of the interface as parameter, which is then passed to the other routines that it invokes for calculating the throughput.

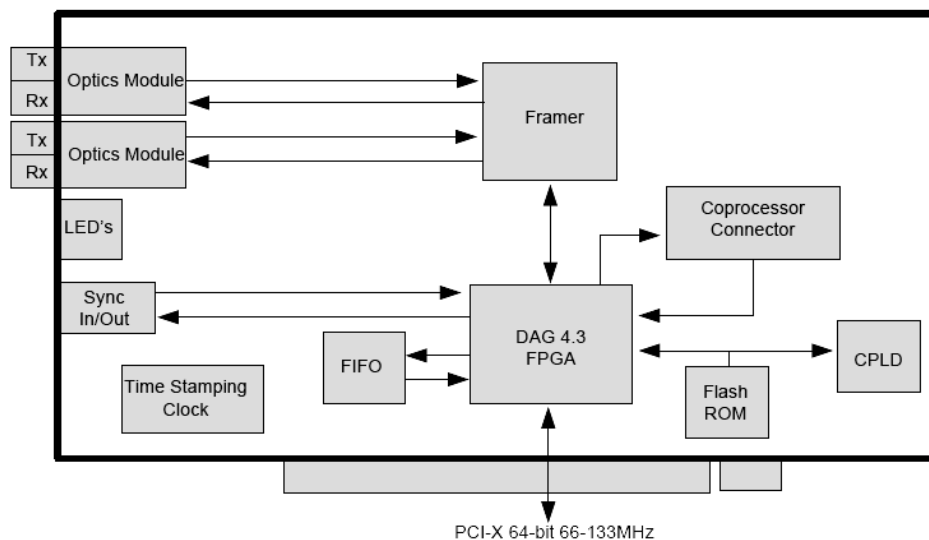
It is possible to combine the packet capture with filtering in order to determine the amount of bandwidth used by a certain application. The Network Measurement Agent does not implement this feature as this might be quite time consuming and we preferred to rely on our own dedicated routine for flow-based analysis and use the PCAP feature only for full-link monitoring, from the throughput point of view. It is important to have in mind that the PCAP service will only operate with Ethernet controllers, that's why when the application has been started in graphical mode, the network activity can be inspected only for the links to which that station is connected through already configured Ethernet devices.

### 3.2.7. The Endace Card Module

#### 3.2.7.1. Overview of the Endace DAG 4.3 GE Card

The Endace DAG 4.3 GE card is a PCI-X hardware network controller dedicated for 100% line rate packet capture and generation. It is able to collect packet header and payload from Ethernet networks with a high precision timestamp and it is protocol independent. It can be connected to the network through two 1000 Base SX optical interfaces and can be externally synchronized to a GPS receiver with nanosecond precision. The architecture of the card is presented in figure 3.11. The minimal system requirements specified by the manufacturer are listed below:

- PC, at least Intel Xeon 1.8GHz or faster
- Intel E7500, ServerWorks Grand Champion LE/HE, or newer chip set
- 256 MB RAM
- At least one free PCI-X 1.0 slot supporting 66-133MHz operation



**Figure 3.11.** The architecture of the Endace DAG 4.3 GE card

The card supports Windows XP/Server 2000/ Server 2003, FreeBSD and Debian Linux operating systems. We have performed the tests using Debian Linux, which is the operating system recommended by the manufacturer. It is recommended that the DAG card should be the only device on a PCI-X bus of a computer, as it makes heavy use of the bus data transfer resources.

Once the card has been inserted into the PCI-X slot, the optical transceivers have been installed, the optical power has been measured and the fibre has been connected to the interfaces, one may inspect if the connection is correctly made by checking the LED status on the back of the card. The LED display functions are listed in table 3.2.

Once the hardware installation has been completed and the status of the card indicates that the device is ready to operate the drivers and tools need to be compiled. Also, as we didn't have sufficient equipment to provide GPS synchronization, the machine that held the DAG card has been configured to synchronize to a NTP server, but the accuracy obtained is within microsecond domain.

LED	Description
1	FPGA successfully programmed.
2	Data capture in progress.
3	Port A Signal Detect – valid optical signal seen by optical receiver.
4	Port A Link Error.
5	Port B Signal Detect – valid optical signal seen by optical receiver.
6	Port B Link Error.
7	PPS Out: Pulse Per Second Out – indicates card is sending a clock synchronization signal.
8	PPS In: Pulse Per Second In – indicates card is receiving an external clock synchronization signal.

**Table 3.2.** The LED definitions for DAG 4.3 GE

Once the hardware installation has been completed and the status of the card indicates that the device is ready to operate the drivers and tools need to be compiled. Also, as we didn't have sufficient equipment to provide GPS synchronization, the machine that held the DAG card has been configured to synchronize to a NTP server, but the accuracy obtained is within microsecond domain.

### 3.2.7.2. DAG Card Configuration

Before being able to use the card with the Network Measurement Agent, prior configuration is required. First of all, after a reboot the DAG driver needs to be loaded using the `dagmem` command. Upon driver loading, memory is reserved for the card (up to 64 MB) and this will not be available to any other process.

Prior to configuration, the most recent version of the FPGA image has been loaded on the card as shown below. For displaying/modifying the current configuration `dagfour` command is used.

```
root@macabeu:~$ dagrom -rvp -d dag0 < dag/xilinx/dag43gepcix-terf.bit
root@macabeu:~$ dagfour -d /dev/dag0
linkA nonic noeql norxpmts notxpmts crc long=1500 enablea
linkB nonic noeql norxpmts notxpmts crc long=1500 enableb
packet varlen slen=48 noalign64
packetA drop=0
packetB drop=0
pcix 66MHz 64-bit buf=32MB rxstreams=1 txstreams=1 mem=0:0
```

As the card has not been tested for traffic generation by the manufacturer and our attempt to use its tools for this purpose has been unsuccessful, the traffic generation feature is not supported by the measurement agent. Also, the card cannot be assigned an IP address and does not have a MAC address. Consequently it cannot be identified as an endpoint for communication and is unusable for management purposes.

In order to make a DAG card available for a measurement agent, the device should be set to allow only packet capture, the link auto-negotiation should be enabled, the snap length should be set to 1516 bytes (maximum Ethernet frame size, without preamble and FCS) and it should be set to variable length snap mode, i.e. it should be able to capture any Ethernet frame, regardless its size. Nevertheless, we have allocated all the available memory of the DAG buffer (128 MB) for

reception. The loopback option has been disabled. An example of enabling the DAG card for capturing on the B port is shown below:

```

root@macabeu:~$ dagfour -b nic noeql slen=1516
linkA nonic noeql norxpmts notxpmts crc long=1500 enablea
linkB nic noeql rxpmts notxpmts crc long=1518 enableb
packet varlen slen=1516 noalign64
packetA drop=0
packetB drop=0
pcix 66MHz 64-bit buf=128MiB rxstreams=1 txstreams=0 mem=128:0
    
```

Assuming all the above configuration steps have been performed, the DAG card is ready to be used by the measurement agent. The measurement process using the DAG card is presented in what follows.

### 3.2.7.3. DAG Module Implementation

Every Endace card is supplied with a set of open-source tools that should either return the throughput of the link or dump all the data on a hard-drive. When performing measurement, if one is interested in the value of the throughput, it may visualize it inside a console and in case further QoS parameters need to be computed the tasks becomes difficult. An additional piece of software is required to interpret the dump file, which is not provided by the manufacturer. Data is stored in an ERF (Endace Extensible Record Format) which is shown in figure 3.12. The other disadvantage is that although the card is able to capture 100% of the links traffic, the disk writing operation is much slower resulting in buffer overflows and thus considerably affecting the quality of the results. On the other hand not all the dumped data is useful, so a lot of disk space is uselessly consumed.

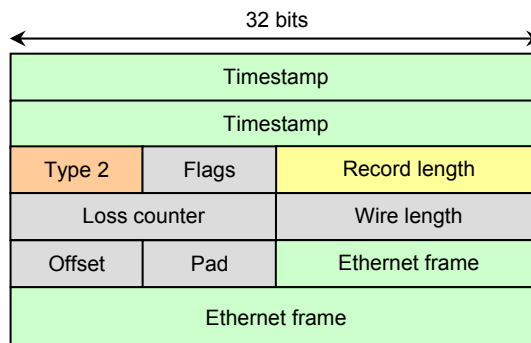


Figure 3.12. The Ethernet variable length record

The DAG module is a modified version of the original *dagsnap* tool provided by the manufacturer that tries to process in real-time the packets rather than storing them on a hard drive. It is also possible to store data for offline global analysis, but in this case only information that is useful from the measurements point of view will be stored, so the efficiency is increased. On the other hand, one must have in mind that the computation of QoS parameters is valid only if at the transmission end a similar agent generates the traffic, so that the receiver knows what to expect.

When a simple throughput analysis is commanded by a management console, the module will only count the number of bytes received per second, without interpreting any data, so a conclusion about the overall link usage could be drawn. In both situations the results are updated every second. This is done by changing the routine associated with the alarm signal of the system with a user routine that computes the required parameters. Practically, regardless if a simple full-

link throughput analysis or a flow oriented analysis has been selected, there are some common steps to be executed:

1. Obtain a file descriptor for the device.
2. Set-up the card for capturing
3. Attach a new stream to the device
4. Set the new alarm signal handler
5. Initialize DAG polling parameters
6. Start the capture loop
7. Exit the loop when runtime expires
8. Detach stream and close descriptor

Inside the main loop, a specialized function extracts the captured data from the card's buffer. The processing done of each record depends on the type of selected analysis. In case a link throughput is the target of the test, for each record, the number of captured bytes will be incremented, and the *'report'* routine will use this for calculating the average throughput in megabits per second. If the manager uses the agent combined with a transmitter, it will prior specify to the receiver what kind of traffic will be transmitted (Ethernet, IP or UDP). Consequently, for each type of protocol, the function will extract the stamp encapsulated at transmission, will use the timestamp recorded by the DAG card and will compute parameters like one way delay, packet delay variation or number of out-of-order packets. After the record is processed it will be discarded from the buffer in order to allow the capture of incoming data. This approach improves the original tool but has also a little disadvantage. The problem occurs at very high packet rates and for long test duration. One may experience packet losses, though the manufacturer guarantees the card is capable of capturing everything. In deed, the DAG card will capture of the traffic on the link, but the additional processing we perform on the records will not empty the memory buffer of the device as fast as the packets are received. Consequently, at some point, if the buffer becomes full because of post-processing latency, some of the packets may be discarded. The idea is that the system running the device should be fast enough in order to guarantee that no packets will be dropped.

For calculating OWD and PDV some additional processing needs to be performed. The timestamp inserted by the transmitter represents the time in seconds and microseconds since 00:00:00 UTC, January 1, 1970. The DAG firmware adds a reception time stamps using the same convention, but the format of the data is rather different. The first half of the 64-bit timestamp represents the second, while the second half represents a fraction of the second with a maximum resolution of  $2^{-32}$  seconds (approximately 233 picoseconds). Consequently, a conversion to microseconds is necessary. This also introduces a certain error in the delay calculation. The card could be used at maximum performance only if the sender would run on a real-time system that can add nanosecond precision timestamps and both ends of the link are synchronized through a high precision common signal, e.g. a GPS receiver.

In case an offline analysis has also been selected, the transmission stamp, DAG timestamp and packet size are stored in a dump file using the same principle used by the capture with Ethernet cards (the filename contains the flow id inside its name). This enables the routine to analyze the dump file using the same function used by the other types of analysis and return the global results of the test. Also, for flow oriented analysis, the QoS parameters will only be computed for the packets that match the predefined requirements: Ethernet frames should have as MAC source address and flow identifier the expected ones, IP packets should originate from the specified IP address and match the flow id and so must UDP datagrams.

For further details about the implementation of the DAG card based analysis and the functioning of the software one can consult the open-sources of the tools or the source code of our module. Basically, our work has selected only the needed piece of code from the tools and modifications have been performed in order to suit our needs. It is very probable that the DAG module would not work on every distribution of Linux as normally the tools need to be compiled for the OS in use. Our module has been though tested on the 2.6 version of the Linux kernel and proves to be reliable.

One last thing that needs to be known when working with the DAG card is that one cannot start multiple simultaneous capture sessions on a single device, as it is not supported by the driver, since no distinction of the incoming flow is initially performed and all the packets receive equal behaviour. In case a manager will try by mistake to start two simultaneous capture sessions on an Endace DAG card, it will be appropriately informed that this is not possible, but this attempt should be rather avoided, since it may results in unpredictable behaviour of the device. Also, when performing consecutive tests, it is recommended to manually empty the device's buffer from time to time. This is necessary because, as we specified before, the packet processing may live some data inside the buffer before closing a stream and this can accumulate in time.

Supposing a synchronisation module is attached to the card, the manufacturer provides a dedicated (`dagpps`) which can be used to prior synchronise the device to the GPS receiver. The Endace DAG card proves to be a powerful tool for network measurements but it still needs serious improvements and its main drawback is the high cost, which unfortunately overpasses the cost of a very good performance system. The provided software is not yet optimized for QoS testing, not all the expected features work as expected and additional configuration and tools are required.

### 3.2.8. The Management Service

Each agent has a management interface that is implemented similarly to the one of the managers. Details about its implementation can be found in [5]. Its purpose is to ensure the communication between the central points of the measurement systems and the agents. The interface is a standardised one, and respects SNMPv1. In order to ensure that the agent is able to receive commands and reply to a manager, several parameters have to be correctly configured on each agent.

First of all, it should be decided which of the available network interfaces are to be used for management purposes. In case several NICs are installed, it is recommended to use separate links for management in order to ensure a good quality of the measurement results, but in case this is not possible, an interface can be used both for management and traffic generation/capturing. For systems that perform measurements using an Endace card, it is obligatory to have an Ethernet controller that will permit the control of the agent. When enabling a device for management, the value for the *polling* and *trap* ports are set to the default values of 161, respectively 162. As notification message have not been implemented the *trap* port is never used but it is specified in order to respect the standard. The port values can be changed at any time but the agent needs to be restarted.

Another thing related to the management interface that needs to be configured in order to allow communication is community names and associated access rights. This provides security, together with the IP filter. The IP filter can contain a list of IP addresses which may be

considered as allowed/restricted. In this manner, only management messages coming from allowed sources will be interpreted.

### 3.2.9. The Message Dispatcher

The message dispatcher has an essential role in transferring the tasks from remote management consoles to each of the services implemented by the distributed agents. The dispatcher is periodically invoked by the management service and its first task is to check for new message in the message inbound queue. In case a message arrived it will be retrieved out of the queue and processed.

Before interpreting a new message, the dispatcher will verify if the message has a known community name. Messages with unknown community names will not be further processed and will be replied with an `SNMP_NO_SUCH_NAME` error message. Assuming the community name exists in the agent's list, the function will verify whether the community name has sufficient access rights in order to perform the requested operation (GET/SET). Commands are sent through SET messages and results or status is collected through GET messages. The agent never sends GET/SET messages, he only replies to managers with `GET_RESPONSE` messages.

When a received message passes the community name and access rights test, the dispatcher will check its MIB (Message Information Base) for the object identifiers and perform appropriate operations. We have defined our own MIB which respects the SMIV2 (Structure of Management Information version 2) standard. The objects are part of the experimental class, having the 1.3.6.1.3 prefix (*iso.org.dod.internet.experimental*). Based on the object identifiers, the agent will know how to interpret the message and in case an unknown OID is received, the sender will be replied with a message having the `SNMP_BAD_VALUE` error status. The hierarchical structure of the defined objects of the experimental class is shown in figure 3.13 and figure 3.14.

Logically, the objects are used for three main purposes: commands, results acquisition and information exchange. An agent can be requested to identify itself by receiving a specific code, and can receive a manager identification or a dismiss object in order to update its local list of management nodes. Other information that can be requested by a manager is related to the network interfaces installed on the system (identifier, physical address, IP address, etc.). This will be used for enabling the engineer performing the tests to choose a certain interface for a specific purpose.

The most important are the traffic related objects. Basically, when a new session is started, a set of objects will be send that will specify the type of session (generation/analysis), the protocol to be used, test duration, identification of the agents performing the test (addresses, port numbers, etc.) time distribution of packets, flow identifier and so on. All those parameters will be gathered by the dispatcher and when all the necessary data was received, it will notify the traffic control service that a new session needs to be started and will pass all the parameters to it. The traffic service will inform the dispatcher about the id of the new process and the dispatcher will wait for this to start. In case some parameters were invalid or some error occurred and the new process failed to start within a certain time interval, the dispatcher will put a message with an appropriate error status in the outbound message queue, otherwise a `GET_RESPONSE` message with `NO_ERROR` status will be sent. It is always important to reply to messages, as the managers expect this and have retransmission mechanism. If responses are not sent, after a while they will consider that the agent failed to reply.

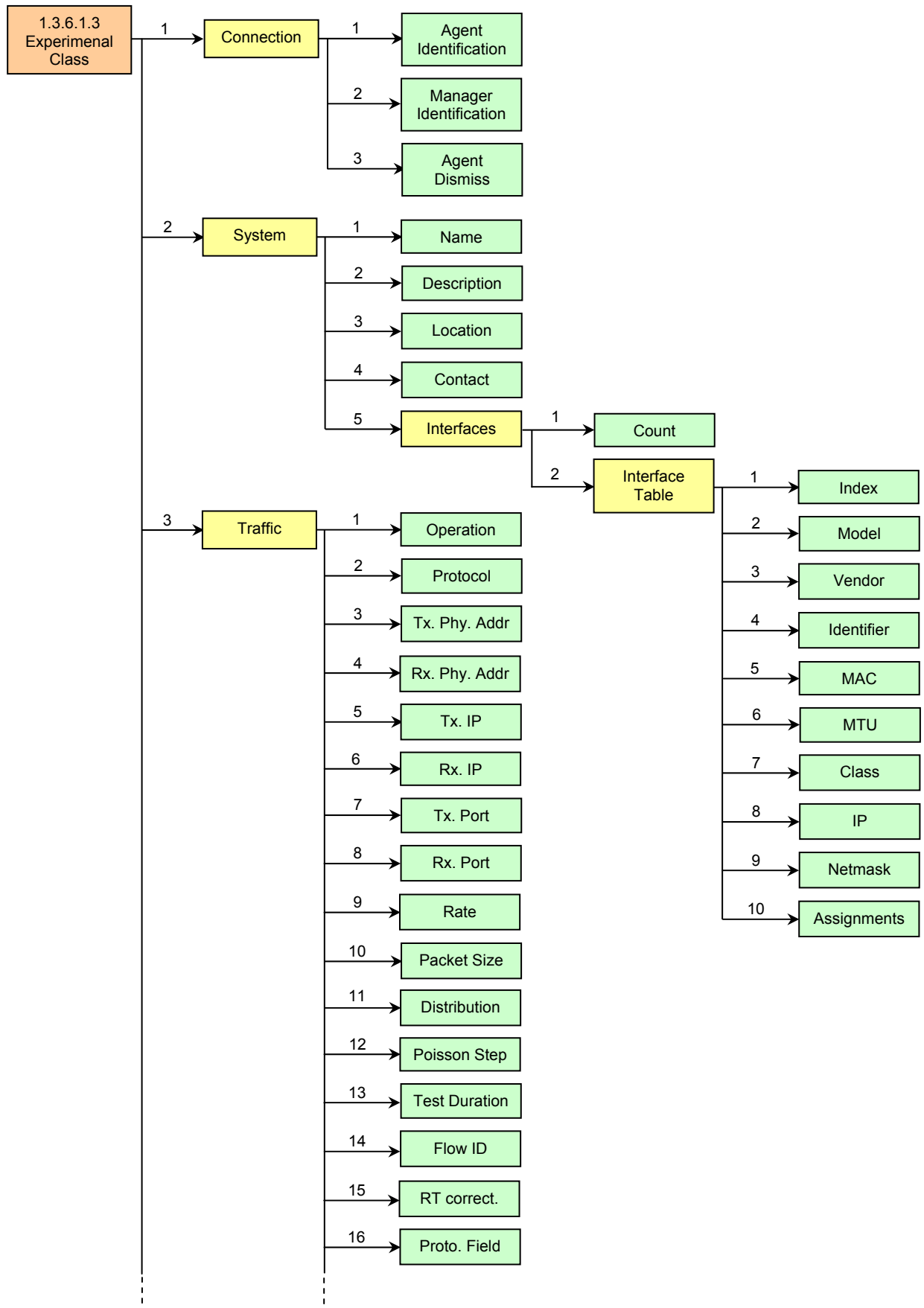


Figure 3.13. The management MIB



Figure 3.14. The management MIB (continued)

The results are collected through GET messages. The first thing that a manager sends when requesting a parameter (packet count, last transmission time, average OWD, throughput, etc.) is the flow id. As there may be several tasks running on an agent, and at the same time, a manager is operating with different agents, the identification of a certain test is done based on the flow id. When an agent receives an object that identifies a flow, it will scan through the list of flows in order to first retrieve the local index of the task. Based on this, it will know what traffic parameters, requested by the message, are to be sent. The received objects have only appropriate identifiers and no data, consequently the data will be automatically filled by the dispatcher, the type will be changed to GET\_RESPONSE and the modified messages will be placed in the

outbound queue. The procedure is exactly similar for operation implying the Endace card, except that the parameters can be automatically retrieved, as only on task can run at a given moment on the DAG device.

For traffic capture/generation session, a safety mechanism has been implemented, meaning that after a task has been completed, data will be kept until the management console acquires the last calculated values or until a time-out expires. This feature is also combined with offline analysis, where it is extremely important to make sure that the data has been collected after the test, as the value is never updated. When an offline analysis has been performed, the manager will periodically enquire the agent at the end of the test whether the statistics have been extracted out of the dump file. The use of this object makes sense as for long tests, the offline analysis make take some time and it would be wrong to treat it like an online analysis, because while the agent processes the trace file, the manager might suppose that it has blocked since parameters are not updated.

Another essential item related to the SNMP message dispatcher is the synchronisation mechanism. The dispatcher uses a global pointer to indicate the location of the memory where the parameters of the new task have been placed. Upon a notification, the main control thread will inspect those parameters process them and pass the necessary information to dedicated routines. In order to ensure that the parameters are not modified while the traffic service still uses them, a global variable is used as a lock, so in case a new message arrives that commands the starting of a new session, the dispatcher will check the ‘lock’ and wait until the other thread has finished his job, if necessary. Typically, when this sort of situations occur, the dispatcher does not need to wait for a long period of time, so the manager will not have to unnecessary perform retransmissions. Actually, after the sockets are created and additional configurations were set, the traffic parameters are copied locally and passed to independent routines, hence, the memory location will be available for new data. As explained in paragraph 3.2.5.1, the traffic control service will itself inspect the ‘lock’ before processing the parameters.

### 3.2.10. Measurement Agent Configuration

In what follows, the paper will focus on presenting an example of a measurement agent configuration procedure. It will be explained how the settings can be made on an agent started with the GUI on and another one in CLI mode. Once the agent software has been installed, no configuration file can be found. Consequently, the agent is not able to communicate with any management console. An agent will be configured using GUI on *Pavilion* and a second one will be remotely configured on *Macabeu* using the CLI. The configuration of the two machines is presented in figure 3.15.

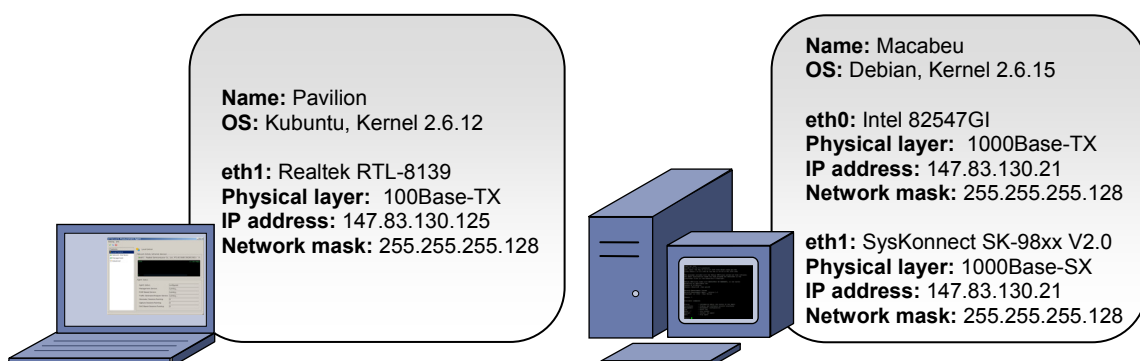


Figure 3.15. Configuration of the machines running the measurement agent

As the configuration file is missing, the GUI based agent will open a dialog window which informs the user that the software failed to load settings and that the software requires reconfiguration in order to operate.

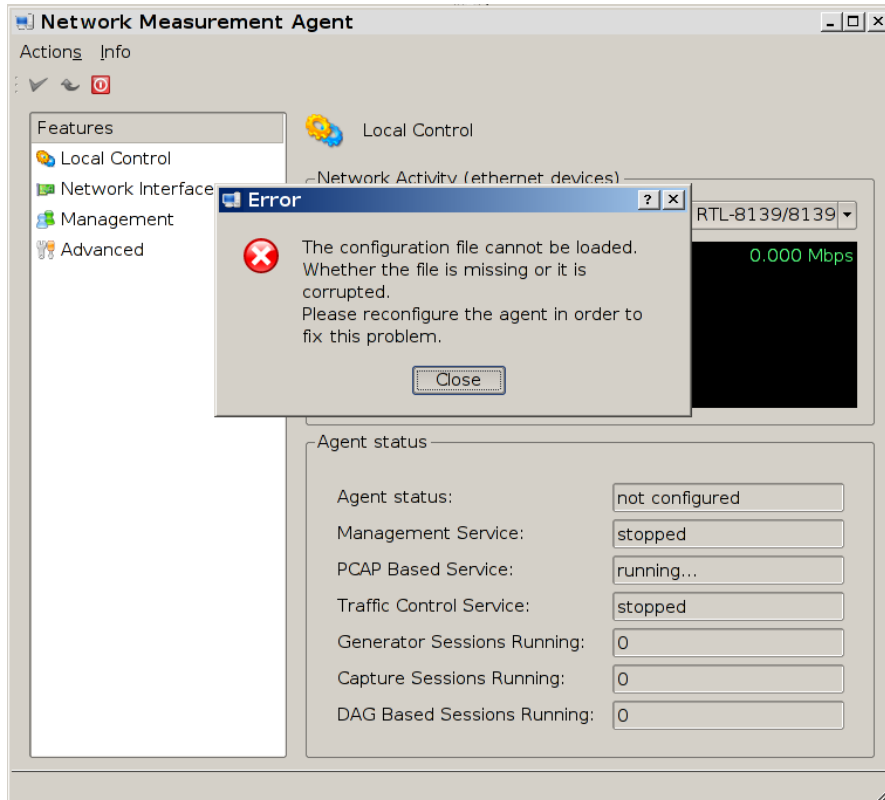


Figure 3.16. First run of a Network Measurement Agent in GUI mode

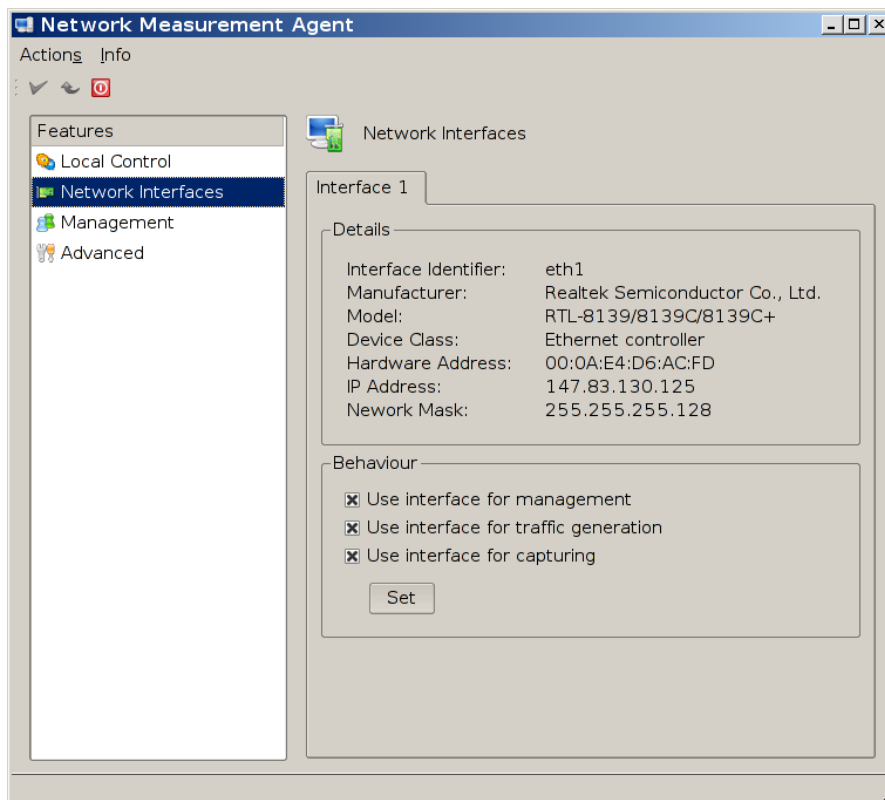


Figure 3.17. Network interfaces assignments

Also, the *Local Control* panel will display suggestive status information, pointing out that the agent is not configured and that traffic and management services have not been started (figure 3.16).

First thing that usually needs to be inspected is the *Network Interfaces* panel. This may inform that no NIC is present or configured or display a separate tab for each installed network controller. First of all, one may inspect the characteristic of each card (identifier, manufacturer, model, device class, hardware address, IP address and network mask).

Then, it is possible to assign various attributes to each card, i.e. specify whether it should be used or not for management, traffic generation and traffic capture purposes through a set of check boxes. Once the *Set* button is pressed, the configuration file will be update and the changes made will be available for future use (figure 3.17).

The most consistent part of the configuration process is related to the management service. The *Management* panel enables the configuration / check of various management related parameters. One can select the *Agent* tab and start editing the identification information by clicking *Change* (figure 3.18). Once the name, description, location and contact have been edited, these will be used to identify the agent to remote management consoles. Similarly, a manager can send such information upon a registration of the agent and the *Managers* list of the agent will be updated (figure 3.19). The list of manager will be also update when a management consoles decide to no longer use a certain agent for measurement and sends a ‘dismiss’ message to it.

The *Network* panel will contain a list of Ethernet controllers that have been previously assigned management functions. For each of them, the interface identifier, IP address, polling and trap ports can be viewed. By default, the port values are set to 161, respectively 162, but they can be changed by selecting the desired interfaces and clicking the *Change Port Settings* button.

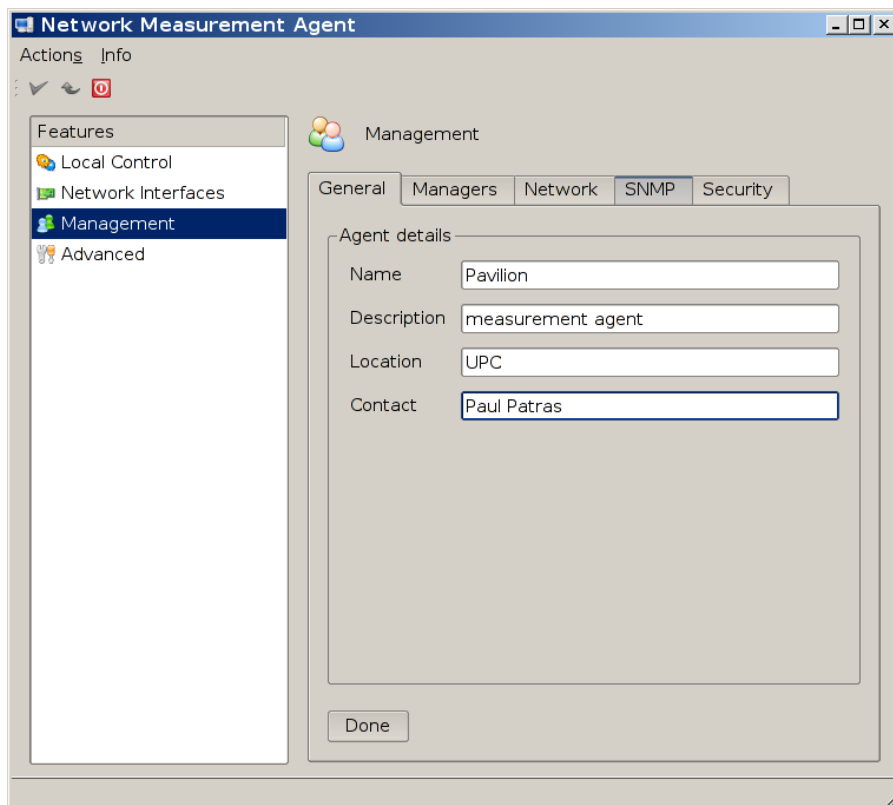


Figure 3.18. Agent details set up

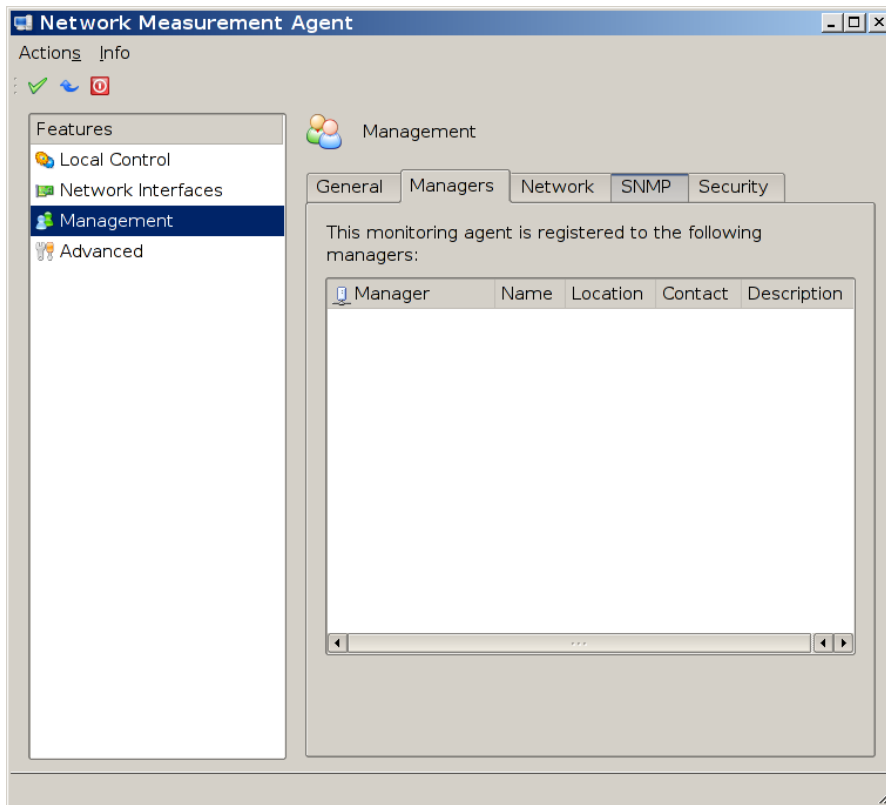


Figure 3.19. The *Managers* tab

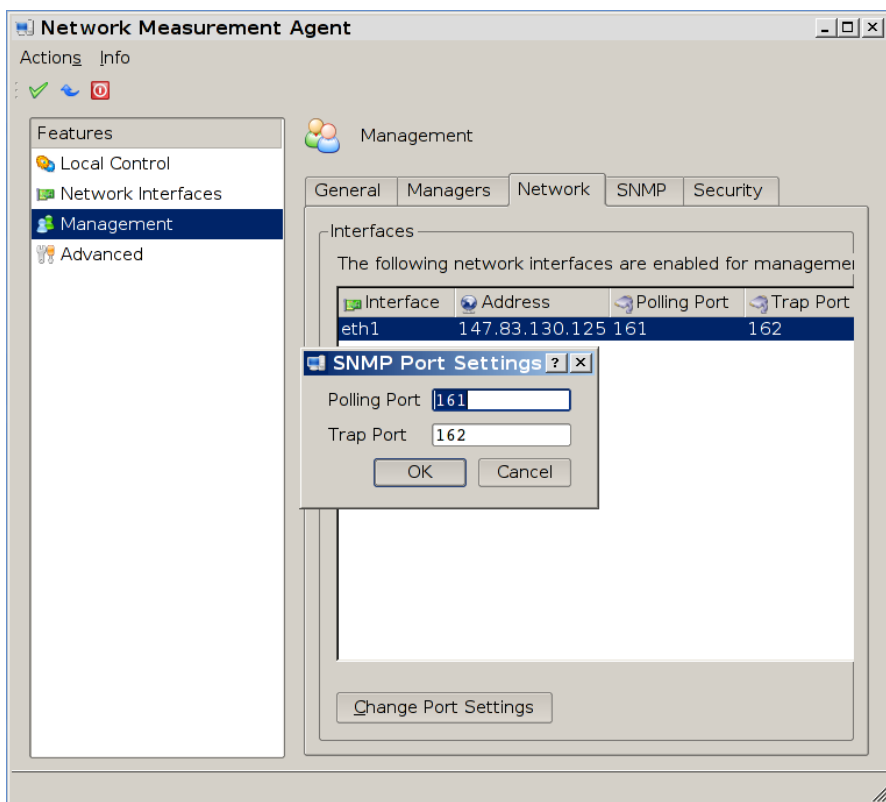


Figure 3.20. Ports configuration for management interfaces

This will open up a dialog which will enable the user to modify the already configured values (figure 3.20). This feature is extremely useful, especially when other management applications are running on the machine or when the same machine runs both an agent and manager software,

in order to avoid *bind* errors and ensure everything operates correctly. It is appropriate to check the error log after restarting the agent with new settings to make sure that such errors did not occur.

In order to allow management consoles to start measurement sessions, appropriate community names and access rights need to be setup. It is possible to add, delete or edit a list of community names that will be used for management as seen in figure 3.21. Only SNMP messages with the known names and sufficient rights will be interpreted by the message dispatcher. When adding a new community name it will be checked first whether this already exists. The access rights can be NONE, NOTIFY, READ ONLY, READ WRITE or READ CREATE.

Another security issue is the IP filter. In figure 3.22 it can be seen that Pavilion has been set up only to accept message from addresses in the list. The picture illustrates how to add a new address using the dialog. The removal of an address is also possible by clicking the *Remove* button on the *Security* panel. One may also change at any time the behaviour of the list (deny / allow) by using the radio buttons.

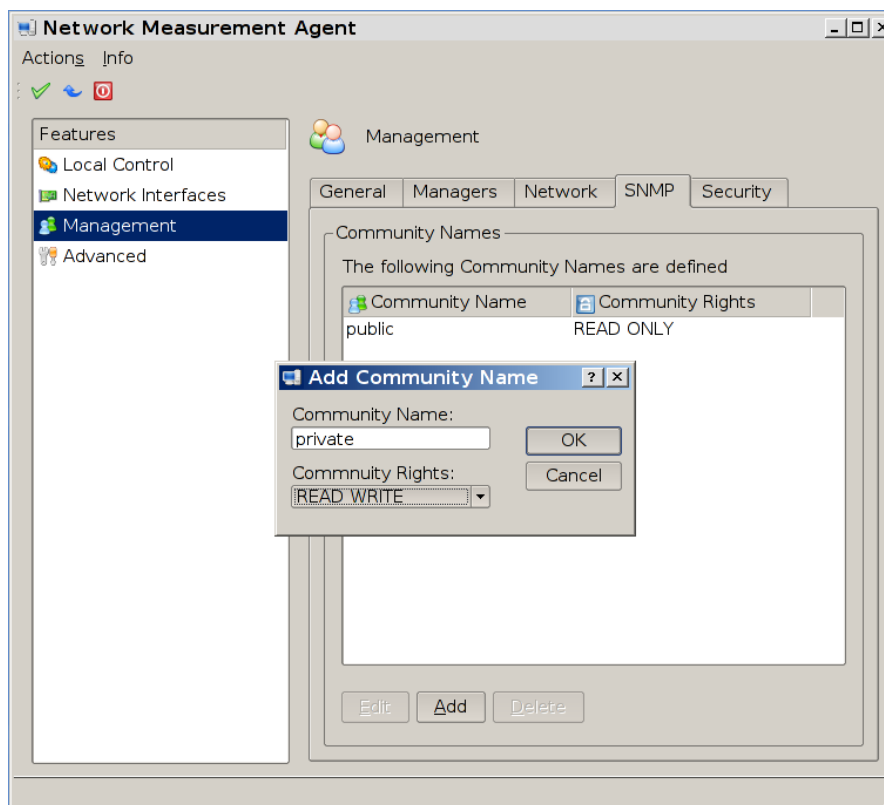


Figure 3.21. SNMP community names list

Once all those settings have been made it is possible to undo the changes or save them into the configuration file by using the buttons in the toolbox of the window or with the help of the *Actions* menu. Once the agent is restarted the settings will be loading and all the services should start.

It is possible to inspect the error log by selecting the *Advanced* item of the features list. When a agent is installed and run for the first time, the only error that is displayed is related to configuration file loading failure, but this list may help an engineer fix problems in case some errors occur. The error log will display for each event the time of occurrence, an appropriate code, the source of the error and a brief description (figure 3.23). The errors can be sorted by these items and the log can be cleared by pressing the *Clear log* button.

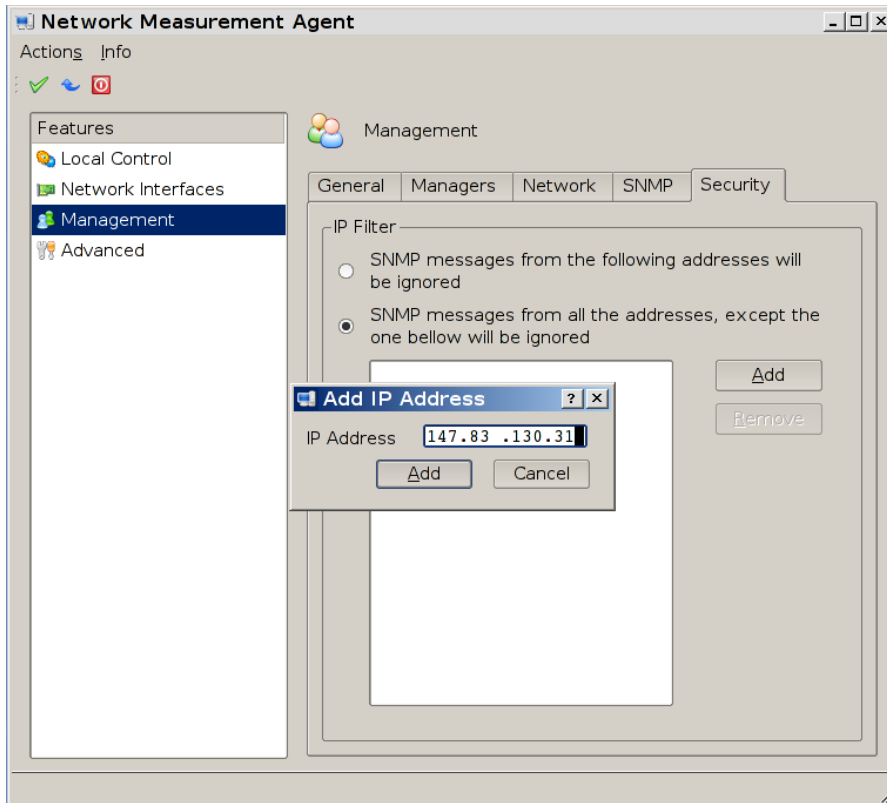


Figure 3.22. The IP filter

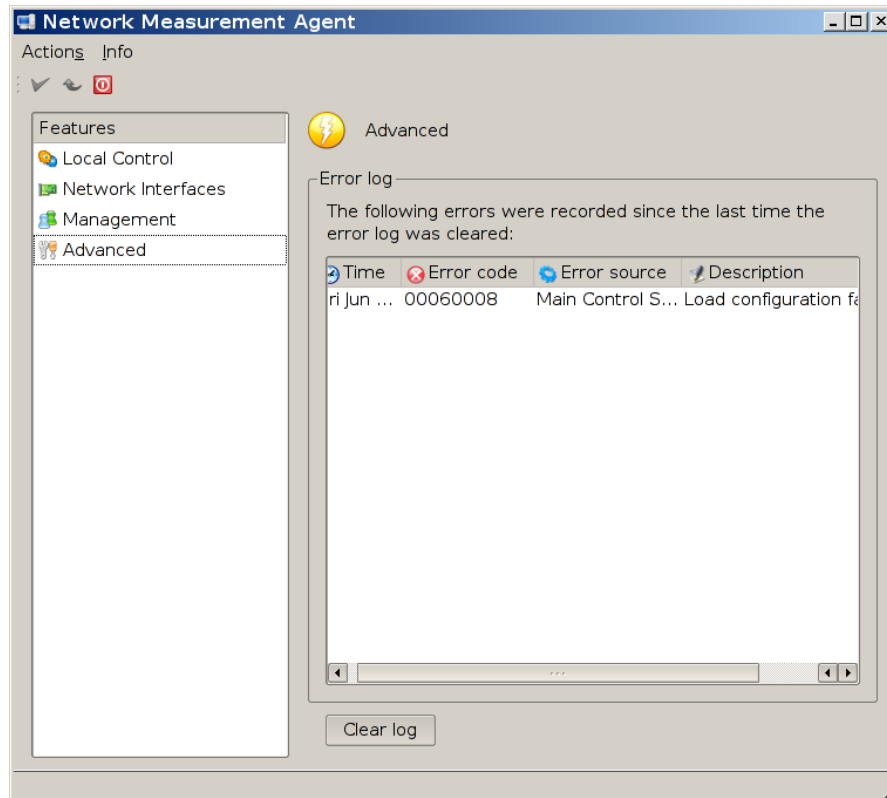


Figure 3.23. The error log

All the settings that have been presented so far can be also performed through a CLI. For example let's assume we have just installed the Network Measurement Agent software on *Macabeu* and we have started a secure shell connection to this machine.

```
root@macabeu:~/Monitor# ./mon guioff

Network Measurement System
Network Measurement Agent - version 1.0
Copyright (c) 2006 - Paul Patras

Warning: Failed to load configuration file
agent: ?
Available commands:

status          - information about the status of the agent
interfaces      - display and configure network interfaces
management     - management configuration
log            - error log
help | ?       - this screen
restart        - restart the agent
exit          - stop agent
```

The agent will be started with the `guioff` command line parameter. As no configuration file is present, the prompter will only consist of the `agent: string`. A set of commands will be available after starting the agent by typing 'help' or '?' as seen in the previous text box.

The `status` command offers almost the same information as the *Local Control* panel of the GUI and can be used to check which services are running or the number of sessions (traffic generation, traffic capture, DAG based) currently started on the agent. The output of the command looks like the following:

```
agent: status

Agent information:

Agent status:          not configured
Management service:   stopped
PCAP based service:   running...
Traffic generator/analyzer service: stopped
Generator sessions running: 0
Capture sessions running: 0
DAG based sessions running: 0
```

In order to inspect the installed network interfaces cards and configure them, the `interfaces` command can be used. This will access the NIC related menu, which will offer a set of specific commands but it is possible to return to the previous menu by typing `back` or to close the agent using the `exit` command. Actually those to commands and the '?' are available in all submenus of the CLI, the question mark displaying the list of commands specific to a certain submenu. One may view the characteristics of all the cards or of a one given by an index, similar to what the *Network Interfaces* panel of the GUI displays. By typing `list` the properties of all the cards are displayed (vendor, model, MAC address, IP address, network mask, device type and assignments) and if the command is followed by a number, the characteristic of the card with that index will be shown, in case the value represents a valid index. An example is presented below.

```
agent: interfaces
agent\interfaces: list
Available network interfaces:
```

```

Interface No. 1
Interface identifier:      eth0
Vendor:                  Intel Corporation
Model:                  82547GI Gigabit Ethernet Controller
Type:                   Ethernet controller
Physical address:        00:30:48:82:88:10
Logical address:         147.83.130.21
Network mask:           255.255.255.128
Used for management:    no
Used for traffic generation: no
Used for traffic capture: no
-----

Interface No. 2
Interface identifier:      eth1
Vendor:                  SysKonnect
Model:                  SK-98xx V2.0 Gigabit Ethernet Adapter
Type:                   Ethernet controller
Physical address:        00:00:5A:9F:4A:38
Logical address:         147.83.131.21
Network mask:           255.255.255.128
Used for management:    no
Used for traffic generation: no
Used for traffic capture: no
-----

```

In order to assign the functions of a card from the Network Measurement Agent point of view it is possible to use the `configure` command, followed by the index of the interface to be configured. At this point that interface can be set to be used or not for management, traffic generation or traffic capture as illustrated in the next example performed on *Macabeu*.

```

agent\interfaces: configure 1
agent\interface\[eth0]: set man=1 gen=1 cap=1

Changes saved.
If any modifications concerning management were made, it is recommended to
restart the agent.
agent\interfaces:
agent\interfaces: configure 2
agent\interface\[eth1]: set man=0 gen=1 cap=1

Changes saved.
If any modifications concerning management were made, it is recommended to
restart the agent.

```

Once the interfaces have been configured, if at least one has been enabled for management purposes, it is possible to set up the parameters used by the management service and the message dispatcher. If no interface is selected for management or the SNMP community names and access rights have not been established, the agent will be inoperable. In this example we have chosen to use the first interface for management purposes, consequently we can enter the management command and start configuring the service. Similar to the GUI, this menu will offer a set of commands for defining the identification of the agent, configure the trap / polling ports, inspect the list of management consoles, define community names or configure the IP filter. The name, location, contact and description of the measurement agent can be defined using the `set` command of the `agent` submenu as exemplified below.

```

agent: management
agent\management: agent
agent\management\agent: set name Macabeu
agent\management\agent: set location UPC
agent\management\agent: set description measurement station
agent\management\agent: set contact Paul Patras

```

After those have been set up, the identification can be viewed through the `show` command:

```
agent\management\agent: show
Name: Macabeu
Location: UPC
Contact: Paul Patras
Description: measurement station
```

From the *management* menu, the list of managers that have registered the running agent can be viewed using the `managers` command. In this example, as the configuration is not yet completed and the agent has not been restarted, the managers list is empty:

```
agent\management\agent: back
agent\management: managers
The agent is not currently registered to any manager.
```

After a manager will register a new measurement agent, the managers list will be update and when launching the `managers` command, for every manager the text interface will display its IP address, name, location, description and contact.

The `net` submenu allows the user to view the configuration of the interfaces used for management or change the port settings of a certain interface as shown below.

```
agent\management: net
agent\management\net: show
Interface number:      1
Interface identifier:  eth0
Logical address:      147.83.130.21
Polling port:         161
Trap port:            162
-----

agent\management\net: set if=1 poll=171 trap=172
Port settings made successfully.
```

The community names and their corresponding access rights can be edited after accessing the `snmp` menu. To add a new community name the `add` command can be used followed by the name of the new community and a number representing the access rights (0 – NONE, 1 – NOTIFY, 2 – READ ONLY, 3 – READ WRITE, 4 – READ CREATE). If a community name already exists, the user will be asked whether he wants to modify the rights of the existing entry. To delete a community name the `del` command can be used, followed by the index of the community name in the list. In this example we have created two community names and then deleted the first one:

```
agent\management: snmp
agent\management\snmp: show
There are no community names available.
agent\management\snmp: add private 3
Community name added.

agent\management\snmp: add public 2
Community name added.

agent\management\snmp: show
1.      private      READ WRITE
2.      public       READ ONLY

agent\management\snmp: del 1
agent\management\snmp: show
1.      public       READ ONLY
```

Finally, it is possible to configure an IP filter in order to accept SNMP messages only from certain addresses. One may choose between accepting all messages except the ones from the addresses in the list or only the messages from a list of known hosts. In our example the agent has been set up to accept messages only from a single address where a management console will be started (Mode 1). The set of available commands for modifying the items and behaviour of the IP filter consists of the following: `show`, `add`, `del` and `mode` and their usage is shown below.

```
agent\management: filter
agent\management\filter: show
The list is empty.
agent\management\filter: add 147.83.130.1
agent\management\filter: show
The messages from the following addresses are ignored (Mode 0):
1) 147.83.130.1

agent\management\filter: mode 1
agent\management\filter: show
Messages from all addresses, except the following, are ignored (Mode 1):
1) 147.83.130.1
```

Coming back to the main menu, another feature is the error log. This can be inspected and cleared after entering the `log` submenu. As for the GUI, the error log entries will contain the date and time of the event, its code, the source of error and a brief description:

```
agent: log
agent\log: show
Fri May 26 16:27:10 2006 0x00060008 Main Control Service Load configuration
failure.
agent\log: clear
Error log was successfully cleared.
```

After the Network Measurement agent has been configured all the settings have been saved. Consequently, after a `restart` it should be able to receive messages from management consoles and start capture or generation sessions. Also, the prompter of the CLI will contain the name assigned to the agent and all the services should be running once the configuration file has been successfully loaded:

```
agent: restart
Network Measurement System
Network Measurement Agent - version 1.0
Copyright (c) 2006 - Paul Patras

Macabeu: status

Agent information:

Agent status:                configured
Management service:         running...
PCAP based service:         running...
Traffic generator/analyzer service: running...
Generator sessions running: 0
Capture sessions running:   0
DAG based sessions running: 0
```

As mentioned before, it is very efficient to start an agent in CLI mode when remotely accessing a machine inside a network, but on the other hand the CLI makes it possible to run the agent on operating systems that have no graphical engine installed. This is very common for boxes serving only for measurement purposes.

### 3.3. EXPERIMENTAL RESULTS

#### 3.3.1. Evaluating the Performances of Some Gigabit Ethernet Cards

One of the first things that determined us to develop this specialized network measurement software was the need of evaluating the performances of a set of Gigabit Ethernet network interface cards. When performing QoS testing on Gigabit links, it is necessary to have an idea of the capabilities of the devices involved in the measurements, in order not to wrongly interpret the obtained results. Actually, except the hardware dedicated cards (such as Endace DAG cards) most of the systems equipped with software NICs seem unable to generate or capture 100% Gigabit line rate. The problem does not lie strictly within the NIC itself but the limitation is imposed by the overall system. Even fast processors are not able to execute the code that generates / captures packets fast enough to ensure that a link is loaded to its maximum capacity. Another factor that limits the performance of a measurement station is the speed of the data bus. Nevertheless, calculating QoS parameters in real-time will slow down the capture process and when the analysis is performed offline the limitation lies within the hard-disk writing speed.

We have tested the performance of some powerful PC systems equipped with Gigabit Ethernet cards and tried to evaluate their performance and determine which is the bottle neck of the system but also the limits of the software. In this paragraph the performances of a system with an Intel® 82545GM Gigabit Ethernet Controller. Performance analysis of systems equipped with other NIC can be found in [5].

Although the card is installed on the PCI bus of the computer it has proved that it has relatively good performances for an ordinary Gigabit Ethernet controller, especially when used for traffic generation. First, the card has been connected through multi-mode optical fibre directly to a DAG card which should capture with high accuracy all the traffic generated. LC connectors have been used at both end of the fibre and Layer 2 frames have been generated between the two interfaces. Two network measurement agents were running on the machines hosting the network controllers and a management console on a third machine was used for test command and control. The synchronisation of the two agent machines has been realized using NTP, both agents being connected to the same NTP server. The experimental setup is presented in figure 3.24.

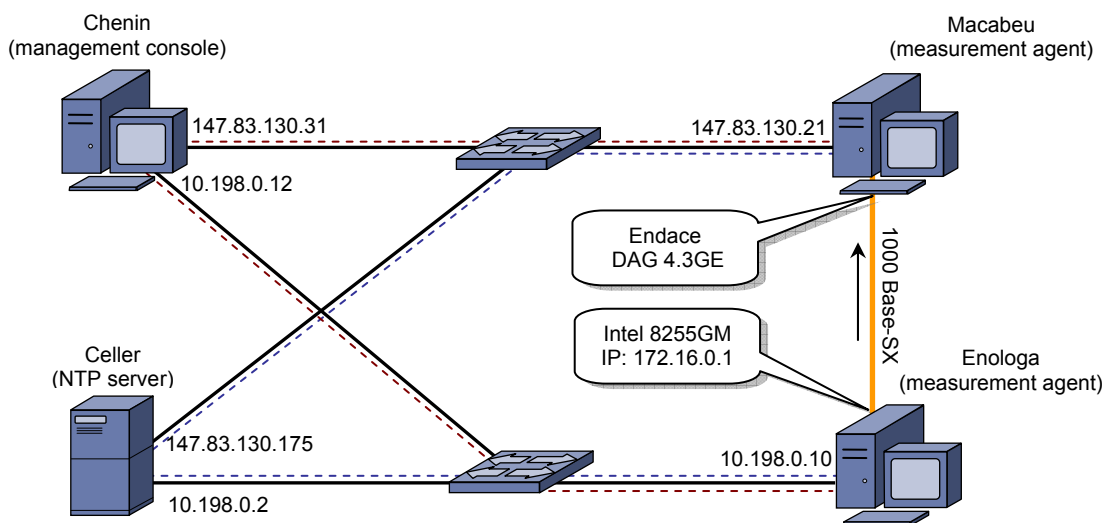


Figure 3.24. First experimental set-up for testing the Intel card

The blue dashed lines represent synchronisation connections and the red dashed lines management connections. We have generated Ethernet frames with the Intel card, repeating the tests for various frame sizes and frame rates in order to determine the QoS parameters. Although the two agents are synchronized using NTP, this method is not very precise, consequently the value of the one-way delay will not be accurate. The main focus was to examine how many frames can be delivered during a second, which is the maximum throughput that can be reached, what is the average value of the packet delay variation and which are its minimum maximum values within a test. Also, we tried to examine at which point and how often real-time errors occur, i.e. when the software is not able to respect the time distribution of the packets. Periodic and Poisson distributed frames have been generated but we also tried to load the link to the maximum capacity. The duration all test was 100 seconds.

When generating periodic small frames (256 bytes) it turned out that the maximum number of packets that can be transmitted over a 100 second test is bellow 10 million (figure 3.25). The x axis represents the value of frames per second that has been passed as parameter to the layer 2 traffic generating routine and y the number of transmitted/received packets.

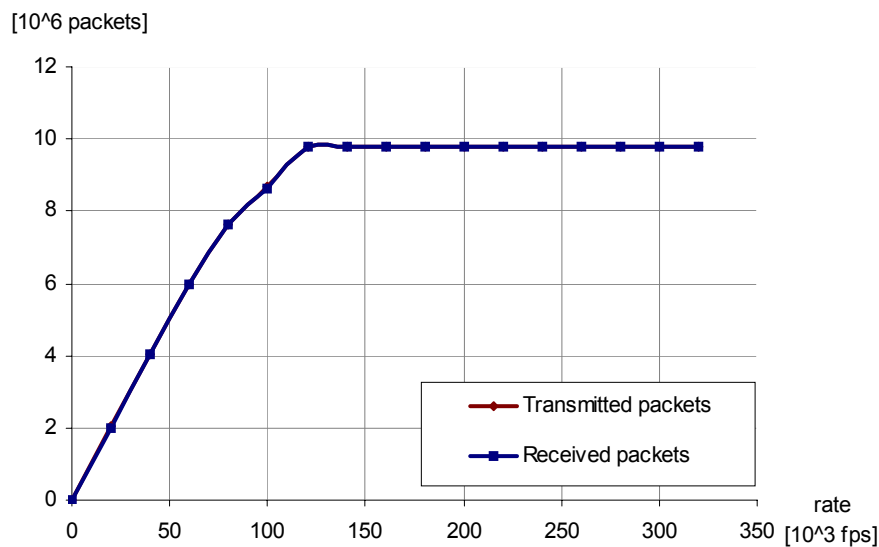


Figure 3.25. Packet count for 256-byte periodic frames

Although one may expect to have exactly the same number of packets at reception as the transmitted one, and the plot may seem to confirm that, in fact some frames are lost. It is sure that they are not lost on the few meter long optical fibre that connects the two NICs. The source of loss lies actually within the software that retrieves the frames from the buffer of the DAG card and computes the QoS parameters. The parameter computation process introduces some processing delay and the buffer of the interfaces may become full at a certain moment when the packet rate is very high and the software does not fetch fast enough all the data. In this situation some frames will be dropped but the packet loss ratio does not exceed 0.3 percent. The packet loss-ratio in this case is shown in figure 3.26.

Another interesting aspect is related to the real-time failures. The NIC may still be able to deliver higher packet rates but at one point the frame generating routine and the system configuration will limit the packet count. As illustrated in figure 3.27, real-time failures occur above 60,000 frames per second but the measurement agent will try to adjust this through one of the selected correction mechanism. But above about 150,000 fps it turns out that the packet distribution can no longer be followed.

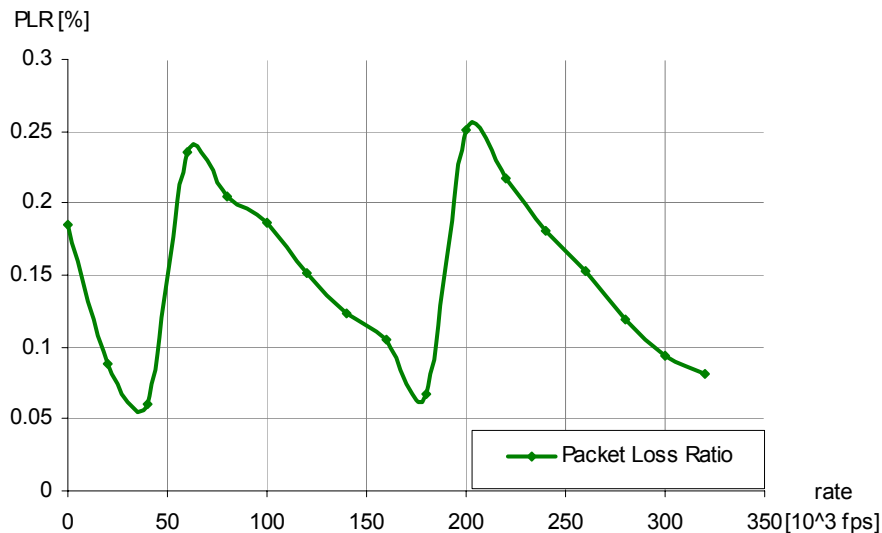


Figure 3.26. Packet loss ratio for 256-byte periodic frames

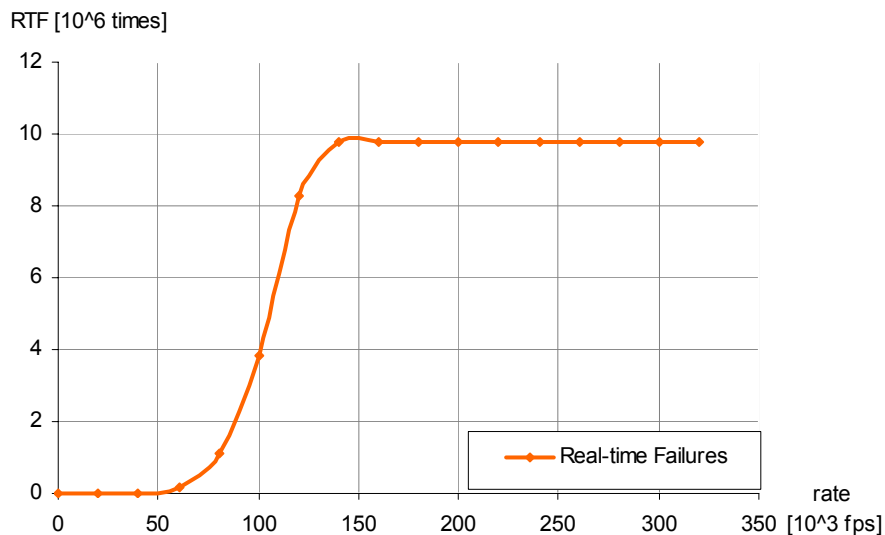


Figure 3.27. Real-time failures for 256-byte periodic frames

Another examined parameter was the jitter. The average value of the packet delay variation has been updated every second of the test but the maximum and minimum values have been computed for the overall test, as we were interested in the worst values rather than the extreme values during shorter periods. As the capture is done very precise and does not introduce any additional delays, the only source of jitter is the transmitter. The Intel 82545GM Gigabit Ethernet controller proves to keep the value of the packet delay variation within hundreds of microseconds (figure 3.28), which is not bad at all, especially for a cheap PCI card.

Maybe one of the most important parameters that we wanted to investigate was the throughput. Actually all the tests were designed in such manner that, at least theoretically, to obtain 1 Gbps throughput regardless the frame size. For small frame high rates are obtained at high packet rates and as the bottleneck is set by the software and the system resources, the performances are poor for 256-byte frames. The maximum achieved throughput was around 200 Mbps (figure 3.29).

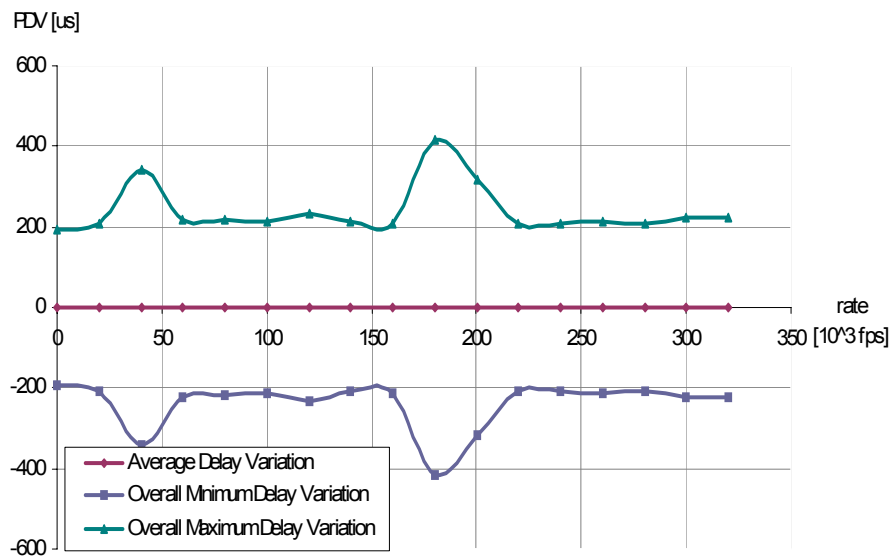


Figure 3.28. Packet delay variation for 256-byte periodic frames

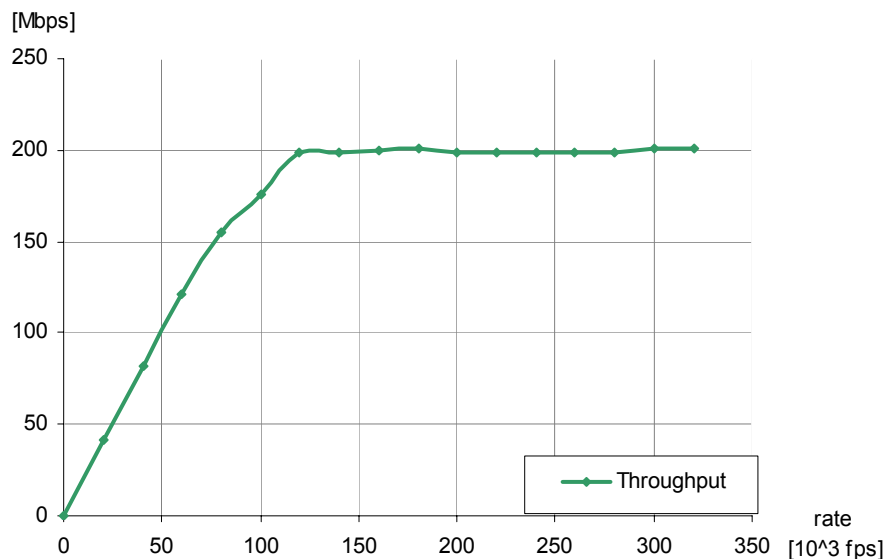


Figure 3.29. Throughput for 256-byte periodic frames

Increasing the size of the frames, the number of packets per second needed to achieve the same rate will decrease, consequently the transmission will become more efficient and a higher throughput will be reached before encountering real-time failures. Whether we chose a periodic or Poisson distribution of the packets, the maximum throughput was limited around 700 Mbps, the maximum delay variation is around one millisecond and the packet loss ratio is around 0.2 percent. The results are presented in figures 2.30 – 2.33 . After performing tests with different frame sizes and varying the packet rate, we may conclude that the best approach when analyzing the performance of a card is to use MTU size frames. On the other hand, the overall system and software performance could be best tested with small frames and high packet rates. Despite our efforts it results that Gigabit testing is still a problem when relying on software network interface cards, but a powerful machine will certainly not influence performance analysis results of Fast Ethernet cards. Unfortunately, dedicated hardware is required for testing at full link capacity and the costs of such devices are extremely high for the moment. It is important to have in mind that synchronisation is essential for accurate measurements and the best solution seems to be the use of GPS receivers.

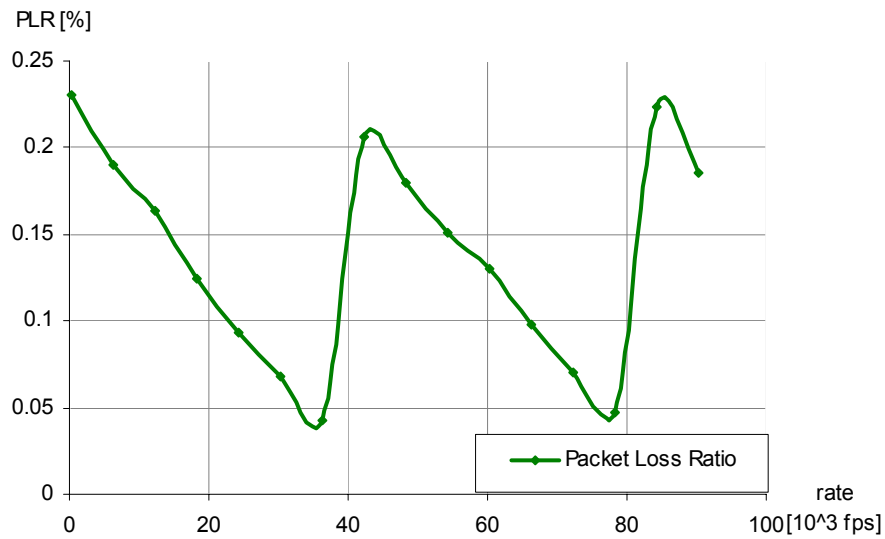


Figure 3.30. Packet loss ratio for 1500-byte Poisson distributed frames

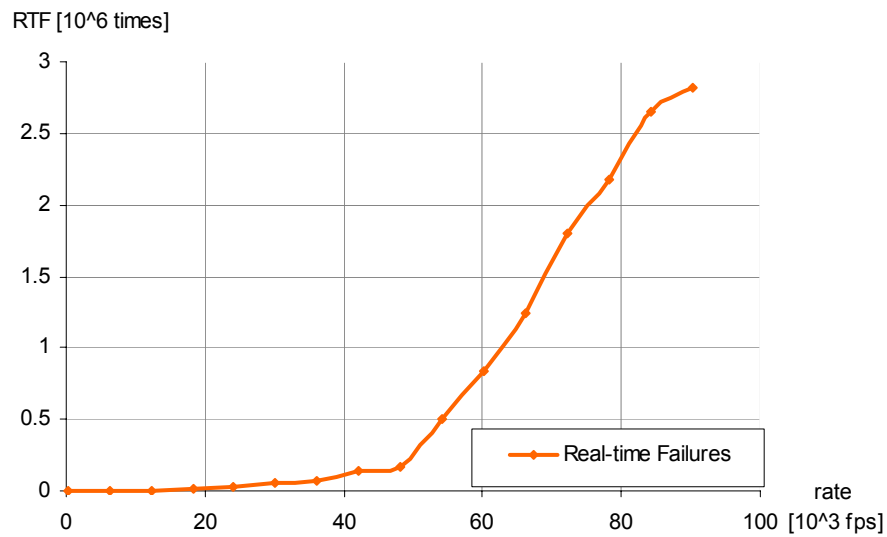


Figure 3.31. Real-time failures for 1500-byte Poisson distributed frames

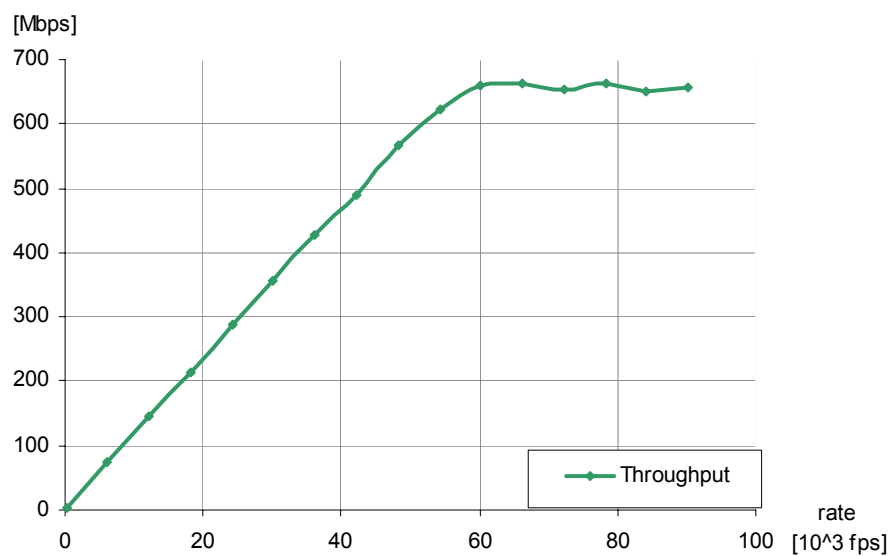


Figure 3.32. Throughput for 1500-byte Poisson distributed frames

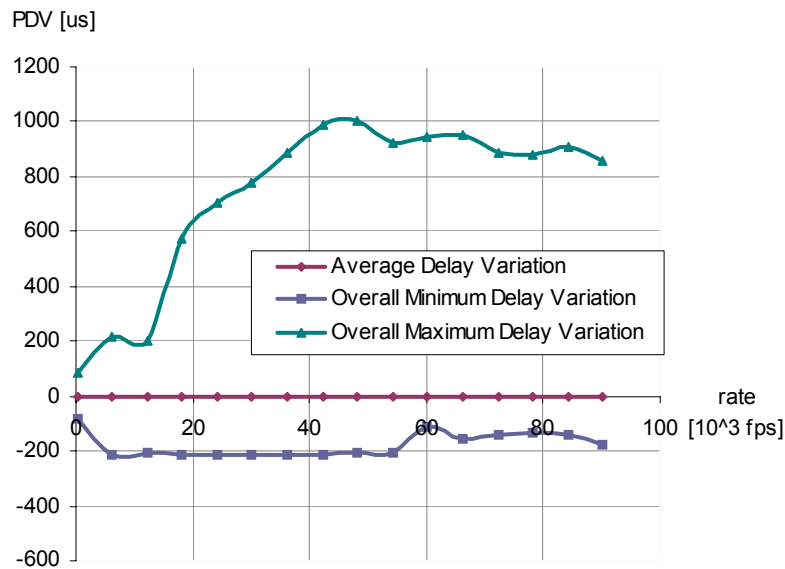


Figure 3.33. Packet delay variation for 1500-byte Poisson distributed frames

In the end we have made an attempt to flood the link with packets of different sizes. The measurements turned out that the software is able to send around 60,000 frames per second and the maximum throughput is achieved for 1500-byte frames, its value being approximately 670 Mbps. The plot of the throughput for a 100 seconds flood test with MTU size frames is shown in figure 3.34.

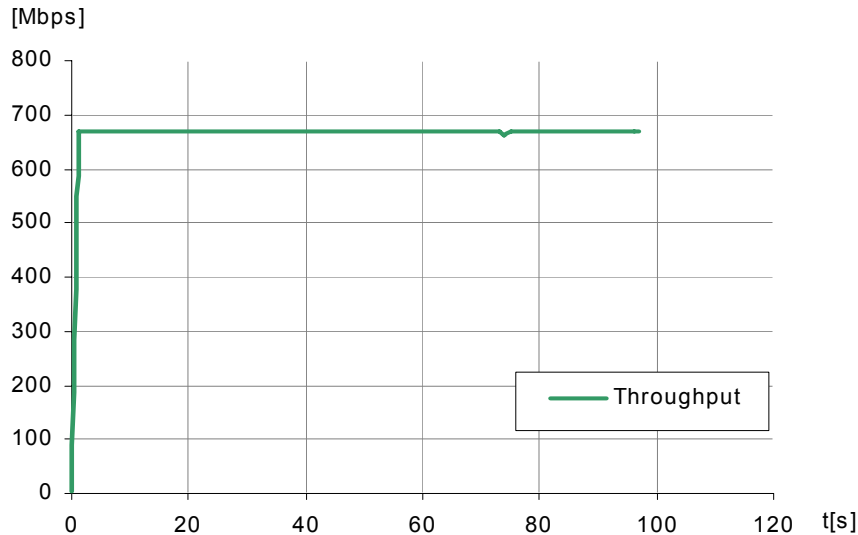


Figure 3.34. Throughput for link flood with 1500-byte frames

In the second scenario we generated traffic with another Gigabit Ethernet card and tried to analyze the frame using the Intel 82545GM controller. The generating interfaces was SysKconnect SK-98xx V2.0 Gigabit Ethernet adapter which was connected on the PCI-X bus of a computer and also some enhanced drivers have been installed for it to ensure maximum transmitted throughput possible. The ideal case would have been to be able to generate Ethernet frames with an Endace card but the model we had under testing failed to provide this feature. The experimental set up of the second set of tests is presented in figure 3.35. As in the previous experiment, 256, 512, 768, 1024 and 1500 byte frames were generate with different rates.

Ethernet traffic has been chosen in order to minimize the additional delays introduced by encapsulation / decapsulation performed by the operating system for higher level protocols.

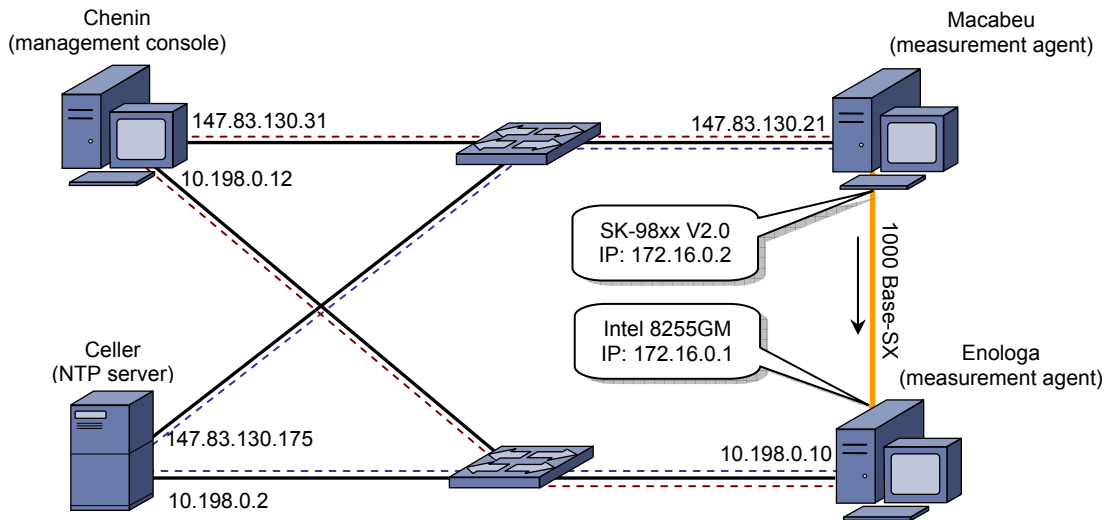


Figure 3.35. Second experimental set-up for testing the Intel card

In this situation we could not say that only the capturing performance of the Intel card are tested but also the generating capabilities of the SysKconnect adapter. In the second experiment the analysis is performed after reading layer 2 socket descriptors and it turns out that the packet loss ratio is zero almost all the time, very few frames being occasionally lost perhaps due to reception buffer overflows. Another thing that was noticed is that the traffic becomes bursty and out-of-order packets are received.

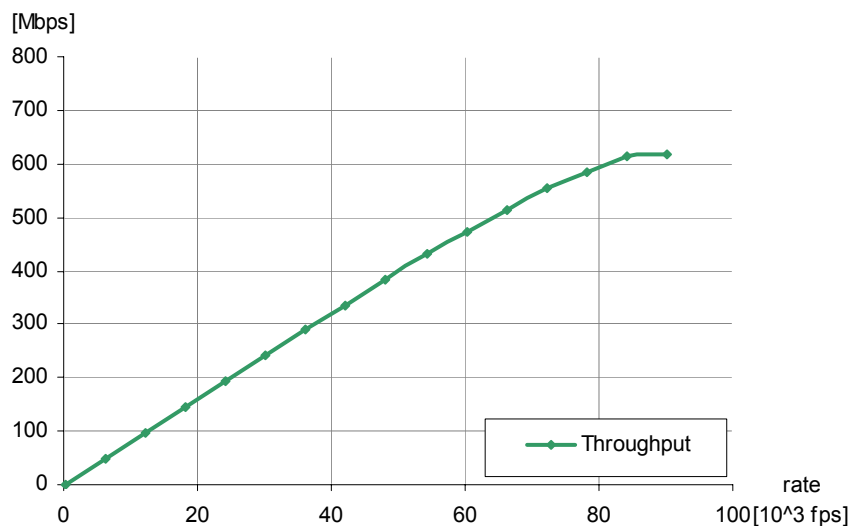


Figure 3.36. Throughput for 1500-byte periodic frames

The jitter has higher values but this is caused by both ends of the connection. Although the maximum values of the jitter can reach a couple of milliseconds, the average packet delay variation is zero. Overall, the Intel card proves to be able to analyse traffic up to 620 Mbps and has good performances for a PCI network device. Figure 3.36 shows the plot of the throughput for a set of tests with 1500-byte periodic frames for packet rates up to 90200 fps. Out-of-order packets are shown in figure 3.37 and the packet delay variation is plotted in figure 3.38.

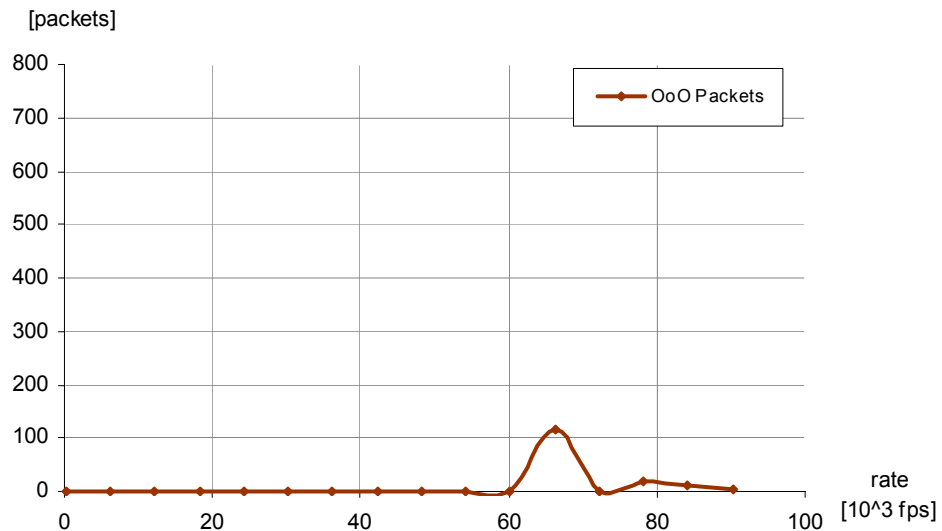


Figure 3.37. Out-of-order packets for 1500-byte periodic frames

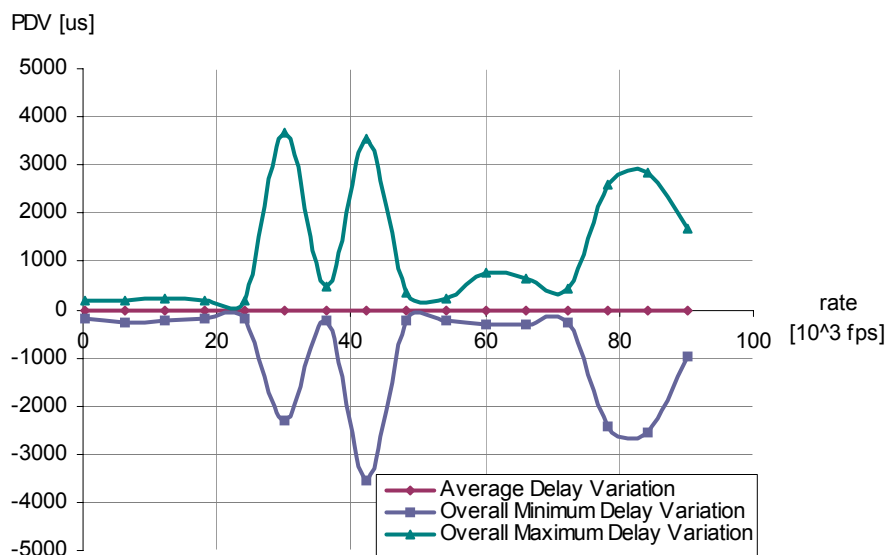


Figure 3.38. Packet delay variation for 1500-byte periodic frames

### 3.3.2. Comparing the Network Measurement Agent with other Tools

The first approach for evaluating the performances of the Intel card involved an experimental set-up similar to the one in figure 3.24 but the testing procedure was a little bit more complicated. The management connections were at that time just simple secure shell connections to the test machines. On the computer hosting the Intel card we wrote some traffic scripts and used them with MGEN to generate UDP traffic. MGEN is a powerful packet generator which is also able to work as a periodic or Poisson source. On the other end, the traditional tools of the DAG card have been used to capture all the frames and after the test has been completed the dump file has been analyzed offline to extract the QoS parameters. The measurement procedure involved multiple steps, a lot of additional operation and permanent assistance.

First the *dagsnap* tool need to be started, then MGEN and the test had to be assisted until completion. After the test was finished additional software had to be run to compute the

parameters. Another disadvantage of the approach was that additional configuration was required at the transmitter (static ARP entry). Nevertheless, the measurements demanded a lot of time to be performed.

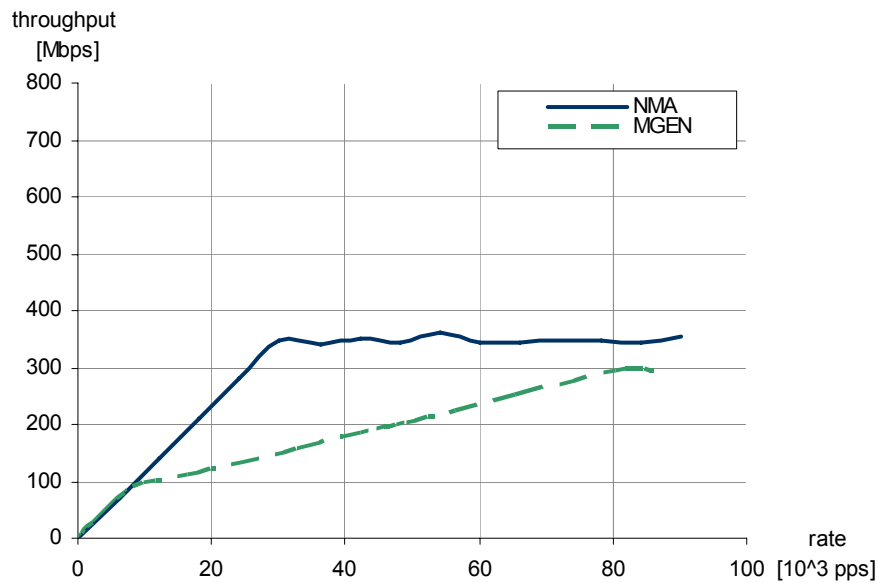


Figure 3.39. NMA vs MGEN performances

After developing the Network Measurement System, we performed the experimental setup shown in figure 3.35 and UDP datagrams were sent with the Intel card to the SysKconnect card. Although the capture has been done with a dedicated card in the first scenario it turns out that the limitations lie at the transmitting end. Multiple tests have been run with MGEN and with the Network Measurement Agents using UDP traffic with different datagram size (i.e. different Ethernet frame size) and increasing the packet rate up to theoretically reaching 1 Gbps. The results have shown that the performances of our tool are comparable with the performances of MGEN. For small packet size, MGEN proves slightly better performances than the developed measurement agents, but at larger packet size the traffic generation capabilities of the agent seem to be a little bit better than those of MGEN.

Another major advantage of the developed software compared to the classical tools is that the set of measurements can be automated, the results are collected in real-time and the software requires no human attendance during the tests. Also, the results are presented in a user-friendly manner and can be exported for later use.

# 4

## CONCLUSIONS

Network measurement plays an essential role in the attempt to characterise Ethernet traffic and provide Quality of Service in next generation networks. Despite numerous projects in this field the measurement process still involves advanced skills in operating various software and it is a time consuming task. This project aimed to summarise the theoretical fundamentals of Internet measurement and monitoring and proposed a complete solution for evaluating the performance of high speed links and network interface cards.

The developed software is part of a distributed network measurement systems, which aims to provide QoS parameters measurement facilities, is able to remotely control multiple tasks, gather the results and present them in a professional manner with as little user assistance as possible. The measurement system is composed of a management console that coordinates the measurements and collects the results from a single point and several measurement agents placed on remote machines. The project was focused on the implementation of these network measurement agents.

The agent software is capable of generating and analysing Data Link, Network and Transport Layer protocol data units, but it also provides facilities for full link monitoring and a module for performing QoS measurement using some dedicated hardware. The performances of some Gigabit Ethernet cards have been evaluated and the developed tool was compared with similar applications.

In order to provide accurate results the developed application will require future testing and improvements. For evaluating the QoS parameters of networks under test, the machines running the distributed agents should be very closely synchronised and if possible GPS should be used.

Although we have tried to improve the software of some dedicated capture cards, it turns out that future work should be carried on such that optimisation is achieved. Initially, the measurements performed with an Endace card could only be interpreted off-line. The proposed solution enables real-time QoS measurements but it turns out that for long tests at high rates, occasionally packets are dropped. This is due to the fact that the packet processing time is not short enough and buffers can overflow. Also, the parts of the software are resource consuming due to the fact that precise execution thread suspension could not be implemented on the development platforms. Future work could be performed to transpose the agent software on real-time operating system and optimise it in such manner that less processor time will be required.

## REFERENCES

- [1] T. Zseby, F. Schreiner, "QoS Monitoring and Measurement Benchmarking", IST-2000-26418, 2002
- [2] I. Miloucheva, A. Nassri, U. Hofmann, "Traffic Measurement and Monitoring Roadmap", IST-2000-26418, 2002
- [3] A. Beben, W. Burakowski, M. Dabrowski, "Reference Measurement Points for validation end-to-end QoS in heterogeneous multiple domain network", Warsaw University of Technology, 2003
- [4] Cisco Systems, "Cisco IOS 12.0 Quality of Service", Cisco Press, Indianapolis, 1999.
- [5] A. Bikfalvi, "The Management Infrastructure of a Network Measurement System", Diploma Project, 2006
- [6] V. Dobrotă, "Digital Networks in Telecommunications" Vol. III, Mediamira Publishers, Cluj-Napoca, 2002
- [7] G. Huston, "Next Steps for the IP QoS Architecture" (RFC 2990), 2000
- [8] Cooperative Association for Internet Data Analysis (CAIDA), <http://www.caida.org/tools/measurement/>
- [9] R. Serral, "NetMeter: graphical representation details", UPC Barcelona, 2004
- [10] The Multi-Generator Toolset (MGEN), <http://mgen.pf.itd.nrl.navy.mil>
- [11] S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service" (RFC 2212), 1997
- [12] Microsoft TechNet, "QoS Technical Reference", 2003, <http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/TechRef/dc9b7edf-e7bc-4bd6-af42-afcd0acb415c.msp>
- [13] G. Almes, S. Kalidindi M. Zekauskas, "A One-way Delay Metric for IPPM" (RFC 2679), 1999
- [14] G. Almes, S. Kalidindi M. Zekauskas, "A Round-trip Delay Metric for IPPM" (RFC 2681), 1999
- [15] C. Demichelis, P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics" (RFC 3393), 2002
- [16] G. Almes, S. Kalidindi M. Zekauskas, "A One-way Packet Loss Metric for IPPM" (RFC 2680), 1999
- [17] J. Mahdavi, V. Paxson, "IPPM Metrics for Measuring Connectivity" (RFC 2678), 1999
- [18] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification" (RFC 2460), 1998
- [19] M. Dabrowski, A. Beben, P. Owezarski, X. Masip, R. Serral Gracia, G. Garciade Blas, EuQoS and Measurements, EE QoS Workshop, Paris, 22 June 2005
- [20] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, "Framework for IP Performance Metrics" (RFC 2330), 1998
- [21] D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis" (RFC 1305), 1992
- [22] P. Dana, "Global Positioning System Overview", [http://www.colorado.edu/geography/gcraft/notes/gps/gps\\_f.html](http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html)

## APPENDIX A

Abbreviations used in this project:

ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BMWG	Benchmarking Methodology Working Group
CAR	Committed Access Rate
CBWFQ	Class-based Weighted Fair Queuing
CLI	Command Line Interface
CPU	Central Processing Unit
CQ	Custom Queuing
DF	Don't Fragment
DiffServ	Differentiated Services
DNS	Domain Name System
DSCP	DiffServ Code Point
ECN	Explicit Congestion Notification
ERF	Extensible Record Format
FCS	Frame Check Sequence
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IHL	Internet Header Length
IntServ	Integrated Services
IPPM	IP Performance Metrics
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPX	Internetwork Packet Exchange
IRQ	Interrupt Request
ISP	Internet Service Provider
ISSLOW	Integrated Services System for LOW bit-rate lines
LAN	Local Area Network
LED	Light Emitting Diode
MAC	Media Access Control
MC	Measurement Controller
MGEN	Multi-generator toolset
MIB	Message Information Base
MP	Measurement Point
MPLS	Multi Protocol Label Switching
MTU	Maximum Transmission Unit
NBAR	Network-based Application Recognition
NIC	Network Interface Card
NMA	Network Measurement Agent
NTP	Network Time Protocol

OID	Object Identifier
OoO	Out-of-Order
OS	Operating System
OSI	Open System Interconnection
OWD	One-way delay
PCI	Peripheral Component Interconnect
PCI-X	Peripheral Component Interconnect Extended
PDU	Protocol Data Unit
PDV	Packet Delay Variation
PHB	Per-hop Behaviour
PLR	Packet Loss Ratio
PQ	Priority Queuing
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RSVP	Resource Reservation Protocol
RTP	Real Time Protocol
RTT	Round-trip Time
SCSI	Small Computer System Interface
SLA	Service Level Agreement
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Network
SSH	Secure Shell
TCP	Transport Control Protocol
ToS	Type of Service
TTL	Time-to-live
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
VLAN	Virtual LAN
WAN	Wide Area Network
WFQ	Weighted Fair Queuing