Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 2, 2006

Software Tool for Passive Real-Time Measurement of QoS Parameters

Mihai Vlad [1], Ionut Sandu [1], Virgil Dobrota [2], Ionut Trestian [2], Jordi Domingo-Pascual [3]

Abstract – The paper presents the designing of a software tool for real-time measurement of the following quality of service parameters: one-way delay, average one-way delay, IP packet delay variation and average IP packet delay variation. The solution is an improved version of OreNETa (One-way delay REaltime NETwork Analyzer), by optimizing the traffic between the meter and the analyzer. When a new flow is detected, the meter assemblies a flow descriptor and sends it to the analyzer. Following the flow recording, it will announce the meter to send a shorter message, called header, for all the packets belonging to the newly registered flow.

Keywords: measurement tool, OreNETa, QoS parameters

I. INTRODUCTION

A major step toward the next generation networks is to implement the quality of service mechanisms for IP parameters. The work carried out in this paper is related to the FP6 European project *EUQoS,* focused on end-to-end quality of service support over heterogeneous networks. Its main objectives address: a) the standardization of end-to-end QoS issues in European and International bodies (especially the IETF); b) promoting the creation of new business models to enable the deployment of QoS applications by the Internet community and; c) foster the interoperability of end-to-end QoS solutions for the end user, across heterogeneous research, scientific and industrial network domains [1]. A flexible and secure QoS assurance system could be validated within EUQoS by using the herein proposed software tool for passive real-time measurement. Moreover, this application can be used in monitoring the SLA (Service Level Agreement) between partners and spot some errors during the testing phase. The initial functionalities of the Abel Navaro's *OreNETa (One-way delay REaltime NETwork Analyzer),* described in [2], were extended. The new proposed version passively captures the traffic already existing on a network and it measures a series of QoS parameters (one-way delay, IP packet delay variation) in real-time. It also logs all the captured data for offline processing. The passive measurements were chosen because they provide information about the existing current traffic within the network section investigated. Since no test traffic is generated, they can be applied for most applications where statements about the actual situation in the network are required (like SLA validation, traffic engineering). Active measurements can always be applied supplementary, in order to predict the future network situation during times where no regular traffic is transmitted. The reliability and quality of the link can be expressed in terms of number of packets lost too. Every time a packet belonging to a flow does not reach its destination, a counter is incremented to express the packet loss.

II. QUALITY OF SERVICE PARAMETERS

The QoS parameters that are intended to be measured herein using the proposed software tool are the following: one-way delay, average one-way delay, IP packet delay variation, average IP packet delay variation and packet loss. OWD (One-Way Delay) represents the time that takes a packet to travel through the network from source to destination, which means the time passing between the moment when the *first* bit of the packet leaves the source host and the moment when the *last* bit of the same packet reaches the destination host. This definition can be expressed mathematically by:

$$OWD_i = t_{1i} - t_{0i}\text{, for } 1 \le i \le N\ . \qquad (1)$$

where *N* is the total number of packets belonging to a flow. Fig. 1 illustrates the one-way delay for an *n*-byte packet traversing a network segment. The same

[1] Alcatel Romania, Strada Gheorghe Lazar 9, 300081 Timisoara, Romania, e-mail {Mihai.Vlad, Ionut.Sandu}@alcatel.ro
[2] Technical University of Cluj-Napoca, Communications Department, Strada George Baritiu 26-28, 400027 Cluj-Napoca, Romania, e-mail {Virgil.Dobrota, Ionut.Trestian}@com.utcluj.ro
[3] Universitat Politecnica de Catalunya, Jordi Girona 1-3, Barcelona, Spain, e-mail Jordi.Domingo@ac.upc.es

packet i sent at t_{0i} by the source is received at t_{1i} by the destination.

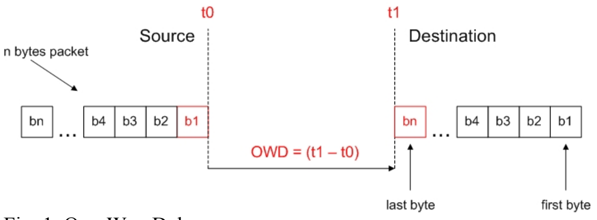


Fig. 1. One-Way Delay

AOWD (Average OWD) could be computed as follows:

$$AOWD = \frac{\sum_{i=1}^{N} OWD_i}{N} \,. \qquad (2)$$

The term "jitter" refers to the variation of a parameter with respect to some reference parameter. A definition of IPDV (IP Packet Delay Variation), also referred to as *delay jitter,* can be given for packets inside a stream of packets. The IPDV is defined for a given pair of consecutive packets within the stream going from measurement point *MP1* to measurement point *MP2*. It is actually the difference between the one-way-delays of two consecutive packets.

$$IPDVi = OWD(i-1) - OWDi \text{, for } 1 < i \leq N \,. \qquad (3)$$

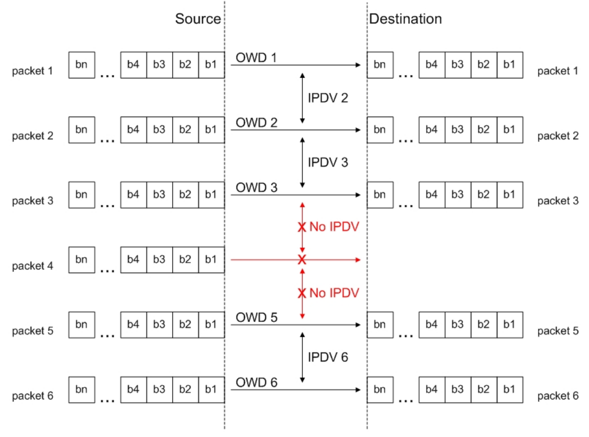


Fig. 2. IP Packet Delay Variation

In Fig. 2 the source is *MP1*, whilst *MP2* is the destination. If a packet is lost (e.g. packet 4), the IPDV (with respect to its adjacent packets) cannot be computed. Similar to one-way delay, AIPDV (Average IPDV) can be calculated as:

$$AIPDV = \frac{\sum_{i=2}^{N} IPDVi}{N} \,. \qquad (4)$$

III. DESIGNING OF THE MEASUREMENT TOOL

The main building blocks for implementing a measurement tool are shown in Fig. 3. The processes involved are packet capturing, time-stamping, generation of flow ID, classification, generation of a packet ID and transfer of measurement data. Each of these processes adds a piece of information to the final message sent to the control application.

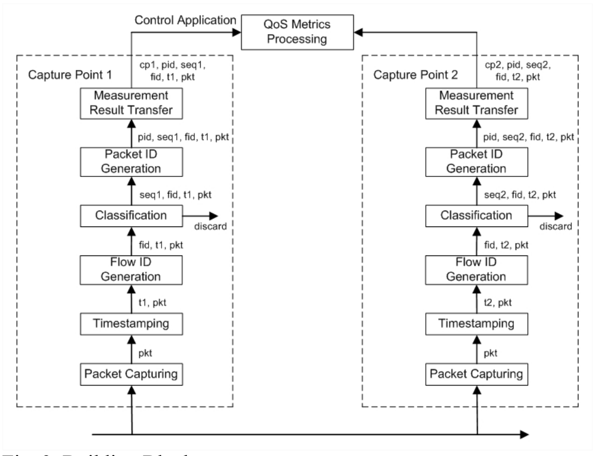


Fig. 3. Building Blocks

A. *Packet Capturing*

A certain amount of bytes needs to be captured per packet as basis for the generation of a packet ID. The packet ID collision probability (see subsection E) depends on the generation function and the number of bytes that are used as input. The first 40 Bytes starting at the IP header are considered to be sufficient for this purpose. However, the number of bytes that can be captured also depends on the processing power remaining for the measurement task. The packet capturing performance of a machine is limited by the following parameters: number of interrupts generated by the NIC; number of context switches; amount of bytes transferred to user space; and current load of the machine caused by other processes (e.g. packet ID generation) [3].

B. *Time-stamping*

A number of issues have to be considered for the basic function of assigning timestamps to packets for subsequent delay calculation. Internal buffering in the hardware and on the way through the kernel causes additional packet delay. Even if all involved measurement devices are equipped with the same hardware and operating system, packets can experience different delays (e.g. due to CPU load and the level of buffer filling). In order to reduce effects from additional variable delays, the timestamp should be assigned to the packet as early as possible. A further problem that has to be solved when using two measurement points is clock synchronization between both points. Best results are based on GPS (Global Positioning System).

C. *Flow ID Generation*

A flow is a sequence of packets sent from the same source to the same destination and receiving the same level of service from the nodes. In order to obtain the same ID for every packet belonging to a given flow we must refer to a combination of fields found within Layer 3 and Layer 4 headers [4]. A similar process can be found in routers also, and is often referred to as flow identification. Since there is no specific information in the packet header to indicate if a packet belongs to a given flow or not, flow identification must be performed on every packet. However, modern routers must support wire-speed forwarding. To support that, a router must cope with the worst-case scenario rather than the average one. In the worst case, multiple packets that belong to reserved flows may arrive at the speed of the incoming link; that is, packets arrive back-to-back. A router must be able to deal with such a scenario.

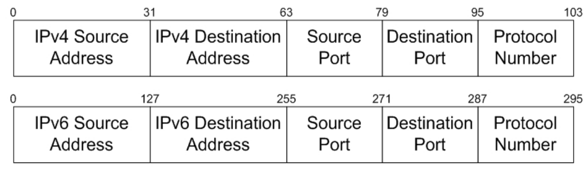


Fig. 4. IPv4 and IPv6 Five-Tuple

At high speeds the per-packet processing time is extremely small. For example, to support 64-byte packets at OC12 speed (622 Mbps), the per-packet processing time is less than 1 microsecond. Service providers are expected to upgrade their backbone to OC48 (2.5 Gbps) and OC192 (10 Gbps) soon. As the Internet expands, the number of concurrent flows can also be very large. An OC12 backbone trunk currently may have tens of thousands of concurrent flows. Thus the design of a flow identification module must be able to perform lookup at high speeds with a large number of flows [4].

D. *Classification*

After the flow ID has been generated for the captured packet we are able to take some decisions based on the flow identified by the new flow ID. There can be a discussion upon where to place the classification in the processing chain; before time-stamping or after it. Placing the classification before the time-stamping will introduce a variable delay equal to the classification processing time. Depending on the size of the classification table, this delay can be significant, and will lead to erroneous measurements. The benefit is that the timestamps will be generated only for relevant packets (i.e. with proper flow ID). The best option is to go for an early time-stamping to obtain accurate results.

E. *Packet ID Generation*

In order to get the same packet ID for one packet at two or more measurement points the packet ID generation should be based on the following fields that: a) already exist within the packet; b) are invariant or predictable during the transport; c) are highly variable between the different packets. The goal is to achieve an acceptable low probability of collisions with a packet ID that does not exceed the available capacity for the measurement result data transfer [3]. As for the timestamp, the packet ID only needs to be unique in the given time interval. This limits the possible combinations to the number of packets that can be observed within this interval. For example, on an 155 Mbps link with an average packet size of 512 bytes and a maximum time to traverse the network of 10 seconds, the maximum number of packets would be 378,421 (19,375,000/ 512*10 = 378,421). This amount of combinations can be represented by 19 bits (2^{19} = 524,288). Knowledge about the expected traffic mix therefore can reduce the number of required bits.

F. *Measurement Result Transfer*

In order to calculate the QoS parameters herein at least two timestamps have to be compared. If more than one measurement point is involved the results (timestamps and packet ID) from the different MPs have to be collected at a common location. This point can be located on a separate host or it can be co-located with one of the MPs. The transfer of the measurement results involved the following methods:

a) in-packet: timestamps and packet ID are carried within the packet, which should be modified.
b) in-band: the results are sent directly on the same path as the data.
c) out-of-band: a separate path is needed.

In all the cases additional capacity (either on the existing network or on a separate network) is required. For economic reasons even a separate reporting network would probably have a lower capacity then the “production network”. Therefore to save resources (storage capacity and bandwidth) the measurement results traffic should be kept as low as possible.

IV. PROPOSED ARCHITECTURE FOR PASSIVE REAL-TIME MEASUREMENTS

In order to fulfill the requirements especially that ones related to resource restrictions, real-time representation and data collection, a distributed architecture is proposed, with four components as in *OreNETa*, but with extended functionalities.

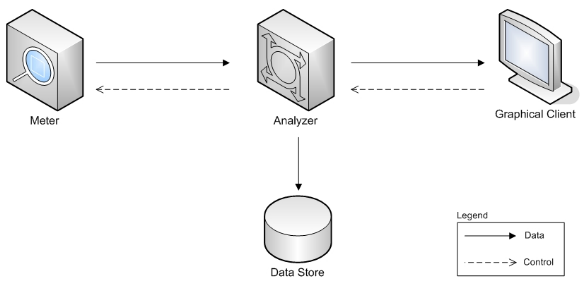


Fig. 5. Proposed Architecture

The *meter* is the component responsible for the first five processes (packet capturing, time-stamping, flow ID generation, classification and packet ID generation). This component can be placed either on the routers of the network being tested, either on dedicated machine for capturing traffic (behind a monitoring port of a switch or behind an optical splitter). For packet capturing and time-stamping, the *meter* uses a shared library *libpcap* (involved in applications like *tcpdump* and *ethereal*).

The *analyzer* is the control application responsible for collecting the measurements results from all the *meters* deployed in the tested network. It is the core component of the architecture and performs the synchronization of the received messages and the computation of QoS parameters. Other functionalities include logging the received measurement results in a data repository, converting the logged data to standardized formats (like MGEN) and supplying real-time statistics about the current traffic. The *analyzer* determines the functionality of the *meters* by sending control messages.

A *data store* is represented by a set of binary files containing all the measurement results received from each *meter*. The reason for not using a dedicated database engine was related to the poor performance of inserting records at very high rates. As an improvement one can develop a conversion tool for moving the data from the binary files to a database.

The *graphical client* is no more than a proof-of-concept application for highlighting the blueprints in extracting real-time statistics from the analyzer.

In Fig. 5 one can observe a flow of control messages along with the data flow between the four components. In order to address the requirements regarding the size of the messages and the limited resources of the control network, a special communication protocol needs to be designed. The bottleneck of the communication is represented by the segments joining the *meters* and the *analyzer*. All other segments presented in Fig. 5 are not relevant, since the *analyzer* and the *data store* are deployed on the same machine (and the operating system will be responsible for exchanging of information), and the messages between the *analyzer* and the *graphical client* are very rare (one every second). Before going any further with the explanation we need to define the term - *flow*, as the collection of packets having the same unique combination of Layer 3 Address, Layer 4 Protocol Type and Layer 4 Ports. The first version of the communication protocol between the *meter* and the *analyzer* was defined in [2]. The major drawback of this version is represented by the big amount of redundant data sent between the two components, especially for IPv6 measurements. For each captured packet on the *meter*, a packet report message was aggregated and sent to the *analyzer*. The fields "L3 Protocol", "IPv4 Src Address", "IPv4 Dst Address", "L4 Protocol", "L4 Src Port" and "L4 Dst Port" are the same for each packet belonging to a flow. Therefore, this data characterizes a flow, not a unique packet and the set of bytes was sent along with each packet report, increasing the redundancy: 14 bytes for IPv4, 38 bytes for IPv6. In order to optimize the traffic between the *meter* and the *analyzer*, a mechanism was developed to send two kinds of messages: *flow descriptors* and *headers*, as in Fig. 6.

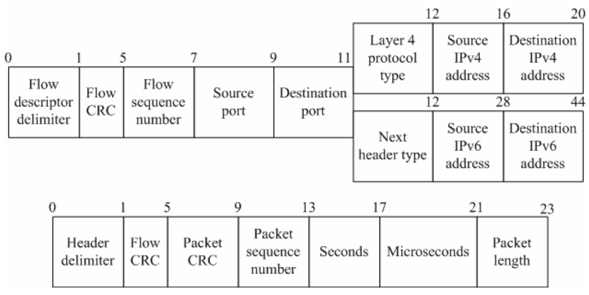


Fig. 6. Flow descriptor and header

When a new flow is detected (a packet is captured and it does not belong to an already detected flow), the *meter* assembles a *flow descriptor* and sends it to the analyzer. Using this message, the *analyzer*, creates a structure which uniquely identifies the new flow. All packets captured later will be "routed" internally by the *analyzer* into one of these structures. This action is equivalent to the "Flow Identification" performed by real routers. After the flow has been recorded by the *analyzer*, it will announce the *meter* to send a shorter message, called *header*, for all the packets belonging to the newly registered flow. The information contained in the *header* is represented by data that is different from one packet to another and uniquely identifies it. One must not mistake this *header* for the IP or TCP headers. The name was chosen like this because the information sent in each *header* is extracted mainly from the real headers of the packets. By means of this mechanism, an important amount of traffic is reduced, mainly because the values identifying a flow (*five-tuple*) are only sent with the *flow descriptor*. This reduces the size of old packet reports from 28 bytes to 23 in the case of IPv4 and from 52 to 23 bytes for IPv6 (more than twice).

Another problem encountered within the first version of *OreNETa* was the fact that packet reports were sent for all captured packets, including those which were not part of any flow. For instance ICMP and ARP packets are not relevant to measuring OWD and IPDV, since a flow is defined using TCP or UDP. In the current version of the program, all the irrelevant packets are discarded at the meter, improving though the bandwidth used for the control network.

V. EXPERIMENTAL RESULTS

In order to test the new proprietary protocol, the testbed included two Debian Linux-based computers (*pampol* and *verdi*) running the *meter*, and one

Slackware Linux-based computer hosting the *analyzer*.

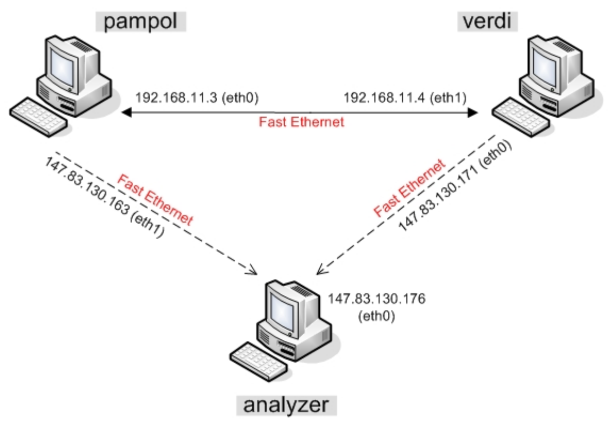


Fig. 7. Communication Protocol Stress Test

The connection between *pampol* and *verdi* represents the tested network, while the link connecting the *meters* with the *analyzer* would be the control network. These two networks are separated using VLANs configured on a Catalyst 2950 switch.

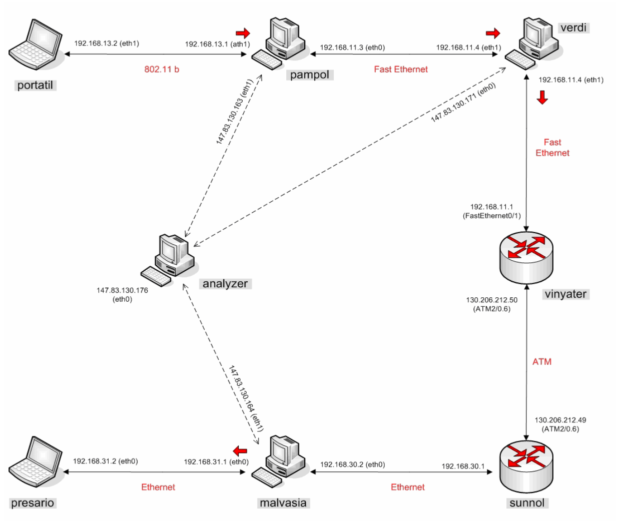


Fig. 8. Testbed

Experiment 1. The test consists in generating a large number of packets per second and monitoring any buffer overflows that might occur at the *meters* or at the *analyzer*. For this scenario, MGEN tool is installed on *pampol*, and different UDP flows are generated through the tested network using the following command:

```
root@pampol:~# mgen -i eth0 -b 192.168.11.4
2000000 -s <packet_size> -r <packet_rate>
```

The packet size did not exceed 1500 bytes, i.e. the Layer 2's MTU (Maximum Transmission Unit), to avoid the packet fragmentation. MGEN generated a set of up to 17,000 pps to *verdi*. The maximum packet rate is limited because of too many resynchronizations needed at the *analyzer* side, the packets being lost due to the limitations of *libpcap*, which is too slow. A new capturing solution may increase the performances of this software tool.

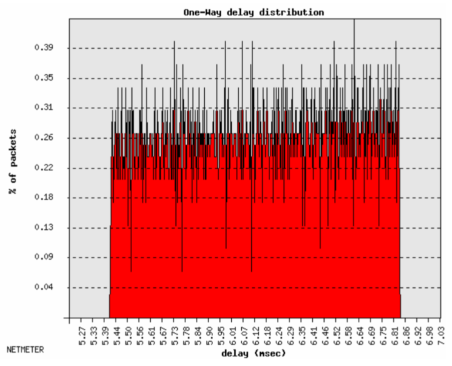


Fig. 9. OWD at 100 packets/s with 370 bytes/packet

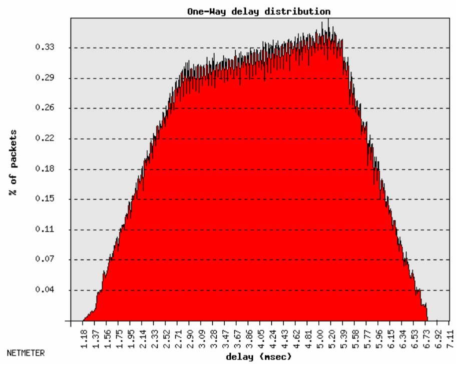


Fig. 10. OWD at 8,000 packets/s with 1,300 bytes/packet

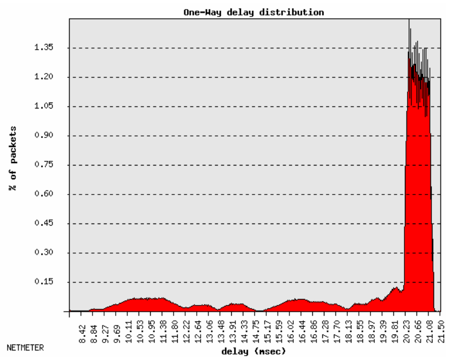


Fig. 11. OWD at 14,000 packets/s with 872 bytes/packet

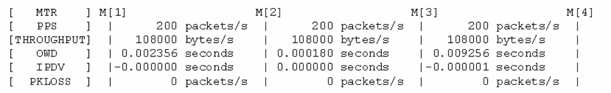
Experiment 2. To prove that the software modules developed are able to measure the QoS parameters over a heterogeneous network, a more complex testbed, presented in Fig. 8, has been used. It included several Linux-based computers, as well as two Cisco 7600 routers. The links tested were Ethernet, Fast Ethernet, 802.11 WLAN and ATM. Table 1 presents the configuration used for all four capture points. The test traffic will be generated from *portatil* to *presario* using MGEN. The capture points and the direction of the traffic are marked with arrows.

Meter *verdi* is used twice in this configuration, but the two instances running on this computer will extract the traffic at different capture points and with different directions (eth1-IN vs. eth1-OUT). The same interface is used on *verdi* both for incoming and outgoing traffic, so the *meters* should be able to distinguish the direction of the traffic.

Table 1 Experiment 2 Network Configuration

Meter	Control network	Tested network	Intf.	Link	I/O
Pampol	147.83.130.163	192.168.13.1	eth1	802.11	I
Verdi	147.83.130.171	192.168.11.4	eth1	FE	I
verdi	147.83.130.171	192.168.11.4	eth1	FE	O
malvasia	147.83.160.164	192.168.31.1	eth0	ETH	O

```
root@portatil:~# mgen -i eth1 -b 192.168.31.2
2000000 -s 512 -r 200
```



```
[   MTR    ] M[1]                    M[2]                    M[3]                    M[4]
[   PPS    ]  |      200 packets/s  |      200 packets/s  |      200 packets/s  |
[THROUGHPUT]  |   108000 bytes/s    |   108000 bytes/s    |   108000 bytes/s    |
[   OWD    ]  | 0.002356 seconds    | 0.000180 seconds    | 0.009256 seconds    |
[   IPDV   ]  |-0.000000 seconds    | 0.000000 seconds    |-0.000001 seconds    |
[  PKLOSS  ]  |        0 packets/s  |        0 packets/s  |        0 packets/s  |
```

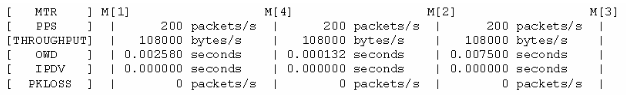
Fig. 12. Experiment 2 results

Note that the number of packets captured on each segment is exactly as expected (200 packets/s). The packet loss is zero since the network is not under stress (108,000 bytes/s). The throughput recorded is bigger than the expected one (200 packets x 512 bytes/packet = 102,400 bytes). The difference in throughput was due to the fact that MGEN appends another 28 bytes to each sent packet. Therefore the size would grow to 512 + 28 = 540 bytes (in this case the computation is correct: 200 packets x 540 bytes/packet = 108,000 bytes). IPDV has negative values, which is correct since the variation of OWD might lead to negative results. Note that the resolution of 6 digits after the decimal point could be extended to improve the granularity of the results. The OWD obtained may vary depending on the Layer 2 link. The very small value of OWD (0.000180 seconds) between *meter 2* and *meter 3* is correct since the two meters run on the same computer. The biggest OWD is obtained for the segment which contains the extra to Cisco hops and the ATM line. Since the routers might be heavily used, by other traffic from the production network, the OWD will increase.

Experiment 3. Suppose the *meters* are perfectly synchronized in time. The *analyzer* is able to determine the route for the requested flows irrespective of the order of the meters supplied at command line. Let us change now the order of the meters, by placing M[4] not at the edge of the testbed, but on one of *verdi's* interfaces. The new configuration is presented in Table 2. We generate the same traffic as in the previous test using MGEN.

Table 2 Experiment 3 Network Configuration

Meter	Control network	Tested network	Intf.	Link	I/O
pampol	147.83.130.163	192.168.13.1	eth1	802.11	I
verdi	147.83.130.171	192.168.11.4	eth1	FE	O
malvasia	147.83.130.164	192.168.31.1	eth0	ETH	O
verdi	147.83.160.171	192.168.11.4	eth1	FE	I



```
[   MTR    ] M[1]                    M[4]                    M[2]                    M[3]
[   PPS    ]  |      200 packets/s  |      200 packets/s  |      200 packets/s  |
[THROUGHPUT]  |   108000 bytes/s    |   108000 bytes/s    |   108000 bytes/s    |
[   OWD    ]  | 0.002580 seconds    | 0.000132 seconds    | 0.007500 seconds    |
[   IPDV   ]  | 0.000000 seconds    | 0.000000 seconds    | 0.000000 seconds    |
[  PKLOSS  ]  |        0 packets/s  |        0 packets/s  |        0 packets/s  |
```

Fig. 13. Experiment 3 results

As expected, the order of the meters changed according to the path of the captured flow. M[4] follows M[1] since they are close to each other (*pampol-IN* and *verdi-IN*). The other 2 meters, M[2] placed on *verdi-OUT* and M[3] placed on *malvasia-OUT* are ordered accordingly to the route presented in Fig. 10. If the time synchronization between the *meters* is not accurate, this functionality will be useless, as well as the whole project. If the time difference between two *meters* is almost equal to the OWD value, we might expect to obtain negative values for the OWDs.

VI. CONCLUSIONS

Compared to the previous version of *OreNETa* the present work came up with a set of improvements, optimizations and additional functionalities. In order to optimize the traffic between the *meter* and the *analyzer*, a mechanism to send two kinds of messages (*flow descriptors* and *headers*) was developed. An important amount of traffic is reduced, mainly because the values identifying a flow (*five-tuple*) are only sent with the *flow descriptor*. This reduces the size of old packet reports from 28 to 23 bytes in the case of IPv4, and from 52 to 23 bytes for IPv6. All the irrelevant packets (e.g. ICMP and ARP) were discarded at the meter, improving though the bandwidth used for the control network. Another improvement is the possibility to connect more than one *analyzer* to a *meter* and vice-versa. The flow processing was implemented based on the fact that the packet IDs will always be captured in the same order by every *meter*, so the *headers* will be arranged chronologically when they arrive at the *analyzer*. A mechanism was implemented to order the *meters* on the tested network by comparing the timestamps of the first packets received. This simple mechanism is useful for tracing a route a flow is using, in case all the *meters* are perfectly synchronized in time. The tool uses pure binary files to store only the needed data, and the processing is performed later, when the capture is finished.

REFERENCES

[1] ***, EuQoS End-to-end Quality of Service Support over Heterogeneous Networks, http://www.euqos.org/

[2] A. Navarro, "OreNETa". Master Thesis, *Universitat Politecnica de Catalunya*, Barcelona, Spain, 2004

[3] T. Zseby, S. Zander, G. Carle, „Evaluation of Building Blocks for Passive One-Way Delay Measurements", *GMD FOKUS*, 2001

[4] Z. Wang, *Internet QoS: Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, 2001

[5] ***, MGEN-UDP Traffic Generator Tool, http://manimac.itd.nrl.navy.mil/MGEN/

[6] ***, IST-MOME Cluster of European Projects Aimed at Monitoring and Measurements, http://www.ist-mome.org/cluster/associated.html