Active traffic monitoring for heterogeneous environments

Hélder Veiga, Teresa Pinho, José Luis Oliveira, Rui Valadas, Paulo Salvador, and António Nogueira

University of Aveiro / Institute of Telecommunications Aveiro
Campus de Santiago, 3810-193 Aveiro, Portugal
{jlo,rv}@det.ua.pt,{hveiga, salvador, nogueira}@av.it.pt

Abstract. Traffic management of IP networks comprises increasing challenges due to the occurrence of sudden and deep traffic variations that can be mainly attributed to the combined effects of several factors, like the great diversity of supported applications and services, different user's behaviors and different mechanisms of traffic generation and control. In this context, active traffic monitoring is particularly important as it enables characterizing essential aspects in network operation, like for example, quality of service as measured in terms of packet delays and losses. The main goal of this work is to carry out active measurements in a real operational network consisting in a heterogeneous environment that includes both wired and wireless LANs. In order to perform this task, a measurement methodology, and its corresponding measurement platform, will be proposed. The measurement methodology is based on the One-Way Active Measurement Protocol (OWAMP), a recent proposal from the Internet2 and IETF IPPM groups for active measurements of delays and losses in a single direction. The measurement platform was implemented, tested and conveniently validated. This paper begins by a brief presentation of the measurements that we intend to perform, then it describes the OWAMP protocol and the developed measurement system, including its implementation, test and validation through its application to different network scenarios.

keywords: Network management, traffic monitoring, active measurement, OWAMP.

1 Introduction

The relevance of traffic monitoring in the global management of IP networks has been growing due to the recent acknowledgment that sudden and deep traffic variations demand for frequent traffic measurements. This peculiar behavior of network traffic can be mainly attributed to the combination of different factors, like for example the great diversity of supported applications and services, different user's behaviors and the coexistence of different mechanisms for traffic generation and control.

Traffic monitoring systems can be classified in active and passive ones [1], [2], [3]. Passive systems simply perform the analysis of the traffic that flows through the network, without changing it. Usually, they are used to identify the type of protocols involved and to measure one or more characteristics of the traffic that flows through

the measurement point, like the average rate, the mean packet size or the duration of the TCP connections. Nowadays, there are several passive monitoring systems, like for example NeTraMet [4] and NetFlow [5]. Active systems insert traffic directly into the network. Usually, they are intended to provide network performance statistics between two distinct measurement points, like for example mean packet delay and packet loss ratio. Those statistics can be one-way statistics, when they refer to a single direction of traffic flow, and round-trip statistics, when they refer to traffic that flows in both directions.

Passive measurements involve measurement intervals that can stretch from several milliseconds to weeks or even months, thus forcing the storage and processing of huge data quantities. In active measurements, the only packets that are sent to the network are the ones that will be processed, and measurement intervals are in the order of seconds or minutes. However, it is usually necessary to guarantee the synchronization of the involved measurement points, using for example GPS (Global Positioning System) or NTP (Network Time Protocol).

The IETF IPPM (IP Performance Metrics) group established in the last few years a set of recommendations in order to assure that measurement results obtained from different implementations are comparable, namely regarding measurements of one-way packet delays and losses [6], [7]. However, these recommendations do not address the interoperability of the measurement elements, that is, the possibility of having traffic senders and receivers that belong to different administrative domains and are developed by different entities. OWAMP is a proposal for a one-way active measurement protocol that intends to solve this problem [8].

In this work, we intend to perform a set of active measurements in a real operational network consisting in a heterogeneous environment that includes both wired and wireless LANs. Thus, instead of using available tools (like PING, for example), some of them with a limited scope of applications, we have decided to implement a complete measurement platform (freely available at http://www.av.it.pt/JOWAMP/). In order to guarantee its compliance with other available platforms, its measurement methodology is based on the OWAMP protocol.

The paper is structured in the following way: section 2 describes the architecture and the operational details of the OWAMP protocol, that forms the basis of the implemented solution; section 3 presents the details of the implemented solution; section 4 presents the active measurements experiments, and their corresponding scenarios, that we want to carry out in this work; section 5 presents and discusses the results obtained from its application to the defined measurement scenarios and, finally, section 6 presents the main conclusions.

2 One-Way Active Measurement Protocol (OWAMP)

The One-Way Active Measurement Protocol (OWAMP) is a recent proposal from the Internet2 group, developed under the scope of the End-to-End Performance Initiative project [9], [10], for performing active measurements in a single direction. This proposal is also promoted by the IETF IPPM work group [8].

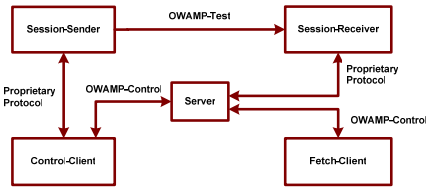

Fig. 1. OWAMP architecture.

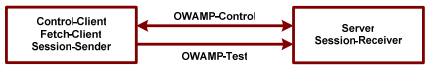

Fig. 2. OWAMP simplified architecture.

2.1 Architecture

The OWAMP architecture is based on two inter-dependent protocols, the OWAMP-Control and the OWAMP-Test, that can guarantee a complete isolation between client entities and server entities. The OWAMP-Control is used to begin and end test sessions as well as receive the results of those tests, whereas the OWAMP-Test protocol is used to allow the exchange of test packets between any two points that belong to the monitored network.

The proposed architecture includes the following elements (figure 1):

- Session-Sender: the sender of the test packets;
- Session-Receiver: the receiver of the test packets;
- Server: the entity that is responsible for the global management of the system; it can configure the two terminal elements of the testing network and receive the results of a test session;
- Control-Client: a terminal system that programs demands for test sessions, triggers the beginning of a session set and can also finish one or all ongoing sessions;
- Fetch-Client: a terminal system that triggers the demands for results of test sessions that have already ended or are still running.

A network element can carry out several logical functions at the same time. For example, we can have only two network elements (figure 2): one is carrying out the functions corresponding to a Control-Client, a Fetch-Client and a Session-Sender and the other one is carrying out the functions corresponding to a Server and a Session-Receiver.

The OWAMP-Control protocol runs over TCP and is used to begin and control measurement sessions and to receive their results. At the beginning of each session, there is a negotiation about the sender and receiver addresses, the port numbers that both terminals will use to send and receive test packets, the instant of the session beginning, the session duration, the packets size and the mean interval between two consecutive sent packets (it can follow an exponential distribution, for example).

The OWAMP-Test runs over UDP and is used to exchange test packets between sender and receiver. These packets include a Timestamp field that contains the time instant of packet emission. Besides, packets also indicate if the sender is synchronized with some exterior system (using GPS or NTP) and each packet also includes a Sequence Number.

OWAMP supports test packets with service differentiation: DSCP (Differentiated Services Codepoint), PHB ID (Per Hop Behavior Identification Code) or Best-effort. Additionally, OWAMP supports some extra facilities like cypher and authentication for the test and control traffic, intermediary elements called Servers that operate as proxies between measurement points and the exchange of seeds for the generation of random variables that are used in the definition of transmitted test flows. The OWAMP specification also allows the use of proprietary protocols (that can be monolithic or distributed programming interfaces) in all connections that do not compromise interoperability.

2.2 Protocol

In the architecture of the developed system (figure 1) we choose to use the OWAMP-Control as the communication protocol between client and sender and between server and receiver (protocols used in these connections are not specified in the OWAMP model). This approach guarantees a higher independence of the different modules, that can thus be shared between several systems.

A measurement session is started by the client, who establishes a connection with the server. This connection comprises the establishment of a TCP connection and the exchange of three messages, designated by *Connection Setup Server Greating*, *Connection Setup Client Response* and *Connection Setup Server Response*. All subsequent connections between network elements follow this format. From this moment on, the client is able to make requests of test sessions to the server. However, before each request the client establishes a connection with the involved sender in order to determine if it exists and is in an active state (figure 3). If the sender is in an active state, the established TCP connection is maintained and the client sends a *Request-Session* command to the server. When the server receives this command it finds the involved receiver and establishes a new connection with it. If the involved receiver is active, the server re-sends it the *Request-Session* message. The receiver answers with an *Accept-Session* message, indicating whether it accepted or rejected the request. The server waits for this response and re-sends the *Accept-Session* message to the client. If the *Accept-Session* is positive, the client sends a *Request-Session* command to the involved sender, which will answer again with an *Accept-Session* message. If the *Request-Session* request is rejected by the receiver, the client must close the TCP connection that was previously established with the sender.

If the *Request-Session* request(s) are accepted, the client starts the test sessions by sending each sender a *Start-Sessions* message, and the senders will respond with a *Control-Ack* packet (figure 4). If the sender accepts the session request, it will wait until the start time (the instant of session beginning) in order to start sending test packets to the receiver. However, if any of the senders' responses is negative, the client must close all connections by sending a *Stop-Sessions* command to the server. If all senders accept the request, the client sends a *Start-Sessions* message to the server, which in turns re-sends this message for each one of the involved receivers. These receivers will in turn respond to the server with a *Control-Ack* packet, informing it if they accept or reject the request. If all responses are positive, the server accepts the *Start-Sessions* request by sending a positive *Control-Ack* packet to the client. Otherwise, the server sends a

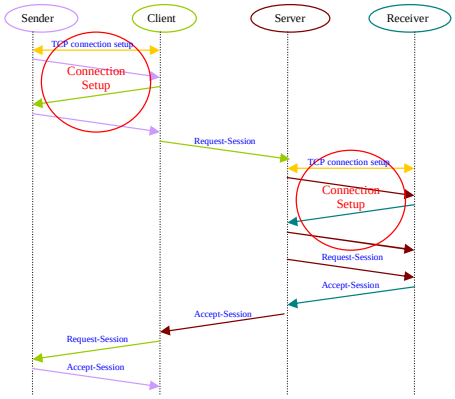

Fig. 3. Temporal diagram of all messages that are exchanged during the establishment of a *Request-Session* request.

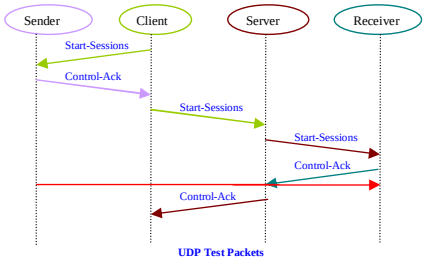

Fig. 4. Temporal diagram of all messages that are exchanged during the establishment of a *Start-Sessions* request.

negative *Control-Ack* packet to the client. Whenever a receiver accepts a *Start-Sessions* request, it will be automatically waiting for the start time in order to start receiving test packets.

In the monitor, whenever there is a request for results a connection is firstly established between the monitor and the server (figure 5). Then, a *Fetch-Session* packet is sent from the monitor to the server. Based on the session identifier (SID), that is included in the packet, the server identify the receiver address and re-routes the reading request to it. The receiver verifies if it has any available results that can satisfy the monitor request and sends a positive or negative *Control-Ack* packet to the server according to the available results. The server re-sends this packet to the monitor, which in case of a positive response, will remain waiting for the results. Immediately after a positive *Control-Ack*, the receiver sends to the server a *Session-Data* packet followed by a 16 bytes *Integrity Zero Padding* packet. The server re-routes these packets to the monitor.

A test session can be terminated even before its beginning or before sending all test packets. This can be achieved using the *Stop-Sessions* command. This command can be sent by both the client and the server. This is the only command that can be used after the beginning and before the end of a test session, that is, during the session. In order to perform complete measurement sessions, both the client and the server must wait until the end of the session (which is marked by a Timeout since the last test packet has been sent) to exchange *Stop-Sessions* packets.

3 J-OWAMP: a system based on OWAMP

In order to create an innovator platform for active measurements, that can also represent a basis for the development and test of new algorithms and models, we built a system designated by J-OWAMP (a Java implementation of the OWAMP protocol). The

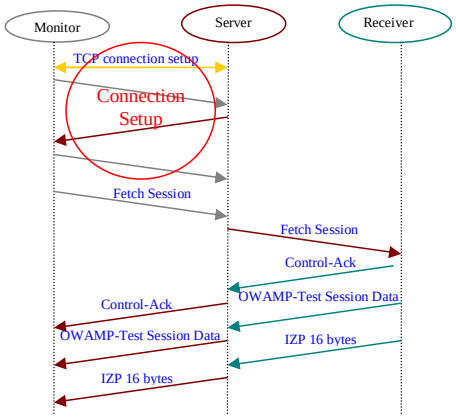

Fig. 5. Temporal diagram of all messages that are exchanged during the establishment of a *Fetch-Session* request.

current version implements the May 2004 OWAMP proposal. The developed system corresponds to the scenario depicted in figure 1, a more general architecture. This architecture allows the definition of only one client and one server in the network (possibly installed in machines with the highest processing capacity) and allows the installation of senders and receivers in any machine of the network, which leads to a lower processing impact. In this way, the network manager can perform tests all over the network from a single machine, which is not possible in the simplified scenario of figure 2.

3.1 Structure and implementation

The J-OWAMP system was developed in Java language because this language presents a set of favorable characteristics, like semantic simplicity, portability and a set of classes that greatly simplify the construction of distributed applications.

The structure of the system is based on two levels: Messages and Entities. At the Messages level, we developed a set of classes corresponding to each one of the data packets that are exchanged in the OWAMP protocol. The main class Packet is the basis for all messages (derived classes), so it contains all methods (basic functions) for manipulating and formating the different data types involved in the protocol. It also contains the methods that are needed to receive and send packets using a socket.

Each type of packet is defined in a class that derives from the Packet class. Each one of these subclasses includes also two new subclasses, one that is used to send - *class ...ToBeSend* - and another that is used to receive the packet - *class ...Receiver*. Sending a packet always implies transferring all its bytes to a buffer and send it through a socket, so the subclass that is responsible for sending a packet always includes two methods: a *datagramToBuffer* method and a *sendTCP..* method, that implement these two functionalities, respectively. In the same way, the reception of a packet implies its reception through a socket and transferring information from the reception buffer to the

respective packet format. Thus, the reception subclass includes two methods, *receive...* and *create_Datagram_...*, that implement these two functionalities.

The Packet class also contains the Timestamp subclass that joins the methods and objects that are needed to obtain and process the temporal information of the system.

At the Entities level, a set of seven classes was developed in order to implement the five elements (*Client*, *Server*, *Session-Sender*, *Session-Receiver* and *Fetch-Client*) of the OWAMP architecture:

- *OWAMP_Base* extends *Thread*
- *OWAMP_ControlClient* extends *OWAMP_Base*
- *OWAMP_Server* extends *OWAMP_Base*
- *OWAMP_SessionTerminal* extends *OWAMP_Base*
- *OWAMP_SessionReceiver* extends *OWAMP_SessionTerminal*
- *OWAMP_SessionSender* extends *OWAMP_SessionTerminal*
- *OWAMP_FetchClient* extends *OWAMP_Base*

We now give a more detailed description of each developed class.

The *OWAMP_Base* class was defined in order to group in a same class all methods that are common to the several classes that implement the different elements of the OWAMP architecture (for example, the method that is used to establish the communication between two OWAMP elements - *Connection Setup*). This class extends the *Thread* class, because some of its derived classes (like, for example, classes related to the server, sender and receiver) use this multiprocessing concept to process the various received requests. All remaining system classes that are part of the entities constitute subclasses of this class.

The *OWAMP_ControlClient* class is a subclass of the *OWAMP_Base* class and implements the *Client* element of the protocol. *Client* is the entity that makes the requests for test sessions, and can also be the entity responsible for making the requests for results. This class includes all the objects and methods that are necessary to establish the communication between client and sender or server. It includes, among others, a method to send *Request-Session* packets to the server and the sender, a method that implements *Start-Sessions*, a method to send *Stop-Sessions* commands and another one to test this sending, and a method to evaluate if all sessions are finished. This class also comprises a subclass, named *RandomExponentialDistribution*, to generate random numbers having exponential distribution. System configurations can be done from the command line or using a configuration file.

The *OWAMP_Server* class is a subclass of the *OWAMP_Base* class and implements the *Server* entity of the protocol. The server is a process that is continuously running and acts as an intermediary between clients and receivers and between monitors (if they exist) and receivers. This class comprises all the objects and methods that enable the server to respond to all *Request-Sessions* and *Start-Sessions* requests from the clients, guarantying that the receiver is ready to accept the session. Whenever a client establishes a new connection with the server (Connection Setup), a new process is triggered in order to accept all *Request-Session* requests that exist in this *OWAMP-Control* connection. In this way, the server becomes available to accept requests from a new *OWAMP-Control* connection. This class also contains a method to test the sending

of *Stop-Sessions* commands and a method that is able to answer to requests for results (fetch) that are made by the monitor. Whenever a new reading request is made, a new process is started in order to process this request.

The *OWAMP_SessionTerminal* class is a subclass of the *OWAMP_Base* class and groups in a same class all methods that are common to the *OWAMP_SessionSender* and *OWAMP_SessionReceiver* classes. This class contains all methods and objects that are necessary for the sender and receiver to answer to *Request-Session*, *Start-Sessions* and *Stop-Sessions* requests that come from the client or server, respectively. We chose this configuration because both the sender and receiver answer in the same way to these messages.

The *OWAMP_SessionReceiver* class is a subclass of the *OWAMP_SessionTerminal* class that implements the *Receiver* entity of the protocol. The receiver is responsible for accepting the test packets and mark them with their arrival instant. This class contains all methods and objects that are necessary for the receiver to answer to *Fetch-Sessions* requests that come from the server, and to receive and process all test packets. Whenever a new test session coming from the same receiver is started, a new process is created in order to receive and process packets.

The *OWAMP_SessionSender* class is a subclass of the *OWAMP_SessionTerminal* class and implements the *Sender* entity of the protocol. The sender is a process that is continuously running in a specific pot number and is responsible for marking the test packets with a sending timestamp and sending these packets to the receiver, using UDP. This class contains all methods and objects that are necessary for the sender to create and send test packets to the receiver. Whenever a new test session coming from the same sender is started, a new process is created in order to process and send packets.

The *OWAMP_FetchClient* is a subclass of the *OWAMP_Base* class and implements the *FetchClient* entity of the protocol. The main feature of this entity is to allow the reading of results belonging to a session that is still running. In this way, it is not necessary to wait for the end of the session in order to get some results (related to the first 20 packets, for example). This class comprises the methods and objects that are necessary to read and print the results, as well as process global statistics corresponding to one or more sessions.

3.2 Compliance tests

In order to guarantee the compliance of the developed system with the OWAMP proposal we have performed a set of tests involving an implementation for a UNIX platform developed by the Internet2 group and publicly available in [9]. The tests were carried out in the private IT-Aveiro network using, in a first experiment, the J-OWAMP modules as the client, monitor and sender modules and using the Internet2 modules as server and receiver modules and, in a second experiment, the J-OWAMP and Internet2 modules in the reverse order (figure 6).

The communication between the J-OWAMP modules (developed in Java language) and the Internet2 modules (developed in C language) was correctly established, in both directions. Using the Ethereal traffic analyzer, we have verified that the control messages and the test packets are correctly exchanged, as specified in the protocol.

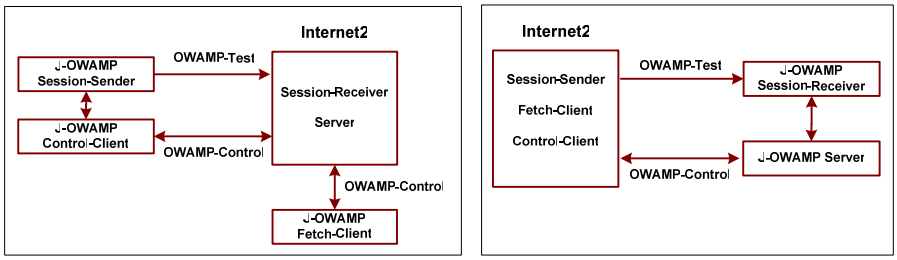

Fig. 6. Configuration of the compliance tests.

4 Measurement scenarios

Before carrying out active traffic measurements in the real network involving a heterogeneous environment, we have first established a laboratorial measurement setup that was used to test the developed measurement solution in a more controllable environment.

4.1 First scenario: laboratorial environment

The measurement setup for this scenario is illustrated in figure 7. Routers 1 and 2 are connected through a serial link configured with a transmission capacity of 64 Kb/s and three networks are configured with the following structure: network 192.0.0.0, that contains PC1 running the OWAMP sender; network 192.0.2.0, that contains PC2 running the traffic generator MGEN and network 192.0.1.0 that contains PC3 where we have previously installed the OWAMP client, server and receiver elements as well as a receiver (Drec) of the traffic generated by the MGEN application running on PC2. The service discipline for all queues belonging to the serial interfaces of routers 1 and 2 is FIFO. PCs 1 and 3 are synchronized through NTP.

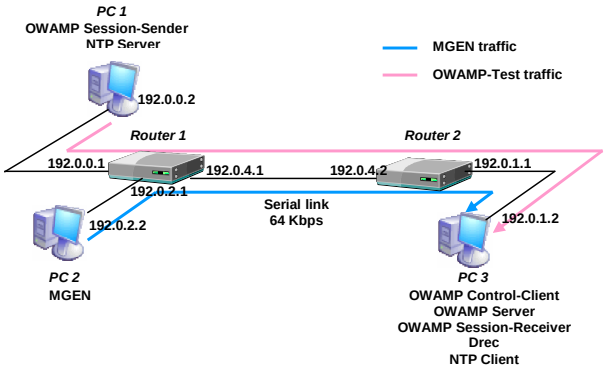

Fig. 7. Network corresponding to the first measurement scenario.

Using this scenario we intend to measure and study the packet delays that occur in the queuing system of Router 1 and are due to the transmission capacity of the serial link between routers 1 and 2, for different values of the traffic load in that serial link. So, we have configured the MGEN application to generate traffic according to a Poisson distribution and send it to PC3 using the serial link. Using the sender installed in PC1 and the receiver installed in PC3 we are able to measure the delay values that occur in the queue of Router 1 serial interface, for different values of the traffic load. Arrows represented in figure 7 show the directions that are followed by the traffic generated with MGEN and the traffic consisting of test packets generated by the developed measurement system.

4.2 Second scenario: University of Aveiro wireless network

In this scenario we want to make some measurements in the wireless network of University of Aveiro (UA), trying mainly to evaluate the performance of accessing this network from the students' residences. In order to do that, we made some measurement experiments between a PC located in the laboratory of Institute of Telecommunications (IT), named Lab PC, and another one located in a students' residence of the University campus, named Residence PC (figure 9).

Executing the *'tracert 192.168.140.47'* command in the Lab PC we can get the path and the round-trip delays between the Lab and the Residence PCs (figure 8).

```
D:\>tracert 192.168.140.47

Tracing route to FRANCIS [192.168.140.47]
over a maximum of 30 hops:

  1    1 ms    1 ms    1 ms  gtav.it.pt [193.136.92.1]
  2   37 ms   97 ms   41 ms  gt-cicua.core.ua.pt [193.136.86.193]
  3   86 ms   33 ms   51 ms  VPN-WIRELESS [192.168.140.253]
  4   52 ms   94 ms   36 ms  FRANCIS [192.168.140.47]
Trace complete.
```

Fig. 8. Result of the 'tracert' command between Lab PC and Residence PC.

We studied the traffic that flows between the Residence and the Lab PCs in both directions. We installed the client, server and receiver in the PC that receives the test packets and the sender in the PC that is used to send the packets. Both PCs are synchronized via NTP. Note that Internet access from the residences is performed through the UA network. So, traffic in the downstream direction includes the downloads from the Internet to the residences.

All tests were performed in a 24 hours period. In each hour, sets of 10 tests (including both packet delay and loss) were performed, making a total of 240 tests. In each group, the tests beginning instants were separated by 2 minutes. All tests lasted for 1 minute and consisted in sending 60 packets of 14 bytes each, at an average rate of 1 packet/second. In order to conveniently characterize the packet average delay and packet loss ratio, we have calculated 90% confidence intervals based on the 10 average values obtained in each test belonging to a group of 10 tests.

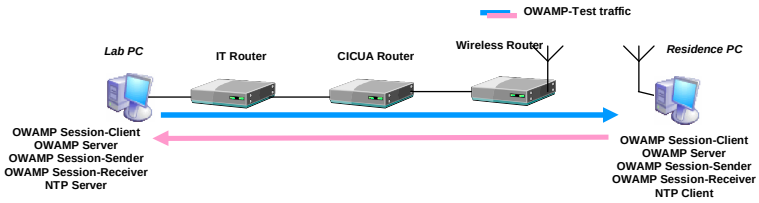

Fig. 9. Network corresponding to the second measurement scenario.

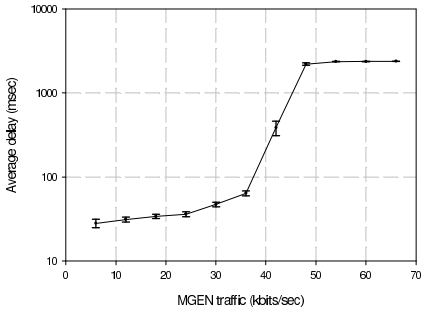

Fig. 10. Results of the first scenario: average packet delay versus MGEN generated traffic.

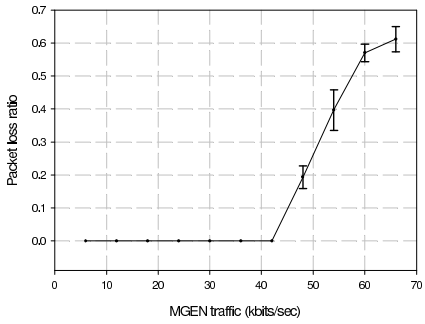

Fig. 11. Results of the first scenario: packet loss ratio versus MGEN generated traffic.

5 Results

5.1 Results of the first scenario

Figures 10 and 11 present the results corresponding to the packet delay and packet loss tests conducted in the first scenario, for different values of the MGEN generated traffic. From the analysis of the obtained results we an verify that, as expected, there is an increase in packet delays and losses with increasing network load. For network load values that are far from the maximum value supported by the serial link (64 Kb/s), there are no packet losses. However, packet loss values increase very fast as network load approaches the limit load supported by the serial link that connects both routers.

5.2 Results of the second scenario

For this scenario, we have studied the traffic that flows between the Residence PC and the Lab PC (upstream) and in the reverse direction (downstream). The results of the average packet delay and packet loss ratio for the upstream direction are presented in figures 12 and 13, respectively, and the analogous results corresponding to the downstream direction are presented in figures 14 and 15, respectively.

From the analysis of the packet delay and packet loss values we can verify that in the upstream direction delays vary between approximately 30 and 120 milliseconds

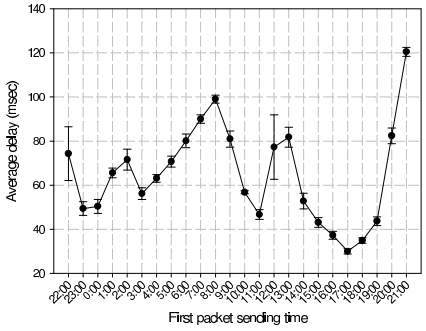

Fig. 12. Results of the second scenario, upstream direction: average packet delay versus first packet sending time.

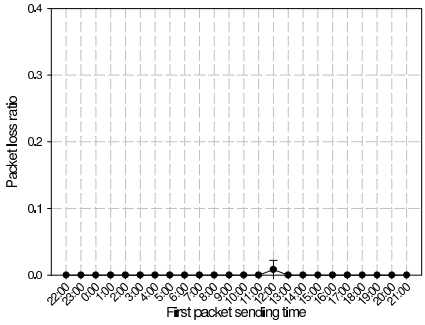

Fig. 13. Results of the second scenario, upstream direction: packet loss ratio versus first packet sending time.

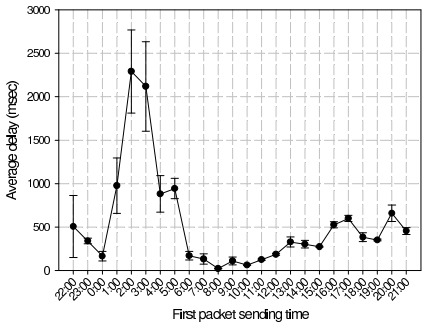

Fig. 14. Results of the second scenario, downstream direction: average packet delay versus first packet sending time.

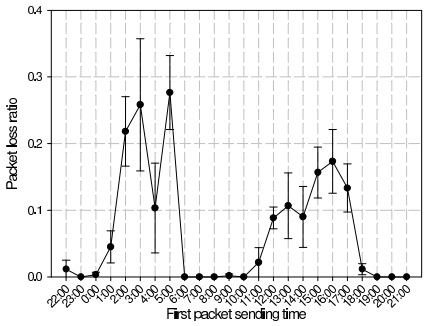

Fig. 15. Results of the second scenario, downstream direction: packet loss ratio versus first packet sending time.

and are much lower that the corresponding values of the downstream direction, that vary between 20 and 2300 milliseconds. Packet losses are null in the upstream direction but have non zero values in the downstream direction. As expected, there is a direct relationship between packet delays and losses: higher packet delay values also correspond to higher packet loss values. In the performed tests, downstream traffic was much higher than upstream traffic, which is a typical result for these kind of scenarios. In the downstream direction, the highest delay and loss values were observed in the night and afternoon (between 2PM and 6PM) periods. These values can be attributed to the use of file sharing applications. In the night period, the utilization level of these applications is even higher, mainly from the students' residences. In the afternoon period, the utilization of these applications is mainly from the library building, which is also covered by a wireless network.

6 Conclusions

Traffic monitoring through active measurements is having an increasing relevance in the IP networks management context, since it enables to directly monitor quality of service parameters, like for example average packet delay and packet loss ratio. The IETF IPPM group has recently proposed a protocol for conducting active traffic measurements in a single direction, the OWAMP (One-Way Active Measurement Protocol).

This paper presented a solution (based on the OWAMP protocol) for performing active measurements in a heterogeneous network, including its implementation, validation and some examples that allow a further exploration of the OWAMP protocol. The proposed system was developed in Java language, mainly due to its portability. Several compliance tests with the only known implementation (from the Internet2 group) were successfully conducted. The system was evaluated through a set of performed tests, conducted both in a laboratorial environment and in a real operational network. The obtained results show that the implemented system is a very useful active measurement tool that can be used for characterizing quality of service in IP networks.

Acknowledgments: This research was supported by Fundação para a Ciência e a Tecnologia, project POSI/42069/CPS/2001, and European Commission, Network of Excellence EuroNGI (Design and Engineering of the Next Generation Internet).

References

1. A.Pasztor, D.Veitch: High precision active probing for internet measurement. In: Proceedings of INET'2001. (2001)
2. Corral, J., Texier, G., Toutain, L.: End-to-end active measurement architecture in ip networks (saturne). In: Proceedings of Passive and Active Measurement Workshop PAM'03. (2003)
3. Grossglauser, M., Krishnamurthy, B.: Looking for science in the art of network measurement. In: Proceedings of IWDC Workshop. (2001)
4. NeTraMet home page: (http://www.auckland.ac.nz/net/netramet/)
5. White Paper - NetFlow Services and Applications: (http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm)
6. Almes, G., Kalidindi, S., Zekauskas, M.: RFC 2679: A one-way delay metric for ippm (1999)
7. Almes, G., Kalidindi, S., Zekauskas, M.: RFC 2680: A one-way packet loss metric for ippm (1999)
8. Shalunov, S., Teitelbaum, B., Karp, A., andMatthew J. Zekauskas, J.W.B.: A one-way active measurement protocol (owamp), internet draft (2004)
9. Internet2 End-to-End Performance Initiative: (http://e2epi.internet2.edu)
10. Boyd, E.L., Boote, J.W., Shalunov, S., Zekauskas, M.J.: The internet2 e2e pipes project: An interoperable federation of measurement domains for performance debugging (2004)