**University College London**
Department of Computer Science
M.Sc Data Communications, Network and Distributed Systems

# LINX Network Monitoring
## Group Report

## Group Members
Agron Fazliu
Alexander Papitsch
Andreas Protopapas
Felipe Huici
Giorgos Savvides

## Supervisor
Saleem Bhatti

## In Collaboration With
The London Internet Exchange

September 6th, 2004

# Abstract

Traditionally, Internet Exchange Points (IXPs) have been developing network monitoring toolkits that are specifically suited to their needs, requirements and infrastructure. Because of this approach, it has rarely been possible for one IXP to use a monitoring tool developed by another IXP. The LINX network monitoring project, conducted with the collaboration of the London Internet Exchange Point, addresses precisely this problem. This report describes the design of an entirely new system architecture on which future network monitoring tools can be based so that they can be deployed on IXPs having disparate hardware and software infrastructures. Further, this report discusses the implementation of a network monitoring toolkit that is based on said system architecture in addition to a description of the necessary set of standards used to make this inter-IXP compatibility possible. The actual monitoring functionality implemented is based on requirements provided by LINX throughout the course of the project. Moreover, the report touches on project management including the methodology used, the project's requirements and the technologies employed. Finally, the report discusses future work that could be carried out to improve on the achievements of this project.

# Acknowledgements

We would like to acknowledge Dr. Saleem Bhatti who kindly supervised this DCNDS project. We would also like to thank him for his invaluable suggestions, feedback, resources and ideas that he provided during all stages of this project.

In addition, we would like to thank John Souter and Robert Lister from LINX who made this project possible and also spared time and resources for analysis, testing and deployment purposes at LINX headquarters.

# Table of Contents

# List of Figures and Tables

## Chapter 5



## Chapter 6



## Chapter 7



## Chapter 8



## Chapter 9



## Chapter 11



## Chapter 12



## Chapter 13



## Appendix A

# Chapter 1: Introduction

## 1.1 Purpose

The purpose of this document is to provide its readers with a thorough description of the project. It aims to give a detailed account of the processes adopted and the many decisions that were made by the group during the different phases of the system development lifecycle that allowed the implementation of a functional network monitoring toolkit that met the client's needs. Furthermore, this report incorporates a detailed outline of the project's objectives, its scope, the requirements analysis, the system design and the implementation phases, and discusses the project management techniques employed. The report also includes an account of system testing and an evaluation of whether the objectives have been met. Finally, this report concludes with a section on future work suggesting enhancements to the system.

## 1.2 Client Background

This project differs from many other DCNDS projects which have been undertaken in the past and are currently being undertaken in that it is being carried out on behalf of a real customer. The customer in this case is LINX, which stands for the London Internet Exchange. A detailed description of what an Internet Exchange Point (IXP) is and what purpose it serves can be found in Section 2.2 of this report.

LINX was founded in 1994 and is regarded as the largest and most successful Internet Exchange Point in Europe with a world class reputation for quality, performance and technical excellence. Being a non-profit organisation, LINX is also one of the founder members of Euro IX, the European Internet Exchange Association. Euro IX was founded in May of 2001 with the goal to further develop, strengthen and improve the Internet Exchange Community throughout Europe.

## 1.3 Existing Tools

The Milan Internet Exchange (MIX) created, in cooperation with the University of Padova, the Inter-Provider Network Analyzer (IPNA). Although this tool also serves for network monitoring purposes, it is very different from the tool delivered by this project. More specifically, IPNA spots anomalies in traffic, analyzes the relationships among these anomalies and displays these results in a user interface. It bases its analysis on Expected Traffic Intensity (ETI), tracking fluctuations in the

amount of traffic that runs through the network. While IPNA is a useful tool, it was specifically designed for MIX: the format that the data is captured in and that is subsequently used to display graphs is not standardized, nor is the actual display of the graphs. As a result, IPNA is not easily ported, so other IXPs cannot benefit from its analysis.

Another tool that exists is SmokePing. SmokePing is a network latency monitoring tool which is written in Perl. It consists of a daemon process responsible for data collection and a CGI script used for displaying the data in graphical form on the web for easy interpretation. SmokePing has some probes which integrate the pinging utility `fping` in order to measure round trip times. Because some IXPs are already utilising SmokePing in order to monitor the status of their networks, it would be relatively easy for them to integrate LinkStatus and MinMaxPing, two of the tools developed for this project, since they both make use of `fping` to generate data (please refer to Sections 6.4 and 6.5, respectively, for a description of these tools). These IXPs could then benefit from the fact that the toolkit developed for this project does not only yield graphs to the client like SmokePing does, but rather the data themselves; this provides an IXP with much greater flexibility (for instance, a programmer would not be forced to create a graphical user interface when a much simpler text-based client would suffice to display the data).

Yet another tool is the Multi Router Traffic Grapher (MRTG). MRTG is written in Perl and the time-critical sections in the C programming language. This tool is currently available for UNIX and Windows NT. MRTG measures the traffic load on network links, constructs relevant graphs and generates HTML-format pages containing PNG images which provide a live visual representation of this traffic. Like SmokePing, MRTG has the shortcoming that it gives graphs of the data, not the data themselves. If the network administrator is not content with the type of graph that MRTG uses, he or she would not be able to create a more suitable graph since MRTG does not yield the actual data. The toolkit developed for this project does not have this limitation and, unlike MRTG, it is platform independent.

## 1.4 Problem Space

As part of a DCNDS weekly seminar, John Souter from LINX gave a presentation on the issue of monitoring network traffic at the London Internet Exchange. During this presentation it became apparent that collaborating IXPs find it

difficult to share traffic data for comparison and analysis purposes. Yet another problem is that of one IXP wanting to share a tool it has created with other IXPs, since each IXP runs different types of hardware and the tool is unlikely to work with all of them.

As discussed during the presentation, LINX has developed a network and alarm monitor toolkit which is suited to LINX's hardware and meets its requirements. The presentation also made it clear that it is common practice for each IXP to develop its own network monitoring tools. Since these tools are typically built to work with the existing network hardware, it is unlikely that they will also work with hardware from other IXPs. As a result, IXPs retrieve, store, analyse and display traffic data in a variety of different ways, which makes it nearly impossible for data and software to be shared with other IXPs.

This project therefore aimed to investigate a possible solution which would make it possible for IXPs to use the same network monitoring toolkit and allow them to share data as a result of using a common data representation. Once a solution was found, a system architecture was designed and a network monitoring toolkit implemented based on this architecture.

## 1.5 Motivation for the Work

Most of the major Internet Exchange Points in Europe share the same problem. Due to in house development of network monitoring tools which do not comply with any standard, most Internet Exchange Points cannot exchange traffic data for troubleshooting purposes. Having custom built tools means that data are represented in different manners, sometimes in a text-based form, others visually in a graphical user interface; this makes it complicated for interested parties to compare their internal data with data from other IXPs. Many IXPs would be interested in a solution that not only conquers this problem but also allows for remote access to real time data of other member IXPs. When John Souter came to University College London on November 25th, 2003, he outlined the data exchange problems IXPs were facing and suggested this as a challenging and rewarding project for a DCNDS group. The group was keen to undertake this project, as it entailed finding a solution to a real problem and developing a system that could potentially be used by many IXPs throughout Europe and perhaps even beyond.

# Chapter 2: Background

## 2.1 Introduction

This chapter provides the reader with some background information on general and technological aspects that are relevant to this project. Since the project was carried out on behalf of an IXP some background information on IXPs is included.

## 2.2 Internet Exchange Point (IXP)

According to Euro-IX, an Internet Exchange Point (abbreviated as IXP in this text) is defined as a physical network infrastructure operated by a single entity with the purpose of facilitating the exchange of Internet traffic between Internet Service Providers.

An IXP is a network to which many ISPs can connect. Any ISP that is connected to the IXP can exchange traffic with any of the other ISPs connected to the IXP; this is done using a single physical connection to the IXP, thus overcoming the scalability problem of having multiple individual interconnections. Also, by enabling traffic to take a more direct route between many ISP networks, an IXP can improve the efficiency of the Internet, resulting in better and cheaper service for the end user. Furthermore, since many networks have more than one connection to the Internet, it is not unusual to find several routes to the same network available at an IXP, thus providing a certain amount of fault tolerance.

Since it acts as a central point for traffic exchange, an IXP must ensure the effective and reliable operation of its network infrastructure. IXPs therefore rely on software tools which allow them to monitor network traffic and inform them of any occurring or potential problems. To ease the troubleshooting process it would be beneficial if IXPs could exchange information and monitor other IXPs' traffic to share knowledge of past problems as well as current traffic patterns. Currently remote monitoring of other IXPs and information exchange are not possible due to the incompatibility of each IXP's monitoring tools and, consequently, its data. As a result, this project implemented a standardised network monitoring toolkit that could be used to address this exchange problem.

## 2.3 Peering

Peering is the arrangement of traffic exchange between Internet Service Providers. An Internet Exchange Point is the physical location where Autonomous Systems are physically interconnected with each other and where traffic is forwarded to the peering partners. Peering agreements are usually negotiated between ISPs. IXPs are not, generally, involved in the peering agreements between connected ISPs; whom an ISP peers with, and the conditions of that peering, are a matter for the two ISPs involved. IXPs such as LINX do however have requirements that an ISP must meet to connect to the IXP.

## 2.4 Autonomous System

An autonomous system (AS) is a collection of IP networks under control of a single entity. This single entity is usually assumed to be an Internet Service Provider. Each autonomous system is allocated a unique AS number to be used in BGP routing. AS numbers are assigned by the same authorities that assign IP addresses. There are public AS numbers that range from 1 to 64511 usually used on the Internet, as well as private AS numbers which are in the range from 64512 to 65535 and are to be used within an organisation.

## 2.5 Secure Sockets Layer

The Secure Sockets Layer (SSL) is a protocol developed by Netscape with the purpose of transmitting private information across the Internet in a secure way. SSL operates at the socket interface, located between transport and application layer in the TCP/IP model. SSL can either be used to perform one-way authentication, which is usually the authentication of the server by the client, or can also be used to perform mutual authentication, which includes both the authentication of the server by the client and the authentication of the client by the server. Public key cryptography and certificates can be used to obtain such authentication and all communication is encrypted using the strongest cipher common to both communicating parties.

## 2.6 Current LINX network monitoring tool (IXP-watch)

The current network monitoring tool used by LINX is called IXP-watch. IXP-Watch combines information from SNIPS and Syslog-ng (see sections 2.6.1 and 2.6.2 respectively). It suppresses downstream alarms and, at the same time, it correlates

and classifies events and decides how to treat these. The created output contains information which is a combination of fragments from all logs that each monitoring tool recorded. IXP watch filters these logs and creates an output containing only information of high importance.

### 2.6.1 Syslog-ng

Syslog-ng provides a centralized, securely stored log of all devices on a network, whatever platform they run on. System logs contain a lot of false positives so there is a need to suppress these messages. Syslog-ng was designed to make message filtering much more efficient and filters messages in a content-based manner. In this way only the messages that are of particular importance manage to reach the destination.

### 2.6.2 SNIPS

SNIPS is a network monitoring tool that runs under UNIX. Among other tasks, this tool checks the reachability of different hosts and routes system resources. It is important to note that SNIPS logs an event only if a user's defined threshold value is passed. For example, if the threshold for a round trip time (RTT) value is set to 50 ms, only values bigger then 50 ms will be recorded.

SNIPS also has three threshold indicator levels: warning, error and critical.

1. Warning indicates that one poll is missed or the value passed the first threshold.
2. Error indicates two polls are missed or the value passed the second specified threshold value.
3. Critical indicates three polls are missed or the value passed the third specified threshold value.

Another level called "info level" is present, which provides information about the current status of the devices and whether they are currently up and running properly. It also serves to provide information when a device comes back from the Warning indicator level.

# Chapter 3: Objectives and Scope

## 3.1 Objectives

### 3.1.1 Standardise Data Representation and Format

This objective targets a common problem that IXPs currently face, discussed in the introductory part of this report. Briefly, IXPs have not established or follow any data representation standards when implementing and utilising network monitoring software. As a result, data formats used by one IXP are unique, which implies that its data formats and, consequently, its toolkits cannot be adopted by other IXPs without the need for major changes. The objective was therefore to design a common data representation for network traffic.

### 3.1.2 Standardise Data Display

As IXPs currently deploy their own toolkit for monitoring their network infrastructures, data are represented and visualised in different ways. Due to these presentational differences, IXPs face again yet another hurdle in sharing information effectively. The objective was to overcome these presentational differences by designing a standardised data display which allows data from different IXPs to be presented in the same manner.

### 3.1.3 Develop a Prototype

Another objective was to implement a prototype to provide proof of concept for the project's system architecture. The prototype did not necessarily provide any network monitoring functionality; instead, it had the sole purpose of demonstrating that the system architecture was feasible. The prototype served as a stepping stone before beginning the actual implementation of the system.

### 3.1.4 Create a Toolkit for Data Analysis

As outlined before, IXPs currently have developed their own set of network monitoring toolkits which are unlikely to be compatible with other IXPs' hardware. The objective was therefore to create a toolkit for data analysis that could be used by all IXPs, alleviating the problem of information sharing.

### 3.1.5 Integration at LINX

Because this project was suggested by LINX and because the desired outcome was an implementation of a system providing network monitoring functionality, it was essential to integrate the developed system at our client's location. Testing the system there provided feedback on whether or not it could operate in an IXP environment.

### 3.1.6 Perform Client Acceptance Testing

Since this project has a real client, a vital part of measuring its success was performing an acceptance test to see if the system developed met the client's expectations.

### 3.1.7 Meet the Client's System Requirements

Since this is a project that could potentially be used by many IXPs, and since each IXP has, generally, quite different hardware and software platforms, it was an important objective to create a toolkit that could be deployed with little or no change to the IXP's current infrastructure.

## 3.2 Scope

Since this project was suggested by LINX and being undertaken on its behalf, the project concentrated on the requirements of this client. The use and testing of the toolkit by other IXPs was not considered due to time constraints and lack of access to the relevant resources. However, upon completing the implementation phase relevant documentation was created that gives detailed instructions to other interested IXPs on how to deploy the toolkit in their networks.

# Chapter 4: Requirements Analysis

## 4.1 Overview

In order to clearly understand the requirements, it is important to illustrate the problems LINX and other IXPs had identified. A short summary is given below.

- Each IXP/ISP has different hardware supporting different monitoring tools.
- Different IXPs retrieve, store, analyze and display traffic data in different ways. They have their in-house built tools that cannot be used by other IXPs.
- Storing the data depends on the method of retrieving them. Many IXPs use SNMP and MySQL, while others use SNIPS and RRDs.
- A traffic analysis tool developed by one IXP cannot be used by another one because their data are incompatible and the data analysis techniques used are likely to be different.
- There is no standard on what method should be used for displaying the data. LINX, for instance, uses HTML pages which follow the SNIPS design and standard of displaying the data.

## 4.2 Functionality Requirements

Initial requirements were set out during the presentation John Souter gave as part of the DCNDS seminars. One of the key requirements outlined during this presentation was the need for a rudimentary network monitoring toolkit that could be used by multiple Internet Exchange Points. Since it was stated that IXPs would not want to change their existing data representation formats, the initial step suggested was to design a completely new system architecture on which the suite of monitoring functions could be based.

An additional number of requirements for such a network monitoring toolkit were set out by LINX during the first meeting at the client's location. Monitoring of an IXP's network infrastructure was required to take place from any location, suggesting a client-server architecture. Another requirement of the toolkit was to allow remote monitoring of other member IXPs. This feature would allow an IXP such as LINX to examine the network performance of other IXPs and enable LINX to investigate and suggest any solutions to problems the remotely monitored IXP might

encounter. The monitoring toolkit was also required to be easily extensible in its functionality, meaning that it should be possible to add or remove monitoring components at later stages. This requirement suggested a system consisting of small modular components that could be easily plugged in or easily changed without affecting other modules. Another requirement for the toolkit was the ability to monitor the network in real time.

In terms of the actual monitoring functionality of the toolkit, the requirements were not as clearly defined as some of the general requirements and it was therefore suggested, as a guide, to research the functionality of existing monitoring tools such as IXP-watch. However, it was made apparent that monitoring to see if a device or a link is up and testing whether round trip times exceed a certain threshold value were some of the necessary features.

## 4.3 Data Display Requirements

Before this project, the network monitoring solution utilised by LINX did not provide a very powerful package for the graphical visualisation of data. Most of the data such as round-trip times were displayed in text format while throughput was visualised though a simple graphing component which did not provide many features. LINX stated the need for a graphical user interface from which the user could select what kind of data to display. The graphs displayed by the graphical user interface were required to be standardised so that the visualisation of data from different IXPs was the same. Such a GUI along with its standardised graphs would ensure that network administrators could understand displayed data very easily even though it might have come from a variety of IXPs.

In addition, since the toolkit was required to allow for the remote monitoring of other IXPs, the graphical user interface had to provide a mechanism to display data from different IXPs side by side. This would aid and simplify the comparison of data from a variety of IXPs.

Because LINX has to monitor many machines on their infrastructure one of the requirements was the ability for a user to define certain parameters. These parameters included defining whether all interfaces or just one should be monitored and defining a specific monitoring interval for real-time tools, meaning that the user would be able to specify the age of the data to be displayed (for instance, display data that are at most five minutes old). In addition, the need for a prioritisation mechanism

was mentioned during the requirements capture meeting. This mechanism would list and display the data of any machines in critical conditions before the less critical ones, which would be displayed further down the list.

Finally, as Internet Exchange Points tend to collaborate with each another, it was suggested that a particular nice feature would be the ability to save the data of a graph to a file in order to share it with an interested party. More concretely, it was suggested to implement a save and open menu bar that would allow a user to save the displayed data in a certain standard file format. This file could then be forwarded to another IXP by means of email or another mechanism and then be opened and displayed again for analysis.

## 4.4 Security Requirements

As the data LINX generates from monitoring their customers' traffic and their own hardware have to be kept confidential, there was a requirement to ensure its confidentiality through encryption as well as to provide mutual authentication of the parties involved in the exchange of these data.

# Chapter 5: System Architecture

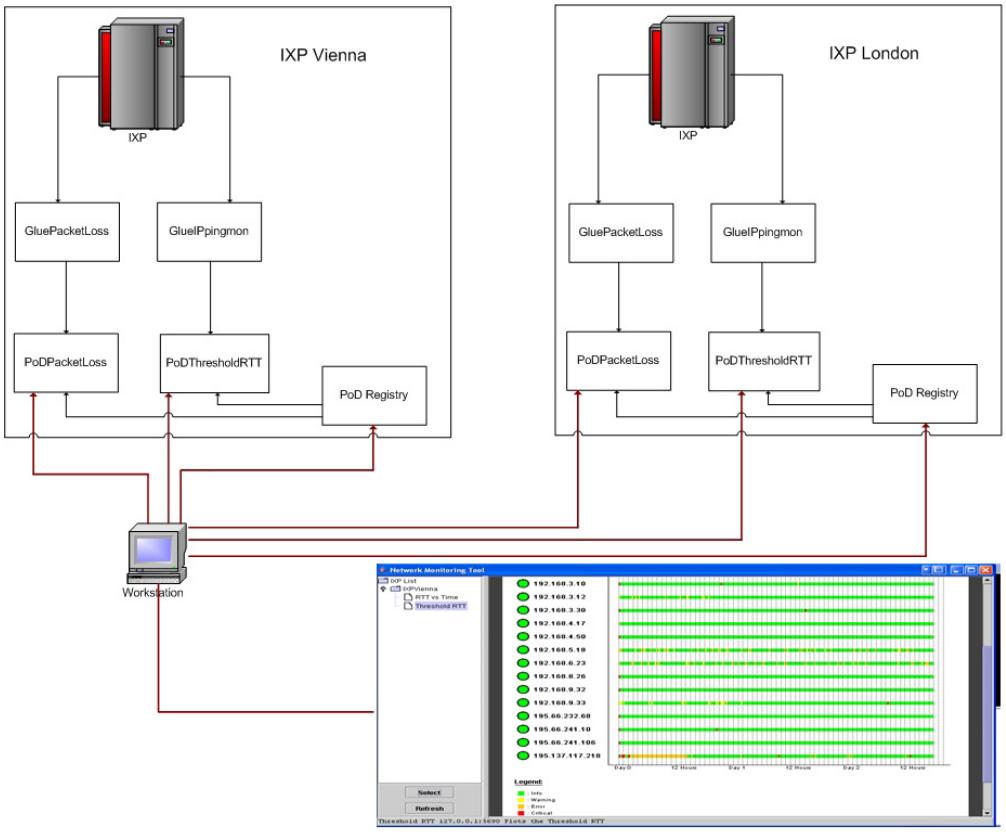



**FIGURE 5.1: System architecture**

The system architecture consists of several components. The process begins when a real resource such as a switch or router generates data (perhaps in the form of log files); these are shown in Figure 5.1 as grey boxes and are labelled IXP. The next component, called a Glue, then gathers these data and puts them in a standard format. In the case of a log file, a Glue simply reads each line, parses the fields in the line and stores them. The storing of the data, which are now in a standard format, can be done using another file, a data structure or even a database. Thus this component provides the "glue" between the real resource and the rest of the system.

The next component, called a PoD, performs two functions: it analyses the data supplied by the Glue and acts a server that provides the analysed data to clients (each request from clients is handled by spawning a separate thread). The final component consists of a Graphical User Interface (GUI), essentially the client. It is in charge of connecting to a PoD, requesting the data and displaying the results

graphically in a standard way. In addition, the GUI provides a list of all available PoDs that the user can access information from.

Each set of real resource, Glue, PoD and GUI represents a Remote Monitoring Function (RMF) or tool. The concept is shown in Figure 5.1, where the IXP on the left contains two different tools, one focusing on packet loss and the other one on round trip-times for pings. Since the tools are disparate, each component in them performs a different function than its counterpart. For instance, the PoD in the first tool will be basing its analysis on how many packets have been lost, while the PoD in the second tool will look at how long the round-trip times are. They both still perform analysis and act as servers, but their specific tasks depend largely on the type of data they are dealing with. Figure 5.2 shows a closer look at the architecture of a particular Remote Monitoring Function. Note how all communication between components (except between the Glue and the real resource, G-RP) is standard. Also, the Glue is logically divided into two parts, the resource-specific part (G-RS) that deals with data in a non-standard format and the resource-independent part (G-RI) that communicates with the PoD:

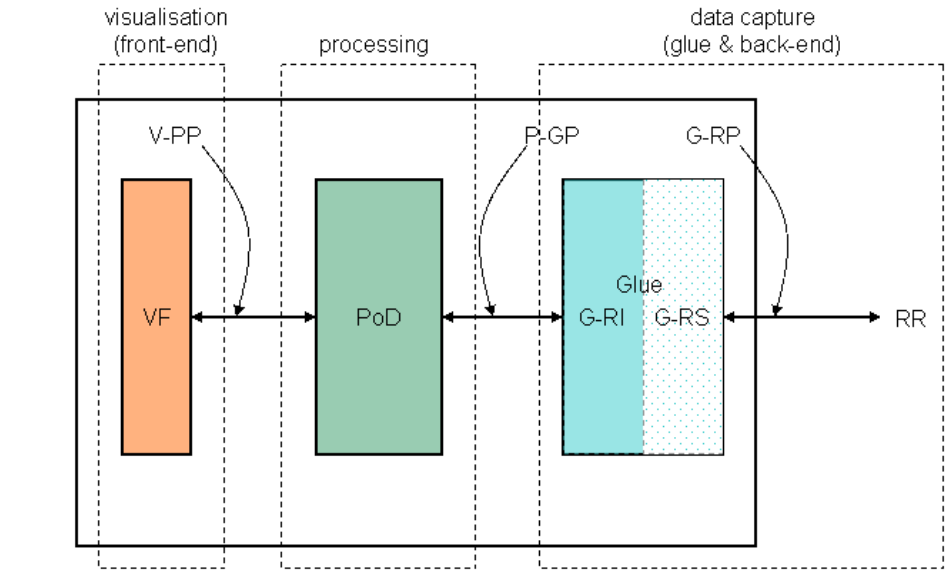



**FIGURE 5.2: Architecture of a Remote Monitoring Function**

The system also needs to have some mechanism whereby the GUI can find the PoDs. To solve this problem, a PoDRegistry was designed. Each IXP has its own PoDRegistry, which is initially empty. When a PoD begins running, but before it starts to accept connections from clients, it registers itself with the PoDRegistry, telling the registry its name, its address, its port number, what type of PoD it is (see Section 7.2) and a short description of what it does. The PoD knows where to contact the registry because the registry runs at well-known port 5555 and at a well-known address determined individually by each IXP. Once all PoDs are registered with their local registries, it is up to the GUI to contact these registries and to request all entries from them. The location of all registries is done through an out of bounds mechanism, perhaps using a URL such as podregistry.ixpname.com and of course well-known address 5555. In the final step the GUI displays all the PoDs it found and the user selects those he or she wishes to see data from.

Since the data being transmitted are likely to be confidential, according to one of the client's requirements all communication between a PoD and a GUI is done using SSL and mutual authentication (see Chapter 9). Communication between a PoD and the PoDRegistry and again between the GUI and the PoDRegistry is unencrypted, since the location of a PoD is not confidential.

One of the great advantages of this architecture is that it allows a tool developed by an IXP to be easily ported over to another IXP. If the real resources in each IXP differ, the second IXP need only rewrite the G-RS: the new G-RS will be capable of talking to the new real resource and put the obtained data in the same standard format as the one being used by the G-RS in the first IXP. In this way, IXPs can share tools they develop, and, as a result of this collaboration, significantly improve their network monitoring capabilities.

Another important advantage is the system's modularity: since all communication between components is performed in a standard way (either through protocols as described in chapter 8 or through standard data formats) each component (Glue, PoD, GUI, even the PoDRegistry) can be written individually from the others and in different programming languages. Thus if while porting a tool an IXP discovers that it is unsatisfied with the way the other IXP implemented a particular component, it can replace it with its own without having to change any of the other components. This also allows for easy upgrades, since upgrading a component has no

effect on the others, provided that the new component still communicates using the same protocols and standards as the previous version.

A final advantage is the fact that the graphical display of the data is done in a standard fashion, allowing an IXP to quickly compare data from its network to data from other IXPs' networks by displaying graphs side by side. This could potentially be used to roughly predict when a fault will occur: if an IXP saved a graph right before an error occurred, another IXP could use it as a signature for identifying a problem that is about to take place.

# Chapter 6: Remote Monitoring Functions

## 6.1 RTTvsTime

### 6.1.1 Introduction

RTTvsTime began its life as the prototype described in Section 3.1.1, eventually developing into a full RMF. RTTvsTime consists of a simple tool that monitors and graphs the round trip-time for a given interface. This is done in non-real time, that is, the RMF retrieves all data for that interface regardless of how stale that information is. In addition, the PoD performs no real analysis, it simply allows clients to connect to it and retrieve the data. The reason for creating such a simple RMF was to show proof of concept for the system's architecture. Even still, if the data are relatively up to date, the RMF provides a useful graph depicting the fluctuation of round-trip time over time. All components of this RMF are implemented in Java (the names of the actual class files appear on top of the components in Figure 6.1).

Figure 6.1 gives a detailed description of the RMF and in particular the different standard data formats used. For instance, the standard output from the Glue is shown: each entry is an array of strings containing the date, the time, the IP address and the round-trip time. While communication between this Glue and the PoD in the current implementation is done through Java's `Vector` object, it could just as easily be done over a network or even a file. In both cases the same standard data format would have to be used in order to comply with the system's architecture. In the latter case, each line in the file would contain one entry, separating the entry's fields by commas; in the former case, an array of bytes containing the different arrays of strings would be sent using a protocol similar to the VP protocol described in Section 8.3. Such a protocol is considered future work (see Chapter 13). Note that all the RMFs in this chapter contain a diagram similar to that found in Figure 6.1 and that the explanation just provided applies to these as well.

Finally, the graphics display is also standardised, in this case by using Ujac's charting library.

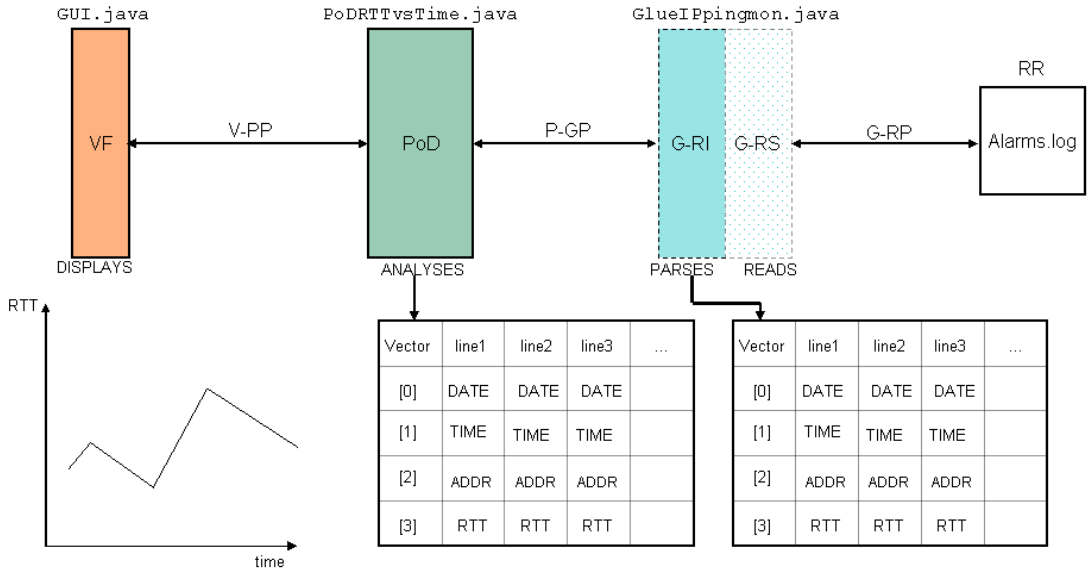



**FIGURE 6.1: RMF RTT vs. Time**

## 6.1.2 Glue and Backend

The real resource as depicted in Figure 6.1 consists of a log file named `alarms.log`. This file is created by IXPwatch, a tool developed by LINX. A typical entry in this log file looks as follows:

```
2004-06-07 00:08:13 :  : ippingmongw : tr2.tfm7 : ICMP-RTT : down :
Warning/infrastructure : tr2.tfm7 (195.66.232.62) ICMP-RTT 83 down
```

Since the file contains entries for all types of events, the Glue reads in only those that contain the literals “ICMP-RTT” and “ippingmongw”; in other words, the Glue retrieves only round-trip time events generated using the Internet Control Message Protocol. As a result the Glue / Resource Protocol (G-RP) depicted in Figure 6.1 is in this case the Java file libraries used to read `alarms.log`. Once a line is read, the Glue begins populating a `Data` object (shown in Figure 6.2), which is essentially a `Vector`.

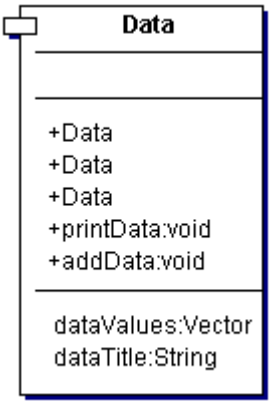



**FIGURE 6.2: The Data object**

Each element in the `Vector` consists of an array of Strings representing the fields in one line of the `alarms.log` file. For instance, the first element in the `Vector` is an array of Strings for the first line in the log file which contained the two literals desired, and the first element in the array is the date on which the event was logged. Figure 6.1 shows that each array of Strings contains in fact four elements: the date, the time, the interface's IP address and the measured round-trip time.

### 6.1.3 PoD

The PoD begins by retrieving the `Data` object from the Glue. Thus, the PoD / Glue protocol (P-GP) shown in Figure 6.1 is represented by this object. Since, as previously mentioned, the PoD performs no real analysis, the PoD does not change any of the data given to it by the Glue (again, see Figure 6.1). It simply services requests from clients for this data.

### 6.1.4 GUI

While the GUI implemented for this project can be used as part of RMFs of different types, this discussion will be limited to the display of the RTT vs. Time RMF. First, the GUI contacts the PoDRegistry for the desired IXP and retrieves the information for the PoD. This information consists of an object of type `PoDInformation`, as shown below:

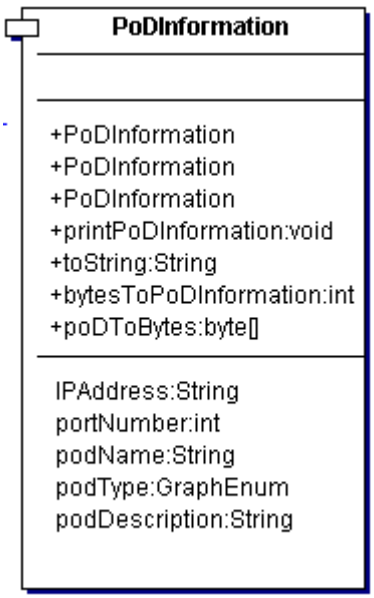



**FIGURE 6.3: The PoDInformation object**

The GUI uses the `podType` to determine what type of graph to display (type `GraphEnum` is a simple enumeration class for types of graph; see Appendix B, class diagrams, for more details). In the case of this RMF the graph type is "NORMAL". Next the GUI displays the PoD's information, the user clicks on the PoD and hits the Select button. This causes the GUI to display a dialog box asking the user to specify an IP address for the interface to obtain data for. Once the user has selected an address, the GUI contacts the PoD and retrieves the `Data` object. Note that this communication uses SSL layered on top of the VPprotocol described in section 8.3. Once the object has been received, the GUI uses Ujac's charting library to display the round-trip times over time. Figure 6.4 shows a screen capture of a graph being displayed; also, note on the left hand side the fact that the RTT vs. Time PoD has been selected.

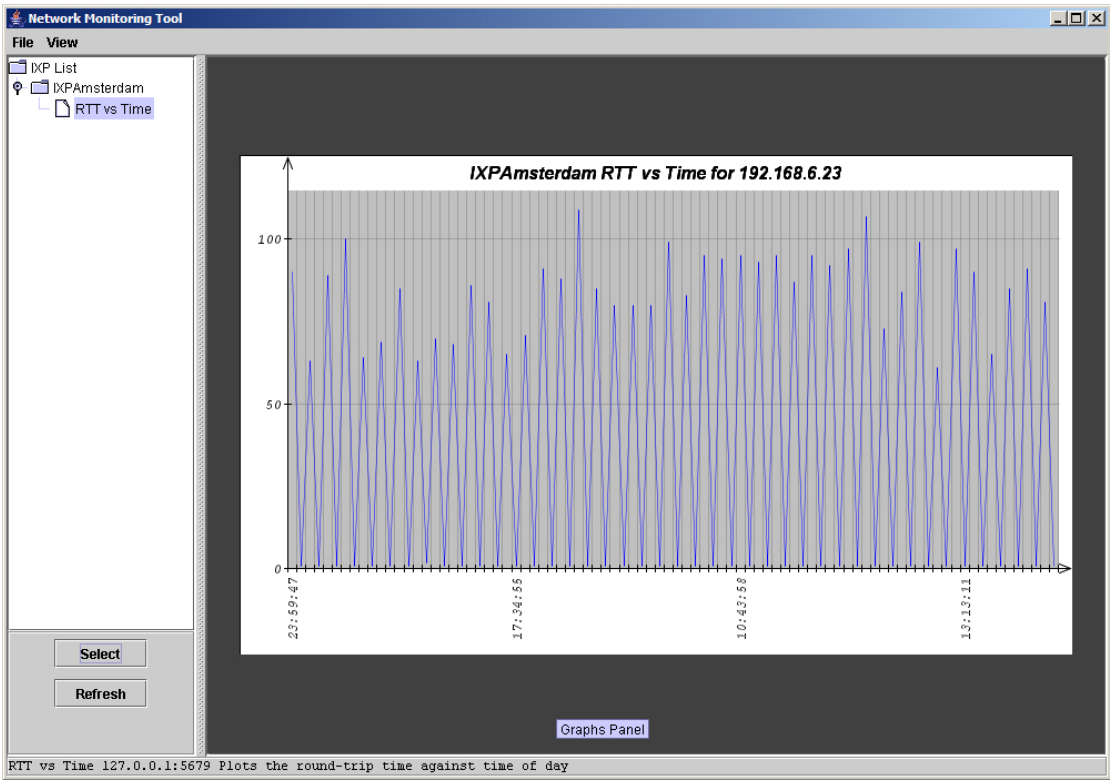



**FIGURE 6.4: Screen capture of the RTTvsTime RMF**

## 6.2 ThresholdRTT

### 6.2.1 Introduction

While the previous RMF graphed round-trip times, this tool bases its analysis on thresholds. The concept for this tool arose from output created by IXPwatch (the `alarms.log` file) and a suggestion by the client that this tool could come in handy when monitoring the network's health. A threshold is simply a level that separates two status zones. As far as this RMF is concerned, three thresholds exist, making round-trip times fall into one of four categories: info, warning, error and critical. Thus the first threshold separates the status info from the status warning, the second one separates the status warning from the status error, and so on. The PoD takes each round-trip time and assigns a status to it, allowing the GUI to later graph this status in a standard format using one of four colours: green for info, yellow for warning, orange for error and red for critical. This RMF is non-real time and all its components are implemented in Java (the names of the actual class files appear on top of the components in Figure 6.5). Also, should the user wish to query all interfaces, he or she can input "all" at the relevant prompt from the GUI.

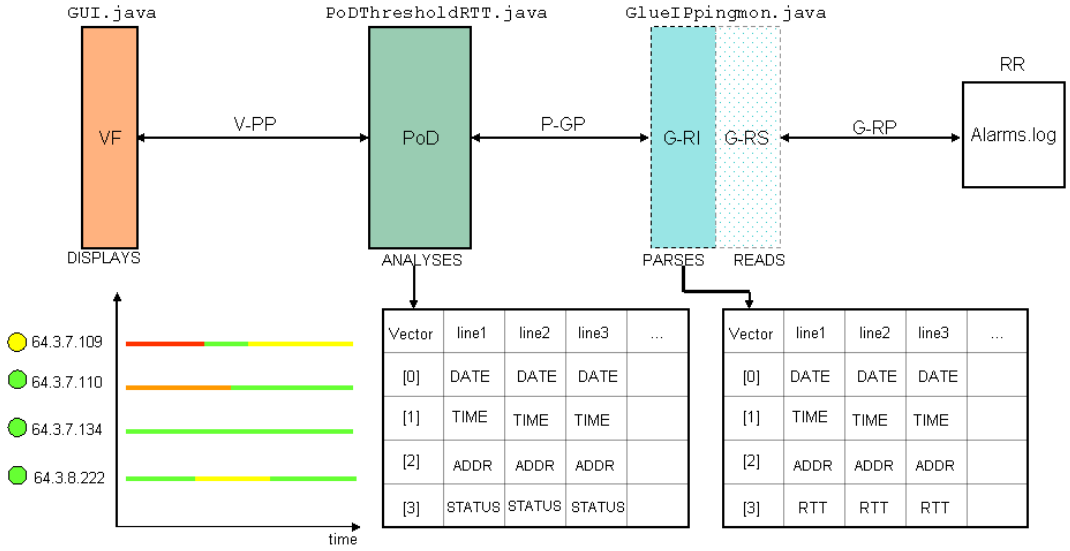



**FIGURE 6.5: RMF ThresholdRTT**

### 6.2.2 Glue and Backend

Since the data for this tool also come from the `alarms.log` file, the Glue for this RMF is in fact exactly the same as the Glue for the previous RMF: it simply reads

the file, filters out the lines it does not care for and stores the parsed fields in a `Data` object.

### 6.2.3 PoD

For each given round-trip time the PoD must figure out its status. It does so by using the three thresholds that divide the four status zones. Each threshold is configurable, but in the case of `alarms.log` the thresholds used by LINX were 60, 80 and 150 milliseconds. Thus, for example, a round-trip time of 65 ms. would be deemed warning since it falls between the first and second thresholds, while one of 55 ms. would be considered info since it is below the first threshold. The Data object returned by the PoD is then exactly the same as the one given by the Glue, except that the last field of each array of Strings (the fourth element) now contains the status as opposed to the round-trip time (see Figure 6.5). The status can be one of the following literals: “INFO”, “WARNING”, “ERROR” or “CRITICAL”.

### 6.2.4 GUI

The display for this tool varies significantly from that of the RTTvsTime RMF, so much so that it was discovered that Ujac’s charting library was not flexible enough to accommodate its data. Consequently, new graphics capabilities had to be built from scratch using Java’s 2D graphics libraries. A description of this standard display follows. Essentially, the display is a horizontal bar graph, where each bar corresponds to a particular interface. Each bar is composed of segments of different colours, each of which represents the amount of time that the interface remained in that particular state (the x-axis represents time). In addition, the display shows a coloured circle next to each interface’s IP address, signifying its current state. Finally, all the interfaces are sorted by current status, listing those that are critical at the top, followed by those in the error state and so on. Figure 6.6 shows a screen capture of such a graph being displayed. Note that the first segment for all interfaces but one is grey. This is because before drawing the graph, the graphics package that plots it determines the leftmost and rightmost (oldest and newest) event from *all* interfaces. Since only one interface had the oldest event, all others display a grey segment (no data available) from the leftmost point on the graph. This type of graph allows a system administrator to very quickly determine which interfaces require immediate attention: those in the critical state will fill in the first rows of the graph.

Finally, should the administrator wish to focus on a particular interface, this RMF allows him or her to display it on its own. This graph, as well as the ones for ThresholdRTTRealTime, LinkStatus and MinMaxPing (see sections 6.3.4, 6.4.5 and 6.5.5, respectively), was designed according to some of the guidelines found in the book "The Visual Display of Quantitative Information" by R.E. Tufte. In particular, it aims to avoid graphical distortion by providing a clear labelling of the different interfaces and a legend explaining the meaning of the colours being displayed. Another guideline is the presentation of many numbers in a small space, which ThresholdRTT certainly accomplishes since it presents a vast amount of data in usually just one screen full. Further, Tufte calls for a graph that encourages the eye to compare different pieces of data. This graph does so by not only presenting a history of all interfaces, but by drawing vertical lines representing what the state of the network was at a specific point in time. A final guideline is to have the graph reveal the data at several levels of detail, from a broad overview to a fine structure. This graph provides a broad overview by displaying the circles that signify the current state of the interfaces; conversely, it presents a finer structure by allowing its user to identify the state of any interface at any point in time.

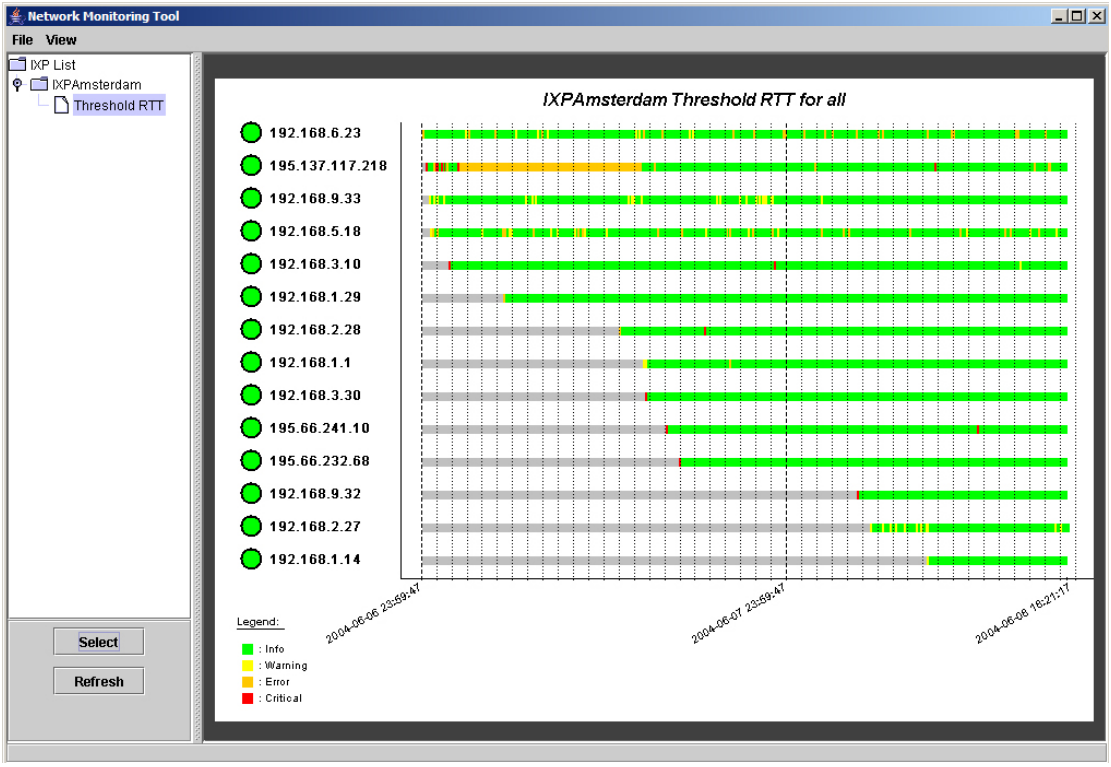



**FIGURE 6.6: Screen capture of the ThresholdRTT RMF**

## 6.3 ThresholdRTTRealTime

### 6.3.1 Introduction

As useful as the previous RMF would be to a system administrator, it would be even more useful if it was capable of displaying the same type of information in real time, updating itself at regular intervals with the latest events logged in the `alarms.log` file. This is precisely the aim of ThresholdRTTRealTime.

While the type of information handled and eventually displayed is the same as the ThresholdRTT tool, the fact that this is a real-time RMF mandated changes from the backend all the way to the GUI. In particular, the PoD (and consequently the Glue) must be able to return all data that is newer than a given amount of time. For instance, the PoD could return all events that took place in the last five minutes. All components of this RMF are written in Java (the actual class names appear at the top of each component in Figure 6.7). Also, should the user wish to query all interfaces, he or she can input "all" at the relevant prompt from the GUI.

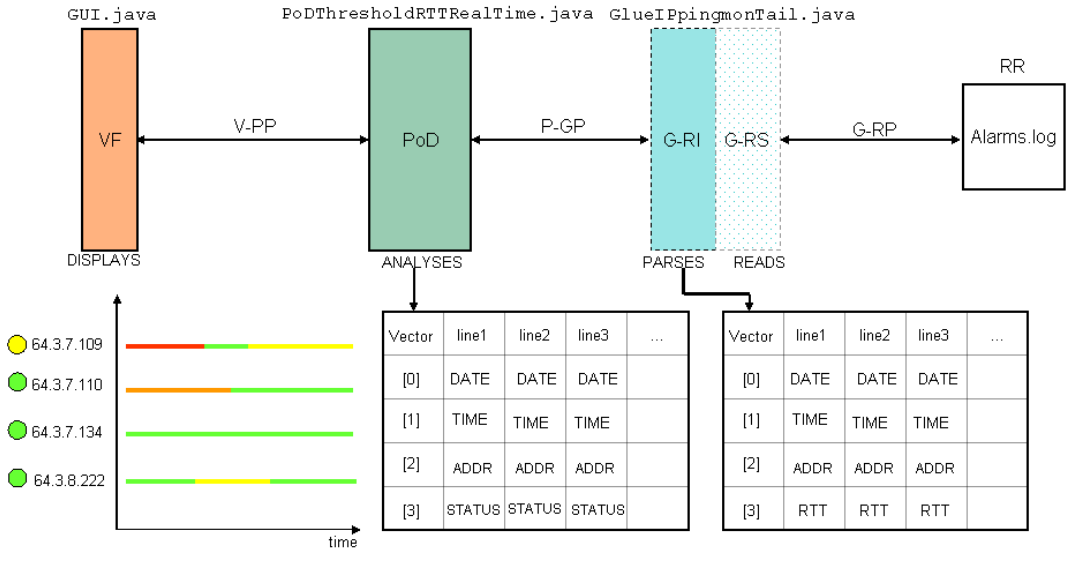



**FIGURE 6.7: RMF ThresholdRTTRealTime**

### 6.3.2 Glue and Backend

The backend is again the log file `alarms.log`. Since the Glue needs to retrieve the most recent events, it has to be able to read the file backwards, that is, read the last line first, the second to last line next, and so on (log files generally store the most recent events at their end). To do so, the Glue makes use of the auxiliary

class `TailFileReader`, and more specifically its `readUnixPreviousLine()` and `readWindowsPreviousLine()` functions. As is probably apparent, this Glue can handle both UNIX files which use only the line feed character (`\n`) to separate lines and Windows files which use the carriage return followed by the line feed character (`\r\n`) to separate lines. In addition, the Glue makes use of another auxiliary class, `Timer`, to determine when it has read enough lines. It does so by first taking the difference between the newest time (the timestamp for the last line in the file that contained the literals "ICMP-RTT" and "ippingmongw") and the current time (the timestamp of the line the Glue is currently processing); if this difference is greater than the time submitted by the user to the GUI (and subsequently to the PoD and the Glue) then the Glue stops reading.

### 6.3.3 PoD

The PoD for this tool is in fact exactly the same as the PoD for the ThresholdRTT RMF with one minor difference: it supplies the Glue with how much data it should read (in terms of time).

### 6.3.4 GUI

Because it is a real-time tool, when the user selects the PoD for this RMF the GUI will prompt him or her to input not only the IP address to display information for (possibly all of them) but also how recent the data to be displayed should be (in seconds). Thus if the user wanted to retrieve data for all the interfaces in the last five minutes he or she would type "all" in the first dialog box and 300 in the second one. Finally, the user must tell the GUI how often to refresh the information, again in terms of seconds. If the user types 5, then every 5 seconds the GUI will return data for all the interfaces that are no older than five minutes and display them on the bar graph described in section 6.2.4. This provides a very powerful tool for a system administrator: at a glance, he or she can see exactly which interfaces require attention, follow how their status develop and also see a brief history of the status of these interfaces. Note that no screen capture is provided since it would look nearly the same as the one provided for RMF ThresholdRTT. For the justification of the graph's design please see section 6.2.4.

## 6.4 LinkStatus

### 6.4.1 Introduction

This tool is based around the program `fping`, a more powerful version of the `ping` program, and arose again from a client requirement of being able to, at a glance, see the status of relevant interfaces. The administrator sets up a configuration file with those interfaces he or she is interested in testing as well as thresholds for determining when a round-trip time should require further attention. Since this is a real-time tool, the RMF then periodically fpings the desired interfaces and displays the results graphically. All components of this RMF are written in Java except for the resource-specific part of the Glue (see G-RS in Figure 6.8) which is written in C++. The reason for this was to show proof of concept that the different components of an RMF could be written in different languages. Also, should the user wish to query all interfaces, he or she can input "all" at the relevant prompt from the GUI.

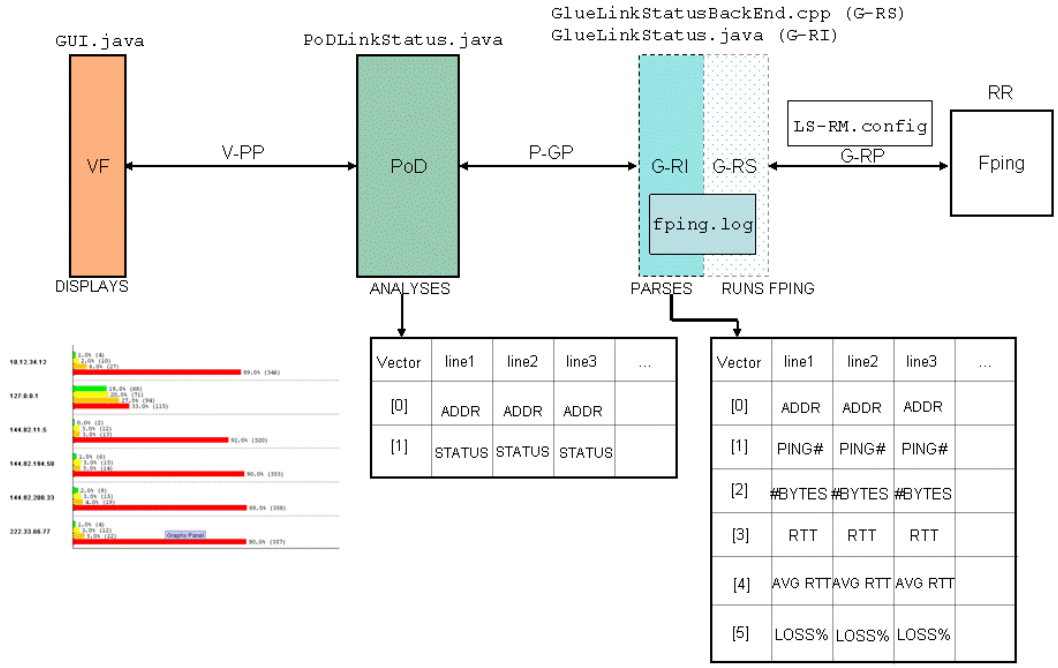



**FIGURE 6.8: RMF LinkStatus**

### 6.4.2 Configuration File

Both parts of the Glue make use of a configuration file named `LS-RM.config` (standing for Link Status – Remote Monitoring). A typical file looks as follows:

```
0.0.0.0   I_D_10 I_L_5 W_D_20 W_L_10 E_D_30 E_L_15 C_D_40 C_L_20 N_2 T_100 R_10000
127.0.0.1 I_D_30 I_L_33 W_D_50 W_L_44 E_D_70 E_L_55 C_D_90 C_L_66 N_3 T_200
```

```
144.82.194.58
144.82.200.33
144.82.11.5
222.33.66.77
10.12.34.12
```

The first line must contain the address 0.0.0.0 to denote that this is the line for default values. If any of the lines that follow is missing any value, the value from the first line will be used. Most of the fields are used by the PoD to perform its analysis, so an explanation of them is discussed in section 6.4.4. However, here is a brief description of their semantics:

| Field | Description |
|---|---|
| `I_D` | "Info" threshold for the delay (the round-trip time). |
| `I_L` | "Info" threshold for the loss (the amount of ping packets lost due to timeouts). |
| `W_D` | "Warning" threshold for the delay. |
| `W_L` | "Warning" threshold for the loss. |
| `E_D` | "Error" threshold for the delay. |
| `E_L` | "Error" threshold for the loss. |
| `C_D` | "Critical" threshold for the delay. |
| `C_L` | "Critical" threshold for the loss. |
| `N` | The number of pings to send to this particular interface. |
| `T` | The timeout value for the ping to this particular interface. |
| `R` | Refresh time, the amount of time (in milliseconds) that the program should sleep for after having pinged all interfaces in the configuration file and before pinging them again (this value is only allowed in the first line of the file). |

**TABLE 6.1: LS-RM.config fields**

Thus in the example configuration file given above, all interfaces except 127.0.0.1 will be pinged twice (`N_2`) and their timeout will be set at 100 milliseconds (`T_100`). Likewise, all these interfaces will share the same thresholds, except 127.0.0.1, for which the default values are overridden. The program sends pings to all the interfaces and goes to sleep for 10 seconds (`R_10000`) before pinging all interfaces again.

### 6.4.3 Glue and Backend

As previously mentioned, the Glue, unlike those in the previous RMFs, is physically split into two parts, following the logical partition shown in Figure 6.8. This discussion will first focus on the resource-specific part of the Glue (G-RS). The real resource in this case is not a log file but rather the `fping` program used to obtain round-trip time (delay) and packet loss information. The G-RS runs the fping

commands and stores the results in a file called `fping.log`. Here are some typical entries from such a file:

```
144.82.194.58 : [0], 84 bytes, 22 ms (0.06 avg, 21% loss)
144.82.194.58 : [1], 84 bytes, 0.05 ms (0.05 avg, 0% loss)
127.0.0.1 : [0], 84 bytes, 32 ms (0.06 avg, 45% loss)
127.0.0.1 : [1], 84 bytes, 0.04 ms (0.05 avg, 0% loss)
```

The first field denotes the interface that was pinged, the second field is the ping number (since each interface can be pinged more than once), the third field is the number of bytes in the ping packet, the fourth field is the round-trip time, the fifth field is the average round-trip time and the sixth field is the percentage loss. Note that some of the actual values have been altered by hand for testing purposes.

It is now up to the resource-independent part of the Glue (G-RI, see Figure 6.8) to read and parse the fields of the file `fping.log`. For each line it creates an array of strings of size 6, as shown in Figure 6.8. In addition, the G-RI provides a function to read the configuration file that is used by the PoD to retrieve the threshold values (8 in total, see table 6.1) for each interface.

### 6.4.4 PoD

The PoD calculates the status (either “info”, “warning”, “error” or “critical”) for each ping result (each line in the `fping.log` file). It does so by first matching the IP address of the result it is currently processing with an entry in the configuration file. When it finds a match, it retrieves all thresholds for that particular interface. It then uses these values along with the round-trip time (delay) and the percentage loss for the ping result to determine the status in the following way:

- If both the delay and loss values are below the “info” thresholds, then the status is “info”.
- If the previous statement fails and the delay and loss values are below the “warning” thresholds, then the status is “warning”.
- If the previous statement fails and the delay and loss values are below the “error” thresholds, then the status is “error”.
- Finally, if the previous statement fails then the status is “critical”.

For instance, if the log file contained the fping result

```
144.82.194.58 : [0], 84 bytes, 22 ms (0.06 avg, 21% loss)
```

and the configuration file contained the entry

```
144.82.194.58 I_D_30 I_L_33 W_D_50 W_L_44 E_D_70 E_L_55 C_D_90 C_L_66 N_3 T_200
```

then the status would be “info”, since both values (22 ms and 21%) fall under the “info” thresholds. If the value for the delay had been, for instance, 72 ms, then the status would have been “critical”, since the value is above the “error” threshold of 70.

### 6.4.5 GUI

The PoD gives the GUI a set of pairs of the form < IP address, status>. The GUI begins by grouping all the different ping results by IP address. It then displays the information using a horizontal bar graph, where each interface is allotted, in fact, four bars, one for each type of status. Thus, the number of “info” results is plotted as a horizontal green bar, the number of “warning” results as a yellow bar, the number of “error” results as an orange bar and the number of “critical” results as a red bar. In addition, the actual number of results is printed at the end of each bar, along with a percentage relative to the other types of results for the same interface. For instance, if an interface had two results in each of the four categories, each bar would display 25% (two divided by eight) followed by the number two. Figure 6.9 shows a screen capture of the GUI displaying a graph for this RMF. Like the Threshold graphs, this one follows the Tufte guidelines discussed in section 6.2.4. More specifically, LinkStatus gives large amounts of data (many round-trip time results) in a very small space, usually less than a screen full. In addition, it encourages the eye to compare different pieces of data by using a different colour for each status; in this way, the user can easily compare, for instance, the number of pings in the “error” level for all interfaces (the same could be done, of course, for all the other types of status). Finally, the graph presents different levels of detail, from the broad overview of the colours and the length of the bars to the more detailed percentage and number of pings found at the end of the bars.

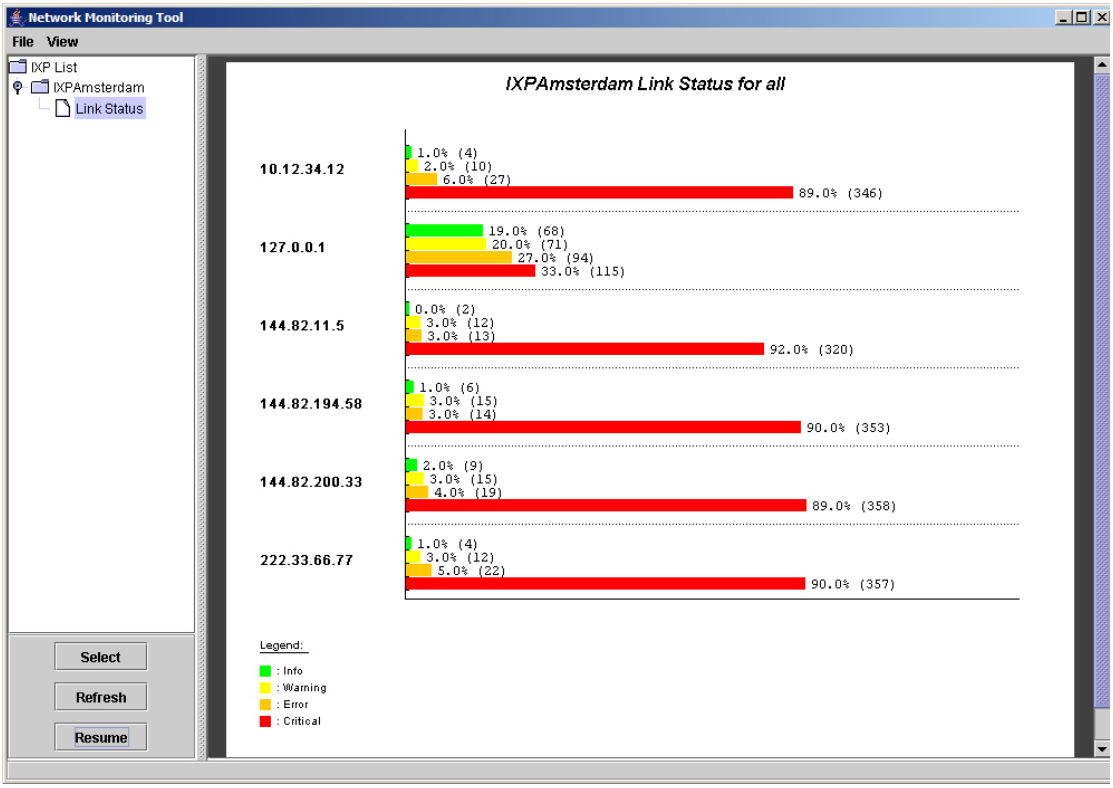



**FIGURE 6.9: Screen capture of the LinkStatus RMF**

Live the previous real-time RMF, this tool asks the user to supply how often to refresh the information. The user also has the option to display information from a single interface or from all interfaces in the log file. The strongpoint of this tool is that it condenses quite a bit of information (8 thresholds, a delay value and a loss percentage value) into a single coloured segment on a graph, allowing a system administrator to quickly gauge which interfaces require further attention.

## 6.5 MinMaxPing

### 6.5.1 Introduction

Like LinkStatus, this RMF is based on fping. Its purpose is to use the results from ping tests to derive statistical measures such as average, minimum and maximum round-trip times, maximum outliers and standard deviation. MinMaxPing is very useful because it gives the user the range of RTT values: if these values are very different they will result in a fluctuation in the ranges plotted in the graph, alerting the administrator that there might be a network problem or that abnormal packet processing at the destination host is taking place.

The process begins when the network administrator sets up a configuration file specifying which interfaces to ping as well as the timeout value and the number of pings to perform for each interface. MinMaxPing is a real-time RMF, so the user of the GUI specifies how often to refresh the information being displayed. All components of this tool are written in Java except for the resource-specific part of the Glue, which is written in C++. Also, should the user wish to query all interfaces, he or she can input "all" at the relevant prompt from the GUI.

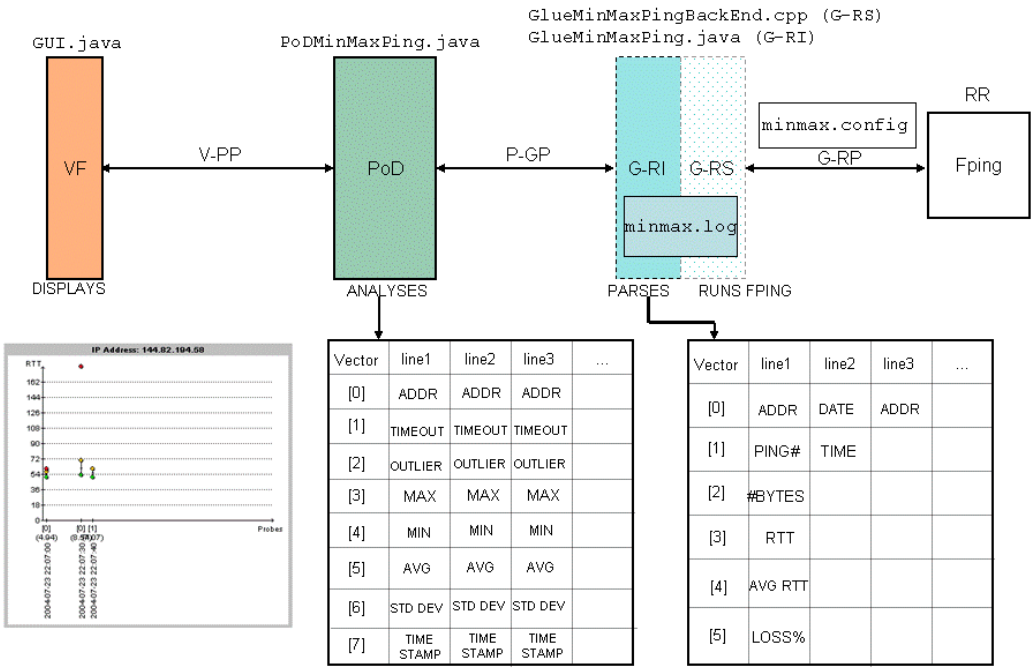



**FIGURE 6.10: RMF MinMaxPing**

## 6.5.2 Configuration File

The configuration file for this tool, called `maxmin.config`, is considerably simpler than the one used in the LinkStatus RMF. A typical file looks like this:

```
0.0.0.0 T_200 N_10
127.0.0.1 T_300 N_8
144.82.194.58
```

As with the previous RMF, the first line of the configuration file represents the defaults and must have 0.0.0.0 as the IP address. The only two values that can be specified are how many times to ping a particular interface (in this case the default is 10) and the timeout (200 milliseconds in the default case). Any subsequent line in the

file can override these defaults, but should a value be missing, the default value is used.

### 6.5.2 Glue and Backend

The real resource in this case is `fping` and the resource-specific part of the Glue is a program written in C++. A probe begins when this program reads the first line in the configuration file and executes the fping command specified therein. Once it has finished executing the pings for all the lines in the configuration file, the G-RS prints out a time stamp containing the current time; this marks the end of the probe. The G-RS then waits a constant amount of time (currently set at 2 seconds, enough time for the timestamp to be printed) before starting the next probe. All results from the pings including the timestamp that marks the end of the probe are outputted to a file called `maxmin.log`. Thus, a typical log file looks as follows:

```
127.0.0.1 : [0], 84 bytes, 62 ms (60 avg, 0% loss)
127.0.0.1 : [1], 84 bytes, 50 ms (50 avg, 0% loss)
144.82.194.58 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
144.82.194.58 : [1], 84 bytes, 57 ms (50 avg, 0% loss)
144.82.194.58 : [2], 84 bytes, 50 ms (50 avg, 0% loss)
2004-07-23 22:07:00
127.0.0.1 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
127.0.0.1 : [1], 84 bytes, 53 ms (50 avg, 0% loss)
127.0.0.1 : [2], 84 bytes, 45 ms (50 avg, 0% loss)
127.0.0.1 : [3], 84 bytes, 50 ms (50 avg, 0% loss)
127.0.0.1 : [4], 84 bytes, 49 ms (40 avg, 0% loss)
144.82.194.58 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
144.82.194.58 : [1], 84 bytes, 180 ms (50 avg, 0% loss)
144.82.194.58 : [2], 84 bytes, 70 ms (50 avg, 0% loss)
144.82.194.58 : [3], 84 bytes, 53 ms (50 avg, 0% loss)
2004-07-23 22:07:30
144.82.194.58 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
144.82.194.58 is unreachable
144.82.194.58 : [2], 84 bytes, 50 ms (50 avg, 0% loss)
127.0.0.2 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
2004-07-23 22:07:40
```

Notice how the timestamps act essentially as delimiters between probes. Also, the reason that the number of pings and even some of the IP addresses change between probes is because the configuration file was changed.

The resource-independent part of the Glue has the task of reading in this log file and parsing all of its entries. The G-RI ignores any incomplete probes, that is, any probes that have not been time stamped yet. In particular, it begins reading from the end of the file, ignoring all lines until it finds a line containing a time stamp; the G-RI determines that it has found a time stamp by searching for a dash (-) character

in the line. For each complete probe, the G-RI parses each line and stores the fields in it in an array of strings. Since the line can have a different number of fields depending on whether it is a regular line, a time stamp or a line representing a timeout, the number of elements in each array of string varies. For a regular line, the array contains 6 elements:

```
[0] The IP address
[1] The ping number for this IP address
[2] The number of bytes for the ping packet
[3] The round-trip time
[4] The average round-trip time
[5] The percentage loss
```

For a line containing a timestamp, the array contains two elements:

```
[0] The date
[1] The time
```

For a line representing a timeout, the array contains a single element:

```
[0] The IP address
```

Finally, since MinMaxPing is a real-time RMF, the Glue receives as input how old the data to be read should be and stops when the timestamp of the next (older) probe is older than the desired time specified by the user.

### 6.5.4 PoD

The PoD for this tool returns arrays of strings of size 8. Each one of these arrays represents statistical values calculated from the pings to one interface in a specific probe. For instance, if the Glue was working with the following log file,

```
127.0.0.1 : [0], 84 bytes, 62 ms (60 avg, 0% loss)
127.0.0.1 : [1], 84 bytes, 50 ms (50 avg, 0% loss)
144.82.194.58 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
144.82.194.58 : [1], 84 bytes, 57 ms (50 avg, 0% loss)
144.82.194.58 : [2], 84 bytes, 50 ms (50 avg, 0% loss)
2004-07-23 22:07:00
127.0.0.1 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
127.0.0.1 : [1], 84 bytes, 53 ms (50 avg, 0% loss)
144.82.194.58 : [0], 84 bytes, 60 ms (60 avg, 0% loss)
144.82.194.58 : [1], 84 bytes, 180 ms (50 avg, 0% loss)
144.82.194.58 : [2], 84 bytes, 70 ms (50 avg, 0% loss)
2004-07-23 22:07:30
```

the PoD would generate four arrays: one for address 127.0.0.1 for the probe time stamped 2004-07-23 22:00:30; another one for address 144.82.194.58 for the same probe; a third one for address 127.0.0.1 for the probe time stamped 2004-07-23

22:07:00; and a final one for address 144.82.194.58 for the same probe (provided that the user has selected to view data that are older than the 30 seconds separating the two probes).

Each of these arrays contains the following 8 elements:

```
[0] The IP address
[1] The number of timeouts for all pings of this IP in this probe
[2] The maximum outlier (if any, more on this later)
[3] The maximum round-trip time
[4] The minimum round-trip time
[5] The average round-trip time
[6] The standard deviation
[7] The time stamp of the probe that this set of pings belongs to
```

The PoD begins its analysis by grouping all the round-trip times from the different pings of an interface in an array of integers. It continues by calculating the percentile in terms of milliseconds, the same units as the round-trip times. Currently the $70^{th}$ percentile is calculated (a constant) since this value was observed to be optimal in terms of drawing the attention of whoever is watching the GUI's graph to problematic interfaces. Any round-trip time values that are higher than the percentile are considered to be outliers. To explain how this percentile is calculated, the following example is presented. Suppose that the array of integers (the set of round-trip values) contains the elements 10, 20, 30, 40 and 50 (the elements in the array must be sorted). The process begins by calculating

```
(N + 1) x percentile    N = number of elements in array values
 5 + 1 x 0.7 = 6 x 0.7 = 4.2
```

Next, the result (4.2) is split into its whole part (w = 4) and its fractional part (f = 0.2). The percentile (in terms of millisecond) can then be computed as follows:

```
(1 – f) x values[w - 1] + f x values[w]
(1 – 0.2) x values[3] + 0.2 x values[4]
0.8 x 40 + 0.2 x 50 = 32 + 10 = 42
```

Thus, in the given example the last entry in the array, 50, is a maximum outlier since it is higher than the calculated value of 42. Note that it is possible, given a different set of data and percentile, for the index to the array (in the above example w) to be higher than the last element; in this case the percentile (in terms of milliseconds) gets set equal to the last element in the array and there are therefore no outliers.

Since outliers tend to skew statistical measures like the average, these are filtered out of the array of integers, but not before making a note of the maximum outlier, which is needed for display by the GUI. Note that it is possible that none of the round-trip times are outliers, in which case clearly there will not be a maximum outlier. Next the PoD proceeds to calculate the maximum, the minimum, the average and the standard deviation of those round-trip values that were not outliers. This final measure is useful since it alerts the user about fluctuations in round-trip times, giving an indication of possible network problems such as congestion. In fact, this measurement is so useful that Mandrake 10, a LINUX distribution, includes it in its implementation of the `ping` program. Finally, the PoD makes a note of how many of the pings for this interface resulted in timeouts.

### 6.5.5 GUI

The GUI begins by calculating and storing all the unique IP addresses in the data given by the PoD. The information for each IP address is displayed in its own separate graph; thus, the GUI displays as many graphs as there are unique IP addresses. The graphs are sorted by increasing IP number.

For each graph, the GUI runs through the set of arrays given by the PoD and retrieves only those arrays whose IP address match the address of the graph. Each of these matching arrays comes from different probes with different time stamps and is consequently graphed as a separate entry on the x-axis (the x-axis represents time). The y-axis represents round-trip time, usually in milliseconds. For each array, then, the GUI plots the minimum, the maximum, the average and the maximum outlier (if any): the minimum value is a green dot, the maximum value is an orange dot, the average is a black cross and the maximum outlier is a red dot. In addition, the GUI displays the time stamp of the probe as the x-axis label along with the standard deviation for the values in the probe (between parentheses) and the number of timeouts (between brackets). Figure 6.11 shows a screen capture of the GUI displaying a graph for this RMF. The graph, again, follows Tufte's guidelines. It presents many data in a small space and encourages the eye to compare different pieces of data (maximums, minimums, averages and maximum outliers)

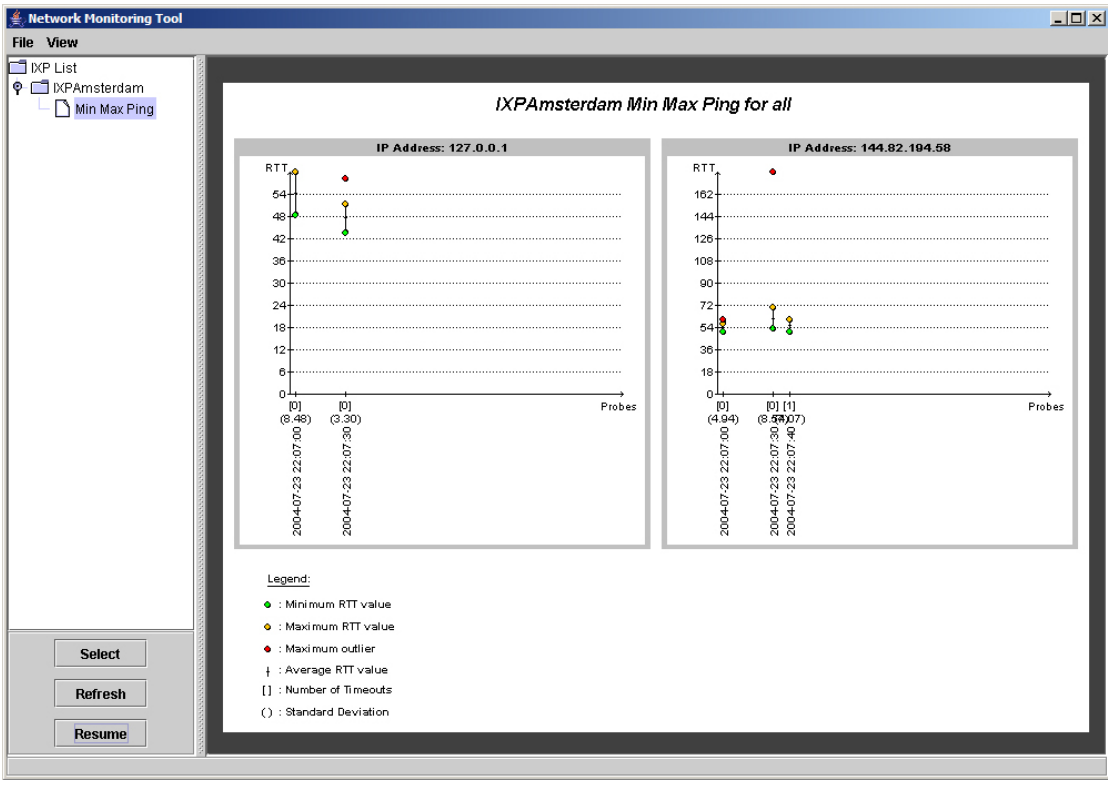



**FIGURE 6.11: Screen capture of the MinMaxPing RMF**

# Chapter 7: Other Components

## 7.1 PoDRegistry

The PoDRegistry offers a mechanism by which PoDs from any of the Remote Monitoring Functions discussed in Chapter 6 can announce their presence; clients can then contact the registry to obtain the PoDs' location. In this way, the registry provides bootstrapping between servers (PoDs) and clients (in this case the GUI). The current implementation of the registry is in Java, but like the other components in the system it could have been written in any other language, provided that the same communication protocol is used (see Section 8.2 for a description of the protocol that the registry uses to communicate).

An entry in the registry consists of a `PoDInformation` object (see Figure 6.3), which contains the PoD's IP address, its port number, its name, a brief description and its type. This last attribute is used by the GUI to determine what type of graph to display. In short, the registry keeps track of all the PoDs it knows about by storing `PoDInformation` objects in a `Vector`.

The registry runs on well-known port 5555 and spawns a new thread to handle each request from clients. It can perform four operations:

1. **Add an entry**: This is performed when a PoD starts up and wishes to register itself with the registry. An entry is added to the list of `PoDInformation` objects only if the list does not already contain a PoD with the same name attribute.
2. **Lookup an entry**: Uses the PoD's name as the searching key, returns the `PoDInformation` object if a match occurs and null otherwise.
3. **List all entries**: Returns all `PoDInformation` objects currently registered.
4. **List IXP name**: Since, as discussed in Chapter 5, each registry belongs to a particular IXP, the registry has an IXP name associated with it (given as a command line parameter). This operation returns this name.

Finally, since PoDs do not explicitly deregister when they stop running, the registry spawns a separate thread that takes care of cleaning up any entries of PoDs that are no longer running. The reason for not having a PoD deregister when it stops running is that if it were to crash it would fail to deregister, leaving a permanent and invalid entry in the registry.

The cleanup thread determines whether a PoD is still running by attempting to connect to it: if the attempt fails, the PoD is no longer running and can be safely removed from the registry. The thread repeats this process for all the entries in the registry and then goes to sleep for a constant number of milliseconds, currently set at 5000. The cleanup thread performs this operation as long as the PoDRegistry is running.

## 7.2 GUI

The previous sections in Chapter 6 describing the GUI focused on the different types of graph it can draw. This section will, instead, focus on all its other features. While in terms of system architecture a GUI need only be able to display one type of graph, the GUI that was actually implemented can handle data from PoDs belonging to any of the RMFs previously discussed.

When it begins running but before the user can begin interacting with it, the GUI contacts all the registries it knows about and retrieves the information for all the PoDs in them. The IP addresses of these registries are given as command line parameters, but the GUI could find these out by an out-of-bounds means such as a well-known DNS name like, for instance, registry.ixpname.com:5555. The GUI then populates the tree that appears on its left hand side. The highest hierarchical level in the tree has only one component labelled “IXP list”; the second level contains one element for each registry that has been contacted and is labelled with the name of the IXP that the registry belongs to; finally, the third level has all the PoDs currently registered in a registry.

The user is now free to select as many PoDs as desired, even if they are not from the same IXP. This fulfils the client’s requirement that graphs from different IXPs be displayed side by side for comparison purposes. When the user selects a particular PoD, its attributes (name, IP address, port number and a brief description) are displayed in the status bar at the bottom of the GUI, allowing the user to decide whether the selected PoD is in fact the desired one. At this point, hitting the “Select”

button causes the selected PoDs to be displayed on the area at the right of the GUI. If more than one PoD was selected the graphs are displayed top to bottom, with a vertical scrollbar appearing at the right of the graphs. The “Refresh” button simply clears any graphs that are currently being displayed, essentially reverting the GUI to the state it was in when it first began running. The same function is available from the menu bar under “View”.

Another powerful feature of the GUI is that it allows the user to save the graph currently on display to a file (this is accessible through the menu bar found at the top of the GUI, though only one graph must be on display for this feature to work). The addition of this feature arose directly from a client requirement. Once a file is saved it can be loaded back into the GUI, causing the saved graph to reappear. If a network administrator finds the data on display troubling, he or she can save it to a file to be able to share it with others, perhaps via email. The person receiving the file can easily restore it, provided that he or she has the same GUI or one that understands the format that the file is in. The file is saved with extension .dat, but it is basically a text file. Here is a sample saved file for the RMF MinMaxPing:

```
MIN_MAX_PING
IXPAmsterdam Min Max Ping for all
150
127.0.0.1 , 0 , -1 , 62 , 50 , 56.0 , 8.48528137423857 , 2004-07-23 22:07:00
144.82.194.58 , 0 , 60 , 57 , 50 , 53.5 , 4.949747468305833 , 2004-07-23 22:07:00
127.0.0.1 , 0 , 60 , 53 , 45 , 49.25 , 3.304037933599835 , 2004-07-23 22:07:30
144.82.194.58 , 0 , 180 , 70 , 53 , 61.0 , 8.54400374531753 , 2004-07-23 22:07:30
144.82.194.58 , 1 , -1 , 60 , 50 , 55.0 , 7.0710678118654755 , 2004-07-23 22:07:40
127.0.0.2 , 0 , -1 , 60 , 60 , 60.0 , -1.0 , 2004-07-23 22:07:40
```

The first line is the type of graph to be drawn. The following table describes the types of graphs corresponding to the different RMFs:

| RMF Name | Graph Type Literal |
|---|---|
| RTTvsTime | NORMAL |
| ThresholdRTT | THRESHOLD_RTT |
| ThresholdRTTRealTime | THRESHOLD_RTT_REAL |
| LinkStatus | LINK_STATUS |
| MinMaxPing | MIN_MAX_PING |

**TABLE 7.1: Graph type literals for PoDInformation’s podType member**

The second line of the saved file contains the title of the graph. The third line is the value entered by the user for how recent the data should be; for those RMFs to which this does not apply the GUI outputs a 0. Following this third line come the lines for

the actual data. Each field in each line is comma-separated, and the number of fields depends, naturally, on the type of RMF being dealt with. These fields are described in Chapter 6 in the Figures appearing in the Introduction section of each RMF. The fact that the data are saved as text in the standard format just described means that a programmer could easily implement a different GUI that could read in the file.

A final feature is the ability to pause the display of a real-time RMF by clicking on the "Pause" button that appears under the "Refresh" button when a real-time RMF is being displayed. This gives a network administrator the ability to analyse the data currently on display in more detail and to save it in order to share it with colleagues. The following screen capture shows the GUI displaying two graphs, one for RMF RTTvsTime and another one for RMF ThresholdRTT:

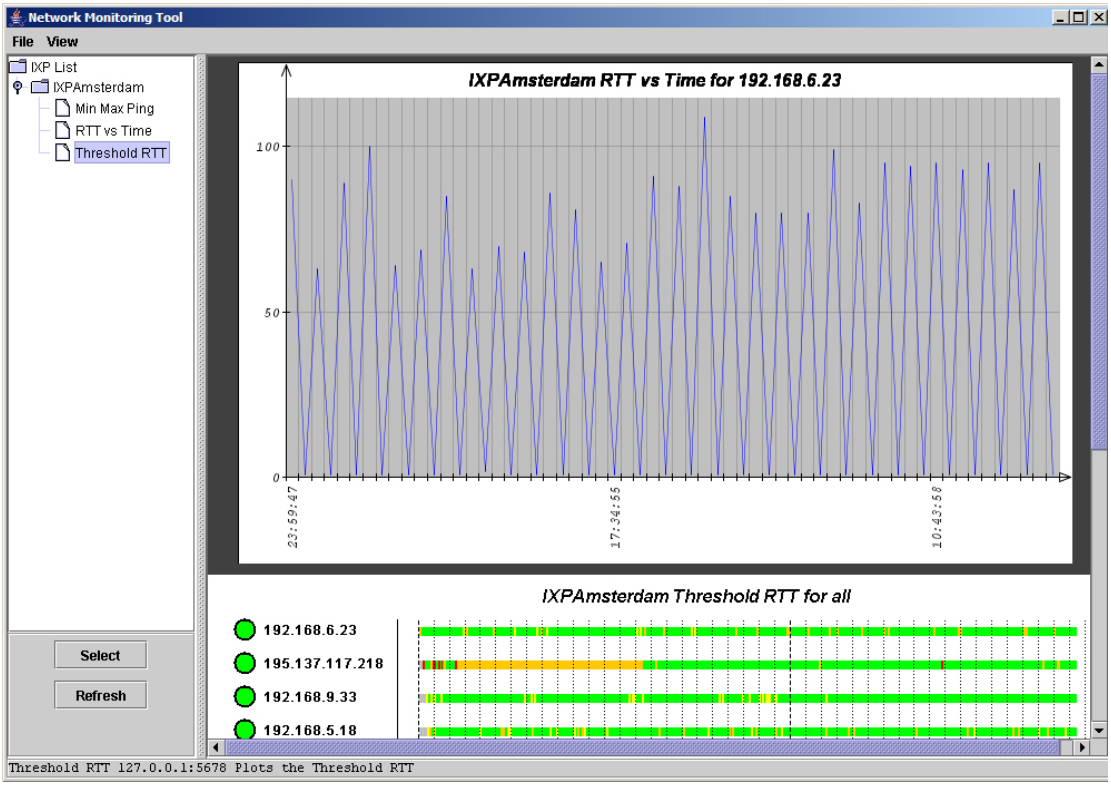



**FIGURE 7.1: Screen capture of the GUI**

# Chapter 8: Protocols

## 8.1 Introduction

Two protocols were designed to provide communication over a network between the different components defined in the system architecture: the PoDRegistry protocol used by the PoDRegistry and the VP protocol used between the clients (the GUI) and the PoDs.

While communication could have taken place using an established language such as XML (Extensible Markup Language), this presented some disadvantages. In the case of the VP protocol which is used to transfer potentially large amounts of data, the overhead incurred from the XML tags would have hindered the performance of the transfer and, consequently, the effectiveness of the graphs displaying real-time data. In addition, a simpler protocol like the VP protocol alleviates some of the performance penalties suffered from the use of SSL-encrypted communications.

In the case of the PoDRegistry protocol, it was felt that using XML would be overkill and that all necessary functionality could be accomplished with a new, simpler and more efficient protocol.

## 8.2 PoDRegistry Protocol (version 1)

The PoDRegistry protocol is used for all communications between the PoDs and the registry and again between clients and the registry. Note that all integers are four bytes long and are encoded using NetByte to ensure that there are no problems with little or big endian encodings and that the graph type is encoded as an integer using the conversion in Table 8.2. The protocol supports the following eight operations:

1. Request to add an entry to the registry, used by a PoD when it first starts up to register itself with the registry. The encoding of the request should return an array of bytes containing the protocol version id, the operation id and the PoD's information, as described in table 8.1.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 1 | Integer (4 bytes) |
| Number of bytes for the port | Integer (4 bytes) |
| Bytes for the port number | Variable |
| Graph type | Integer (4 bytes) |
| Number of bytes for the IP address | Integer (4 bytes) |
| Bytes for the IP address | Variable |
| Number of bytes for the name | Integer (4 bytes) |
| Bytes for the name | Variable |
| Number of bytes for the description | Integer (4 bytes) |
| Bytes for the description | Variable |

**TABLE 8.1: PoDRegistry protocol, byte encoding for operation id = 1**

| Encoding | Graph Type |
|---|---|
| 1 | NORMAL |
| 2 | THRESHOLD_RTT |
| 3 | THRESHOLD_RTT_REAL |
| 4 | LINK_STATUS |
| 5 | MIN_MAX_PING |

**TABLE 8.2: PoDRegistry protocol, graph type literals**

2. Request to look up an entry in the registry (the name of the PoD is used as the key to search for). The encoding of the request should return an array of bytes containing the protocol version id, the operation id and the name of the PoD to search for, as described in table 8.3.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 2 | Integer (4 bytes) |
| Number of bytes for the name | Integer (4 bytes) |
| Bytes for the name | Variable |

**TABLE 8.3: PoDRegistry protocol, byte encoding for operation id = 2**

3. Request to retrieve all the entries currently in the registry. The encoding of the request should return an array of bytes containing the protocol version id and the operation id, as described in table 8.4.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 3 | Integer (4 bytes) |

**TABLE 8.4: PoDRegistry protocol, byte encoding for operation id = 3**

4. Reply to a look up entry request. If the searched failed, the encoding of the reply should return an array of bytes containing the version id, the operation id and the number of results (0). If it is successful, the encoding of the request

should return an array of bytes containing the protocol version id, the operation id, the number of results (1) and the PoD's information, as described in table 8.5.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 4 | Integer (4 bytes) |
| Number of results (0 or 1) | Integer (4 bytes) |
| Number of bytes for the port | Integer (4 bytes) |
| Bytes for the port number | Variable |
| Graph type | Integer (4 bytes) |
| Number of bytes for the IP address | Integer (4 bytes) |
| Bytes for the IP address | Variable |
| Number of bytes for the name | Integer (4 bytes) |
| Bytes for the name | Variable |
| Number of bytes for the description | Integer (4 bytes) |
| Bytes for the description | Variable |

**TABLE 8.5: PoDRegistry protocol, byte encoding for operation id = 4**

5. Reply to a request to list all the entries currently in the registry. The encoding is exactly the same as for the previous operation (Table 8.5), except that the operation id is 5 and that if the registry contains more than one PoD, the array of bytes given by the encoding will contain the information for each PoD one after the other. In other words, rows 4 through 12 of Table 8.5 will appear once for each PoD in the registry, so that the last byte of the description of the first PoD (row 12) will be followed by the first byte of the number of bytes for the port of the second PoD (row 4).

6. Request to obtain the IXP name that the registry belongs to. The encoding of the request should return an array of bytes containing the protocol version id and the operation id, as described in table 8.6.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 6 | Integer (4 bytes) |

**TABLE 8.6: PoDRegistry protocol, byte encoding for operation id = 6**

7. Reply to a request to obtain the IXP name that the registry belongs to. The encoding of the request should return an array of bytes containing the protocol version id, the operation id and the IXP's name, as described in table 8.7.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 7 | Integer (4 bytes) |
| Number of bytes for the IXP name | Integer (4 bytes) |
| Bytes for the IXP name | Variable |

**TABLE 8.7: PoDRegistry protocol, byte encoding for operation id = 7**

8. Reply to an add entry to registry request, used by the registry to tell the PoD whether it was successfully added to the registry or not. The encoding of the request should return an array of bytes containing the protocol version id, the operation id and whether the operation succeeded (1) or failed (0), as described in table 8.8 (the operation could have failed because the PoD was already registered).

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 8 | Integer (4 bytes) |
| Entry added ID (1 if ok, 0 if failed) | Integer (4 bytes) |

**TABLE 8.8: PoDRegistry protocol, byte encoding for operation id = 8**

Note that for all operations an array of bytes is what actually gets transferred over the network, ensuring that the different components that communicate can be written in different programming languages. In addition, the use of NetByte ensures that there will not be any conflicts with little or big endian encodings. These two factors make this protocol entirely platform independent and supports the modularity of the architecture discussed in Chapter 5 (note that while NetByte is written in Java a similar package could be written for other programming languages).

The implementation of the protocol just specified was actually done in Java using `PoDInformation` objects. This object has two key functions: `podToBytes`, which encodes all the data members of the object into an array of bytes according to the tables in this chapter; and `bytesToPoDInformation`, which receives an array of bytes and converts it back into a `PoDInformation` object. The rest of the encodings, such as the version id and the protocol id are done by the protocol itself, implemented in the class `PoDRegistryProtocol`.

## 8.3 VP Protocol (version 1)

The visualization / PoD protocol is used by a client to request data from a PoD and by the PoD to return the requested data to the client. All the fields of the actual

data are arrays of bytes converted from strings. Regardless of the actual implementation of the protocol, all the data begins as a collection of arrays of strings. In addition, the number of elements for each array of strings is the same for each type of RMF (MinMaxPing, for instance, deals with arrays of size 8, as shown in Figure 6.10, while RTTvsTime has arrays of size 4, as shown in Figure 6.1). The VP protocol accommodates these differences as well as the fact that the number of bytes for each of the individual elements in one of these arrays varies. Like the PoDRegistryProtocol, this protocol uses NetByte to encode integers and longs.

The protocol has four operations, three to request data and one to reply with the data requested. This last one is by far the most involved and is discussed last. The three operations for requesting data are as follows:

1. Request for data from a PoD belonging to the RMF ThresholdRTTRealTime. The encoding of the request should return an array of bytes containing the protocol version id, the operation id, the IP address of the interface to retrieve data for and the age of the data (in seconds). If, for instance, 300 seconds is given as the age, then the data will be at most 300 seconds old. This encoding is described in table 8.9.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 2 | Integer (4 bytes) |
| Number of bytes for the IP address | Integer (4 bytes) |
| Bytes for the IP address | Variable |
| Number of bytes for the age of the data | Integer (4 bytes) |
| Bytes for the age of the data | Long (8 bytes) |

**TABLE 8.9: VP protocol, byte encoding for operation id = 2**

2. Request for non-real time data. The encoding of the request should return an array of bytes containing the protocol version id, the operation id and the IP address of the interface to retrieve data for, as shown in table 8.10.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 3 | Integer (4 bytes) |
| Number of bytes for the IP address | Integer (4 bytes) |
| Bytes for the IP address | Variable |

**TABLE 8.10 VP protocol, byte encoding for operation id = 3**

3. Request for data from a PoD belonging to the RMF MinMaxPing. The encoding of the request should return an array of bytes containing the protocol version id, the operation id, the IP address of the interface to retrieve data for and the age of the data (in seconds). If, for instance, 300 seconds is given as the age, then the data will be at most 300 seconds old. This encoding is described in table 8.11.

| Field Description | Field Type and Size |
|---|---|
| Protocol version ID = 1 | Integer (4 bytes) |
| Operation ID = 4 | Integer (4 bytes) |
| Number of bytes for the IP address | Integer (4 bytes) |
| Bytes for the IP address | Variable |
| Number of bytes for the age of the data | Integer (4 bytes) |
| Bytes for the age of the data | Long (8 bytes) |

**TABLE 8.11: VP protocol, byte encoding for operation id = 4**

The operation that replies with the data (operation id 1) takes a set of arrays of strings and converts it to an array of bytes. How this task is accomplished can be best described by the example shown in Figure 8.1:

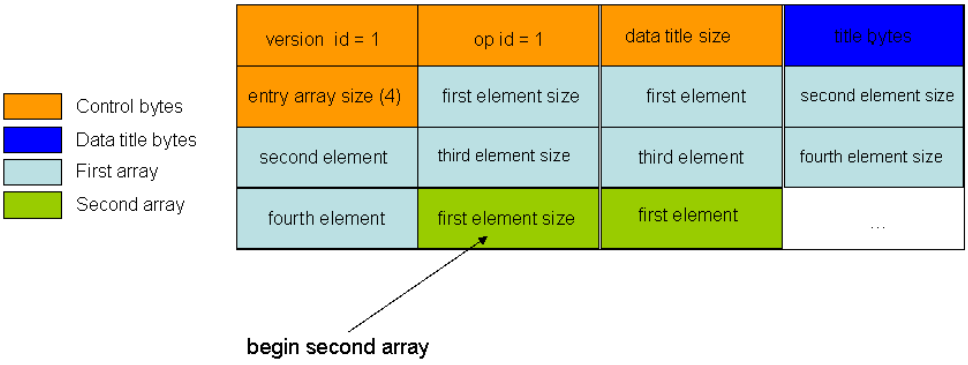



**FIGURE 8.1: Example of VP protocol's operation id 1**

For the following discussion note that all integers and longs are encoded with NetByte, each taking 4 and 8 bytes respectively. The array of bytes begins with the encoding of integer 1 (the version id), which takes four bytes. The next four bytes contain the encoding of integer 1, representing the operation id. The next four bytes contain an integer indicating how many bytes the title of the data takes up. The next field is of variable length, depending on what the title of the data is. Immediately following the last byte for the data title are four bytes containing an integer that indicates how many elements all the arrays of strings have (remember that all the

arrays for one RMF have the same number of elements, so this number needs to appear only once in the array of bytes). In the example shown in Figure 8.1 each array has four elements.

Next comes the first fields of the data. The first four bytes indicate the number of bytes taken up by the first element of the first array; these are followed by the bytes of the element itself. This process is repeated for all the elements in the array. The encoding for the second array begins right after the end of the encoding for the first array, so that the bytes for the last element of the first array are followed by the four bytes denoting the number of bytes of the first element of the second array, as shown in Figure 8.1.

All these fields give enough information to the decoding process to allow it to properly transform the array of bytes back into the original set of strings. The process knows that it has reached the last array of strings when it reaches the end of the array of bytes. Like the PoDRegistryProtocol, since all communication takes place in the form of array of bytes, the parties using the VP protocol can be written in different programming languages and environments.

Finally, note that all control fields, such as the number of elements in each array and the length of each field, were chosen to be quite long (4 bytes), so that the protocol will work even when presented with arrays with many elements and with elements of great size.

# Chapter 9: Security

## 9.1 Motivation

One of the client's requirements regarding the implementation of the system architecture was that the transfer of data over public networks such as the Internet be done using some sort of encryption. In addition, the client wanted to ensure that the PoD could authenticate the GUI before yielding any data and vice versa. These requirements led to the adoption of the Secure Sockets Layer since it is both secure and available as a package in several programming languages. Note that SSL runs on top of the VP protocol (see section 8.3) and encrypts all information exchanged between PoDs and the GUI. SSL is not used, however, in conjunction with the PoDRegistry protocol, since information about PoDs is not considered confidential: adding encryption results in a decrease in performance, so it is used strictly when necessary.

## 9.2 Implementation

Since all the PoDs and the GUI are written in Java, the current implementation makes use of Java's SSL package. Also, Keytool (see Section 10.2.6) was used to generate the necessary certificates and, consequently, some of the terminology found in this section derives from it. The whole process, including creating, exporting and importing certificates is depicted in Figure 9.1 (to view the precise Keytool commands needed to perform these operations please refer to Appendix A):

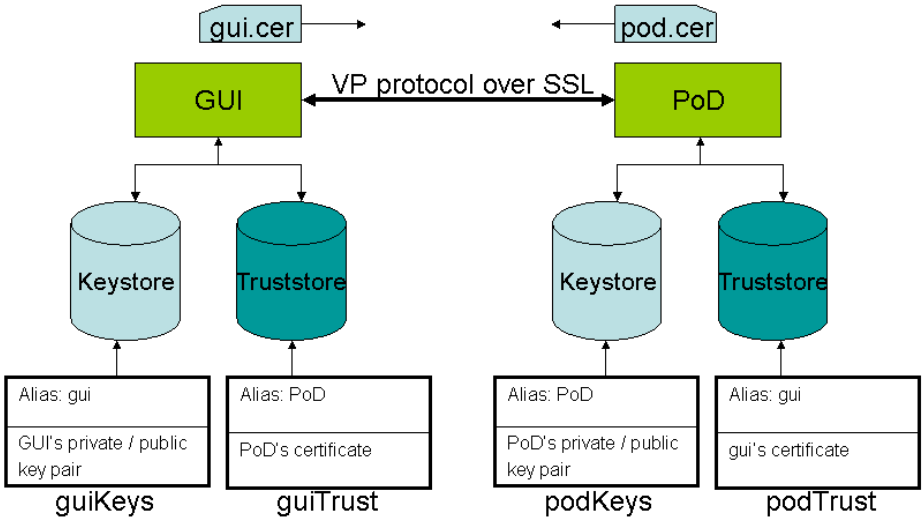



**FIGURE 9.1: Creating, exporting and importing certificates for SSL**

Each component interested in communicating via SSL with mutual authentication must have a keystore and a truststore. The former is used to keep public and private key pairs for the component and the latter contains certificates of parties the component trusts. To begin the process, a component like the GUI must generate its public and private key pair and store it in its keystore, named "guiKeys" in Figure 9.1. Each entry in the store is identified by an alias, which in the case of this example is "gui". In addition, a keystore has a password as does each individual entry in it (these are not shown on the Figure for simplicity's sake). Once the GUI has generated its keys, it must export its public key into a certificate and send it to all PoDs; the certificate is basically a file called `gui.cer`. Note that the actual transmission of this file is at present time done by some out-of-bounds means such as FTP or Secure FTP.

Upon arrival of the certificate, the PoD must import it into its truststore, essentially declaring that it trusts the GUI and will accept to share an SSL connection with it. Once the certificate file has been imported it can be disposed of. This completes client authentication, where the GUI (the client) authenticates itself with the PoD (the server). The same process must now be carried out in the opposite direction, to provide mutual authentication. Briefly, the PoD generates its keys, creates a certificate for its public key and sends it to the GUI, who upon receiving it imports it into its truststore. Verifying that the certificate or public key just received actually belongs to its stated owner is usually done by a Certification Authority; however, since this can be a costly operation, at present it is up to the user importing the certificate to ensure (perhaps by calling the issuer of the certificate) that its fingerprint is valid and has not been changed in transit.

Once these exchanges are completed the parties interested in communicating via SSL can be started. Using Java's SSL package the PoD, for instance, loads the keystore and the truststore into a `KeyManagerFactory` and a `TrustManagerFactory`, respectively, which are then used to initialize the SSL context (and object of type `SSLContext`). The final step is to obtain a `ServerSocketFactory` object that will create the SSL server socket needed to listen for requests from clients on. Once this socket is obtained it can be used as if it was a regular, non-SSL server socket. The process is much the same for the GUI, except that instead of obtaining a `ServerSocketFactory` from the `SSLContext` it retrieves an `SSLSocketFactory`.

# Chapter 10: Development Tools & Technologies

## 10.1 Programming Languages

### 10.1.1 Java

Java was selected because it is platform independent, allowing easy deployment into varied environments like those of different IXPs. Any computer running the Java Virtual Machine is able to interpret Java's byte code regardless of its platform and therefore run any application written in Java; Java is therefore used to overcome platform incompatibility.

In addition, Java was chosen for its strong capabilities for graphics and networking, making it ideal for implementing the graphical user interface and the protocols for communication across networks. The choice was further justified by the fact that Java provides an API for SSL, easing the implementation of secure communication of data.

A final motive for using Java was the familiarity of the individual group members with this particular language. This meant that the implementation of the prototype system and most of the individual components for the network monitoring toolkit could progress in a fast-paced manner, clearly an advantage considering this project's rather short development time.

### 10.1.2 C++

During requirements capture meetings with LINX, it became clear that staff presently working for our client did not have a strong programming background in the Java language. Since the majority of components such as the graphical user interface, the PoDs and the Glues were implemented using Java, the group wanted to demonstrate that the system architecture is not dependent on any specific programming language. The most appropriate component to demonstrate this concept is the resource-specific part of the Glue (G-RS). The reason for this is that it is the one component that has to be rewritten by each IXP to suit its particular hardware. Consequently, the G-RS was implemented using C++ since in general network administrators are somewhat familiar with this language; this implementation gave proof of concept that the designed system architecture is language independent, and

that IXPs can write their specific Glues (indeed, any of the architecture's components) in any language they wish, while still being able to utilise standardised components such as the graphical user interface and the PoDs, which are currently implemented in Java.

## 10.2 Tools

### 10.2.1 Ujac

Ujac is the abbreviation for Useful Java Application Components. This project made use of one of these components, a charting library for drawing simple graphs. This charting component provides a variety of designed charting types which are available through a simple but generic interface, supporting the generation of 2D pie, 3D pie, 2D line, 2D bar charts and many other types. This particular charting component proved to be useful in visualising the display of data in a meaningful form according to the requirements that LINX provided.

Another advantage of Ujac is that it is freely distributed. Most of the other packages that were found online required its users to buy an expensive development license. This was considered an unacceptable option, as this project did not have the kind of funding needed to pay for these licenses. In addition, Ujac is quite easy to learn and use, while other packages found were full blown graphics packages which required a steep learning curve and were therefore not regarded as a sensible option given the time constraints of the project.

The Ujac suite was primarily used for the implementation of the prototype which is described in earlier sections of this report. When the requirements for the graphical visualisation of processed data became clearer through requirements capture meetings with LINX, it was found that Ujac was not flexible enough to display all the graphs needed by the different tools in the kit. Instead, the implementation of the graphical display for most of the tools relied on Java's 2D package.

### 10.2.2 Fping

Fping is a network utility based on the original ping utility. The original version of ping is limited in its functionality. For instance, it can only ping hosts once a second, a limitation which Fping does not have. Further, Fping allows the user to ping any number of hosts on the command line, or to specify a file containing a list of

these hosts. In addition, instead of trying one host until it times out or replies like it is the case with ping, fping will send out a ping packet and move on to the next host in a round-robin fashion. If a host replies, it is noted and removed from the list of hosts to check; if a host does not respond within a certain specified time and/or retry limit it will be considered unreachable. This a useful feature in terms of the project since ping's blocking mechanism could prove very inefficient when faced with many hosts to ping.

One last advantage of Fping is that it has been designed for use in scripts and its output is easy to parse.

### 10.2.3 IXP-watch

IXP-watch provides the functionality of polling hardware like switches and routers to obtain raw data such as round trip times. Once the hardware has been polled and the information processed, IXP-watch provides a log file named `alarms.log`. The resource-specific part of the Glue (G-RS) for some of the RMFs in this project make use of this file and have been implemented to understand the data representation format that IXP-watch provides. The G-RS translates data in `alarms.log` into a standard data format that can then be used by a PoD for further processing, since the PoD understands this standard data format.

### 10.2.4 Javadoc

Any person that has programmed in Java will have come across the API specification created by Javadoc. The Java API is a catalogue which provides a description of the available classes, inner classes, interfaces, constructors, methods and fields to the programmer. The Java API specification therefore eases the development process.

Because an IXP interested in using the network monitoring toolkit is likely to want to modify the Glue, PoD or GUI, it was imperative to provide clear documentation to their APIs. For this purpose the Javadoc tool was used. Javadoc is a tool for generating API documentation in HTML format from source files. API documentation generation takes place by parsing the declarations and documentation comments in a set of source files and producing a set of HTML pages describing the classes. In order to create this documentation, comments which are specifically understood by the Javadoc parser were added to all source files.

### 10.2.5 CppDoc

CppDoc is another tool which allows the generation of API documentation in HTML format from source files. The only difference between Javadoc and CppDoc is that CppDoc can generate documentation from source code written in the C++ programming language, whereas Javadoc generates documentation from source code written in Java. CppDoc was chosen because its output is virtually identical to the one provided by the Javadoc tool, providing documentation for the entire project that is homogeneous and, consequently, easier to use.

### 10.2.6 Keytool

Keytool is a key and X.509 certificate management tool. Keytool was employed in this project to create private and public key pairs and their associated certificates in order to use SSL to provide secure communications. It was chosen because it comes standard with the Java SDK.

### 10.2.7 Together ControlCenter 6.0.1

Together ControlCenter is an integrated development platform designed to simplify and accelerate the analysis, design, development and deployment of complex enterprise applications. It includes powerful utilities for the modelling of unified modelling language (UML) diagrams, including class, use case, sequence, collaboration, activity, state, component and deployment diagrams. Together was used to generate class diagrams straight from source code, a much more efficient way of creating this documentation than drawing it by hand.

### 10.2.8 Microsoft Visio

Visio is a software package developed for the creation of all types of diagrams such as block, brainstorming business process, flowcharts, organisational charts, project schedule and timeline charts diagrams, just to name a few. Visio was mainly employed to create system architecture diagrams to aid with the implementation process. Furthermore, Visio was also used to create project management diagrams such as project schedules and organisational charts.

## 10.3 Operating Systems

### 10.3.1 Windows

Windows XP was chosen as the main development platform for the prototype and for the implementation of the majority of the components for the network monitoring toolkit. The reason for choosing windows was convenience: all group members had windows XP already installed on their machines. Due to the variety of powerful development packages available under windows, the group initially saw no particular reason to use another operating system. Development packages such as NetBeans, JEdit and editors such as Emacs for windows were used throughout the project. The choice of platform was not dependent on the programming language used, as the Java programming language itself provided platform independency and C++ compilers are available for all major operating systems.

### 10.3.2 Solaris

The UNIX distribution Solaris was used to conduct testing in a larger network than that created by the group members' own machines. Solaris was used because it is the operating system deployed at University College London Computer Science labs where this testing took place.

### 10.3.3 Linux (Knoppix, Penguin Sleuth Kit)

Knoppix is a full distribution of the Debian operating system, which has the advantage that it can be booted from a CD without the need for installation on any host machine. The reason for using it was to progress with the implementation of two RMFS, LinkStatus and MinMaxPing, which rely on `fping` to generate the data. Since `fping` does not come with Windows nor is it easily installed on that platform and since it was not at first installed in the computer labs at University College London, the Penguin Sleuth Kit (which does come with `fping`) was used on the group members' machines.

# Chapter 11: Project Management

## 11.1 Client Interaction

This project differs from most other DCNDS projects in that it has a real client, the London Internet Exchange. As such, it involved additional phases, including user acceptance testing, thorough system testing on the client's hardware infrastructure and the final deployment of the system.

Thorough analysis and planning are some of the core disciplines necessary when implementing a project according to real business requirements. In addition, it is necessary to stay in constant contact with the client in order to provide information on the progress of the project and to be aware of any changing user requirements so that they can be added to the next implementation release. Project management techniques and development methodologies should be chosen in such a way that the client is seen as part of the project and not just an external entity.

## 11.2 Scope

Prior to starting this project, the group had to agree on its scope and its expected outcomes. This is good practice and is usually done for any professional project prior to the implementation process. Defining the scope is necessary in order to be able to estimate if the objectives can be successfully met within the given time interval. If it turns out that more could be achieved within this time limit then the scope of a project can be extended.

The scope for this particular project has roughly been defined in chapter 3 of this report, which illustrates the objectives. A more detailed definition of the scope is given below.

- Design a system architecture which is not dependent on any hardware or programming language.
- Implement a prototype as proof of concept for the system architecture.
- Define a standard data format.
- Implement a network monitoring toolkit which is specifically built to suit LINX's existing hardware and software.
- Carry out user acceptance testing.

- Deploy the network monitoring toolkit at LINX.
- Provide relevant documentation for user training.
- Provide guidelines for other IXPs to implement the Glue module and make changes to existing modules.

The scope of this project was limited to LINX. Therefore, any network monitoring software was developed according to LINX's requirements, their specific hardware infrastructure and their existing data representation formats. Due to tight time constraints, the group agreed that it would not be possible to analyse the data representation format of other interested IXPs in order to build versions particularly suited to their needs and hardware. However, one of the deliverables of this project was a set of relevant documentation including user manuals and a Javadoc description of the various APIs of the implementation, which would allow other interested IXPs to write their own G-RS suited to their current data representation format. Finally, the scope of the project included a testing and deployment stage to ensure that the final product ran properly on the client's infrastructure.

## 11.3 Assumptions and Constraints

An important project management aspect prior to starting this project was to define some assumptions and constraints. These were discussed and specified during the first meeting with LINX, and helped in mitigating some risks arising from the fact that the project has a real client. The constraints discussed with LINX gave the group some rough guidelines on what could and could not be done. The assumptions and constraints are as follows:

**Assumptions:**

- LINX will provide the raw data files needed to develop their G-RS.
- LINX will provide a contact (a customer) to refine requirements and to evaluate results (Rob Lister).
- LINX will allow access to their network and relevant hardware in order to carry out a testing and deployment phase.

**Constraints:**

- IXPs want to use their own internal data format.

- IXPs will not be able to give extensive test time on their networks.
- Access to data from other IXPs may be restricted.
- Delivery Deadlines
  - 25th of May: Working prototype.
  - 28th of May: Project management presentation.
  - 27th of August: Final demonstration at LINX.
  - 6th of September: Submission of all deliverables.

## 11.4 Development Methodology

At the beginning of the project, the requirements LINX had were only poorly defined and not very well understood. In addition, since LINX is a client working in a real, fast-paced business setting where requirements are likely to change on a day-to-day basis, it was necessary to adopt a methodology that allowed for the constant refinement of requirements and their implementation. More standard development methodologies such as the waterfall model require that the requirements are precisely stated before any implementation takes place. This means that under these models it is extremely difficult to go backwards and deal with new requirements that might arise throughout the project. For these reasons it was decided to adopt an iterative an incremental approach; in particular, the Extreme Programming (XP) methodology was chosen.

XP is a lightweight methodology for small to medium-sized software development teams. Having relatively short release cycles is one of XP's most common practices which empower developers to confidently respond to changing customer requirements even late in the development cycle. The group members decided to have iterations fixed to one week throughout the whole project. This meant that necessary or additional required features could be included on a weekly basis. In particular, each week the group dealt with the highest priority features first and worked on secondary objectives only if time permitted, mitigating the risk of wasting time on details while neglecting main objectives.

Extreme Programming heavily emphasises teamwork. The group, which consists of developers, testers and managers, are all seen as part of a team dedicated to delivering a software product. XP implements a simple yet effective way to enable group development by utilising the pair programming practice, where all code is written with two programmers at one machine. Pair programming allows for rapid

and effective communication and also minimises the risk of bugs. It has been found to work very well throughout this whole project. Some other XP practices adopted for this project are listed below:

- Small releases.
- Simple design.
- Testing.
- Refactoring.
- Pair programming.
- Collective ownership.
- Continuous integration.
- 40 hour week.

## 11.5 Team organisation

In general the group met and worked together on a daily basis. Part of the planning phase of this project was to design and agree on a weekly work schedule which roughly outlined the working hours as well as what general tasks needed to be accomplished on that day. The group agreed to a schedule beginning at 10 am from Monday till Friday. Prior to starting work, the group held a meeting as an overview of the day and the tasks to be performed. In addition, the schedule called for a stop to the work at 4 pm followed by a brief meeting on what had been achieved during that day. This schedule was interrupted on Tuesdays for a meeting with the project's supervisor (more on this in the next section).

The work schedule provided some basic working guidelines for the team and tried to ensure that no member had to work longer hours than others. Each individual member was assigned a certain set of tasks which had to be accomplished. The specific tasks each group member dealt with and how he organised his time is discussed in that member's individual report.

## 11.6 Team and Client Communication

Carrying out a group project of such magnitude requires efficient and reliable communication between all five group members. Furthermore, it is seen to be of equal importance to stay in constant contact with client and supervisor since they are a significant source of detailed system requirements, problem resolution and feedback on what has been achieved so far.

The group arranged several face to face meetings with LINX for the purpose of defining and understanding the systems requirements of the network monitoring toolkit. One of the top priorities of the first meeting was to set up a list of customer contacts to use in case issues arose. Rob Lister agreed to be available for technical issues, while it was decided that John Souter would be the general contact person for any other matters. Communication primarily took place with Rob through email, as he was the person that was put in charge of answering questions regarding the structure of the provided log files and also setting up a machine for testing and deployment purposes on their premises. Meetings with LINX were supposed to take place on an approximately monthly basis, but because some of LINX's staff was particularly busy during certain time periods this was not always possible.

The main communication method between members was in person, since for the most part the group met on a daily basis, especially during development. If group members were working away from each other due to certain circumstances, regular weekly meetings were scheduled. These meetings were held for several purposes including discussing the progress of the previous iteration, gaining feedback on what individual group members had achieved, planning the next iteration, defining the requirements for next iterations and prioritising tasks. When unexpected problems of any nature arose, ad hoc meetings were arranged to solve them as quickly as possible.

Group email was used extensively between all group members for several purposes. In particular, it facilitated the exchange of messages regarding the arrangements of meetings and also to receive feedback from members working on individual tasks. In addition, email was also used for the distribution of documents and for communicating with the group's supervisor.

It was also agreed to meet the supervisor on a weekly basis every Tuesday. These meetings were held to keep the supervisor updated on the group's progress. In addition, they served as a forum for demonstrations of the current system implementation and for discussing any further requirements and problems. Detailed minutes of these meetings (as well as those held with LINX) were generated for later consultation.

The group set up a web site dedicated to the project containing all relevant information and documentation. Among the documents found online were the logs from the weekly meetings. In addition, presentations held during the project, architecture diagrams, documentation on using Javadoc, references and other

important documents were placed online so the whole group had access to them when needed. A description of the current source code releases and their current functionality could also be found, but not the code itself since it was decided that it should not be publicly available. Not only did the project web site prove useful in supplying the group members with relevant documents, it was also seen as a source of information to anyone outside the group interested in this particular project. Further, it was possible for the supervisor and the client to track the progress of the project by accessing the meeting minutes and other materials. The web site can be accessed at http://www.cs.ucl.ac.uk/students/z15_5/.

MSN messenger was also utilised during this project, allowing remote collaboration when members could not meet in person.

## 11.7 Project Schedule and Milestones

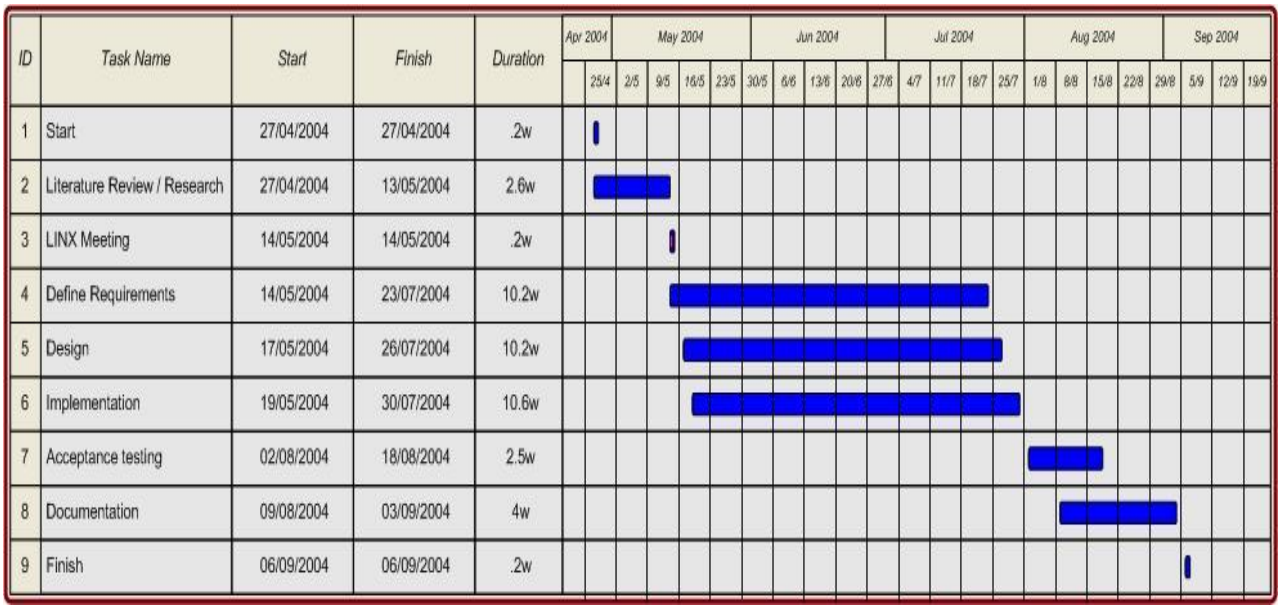



**FIGURE 11.1: Project schedule**

When this project was in its early planning phase it was difficult to define any major project milestones as the key requirements had not been laid out yet. It was difficult, as a result, to develop any schedules which would specifically outline what each XP iteration would entail. Nonetheless, while a precise schedule was not possible, Figure 11.1 shows the general schedule that was agreed upon. The schedule shows the different stages of the project and how much time was to be spent on each stage. Each square represents an iteration of one week.

Research and reviewing of any relevant literature was scheduled to occupy around two and a half weeks. At this point no user requirements were defined, so the group had to wait until a meeting with LINX was finally possible. This meeting took place on May 14th, 2004 and was used to shape the objectives and to define the client requirements.

After this was accomplished the stages of system design and implementation could be carried out. Notice that in Figure 11.1 these stages overlap: this is due to the XP methodology, whereby both client requirements and the system's design are constantly refined, a process called refactoring. Once certain requirements were agreed upon, the implementation process, which was scheduled for around eleven weeks, began. Upon completion of this stage, a thorough testing and deployment stage followed, lasting two and a half weeks. The following stage consisted of four weeks of documentation, producing the group report, the individual reports, a user manual and descriptions of the APIs generated.

Milestones were not defined until the client's requirements themselves began to materialize during the first meeting on May 14th, 2004. One such milestone was the development of a prototype to test the feasibility of the system architecture. This was a milestone set by the supervisor and the delivery deadline was set for May 25th, 2004. Another milestone was completed on May 28th, 2004, the deadline for the project management presentation. The following bullet points summarize the project's milestones:

**Milestones:**

- 14th of May: Requirements meeting with LINX.
- 25th of May: Working prototype.
- 28th of May: Project management presentation.
- 31st of July: Finish coding, begin deployment at LINX.
- 1st of August: Begin documentation and report.
- 15th of August: Complete deployment at LINX.
- 27th of August: Final demonstration at LINX.
- 6th of September: Final submission of all deliverables.

## 11.8 Status Monitoring

Status monitoring ensures that the project is progressing efficiently and at a steady rate; it identifies any backlogs and tries to resolve issues as rapidly as possible. The project web site, mentioned earlier in Section 11.6, proved to be a useful tool for status monitoring since important documents or other relevant information could be accessed there. The description of system functionality for each release gave an indication of the implementation's progress. In addition, meeting minutes, also available online, presented the reader with a review of what had been achieved in the previous week and the tasks to be accomplished the following week, as well as a measuring stick for gauging whether the weekly goals set out had been realistic. The minutes acted as simplified task cards, which are usually utilised in an Extreme Programming approach. The project schedule is another means for supervisor and group members to track the progress of the project. If certain milestones indicated on the project schedule on a certain date have not been achieved, then the project supervisor can assume that the project is not progressing according to schedule.

## 11.9 Risk Management

Throughout the project several strategies were adopted to minimise risks. Since this project was dealing with a real client, it was apparent that many risks would be directly associated with the client. This chapter gives an explanation of the project risks identified and the actions taken to address them.

The Extreme Programming methodology adopted for this project proved to be very useful, as it mitigated many of these risks. Pair Programming, for example, ensured that an individual did not have to deal with difficult tasks on his own, but was assisted by a second member of the team. Working in pairs reduced the risk of not being able to accomplish tasks and also reduced the likelihood of bugs in the code.

The weekly feedback meetings which have been discussed in section 11.5 were particularly useful in addressing any new issues that were identified and to assess existing ones to see if they had been overcome or not. Risks and issues were noted on a log sheet accessible to everyone. Risks were identified in terms of seriousness and the likelihood of them occurring. Each risk was classified to have either low, medium or high seriousness and likelihood. A few of these risks follow:

**Users don't want the system:**

This is a risk which has been classified to be of medium seriousness. As this project is being undertaken for LINX, the likelihood that the client rejecting a working system built according to their requirements is low. In order to minimise the risk of the client rejecting the finished implementation, it was decided to give demonstrations of the system at regular intervals. This allowed LINX to examine the current release and state whether it met their requirements or not. If it was the case that the implemented functionality did not meet LINX's expectations, then thanks to employing the Extreme Programming methodology, system requirements could be redefined and the required functionality implemented in the next iteration.

**Limited knowledge of user requirements:**

Building a system which does not match the user's defined requirements because these were poorly understood could render it completely useless. In order to minimise this risk, meetings were arranged with LINX to capture its requirements. If it turned out that these were not very clear or poorly understood, it was good practice to contact LINX again to rule out any mistakes. Additionally, since the group's supervisor was present in these meetings, the group could also approach him for clarification purposes, which was often the case.

**LINX denies access to data:**

LINX denying access to data such as log files, which are necessary to build the various G-RSs, posed a threat of high seriousness. The objectives stated that a functional network monitoring tool would be implemented according to the specific requirements LINX provided. Without access to these particular log files, the outcome of the project could have been jeopardised as the G-RS could not be adjusted to the data representation formats found in these files and the system could not, therefore, gather any data for processing, rendering it useless. One of the top priorities during the first meeting with the client was to agree on access to raw data. Another threat consisted of LINX denying the necessary access to its hardware for deployment of the monitoring toolkit. As these issues had been discussed with LINX during the meeting the likelihood that this would happen was very low. However, were this to happen, it was agreed that the group would approach the supervisor who

could then renegotiate with the client the availability of the data and any other issues preventing the group's progress.

**Unfamiliar technology:**

Developing a network monitoring toolkit was a project that none of the group members were initially very familiar with. Adopting new technologies always poses a threat, especially in software development, since the developed software utilising this technology might not perform as required or the user might have had higher expectations regarding the final product's reliability and functionality. The right choice of development methodology, in this case Extreme Programming with its incremental release approach, meant that risks could be mitigated by expanding system functionality incrementally.

# Chapter 12: Testing and Deployment

## 12.1 Testing

Standard development methodologies such as the waterfall model reach the testing phase only after the implementation phase has finished. The Extreme Programming methodology used in this project, on the other hand, called for constant and continuous testing. Whenever a particular part of a component was implemented, no matter how small or trivial, tests were performed to ensure the correctness of this fragment of code. Sometimes testing was done by simply outputting intermediate and final results to the screen, then changing the inputs, recompiling, and observing the output once again. Other times testing was done by visually checking the display of a graph of an RMF. In yet some other cases, separate classes, usually containing a main method, were constructed in order to test the fragment of code or function just implemented. Some of these classes can be found in the `testfiles` subdirectory in the CD included with this report.

After having finished the implementation of the PoDRegistry and the PoDRegistryProtocol, for instance, a text-based client, PoDRegistryClient.java, was used to test them. This class relied on a text-based menu for user input, as follows:

```
Pod registry client menu:
-------------------------
        1) List all pods
        2) Look up a pod
        3) Add a pod
        4) Get IXP name
        5) Quit

Please make your choice:
```

Several other files and classes were created to test various parts of the implementation. A listing of them, along with a brief description for each, follows. Naturally, many more tests than these were conducted during the course of development, so this list should not be considered exhaustive.

- `TestAscendingThresholds.java`: Tests the function `setThresholds` of class **`FpingConfigEntry`** to make sure that it does not accept thresholds that are not in ascending order.

- `TestFlipVector.java`: Tests the class `VectorFlipper` to ensure that it correctly flips the elements of a `Vector` by printing the elements of the original and those of the flipped `Vector` to the console.
- `TestGetSystemTime.java`: A simple class testing the retrieval of the current system time.
- `TestGlueRealTimeRefresh.java`: Tests that the Glue of RMF ThresholdRTTRealTime properly reads data that is at most as old as the number of seconds given to it as input.
- `TestIPAddressSort.java`: Tests that the `sort` method of `Arrays` works properly on an array of objects of type `IPAddress`.
- `TestPoDInformationBytes.java`: Tests the `PoDInformation` function `poDToBytes` which converts a `PoDInformation` object to an array of bytes according to the `PoDRegistryProtocol` for transmission over a network. Likewise, this class tests the function `bytesToPoDInformation` which receives an array of bytes and transforms it back into a `PoDInformation` object.
- `TestPoDRT4Analysis.java`: RMF LinkStatus was previously known as RT4. This class tests the correctness of the algorithm used to determine the status of a particular ping result.
- `TestRandomAccessFile.java`: Used for testing of the class `TailFileReader` during its development.
- `TestTimeStamp.cpp`: Tests that the time stamp created by the function `getTimeStamp` in `GlueMinMaxPingBackEnd.cpp` is in the desired format.
- `TestVPprotocol.java`: Tests the encoding and decoding of data for a very early version of the VP protocol.

The testing of real-time RMFs requires a special mention. Since during development access to real-time LINX log files was not available, two programs, `AlarmsUpdater.java` and `FpingUpdater.java`, were implemented to generate data for RMF ThresholdRTTRealTime and RMF LinkStatus, respectively. In this way it was possible to view a working, real-time version of these RMFs despite the lack of real data. An updater program was not built for RMF MinMaxPing due to time constraints.

Finally, another form of rudimentary but extremely effective form of testing was Extreme Programming's pair programming practice. Many minor bugs were eliminated by simply having a person verify the code as it was being typed by the programmer.

## 12.2 Deployment

Deployment took place remotely on one of LINX's lab machines. Only minor glitches were encountered, mostly having to do with wrong IP addresses used as command line parameters, ports already being in use or the use of stale certificates. After a few minutes of solving these problems, the PoDs of all the RMFs as well as the PoDRegistry were started on LINX's machine. In addition, two clients (two instantiations of the GUI) were ran, one from a Solaris machine at the University College London Computer Science labs and a second one from a Windows XP machine plugged into UCL's network. All RMFs ran without flaws; the screen capture provided in Figure 12.1 shows the RMF LinkStatus running. Note that the status bar at the bottom of the GUI displays 195.66.241.35, the address of LINX's lab machine.

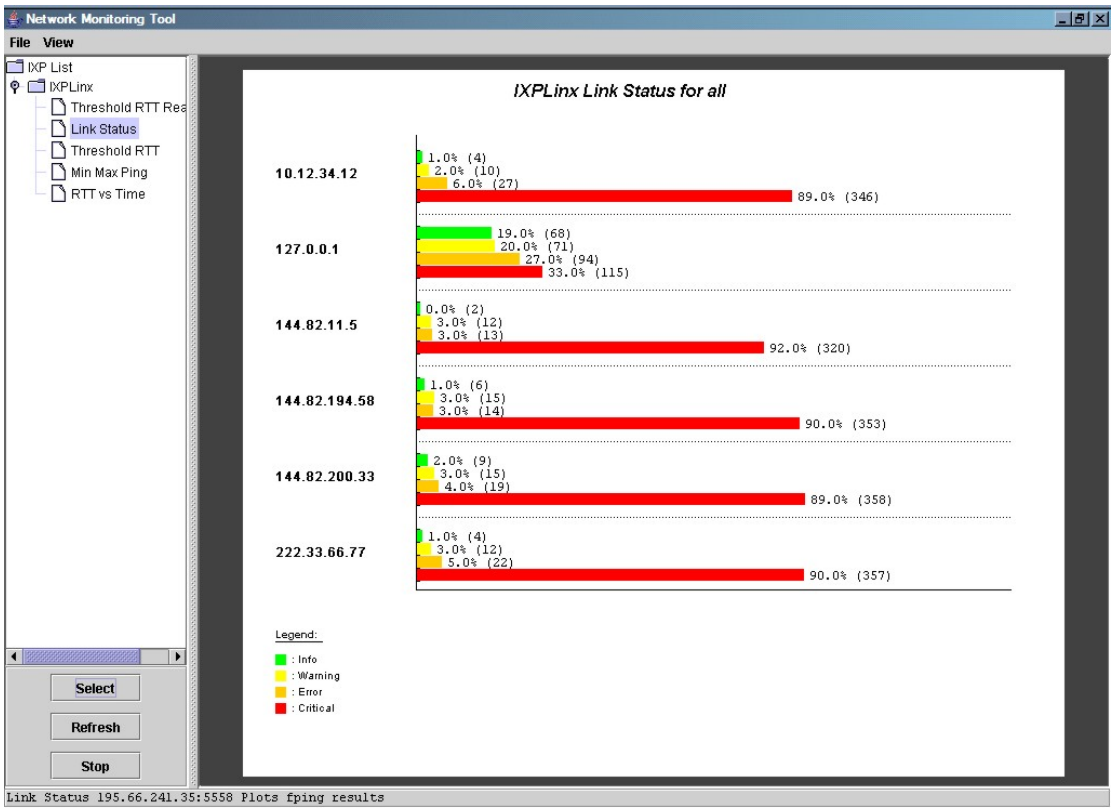



**FIGURE 12.1: Screen capture of data being obtained remotely from LINX**

The final deployment test consisted of having a client running Windows XP connect to two IXPs, LINX's lab machine and UCL's `aldgate.cs.ucl.ac.uk`. In this way, the group was able to test that the GUI could display two real-time RMFs from two separate IXPs side by side, as shown the screen capture of Figure 12.2. At this point LINX gave a list of real interfaces that could be queried and the same was obtained for UCL's network: these were utilized during the final demonstration given at LINX in which three clients, two Windows XP machines and a MacOS machine, where used to simultaneously retrieve data from both sites. It is important to note the significance of having installed the system on a MacOS machine, demonstrating, in actual fact, that it is platform-independent (the source code used for this installation was exactly the one used for the Windows XP machines).

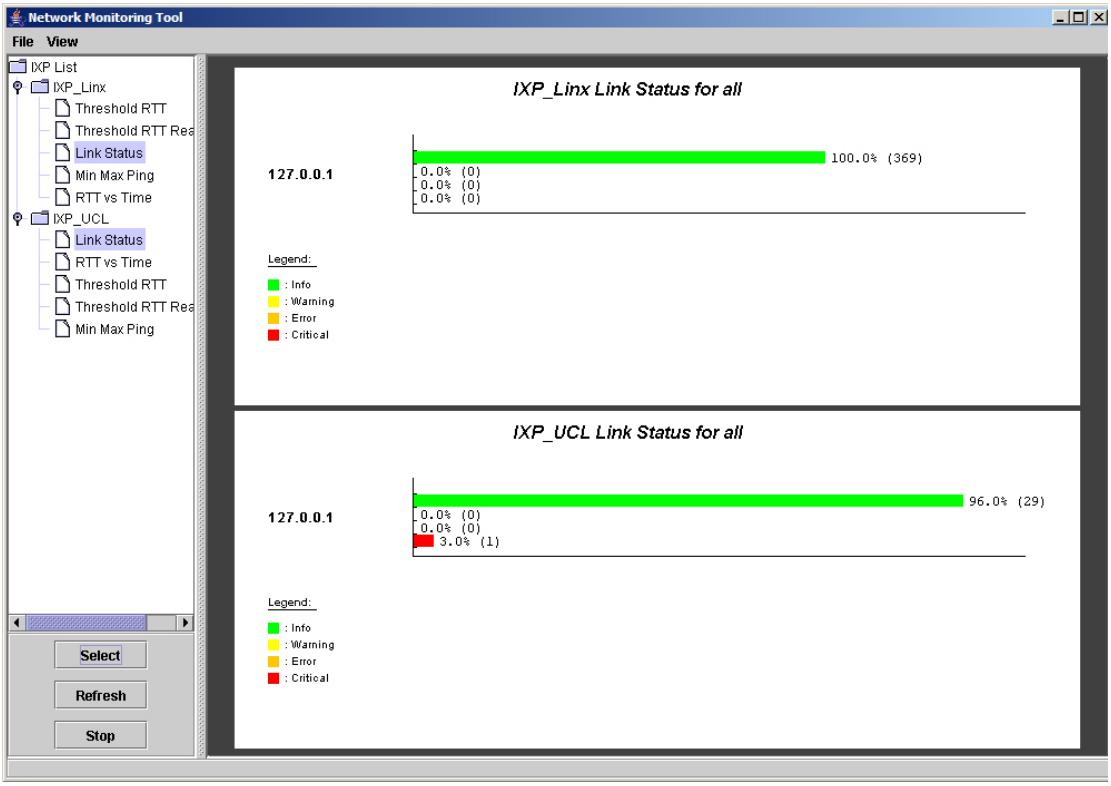



**FIGURE 12.2: Screen capture of real-time data obtained remotely from LINX and UCL**

# Chapter 13: Future Work

Given the time constraints of this project, especially in terms of development and deployment, several possible extensions to the work accomplished exist. First, even though the RMFs RTTvsTime, ThresholdRTT and ThresholdRTTRealTime were successfully run on LINX's machine, the `alarms.log` file that they used to obtain the data from was an old copy used for testing purposes. In the future, the Glues and PoDs for these RMFs would have to be deployed on the same machine running IXPwatch, so that they may obtain up-to-date data.

In addition, while the deployment phase was highly successful, it only involved one real IXP, LINX (the UCL machine was not a real IXP). Further work needs to be conducted to deploy the system on other IXPs of Euro-IX, to see whether the system implemented will, in fact, encourage a greater degree of collaboration and sharing of data between IXPs. While the architecture certainly allows the GUI to obtain data from different IXPs and it is flexible enough that an RMF implemented by one IXP may be used without too many changes by another IXP, these features should be put to the test by conducting further deployments. Also, while the graphs of the RMFs were designed according to accepted guidelines, it would be useful to conduct testing to see if they are in practice useful to network administrators or not.

Another extension to the project would be to physically separate the location of the Glue and the PoD of an RMF, making them communicate over a network. To accomplish this a new communication protocol would have to be designed, though the VP protocol could probably be used. In this way, the machine generating the data would not have to suffer the reduction in performance suffered by having to permanently be running the PoDs that analyse potentially large amounts of data; instead, the machine would only have to run the Glues, which, as a result of conducting no analysis, are much more lightweight. The physical separation of Glue and PoD does mean that the data would have to be transferred over a network, but considering that the internal network of an IXP has very fast links this should not present any real problems.

Perhaps the most obvious extension to the project is the construction of additional RMFs in order to provide a more complete toolkit. One tool that was not implemented because of time constraints is RMF BGPMonitor. BGPMonitor is based on a looking glass, which is a tool that network operators use to see routing views of

other networks. Most IXPs, including LINX, provide a looking glass that is accessible from their web pages. BGPMonitor would send a query to the web sites of other IXPs, retrieving information regarding the presence of LINX AS routes. If there is information about LINX AS routes then this would be marked with a green rectangle in the graph displayed (see Figure 13.1 for an example of what the graph would look like); conversely, the lack of a route would be shown as a red rectangle. If there is a case where LINX AS routes start to disappear from other IXPs' collector routers then the tool would raise an alarm to grab the user's attention. In short, this RMF is very useful since it saves a network administrator the trouble of having to periodically visit the web pages of many IXPs, retrieving data from them and then having to analyze the data to see if a problem is in fact taking place.

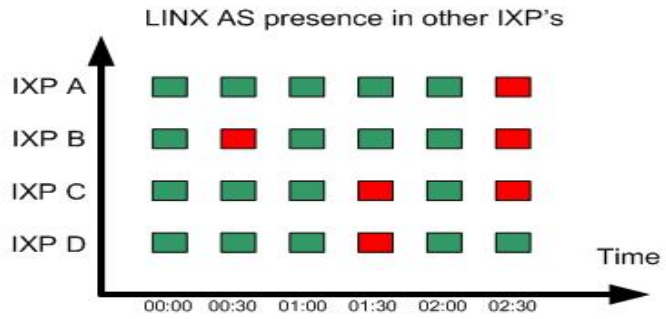



**FIGURE 13.1: Display of RMF BGPMonitor (a future tool)**

Regarding the VP protocol, it would be useful to develop a compiler that, given a set of types to transmit, would generate the actual functions that would convert these into the array of bytes to send. Currently this has to be done by hand, but having such a compiler would certainly speed up the development of new tools. In addition, a more formal specification of types would need to be provided.

Another small shortcoming of the protocol is that it transmits numbers as strings. Clearly this is inefficient, so it would be useful as future work to have the communication protocols convert these strings that have been read from log files into numbers for transmission over the network. A final minor inefficiency is that since some of the round-trip times returned by the fping program have fractional parts, a few values are transmitted as longs. In order to make the communication more efficient, the system could use units of microseconds instead of milliseconds, thus eliminating the fractional part of the value and the need for longs. It may even be the case that IXPs do not need so much accuracy in the first place, in which case it would suffice to simply round the round-trip time values.

Finally, during the final demonstration for LINX the client suggested two possible extensions to the current state of the project. First, it would be useful to create a SuperPoD that would gather data from many PoDs and page the administrator should a particular problem with the network exist. Second, the client suggested that it would be possible for an IXP to use the system developed for this project to give access to data from a particular piece of hardware to that hardware's vendor; this would provide the vendor with a powerful debugging tool that does not currently exist.

# Chapter 14: Conclusion

The two primary objectives of this project were the design of an entire new system architecture and the implementation of a network monitoring toolkit based on it. One of the biggest achievements of the project was the design of a system architecture that allows a tool developed by one IXP to be used by another IXP without this second IXP having to change its data representation formats; in other words, the architecture is IXP and hardware-independent. In addition, it is extensible and customizable since it can use an IXP's existing backend tools and scripts and it can do multi-site monitoring. Further, the architecture is scalable through encapsulation: a PoD can gather data from many other PoDs, essentially acting as a "SuperPoD". An administrator in charge of several IXPs could, in principle, create a SuperPoD that would query all other PoDs and display only problematic data, giving him or her a powerful and efficient mechanism for monitoring many networks at once. It is also worth noting that the architecture is entirely modular, given complete independence between the client (in the case of this project's implementation the GUI) and the PoDs. This means that the client is not only platform-independent but can also be developed independent of the PoD, giving the programmer a great deal of freedom. A client need not even be graphical: sometimes a simple text-based client may suffice. Yet another advantage of the architecture is that it allows an administrator to specify which parts of the network or interfaces to monitor through the use of simple configuration files. A very simple Remote Monitoring Function (RMF), RTTvsTime, was built based on this architecture, thus proving its feasibility.

Another achievement was the specification and implementation of standardized ways to graphically display the data for each different RMF or tool. In addition, the display for RMFs ThresholdRTT (both versions) and LinkStatus are unlike anything provided by current monitoring tools: these displays were designed and built in such a way that a system administrator can easily spot problems such as a certain machine being down or round-trip times exceeding fixed threshold values.

Since in general the network data is confidential and is, in the current implementation, sent over a public network, another achievement of the project that arose straight from a client requirement was the use of encryption and mutual

authentication by means of SSL. In this way the IXP can be sure that its data is only being accessed by authorized parties.

Another advantage of the system is that the client is approximately 512KB and takes a few minutes to install, making it effortless for a network administrator to install it on any machine to perform remote monitoring; indeed, the system could be made available from a web page for download.

One of the major challenges of the project was meeting the requirements of a real client. To ensure that the product being developed was exactly what the client desired, the group regularly gave the client updates on the current status of the project. Even if the project went off-track, these regular updates guaranteed that the client could quickly state that the current development did not seem to fit his or her requirements and allow the group to implement minor refactoring in order to get the project back on track; in a similar manner, the client could specify additional requirements that arose during the course of the development. The involvement of the client was also crucial during deployment, aiding the group to set up the necessary components on the client's machines.

Perhaps the culmination of all these achievements was the successful deployment of the system at LINX, using it to monitor real interfaces from Europe's largest and most successful IXP. As a result of this deployment the project received the customer's acceptance in the form of a letter from the Chief Execute Officer, John Souter, stating that LINX were more than satisfied with the product delivered.

Despite the many challenges along the way, the project has been vastly rewarding and a great success: all team members contributed to developing a product that is technically sound and that meets the requirements given by the client. It is the sincere hope of the group that additional work will be undertaken to further the achievements that this project has already accomplished.

# Appendix A: User's Manual

## A.1 System Requirements

The system requirements are as follows:

- The Java compiler, preferably the latest one.
- A C++ compiler such as g++.
- The Java Virtual Machine, preferably the latest one.
- Keytool, a command line tool for creation of keys and certificates.
- Fping
- A decompressing utility, for instance Winzip (Windows) or tar (UNIX / Linux)
- Port 5555 (for the registry) and five other ports (one for each PoD) must be available.
- The Ujac charting library.

## A.2 Installation

### A.2.1 Compiling the Source Files

Once all the items in section A.1 have been installed the first step is to uncompress the source files; the procedure to do this will vary depending on the operating system being used. Next, all files in the main directory and in the subdirectories `auxiliary`, `auxiliaryGUI` and `rmfs` must be compiled using the Java compiler. A sample script for doing just this in Windows follows:

```
javac *.java
cd auxiliary
javac -classpath .. *.java
cd ..\auxiliaryGUI
javac -classpath .. *.java
cd ..\rmfs\RMFLinkStatus
javac -classpath ..\.. *.java
cd ..\RMFRTTvsTime
javac -classpath ..\.. *.java
cd ..\RMFThresholdRTT
javac -classpath ..\.. *.java
cd ..\RMFMinMaxPing
javac -classpath ..\.. *.java
cd ..\RMFThresholdRTTRealTime
javac -classpath ..\.. *.java
cd ..\..
```

Note that the `rmfs` directory contains five directories, one for each RMF discussed in Chapter 6. After this the C++ file `GlueLinkStatusBackEnd.cpp` found in the `RMFLinkStatus` subdirectory and the C++ file `GlueMinMaxPingBackEnd.cpp` found in the `RMFMinMaxPing` subdirectory must be compiled using the following commands from their respective directories:

```
g++ GlueLinkStatusBackEnd.cpp –o GlueLinkStatusBackEnd.out
g++ GlueMinMaxPingBackEnd.cpp –o GlueMinMaxPingBackEnd.out
```

The files `GlueLinkStatusBackEnd.out` and `GlueMinMaxPingBackEnd.out` must then be copied to the application's main directory (the directory where `GUI.java`, for instance, is located).

### A.2.2 Creating Key Pairs and Certificates

Once all files have been compiled Keytool must be used to create the necessary keys and certificates so that the SSL handshake will succeed. Since the system requires mutual authentication, all components must identify themselves and, consequently, must each create its public and private key pair as well as a certificate from its public key. This will results in six certificates: five for the PoDs in each of the five RMFs and one for the GUI. The following two commands create the keys and certificate for the GUI:

```
keytool -genkey -alias gui -keystore guiKeys -storetype jks -storepass 123456 -
keypass 123456 -keyalg "RSA" -keysize "512" -dname "CN=gui, OU=gui unit, O=gui inc,
L=london, S=london, C=UK"

keytool -export -alias gui -keystore guiKeys -storetype jks -file gui.cer -
storepass 123456
```

The first command tells Keytool to create a keystore with name "guiKeys" and password "123456" and to create a key pair under the alias "gui" and password "123456" using RSA with strength 512. The details of the owner of the key pair are also given. If any of these details are missing from the command Keytool will prompt the user for them. Note that it is possible to assign a different password to the alias "gui" than that of the keystore; in the case of this project's implementation they are the same. The second command creates the certificate file `gui.cer` from the public key of the alias "gui" in the keystore "guiKeys".

The next step is to exchange all six certificates. The GUI must be able to authenticate all five PoDs and needs, therefore, a certificate from each of them.

Conversely, each PoD must be able to authenticate the GUI, so each needs the GUI's certificate. The actual transferring of the files could be done using FTP or perhaps Secure FTP.

Once all the certificates have been transferred, each party must import those it has received into its truststore. The following command imports the certificate for the PoD belonging to RMF RTTvsTime into the GUI's truststore:

```
keytool -import -alias podRTTvsTime -keystore guiTrust -storetype jks -file
podRTTvsTime.cer -storepass 654321
```

Since no certification authority is being used to verify that the certificate actually belongs to whom it claims to belong to, upon entering the command Keytool will print the certificate's details along with its fingerprint and ask the user to confirm the import operation. To ensure that the certificate is valid the user could telephone its issuer and verify that its fingerprint is correct. In all, ten import operations must be carried out: five for the GUI importing the PoDs' certificates into its truststore and another five for each of the PoDs importing the GUI's certificate into its truststore. Tables A.1, A.2 and A.3 show the names of the keystores, truststores and aliases, respectively. The password for all keystores is "123456" and the password for all truststores is "654321".

| Component | Keystore |
|---|---|
| GUI | guiKeys |
| PoDRTTvsTime | podRTTvsTimeKeys |
| PoDThresholdRTT | podThresholdRTTKeys |
| PoDThresholdRTTRealTime | podThresholdRTTRealTimeKeys |
| PoDLinkStatus | podLinkStatusKeys |
| PoDMinMaxPing | podMinMaxPingKeys |

**TABLE A.1: Keystore names**

| Component | Truststore |
|---|---|
| GUI | guiTrust |
| PoDRTTvsTime | podRTTvsTimeTrust |
| PoDThresholdRTT | podThresholdRTTTrust |
| PoDThresholdRTTRealTime | podThresholdRTTRealTimeTrust |
| PoDLinkStatus | podLinkStatusTrust |
| PoDMinMaxPing | podMinMaxPingTrust |

**TABLE A.2: Truststore names**

| Component | Alias |
|---|---|
| GUI | gui |
| PoDRTTvsTime | podRTTvsTime |
| PoDThresholdRTT | podThresholdRTT |
| PoDThresholdRTTRealTime | podThresholdRTTRealTime |
| PoDLinkStatus | podLinkStatus |
| PoDMinMaxPing | podMinMaxPing |

**TABLE A.3: Aliases**

### A.2.3 Configuration Files

Before running the application, if the user is interested in running RMFs LinkStatus and/or MinMaxPing he or she must configure the files LS-RM.config and maxmin.config, respectively, to specify which interfaces to query. The format for these files was discussed in Sections 6.4.2 and 6.5.2, respectively.

## A.3 Running the Application

### A.3.1 Server Side (PoDs and PoDRegistry)

First, the two C++ back ends must be run so that they'll begin populating the necessary log files with data. To do so, make sure that the executables for them are in the main directory of the application and run them from the command line. In UNIX, for instance, this would be done with the commands:

```
./GlueLinkStatusBackEnd.out
./GlueMinMaxPingBackEnd.out
```

Next, the PoDRegistry must be started with the following command (execute this from the main directory):

```
start java PoDRegistry IXPLondon (Windows)
java PoDRegistry IXPLondon & (UNIX / Linux)
```

Finally, the desired PoDs must be started. The following five commands show all PoDs being started on ports 5678, 5679, 5680, 5681 and 5682 under Windows. For UNIX / Linux just remove the word "start" and add an & at the end of the command.

```
start java rmfs/RMFThresholdRTT/PoDThresholdRTT 128.64.111.10 5678
start java rmfs/RMFRTTvsTime/PoDRTTvsTime 128.64.111.10 5679
start java rmfs/RMFThresholdRTTRealTime/PoDThresholdRTTRealTime 128.64.111.10 5680
start java rmfs/RMFLinkStatus/PoDLinkStatus 128.64.111.10 5681
start java rmfs/RMFMinMaxPing/PoDMinMaxPing 128.64.111.10 5682
```

Note that the IP address given must not be "localhost" nor the loopback interface, as this address will be used by clients to connect to the PoDs.

### A.3.2 Client Side (GUI)

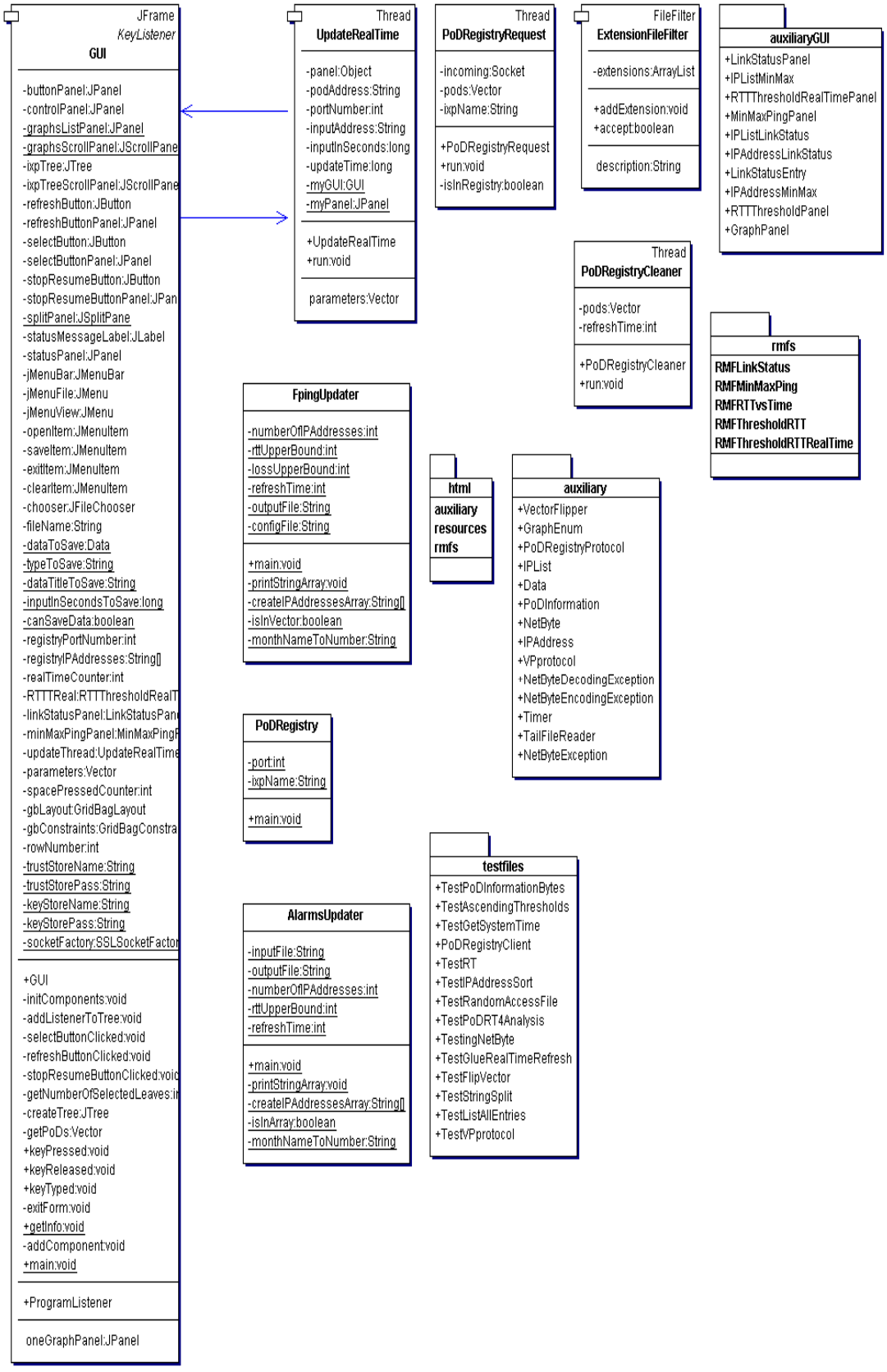
The GUI requires that the addresses of the registries it needs to contact are supplied as command line parameters. Thus, to run the GUI and have it contact registries at addresses 144.168.12.14 and 68.111.45.34 the user needs to type the command:

```
java GUI 144.168.12.14 68.111.45.34
```

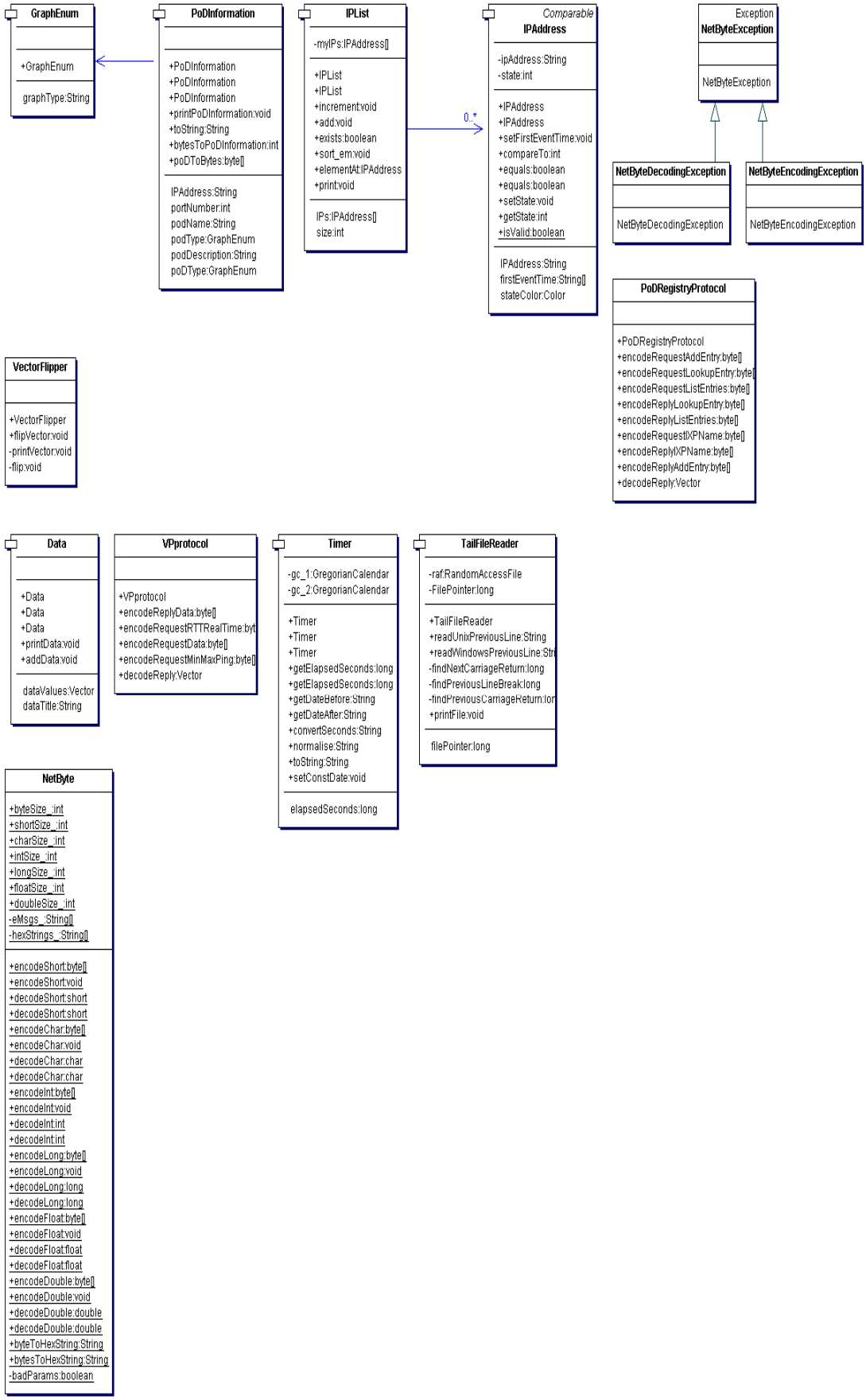
For information on how to use the GUI please refer to Sections 6.1.4, 6.2.4, 6.3.4, 6.4.5, 6.5.5 and 7.2.

# Appendix B: Class Diagrams

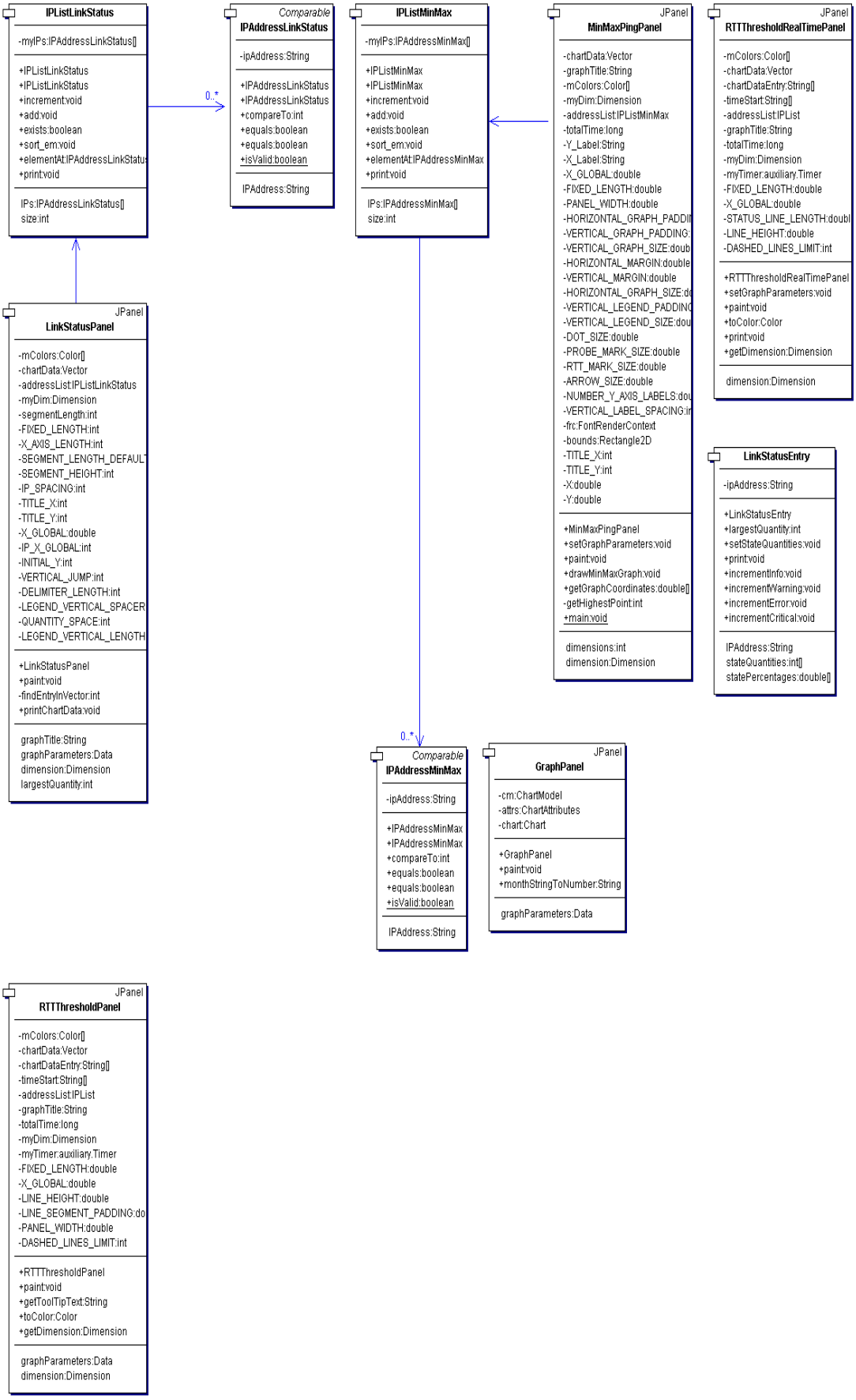
The diagrams beginning here until page 90 show the detailed structure of the classes. Those following show the relationships between them.



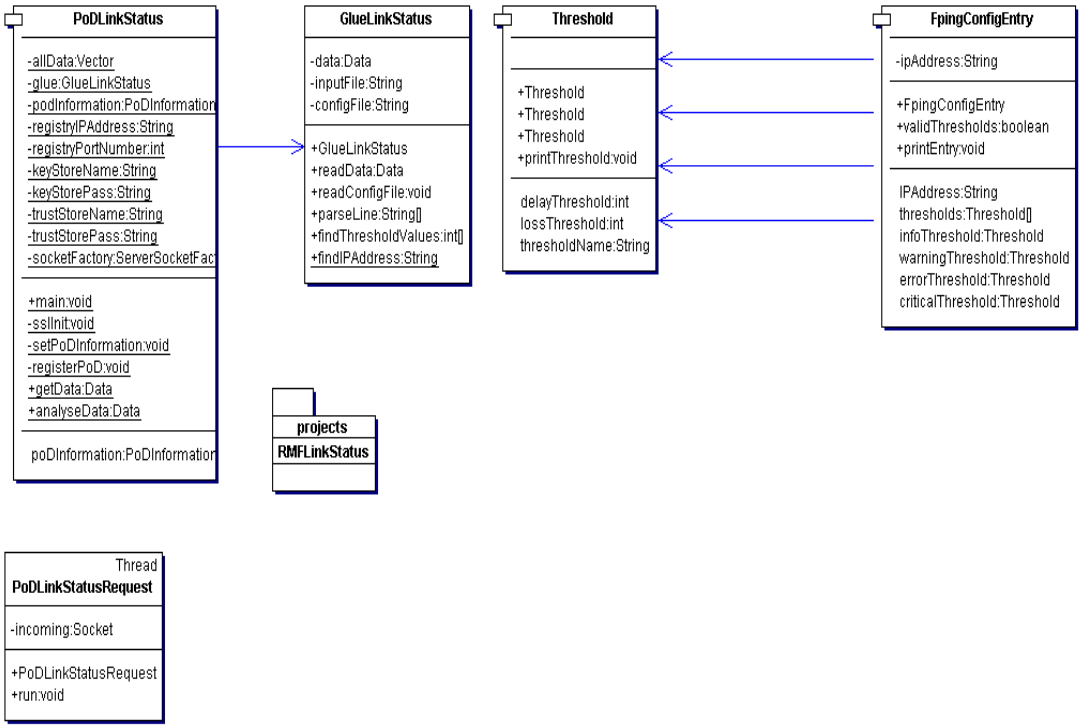
## Auxiliary


GraphEnum
+GraphEnum
graphType:String
PoDInformation
+PoDInformation
+PoDInformation
+PoDInformation
+printPoDInformation:void
+toString:String
+bytesToPoDInformation:int
+poDToBytes:byte[]
IPAddress:String
portNumber:int
podName:String
podType:GraphEnum
podDescription:String
poDType:GraphEnum
IPList
-myIPs:IPAddress[]
+IPList
+IPList
+increment:void
+add:void
+exists:boolean
+sort_em:void
+elementAt:IPAddress
+print:void
IPs:IPAddress[]
size:int
0..*
Comparable
IPAddress
-ipAddress:String
-state:int
+IPAddress
+IPAddress
+setFirstEventTime:void
+compareTo:int
+equals:boolean
+equals:boolean
+setState:void
+getState:int
+isValid:boolean
IPAddress:String
firstEventTime:String[]
stateColor:Color
Exception
NetByteException
NetByteException
NetByteDecodingException
NetByteDecodingException
NetByteEncodingException
NetByteEncodingException
PoDRegistryProtocol
+PoDRegistryProtocol
+encodeRequestAddEntry:byte[]
+encodeRequestLookupEntry:byte[
+encodeRequestListEntries:byte[]
+encodeReplyLookupEntry:byte[]
+encodeReplyListEntries:byte[]
+encodeRequestIXPName:byte[]
+encodeReplyIXPName:byte[]
+encodeReplyAddEntry:byte[]
+decodeReply:Vector
VectorFlipper
+VectorFlipper
+flipVector:void
-printVector:void
-flip:void
Data
+Data
+Data
+Data
+printData:void
+addData:void
dataValues:Vector
dataTitle:String
VPprotocol
+VPprotocol
+encodeReplyData:byte[]
+encodeRequestRTTRealTime:byt
+encodeRequestData:byte[]
+encodeRequestMinMaxPing:byte[]
+decodeReply:Vector
Timer
-gc_1:GregorianCalendar
-gc_2:GregorianCalendar
+Timer
+Timer
+Timer
+getElapsedSeconds:long
+getElapsedSeconds:long
+getDateBefore:String
+getDateAfter:String
+convertSeconds:String
+normalise:String
+toString:String
+setConstDate:void
elapsedSeconds:long
TailFileReader
-raf:RandomAccessFile
-FilePointer:long
+TailFileReader
+readUnixPreviousLine:String
+readWindowsPreviousLine:Str
-findNextCarriageReturn:long
-findPreviousLineBreak:long
-findPreviousCarriageReturn:lon
+printFile:void
filePointer:long
NetByte
+byteSize_:int
+shortSize_:int
+charSize_:int
+intSize_:int
+longSize_:int
+floatSize_:int
+doubleSize_:int
-eMsgs_:String[]
-hexStrings_:String[]
+encodeShort:byte[]
+encodeShort:void
+decodeShort:short
+decodeShort:short
+encodeChar:byte[]
+encodeChar:void
+decodeChar:char
+decodeChar:char
+encodeInt:byte[]
+encodeInt:void
+decodeInt:int
+decodeInt:int
+encodeLong:byte[]
+encodeLong:void
+decodeLong:long
+decodeLong:long
+encodeFloat:byte[]
+encodeFloat:void
+decodeFloat:float
+decodeFloat:float
+encodeDouble:byte[]
+encodeDouble:void
+decodeDouble:double
+decodeDouble:double
+byteToHexString:String
+bytesToHexString:String
-badParams:boolean

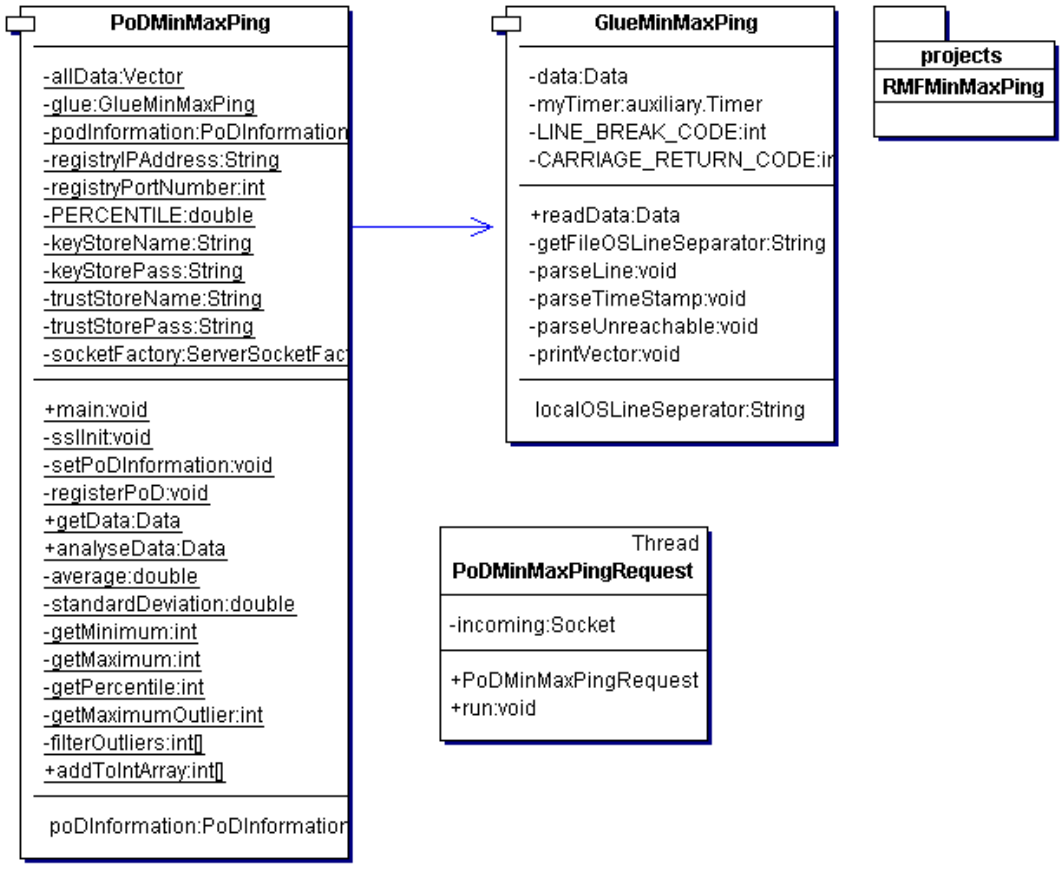
# AuxiliaryGUI



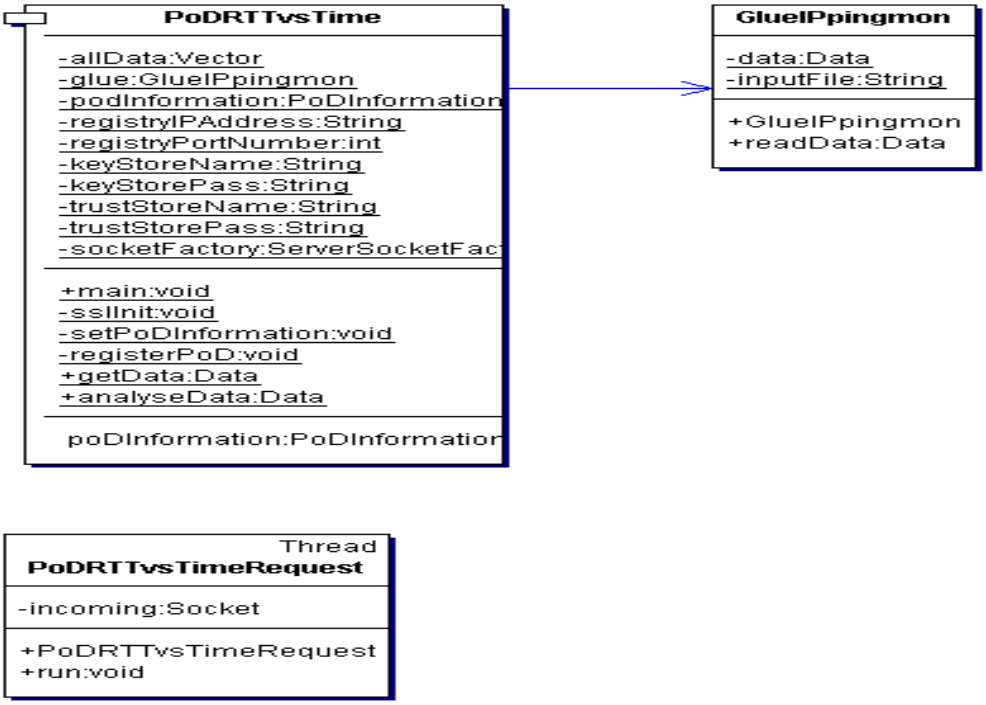
## RMFLinkStatus



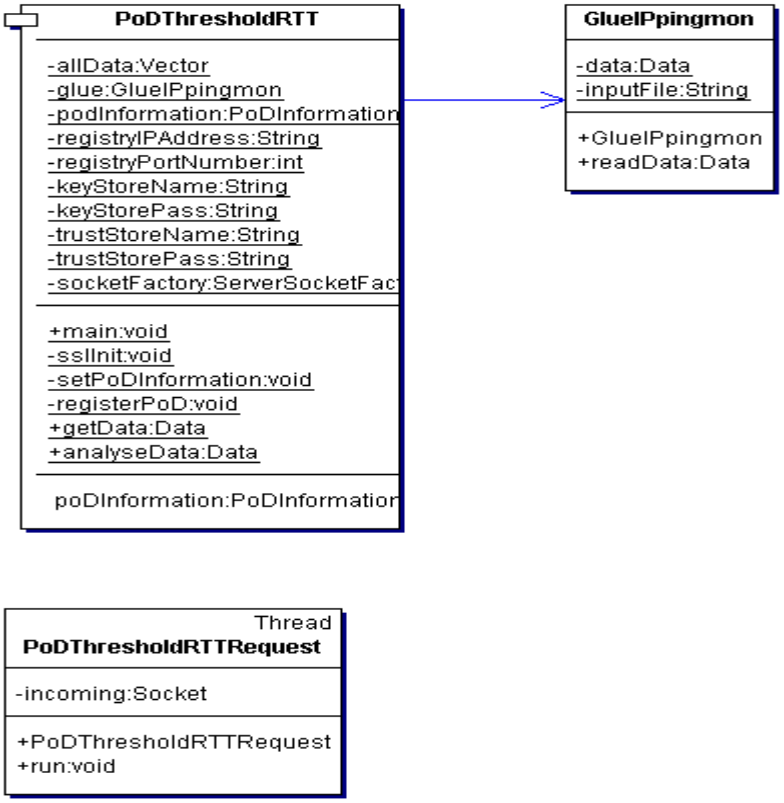

## RMFMinMaxPing



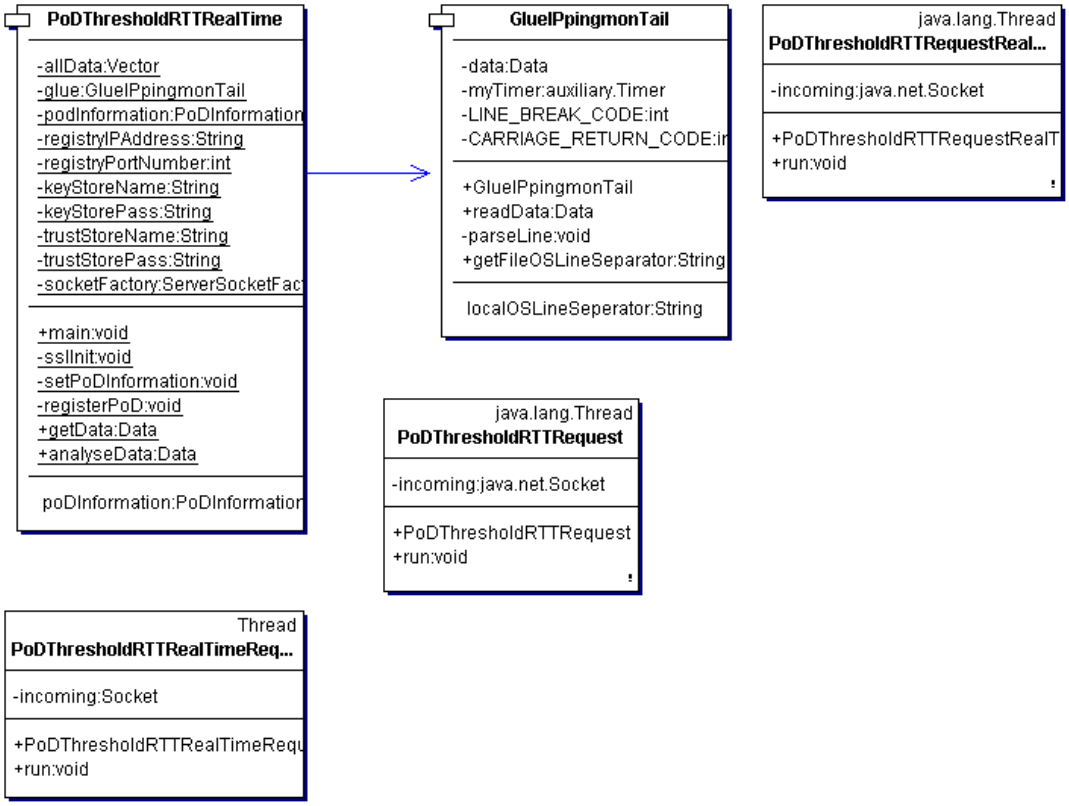
## RMFRTTvsTime

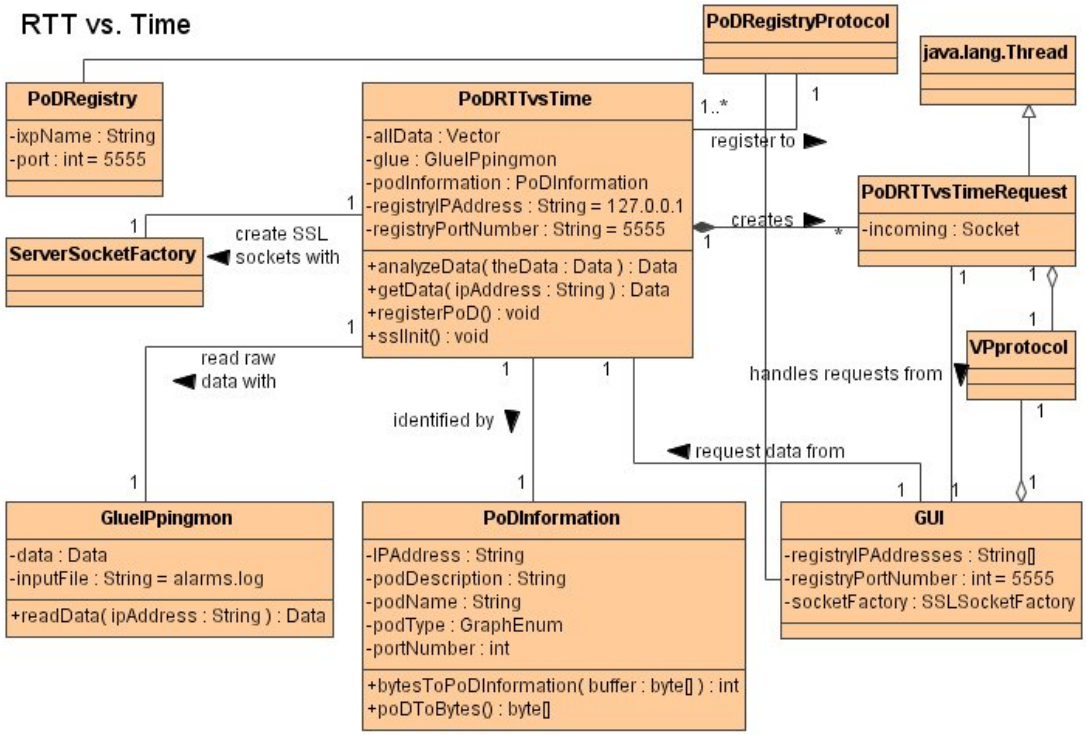


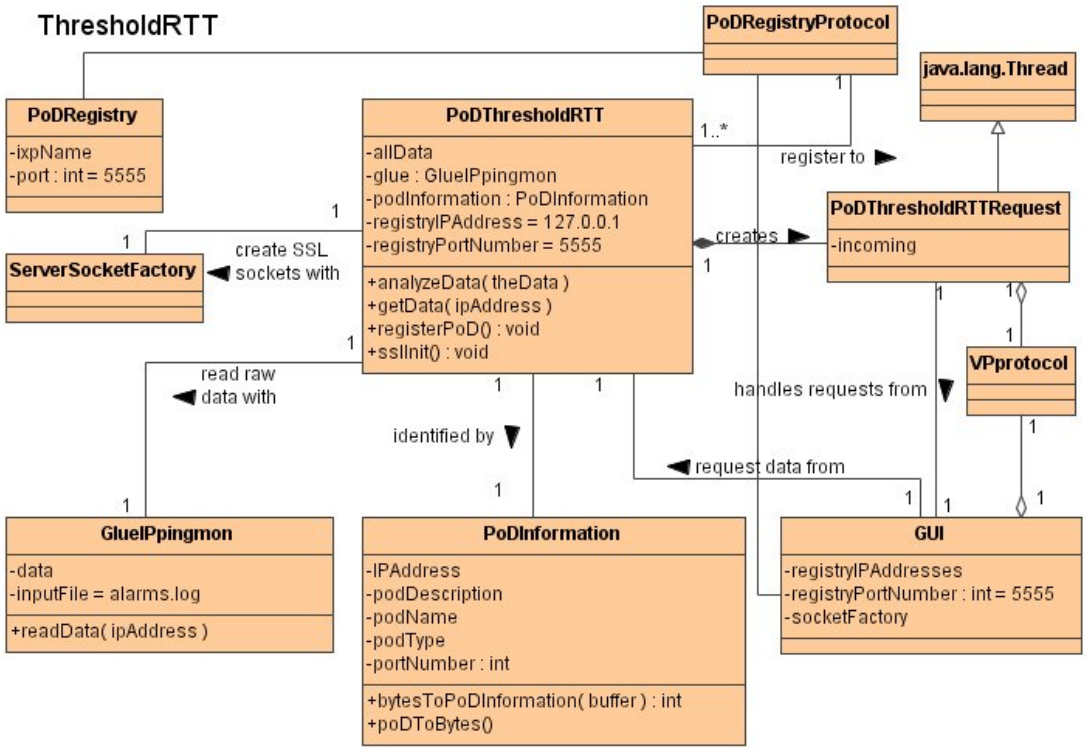

## RMFThresholdRTT

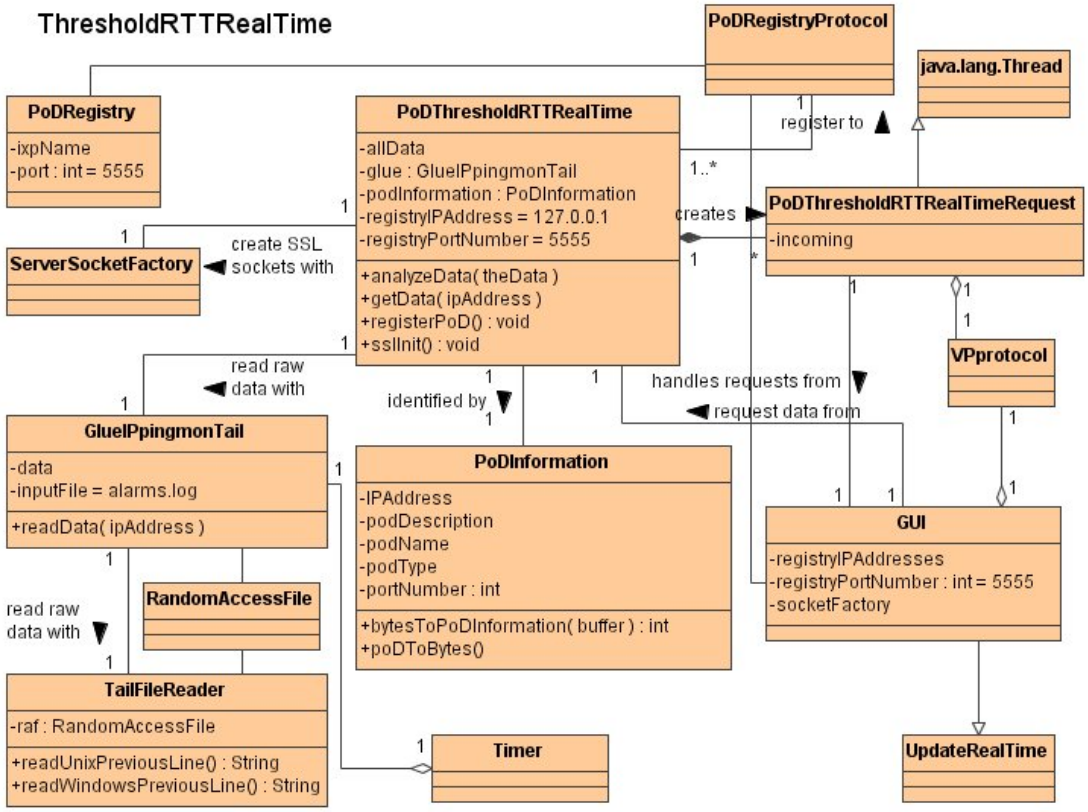


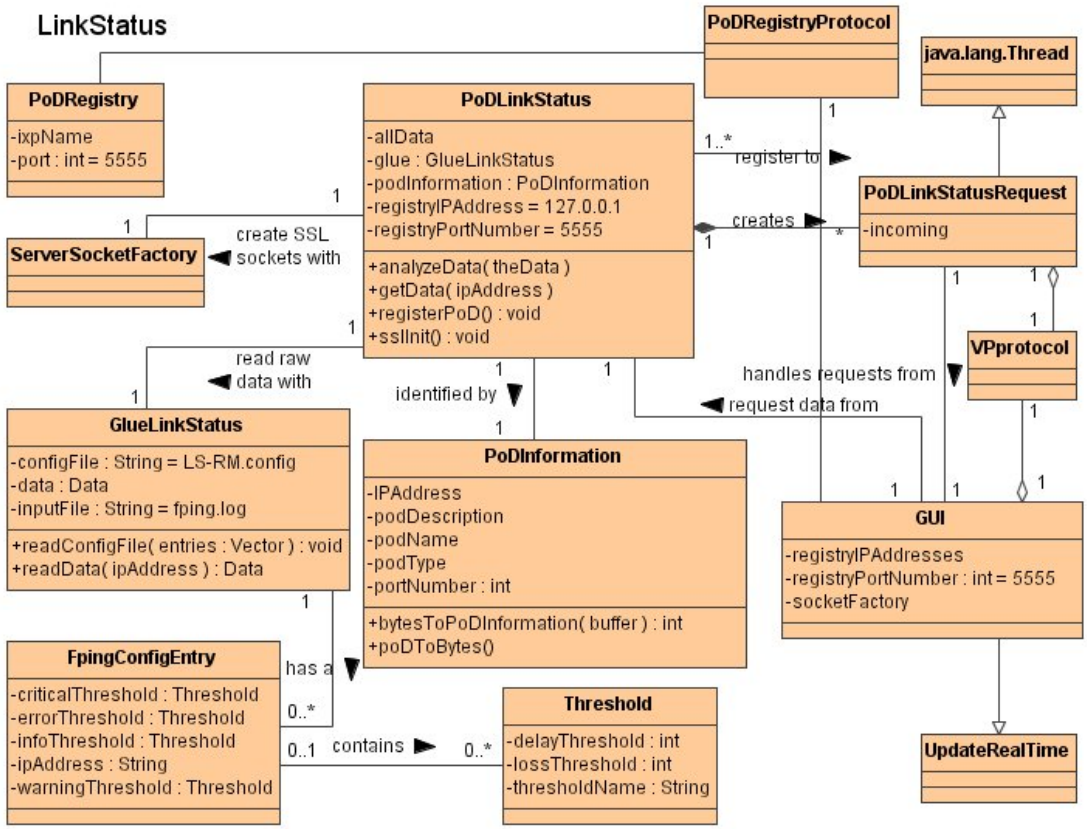
## RMFThresholdRTTRealTime

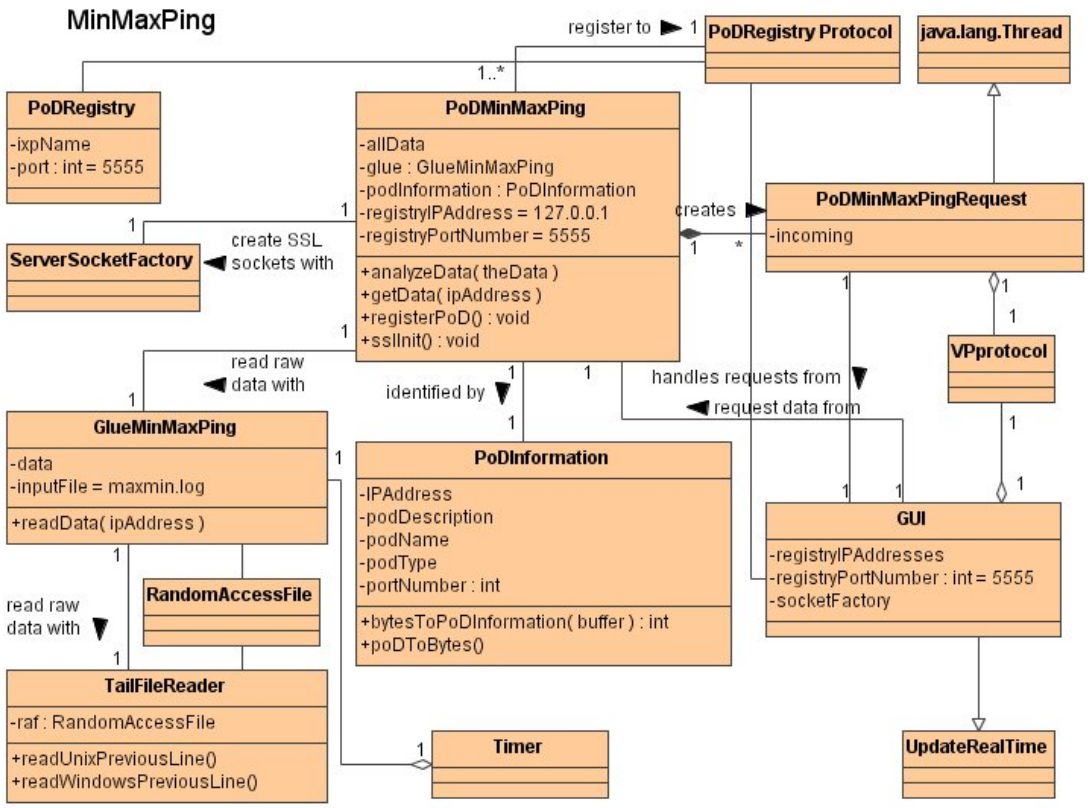



RTT vs. Time
PoDRegistryProtocol
java.lang.Thread
PoDRegistry
-ixpName : String
-port : int = 5555
PoDRTTvsTime
-allData : Vector
-glue : GlueIPpingmon
-podInformation : PoDInformation
-registryIPAddress : String = 127.0.0.1
-registryPortNumber : String = 5555
+analyzeData( theData : Data ) : Data
+getData( ipAddress : String ) : Data
+registerPoD() : void
+sslInit() : void
1..*
register to
PoDRTTvsTimeRequest
-incoming : Socket
creates
ServerSocketFactory
create SSL sockets with
read raw data with
identified by
handles requests from
VPprotocol
request data from
GlueIPpingmon
-data : Data
-inputFile : String = alarms.log
+readData( ipAddress : String ) : Data
PoDInformation
-IPAddress : String
-podDescription : String
-podName : String
-podType : GraphEnum
-portNumber : int
+bytesToPoDInformation( buffer : byte[] ) : int
+poDToBytes() : byte[]
GUI
-registryIPAddresses : String[]
-registryPortNumber : int = 5555
-socketFactory : SSLSocketFactory


ThresholdRTT
PoDRegistryProtocol
java.lang.Thread
PoDRegistry
-ixpName
-port : int = 5555
PoDThresholdRTT
-allData
-glue : GlueIPpingmon
-podInformation : PoDInformation
-registryIPAddress = 127.0.0.1
-registryPortNumber = 5555
+analyzeData( theData )
+getData( ipAddress )
+registerPoD() : void
+sslInit() : void
1..*
register to
PoDThresholdRTTRequest
-incoming
creates
ServerSocketFactory
create SSL sockets with
read raw data with
identified by
handles requests from
VPprotocol
request data from
GlueIPpingmon
-data
-inputFile = alarms.log
+readData( ipAddress )
PoDInformation
-IPAddress
-podDescription
-podName
-podType
-portNumber : int
+bytesToPoDInformation( buffer ) : int
+poDToBytes()
GUI
-registryIPAddresses
-registryPortNumber : int = 5555
-socketFactory

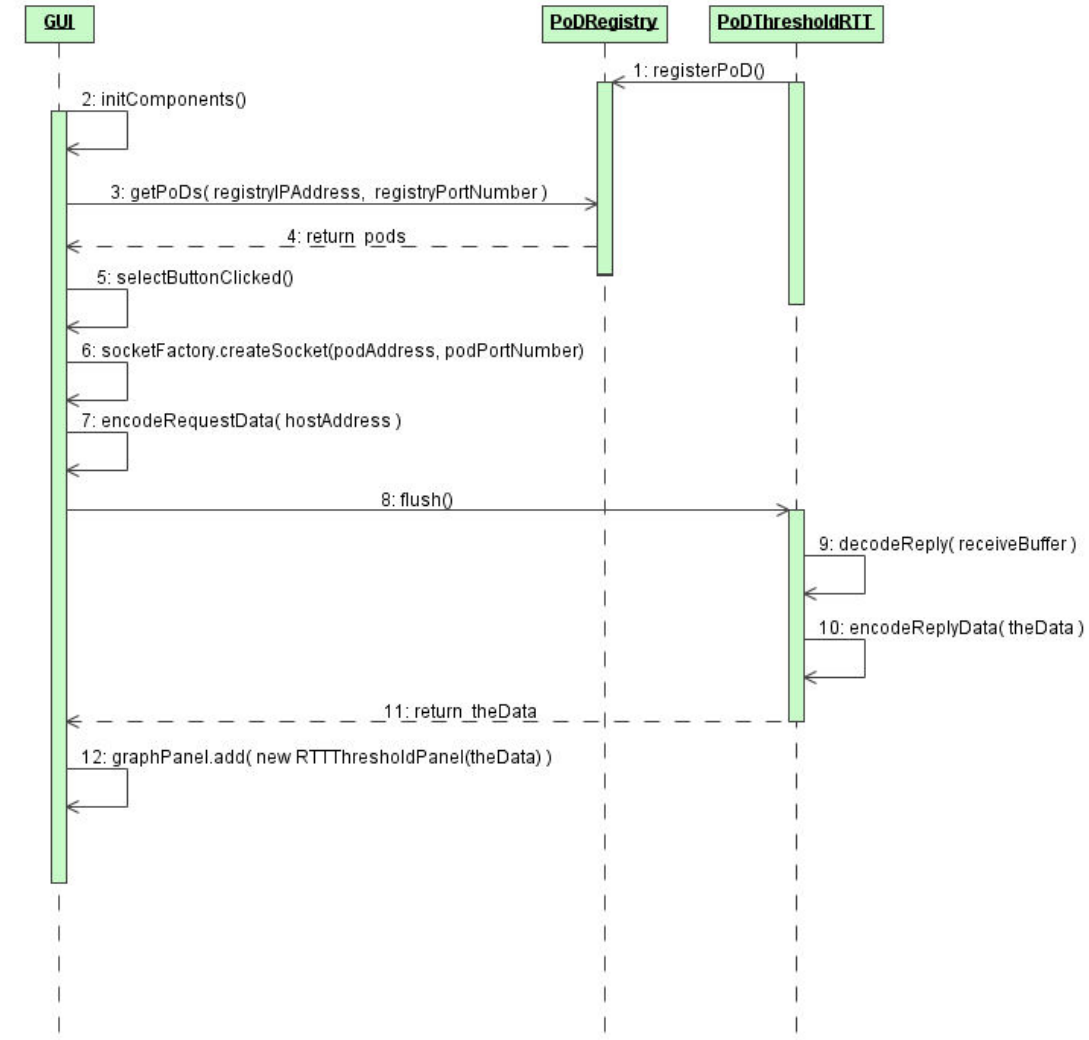

ThresholdRTTRealTime
PoDRegistryProtocol
java.lang.Thread
PoDRegistry
-ixpName
-port : int = 5555
PoDThresholdRTTRealTime
-allData
-glue : GlueIPpingmonTail
-podInformation : PoDInformation
-registryIPAddress = 127.0.0.1
-registryPortNumber = 5555
+analyzeData( theData )
+getData( ipAddress )
+registerPoD() : void
+sslInit() : void
register to
creates
PoDThresholdRTTRealTimeRequest
-incoming
ServerSocketFactory
create SSL sockets with
read raw data with
identified by
handles requests from
request data from
VPprotocol
GlueIPpingmonTail
-data
-inputFile = alarms.log
+readData( ipAddress )
PoDInformation
-IPAddress
-podDescription
-podName
-podType
-portNumber : int
+bytesToPoDInformation( buffer ) : int
+poDToBytes()
GUI
-registryIPAddresses
-registryPortNumber : int = 5555
-socketFactory
RandomAccessFile
read raw data with
TailFileReader
-raf : RandomAccessFile
+readUnixPreviousLine() : String
+readWindowsPreviousLine() : String
Timer
UpdateRealTime


LinkStatus
PoDRegistryProtocol
java.lang.Thread
PoDRegistry
-ixpName
-port : int = 5555
PoDLinkStatus
-allData
-glue : GlueLinkStatus
-podInformation : PoDInformation
-registryIPAddress = 127.0.0.1
-registryPortNumber = 5555
+analyzeData( theData )
+getData( ipAddress )
+registerPoD() : void
+sslInit() : void
register to
creates
PoDLinkStatusRequest
-incoming
ServerSocketFactory
create SSL sockets with
read raw data with
identified by
handles requests from
request data from
VPprotocol
GlueLinkStatus
-configFile : String = LS-RM.config
-data : Data
-inputFile : String = fping.log
+readConfigFile( entries : Vector ) : void
+readData( ipAddress ) : Data
PoDInformation
-IPAddress
-podDescription
-podName
-podType
-portNumber : int
+bytesToPoDInformation( buffer ) : int
+poDToBytes()
GUI
-registryIPAddresses
-registryPortNumber : int = 5555
-socketFactory
FpingConfigEntry
-criticalThreshold : Threshold
-errorThreshold : Threshold
-infoThreshold : Threshold
-ipAddress : String
-warningThreshold : Threshold
has a
contains
Threshold
-delayThreshold : int
-lossThreshold : int
-thresholdName : String
UpdateRealTime

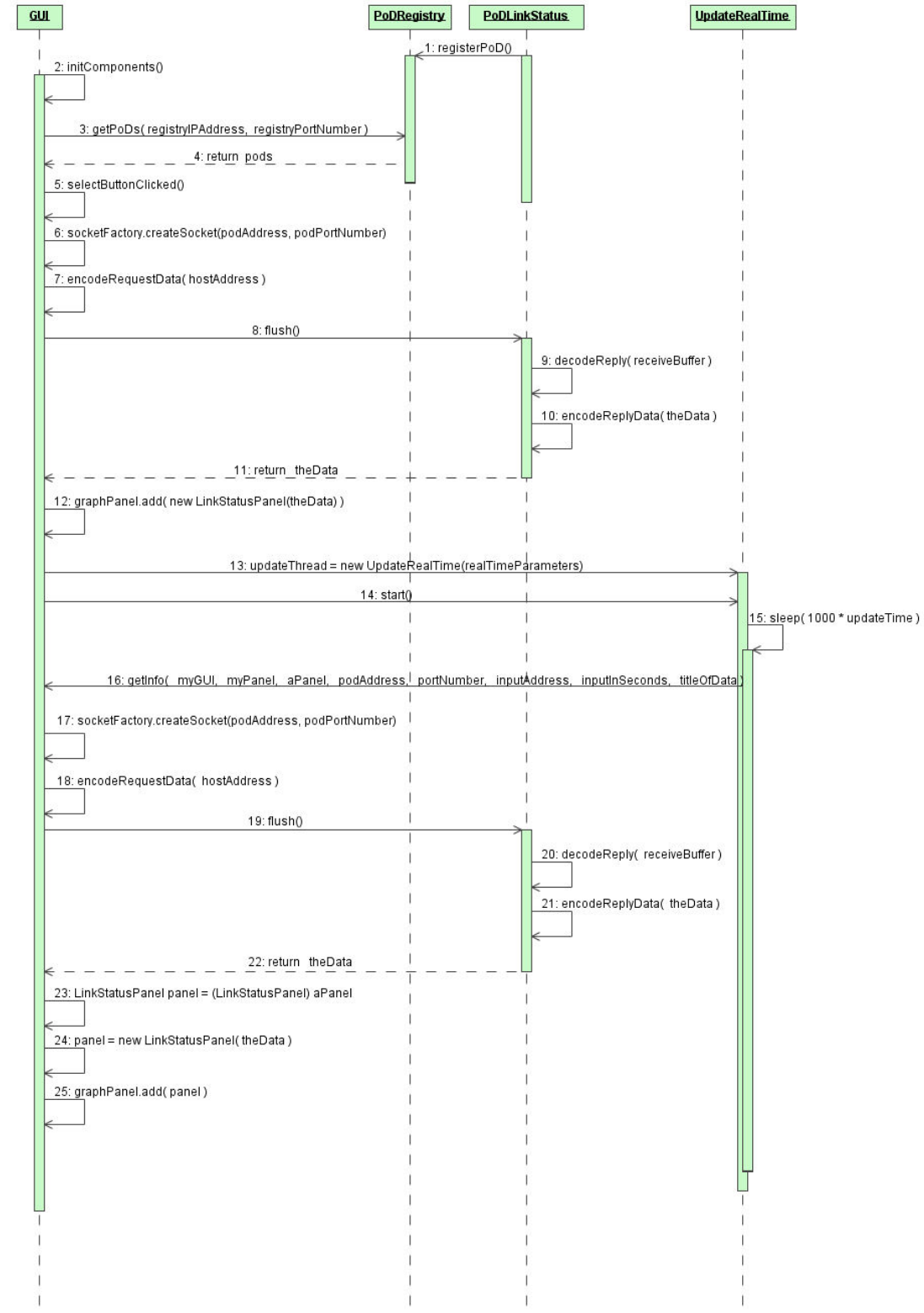

MinMaxPing
register to
1
PoDRegistry Protocol
java.lang.Thread
1..*
PoDRegistry
-ixpName
-port : int = 5555
PoDMinMaxPing
-allData
-glue : GlueMinMaxPing
-podInformation : PoDInformation
-registryIPAddress = 127.0.0.1
-registryPortNumber = 5555
+analyzeData( theData )
+getData( ipAddress )
+registerPoD() : void
+sslInit() : void
creates
PoDMinMaxPingRequest
-incoming
ServerSocketFactory
create SSL
sockets with
read raw
data with
identified by
handles requests from
request data from
VPprotocol
GlueMinMaxPing
-data
-inputFile = maxmin.log
+readData( ipAddress )
PoDInformation
-IPAddress
-podDescription
-podName
-podType
-portNumber : int
+bytesToPoDInformation( buffer ) : int
+poDToBytes()
GUI
-registryIPAddresses
-registryPortNumber : int = 5555
-socketFactory
RandomAccessFile
read raw
data with
TailFileReader
-raf : RandomAccessFile
+readUnixPreviousLine()
+readWindowsPreviousLine()
Timer
UpdateRealTime

# Appendix C: Sequence Diagrams

### Non-real Time Interaction



## Real-time Interaction



# Appendix D: Bibliography


Beck, Kent. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Pub Co, 1999.

"Case Studies: Active Measurements Data Analysis Techniques". Retrieved 10/07/04 http://moat.nlanr.net/Papers/AMP_case_studies/case_studies.pdf

Cohoon, James P., Davidson, Jack W. *C++ Program Design: An Introduction to Programming and Object-Oriented Design.* New York: Irwin/McGraw-Hill, 1997.

CppDoc Home Page . Retrieved 02/08/04. http://www.cppdoc.com/

Deitel, H. M. and Deitel, P. J. *Java, How to program, 3rd Edition*. Prentice-Hall Inc., 1999.

Dennis, Alan and Wixom, Barbara H. *Systems Analysis and Design.* John Wiley & Sons Inc, 2000.

Euro-IX Current Projects – IXP Network Monitoring. Retrieved 22/07/04 http://www.euro-ix.net/projects/approved/ixpnetwork.shtml

Euro-IX What is an IXP? Retrieved 22/07/04 http://www.euro-ix.net/about/whatis.shtml

Extreme Programming Homepage. Retrieved 08/08/04 http://www.extremeprogramming.org

Fping - A program to ping hosts in parallel. Retrieved 04/08/04. http://www.fping.com/

Harold, Elliotte Rusty. *Java Network Programming Second Edition.* Sebastopol: O'Reilly & Associates, Inc., 2000.

Horstmann, Cay S., Cornell, Gary. *Core Java Volume II: Advanced Features.* Palo Alto: Sun Microsystems Press, 2002.

IXP Network Monitoring Tool Portability Presentation by LINX. Retrieved 02/08/04 http://www.cs.ucl.ac.uk/teaching/dcnds/seminars/2003-11-25-linx.pdf

Jain, Raj. *The Art Of Computer Systems Performance Analysis*. Wiley, 1991.

Javadoc Tool Home Page. Retrieved 04/08/04. http://java.sun.com/j2se/javadoc/

Keytool - Key and Certificate Management Tool. Retrieved 05/08/04. http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html

Labanti, M., Rossi, V. from Milano Internet exchange; E. Glerean, N.Zingirian from University of Padova (IT); "*Inter Provider Network Analyzer*" Source: The London Internet Exchange.

SearchNetworking.com – Peering. Retrieved 27/07/04. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212768,00.html

Stallings W. *Cryptography and Network Security – Principles and Practices (Third Edition).* New Jersey (USA): Pearson Education Inc, 2003.

Taha, A. Hamdy. *Operations Research, Seventh Edition*. Prentice Hall, 2003.

Thomas, Stephen. *SSL and TLS Essentials.* New York: John Wiley & Sons, Inc., 2000.

Tufte, R. E. *The Visual Display of Quantitative Information, 16th printing*. Graphics Press, 1983.

Ujac – Useful Java Application Components. Retrieved 02/08/04. http://ujac.sourceforge.net/chart.html