INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Sonda de avaliação de desempenho de Serviços em rede IP móvel

Abel David Azevedo Martins da Silva

Dissertação para obtenção do Grau de Mestre em

Engenharia de Redes de Comunicação

Júri

Presidente: Prof. José Manuel Rego Lourenço Brázio

Orientador: Profª Teresa Maria Sá Ferreira Vazão Vasques

Vogais: Prof. Paulo Rogério Barreiros D'Almeida Pereira

Novembro de 2007

Abstract

Today there is an increasing demand for network monitoring, specifically to quality of service monitoring. These type of monitoring is becoming crucial for service providers as it can reduce costs associated with network provisioning and also increase customers satisfaction.

This document describes a monitoring system designed to monitor the QoS on a network. This system is based on a number of components that together form a robust and scalable network monitoring system.

The system is divided in three components, a core system, fixed probes and mobile probes. The core system offers the means to configure the probes and to see the monitoring results. The probes perform the monitoring tests based on configurations from the core system. Finally, these tests are performed between the fixed and the mobile probes.

This document also presents the results of some tests performed by this systems while monitoring a VoIP service.

Keywords: QoS quality service monitoring probes

Resumo

Actualmente existe uma maior necessidade de monitorização de redes de comunicação, mais especificamente de monitorização da qualidade de serviços que assentam nessas redes. Este tipo de monitorização está cada vez mais a tornar-se crucial para os fornecedores de serviços porque pode reduzir custos de aprovisionamento da rede e pode também melhorar a satisfação dos clientes.

Este documento descreve um sistema de monitorização desenhado para monitorizar a qualidade de serviços numa rede. Este sistema é baseado num conjunto de componentes que juntos formam um sistema de monitorização de redes robusto e escalável.

Este sistema está dividido em três componentes, um sistema central (core system), sondas fixas (fixed probes) e sondas móveis (mobile probes). O sistema central oferece os meios para configurar as sondas e visualizar os resultados das monitorizações. As sondas realizam os testes com base nas configurações do sistema central. Por fim, estes testes são realizados entre sondas fixas e sondas móveis.

Este documento também apresenta resultados de alguns testes efectuados por este sistema durante a monitorização se um serviço de VoIP.

Palavras chave: QoS qualidade serviço monitorização sondas

Index

Figure Index

Table Index

Acronyms Index

ADSL	Asymmetric Digital Subscriber Line
CSV	Comma-separated Values
DBMS	Database Management System
EJB	Enterprise Java Beans
FP	Fixed Probe
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
IP	Internet Protocol
LAN	Local Area Network
MP	Mobile Probe
NTP	Network Time Protocol
PCE	Probes Communication Endpoint
PIR	Policy Information Repository
QoS	Quality of Service
RPC	Remote Procedure Call
RTT	Round-trip Time
SLA	Service Level Agreement
SLS	Service Level Specification
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
TRR	Test Results Repository
UDP	User Datagram Protocol
VoIP	Voice over IP
VPN	Virtual Private Network
WMI	Web Management Interface
XML	Extensible Markup Language

1 - Introduction

1.1 - Motivation

Today there is an increasing demand for network monitoring, specifically to quality of service (QoS) monitoring. This demand is related to the necessity of satisfying the client needs. By monitoring the network QoS it can be possible to anticipate problems. These problems can be caused by link congestion, network sub-provision or any other kind of situations that affect the user experience. If a problem affects the users experience it makes them unsatisfied with the service being provided. If the users get unsatisfied, the service provider might lose them as customers and this represents financial losses. Instead of letting the users experience problems and losing them as customers, the service provider can be pro-active by using a network monitoring system to anticipate the problems and correct them. This way, users are more satisfied with the service, this represents financial wins due to the increase of customers subscribing this service and the increase of the consumption of this service.

There are many tools available to perform network monitoring but almost all of them depend on specialized hardware and are some kind of “black boxes”, meaning that the client can't change them. This can be problematic, as the service providers may want to adapt these tools to many parameters of their network. Also, the use of special hardware can also be a problem because the service provider may not want to install new hardware along its network infrastructure.

A solution based solely on software can bring advantages, as the service provider may deploy it very easily along its network infrastructure and start monitoring the network. Also, being solely based on software, this solution might be supported by different devices, meaning that a service provider can develop a component that its clients can run themselves to perform problems traceability. Finally, if this solution is open to customizations, the client can adapt it to monitor its own services.

1.2 - Goals

The solution being developed will have the goal of performing network QoS monitoring. It will be a software based solution open enough to allow customizations to the services being monitored. The solution will support important features like fault tolerance or load balancing and it will have a centralized management system to configure it.

Due to some external factors, the architecture has already some characteristics decided. One of these characteristics is that the monitoring system will perform active monitoring, active monitoring will be explained further in this document. Another characteristic is that the monitoring system will perform the monitoring using two different kinds of monitors, one will be fixed and the other mobile, this will be explained further in the architecture section of this document.

In the end, in order to prove the functionality of this system, the monitoring system will be con-

figured to monitor VoIP call QoS on the network. Some tests will be performed to show that the system can display the performance of the network when a VoIP call is being made.

1.3 - Contributions

Some contributions of this work are explained next.

The monitoring system will perform QoS monitoring.

The monitoring system will have the ability to be extended to monitor any IP service.

The monitoring system will be able to perform network monitoring on any part of the network.

The monitoring system will be distributed by components and scalable.

1.4 - Structure of the Document

The rest of this document is organized as follows.

Section 2 discusses related work. This section contains an analysis of works in the same domain as this work. In the end there is a comparison of the characteristics of the works described with the goals of the work being developed.

Section 3 presents the proposed architecture of the system. This section will describe in detail all the aspects of the architecture of this monitoring system.

Section 4 describes its implementation. In this section the implementation details of the monitoring system will be described. Some of these details are: interfaces, data model, and communication model, among others.

Section 5 presents tests performed with the system developed. The tests were performed monitoring VoIP QoS. Some graphics with the results will be presented.

In section 6 some conclusions are presented along with possible future work.

Finally, annex A contains the installation and user manual.

2 - Related Work

This work is related with three QoS domains.

The first is End-to-End and Distributed Monitoring that is related with the distribution of the components of the system.

The second is Active and Passive Monitoring, this one represents the monitoring mode, *i.e.* either the components generate traffic on the network to infer the QoS or they scan it.

The third domain is QoS Metrics, this one refers to the metrics used to infer the QoS based on the results of the tests.

Each of these domains will be explained next.

2.1 - QoS Domains

2.1.1 - End-to-End and Distributed Monitoring

Distributed monitoring differs from end-to-end monitoring for being more complete. Distributed monitoring consists in measuring the QoS in many points of a path instead of performing the monitoring only on the two edge points of that path. In case of QoS degradation, distributed monitoring can provide more information because the network manager benefits from monitoring many points gathering information along the entire data flow path.

Tham *et al.* [1] proposed a distributed monitoring system called RM-based. This system has several main components, Relevant Monitors (RM), a real-time application name server (RTANS) and Network managers (Analysis applications).

The system works like this, all RMs must previously know where the RTANS is located and the RTANS must be functioning. During the monitoring the RMs record the real-time traffic flow attributes and the reference to themselves in the RTANS. The flow attributes include source and destination addresses, traffic counters (*e.g.* packets and bytes) and other elements. In the RTANS, the RMs of each flow are ordered by traffic attributes in order to provide a RTA (real-time application list). Finally, a network manager chooses a flow from the RTA and retrieves a list of all RMs that are part of the flow. With that list, the network manager contacts each one of the RMs and asks all the information needed.

A scheme of this system is shown in Figure 1.

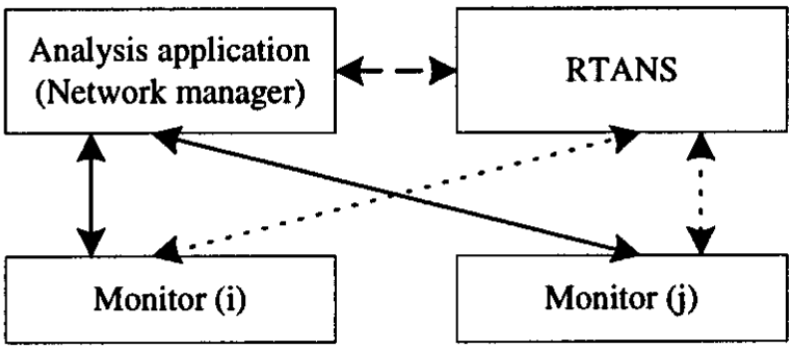

Figure 1: RM-based Model

2.1.2 - Active and Passive Monitoring

Active monitoring consists on generating traffic on the network in order to observe its behaviour. This generated traffic should be as most as possible identical to real traffic. Passive monitoring consists on observing real traffic on the network and providing results based on the observation. In general, the advantages and disadvantages for each mode (active and passive) are described next.

Regarding active monitoring, this mode has the advantage of having the results very close to the reality as the monitoring is performed by emulating a service on the network, *i.e.* to monitor a VoIP call, the monitoring system emulates a call on the network and obtains the results from this emulation. Another advantage is the time needed to obtain the results as there is no need to perform any kind of special analysis of the data. One of the disadvantages of this mode is the impact on the network, this happens when the components are generating traffic on the network. Another disadvantage is that the results represent a state of the network that is not the same as when the network is not being tested. Finally, another disadvantage is the need to exchange data between the components responsible for performing the monitoring.

Regarding passive monitoring, this mode has the advantage of not causing an impact on the network because the generated traffic is negligible (if it even exists). One big disadvantage on this method, when using two point passive monitoring (using two components to perform monitoring), the components on both points must have the clocks synchronized, also there is no great scalability on large scale networks. When using a single point monitoring (using only one component to perform monitoring) it is only possible to monitor TCP based services.

Maia *et al.* [2] proposed a system that monitors multimedia content flows in mobile environments and also improves the user experience of those contents. To make this possible, new communication interfaces are used in a Client-Proxy/Server model as shown in Figure 2.

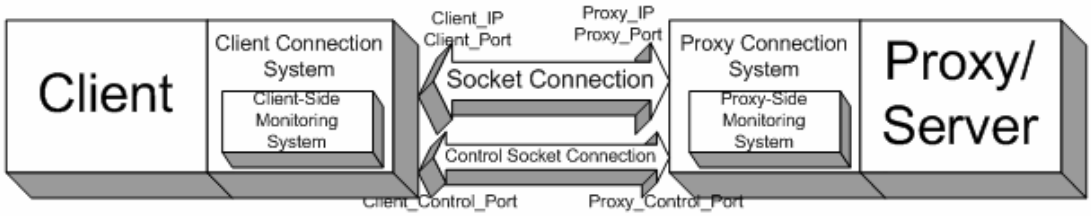

Figure 2: Socket Masking

Using these new interfaces, when a user is watching a video stream, the new interface on the client side can provide information about the quality of the link to the new interface on the Proxy/Server side. With this information, the Proxy/Server can adapt the stream to these conditions by changing the number of frames per second, resolution, and quality of image or any other element that improves the user experience.

This system is based on a passive monitoring model. The system listens to the communications between the client and the proxy/server and infers the quality of the stream. Beyond the purpose of monitoring this system also takes actions to improve the user experience.

Asgari *et al.* [3] proposed a system to monitor SLAs (Service Level Agreements). This system is based on five main components: the Node Monitor, the Network Monitor, the SLS Monitor, the Monitoring Repository and the Monitoring GUI.

The Node Monitor is a component of the system attached to a router which is responsible for performing the monitoring. In this system, the monitoring can be active or passive. The active monitoring is performed between Node Monitors. The passive monitoring is done by consulting a router to which the Node Monitor is attached. After gathering the results, the Node Monitor is responsible for performing some data analysis of those results gathered. Also, while analysing the results, the Node Monitor is responsible for sending notifications if it detects any threshold crossing.

The Network Monitor is basically a component responsible for performing post-processing of measurement data and build a physical and logical network view.

The SLS Monitor is responsible for determining if any violations of the SLAs occur. The SLS Monitor uses the SLS (Service Level Specification) part of the SLA to obtain the thresholds for each SLA to be monitored. To monitor the SLSs, the SLS Monitor accesses the data collected by the Node and Network Monitors stored on a repository and combines that data with each individual SLS. After combining the results with each SLS, the SLS Monitor generates reports about these combinations.

The Monitoring Repository is a repository responsible for storing results data from monitoring components and to store configurations and information about active monitoring processes. The measurement data stored in the Monitoring Repository is used by the Network Monitor, the SLS Monitor and the Monitoring GUI to perform subsequent analysis.

The Monitoring GUI is an user interface to consult the measurement results. It shows graphical views of monitoring statistics from the monitoring data store.

In Figure 3 it is possible to see the architecture of this system and to identify the components described before.

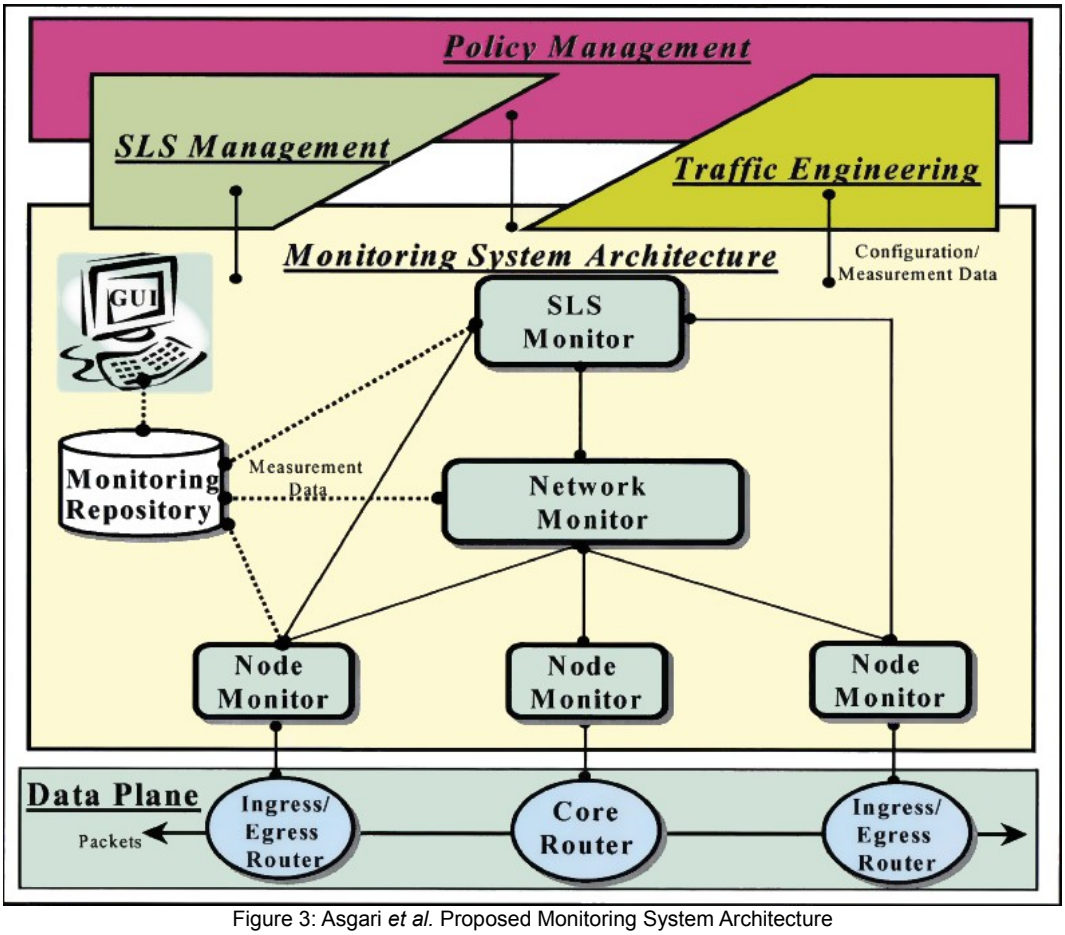

Figure 3: Asgari *et al.* Proposed Monitoring System Architecture

2.1.3 - QoS Metrics

To measure the QoS of a link, a monitoring system has to monitor communications occurring on that link. To avoid incorrect measurement of QoS, the system has to filter which packages to process from that link, *e.g.* when monitoring a VoIP call, the system will only process the packages related to that call and ignore all the others. By processing those packages, or not processing because they never reached their destination, the system can obtain many parameters like packet loss, delay, jitter, round-trip time among many others.

Yang and Chou [4] proposed a series of parameters to measure QoS. These parameters do not include only parameters measured during the communications but also parameters related with capabilities of the systems involved on the monitoring like the nodes or routers.

The proposed parameters are divided in four categories: Node, Router, Link and Generic. Each one of these categories is described in the next sections.

- Node Category

  Inside the node category we have the parameters described in Table 1.

Parameter	Information
Node Processing Capability	When transmitting multimedia data, the node must have a very short computing delay.
Node Buffer size	The buffer size must be correctly configured in order to maintain a stable transmission rate and to reduce the jitter.
Number of Nodes	As the number of nodes increase in the network, the possibility of traffic jams increase and the efficiency of the network decreases.

Table 1: QoS Parameters - Node Category

- Router Category

  In the router category we have the parameters described in Table 2.

Parameter	Information
Router Processing Capability	The router processes the available resources fairly allocations for each packet, the flow status determination and the transmission path cost. At the end, to maintain QoS, the routers need high processing capabilities.
Router Buffer Size	A correctly configured buffer size is needed to synchronize the transmission rates and incoming and outgoing packets among senders and receivers
Number of Intermediate Nodes	The number of intermediate nodes affect the computation delay, transmission delay and negotiation time among nodes.
Network Throughput Range	This indicates the actual status of network traffic that can be represented by the current volume of data flow and the amount of available resources.
Packet Life Time	This represents the maximum interval time between the time a packet as been sent and an acknowledge reply as arrived to the sender.
Packet length	This is the maximum length of a packet sent by an user.
Level of QoS guarantee	This is the level of QoS the application asks to be granted.

Table 2: QoS Parameters - Router Category

- Link Category

  In the link category we have the parameters described in Table 3.

Parameter	Information
Bandwidth	This represents the effective width of spectrum in a link.
Data rate	This is the capacity of the link, it represents the amount of data that can be transmitted in a period of time.
Propagation Delay	This is equal to the distance of the link divided by the velocity of propagation.
Link weight	This is a value assigned to a link in order to have different costs for links.

Table 3: QoS Parameters - Link Category

- Generic Category

  In the generic category we have the parameters described in Table 4.

Parameter	Information
End-to-End Delay	This represents the total delay from source to destination nodes of the network.
Delay Jitter / Delay Variation	This is the difference between the highest and lowest values of End-to-End Delay.
Acceptable error rate	This represents how much an application can tolerate errors in packets.
Packet loss ratio	This is the ratio of packets that never arrive to their destination.

Table 4: QoS Parameters - Generic Category

2.2 - Summary

After reviewing all the related work and relating it with the purpose of the system proposed here some main guidelines had to be decided.

Regarding the architecture, it will have some similarities with the architecture proposed by Asgari *et al.* [3] on his system to monitor SLAs. When relating the goals of the system proposed here with the characteristics of the system Asgari *et al.* proposed, some conclusions can be determined. One big difference is the main goal of both systems, Asgari's *et al.* system was created to monitor SLAs, and the system proposed here has the main goal of monitoring specific services. Asgari's *et al.* system also has the disadvantage of having to place Node Monitors along the network and this should be avoided to prevent changing the existing infrastructure. Also, the system proposed here must be very adaptable, *i.e.* the system must have the ability to be easily customized to monitor different services, and Asgari's *et al.* system has no focus on that characteristic. Finally, another goal of the system proposed here is to have mobile monitors, so that the monitoring does not occur only from the inside of the network; or by network administrators, the monitoring has to performed from any place and must be possible to perform by anyone, including the network customers.

As for the similarities, the proposed system will also be based on some centralized infrastructure to store configurations and results. To perform the monitoring, the system will have specialized entities for that purpose.

The proposed system will be an end-to-end monitoring system. The need to avoid big changes to the network infrastructure makes this the only choice, but the monitoring system will allow the placement of the monitors in any part of the network.

Regarding active and passive monitoring, the system will be designed to perform active monitoring but will not be limited to this. Besides being developed for active monitoring, it will still be possible to adapt the system to perform passive monitoring.

The QoS metrics used by the system will be specific to each type of service being monitored,

e.g. when monitoring a VoIP service it will be important to monitor the delay, jitter and packet loss. Each service will have its own specific QoS metrics to be monitored.

Finally, the refinement and data aggregation will depend on the QoS metrics being monitored. In the last example, when monitoring a VoIP service, probably the delay and jitter can be reported by minimum, maximum and average, the packet loss can be reported by received packets and lost packets.

3 - Architecture

3.1 - General Architecture

The architecture of the proposed system consists primarily on three components, the fixed probes, the mobile probes and the core system. These components are organized on a layered architecture as illustrated in Figure 4.

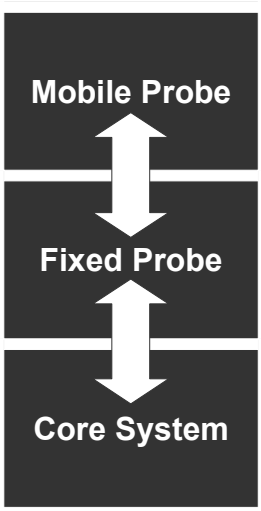

Figure 4: Three layer architecture

The main function of this system is to monitor the QoS of any IP service, like VoIP, video calls or data transfer. To do this, a mobile probe performs some active tests with a fixed probe. These tests may include many different services like those mentioned before. After finishing the tests, the probes perform data analysis and then submit the results to the core system.

In order to have the system working, policies are used to control the probes. These policies are configured through an interface in the core system and stored in a database, also in the core system. The probes are responsible for fetching these policies from the core system, in order to perform the tests as configured.

After fetching the policies, the probes can execute tests to analyse the behaviour of the network, after finishing the tests they perform an analysis of the data collected during the test and report the results of this analysis to the core system.

The fixed probes are responsible for two main activities. One is to wait for a request from a mobile probe to start a test, when this happens, the fixed probe and the mobile probe synchronously perform the test. Another activity is to act like a bridge between the mobile probe and the core system, this allows the mobile probe to request policies and submit results.

The mobile probes are responsible for starting the tests, they communicate with a specific fixed probe defined in the policies and submits a request to start a test. When the test is finished, the

mobile probe submits the results to the fixed probe which will forward them to the core system. The mobile probes are supposed to be untrusted elements in this system, the goal is to have mobile probes that can be installed by network administrators to periodically monitor the network QoS or disposable probes that are executed one time by network clients to diagnose any kind of problems.

This architecture has the advantage of not having the need to expose to core system to the entire network, this way the fixed probes can use some kind of VPN to communicate with the core system. If there was a need for the mobile probes to communicate with the core system it would be necessary to expose the core system to all the network and this wouldn't be a good choice because it would compromise the security of the system. As an alternative, one could provide a way to all the probes to establish some kind of VPN but also, this wouldn't be a good choice in terms of scalability and security because as stated before, the mobile probes are intended to be untrusted components of the system.

In Figure 5 it's possible to identify mobile probes on the left, fixed probes on the right and the core system and its components on right top.

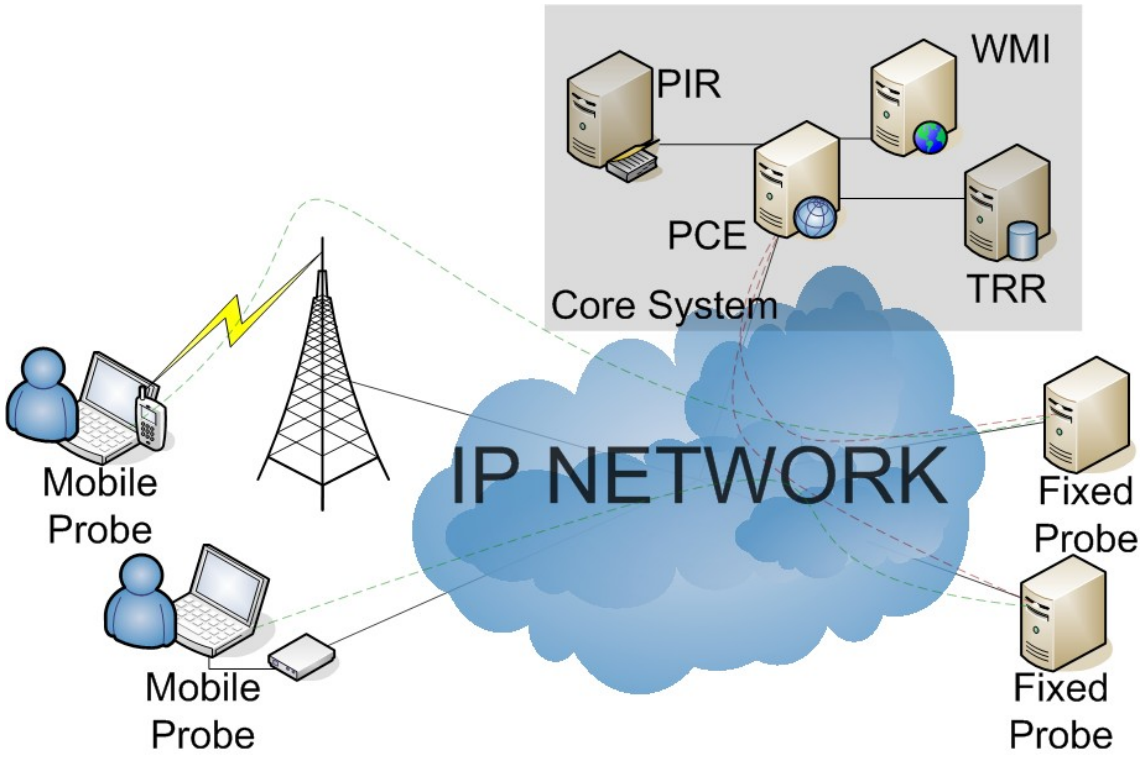

Figure 5: General Architecture

In this monitoring system, the core system is responsible for storing the policies, to store the tests results, to provide a communication interface to the probes and provide interfaces to perform administration tasks and view the tests results. In the administration interface it is possible to configure the probes and the tests to be performed. In the visualization interface it is possible to the view reports of the tests performed. These reports can be organized by categories like days, months, probes, types of tests among others.

With this architecture it is possible to configure a very large and distributed infrastructure by placing the mobile probes on any location of the network. They only need to communicate with a fixed probe. The fixed probes, besides having to allow mobile probes to communicate with them, they need to communicate with the core system. The fixed probes can be placed anywhere on the network as long as this two requirements are met. The core system can be placed in a data center in order to protect the information and also protect the system against misuses.

This architecture allows the monitoring system to have a very high performance since all of the system components are separated from each other. This characteristic allows the placement of each component on a different piece of hardware overcoming any performance issue related to the hardware platform. Regarding the tests performance, it can also have high performance because the tests can be distributed along different mobile and fixed probes.

For load balancing, the architecture of the core system allows all of its components to be replicated. This improves even more the performance capability of the system basically because the probes contact different instances of the communication interface on the core system. Then, this instances contact different instances of the other components. The same happens with the administration and visualization interfaces, both can have more than one instance.

Regarding fault tolerance, this architecture provides the needs to configure the system in a fault tolerant way. As stated before, all the components can be replicated, this has advantages for load balancing and consequently for performance and also for fault tolerance because when one instance of a component fails there is another instance running. Even if the components aren't replicated the system provides some kind of fault tolerance by allowing the system to continue to execute in a degraded form when one or more components fail. When this happens, the part of the system that failed will stop working, *e.g.* if both the administration and visualization interfaces fail, there is no way to configure the system or to view the tests results but the probes will keep working without any problems as they can still fetch policies, perform tests and submit the tests results.

To prevent unauthorized accesses, changes and submission of reports the system the core system has to force authentication to perform this actions. One of the authentication mechanisms must be on the PCE to prevent the submission of false reports. Also, in the interfaces must be another authentication mechanism to require the users to authenticate before configuring the system or viewing reports. Due to the type of data exchanged between the probes and the core system, there is no need to encrypt all the communications, even though, this system can be extended to support this.

Finally, this architecture provides the ability to perform upgrades to the system in an unobstructive way because as described before, it supports the failure (upgrade in this case) of components. If the system is configured in a replicated form it will even allow upgrades without any service degradation. If it is not configured in a replicated from, only the component being upgraded won't be working during the upgrade.

3.2 - Probes Architecture

The tests are initiated by the mobile probes, either by a previously configured schedule or by demand when requested by a network administrator or a network user. The mobile probe performing the test checks which fixed probe is supposed to be used for the corresponding test and starts communicating with that probe. For each test, the mobile probe asks the fixed probe if it supports that corresponding test; if it does, the mobile probe requests the fixed probe to prepare for that test in order to initialize anything needed for the test; then, after both probes are prepared, the mobile probe asks the fixed probe to start the test and the fixed probe responds reporting that will start the test shortly and the mobile probe starts it too. When the test finishes, the fixed probe submits its results to the core system and the mobile probe submits its results to the fixed probe. This fixed probe accepts the results and afterwards will submit them to the core system. The fixed probe doesn't submit the mobile probe results synchronously so the mobile probe can be relieved from the data as soon as possible because the hardware limitations of the mobile probes are not known and memory or battery can be reaching the limits.

An high level presentation of the communications between the fixed probes, the mobile probes and the core system is shown in Figure 6.

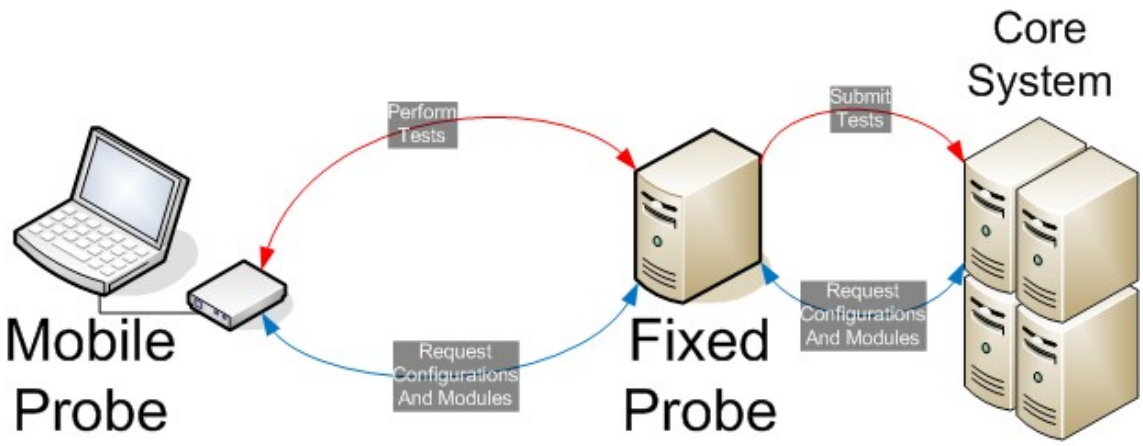

Figure 6: Probes Architecture

The tests referred before are supported by an architecture of dynamic modules. These dynamic modules are loaded on demand by the probes. In order to obtain these modules, when the probes request policies from the core system they also request these dynamic modules. The core system then sends to the probes these modules packaged and the corresponding configurations of the modules. When the probes receive the modules, they load them and the next time they perform tests they will try to use them.

Finally, the mobile probes have schedules for the tests. These schedules are configured through the administration interface in the core system, when the mobile probes request for policies these schedules will also be sent. The mobile probe will use this information to schedule the tests.

Each of these schedules contain information about the start date of the test, the end date, the week days when it is supposed to be run, the time of day to perform it and the information about the fixed probe which is to be used to perform the test.

To guarantee consistency of the results, the probes must authenticate themselves with the PCE before submitting results.

3.3 - Core System Architecture

The core system can be torn down into four parts, the PIR (Policy Information Repository), the WMI (Web Management Interface), the PCE (Probes Communication Interface) and the TRR (Tests Results Repository). This architecture is shown is Figure 7.

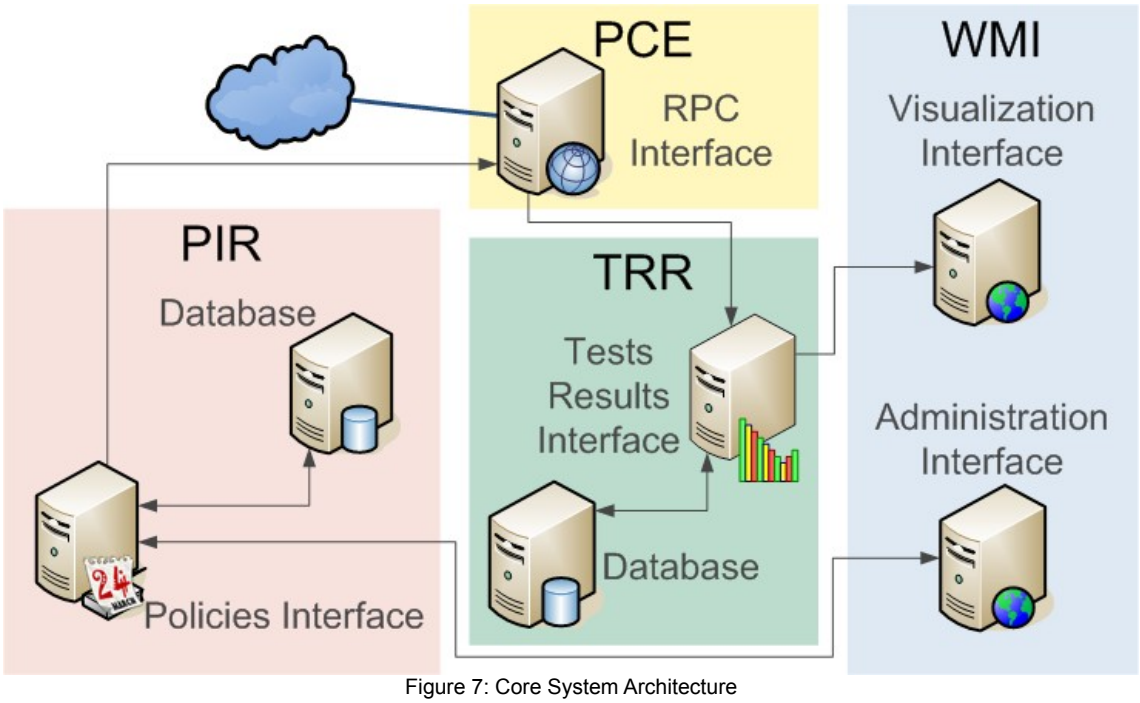

Figure 7: Core System Architecture

Each of this part is responsible for a set of activities and while performing this activities the parts communicate between them.

The PIR is responsible for providing policies and store them in a database. This database doesn't need to be on the same system.

The TRR is a interface to a database. The goal of this database is to store the tests results. This database, like the PIR database, can be on a different system.

The PCE is the endpoint used by the fixed probes to communicate with the core system, this endpoint is an RPC interface following some standard so the probes technology isn't limited to a single one.

The WMI is a part based on two interfaces used for administration and visualization tasks, the

administration tasks are related to configuring the probes while the visualization tasks are related to view the tests results. The administration interface interacts directly with the PIR requesting and submitting information and the visualization interface interacts directly with the TRR basically requesting information.

3.3.1 - Policy Information Repository Architecture

The PIR is responsible for delivering the configurations for the probes so they know how and when to perform the tests. The PIR is also responsible do deliver the dynamic modules to the probes so they can actually perform the tests. These configurations and modules are always requested by the fixed probes, they send the request to the PCE, which will forward it to the PIR. Then, the PIR will respond to the PCE which will forward back the response to the fixed probe. The mobile probes, to retrieve information and modules, contact one fixed probe, which will perform the same process described before but in the end, the fixed probe will forward back the response from the core system to the mobile probe.

This part, the PIR, is formed two elements, a database to store all the persistent data like policies, modules and any other needed information, and an endpoint to manage and query the database. All the requests are sent to the endpoint and never directly to the database. Only the PIR endpoint accesses the PIR database.

Regarding the data stored in the PIR, that data corresponds to policies and dynamic modules. The policies include core system and identifications, probes addresses and schedules. The policies are managed in two layers, probes and virtual probes. The virtual probes are sets of fixed or mobile probes that share the same list of modules and have a default configuration. To create a probe in the system it is necessary to choose a virtual probe to retrieve the default configurations and create an association between both, the probe and the virtual probe. When a request for the dynamic modules of specific probe arrives at the PIR, the PIR looks for the modules on the virtual probe associated with that probe. This allows an easy configuration of specific sets of probes having each set perform different types of tests, *e.g.* one set of probes testing VoIP and another testing Video Streaming.

3.3.2 - Test Results Repository Architecture

The TRR is consisted basically by a database to store the tests results and an interface to access it. This database must be as large as the quantity of tests results needed to be stored.

The tests stored in the database are organized by fixed and mobile probe pair. This means that when a pair of a mobile and fixed probe performs a test the results of that test are stored in the same place in the database.

Each test has information about the time the test was performed, which probes (fixed and mobile) were involved in the test, which dynamic modules were used and of course, the results of the test. These results are split in two parts, one corresponds to the results the fixed probe has submitted

and the other one to the results the mobile probe has submitted. As mentioned before, the mobile probes never communicate directly with the core system so, when a mobile probe wants to submit results, it sends them to the fixed probe which was being used to perform the test. This fixed probe then submits the results to core system.

3.3.3 - Probes Communication Endpoint Architecture

The PCE is simply an endpoint for the fixed probes to interact with the core system. As mentioned earlier the requests sent to the PCE are the requests for policies and modules and the requests for submitting results.

When the PCE receives a request for policies or modules from a fixed probe, it forwards the requests to the PIR and waits for an answer. After receiving the answer the PCE responds back to the probe by forwarding the response it received from the PIR.

When a probe submits results to the PCE, the PCE accepts the results and notifies the probe that the results have been accepted on the system. At a later time, the PCE will submit these results received to the TRR. This early acceptance of the results by the PCE allows the probes to be released from the communication with the core system even when the TRR is unavailable when the probes are submitting the results.

Again, to guarantee consistency of the results, the PCE must enforce the probes to authenticate before submitting results.

3.3.4 - Web Management Interface Architecture

The WMI is formed by two parts, the Configuration Interface and the Visualization Interface. The Configuration Interface only communicates with the PIR and the Visualization Interface only communicates with the TRR.

The communications between the both these interfaces and the core system are described in Figure 8.

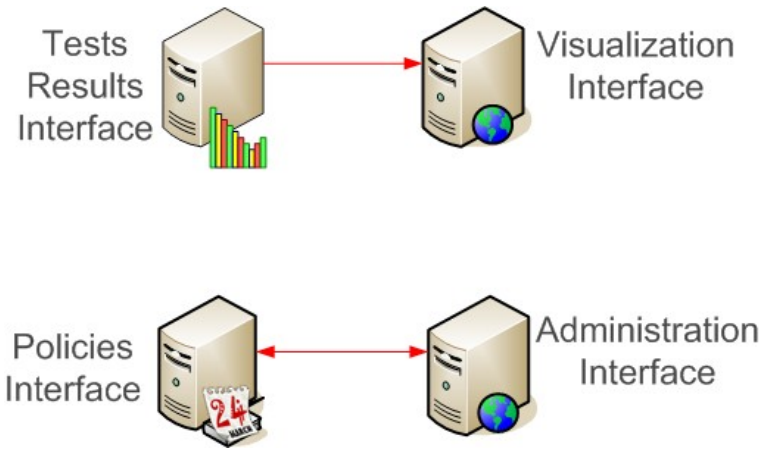

Figure 8: Core System Interfaces Communication

The Configuration Interface is used to configure all the monitoring system. From this interface it is possible to configure the probes and modules. Regarding the probes, in this interface it is possible to configure the probes by changing names, addresses and manage the schedules on the mobile probes, these schedules can be configured to run tests on specified week days, on a specified time of the day, the dates of start and finish of the schedule and the fixed probe which will be used to perform the test.

The Visualization Interface is used to view the results of the tests. These results can be filtered by some parameters like which probes were involved in the test, the dates of the tests and the modules used on the tests.

To guarantee unauthorized changes to the system and visualization of reports, the Administration and Visualization Interfaces must enforce the users to authenticate themselves before performing any of these actions.

3.4 - Dynamic Modules Architecture

As explained before, the modules are loaded by the probes to provide them (the probes) all the needed functionality to perform tests.

The modules are responsible for working in both types of probes (fixed and mobile), to start and stop the tests, to analyse the results and generate reports.

These modules can be dynamically loaded in the system, they are loaded through the Administration Interface and after adding them to a virtual probe, each probe that inherits configurations from that virtual probe will at some time download this new module and enable it without needing any extra action or reconfiguration. When the next test starts the probe will be able to use that new module.

Each module has configurations associated with it, this configurations can be changed in the Administration Interface and will be sent along with the module when the probes download it. These configurations can be parameters like TCP ports, UDP ports, test duration, number of packets per

second or anything else.

These modules can be developed by third party entities as they only need to respect an interface that the modules know. This makes this monitoring system very scalable as it can be adapted to monitor any existing or new protocol.

The interaction between the different entities of system when working with dynamic modules is shown in Figure 9.

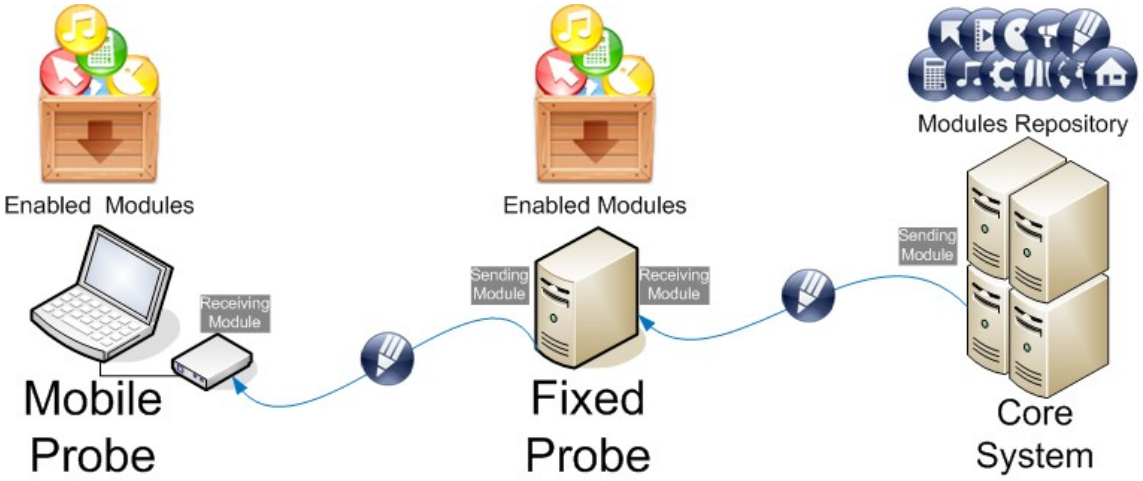

Figure 9: Dynamic Modules

When starting a test, the mobile probe coordinates with the fixed probe to decide the modules to use during the test. After deciding which modules to use, each probe enable a module at a time. This makes both probes to have always the same module enabled at any time during the test. When a module is enabled it is responsible for performing the test between the probes. At the end of the test, the module is responsible for performing data refinement and data aggregation to generate the test results. These results will afterwards be submitted to the core system.

4 - Implementation

The implementation of this system has been based on Java technologies, but it is possible to implement the probes with another technology.

All the parts of the architecture have been implemented except the security one. This one has not been implemented due to time constraints.

4.1 - Technologies

The main technology used to implement this system is Java [5]. For the probes, Java SE [6] was used, for the core system, Java EE [7] was used instead.

This technology has one great advantage, it is portable across all main platforms. No special care needs to be taken in order to make sure this monitoring system is capable of running in different platforms.

The probes are implemented in Java SE. Java provides all the mechanisms needed to implement all the probes features. Even though the probes are implemented with Java they can be implemented with another technology, as long as that technology supports the standards used for communications between the components of the monitoring system.

Java EE is an application server based on Java, the one used was Glassfish [8]. With Glassfish, it is possible to deploy all the components of the core system to this application server. Also, Glassfish supports a cluster configuration in order to configure an architecture that is fault tolerant and also has a better performance.

The DBMS used during the implementation was Java DB [9]. This DBMS was included in the IDE used for development.

The development environment used was basically made up by an IDE. The IDE used was the Netbeans [10]. This IDE provides many features designed to aid in Java EE development. It also had an application server integrated, it was the Sun Java System Application Server 9.1, a version of the Glassfish application server distributed by Sun Microsystems.

Also to develop the system, Java had to be installed and the version installed was Java SE Development Kit 6 Update 2.

Finally, some other technologies that were used are listed in Table 5.

Technology	Use
Ant	Used to build the components of the system. The files were generated by the Netbeans IDE, only some minor additions to the files were needed.
XML	XML files are used to store the probes configurations on the filesystem.
SOAP	SOAP is used in the communications between the probes and the core system.
SQL	SQL is used to store the policies, modules and the tests results.
EJB	EJB is used by the components of the core system. It provides a simpler development model for communications.

Table 5: Other Used Technologies

4.2 - Communication Model

The communications between the components of the core system are very diverse, they correspond to simple remote procedure calls and always follow what has been described in the architecture of the system.

Basically, the PCE communicates with the PIR and the TRR; the WMI also with both PIR and TRR and so on. This communications are transparently implemented with EJB.

To implement an EJB interface it is needed to implement a Java class and the interfaces related to it. The class must have all the methods needed to perform the actions and must have some annotations, *i.e.* a class must be declared like the following:

```
@Stateless
public class SomeClass implements package.SomeClassRemote, package.Some-
ClassLocal
```

The local interface related to the EJB class must be declared like the following:

```
@Local
public interface SomeClassLocal
```

The Remote interface is almost equal to the Local one, the only difference in the source code is that the @Local is replaced with @Remote and the name of the interface is also different, normally the *Local* part of the name is replaced by a *Remote* part. Regarding functionality, the Local interface can only be used inside the same application, if the EJB is being called from a different application, the Remote interface has to be used. In this system, the communication between the different components are performed using the Remote interfaces, but when calling an EJB inside the same component, the communication is performed using the Local interface. An example of when to use the Local interface, could be when the PIR needs to access information provided by itself, in this case, the PIR would use the Local interface because the EJB is inside the same application.

On the client side, to use an EJB normally it is only required to declare a variable like:

```
@EJB
SomeClassRemote someClass;
```

After having the variable declared this way it is possible to call the methods remotely with something like:

```
result = someClass.getAverage(number_1, number_2, number_3);
```

Regarding the communications between the probes and the core system, these are going to be described with the help of communication diagrams.

In Figure 10 it is possible to understand exactly how the probes interact with the core system. First the fixed probe retrieves its configurations by asking the PCE for them and waiting for an answer, the PCE will then forward the request to the PIR and send the answer from the PIR back to the probe, which will load the configurations after receiving them. After the fixed probe loads the configurations, a mobile probe also start the process of asking for configurations, instead of sending the request to the PCE, the mobile probe sends the request to the fixed probe and waits for an answer, the fixed probe will request these configurations in the same it would request configurations for its own. In the end, when the fixed probe sends the configurations to the mobile probe, the mobile probe loads them.

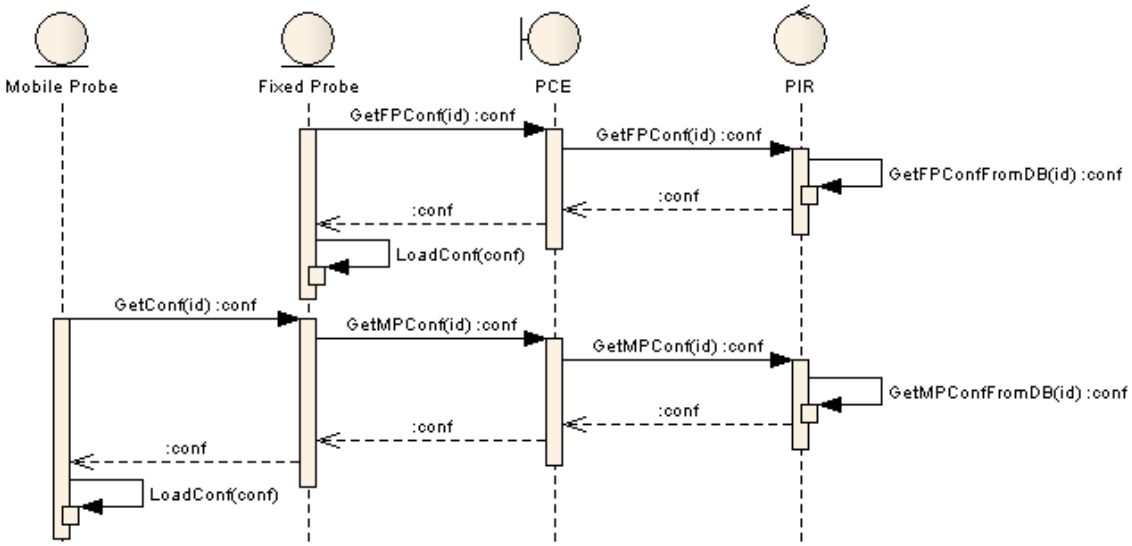

Figure 10: Probes Configuration Retrieval Sequence Model

The actions performed by the probes to retrieve the modules from the core system are described in Figure 11. The way the probes request modules from the system is very similar to the way they request configurations. One difference is that the probes first request a list of modules assigned to them, with this list, they only need to request the modules that are not already in local cache. The probes have to perform a request for each module they want to download from the core system. Both types of request, for the modules and for the modules list, are performed in the same way, the request goes to the PIR and the probes wait for an answer after sending the request.

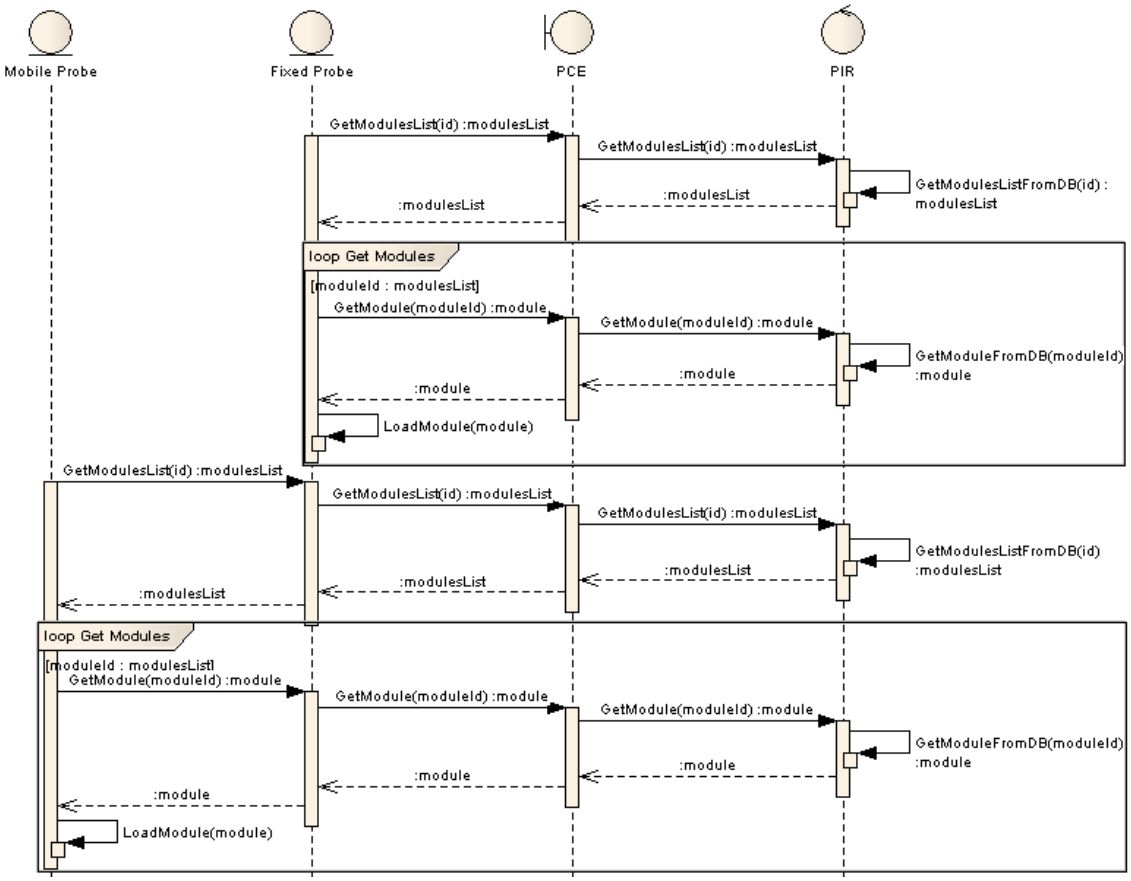

Figure 11: Probes Modules Retrieval Sequence Model

Finally, in Figure 12 it is possible to observe the process of results submittal by the probes. First, these process is asynchronous for the probes, the mobile probes submit the results to the fixed probe and this one accepts them and only latter will send the results to the core system. The same happens when the fixed probe submits the results to the PCE in the core system. The PCE also promptly accepts the results and only afterwards will submit these results to the TRR.

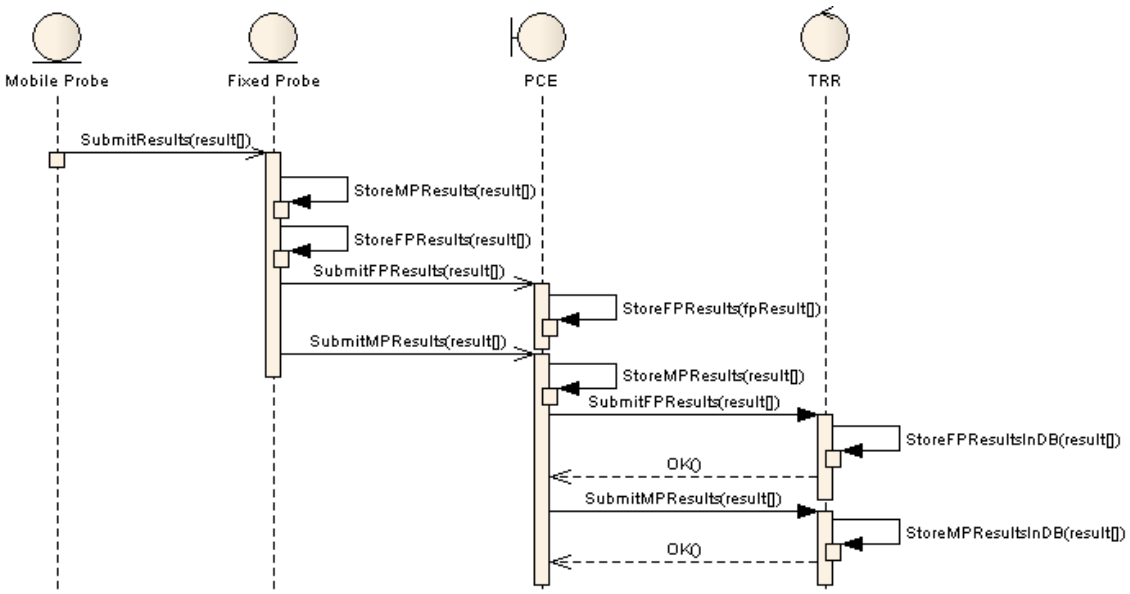

Figure 12: Probes Results Submittal Sequence Model

4.3 - Data Model

Only the PIR and the TRR are represented in the data model since these components are the only ones that contain changeable data in some sort of storage mechanism like an SQL database.

It is important to mention that the database model is automatically created from annotations on the Java classes, *e.g.* the Module class has the following portion of code:

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "ID", nullable=false)
    private Long id;
    @Column(name = "NAME", unique=false, nullable=false)
    private String name;
    @Column(name = "VERSION", unique=false, nullable=false)
    private String version;
    @Column(name = "DESCRIPTION", unique=false, length=1024, nullable=false)
    private String description;
    @Column(name = "HELP", unique=false, length=2048, nullable=false)
    private String help;
    @Column(name = "JAR", nullable=false, length=5120000) // 5 MB
    private byte[] jar;
    @OneToOne(targetEntity=Category.class)
    @JoinColumn(name="CATEGORY_ID", nullable=false,
referencedColumnName="ID")
    private CategoryIface category;
    @OneToMany(cascade=CascadeType.ALL, targetEntity=Configuration.class)
    private Collection<ConfigurationIface> configurations;
```

From that portion of code, a table is generated with a primary key (identified by the @Id) being of type long represented in Java by the id variable. All the other elements are also placed in the table, like name, version, description and help being all strings and jar being a blob. At the end, two relationships are represented, an one to one relationship to Category (identified by the @OneToOne) and an one to many relationship to Configuration (identified by the @OneToMany).

In Figure 13 it is shown the PIR data model. All the components, probes, modules, schedules, modules categories and modules configurations are represented there. Also represented is the rela-

tionship tables, constraints, primary keys and foreign keys.

The main elements from the PIR data model are configuration, category, module, probe and schedule. In the connections between these elements there are another connecting elements, these ones are responsible for helping keeping the data model structured and simple to work with.

The way the elements are connected is like the following. A probe has schedules and modules. Each probe can have any number of schedules and modules. Each schedule belongs to a single probe but each module can belong to any number of probes. Then every module has one category associated and any number of configurations.

This data model was designed in a way to fulfil the PIR architecture needs.

Figure 13: PIR data model

In Figure 14 it is described the TRR data model. All the components, tests and results are represented there. Also represented are the relationship tables, constraints, primary keys and foreign keys.

One test is basically a relationship between two probes. In this relationship it is possible to view all the results of the tests performed between these two probes.

This data model was designed in a way to fulfil the TRR architecture needs.

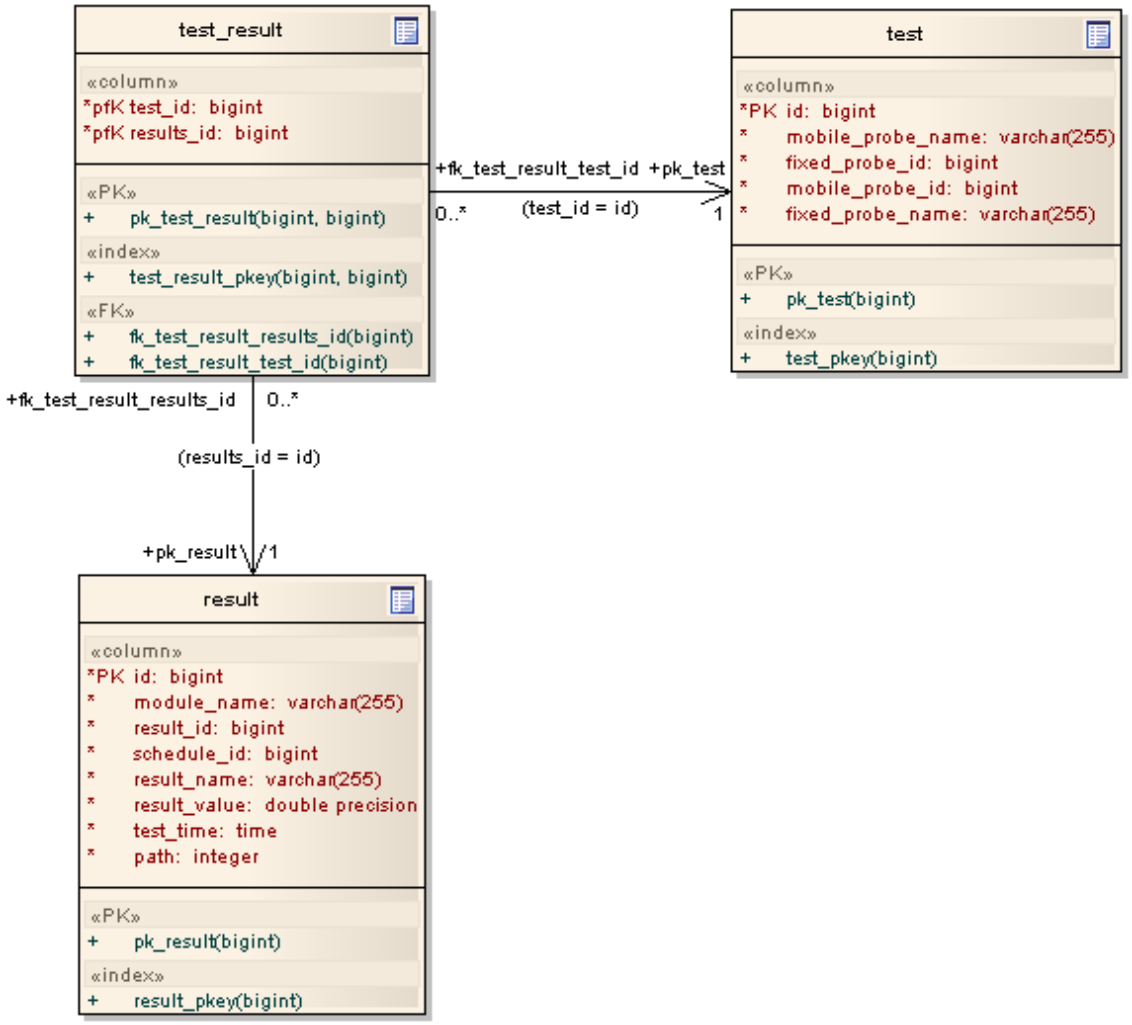

Figure 14: TRR data model

4.4 - Core System Interfaces

Both the Administration and Visualization interfaces are web based. There hasn't been any special care about the interfaces design as it is out of the scope of this project.

4.4.1 - Administration Interface

In the administration interface it is possible to configure all the probes configurations.

In Figure 15 it is possible to view a screenshot of the interface when adding a new virtual probe to the system. The interface asks the user some needed parameters, like name, location and server address and port.

Figure 15: Core System Administration Interface: Add Virtual Probe

In Figure 16 it is possible to view a screenshot of the interface when showing a list of virtual probes on the system. From this view it is possible to manage or delete a virtual probe.

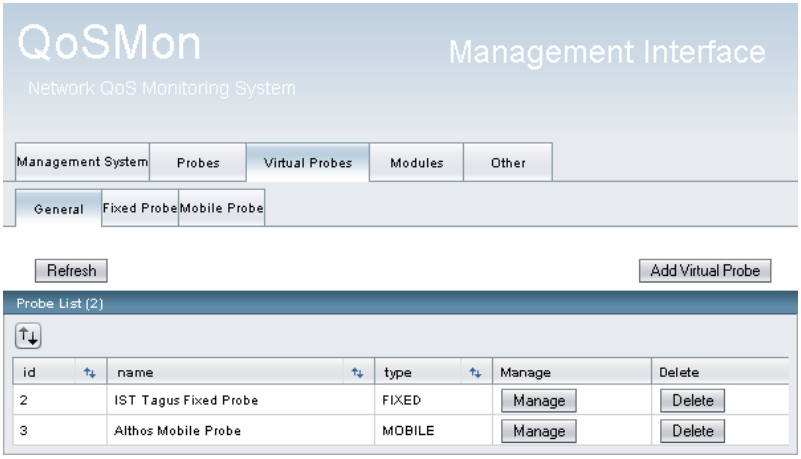

Figure 16: Core System Administration Interface: Virtual Probes List

The virtual mobile probe management screen is almost identical to the virtual fixed probe management screen so it won't be explained here, but a screenshot can be viewed in Annex B, Figure 58.

In Figure 17 it is possible to view a screenshot of the interface when managing a virtual fixed probe. From this view it is possible to change configurations of the virtual fixed probe and enable or disable modules for it.

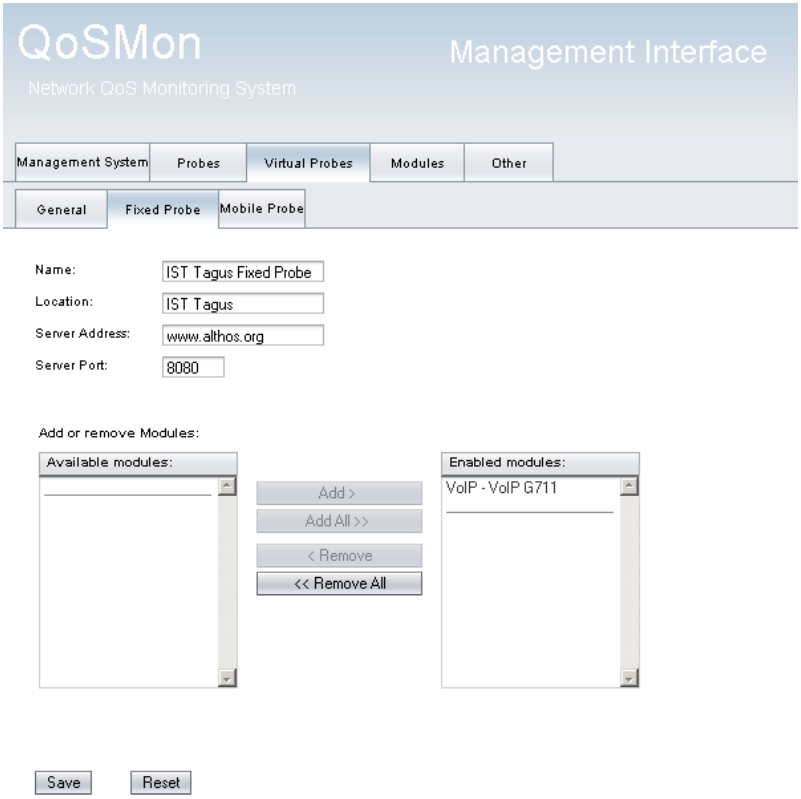

Figure 17: Core System Administration Interface: Manage Virtual Fixed Probe

The probes list is almost identical to the virtual probes list so it won't be explained here, but a screenshot can be viewed in Annex B, Figure 59.

In Figure 18 it is possible to view a screenshot of the interface when adding a probe. To add a probe first a virtual probe is selected form the drop down list of virtual probe. Afterwards some default options appear already filled. When the user fills all the options he can add the probe to the system.

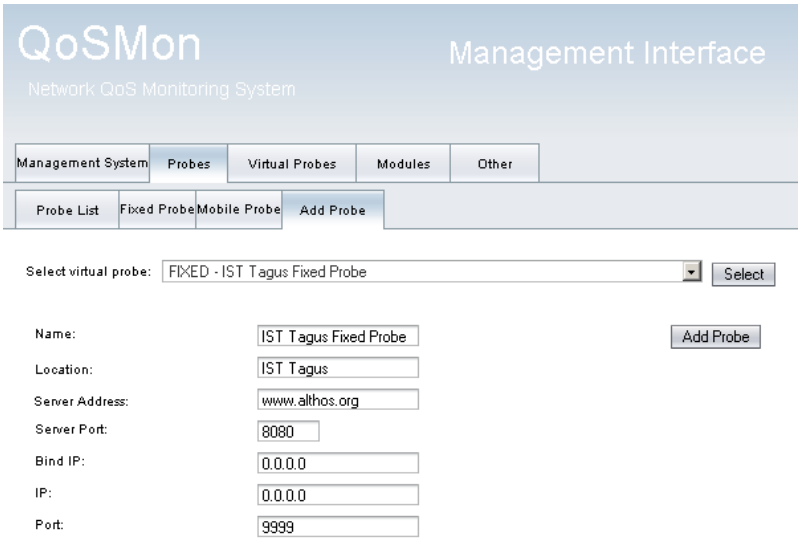

Figure 18: Core System Administration Interface: Add Probe

The screen to manage a fixed probe is very simple, containing only inputs to configurations

like the ones on Figure 18. This way, it won't be described here, but a screenshot can be viewed in Annex B, Figure 60.

In Figure 19 it is possible to view a screenshot of the interface when managing a mobile probe. In this screen it is shown the interface to add a schedule to the mobile probe. Behind the list of fixed probes there are a couple of configuration parameters. When adding a schedule to a probe the user specifies the start and end date, the week days and time when to run the schedule and the fixed probe which the mobile probe will contact to perform the test. On the bottom there is a table with all the existing schedules for the probe. In this table it is possible to enable or disable the schedule or remove it from the probe.

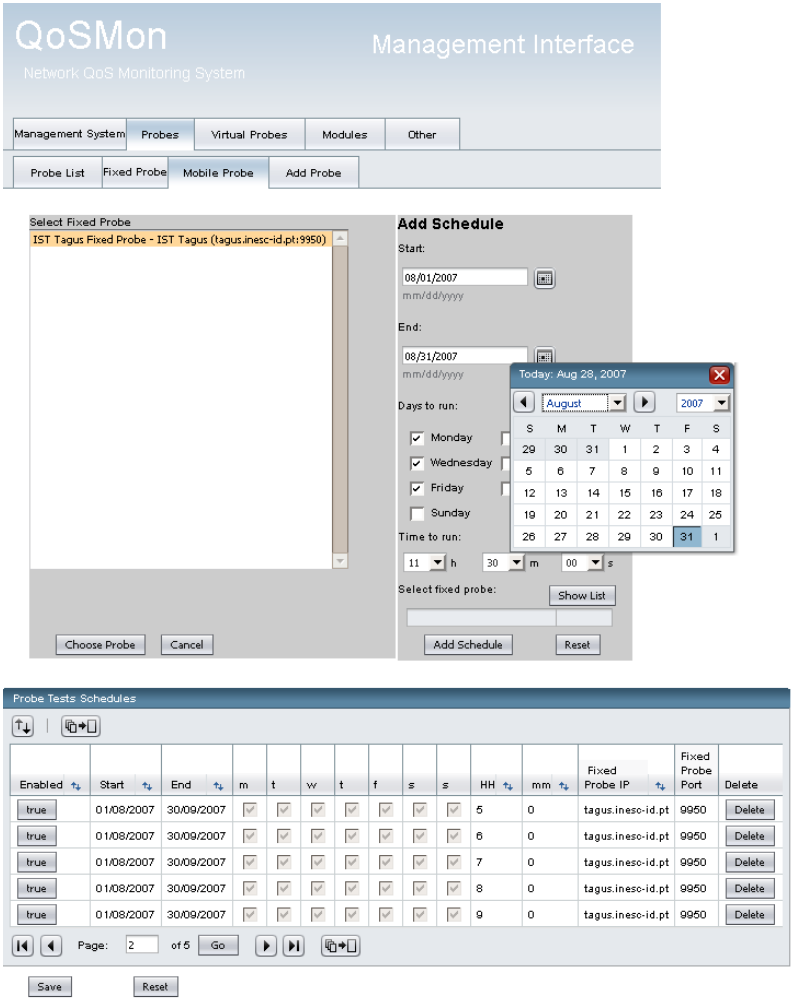

Figure 19: Core System Administration Interface: Mobile Probe Schedules

In Figure 20 it is possible to view a screenshot of the interface when managing modules on the system. To add a module the user just has to upload it to system using the top layer of this screen. To configure a module the user selects it and the interface shows a screen with module informations, a remove module button and a table with the module configurations where a user can edit them.

Figure 20: Core System Administration Interface: Modules

4.4.2 - Visualization Interface

In the visualization interface it is possible to view the tests results.

Figure 21 shows the search screen of the visualization interface. From this screen the user can filter the results by probes and dates.

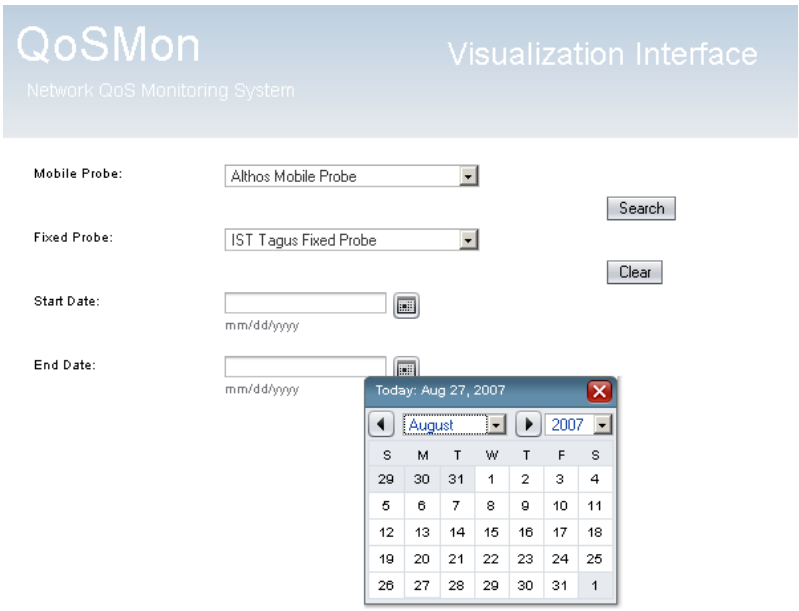

Figure 21: Core System Visualization Interface: Search screen

After selection the probes and dates the user is presented with a screen asking from which

module the user is interested to view the results. This screen is show in Figure 22.

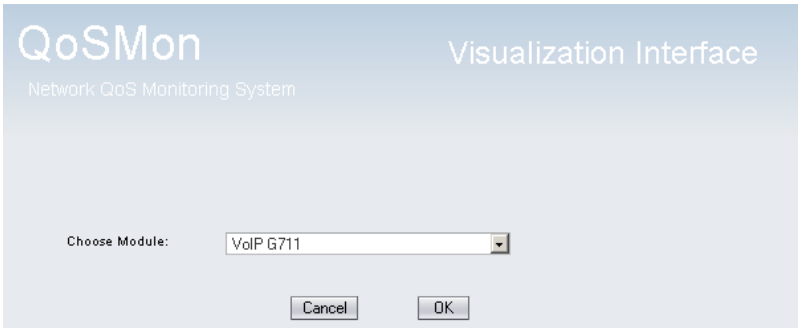

Figure 22: Core System Visualization Interface: Module selection

After the user chooses the module the visualization interface presents a table like Table 6 where many parameters are shown. From this screen, which isn't represented here as a screenshot, the user can follow a link to download this same table as a file in a comma-separated values format (csv).

Test ID	Test Time	FP ID	FP Name	MP ID	MP Name	Path	Module	Schedule ID	Result ID	Result Name	Result Value
2	25/08/2007 23:00:00	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	FP->MP	VoIP G711	38	1	Minimum Delay (ms)	16,16
2	25/08/2007 23:00:00	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	FP->MP	VoIP G711	38	0	Average Delay (ms)	18,09
2	25/08/2007 23:00:00	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	MP->FP	VoIP G711	38	1	Minimum Delay (ms)	16,18
2	25/08/2007 23:00:00	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	MP->FP	VoIP G711	38	0	Average Delay (ms)	18,27

Table 6: Core System Visualization Interface: Results table

4.5 - Probes Management Interface

The management interface for the probes is a command line interface. It is an RPC client program that communicates with the probe, which has an RPC server. This design allows the probes to be started automatically at the system boot, and still allow someone to interact with the probe. Of course, different probes can be developed and integrated with this system, even with different technologies, since all the communications performed by the probes are SOAP based, a standard supported by many technologies.

With this interface it is possible to order the probe to fetch configurations and modules. When the user wants to order a mobile probe to fetch configurations from a fixed probe the parameters passed to the program are: -ip <FIXED_PROBE_IP> -port <FIXED_PROBE_PORT> -id <ID> fetch-config. The FIXED_PROBE_IP is the fixed probe IP, the FIXED_PROBE_PORT is the fixed probe port and ID is the mobile probe id. To order the same action in a fixed probe the syntax is basically the same, the difference is that the IP and port doesn't correspond to a fixed probe, instead they correspond to the core system. To fetch modules from a mobile probe the user passes the following parameters: -ip <FIXED_PROBE_IP> -port <FIXED_PROBE_PORT> fetch-modules. Again, when per-

forming the same action with a fixed probe, the IP and port correspond to the core system IP and port.

The user can also stop, enable or disable a probe by passing the corresponding parameters: stop, enable or disable. If a probe is disabled it won't perform tests, if it is a mobile probe it won't start them, if it is a fixed probe it won't accept starting them.

With the mobile probes it is possible to start a test on demand by passing the following parameters: -ip <FIXED_PROBE_IP> -port <FIXED_PROBE_PORT> run-test. A single test will be started with that fixed probe specified in the parameters.

From this interface it is also possible to view the probe configurations by passing the parameters like list-configurations, list-modules or list-schedules (only on a mobile probe). These commands will show to the user the configurations the probe has.

An example of viewing the configurations of a fixed probe (passing the list-configurations parameter) follows:

```
java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient list-configurations
----------------------------------------
ID: 4
Name: IST Tagus Fixed Probe
Location: IST Tagus
Bind IP: 172.20.41.200
IP: tagus.inesc-id.pt
Port: 9950
Enabled: true
Server IP: www.althos.org
Server Port: 8080
----------------------------------------
```

4.6 - Probes Configurations Storage

The probes configurations are stored on files in the .qosmon directory located in the home directory of the user. Inside .qosmon the probes store the configurations and the modules.

For the configurations the mobile and fixed probes use the file mp-conf.xml or fp-conf.xml respectively.

On the mp-conf.xml file the mobile probes store the following configuration details: id, name, location, fixed probe ip, fixed probe port and schedules.

On the fp-conf.xml file the fixed probes store the following configurations details: id, name, location, bind ip, bind port, server address and server port.

Also, the mobile and fixed probes store in the directories mp-modules and fp-modules respectively the modules downloaded from the server. Along with the modules, it is saved a file with the modules configurations, *e.g.* if a probe downloads a module called VoiceModule this module will be saved in the modules folder as VoiceModule.jar and its configurations inside the file VoiceModule.xml in the same folder.

4.7 - Dynamic Modules

To be used by the probes, the dynamic modules must implement a Java interface, by implementing this interface the probes know how to deal with it. This interface is called ModuleIface.

To submit the modules to the core system, they must be packaged in a jar file. This jar file must have packaged all the Java classes of the module and also an XML file called information.xml. This XML file must contain information about the module, specifically the module name, version, category, description, help and configuration parameters. Each parameter must be represented by its name and value. The reason for this XML file to exist is to allow the PIR to receive the module and retrieve its informations and configurations without having to load it.

When a probe needs to load a module, it scans the jar file for Java classes files (files that end with .class). While scanning each file, the probe checks if the file implements the ModuleIface interface. If it does, the probe instantiates an object from it. This object will represent the module.

4.8 - VoIP Dynamic Module

4.8.1 - Architecture

In order to test and demonstrate this monitoring system a dynamic module has been developed. The module tests a VoIP service. Specifically, it tests a VoIP call with a G.711 codec. This codec has a bandwidth of approximately 87.2 Kbps. The payload of the packets is 160 bytes and has a rate of 50 packets per second, *i.e.* a packet is sent every 20 milliseconds.

The design of this module is divided in two parts. One part to calculate the delay between the two probes and the other to calculate values related to jitter and packet loss.

The first part, the delay test, basically consists on calculating the round-trip time of a packet and divide that by two to obtain the delay. Of course, this does not give the correct delay because in most cases the delay is not equal on both directions. But as stated before, the goal of this module is to test and demonstrate the monitoring system. The delay could not be calculated just by obtaining a timestamp and send it inside the packet so other side calculates the difference to obtain the delay, in order to use that method, it would be necessary to configure both systems with a very accurate time source like Global Positioning System (GPS). Network Time Protocol (NTP) does not assure enough accuracy.

The second part of the test is intended to calculate the average jitter, maximum jitter, minimum jitter and the packet loss rate. This is done by having both modules constantly sending packets every 20 milliseconds (20 milliseconds is the interval between packets specified by the G.711 codec). On the receiving side, jitter is calculated and stored and the number of packets received is counted. In the end all the values are used to generate the needed results.

In Figure 23 it is possible to understand the exchange of packets between the two probes dur-

ing the delay test. For this test each probe sends a packet and waits for its return. When a probe receives a packet and if it was the same packet that was sent before, the probe uses it to calculate the time that passed between the moment the packet was sent and the time the packet was received, this result is the round-trip time. Every time a probe receives a packet that was not sent by it, the probes forwards that packet to same place where it came from. In the end, this behaviour results in a duplication of packets exchanged compared to normal G.711 codec.

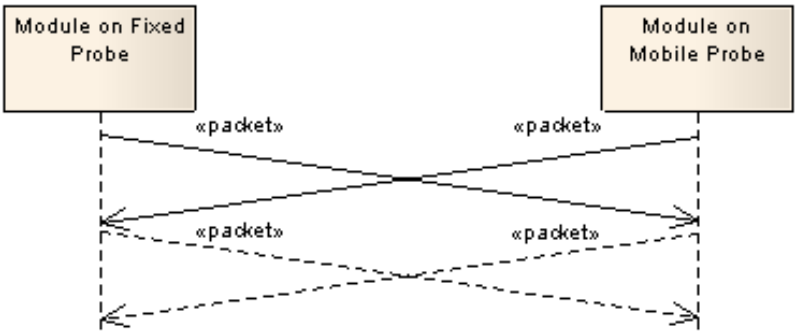

Figure 23: VoIP Dynamic Module Delay Test Packet Exchange

Figure 24 shows the components that perform this test. The send timer sends a packet every 40 milliseconds and stores informations about this action. The receive thread is always waiting for the packet that was sent by the send timer. When it receives a packet it uses the information stored by the send timer to calculate the round-trip time and afterwards the delay.

The reply thread basically receives the packets generated by the other probe and sends them back to where they came from.

Even though the send timer only sends packets every 40 milliseconds, the reply thread also sends a packet every 40 milliseconds, this means that in the end, the bitrate and the packets per second count will be the same as the G.711 specification.

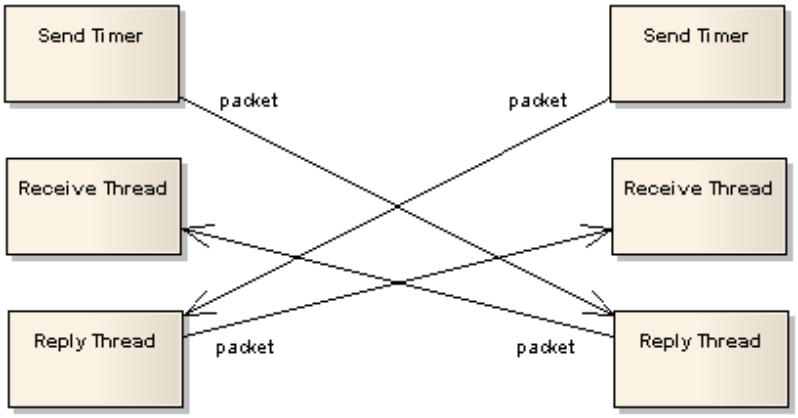

Figure 24: VoIP Dynamic Module Delay Test Components

Regarding the jitter and packet loss test, Figure 25 shows the packet exchange between the two probes during this test. Each probe sends a packet every 20 milliseconds. The difference between the arrivals of the packets is used to calculate the jitter and the number of packets received

is used to calculate the packet loss rate.

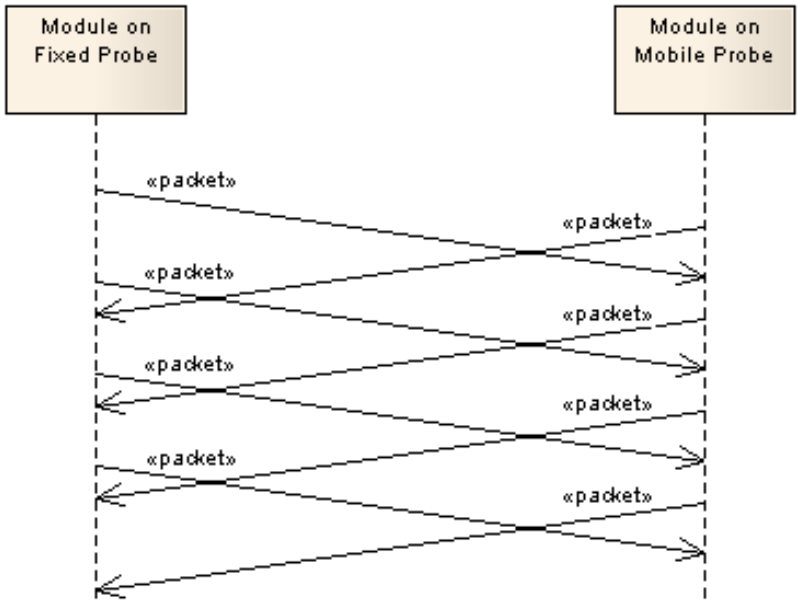

Figure 25: VoIP Dynamic Module Jitter and Packet Loss Test Packet Exchange

In Figure 26 it is shown the components of this test, basically there is on each probe, a timer to send packets periodically and a thread to receive them.

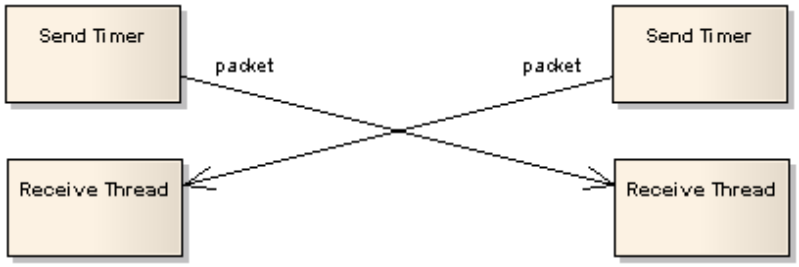

Figure 26: VoIP Dynamic Module Jitter and Packet Loss Test Components

This module has some configuration parameters and defaults for each one configuration. The configurations available are described on Table 7.

Name	Default Value	Description
Time to Live (ms)	5000	Time to live for the packets sent during the test.
Jitter + Lost Test UDP Port	15556	UDP Port to use while executing the jitter + lost test.
Delay Test UDP Port	15555	UDP Port to use while executing the delay test.
Duration Delay (seconds)	60	Duration of the part of the test (in seconds) that corresponds to packet delay measuring.
Duration Jitter + Lost (seconds)	540	Duration of the part of the test (in seconds) that corresponds to jitter and packet loss measuring.

Table 7: VoIP Dynamic Module Configuration Parameters

4.8.2 - Results

The results this module calculates and submits to the core system are: Minimum Delay, Maximum Delay, Average Delay, Minimum Jitter, Maximum Jitter, Average Jitter, Lost Packets, Received Packets and Total Packets.

With any combination of two results from Lost Packets, Received Packets and Total Packets, the packet loss can be calculated.

The process to calculate the delay is like the following: the probe stores a timestamp, sends a packet and waits for that packet to return, when the packet returns, the probe immediately stores another timestamp. The difference between the timestamps allows the probe to calculate the RTT, and dividing that RTT by two, the probe gets the delay. In order to have the packet return to the sender probe, the probe that receives it, only changes the headers of the packets and resends it. To calculate the average delay, when the probe calculates the delay, it adds the delay value to a variable. At the end, the probe divides that variable by the number of packets received to obtain the average. The maximum and minimum values of delay refer to the exact maximum and minimum delay calculated during the entire test.

The process to calculate the jitter is like this: the probe receives a packet from the other probe and stores a timestamp (T1), with this packet the probe does no calculations, after that the probe receives a second packet from the other probe and again, stores a timestamp (T2), also the probe calculates the interval between T1 and T2, then the probe receives the third packet from the other probe and stores a timestamp (T3). Beginning with the third packet, the probe can start calculating the jitter, this is accomplished by calculating again the interval between the last two packets using T2 and T3. With the first and second intervals the probe calculates the difference between them, this is the jitter. If the jitter is negative the probe converts it to a positive value, *i.e.* if the jitter value is -5, the probe uses 5. To calculate the average jitter, when the probe calculates the jitter, it adds the jitter value to a variable. At the end, the probe divides that variable by the number of packets received to obtain the average. The maximum and minimum values of jitter refer to the exact maximum and minimum jitter calculated during the entire test.

To perform calculations with the packets, the probes send inside the payload of the packets the sequence number and the total packets being sent. With this, the probe can detect if a packet has been lost and how many packets are supposed to be sent. To calculate the received packets, the probes basically use a counter and increment it every time a packet is received.

5 - Tests

The best way to test this system was putting it to work in order to perform QoS monitoring and afterwards analyse the results given by the system.

The tests were run from August 25 until September 10 every hour using the VoIP module developed for this system.

The tests were performed with the core system behind an ADSL link with a downlink of 4 Mbps and an uplink of 1 Mbps. The fixed probe was behind a 20 Mbps Etherlan/Metrolan access connected to an 100 Mbps internet access. The mobile probe was behind the same ADSL link as the core system. The tests were performed through the Internet.

Also, just to make sure the system is working correctly, some tests were performed with both probes and the core system in the same LAN.

5.1 - Results

Table 8 shows the results of the average delay and packets sent, received and lost of the tests performed with all the components on the same LAN.

Test	Path	Average Delay (ms)	Lost Packets	Received Packets	Total Packets
1	MP-FP	0,158839	0	13500	13500
1	FP-MP	0,314685	0	13500	13500
2	MP-FP	0,160320	0	13500	13500
2	FP-MP	0,406405	0	13500	13500
3	MP-FP	0,161113	0	13500	13500
3	FP-MP	0,504894	0	13500	13500
4	MP-FP	0,159274	0	13500	13500
4	FP-MP	0,314713	0	13500	13500
5	MP-FP	0,200466	0	13500	13500
5	FP-MP	0,381173	0	13500	13500

Table 8: Results of the tests performed with all the components on the same LAN

It is possible to see that there were no packets lost, showing that the system is functioning properly. Also, the average delay is very low as expected from a LAN.

The graphics that will be described illustrate the results obtained with this monitoring system performing the tests during the period from August 25 and September 10.

In Figure 27 it is possible to view the variation of the average delay of each test. During the time of the tests, the average delay has been mostly between approximately 10 and 25 milliseconds. Also, the average delay of the fixed probe and the average delay of the mobile probe are very close perhaps because, as stated before, the delay is calculated using the RTT and the result is only an ap-

proximation and should be almost identical on both systems.

In this same illustration it is also possible to observe an increase of the average delay during a large period from approximately 12:00 am of September 7 to the end of September 8.

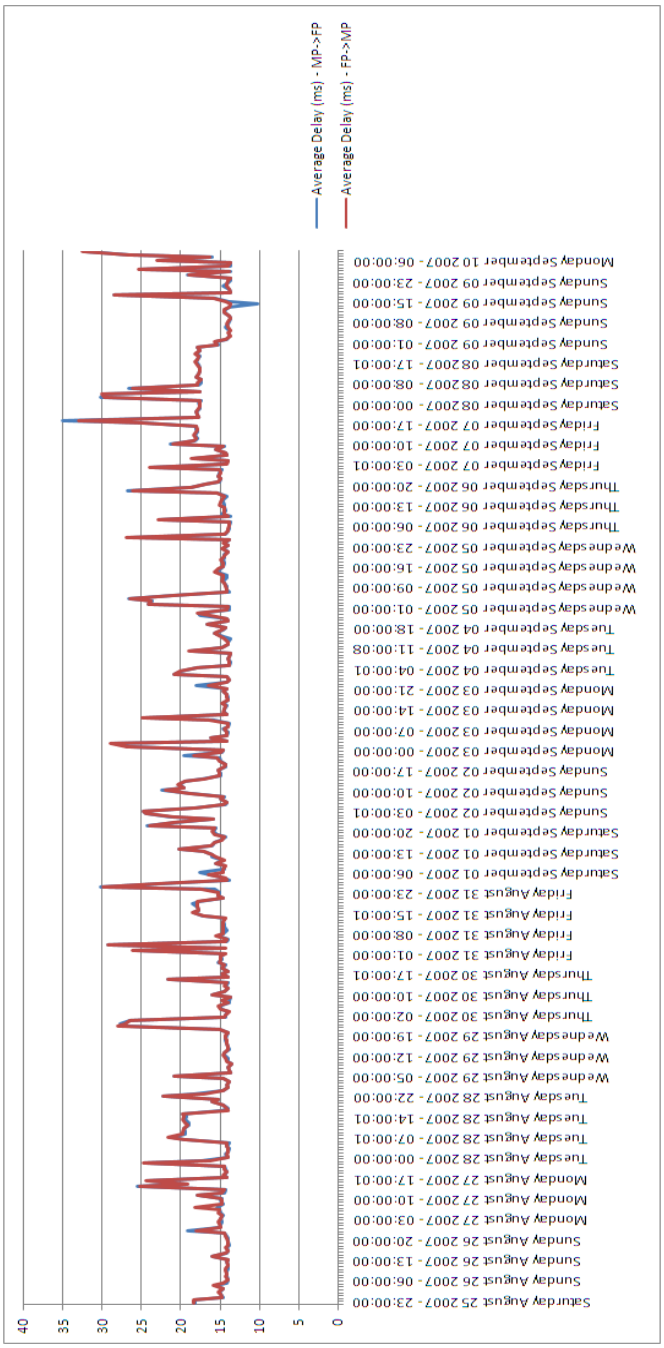

Figure 27: Average Delay (ms)

In Figure 28 it is possible to view the variation of the minimum delay of each test. These values are more constant than the average delay. The average is approximately 13 milliseconds.

It is possible to observe an increase of the minimum delay (like in the average delay results) during a large period from approximately 12:00 am of September 7 to the end of September 8.

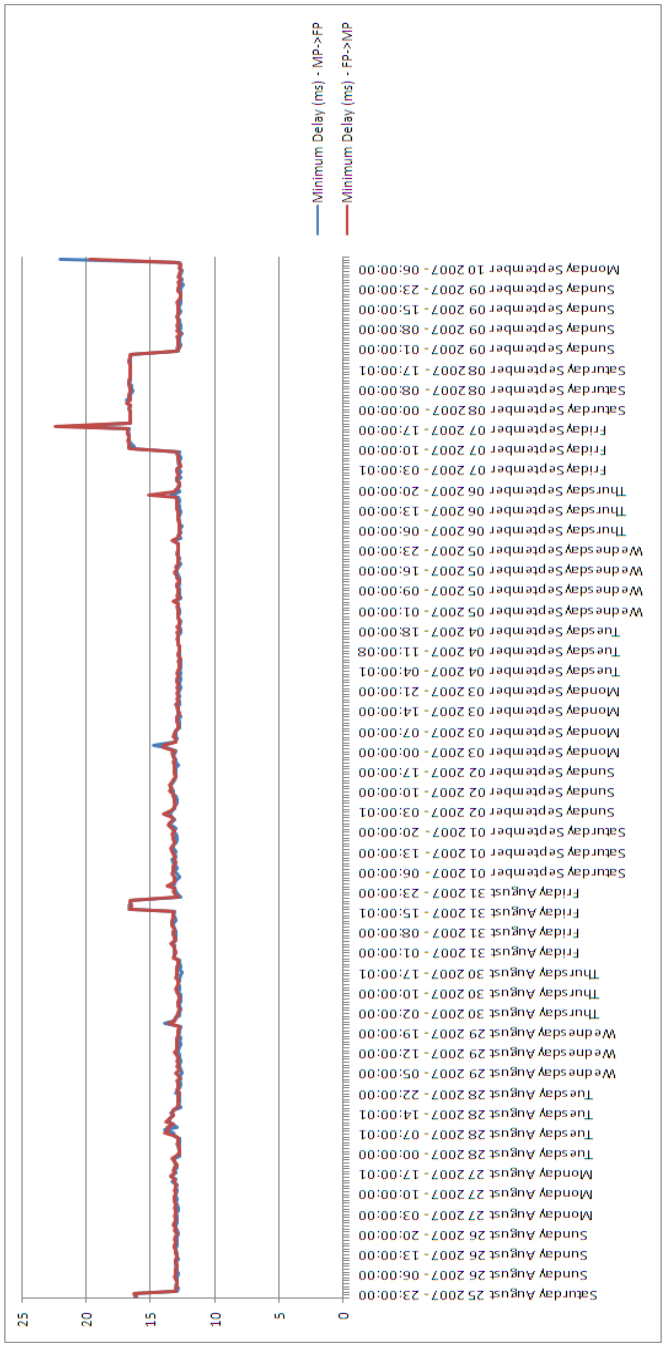

Figure 28: Minimum Delay (ms)

In Figure 29 it is possible to view the variation of the maximum delay of each test. During the tests, the maximum delay has large variations that range from 20 to 450 milliseconds.

In this illustration it is not possible to observe the same increase of the values during the period from approximately 12:00 am of September 7 to the end of September 8. The reason behind this is that factors that affected the delay during that period were not significant enough to affect the maximum delay.

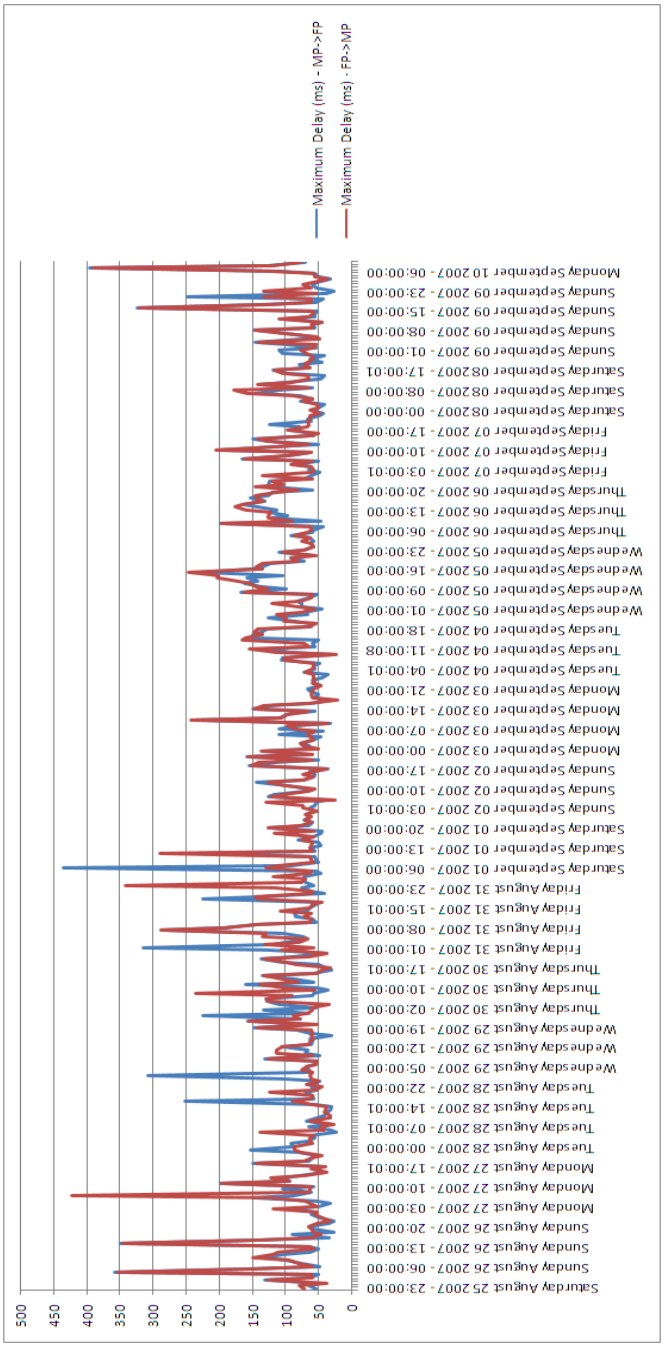

Figure 29: Maximum Delay (ms)

In Figure 30 it is possible to see that in the MP->FP path almost no packets have been lost, on the reverse path, the percentage besides being also very low it increases sometimes.

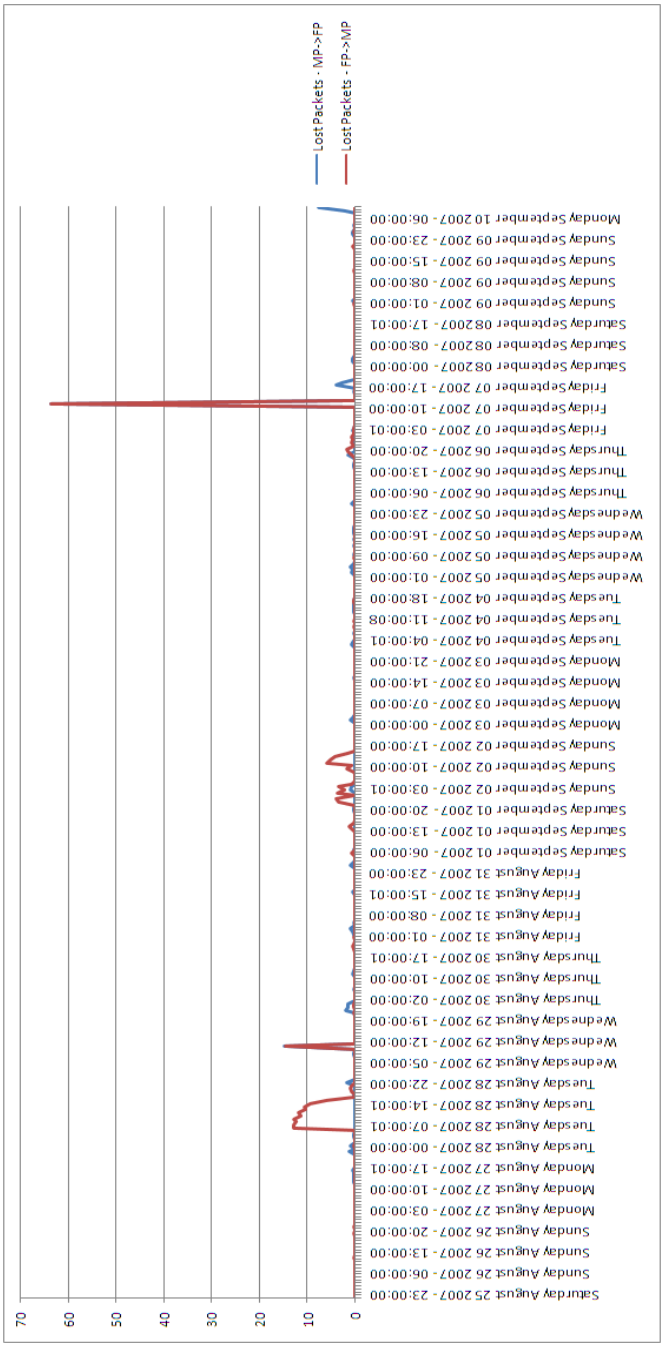

Figure 30: Lost Packets (%)

In Figure 31 it is possible to see the average jitter during the tests. The average jitter in the path FP->MP is slightly lower than in the reverse path. Both paths have an high variation of the average jitter.

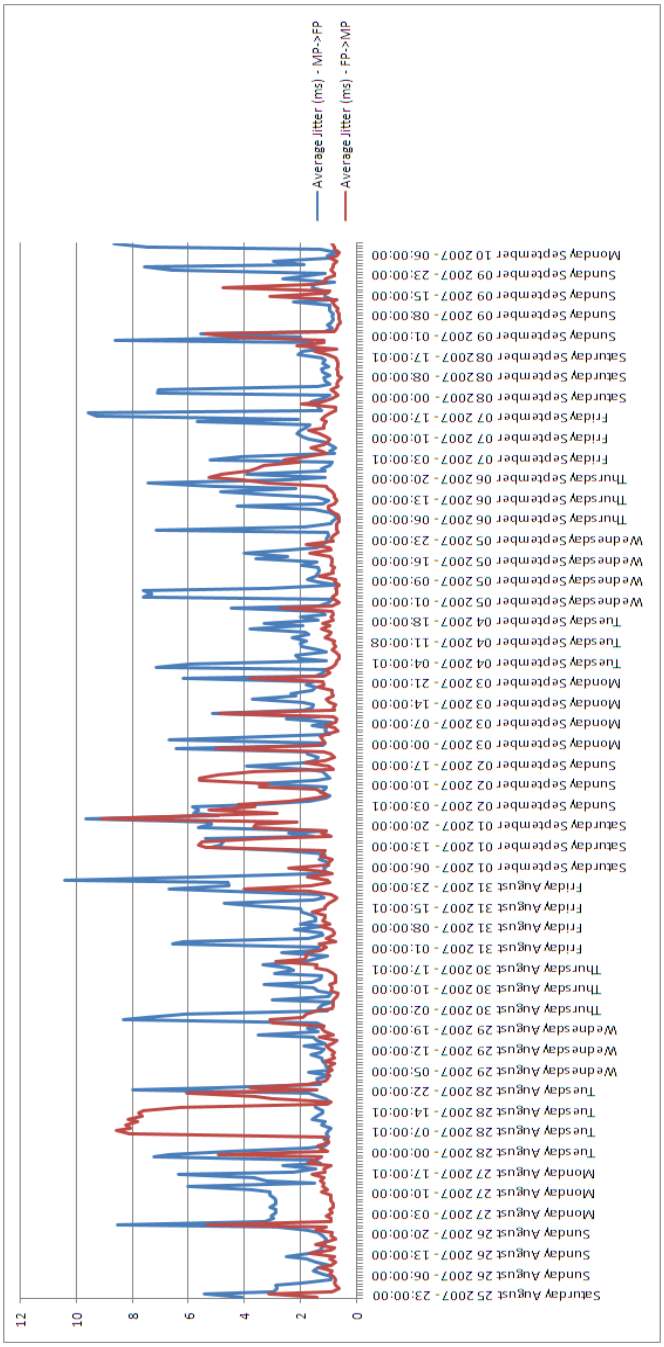

Figure 31: Average Jitter (ms)

In Figure 32 it is possible to see that the FP->MP path has a minimum jitter of almost 0 milliseconds while the reverse path has some variations. Even with these variations the minimum jitter remains mostly under 2 milliseconds.

Figure 32: Minimum Jitter (ms)

In Figure 33 it is possible to see that both paths have a very similar maximum jitter. This maximum jitter, most of the time is approximately 250 milliseconds.

Figure 33: Maximum Jitter (ms)

With this type of results a network administrator can infer the quality of the link between the two probes. This link is stable most of the time, the average delay is approximately 15 milliseconds, the lost packets are very low (almost no packets lost) and the average jitter is also very low, normally under 2 milliseconds.

Of course, these are average values and the results show some peaks, like the average jitter going over 10 milliseconds, the lost packets reaching more than 60% and the average delay reaching 35 milliseconds. These values represent isolated cases but can seriously affect the user experience. For example, if an user starts a VoIP call, and if during this call, the lost packets reach 60%, the user

will not be able to continue the call. The user would be very disappointed with the provided service.

6 - Conclusions

This document described many aspects of network QoS monitoring, including Active or Passive Monitoring, End-to-End or Distributed Monitoring and QoS Metrics. All of these aspects were detailed in order to design an architecture of a network QoS monitoring system.

The architecture of the monitoring system proposed was designed to support Active Monitoring and End-to-End Monitoring. The architecture proposed has the advantage of being based only on software. This provides the ability to adapt the system to the any needs. The architecture of the monitoring system is formed by a core system, fixed probes and mobile probes. The core system is formed by a few separated components. Some of these components provide interaction interfaces while others provide persistence of configurations and results.

Regarding the QoS Metrics, this is the responsibility of each dynamic module used to monitor QoS. These modules are dynamically loaded by the system and the probes. They are used by the probes to perform the tests, and during the tests the modules gather the information they need to acquire the QoS Metrics.

All the monitoring system was developed with Java technology. This technology allows the system to be used in almost every platform. Also, probes can be developed in another technology as long as that technology is compatible with SOAP based communications.

The dynamic modules provide the monitoring system a great scalability to support monitoring different types of services. Anyone can implement a new module and insert it in the system.

Due to time constraints the security part of the monitoring system was not implemented but this has not affected the main function of system.

The monitoring system was tested using one module. The module tests the VoIP service using the G.711 codec. The results show that this single module can provide many information about the quality of a single link, including average delay, packet loss and jitter.

6.1 - Future Work

Regarding future work there are some aspects that can be improved in the monitoring system developed.

First of all, the security part of the architecture should be implemented. This has not been implemented but it is a needed requirement for this system. If not implemented, this system in a production environment could not guarantee the consistency of the results.

One characteristic that could be implemented is the ability for a fixed probe to perform tests with more than one mobile probe at the same time. Currently, the fixed probes do not have the ability to do this.

One problem with the actual implementation is the lack of synchronization of modules configurations before the tests. This could be a problem in some situations, like two probes starting a test but one of them having a configuration to perform the test during five minutes, and another having the same configuration but to perform the same test during 10 minutes. The results of this test would not have any value.

Another problem with the actual implementation is in the administration and visualization interfaces, both of them do not have any kind of explaining to the user the errors he introduced. So, when a user makes a mistake when filling the forms, the interface only presents an error screen and does not show where the error is.

One feature that could be implemented is the ability to monitor SLAs, this would improve greatly the capabilities of this system. Instead of just performing tests, the probes could trigger alarms if at any time, the tests show that the SLA is not being fulfilled. These alarms could anything like an e-mail, an SMS or a phone call.

Finally, another feature that could be implemented is the ability to perform passive monitoring, this could probably be implemented by developing modules that would perform this type of monitoring.

7 - References

1. Tham CK, Jiang Y, Ko CC. Monitoring QoS distribution in multimedia networks. John Wiley & Sons, Ltd.; International Journal of Network Management; 2000; 75-90
2. Maia JL, Zorzo SD. Socket-Masking and SNMP: A Hybrid Approach for QoS Monitoring in Mobile Computing Environments. IEEE Computer Society; Proceedings of the XXII International Conference of the Chilean Computer Science Society (SCCC'02); 2002;
3. Asgari AH, Trimintzios, Irons M, Egan R, Pavlou G. Building Quality-of-Service Monitoring Systems for Traffic Engineering and Service Management. Plenum Publishing Corporation; Journal of Network and Systems Management, Vol. 11, No. 4, December 2003; 2003; 399-426
4. Yang SJ, Chou HC. Adaptive QoS parameters approach to modeling Internet performance. John Wiley & Sons, Ltd.; International Journal of Network Management; 2002; 69-82
5. Java - http://java.sun.com/
6. Java SE - http://java.sun.com/javase/
7. Java EE - http://java.sun.com/javaee/
8. Glassfish - http://glassfish.dev.java.net/
9. Java DB - http://developers.sun.com/javadb/
10. Netbeans - http://www.netbeans.org/

8 - Annex A – User Manual

8.1 - Core System Installation

To install the core system first you need to install Glassfish. Glassfish can be obtained at http://glassfish.dev.java.net/. The installation of Glassfish is not covered in this manual. In the Glassfish website, there are instructions on how to do this.

Also, you will need to install a DBMS. This is not covered in this manual. Check the user manual of the DBMS chosen.

In order to have Glassfish using the DBMS you to configure Glassfish using it's manual.

Finally, to install the core system in Glassfish follow the next steps:

1. Open the **Web Applications** screen and press the **Deploy...** button. This screen is shown in Figure 34.

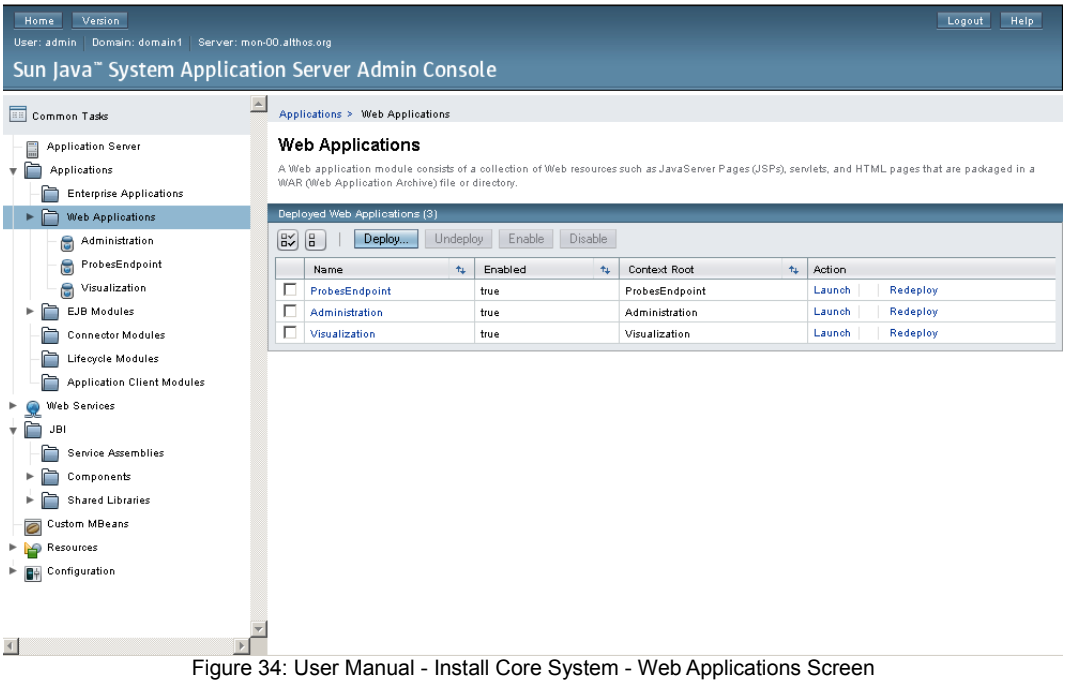

Figure 34: User Manual - Install Core System - Web Applications Screen

2. After pressing the Deploy... button the screen in Figure 35 will be presented. Press Browse... and choose the PCE component of the core system (ProbesEndpoint). Press OK in the end.

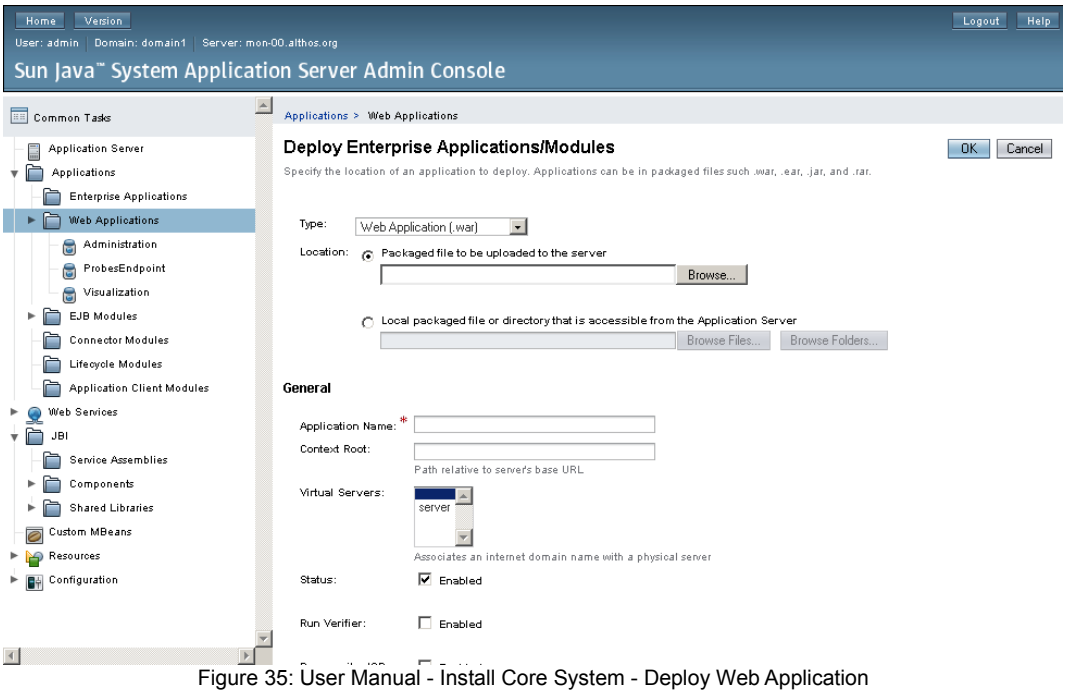

Figure 35: User Manual - Install Core System - Deploy Web Application

3. Repeat the same two previous steps for the components Administration Interface (Administration) and Visualization Interface (Visualization).

4. Open the **EJB Modules** screen and press the **Deploy...** button. This screen is shown in Figure 36.

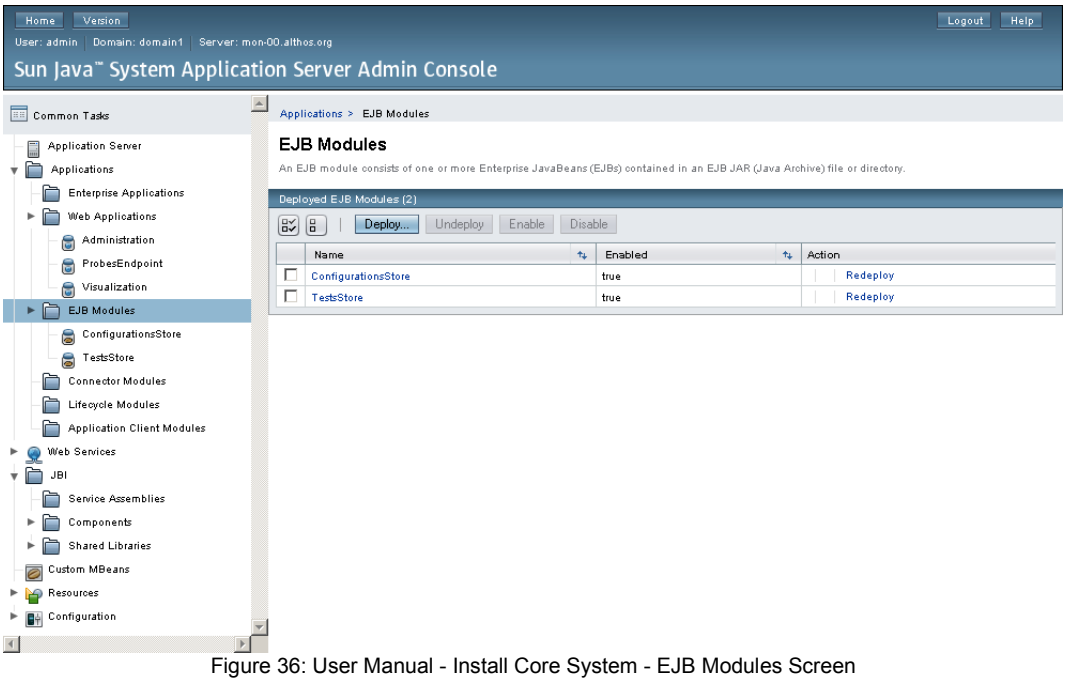

Figure 36: User Manual - Install Core System - EJB Modules Screen

5. After pressing the Deploy... button the screen in Figure 37 will be presented. Press Browse... and choose the PIR component of the core system (ConfigurationsStore). Press OK in the end.

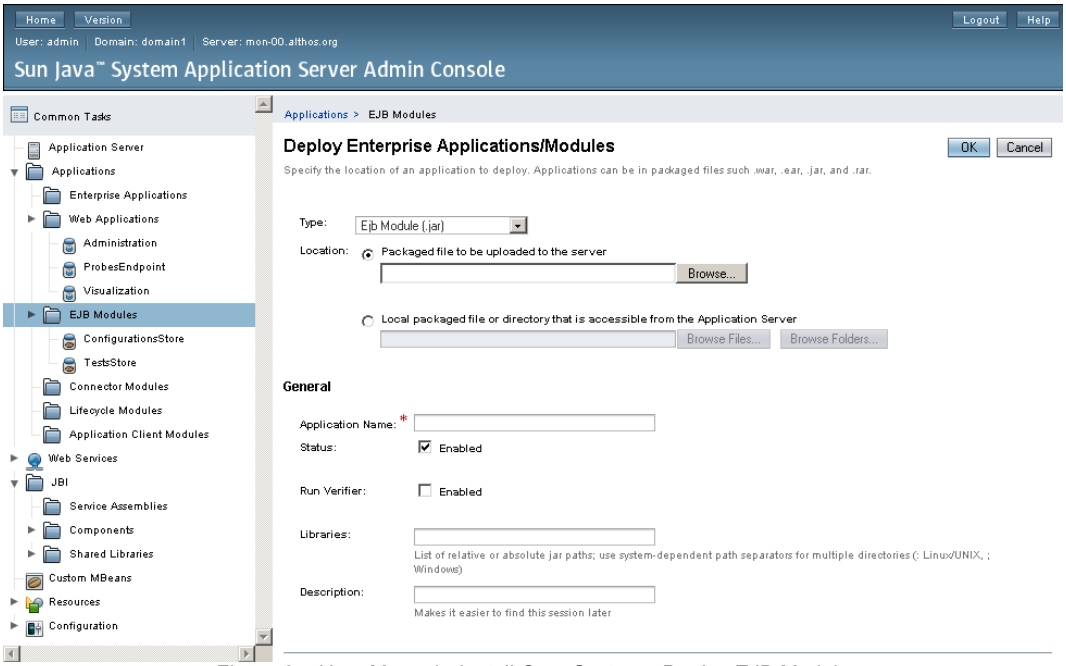

Figure 37: User Manual - Install Core System - Deploy EJB Module

6. Repeat the same two previous steps for the component TRR (TestsStore).

8.2 - Using the Administration Interface

The Administration Interface can be accessed with a web browser using the following address, replacing <server> with the server name:

http://<server>:8080/Administration

The port specified in the address can be different depending on the configuration of the Java Application Server.

8.2.1 - Add Virtual Probes

In order to add a virtual probe to the system you have to press the button shown highlighted in red in Figure 38.

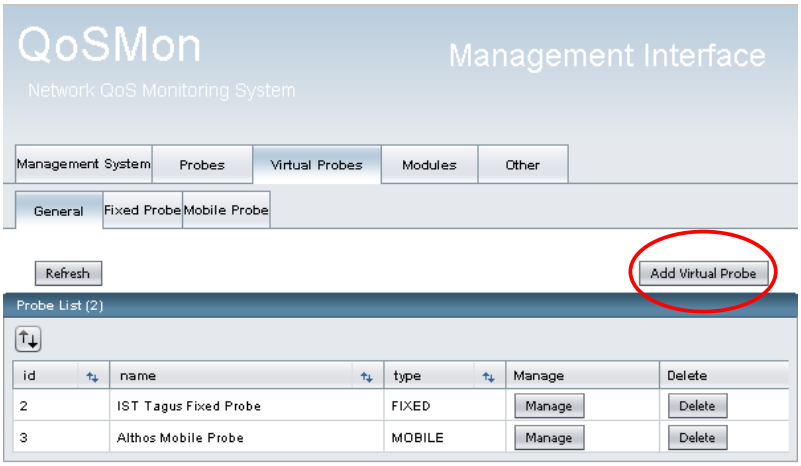

Figure 38: User Manual - Add Virtual Probe

After pressing the button, the screen shown in Figure 39 appears so you can add a virtual fixed probe to the system.

In this screen you can fill all the information about the fixed probe. This information includes the virtual probe name, location, server address that will be used to contact the core system and the corresponding port.

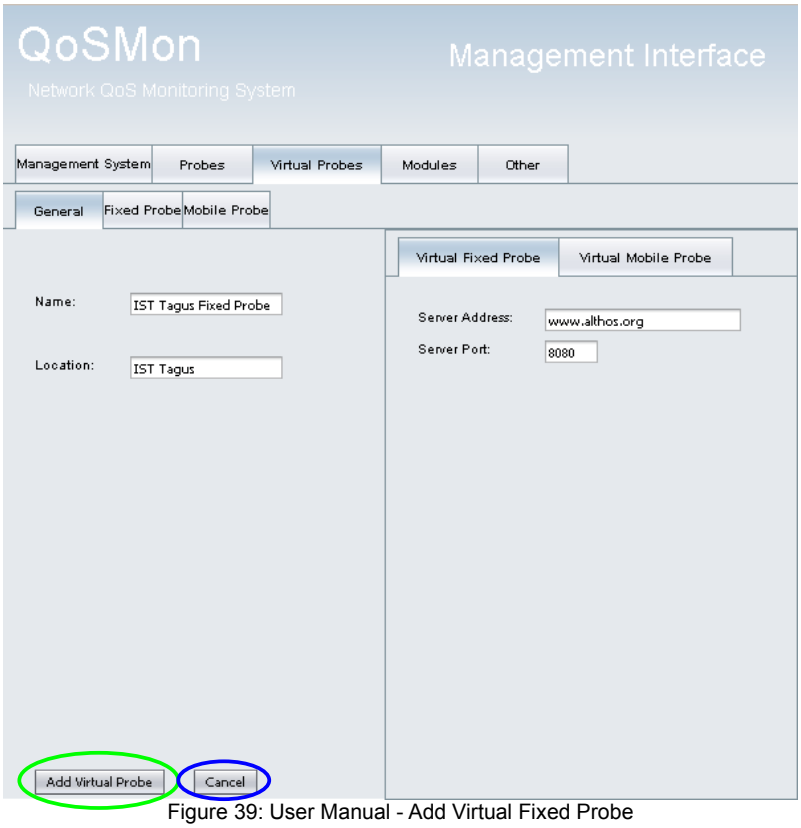

Figure 39: User Manual - Add Virtual Fixed Probe

In order to add a virtual mobile probe instead of a fixed one you will have to change the probe type separator to match a virtual mobile probe. To do this press the area highlighted in red in Figure 40.

The information needed to be filled almost the same as when adding a virtual fixed probe. The difference is that you don't have to provide information about the core system address and port.

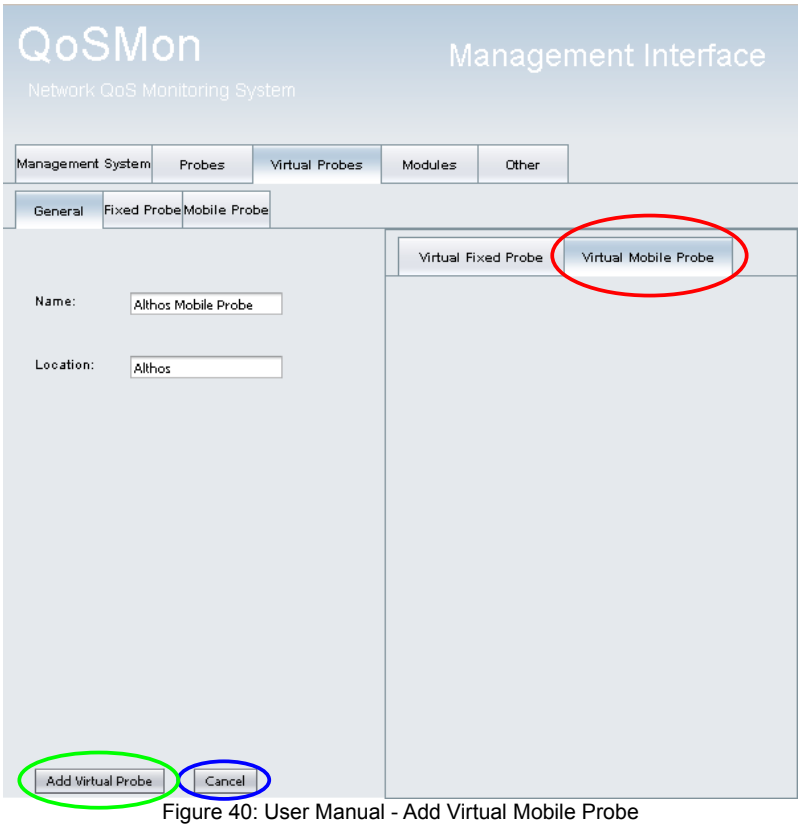

Figure 40: User Manual - Add Virtual Mobile Probe

In both cases after filling all the information you only need to press the “Add Virtual Probe” button highlighted in green in Figure 39 and Figure 40 to commit the changes. If you want to cancel the action just press the “Cancel” button highlighted in blue in Figure 39 and Figure 40. Either way you will end up in the first screen shown in Figure 38 listing all the virtual probes in the system.

8.2.2 - Manage Virtual Probes

To delete a probe you just need to press the “Delete” button highlighted in red shown in Figure 41. The virtual probe corresponding to the “Delete” button you pressed will be deleted from the system and the virtual probes list will refreshed.

To manage a virtual probe you have to press the “Manage” button highlighted in blue shown in Figure 41. This will lead you to another screen where the virtual probe corresponding to the “Manage” button you pressed will be shown and you can change its parameters.

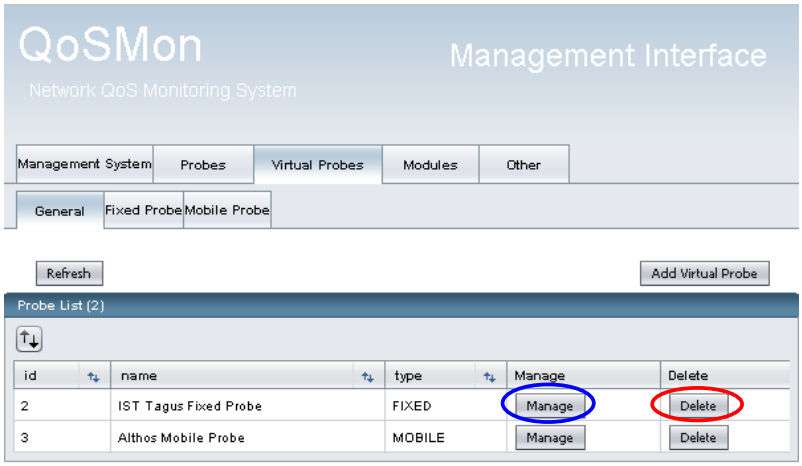

Figure 41: User Manual - Manage Virtual Probes

If the virtual probe you chose was a fixed one, the screen show in Figure 42 appears. From this screen you can not only change the information about the virtual probe you set before when adding it to system but also, you can manage the modules enabled for this virtual probe by using the buttons highlighted in green.

If the virtual probe you chose was a mobile one, the screen in Figure 43 is show. The actions in this screen are exactly the same as in the screen in Figure 42. The difference between these two screens is that the information available to change regarding the probe is different.

When done press the “Save” button highlighted in red to save the changes made or the “Re-set” button highlighted in blue to reset all the changes made. To return to the virtual probes list press the tab highlighted in yellow.

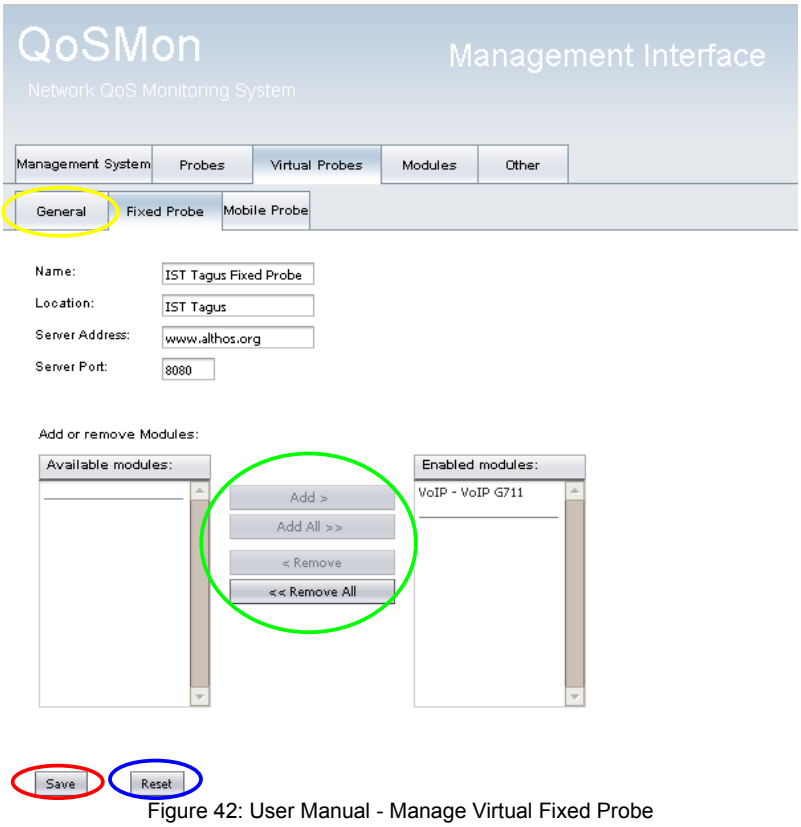

Figure 42: User Manual - Manage Virtual Fixed Probe

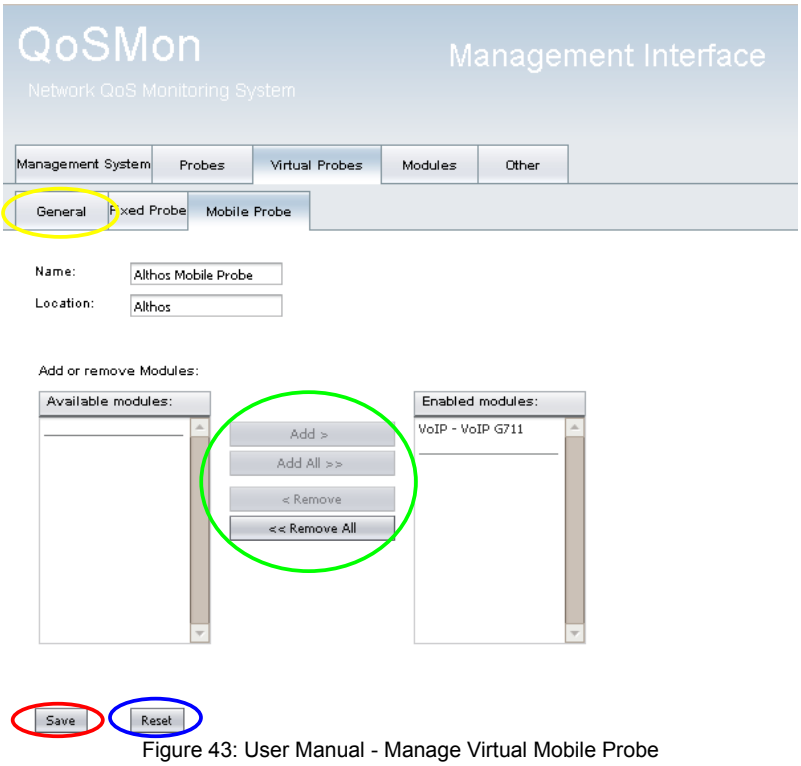

Figure 43: User Manual - Manage Virtual Mobile Probe

8.2.3 - Add Probes

To add a probe to system you must have already added a virtual of the same type of the

probe you are intending to add.

To add the probe navigate through choose the separators highlighted in red in Figure 44. After reaching the screen shown in that illustration select a virtual fixed probe from the drop down list and press the “Select” button highlighted in blue.

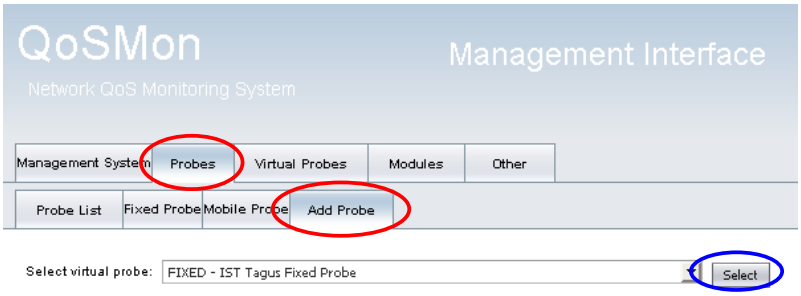

Figure 44: User Manual - Add Probe

After pressing the “Select” button you will be presented with one of the screens shown in Figure 45 and Figure 46. The screen shown in Figure 45 corresponds to the action of adding a fixed probe, while the screen show in Figure 46 corresponds to the action of adding a mobile probe.

When adding a fixed probe you are presented with the information set in the virtual fixed that you have chosen previously. You also have to insert information about the probe IP, the IP to which the probe will bind to and the port which will be used by the probe.

When finished, press the “Add Probe” button highlighted in red.

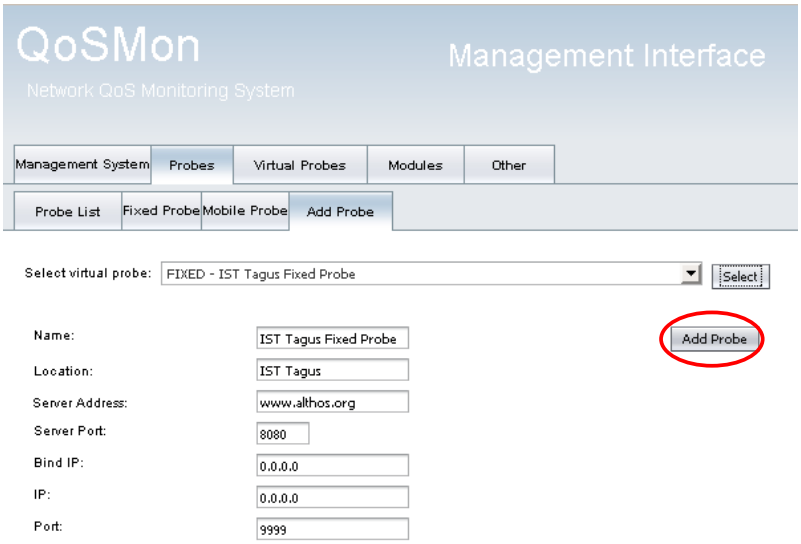

Figure 45: User Manual - Add Fixed Probe

When adding a mobile probe you are presented with the information set in the virtual mobile that you have chosen previously.

When finished, press the “Add Probe” button highlighted in red.

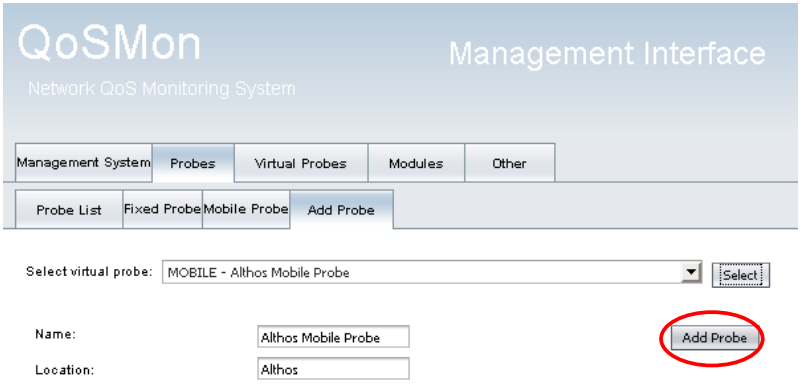

Figure 46: User Manual - Add Mobile Probe

8.2.4 - Manage Probes

You can manage the probes on the system from the screen presented in Figure 47. From this screen you can delete a specific probe by pressing a “Delete” button as the one highlighted in red and manage a specific probe by pressing a “Manage” button as the one highlighted in blue.

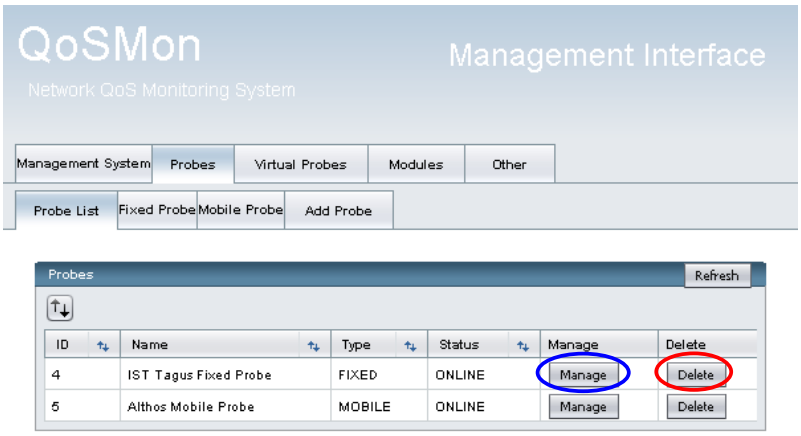

Figure 47: User Manual - Manage Probes

When you choose to manage a fixed probe you will be presented the screens show in Figure 48. From this screen you can change the probe's information about its name, location, IP, port, and the information about the server.

You can save the changes by pressing the “Save” button highlighted in red or reset the changes by pressing the “Reset” button highlighted in blue.

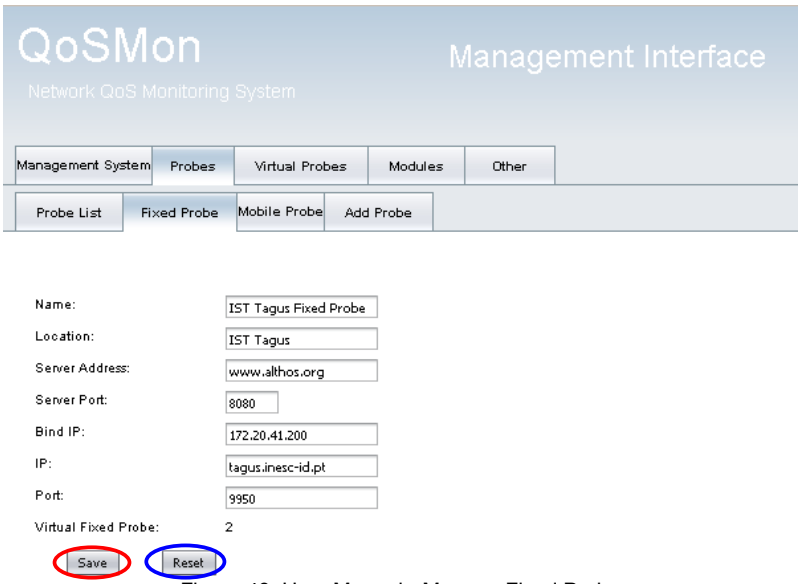

Figure 48: User Manual - Manage Fixed Probe

If you choose to manage a mobile probe you will be presented with a screen like the one shown in Figure 49. This screen, besides allowing you to change the name and location of the mobile probe, it also allows you to manage the mobile probes schedules.

You can add, delete, enable or disable schedules.

To delete a schedule you press the corresponding button, the “Delete” button highlighted in yellow. The schedule will be removed and the schedules list will be refreshed.

To enable or disable a schedule press the button on the “Enabled” column like the one highlighted in yellow. If the button shows the text “true” it means the schedule is enabled, if it shows “false” it means the schedule is disabled.

To add a schedule you use the gray box on the right side of the screen. In Figure 50 you have a more detailed view of the box also showing the list of fixed probe from where you choose which fixed probe will be used to perform the test. On this screen you set the start date and finish date on the input boxed highlighted in blue. If you press the button on the right side of this input boxes a calendar will show up to help you choose the dates. To set the week days when this schedule will be run, use boxes highlighted in green, some of the days can't be viewed in Figure 50 but can be viewed in Figure 49. To set the time when the schedule will run use the drop down lists on the area highlighted in yellow. To choose the fixed probe which will be used to perform the test, press the “Show List” button highlighted in pink. This will bring a list with all the fixed probes on the system, choose one and press the “Choose Probe” button highlighted in black, to cancel press the button “Cancel” highlighted in brown. Finally, to add this schedule to probe press the “Add Schedule” button highlighted in red. To reset all the values press the “Reset” button highlighted in orange.

When you are finished you can save the changes by pressing the “Save” button highlighted in red or reset the changes by pressing the “Reset” button highlighted in blue.

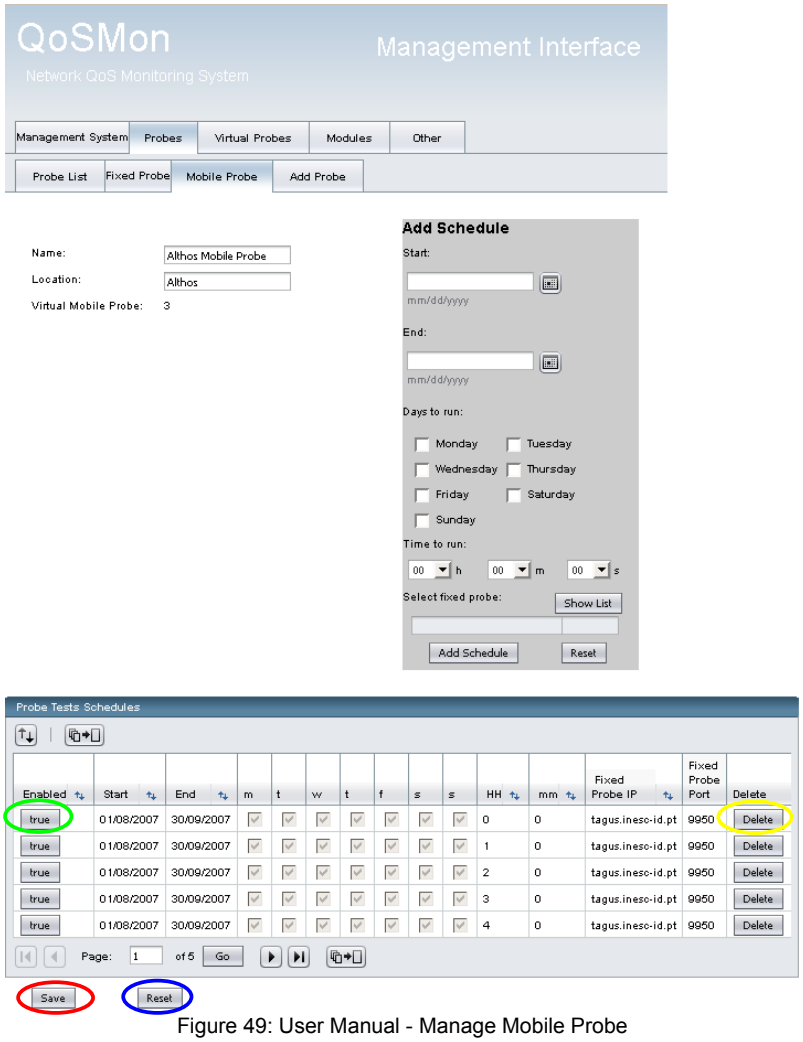

Figure 49: User Manual - Manage Mobile Probe

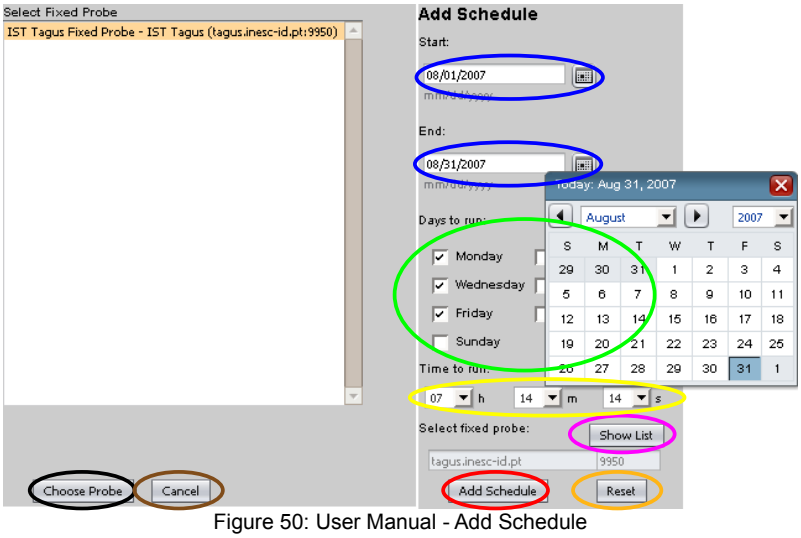

Figure 50: User Manual - Add Schedule

8.2.5 - Add Dynamic Module

To add a dynamic module to the system press the “Browse” button highlighted in blue show in

Figure 51 and search for it on your local system. After you choose the module, press the “Add Module” button highlighted in red.

After adding the module, it should appear in the drop down list of modules.

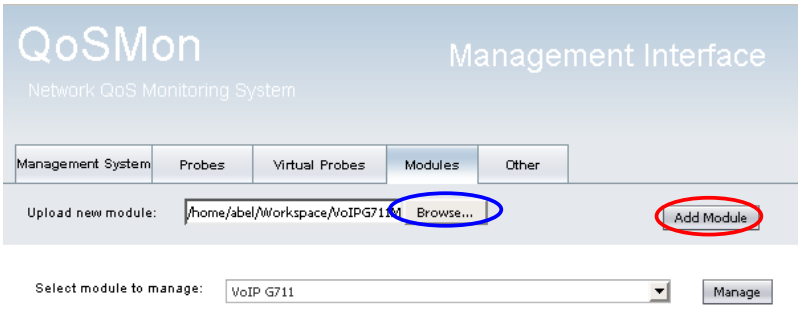

Figure 51: User Manual - Add Dynamic Module

8.2.6 - Manage Dynamic Modules

To manage a module choose it from the drop down list highlighted in blue in Figure 52 and press the “Manage” button highlighted in yellow.

When you press the “Manage” button you see some information about the module, two buttons and properties table for the module.

To remove the module from the system press the “Remove Module” button highlighted in red.

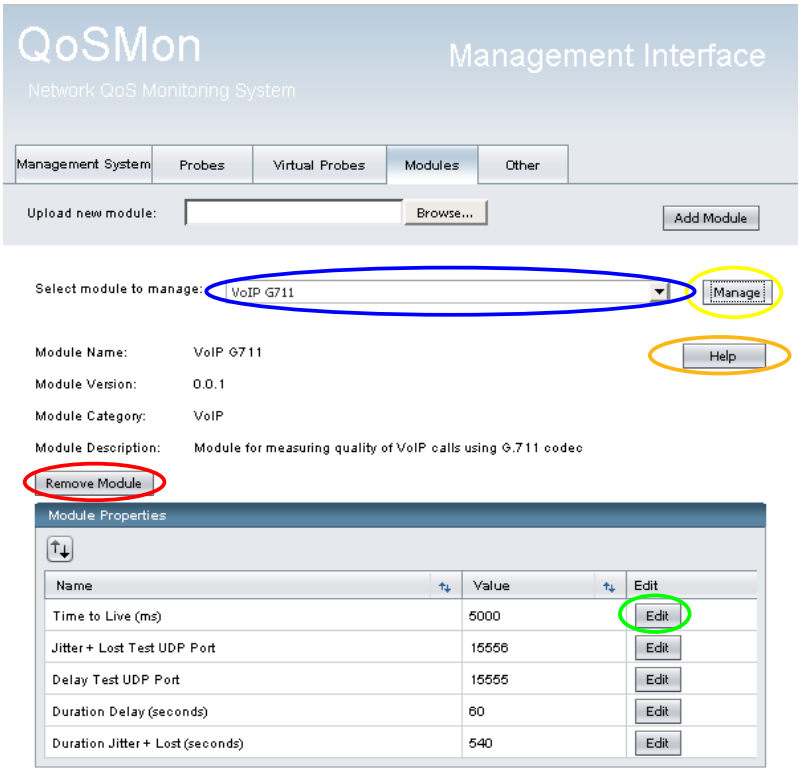

Figure 52: User Manual - Manage Dynamic Module

To view some help associated with the module properties press the “Help” button highlighted

in orange in Figure 52. After pressing the button you will be presented with a screen like the one in Figure 53. To close the help simply press the “Close Help” button highlighted in blue.

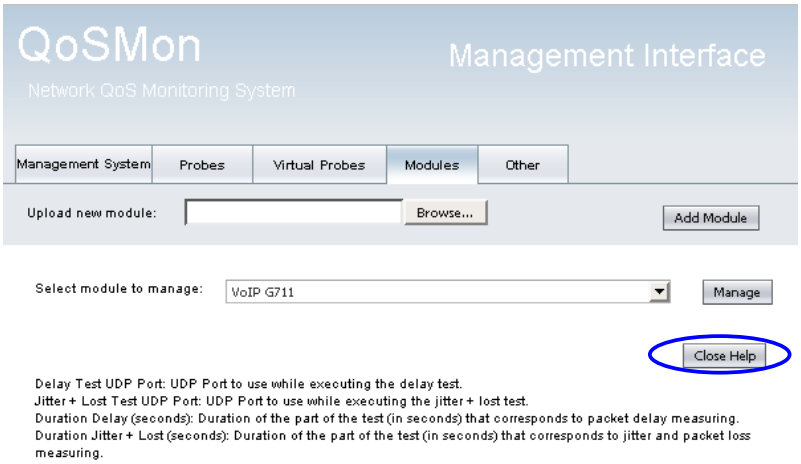

Figure 53: User Manual - Dynamic Module Help

To change the properties of the module press the corresponding “Edit” button highlighted in green in Figure 52. After pressing the button you will be presented with a screen like the one in Figure 54. In this screen you can change the value of the property. When done press the “Save” button highlighted in red. If you want to cancel the change press the “Cancel” button highlighted in blue.

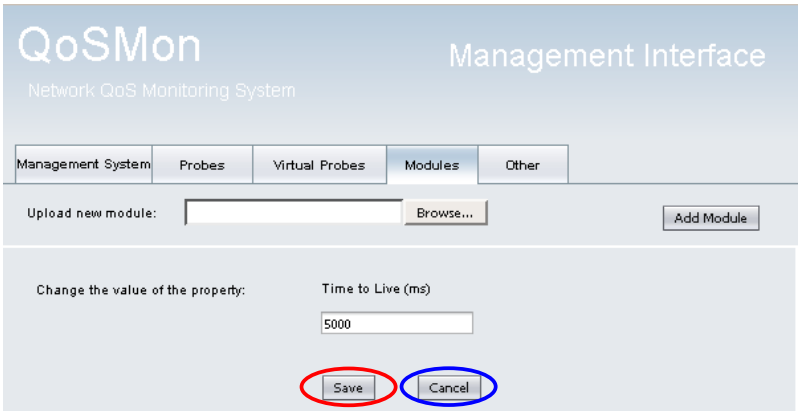

Figure 54: User Manual - Change Module Property

8.3 - Using the Visualization Interface

The Visualization Interface can be accessed with a web browser using the following address, replacing <server> with the server name:

http://<server>:8080/Visualization

The port specified in the address can be different depending on the configuration of the Java Application Server.

8.3.1 - Search for Results

The first screen of the visualization interface is shown in Figure 55. In this screen you can filter the results by probes involved in the tests and dates when the tests were performed. Probes can be selected from the drop down lists highlighted in blue and the dates can be set in the input boxes highlighted in green. If you press a button on the right of one of the date input boxes, a calendar will show up to help you select the desired date.

After applying the filters, you can start the search by pressing the “Search” button highlighted in red. If you want to clear the filters press the “Clear” button highlighted in yellow.

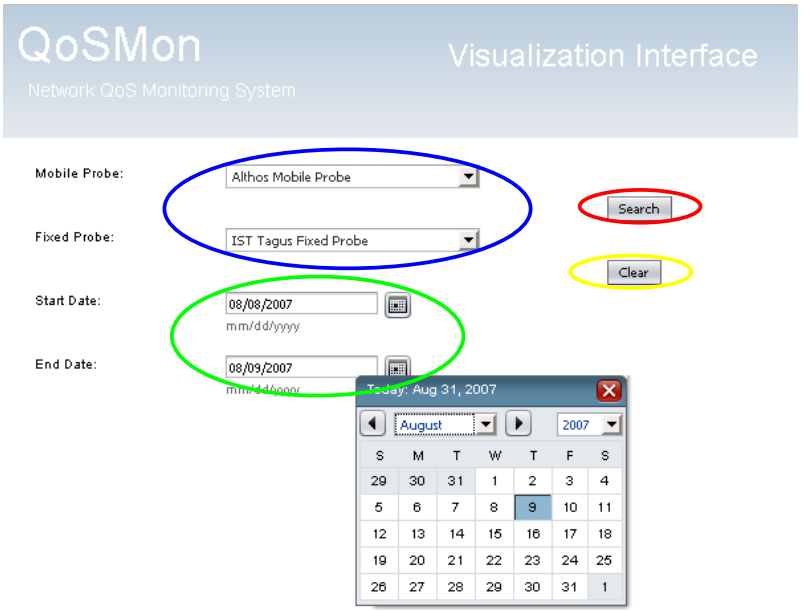

Figure 55: User Manual - Search Results

After starting the search you will see a screen like the one in Figure 56. In this screen you choose from which module you want to view the results. After selecting the module from the drop down list press the “OK” button highlighted in red. To cancel and return to the previous screen press the “Cancel” button highlighted in blue.

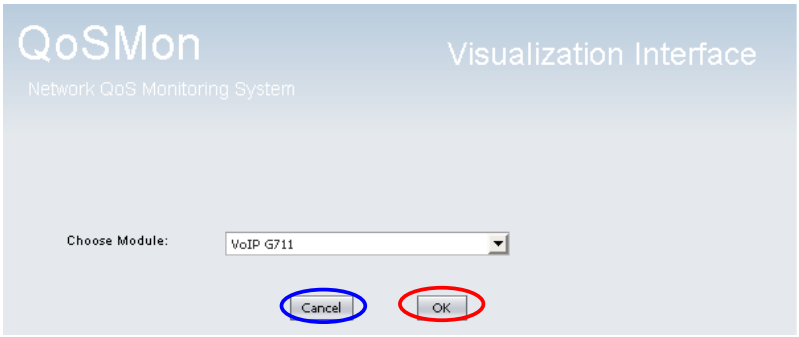

Figure 56: User Manual - Choose Module

8.3.2 - Results Screen

After selecting the module a screen with the results like the one in Figure 57 will appear. From this screen you can see all the results that satisfy the filters you selected.

Test ID	Test Time	FP ID	FP Name	MP ID	MP Name	Path	Module	Schedule ID	Result ID	Result Name	Result Value
2	Tue Aug 28 00:00:00 WEST 2007	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	FP->MP	VoIP G711	15	1	Minimum Delay (ms)	12.7625
2	Tue Aug 28 00:00:00 WEST 2007	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	FP->MP	VoIP G711	15	0	Average Delay (ms)	17.16848096836066
2	Tue Aug 28 23:00:01 WEST 2007	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	MP->FP	VoIP G711	38	4	Received Packets	26984.0
2	Tue Aug 28 23:00:01 WEST 2007	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	MP->FP	VoIP G711	38	3	Lost Packets	16.0
2	Tue Aug 28 23:00:01 WEST 2007	4	IST Tagus Fixed Probe	5	Althos Mobile Probe	MP->FP	VoIP G711	38	1	Minimum Delay (ms)	12.764075

Download CSV file

Figure 57: User Manual - Results List

In the bottom of the screen in Figure 57 you have a link to download a file with the results in a comma-separated values format (csv). This file will contain a content like this:

```
Test ID,Test Time,FP ID,FP Name,MP ID,MP Name,Path,Module,Schedule ID,Result ID,Result Name,Result Value,
2,Tue Aug 28 00:00:00 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,FP->MP,VoIP G711,15,1,Minimum Delay (ms),12.7625,
2,Tue Aug 28 00:00:00 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,FP->MP,VoIP G711,15,0,Average Delay (ms),17.1684809836066,
2,Tue Aug 28 00:00:00 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,FP->MP,VoIP G711,15,7,Minimum Jitter (ms),0.0080,
[..... ]
2,Tue Aug 28 23:00:01 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,MP->FP,VoIP G711,38,4,Received Packets,26984.0,
2,Tue Aug 28 23:00:01 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,MP->FP,VoIP G711,38,3,Lost Packets,16.0,
2,Tue Aug 28 23:00:01 WEST 2007,4,IST Tagus Fixed Probe,5,Althos Mobile Probe,MP->FP,VoIP G711,38,1,Minimum Delay (ms),12.764075,
```

8.4 - Working with the Probes

All the following topics assume that you are working on a folder with the Probes.jar file and the lib folder with the FixedProbesEndpointInterfaces.jar, MobileProbesConfiguration.jar and Module.-jar files.

8.4.1 - Starting the Probes

To start a fixed probe issue the following command:

- `java -jar Probes.jar org.althos.qosmon.probes.start.StartFixedProbe`

To start a mobile probe issue the following command:

- `java -jar Probes.jar org.althos.qosmon.probes.start.StartMobileProbe`

8.4.2 - Managing the Probes

8.4.2.1 - Fetch Configurations

To fetch configurations from a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient -ip <IP> -port <PORT> -id <ID> fetch-config`

To fetch configurations from a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient -ip <IP> -port <PORT> -id <ID> fetch-config`

On both cases replace <IP> by the server IP for the fixed probe and by the fixed probe IP for the mobile probe. Also replace <PORT> by the server port for the fixed probe and by the fixed probe port for the mobile probe. Finally, replace <ID> for the probe id.

8.4.2.2 - Fetch Modules

To fetch modules from a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient -ip <IP> -port <PORT> fetch-modules`

To fetch modules from a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient -ip <IP> -port <PORT> fetch-modules`

On both cases replace <IP> by the server IP for the fixed probe and by the fixed probe IP for the mobile probe. Also replace <PORT> by the server port for the fixed probe and by the fixed probe port for the mobile probe.

8.4.2.3 - Show Configurations and Modules

To show configurations of a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient list-configurations`

This will print a result like this one:

```
java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient list-configurations
----------------------------------------
ID: 4
Name: IST Tagus Fixed Probe
Location: IST Tagus
Bind IP: 172.20.41.200
IP: tagus.inesc-id.pt
Port: 9950
Enabled: true
Server IP: www.althos.org
Server Port: 8080
```

```
----------------------------------------
```

To show configurations of a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient list-configurations`

This will print a result like this one:

```
java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient list-configurations
----------------------------------------
ID: 5
Name: Althos Mobile Probe
Location: Althos
IP:
Port: 11555
Enabled: true
Number of schedules: 24
----------------------------------------
```

To show the modules of a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient list-modules`

This will print a result like this one:

```
java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient list-modules
----------------------------------------

Name: VoIP G711
Version: 0.0.1
Category: VoIP
Description: Module for measuring quality of VoIP calls using G.711 codec

Configurations:
Time to Live (ms): 5000
Duration Delay (seconds): 60
Jitter + Lost Test UDP Port: 15556
Duration Jitter + Lost (seconds): 540
Delay Test UDP Port: 15555

----------------------------------------
```

To show configurations of a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient list-modules`

This will print a result like this one:

```
java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient list-modules
----------------------------------------

Name: VoIP G711
Version: 0.0.1
Category: VoIP
Description: Module for measuring quality of VoIP calls using G.711 codec
----------------------------------------
```

To show the schedules of a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManage-`

```
mentClient list-schedules
```

This will print a result like this one:

```
java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient list-schedules
----------------------------------------

Schedule ID: 37
Schedule Enabled: true
Schedule Start: Thu Aug 02 00:00:00 WEST 2007
Schedule End: Mon Oct 01 00:00:00 WEST 2007
Schedule Weekdays: M T W T F S S
Schedule Time: 22:0:0
Schedule Fixed Probe IP: tagus.inesc-id.pt
Schedule Fixed Probe Port: 9950
----------------------------------------

Schedule ID: 38
Schedule Enabled: true
Schedule Start: Thu Aug 02 00:00:00 WEST 2007
Schedule End: Mon Oct 01 00:00:00 WEST 2007
Schedule Weekdays: M T W T F S S
Schedule Time: 23:0:0
Schedule Fixed Probe IP: tagus.inesc-id.pt
Schedule Fixed Probe Port: 9950
----------------------------------------
```

8.4.2.4 - Enable or Disable a Probe

To enable a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient enable`

To enable a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient enable`

To disable a fixed probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.FixedProbeManagementClient disable`

To disable a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient disable`

8.4.3 - Run a Test

To run a test from a mobile probe issue the following command:

- `java -cp Probes.jar org.althos.qosmon.probes.management.MobileProbeManagementClient -ip <IP> -port <PORT> run-test`

Replace <IP> by the fixed probe IP and replace <PORT> by the fixed probe port.

9 - Annex B – Screenshots

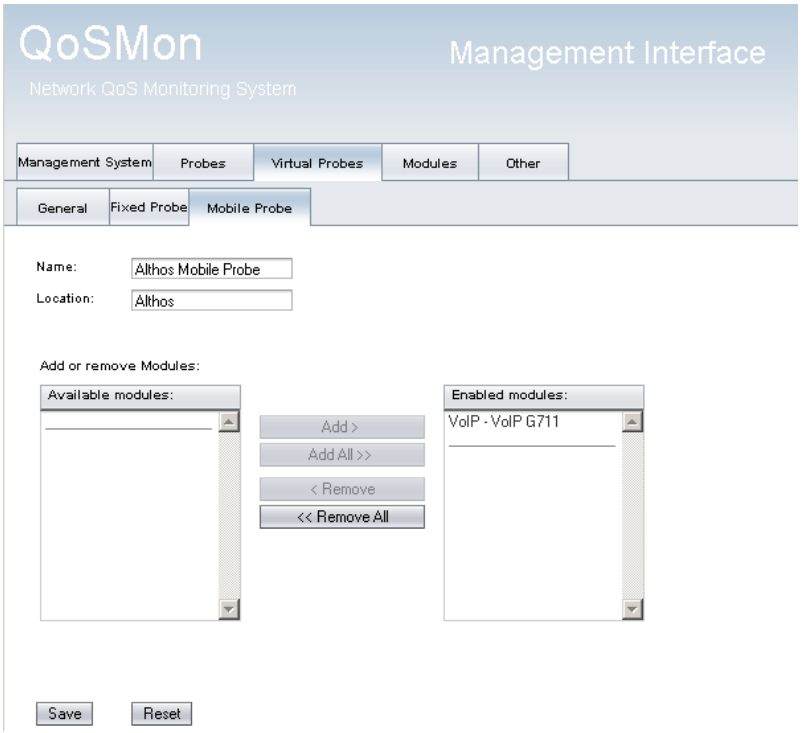

Figure 58: Core System Administration Interface: Manage Virtual Mobile Probe

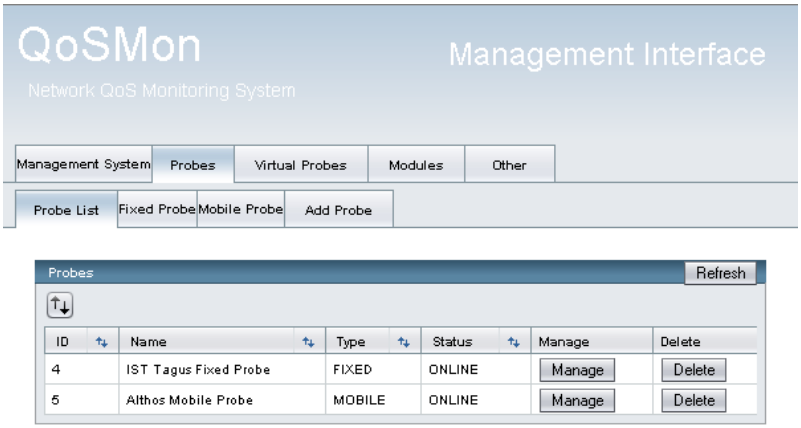

Figure 59: Core System Administration Interface: Probes List

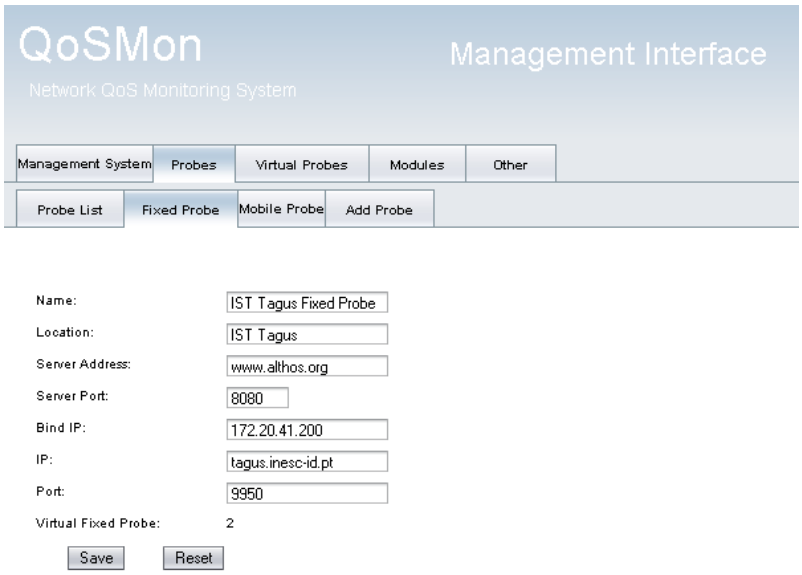

Figure 60: Core System Administration Interface: Manage Fixed Probe