Providing cost-effective QoS monitoring in multiservice networks

Paulo Carvalho, Solange Rito Lima, André Ferreira, Emanuel Freitas, Filipe Leitão
University of Minho, Department of Informatics
4710–057 Braga, Portugal

***Abstract*—In multiservice networks, QoS monitoring needs to be carried out in a per-class basis so that each service class measuring requirements and behavior are met and sensed properly. Facing the shortage of off-the-shelf class-based monitoring solutions, this work is focused on the development of a flexibly QoS monitoring tool oriented to multiservice networks. In this context, after discussing main QoS monitoring issues, we propose a flexible QoS monitoring Java application, totally user parameterized and supporting service differentiation. Benefiting from an edge-to-edge design perspective, this service-oriented tool is able to make a periodic evaluation of relevant QoS metrics for each service class, on an intra-domain or end-to-end path basis. Monitoring results, stored in a MySQL database, are useful to drive both online and offline traffic engineering and service management tasks.**

I. Introduction

Emergent applications and services are putting high demands to network infrastructures, growing side-by-side with the network available bandwidth. With the deployment of multiservice networks and subjacent services, monitoring the Quality of Service (QoS) delivered to user applications assumes a crucial role as it allows a more rational management of network resources, and an efficient service control and auditing. Therefore, a systematic monitoring of service classes is needed to detect eventual changes of network behaviour, which may endanger the fulfilment of negotiated Service Level Agreements (SLAs). The monitoring process should provide measures reflecting the real status of services' quality, without introducing significant overhead or interfering with operational network traffic. Active monitoring carried out on an edge-to-edge basis, i.e. between the network boundaries in which a service level needs to be enforced, is particularly suitable for QoS control and SLS auditing.

Providing measurement analysis has been a hot topic for network researchers for a long time and, in the last years, concrete development has been achieved toward active measurement solutions. Within IETF, IP Performance Metrics working group has standardized methods for Delay [1] and Loss [2] measurements, which converge in the relatively recent One-Way Active Measurement Protocol (OWAMP) [3]. RIPE NCC project [4] is a good example of how to provide active measurements for ISPs, although NCC approach does not support differentiation based on service classes. Other network monitoring solutions mostly evaluate the network status and analyze its behaviour from a service monolithic perspective. These solutions are limitative within multiservice networks context, where the main concerns regarding monitoring are related to a scalable evaluation of differentiated QoS delivery.

In this work, we propose – QMon – a scalable and class-based QoS monitoring tool, which can be deployed within a single domain or over multiple network domains. The tool can be fully parameterized, which means that a user can easily configure the application to monitor services' behaviour, and use it for assessing each service class QoS status and performance. Being a Java multiplatform implementation, it is totally independent from specific monitoring and measurement hardware and network infrastructure. This also fosters portability, while being a cost-effective solution.

To sustain QMon design principles and architecture, this paper starts with a study of relevant characteristics of monitoring systems, focusing the debate on the problem of QoS monitoring on multiservice networks. From the vast range of projects and tools covering multiple network monitoring aspects [5], few are able to provide an encompassing view of QoS on a per class basis (see debate in Section II). This reasoning and analysis grounds the motivation for developing a new service-oriented QoS monitoring tool for performance evaluation of next generation networks.

This paper is organized as follows: related work is discussed in Section II; QMon design goals and architecture are presented in Section III; QMon components are specified in Section IV; the implementation details and current stage of QMon prototype testing are provided in Section V; and the conclusions are presented in Section VI.

II. QoS Monitoring issues and related work

A monitoring system can follow either a centralized or distributed architecture. Centralized approaches facilitate an integrated and consistent view of the network performance, but scalability problems may occur in infrastructures involving large number of monitoring nodes and significant volume of monitoring data. In distributed monitoring systems, data is collected and processed at each measurement point (MP) or, more commonly, at the receiver side of each pair of MPs, following a sender-receiver or peer-to-peer model. These latter approaches are particularly suitable for on-line QoS measurement purposes on an edge-to-edge or path basis. Nevertheless, a central data repository is recommended to facilitate a post-processing QoS analysis, for instance, to support SLS auditing and medium/long-term traffic engineering decisions.

The classic process to obtain QoS measures resorts to passive and/or active measurement methodologies. Active measurements resort to intrusive traffic, or probes, specifically injected in the network for measurement purposes. This methodology provides a straightforward way for assessing QoS objectives, as specific packets are injected in the network containing data to help metrics' estimation. However, probing needs to be tightly controlled so that it does not disturb or interfere with the normal network operation. This concern is further stressed when it is carried out per service class. Although active monitoring has been matter of considerable research, extending its study to a multiclass network paradigm is crucial and further study is needed [6]. As a reference contribution, also to QMon design, the OWAMP standard [3] allows measuring one-way latency and loss related metrics between hosts, based on measurements of synthetic traffic, including support for traffic class differentiation.

Passive measurements use existing network traffic for metric computation. Although passive techniques are usually used to monitor the performance of single nodes, e.g. troubleshooting or accounting, they can also be applied to edge-to-edge measurements, for instance, combining hop-by-hop metrics along the network path. This allows reducing the amount of synthetic traffic, at expense of increasing processing and synchronization needs. An alternative edge-to-edge approach still within the scope of passive measurements relies on the analysis of information of real application flows (e.g., using TCP ACK or RTCP data). This approach is also referred as passive probing [7]. To take advantage of the positive aspects of both methodologies, many authors propose the use of integrated solutions, where passive and active measurements are combined to achieve more scalable monitoring systems [8] [9] [10].

Regarding time granularity, monitoring can be carried out off-line or on-line, i.e., based on a post-processing or real-time data analysis. Off-line monitoring is more oriented to guide long-term decisions and provide a broad view of the network operation, accounting and diagnostic. On-line monitoring is specially oriented to provide feedback to short or medium term network management and traffic control mechanisms, i.e., the monitoring outcome is required to drive reactive mechanisms so that traffic control decisions are not decoupled from the current network status. Currently, SLS auditing, formerly taken as an off-line task, assumes a crescent relevance as an on-line task [4], as customers are increasingly demanding in assessing the provided service levels on a near real-time basis. Figure 1 illustrates these concepts.

The RIPE NCC project, as an operational network-monitoring platform, has been a strong reference for the present work, taking in consideration its successes and limitations. In our opinion, positive aspects include the use of a dedicated measurement infrastructure, active measurements with adjustable probing, and the support for inter-provider network measurements. A current limitation is that the network is viewed as a single class environment, without service differentiation. NetQoS, a commercial QoS monitoring solution, having technical vendor support, is easy to deploy and maintain in production scenarios. However, it has the limitations of a closed and proprietary architecture, concerning the range of applicability scenarios and monitoring parameters.

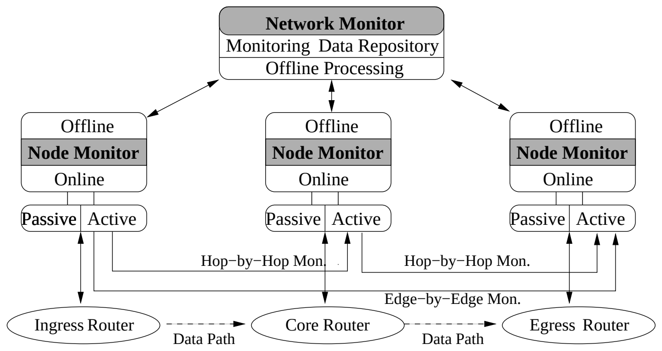


Fig. 1. Generic QoS monitoring architecture

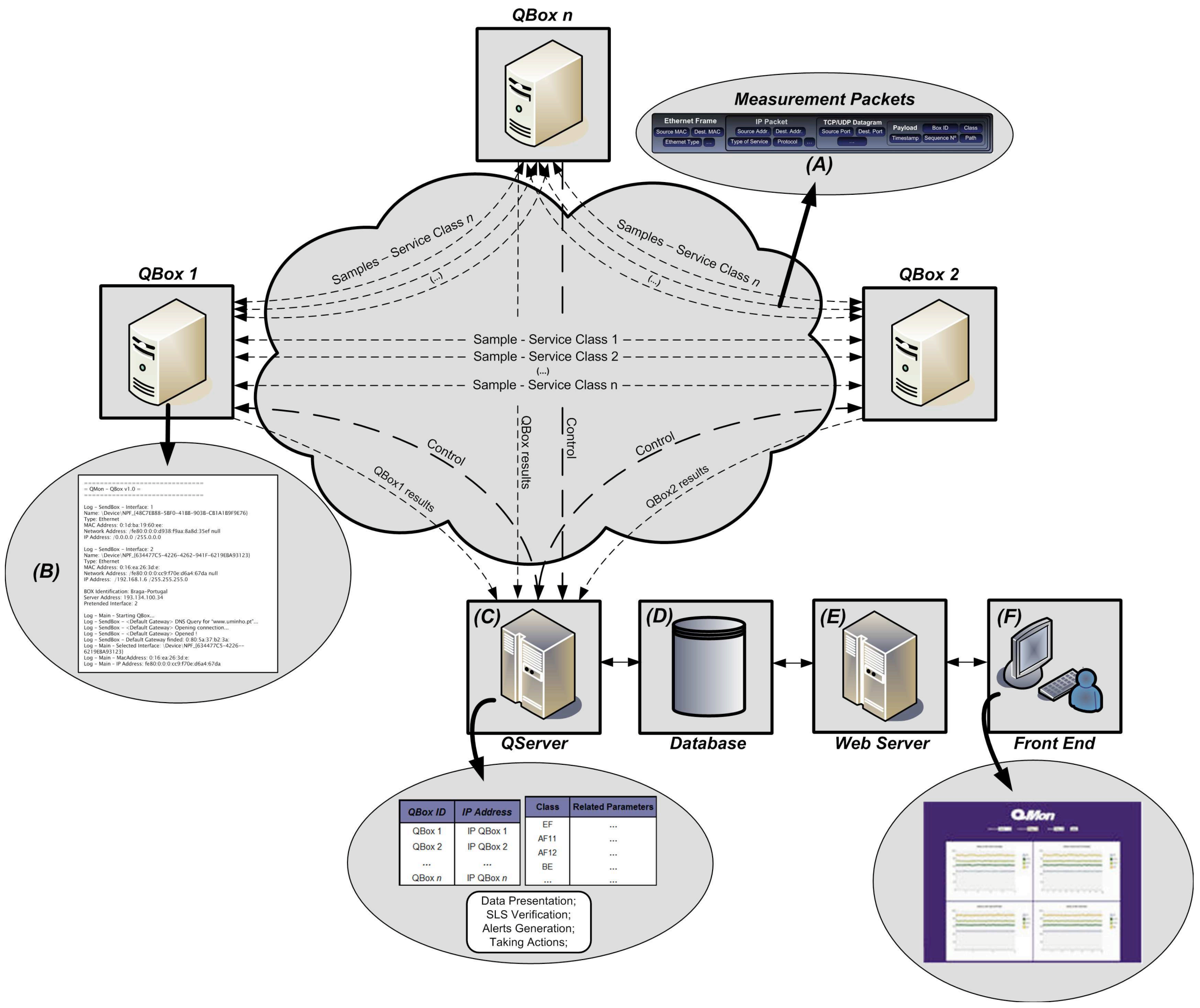
The main objective of QMon, as a flexible, open-source software monitoring tool, is to provide systematic QoS monitoring in multiservice networks. This is an essential task to: (i) keep track of ongoing QoS and network performance levels; (ii) verify Service Level Specification (SLS) compliance; (iii) provide feedback to traffic control mechanisms and trigger network recovery procedures; and generically (iv) support traffic engineering tasks. These aspects, along with the shortage of available service-oriented QoS monitoring tools, stress the relevance and real applicability of the proposed solution.

III. QMon Tool

A. Design goals and architecture

The main objective of this project is to develop a versatile monitoring tool able to perform QoS measurements in a per class basis, following the IETF IPPM Working Group [11] guidelines. The proposed monitoring architecture (see Figure 2) is grounded on the following design goals:

- *one-way measurements* – providing one-way measurements allows to solve the misleading impact that asymmetric routing may have on QoS measurements. In addition, it allows to identify, for instance, the direction in which a eventual problem is experienced;
- *active measurements* – active measurements between network boundaries, in which service levels need to be enforced, are particularly suitable for QoS control and SLS auditing. A generic traffic generation tool is included in the monitoring architecture to control the characteristics of probing traffic injected into each service class;
- *fully configurable* – our solution offers a wide range of configurable options in order to adjust it to different measurement scenarios. This versatility includes the configuration of per service probing, the definition of relevant metrics for each service class, and the identification of involved MPs;



```
==================================
= QMon - QBox v1.0 =
==================================

Log - SendBox - Interface: 1
Name: \Device\NPF_{48C7EB88-5BF0-41BB-903B-CB1A1B9F9E76}
Type: Ethernet
MAC Address: 0:1d:ba:19:60:ee:
Network Address: /fe80:0:0:0:d938:f9aa:8a8d:35ef null
IP Address: /0.0.0.0 /255.0.0.0

Log - SendBox - Interface: 2
Name: \Device\NPF_{634477C5-4226-4262-941F-6219EBA93123}
Type: Ethernet
MAC Address: 0:16:ea:26:3d:e:
Network Address: /fe80:0:0:0:cc9:f70e:d6a4:67da null
IP Address:  /192.168.1.6 /255.255.255.0

BOX Identification: Braga-Portugal
Server Address: 193.134.100.34
Pretended Interface: 2

Log - Main - Starting QBox...
Log - SendBox - <Default Gateway> DNS Query for "www.uminho.pt"...
Log - SendBox - <Default Gateway> Opening connection...
Log - SendBox - <Default Gateway> Opened !
Log - SendBox - Default Gateway finded: 0:80:5a:37:b2:3a:
Log - Main - Selected Interface: \Device\NPF_{634477C5-4226--
6219EBA93123}
Log - Main - MacAddress: 0:16:ea:26:3d:e:
Log - Main - IP Address: fe80:0:0:0:cc9:f70e:d6a4:67da
```

QBox ID	*IP Address*
QBox 1	IP QBox 1
QBox 2	IP QBox 2
...	...
QBox *n*	IP QBox *n*

Class	Related Parameters
EF	...
AF11	...
AF12	...
BE	...
...	...

Fig. 2. QMon Architecture

- *infrastructure independent* - in an end-to-end scenario, it is usual to have packets traversing different domains, following distinct administrative and configuration policies. Therefore, the interaction with network infrastructure and its particularities has to be minimal ;
- *flexible location of MPs* – MPs are here considered at network edges, however, they can be place at any network location;
- *scalable* – the performance of a monitoring solution needs to be, as much as possible, independent of the size of the network being measured. The edge-to-edge nature of the proposed monitoring solution allows improving scalability.

QMon encompasses two main elements: the QBoxes and the QServer. QBoxes are MPs strategically distributed in the network, typically at its boundaries, and their main task is to exchange probing traffic to compute the QoS metrics of each service class. The QServer gathers all measurements and QBoxes' configurations into a central database. Collected data remains available for subsequent analysis and support of management and traffic engineering actions over distinct time scales. These two elements are software-based, more specifically Java applications. QMon is, therefore, highly portable, easy to deploy and cost-effective. As mentioned before, to improve scalability the network is viewed as a black-box. Apart from QMon components setup, the solution does not impose special requirements from network nodes and does not depend on network configuration particularities. This is a clear advantage over hardware-based or proprietary monitoring approaches [4].

B. QServer

Qserver is a central component of the architecture, where the measurements from all QBoxes are stored. As measures are being received, they are inserted into a database (MySQL or Oracle), remaining available to be accessed by external entities, which can perform independent data manipulation. The status of all existing QBoxes and their configuration is also maintained in QServer. This allows data to be changed on-the-fly by sending them updating messages. Qserver is also responsible for notifying the QBoxes when a new one becomes active or inactive, or when new service measurement policies need to be deployed. The communication primitives exchanged between QServer and QBoxes will be described in the next section.

Note that, despite the central nature of data gathering, in the medium/short term, the measures remain available in the QBoxes to support distributed traffic engineering, avoiding the functional dependence and congestion of a single central entity.

C. QBoxes

QBoxes are responsible for generating, sending and receiving probing traffic in order to evaluate the relevant metrics for each service class. This means that the characteristics of probing traffic and the metrics to evaluate are service-dependent, according to a configuration previously received from the QServer for each service class. Receiving probing packets for metrics evaluation is a primary function of a QBox. Once the multiple metrics are calculated, they are reported to QServer.

In practice, the configuration parameters for the QBoxes may be derived from both negotiated SLAs and the internal ISP policies for service configuration and control.

IV. QMON COMPONENTS

A. QServer specification

1) Client-Server communication: Communication among QBoxes and the QServer is made through the protocol illustrated in Figure 3.

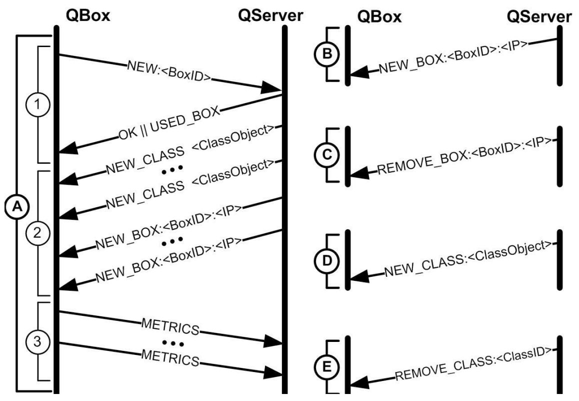


Fig. 3. QBox/QServer Communication Protocol

A. The initialization of a QBox comprises the following phases:
   1. Initially, the new QBox sends its identification to QServer, which registers the QBox and notifies other QBoxes that a new box is in place (B). This allows the QBox to be included in the measuring process and to start receiving probing traffic. If a QBox is already registered with the same identification a proper notification USED BOX is sent, otherwise the QServer sends an OK message;
   2. The QServer sends all the information that the QBox needs before sending probing traffic. This process is based on two distinct sequences of messages: one informing the characteristics of the defined service classes; the other providing each QBox IP address and identification. After this, the new QBox is ready to send and receive probing traffic to and from all QBoxes , according to the service classes characteristics;
   3. At this stage, every time a sample of probing traffic is fully received by a QBox, the metrics of the corresponding service class are calculated and sent to the QServer. This process is done repeatedly while the QBox remains operational. The QServer receives these messages and inserts their contents into a database.

B. After a new QBox is registered, the QServer notifies all other active QBoxes of its existence, using the already open sessions. Thus, adding new QBoxes to the platform dynamically does not interrupt the measurement procedure;

C. When a QBox shuts down, the existing session between this QBox and the QServer is closed. The QServer notifies this occurrence to other QBoxes in order to stop the measurement flow to the offline QBox;

D. Whenever a new service class is defined or updated in the database, its new settings are also sent to all the involved QBoxes. Thus, probing to new service classes may start without interrupting the measurement procedures for the remaining classes;

E. Similarly, whenever a service class is removed from the database, the QServer notifies all the involved QBoxes so that the obsolete class can be removed from their local information structure. Once again, the measurement procedure for the remaining service classes is not disrupted.

2) Database: The routing process in IP networks is usually dynamic, meaning that, from an end-to-end perspective, probing flows can be sent through different routes over time. Performing route pinning on probes helps identifying reasons for QoS variability. Thus, each set of QoS metrics is associated with the route found in the period of time that the measurement was carried out, i.e., the measuring time interval. For each sample, the values of the calculated metrics are stored in a MySQL database, according to the structure represented in Figure 4.

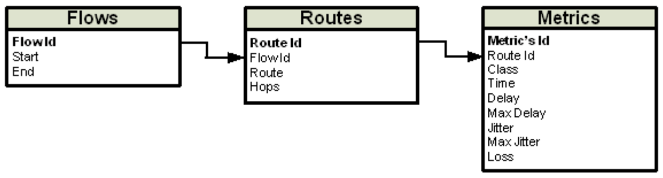


Fig. 4. Database scheme

Table Flows identifies the origin and destination of end-to-end flows between QBoxes, being assigned a unique identifier. Table Routes identifies the routes and their number of hops, each one associated to the existing flow identification. Table Metrics identifies the values of each metric. These metrics are differentiated per service class, and report to average values of a probing sample in the corresponding measurement time interval. Whenever a new set of metrics is added, the existence of a measurement flow between two QBoxes is verified, followed by the verification of the associated route, being created if it is still not defined. This relationship is obtained by forcing each record from Table Metrics to be related with a route between two QBoxes.

B. QBox specification

Each QBox has three main structures (updated on-the-fly by the QServer):

- QBox list – contains the identification and the address of all active QBoxes;
- Class of Service configuration – contains the configuration parameters of all classes, which are: protocol, port, number of packets per sample, time between samples and timeout of a sample;
- Samples - contains the timestamps of packets within a sample and the paths from the sending QBoxes;

and three main processes:

- Sending Process – responsible for creating and sending the samples;
- Receive Process – responsible for receiving the samples from all QBoxes;
- Server Communication Process – responsible for the communication with the QServer.

1) Sending process: An advantage of developing the sending process is to obtain full control of low-level packet building; this is achieved using the Jpcap library [12]. This control allows generating distinct probing patterns (or probing samples), adjusted to each service class being measured.

In more detail, there is one independent sending process for each Service Class, modulated as an ON/OFF source (see Figure 5). The sending process checks the service class configuration for the probing source parameters, the number of packets within an ON period and type of protocol (TCP/UDP) to use. During the ON period, the sending process builds a burst of packets with the corresponding Service Class identifier, sends it to every QBox within its scope. The inter-packet time is either deterministic or probabilistic. Therefore, in addition to Constant Bit Rate (CBR) sources, probing traffic may also follow a Variable Bit Rate (VBR) pattern, with a Poisson distribution regulating the inter-packet waiting time. During the OFF state, the sending process stands by for a configurable time period (a time value between samples). Probing variability is suggested to reduce the chance of traffic synchronization, as periodic network perturbations resulting from probing itself can drive the network into a synchronization state, which may end up affecting the measurements, leading to biased metrics.

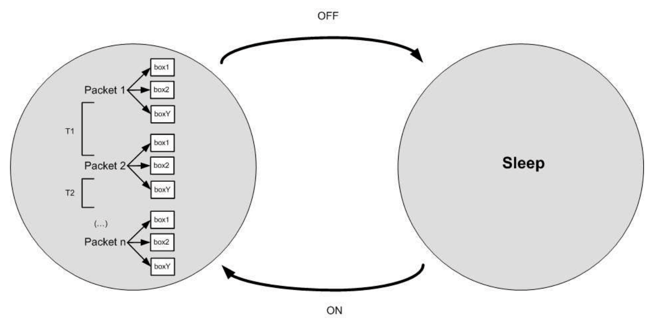


Fig. 5. ON/OFF process scheme; Y is the number of active QBoxes; T is the Poisson inter-packet time; n is the number of packets sent in a Service Class sample.

The traffic generation process is independent of the destination QBox, as no session is established. In this way, the measurement overhead and latency are reduced. As mentioned before, packet building is carried out resorting to the defined settings for the respective Service Class. Having full control of packet building opens the opportunity to use the probe packets payload to convey useful information for the destination box:

```
<boxId>::<serviceClassId>::<timestamp>::<seqNumber>::<path>
```

More specifically, <boxId> is the identifier of the current QBox generating the traffic; <serviceClassId> is the Service Class identifier; <timestamp> is the current system time at the origin QBox, which is going to be synchronized with the destination box using NTP (Network Time Protocol); <seqNumber> provides a sample identification function so that the destination box knows exactly the number of packets sent in this sample and, therefore, loss can be detected; <path> indicates the path between the origin and destination boxes[1].

The sending process described above is fully configurable and flexible. When a new service class is defined, the new QoS parameters will be automatically interpreted and measured in all destination boxes within its scope.

2) Receive Process: For probing traffic reception and analysis, all QBoxes are listening for incoming probes. When a new packet is received, a timestamp is added to it, and then it is validated regarding its origin and payload syntax. Payload data is verified in four stages: (i) the identification of the sender box should match the sender address of the packet; (ii) the identification of the service class is validated; (iii) the sender timestamp is checked against the receiver timestamp; and (iv) the sequence number should match the range of values expected for its class. If a validation stage fails, the packet is discarded.

After this validation and verification process, the information of the packet is gathered into a structure that contains the box identification, the class identification, the path, and the

[1]The path is evaluated using a traceroute-based approach carried out at the beginning of the ON state. The insertion of the path in all probing packets allows to increase their length and, in this way, avoid compression by the networks transferring them, which will affect the transmission time [4].

timestamps generated at sender and receiver. If the received packet is the first of a new sample, the QBox ID, the service class identification, the system time plus the timeout value defined for the class are inserted into a timeout table. The timeout table contains the time limit that a QBox should wait for the missing packets from a sample. When all the packets from a sample are received or when the timeout expires, the QoS metrics are calculated.

In the QoS metrics evaluation, three metrics are considered for each sample as illustrated in Table 1: mean delay, jitter and packet loss ratio. The maximum delay is also determined. Regarding jitter, Ri is the arrival timestamp and Si is the sender timestamp of packet i. When jitter cannot be evaluated (e.g. lack of packets), the error value "-1" is reported to the QServer. After evaluating the QoS metrics, the corresponding values are sent to QServer in a METRICS message, using the following syntax:

```
METRICS:<boxId>::<sendBoxId>::<path>::<timestamp>::<class>::
 <maxDelay>::<meanDelay>::<jitter>::<packetLossRatio>
```

The <boxId> is the identification of the current QBox; <sendBoxId> is the QBox which sent the sample; <path> identifies the path between <sendBoxId> and <boxId>; <timestamp> indicates the mean timestamp of the sample, i.e., the mean between the timestamp of the first and of the last packet within the sample; <class> is the identification of the Service Class; <maxDelay> is the maximum delay found in the sample; <meanDelay>, <jitter> and <packetLossRatio> are the calculated QoS metrics.

3) QServer Communication Process: When a QBox receives a notification from the QServer that other QBox has become active, the identification of the new box is inserted in the list of active QBoxes. A new record to save its samples is also created and inserted in the metrics' structure. If a QBox becomes inactive, the corresponding records are deleted from the list of active QBoxes and from the metrics' structure. When the QServer informs about a new service class, the QBox inserts those values in the Service Class configuration structure and starts the Sending Process for that class. If the QServer notifies that a service class is due to be removed, the QBox stops the Sending Process for that class and removes the corresponding record from the structure.

V. QMon implementation aspects

As mentioned before, the main programming language used in the development of QMon was Java; and Jpcap library [12] for low-level network control.

TABLE I
STANDARD ONE-WAY QOS METRICS

$MeanDelay = \frac{\sum_{i=1}^{NumOfPkts}(ArriveTstamp_i - SendTstamp_i)}{NumOfPkts}$
$Jitter = Jitter + \frac{\lvert D(i-1,i)\rvert - Jitter}{16}$ $D(i,j) = (R_j - R_i) - (S_j - S_i)$
$PacketLossRatio = \frac{ExpectedPackets - ReceivedPackets}{ExpectedPackets}$

This section will focus on the implementation details of the architecture main components: the QServer and QBox.

A. QServer

1) QServer/QBox Communication: Upon a connection requests from QBoxes, the QServer records each IP address and QBox ID, and a thread indicator related to each ongoing QBox/QServer TCP session. In this way, the QServer is able to notify active QBoxes if any state change occurs. This is carried out through the invocation of methods implemented in the communication thread that each QBox has with the QServer. When QBoxes or service classes are added or removed, these methods are invoked, for addressed QBoxes.

Each QBox sends to the QServer messages with the measurement values obtained for each sample, so they can be insert into the database. To interact with the database a generic API has been developed, which allows performing queries and statements to MySQL or Oracle 10g databases. An extension to this API allowing the QServer to insert, remove and query our database has also been created.

2) Front-end: To access the measurement results, a web front-end has been developed, allowing an easy access to this information, with a distributed profile. The front-end consists of a php page rendering different graphs, which illustrates, for all service classes, the QoS metrics behaviour between QBoxes over distinct, configurable time scales. The process of obtaining the metrics' values is carried out querying the database, and filtering the entries for the required time length, origin, destination and metric. The identification of the QBoxes presented in the existing front-end roll boxes is dynamic. So when a reference to a QBox is inserted or removed from the database, the information in the front-end is automatically updated.

B. QBox

QBox has a simple role in the whole architecture, with a plug-and-play profile. For this reason, the QBox application has a straightforward text-based interface that guides the user through the initial configurations. Initially, all active network interfaces that can be used by the application are shown and detailed:

```
==============================
=    QMon - QBox v1.0    =
==============================

Log - SendBox - Interface: 1
  Type: Ethernet
  MAC Address: 0:1d:ba:19:60:ee:
  Network Address: /fe80:0:0:0:d938:f9aa:8a8d:35ef null
/0.0.0.0 /255.0.0.0

Log - SendBox - Interface: 2
  Type: Ethernet
  MAC Address: 0:16:ea:26:3d:e:
  Network Address: /fe80:0:0:0:cc9:f70e:d6a4:67da null
/192.168.1.6 /255.255.255.0
```

Next, the user is questioned for essential information to proceed with the application initialization:

```
QBox Identification: Aveiro
QServer Address: 192.168.1.4
Interface: 2
```

The *QBox Identification* is the key for distinguishing the QBox in the database. The *QServer Address* has to be the one used to establish the session between the QBox and the QServer. Finally, the *Interface*, chosen by the user from the available network interfaces, is the one used to send and receive probes. After the initial configuration, the QBox enters two distinct procedures: packet building and sending; and packet reception and analysis. These procedures, responsible for the behaviour described in Sections IV-B1 and IV-B2, are briefly presented below.

1) Packet building and sending procedure: When the QBox application is initiated, a main thread is going to be actively waiting for QServer communications. When a NEW_CLASS message is received from the QServer, the current thread processes it, launching a new one, which works with the received parameters. This is where the sending process starts for a Service Class. The new launched thread is going to be active while the Service Class is registered on the local database. When a REMOVE_CLASS message is received from the QServer, the class entry is removed and the thread is dropped. This sending process follows the ON-OFF behavior explained previously.

The *sendSample* method creates samples according to the Service Class configuration parameters, and forwards them packet-by-packet to each active QBox registered. Prior to sending a sample, it traces the route for each destination QBox, using ICMP packets. This allows discovering the traversed path beforehand so that it can be inserted in the packet payload for additional information. This information is useful, for instance, to determine the number of hops and to detect route changes, which may justify significant variations in the QoS measures.

After building the payload, the probing packet is marked according to the class *type-of-service* and sent. Once all packets within a sample are sent, the process is restarted and kept active while the Service Class is registered or the user does not stop it. To allow a plug-and-play profile, the default router for the current network is automatically detected.

2) Packet reception and analysis procedure: The packet reception and analysis procedure is implemented in two threads. The first one is responsible for capturing the packets and analyse them. For a correctly received packet, the timestamps (send and arrive) are saved in an ArrayList, according to their sequence number. There are multiple ArrayLists to save the timestamps of all Service Classes from active QBoxes. Those ArrayLists are saved into a HashMap, having as key the sender QBox identification and the Service Class to which those timestamps belong. The maximum size of each ArrayList is the number of expected packets from each Service Class. When an ArrayList reaches its full capacity, the sample is complete and the QoS metrics are calculated and sent to the QServer.

The second thread is responsible for verifying the timeout table, searching for expired time values. When the first packet of one sample is received, the current time of the system and the timeout value defined for the corresponding Service Class, are inserted into a HashMap, where the key is the QBox and Service Class identifier. Along the time, this thread compares the system time with the timeout values in the table; if a timeout values is exceeded, the QoS metrics for that sample are calculated and sent to the QServer.

C. Time synchronization

The accuracy of measurements depends on appropriate time synchronization methods. Inaccuracies on time reference of communicating QBoxes results in unreliable measures between them. Initially, and for testing the QMon application prototype, NTP is used to synchronize all QBoxes and the QServer. A dedicated NTP server, which synchronizes from three public NTPs servers, is used to provide the same time reference to QMon components, reducing inaccuracies that could result from using distinct synchronization sources. As future work, this synchronization method shall be improved.

D. QMon prototype testing

At this stage, testing QMon prototype has involved assuring the correctness and robustness of the developed software components. Based on a prototype network environment comprising service classes with distinct QoS requirements, QMon was tested in two stages. Firstly, the sending process was verified resorting to Wireshark tool, which confirmed the correct time and space characteristics of probing traffic. This was cross-checked with log analysis in each QBox. Secondly, to test the receiving process, synthetic packets with specific timestamps were generated to force distinct delay and jitter for each Service Class. Comparing the QoS measures obtained with QMon with the expected measures considering the packets' timestamp, the correctness of the measurement process was verified.

To visualize the QoS measurements, a web interface updated in real-time was created (report to Figure 2). Each graph represents the temporal series of a specific metric between two QBoxes, calculated resorting to the metrics stored in the database. Graphs measurements in short, medium and long time scales can be rendered, depending on the user configuration.

E. Steps ahead

At the present stage, the Java QMon prototype is fully operational, stable and tested in a local testbed. A short-term aim is to extend this testing to PlanetLab in order to assess and validate the tool over a large-scale network environment. We also intend to further develop or improve several aspects of the proposed QoS monitoring solution, namely: (i) to provide authentication support for users and QBoxes; (ii) to tune and enhance the probing process; (iii) to consider additional monitoring methodologies, such as passive capture of available data within the network nodes along the path, in order to enhance QoS evaluation of existing service classes; and (iv) to reduce the database size and the volume of traffic exchanged between the QServer and the QBoxes (strategies for aggregate measurements and for low-overhead metrics' dissemination are under study). Currently, the configuration parameters are user defined and are the same for all QBoxes. We are planning to derive these service parameters from the negotiated SLSs, sending them only to the QBoxes within the scope of the service contracted in the SLSs.

VI. Conclusions

In this paper, QMon - a flexible QoS monitoring tool for multiservice networks - has been proposed and implemented. This tool is able to measure the QoS of distinct service classes between network measurement points (QBoxes). QBoxes may be dynamically added to or removed from the infrastructure without disrupting monitoring. The measurement results can be accessed directly from the monitoring database or from a web interface available on the QServer. QMon, as a multiplatform and generic tool, is a versatile and cost-effective QoS monitoring solution to be deployed in multiservice network environments, being useful to assist traffic engineering tasks, service management and auditing.

References

[1] Almes, G., Kalidindi, S., Zekauskas, M.: A One-way Delay Metric for IPPM (RFC 2679) (1999)

[2] Almes, G., Kalidind, S., Zekauskas, M.: A One-way Packet Loss Metric for IPPM (RFC 2680) (1999)

[3] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., Zekauskas, M.: A One-way Active Measurement Protocol (OWAMP) (RFC 4656) (2006)

[4] Georgatos, F., Gruber, F., Karrenberg, D., Santcroos, M., Susanj, A., Uijterwaal, H., Wilhelm, R.: Providing Active Measurements as a Regular Service for ISP's. Proceedings of the Passive and Active Measurements Workshop PAM2001, Amsterdam (2001)

[5] http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html.

[6] Whitner, R., Pollock, G., Cook, C.: On Active Measurements in QoS-Enabled IP Networks. PAM'02, Fort Collins CO (2002)

[7] Corral, J., Texier, G., Toutain, L.: End-to-end Active Measurement Architecture in IP Networks (SATURNE). PAM'03 (2003)

[8] Asgari, H., Trimintzios, P., Irons, M., Egan, R., Pavlou, G.: Building Quality-of-Service Monitoring Systems for Traffic Engineering and Service Management. Journal of Network and Systems Management **11**(4) (2003) 399..426

[9] J.C. et al: SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks. 6th IFIP/IEEE MMNS'03 (2003)

[10] Lima, S.R., Sousa, P., Carvalho, P.: Enhancing QoS Metrics Estimation in Multiclass Networks. 22nd Annual ACM Symposium on Applied Computing (ACM SAC'07), Track on Computer Networks, Seoul, Korea (2007)

[11] Internet Engineering Task Force, IP Performance Measurements Working Group (IETF IPPM-WG)) http://www.ietf.org/html.charters/ippm-charter.html.

[12] http://netresearch.ics.uci.edu/kfujii/jpcap/doc/.