# DCNDS Project:
# NRSE
# Network Resource Scheduling Entity

for multi-domain QoS reservation scheduling & QoS signaling
part of the GRID Resource Scheduling project

Andy Liow
Keiko Tada
Richard Smith
Toshihiro Aiyoshi

Supervisor
Saleem Bhatti

September 1, 2003

# Acknowledgements

We would like to acknowledge our supervisor, Saleem Bhatti, who provided invaluable support and advice. The project would not have been possible without the GRS group who conceived the NRSE architecture in [BSCC03]: Saleem Bhatti, Søren-Aksel Sørensen, Peter Clarke, Jon Crowcroft. We are also grateful to Yangcheng Huang, who worked on implementing the hardware-level part of the system.

# Summary

The GRS (GRID Resource Scheduling) project aims to enable GRID users to micro-manage QoS reservations at the edges of their networks on a per-flow basis. We have designed and implemented the NRSE (Network Resource Scheduling Entity) component of GRS. The NRSE provides a service interface, specifying an XML-based signalling protocol for service level requests and notifications. The service levels provided are a subset of IETF DIFFSERV; the NRSE requires a router that implements DIFFSERV. The NRSE also performs admission control functions to ensure there is sufficient capacity at the local router for every reservation. Reservations across multiple domains are made using two NRSEs, one at each end, with the assumption that the core network is over-provisioned.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 The problem

Packet based IP networks are inherently unreliable. They only provide "best effort" service, which means they do not guarantee that a packet will be delivered at all, let alone delivered on time. Protocols used by the end systems, such as TCP/IP, are able to mitigate this problem somewhat, and provide some basic assurance that packets will eventually arrive.

Unfortunately, this is not good enough for many applications. For example, TCP cannot be used for realtime video communications, because the TCP retransmissions will introduce unacceptable jitter which can only be avoided by equally unacceptable buffering delays. Instead we use protocols built on UDP that are able to tolerate some lost packets. Nevertheless, we can only tolerate so much loss, and we would like the network to provide a guaranteed quality of service (QoS). We would like the network to guarantee that is has sufficient bandwidth for our video stream, that it will not drop more than a certain percentage of packets, that our packets will be delivered with less than a certain delay, etc.

Another example, more familiar to most users, is that of file transfer. In this case we don't care about packet loss or delay. We only care that there is sufficient bandwidth available to transfer our file within a certain amount of time. However, we are more flexible than the realtime user, because we can vary the datarate to suit the needs of the network. We might be able to accept a guarantee of very high bandwidth for a short period, rather than mediocre bandwidth for a longer period, as long as the file transfer finishes before our deadline.

The problem of guaranteeing QoS in IP based networks is fairly well understood. The IETF first attempted to solve it via INTSERV, which allowed

the user to reserve bandwidth on a per-flow basis, but unfortunately required all routers along the path to support INTSERV. For this and other reasons it has not been widely adopted.

DIFFSERV is a more lightweight approach that marks packets with a tag as they leave the network to indicate the service level they should receive. The problem with this is that once a DIFFSERV service has been configured by the administrator, all flows are aggregated together, and it is not possible for him to allocate bandwidth to individual user flows as they need it.

GARA is a general purpose resource allocation protocol, but it is not specialised for network resources, and we hope we can improve on it.

INTSERV, DIFFSERV and GARA are examined in more detail below. (These descriptions are taken from those in [BSCC03]).

## 1.2 The state of the art

### 1.2.1 INTSERV (Integrated Services)

The present IP services uses a best-effort approach. Using the IETF INTSERV and the signalling protocol RSVP(reservation Protocol), two service-level specifications are defined, namely controlled-load service, and guaranteed service. A controlled-load service approximates the QoS received from an unloaded, best effort network and a guaranteed service delivers a guaranteed throughput and delay for a flow. RSVP is a signalling protocol that allows messages to be sent between applications, which are then used by the network elements en-route. INTSERV, together with the use of RSVP, has its disadvantages. The Qos guarantee is provided for a single flow. A lot of soft state has to be held for each flow. The effect is compounded as more flows pass through the routers and goes into the core network. There are some other disadvantages, just to name a few, like the extra traffic generated due to the soft state refreshes and the possibility of router failures.

### 1.2.2 DIFFSERV (Differentiated Services)

DIFFSERV is a class based system where packets are marked with a well-known value. It treats marked packets with a different QoS and allows flows to be aggregated. Aggregated flows receive the same service level producing a coarser granularity of service. Packets within the same class share the same resources. Service classes are divided into Premium and Assured classes. Premium service provides a low delay service using EF (Expedited Forwarding). AF (Assured Forwarding) service provides a low loss service using As-

sured Forwarding with different drop precedence assignments. DIFFSERV attempts to provide a simpler and coarse-grained Qos control mechanism without the need to change the end systems. Whereas INTSERV provides a per application/flow end-to-end resource reservation, DIFFSERV aims to provide an SLA-based contract between service networks.

### 1.2.3 GARA (General-purpose Architecture for Reservation and Allocation)

GARA is one of the projects undertaken for QoS reservations. It is implemented by the Globus Project to integrate Qos into Globus even though it only lightly relies on Globus. It is a general purpose platform used to reserve numerous resources including disk space and CPU cycles. It is widely in used and it offers some core network resource reservation. It has two advantages. Firstly, Qos reservations can be made in advance or immediately at the time you need it. Secondly, you can use the same API to make and monitor a reservation regardless of the type of the underlying resource, thus simplifying the implementation of Qos reservations of multiple types of resources. However, it is not without limitations. Some network resources like disk space are fundamentally very different from network capacity. These resources are localised to certain end-systems and reservations can be made at the remote end-systems where such resources are located. Network capacity is a distributed resource requiring reservations at the local and remote end-systems as well as the network path between the local and remote systems.

## 1.3 Our solution

In [BSCC03], the GRID Resource Scheduling project proposes a system to solve the QoS scheduling problem. The NRSE architecture uses the existing DIFFSERV EF service. DIFFSERV can be thought of as a virtual pipe of protected capacity, and the NRSE is an agent that allows users to micro-manage individual flows within that pipe. The NRSE also acts as a 'bandwidth broker', allocating a portion of the DIFFSERV pipe to a user's flow on-demand, something that is not practical for an administrator to do manually. It has a similar role to GARA, but without GARA's limitations. The NRSE is able to automatically negotiate a multi-domain reservation by communicating with its counterpart on the remote network, on behalf of its client. Thus the system is highly scalable.

Figure 1.1 provides an overview of the system architecture.

An integrated services architecture requires four components ([BSCC03]). The NRSE is positioned in the integrated services model as follows.

### 1.3.1 Service level

The service levels supported by the NRSE are provided by the DIFFSERV service, implemented in the routers. NRSE users chose from a subset of DIFFSERV parameters. Therefore defining new service levels is not part of the NRSE project - we are merely using what already exists.

### 1.3.2 Service interface

The NRSE provides a signalling protocol. This is an open, human-readable protocol based on XML that enables users to request that the NRSE add, modify and delete QoS reservations. This client-NRSE protocol is represented by the red arrows on the diagram.

### 1.3.3 Admission control

The NRSE also provides admission control. Each NRSE maintains a database of reservations and only accepts a new reservation if it has sufficient band-

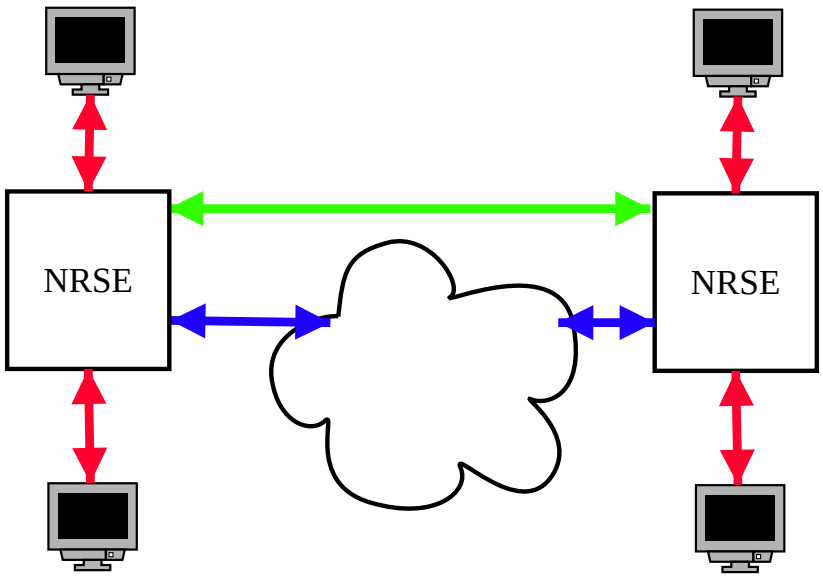


Figure 1.1: NRSE architecture

width unallocated. For non-realtime requests, such as file transfers, it may suggest an alternative booking that still results in the file being transfered before the user's specified deadline.

The NRSE must also have an inter-NRSE protocol, which should be as similar to the client protocol as possible, and this represented by the green arrows on the diagram. This is necessary because each subnet has its own NRSE, and admission control must be performed at both the source and destination networks. (Having the client communicate with both local and remote NRSEs would not scale well, because every NRSE would need to be able to authenticate every client.)

### 1.3.4 Scheduling mechanisms within the network

The NRSE does *not* provide scheduling mechanisms within the network. It relies on the routers of the networks supporting DIFFSERV. However, the NRSE does instruct the network on what QoS filtering should be performed. The blue arrow on the diagram represent the NRSE informing the network router of the DIFFSERV parameters required for a particular reservation.

For more information on this design, see [BSCC03].

# Chapter 2

# Requirements Analysis

From the RUP perspective, there are three 'actors' in our system.

**GRID user** - runs our client software to make reservations with his local NRSE.

**NRSE administrator** - installs, configures and administrates an NRSE on his network.

**GRID programmer** - creates GRID applications that are able to make reservations with the local NRSE.

The following functional requirements are based on those in the [BSCC03] document, which contains a specification of the system wanted by our client. We have arranged the requirements according to actor.

We did not prioritise requirements initially. Prioritisation was performed later, by the customer (our supervisor) during 'the planning game' in each development phase. The programmers estimated how much work it would take to meet each requirement and the customer decided how much 'business value' each feature would create.

## 2.1 GRID user

- The user is able to make reservations of network capacity across multiple domains based on SLAs.
- The user may delete and modify his own reservations.
- Reservations for non-realtime service-requests are flexible, i.e. the NRSE will adjust them to meet specified deadlines.

- The NRSE uses a localised polling mechanism for the application holding the reservation (keep-alive) so that resources can be reclaimed in an application fails.
- The user may elect to receive notification of violations of the SLA.
- The user may specify service class, directionality and policing in the SLA.
- The user specifies network flows using standard TCP/IP header fields.
- The system scales to a large number of NRSEs and users.
- The system is portable across a large number of different platforms.

## 2.2 NRSE administrator

- The administrator may query, modify and delete users' reservations.
- The administrator may add, modify and delete entries in the database of users.
- The system has a hierarchical trust model, so access control mechanisms are localised. This means the administrator only needs to configure access for his own users, and for remote NRSEs, but not for remote users.
- The administrator can configure local policies that control operation of his NRSE, with such policies being autonomously managed.

## 2.3 GRID programmer

- The format used by the client-NRSE protocol is open and well-defined to enable third-party re-implementations.
- Client code is separated into its own module so that it can be called from other applications.
- It is easy for application to receive notifications from the NRSE.

## 2.4 Non-functional requirements

- In order to scale well, the NRSE stores reservation state in a decentralised manner. Reservation state is only held at end-sites which are involved in the reservation.
- The NRSE should run on as many platforms as possible. Therefore the customer has recommended it is written in Java.
- The customer has suggested the use of XML and BEEP (see [Ros01])for the protocol, to meet the openness requirements.
- The customer has specified the format of some the XML documents. (See Appendix E on page 82).
- The program must be tested on the provided testbed, and therefore uses Linux routers. Any Linux-specific code is kept in a module accessed via a well defined interface, so it may be easily replaced with code for another router operating system, such as Cisco IOS.

## 2.5 Resource Management

There follows an overview of the tools and technologies used in the project. Some of these were specified in the requirements, some were recommended by our supervisor, and some were chosen by us.

### 2.5.1 Linux Traffic Control

Our project coordinates substantially with another student Yang Cheng's project, even though it was designed to operate without conflicts if either one of the two projects was modified. Yang Cheng's project implements an EF service with a chosen priority queueing mechanisms from Linux DIFFSERV/ Traffic Control. Traffic Control is chosen as specified by our supervisor and at the same time was developed by another student. Other operating system routing are also available for routing the packets through the network, e.g. Solaris, FreeBSD, IOS. To list a few advantages of using Linux TC for routing the network packets, TC is fast, mature, and small in size.

### 2.5.2 XML and Schemas

The clients and the NRSE communicate and establish a SLS between them. A Qos request comes in the form of a SLA. These SLA are valid and well-formed

XML documents. XML is used for the application-level signalling protocol, used for managing the SLS as defined by DIFFSERV. XML provides for the integration and collation of any data and information irrespective of storage environment or document type. It is in fact the industry standard for human-readable markup language. There are many documents that have to be sent between the clients and NRSE, e.g. Notifications and Add user. They are all XML documents. Schemas have been written to test the data structure of the XML documents.

### 2.5.3 Apache Xerces XML parser

Apache Xerces java package have been utilised to parse the XML elements and serialising them so that elements in the XML elements can be read, validated and verified by the program and stored as objects.

### 2.5.4 Java GUI, JUnit, Java and many of its related packages

Java GUI is manually implemented with its own AWT and Swing packages without using any other GUI constructing toolkit. We think that the whole layout and structure of the GUI can be better controlled and understood this way even though it takes much more time to implement it. JUnit is being used to test the program code developed. JUnit has a regression testing framework that facilitates programmers to implement unit tests in Java. Tests can be written before the application code, which can lead to code that is both tested and only loosely coupled.

### 2.5.5 PGP authentication

PGP is used for the authentication of the users. PGP can be used to establish the confidentiality of the communications between users, guarantee the reliability of the source information, and guarantee the message integrity. The project uses PGP to generate a set of private and public keys for each user for authentication purposes. The private keys will be stored as credentials of the users in the database. Other security alternatives have been taken into account of like BEEP, Java, and OpenSSL X509 certificates. PGP is chosen because one of our group members is familiar with it and it will be easier and faster to implement in PGP compared to the others. Our project is also programmed in such a way that alternative authentication method is also allowed.

### 2.5.6 PostgreSQL database

PostgreSQL database is being used to store the data created by the project. Initially, we thought that We can use PostgreSQL and then port to MySQL later in the project if possible, or port to Oracle is possible. MySQL was slightly more desired by our supervisor, but we continued with PostgreSQL since we have already started on it. We tried to keep SQL statements to conform to SQL99 so that porting to other databases will be easier.

### 2.5.7 GNU XEmacs editor

GNU XEmacs was chosen as the editor as it is a highly customisable open source text editor and application development system. Its emphasis is on modern graphical user interface support and an open software development model, similar to Linux.

### 2.5.8 Apache Jakarta log4j

Apache Jakarta log4j is being used for enabling logging at runtime without modifying the application program. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behaviour can be controlled by editing a configuration file, without touching the application program.

### 2.5.9 UML tools

Simple UML diagrams like use cases and sequence diagrams are being used for specifying, visualising, constructing, and documenting the artifacts of software systems. It attempts to simplify the complex process of our software design, making a ”blueprint” for construction. However, it is not extensively used since XP methodology approach does not emphasise on using UML. UML diagrams are used to guide us along our design of the project. We are using Visio and Dia to produce these diagrams.

### 2.5.10 Mr Project project management

Mr Project was selected to produced a Gantt chart for our project to show the schedule for the group. Mr Project is simple enough to use and is sufficient enough for our project development. Moreover, it comes with the Linux system which facilitates development and viewing without having to install additional project management tools.

### 2.5.11 CVS

CVS is used for version control throughout our project. Work can be delegated to different member working on different files simultaneously without having file conflicts. Any undesirable effects can be undone by reversing the work to a previous version of the file or work. CVS is largely used for such purposes and we made a few releases with it.

### 2.5.12 BEEP

BEEP is used to transmit messages between the NRSE and clients for communication through SLA transmission. BEEP is generally used for the application-level communications using the Beepcore-java library. This was also specified by our supervisor. There are other message exchange protocol for exchanging messages in the transport layer like HTTP. HTTP is already deployed and is mature in many ways. However, BEEP is more sophisticated compared to HTTP. BEEP is more suited for transport for XML-structured messages or documents. BEEP can carries multiple channels on the same TCP connection and the channels are independent of each other. Messages can be originated by either peer. Most importantly, BEEP is an IETF specification, many more IETF specs based on BEEP will emerge soon in the future.

# Chapter 3

# Project Management

## 3.1 Methodology

We chose the Extreme Programming (XP) methodology to manage our project. XP is an agile, lightweight methodology that is well suited to a small team working on a small project, because it dispenses with much of the paperwork that is necessary to manage a large project but which would be an unnecessary burden on a team as small as ours.

Further information on how the project was managed can be found in individual reports.

## 3.2 Schedule

We planned and scheduled the project work using the MrProject tool to produce Gantt charts. These were constantly refined and updated through the project. The final Gantt chart is shown in figure 3.1

We initially planned to have four phases of development with one or two major milestones to mark the end of each phase. (These are not shown on the diagram because there wasn't space.)

**Phase 1** Define protocol. Choose which aspects of DIFFSERV SLS to support and and an XML format. Define a protocol for the client to communicate with the NRSE. An inter-NRSE protocol will be defined later.[1]

[1]This was not actually necessary, because we were able to define a single protocol for both functions.

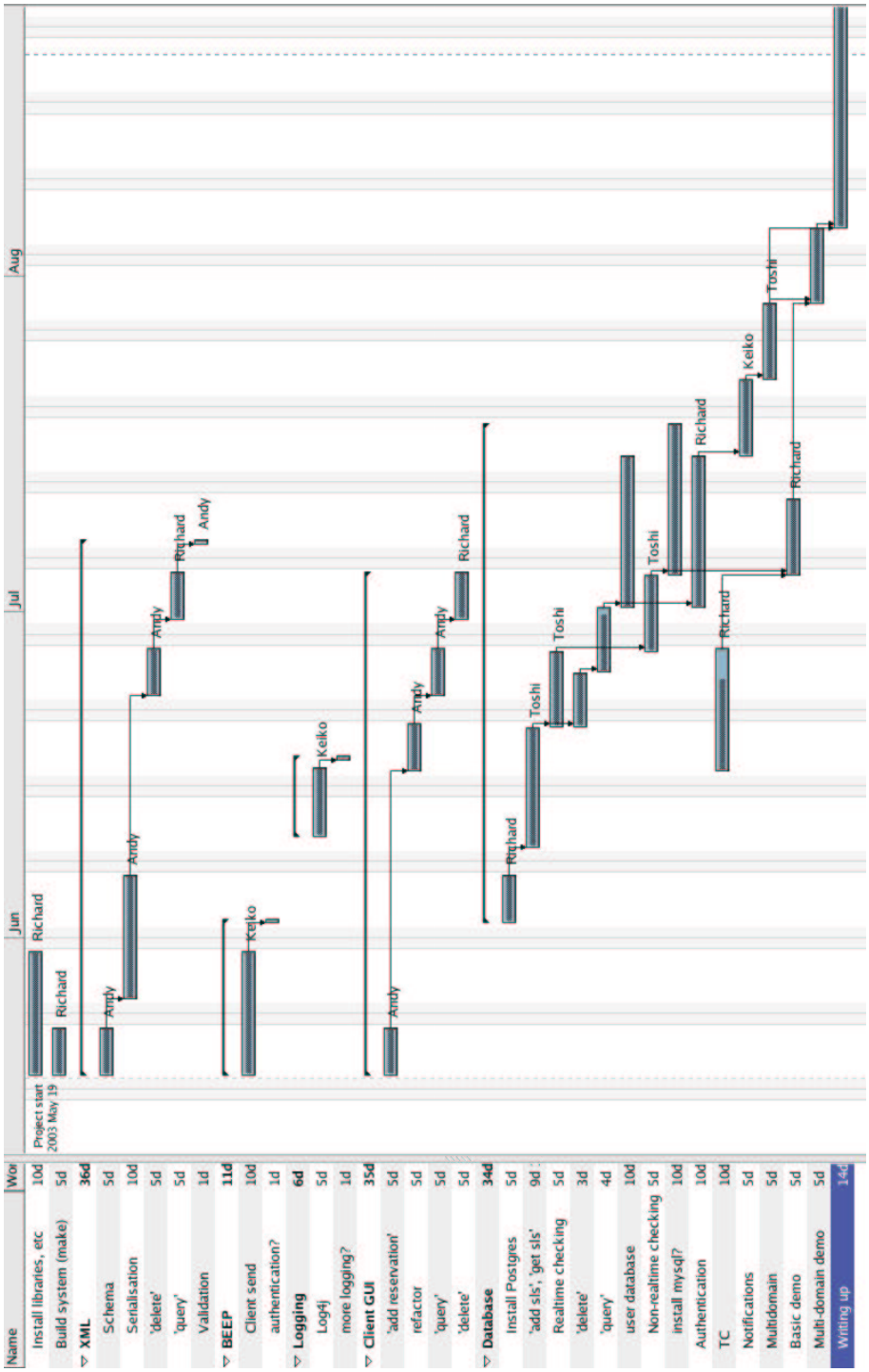


Figure 3.1: Gantt chart

**Phase 2** Single domain scenario. Implement the NRSE to work for a single domain. A client must also be developed concurrently. Concurrent engineering will continue throughout the remainder of the project. The client application will create, delete and edit SLAs. The first milestone is met when the two applications can communicate using the protocol specified in Phase 1 and agree a SLA. For the second milestone, the NRSE must activate a token bucket filter[2] on the router and perform admission control. This is the largest and most risky phase, and determines the feasibility of the rest of the project.

**Phase 3** Multi-domain homogeneous scenario. Extend the system so that two NRSE can negotiate a reservation across multiple domains. Other more advanced features may also be added at this stage, such as authentication, and a GUI for the client application[3].

**Phase 4** Multi-domain heterogeneous Scenario.[4]

## 3.3 CMM

We are aimed to reach CMM level 4. Unfortunately, the XP methodology does not map easily into the CMM level system, so we achieved a 'vertical slice' of targets with some from each of the levels. A thorough discussion of where XP fits into the CMM scale can be found in [Jef00].

## 3.4 Risk Management

In the process of developing a software, one of the key factors for success is risk management. In our project, expectations of the results and deliverables are noted and compared with the actual results. In other words, measures of the possibility of deviation from the expected are done. Before doing this, the risks and hazards of the project have to be defined first, acting as a discipline for helping us to deal with the uncertainties that may arise.

Likelihood of a risk multiplied by its impact can be a useful metric for assessing a risk but we must not neglect very dangerous risks simply because they rarely occur. We deal with the most likely to occur risks and most important risks first.

[2]The requirements were later amended so that this was no longer part of our project. It was instead to be implemented by Yangcheng Huang.

[3]Actually these features were added much earlier

[4]This phase was later dropped since we did not have a heterogeneous testbed available.

The possible risks foreseen will be described as follows.

### 3.4.1 Schedule delay

Projects have deadlines but are often delayed as we noticed from real project developments. Failing to produce the required deliverables can have serious consequences. Extreme programming helps to mitigate risks because of its flexibility to overcome unforeseen problems by developing the project incrementally. As a general rule, we have done the most important work first. Less important work are left out or placed less effort in it, trading off for more time dealing with the more important work. Our key risk management was to make the risk cards(see section **??**User Stories)). Every task that should be implemented was written on the task card and we estimate the risks of each card. Furthermore, we allow more time than possible for some components rather than trying very hard to meet tight deadlines which is unrealistic. We allow time for some delays since every member will have their difficulty in dealing with their own time management.

### 3.4.2 Cost Overrun

Cost does not affect our project much but it is worth mentioning since it is always an important component in every project. Our project uses many of the equipments and software that are readily available to us, including some open source software. Equipments and materials are available from the department, our supervisor, and the Internet without incurring additional costs. But using all the software bundled together leads to another problem being addressed in the next section.

### 3.4.3 Intellectual Property Right

Software copyright are mainly concerned here. We use many open source software with the licenses available to the public. However, using them altogether as a whole for this project seems inappropriate. However, since this project is sponsored by the university and is not disclosed to the public, we can use the software as we like for now. Work being referenced are noted down in the report to avoid infringing intellectual property rights as well.

### 3.4.4 Specification Comprehension

Specification Comprehension is crucial to avoid deviating from the main objective. For this, group discussions are often held and assistance are often

offered for certain aspects of the project. In resolving design issues, we discuss and agreed upon certain issues. On implementing it, comprehension of what the program component should do exactly is important. On the project as a whole, a global perspective is crucial to help to understand what we should actually achieve. Help was sought from another member when confusion or problems arose. Frequent communication is essential and was done for understanding and resolving issues.

### 3.4.5 Specification Changes

As the specification changes in the project, we tried to refactor to the changes in development according to our supervisor specifications. As we were using XP programming methodology by developing the project incrementally, we mitigate this risk of changing requirements. We always tried to make some workable programs before moving onto other programs. Eventually, we made a few working releases with this method. in this way, we always have something that is working at any point in time. If this was not done, we can make a lot of programs without designing and programming it properly, many design issues and program bugs will arise. Redesigning it can then be a big problem and debugging becomes extremely tedious since we do not know where the bug could be in the many programs already developed.

### 3.4.6 Existing software bugs

We have used many software to produce this final project. The software we utilised are not perfect and there are existing bugs and failures in certain parts of it. For example, we used many fixed Java libraries like the BEEP, Log4J, PostgreSQL database. An attempt on implementing BEEP authentication was done but failed since we could only get a poor manual on it. Eventually, we changed our plan and used PGP instead since one of our members is familiar with it and authentication is not as important as other issues we were dealing with. This also saves time which is linked to the schedule delay risk as mentioned. These problems encountered are being documented in the latter section (see section **??**Evaluation)).

### 3.4.7 Testbed

The Testbed is being developed by a PhD student called Yang Cheng. Collaboration is needed to integrate the two different systems even though we programmed the project in a ways such that there is an interface in between the two systems. This interface means that at any point in time, a different

system developed can replaced one of the two current systems and still works in collaboration successfully. Whoever, we were not sure about the availability of the testbed and when we can get to utilise it. In this case, we made a test environment in the CS department. Currently, it is possible to run the program in both environments by changing some configuration parameters and options.

### 3.4.8 Individual skills and experiences

Each of our members skills vary. This happens to all the groups, affecting the project development with the different skills required and the different skills available to develop it. The same applies to the experience of each team member. Some of us are more familiar with C programming instead of Java, whereas others are more familiar with Java. Some of us are more familiar with Windows programming environment whereas others are more familiar with Linux programming environment. XP programming mitigate the risks by allowing pair programming. As a result, the skills required to develop the project are resolved to a certain extent, with the benefit that the less familiar programmer in a certain aspect can learn from a more familiar programmer in the same aspect by carrying out pair programming.

# Chapter 4

# Design

## 4.1 Metaphor

In Extreme Programming, there is no lengthy design phase before programming can begin. We simply chose a metaphor to explain how the system will behave, then begin the first phase of writing user stories (explained below), tests and code.

We chose the simple metaphor of an airline booking agent. The agent receives requests from passengers who specify where they want to fly from and to (source and destination IP addresses) and what time they want to fly. The passenger also specifies what class of seat he would like (QoS service level). The agent checks whether there are sufficient seats available on the flight. If there are, he issues the tickets. If the flight is fully booked, but passenger's journey is not urgent (non-realtime), the agent may be able to suggest an alternative flight. If the agent has to contact another agent to arrange the tickets, this is done transparently without involving the customer.

The travel agent is not concerned with actually flying the aeroplanes, just as we are not concerned with how the router meets our QoS specifications.

## 4.2 User Stories

In Extreme Programming we specify the functional requirements of the system by writing user stories. In a commercial project there would be complicated procedures governing the creation of these stories. For example, the stories might be written by the customer with the help of the programmers. The programmers would then estimate how long it would take to implement the functionality in each story, and the customer would prioritise the stories according to which provide the most 'business value'. There are different

variation on this 'planning game'.

Each story describes one feature that the system should possess. The stories were written onto 'task cards' and each card was given to the programmer responsible for implementing that story. If the programmer estimated the task would take a long time, it was broken up into several sub-tasks. The programmer would usually begin by writing a unit test for the functionality of the story. Then he would implement the story, and he would know that his implementation was correct as soon as it passed the unit test. Then he would mark the task card 'completed'.

In our case we were fortunate to be working on an academic project. We had a supervisor rather a real customer [1] and so initially we wrote user stories based on our interpretation of the requirements document given to us by our customer-supervisor, and set our own priorities in conjunction with our supervisor. XP is a living, iterative process and so the stories frequently changed during implementation. Sometimes we required clarification of how a requirement should be implemented, or our supervisor would notice that one of our prototypes did not behave as he had envisioned, so then he would request changes to the stories. Towards the end of the project when the deadline was approaching, we did prioritise which stories would make it into the final build according to the customer's wishes.

## 4.3 Usecases

Readers familiar with UML and usecases should note that there are some important differences between usecases and user stories.

- Stories are smaller and more specific than usecases - a story must be implemented within a few days.
- Stories do not model the concept of actors. On a system of this small scale it should be obvious who is doing what.
- Stories are modified during implementation
- New stories are created to add to and improve on the functionality of already-implemented stories. For example, the GUI story specifies the creation of a basic GUI as a feature of the client, but later stories improve the GUI with more features.

[1] This was very different from a business customer who would be like to sue us if we didn't deliver the project on schedule

As part of our design work, we drew a usecase diagram, which, for completeness, can be seen in Appendix F. However, we went on to develop XP user stories, and we did not continue with usecase development.

## 4.4 Task cards

Here we present the contents of the task cards we made from the user stories. Please note that we programmed in pairs. Only one partner from each pair was listed as responsible for each task, even if in practice the other partner did a greater share of the work.

### 4.4.1 XML Schema Design

Write a schema to describe the XML format of a SLS request. The NRSE could use this schema to validate the XML.

**Assigned to:** Andy

**Start:** 12/5/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.2 BEEP functionality

Make a simple test program to send and receive BEEP messages which can be used as the basis for NRSEProfile. (This is prerequisite for task 4.4.6.)

**Assigned to:** Keiko

**Start:** 19/5/03

**Estimate:** 10 days

**Completed:** 10 days

**Notes:**

### 4.4.3 GUI

The client application has a GUI in addition to console-based operation.

Initially, the GUI provides the user with these abilities:

- Add reservation
- Delete reservation
- Modify reservation

**Assigned to:** Andy

**Start:** 19/5/03

**Estimate:** 5 days

**Completed:** 10 days

**Notes:** Modify function was deemed unnecessary at this stage.

### 4.4.4 GUI Query

GUI can send a simple query to the NRSE, and display the results. One of the results may be selected for deletion. This function will subsequently be useful for reservation modification as well.

**Assigned to:** Andy

**Start:** 20/6/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.5 GUI Add User

GUI is capable of adding a user to the database. Initially this will be direct access via the Database class, but later administration functions should be handled through the NRSE.

**Assigned to:** Andy

**Start:** 16/6/03

**Estimate:** 5 days

**Completed:** 8 days

**Notes:**

### 4.4.6 Request SLA

User sends SLS to NRSE via BEEP which replies with 'OK'.

This requires the creation of a simple console-based client which is able to open a BEEP channel to the NRSE's BEEP server. An empty message is sent (later this will contain the SLS) and the NRSE's BEEP response of 'OK' is displayed to the user.

**Assigned to:** Keiko

**Start:** 19/5/03

**Estimate:** 5 days

**Completed:** 10 days

**Notes:** Modify function was deemed unnecessary at this stage.

### 4.4.7 Process SLS

When the NRSE receives the user's SLS, it no longer automatically replies with 'OK'. Instead it creates an SLS object and runs TC to implement the reservation. The response sent back to the client depends on the success of the operation.

This task builds upon the Request SLA task, and when these two tasks have been completed, the first milestone has been met.

#### Client produces XML

The console based client is able to take a reservation that has been input by the user and convert it into an XML document. Now, when the client sends a BEEP message to the server, the message contains this document, rather than being empty as it was before.

**Assigned to:** Richard

**Start:** 26/5/03

**Estimate:** 5 days

**Completed:** 10 days

**Notes:**

### NRSE reads XML

When the NRSE receives a request from the client containing a SLS in XML format, it parses the SLS and constructs a SLS object.

**Assigned to:** Andy

**Start:** 26/5/03

**Estimate:** 10 days

**Completed:** 10 days

**Notes:**

### NRSE runs TC

After creating the SLS object, the NRSE runs the Linux TC command with data from SLS object, thereby activating the reservation. Note that at this stage of development reservations are activated immediately rather than stored in a database. Also, there is a requirement that the NRSE is running on the same machine as the router.

**Assigned to:** Richard

**Start:** 15/6/03

**Estimate:** 5 days

**Completed:** 7 days

**Notes:**

### 4.4.8 Software installation

There are some software packages which are likely to require but which are not pre-installed, so we must download, compile and install them ourselves.

- PostgreSQL
- Xemacs
- Dia 0.91
- MrProject
- GTK+ 2.2
- Jikes

**Assigned to:** Richard

**Start:** 19/5/03

**Estimate:** 5 days

**Completed:** 10 days

**Notes:** Dia and MrProject could not be installed because they will not run on a 256 colour display, and unfortunately all the machines at UCL are limited to 256 colours.

### 4.4.9 NRSE delete SLS

The NRSE will process XML requests to delete reservations.

**Assigned to:** Richard

**Start:** 1/7/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:** .

### 4.4.10 Logging

All actions of the NRSE will be logged. It will be possible for the administrator to configure the level of logging required. During development, the programmers will want to see all actions. During production, only errors will be logged. The log4j package ought to be able to provide this functionality.

**Assigned to:** Keiko

**Start:** 10/6/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.11 BEEP Authentication

Investigate and implement authentication of user requests using BEEP's TLS feature.

**Assigned to:** Keiko

**Start:** 29/6/03

**Estimate:** 5 days

**Completed:**

**Notes:** Task was abandoned and rewritten. See following task and also section 7.3 on page 65.

### 4.4.12 PGP Authentication

Database stores usernames and PGP public keys. (For convenience, the client program should be able to generate keys when a user is added to database.) When the user sends a SLS request, he enters his PGP private key and password. The client uses these to generate a signature which is added to the XML document when the SLS is serialised. On receiving the request, the NRSE looks up the user in its database, locates his public key, and uses this to validate the signature. The SLS will only be added to the database if the signature is valid.

**Assigned to:** Richard

**Start:** 7/7/03

**Estimate:** 5 days

**Completed:** 10 days

**Notes:**

### 4.4.13 Notification

If requested by the user, the NRSE will send a notification message via BEEP to warn the user in realtime about the impending start and end of a reservation.

**Assigned to:** Richard

**Start:** 10/7/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.14 NRSE adds a new QoS reservation to the database

NRSE adds new SLA to the QoS_rsv table and sets a schedule to the Schedule table. Does not (yet) check whether there is sufficient bandwidth to meet the request.

**Assigned to:** Toshi

**Start:** 5/06/03

**Estimate:** 10 days

**Completed:** 10 days

**Notes:**

### 4.4.15 NRSE checks resource availability (Real-time, Single domain)

When NRSE receives a QoS reservation request, it checks the Schedule table in the database to decide whether the request can be accepted or not.

**Assigned to:** Toshi

**Start:** 20/6/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.16 NRSE checks resource availability (Non-real-time, Single domain)

If the request as it stands cannot be met, but the request is a non-realtime one, the NRSE searches for space for the request with a new flow specification which has different data rate to the original one between start time and end time. This new SLS is returned.

**Assigned to:** Toshi

**Start:** 28/6/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.17 NRSE checks resource availability (Real-time, Multi domain)

In addition to 4.4.15, a local NRSE asks a remote NRSE if the QoS request is acceptable or not.

**Assigned to:** Toshi

**Start:** 20/7/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:**

### 4.4.18 NRSE deletes a QoS reservation from the database

If the reservation has not been activated, NRSE does the following steps: Firstly, NRSE confirms that the request really exists in the QoS_rsv table. Secondly, NRSE deletes the reservation from QoS_rsv table and ScheduleI table.

If the reservation has already been activated, NRSE tells traffic control to cancel the current reservation.

**Assigned to:** Richard

**Start:** 16/5/03

**Estimate:** 5 days

**Completed:** 5 days

**Notes:** This story has not been fully implemented. Section 5.9.9 explains more details.

### 4.4.19 Database user management

Database class is able to perform tasks related to management of users.

- Add user to database (return error if username is not unique)
- Delete user from database
- Update user's PGP key

**Assigned to:** Toshi

**Start:** 1/7/03

**Estimate:** 10 days

**Completed:** 10 days

**Notes:**

# Chapter 5

# Implementation

The system was implemented as four packages.

**Utility** - library functions for use of NRSE and Client (not dependent on any other package).

**NRSE** - the NRSE server itself (depends on Utility).

**Client** - examples of client programs that control the NRSE (depends on Utility[1]).

**Test** - JUnit tests that exercise the functions of the other three packages.

The relationship between the packages is shown in Figure 5.1.

## 5.1 Protocol behaviour

Figure 5.2 shows how the system makes a multi-domain reservation and the protocol actions involved (with some simplification).

The NRSE class has already been launched, which started a BEEP server listening for requests with the NRSEProfile.

The user enters the parameters for the reservation he would like to book into his client program. (Or the user's application decides the parameters automatically and sends them to the client module.) The client creates an instance of the SLS class and initialises it with these parameters.

The SLS is then serialised into an XML document, which serves as reservation request. The format for this document, and the parameters required by the SLS, can be found in Appendix E on page 82.

[1] In practice there are some dependencies to NRSE code as well, but it will be trivial to eliminate these

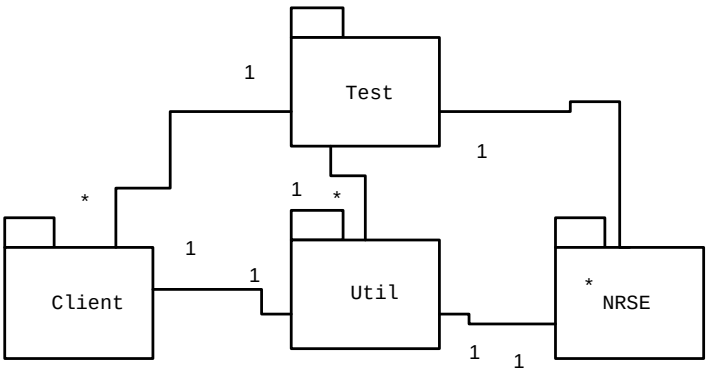


Figure 5.1: Relationship between packages

Using BeepClient, the XML is transmitted to the NRSE, and processed by NRSEProfile. First the SLS is de-serialised to form a copy of the SLS object at the NRSE. Then the credentials field of the SLS is used to authenticate the request by the database of users. If successful, the Database class checks to see whether there is sufficient bandwidth available for the reservation. If so, it formulates its own SLS request, and sends it to the remote NRSE, which is at the site specified as the destination in the SLS.

The remote NRSE performs the same checks as the local NRSE, except it authenticates the local NRSE, not the user. If everything is successful, then both NRSEs add the reservation to their databases. The local NRSE sends a BEEP reply back to the client to indicate success.

When it becomes time to actually activate the reservation, a notification message is sent, again using BEEP, to the BEEP server specified by the client.

The administrator must also run a separate process called SLSActivator. This is the process which actually configures the router to filter and mark traffic flows as requested. Usually this is run on the same machine as the NRSE, but it does have to be. It could be run on any machine that has access to the database. We provide the Router interface, and different low-level router implementations are possible. We have created TCRouter, which configures a Linux router using TC. Therefore SLSActivator must be run on the Linux router, while the NRSE may be on a different machine. It would be possible to create another class to control, for example, a Cisco IOS router.

In the following sections, we provide a summary description of the functionality of each package, and then each class. Readers who desire a more thorough understanding are advised to turn to the Javadoc generated docu-

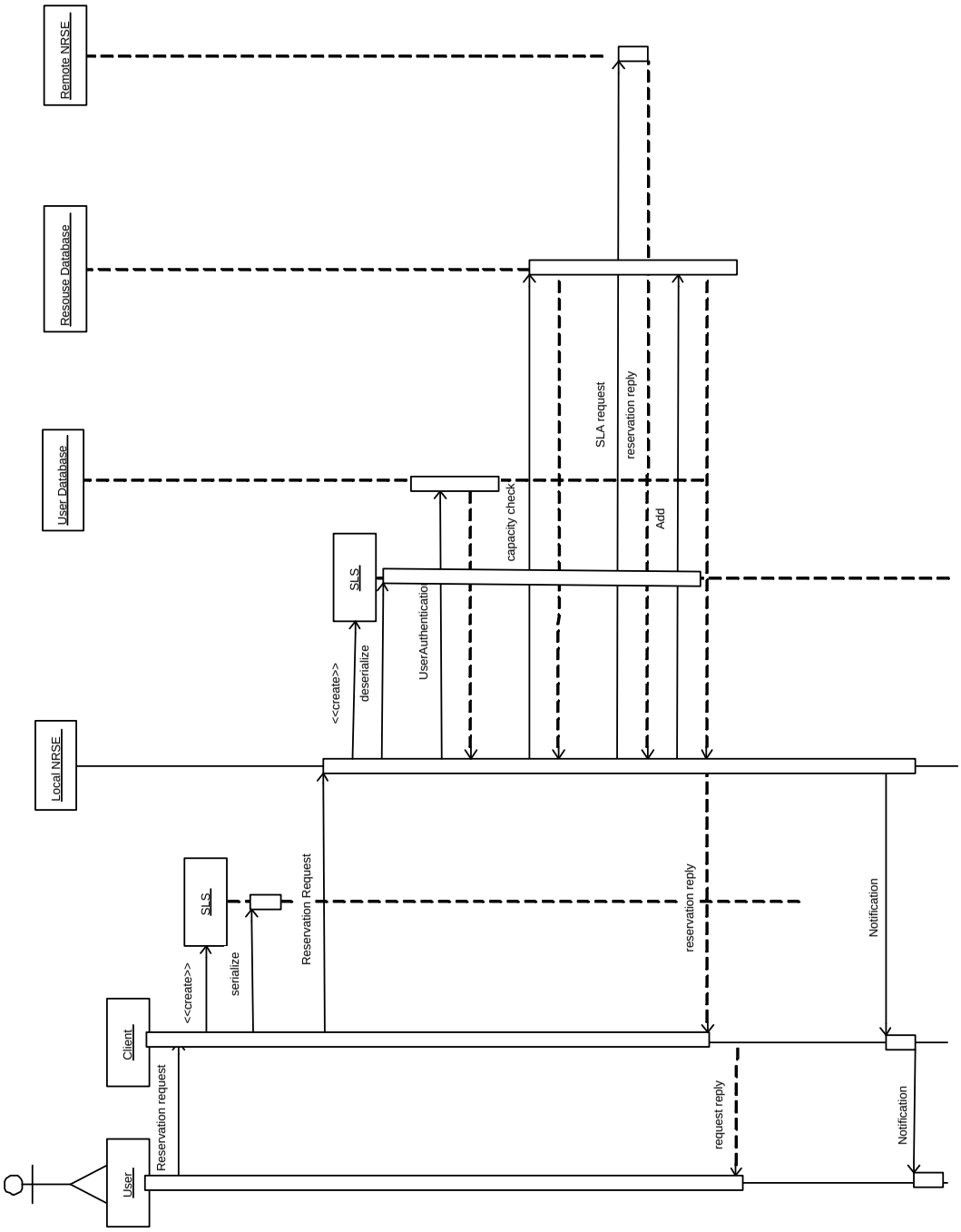


Figure 5.2: Sequence Diagram

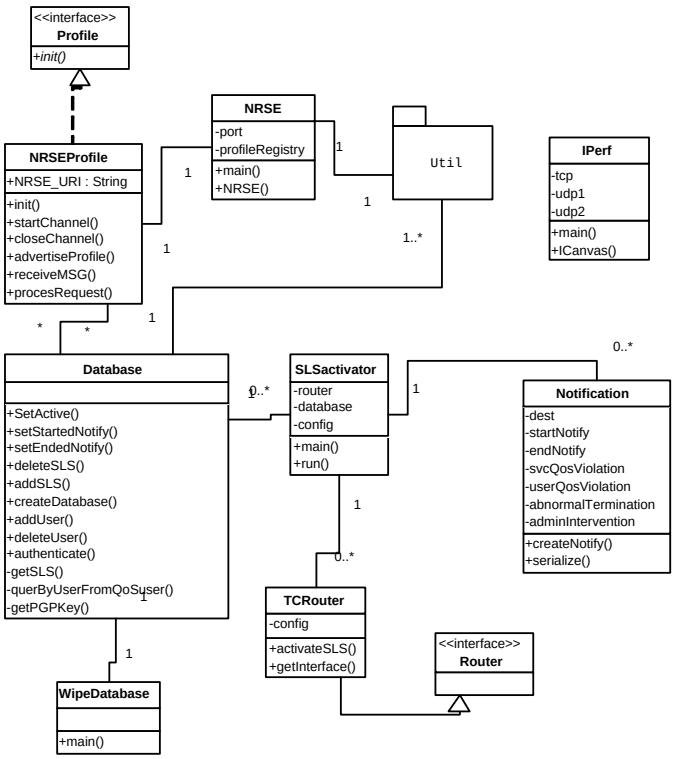


Figure 5.3: NRSE class diagram

mentation in the appendix, or even to consult the source-code itself[2]. (On the other hand, casual readers not interested in gory details may wish to skip to the next chapter.) The DATABASE section is of particular importance because this is where much of the reservation logic is implemented.

## 5.2 NRSE package

The NRSE package includes the NRSE server program and is shown in figure 5.3. NRSE class is the main class and it uses DATABASE and NRSEPROFILE classes. SLSACTIVATOR class runs in a separate thread and polls the database for reservations to activate at regular intervals. TCROUTER IMPLEMENTS the ROUTER interface and SLSACTIVATOR uses TCROUTER to activate an SLS. The Database class handles the all method to manipulate the PostgreSQL. IPERF is a stand alone class to monitor the traffic.

[2]available on CD-ROM or on the web

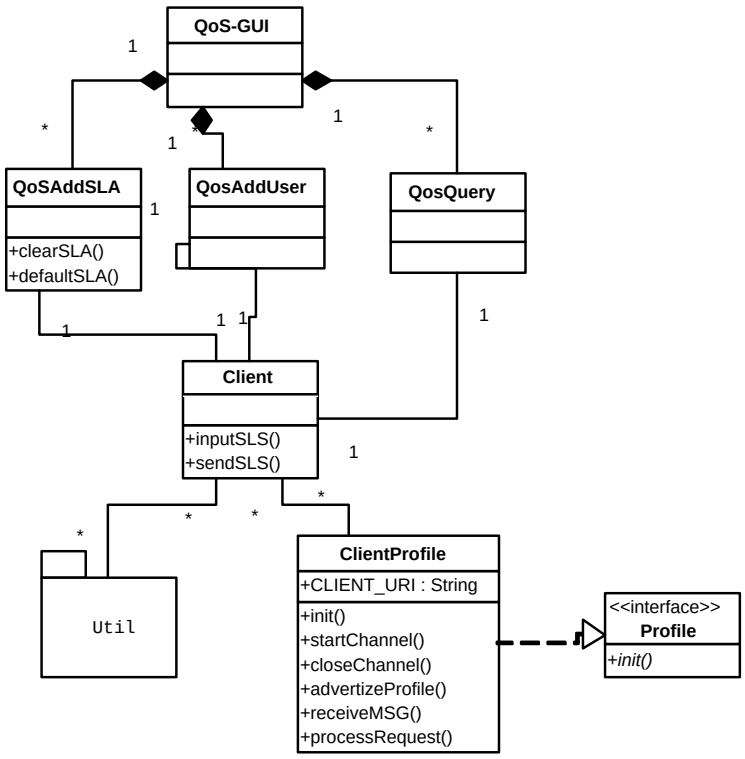


Figure 5.4: Client class diagram

## 5.3 Client package

The Client package (see figure5.4) includes the GUI and the client code. QoS class is the entry point of our client and it creates function classes such as QoSAddSLA, QoSAddUser and QoSQuery. These functions will compose XML and send the XML file by launching Client(which uses BeepClient). ClientProfile implements Profile using Beepcore.

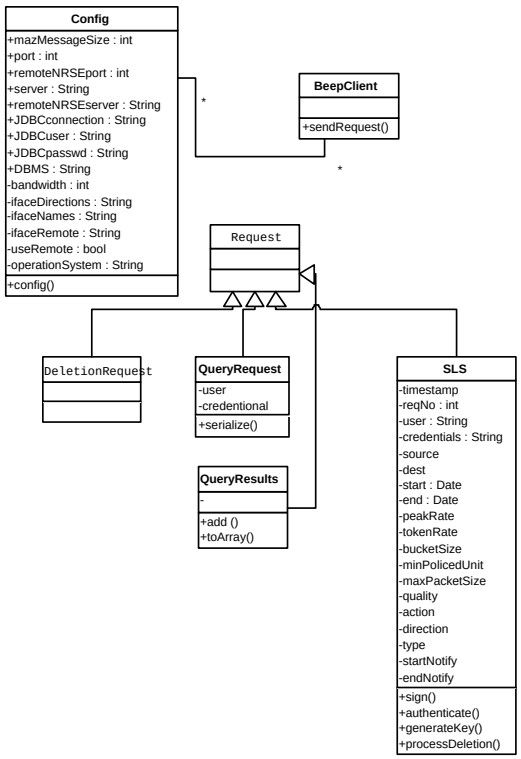


Figure 5.5: Utility class diagram

## 5.4 Utility package

This package includes the classes which are used by both client and server. (See figure5.5) SLS object, BeepClient, Query Deletion and Config are included.

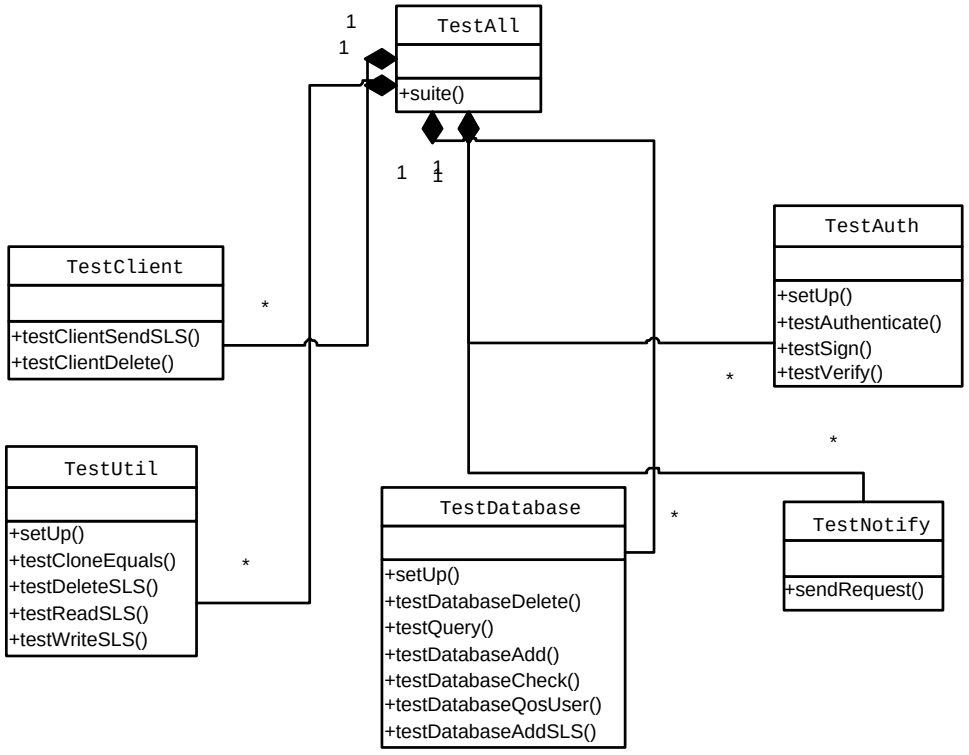


Figure 5.6: Test class diagram

## 5.5 Test package

This package include test cases (see figure 5.6). TestAll calls all the test classes. Test classes are used to test the functions and they are separated into client, utility, authentication, notify and database.

## 5.6 Client classes

### 5.6.1 QoS class

QoS class is the main program that executes the whole client program through a GUI interface. The QoS class utilises many programs from its own client package as well as other packages. QoS class layouts the fields of user name, password, and credentials/private key for user authentication. It also contains fields of server address and port so that the user can connect to the NRSE server.

QoS class enables the user to add a new SLA for a new QoS request, modify an existing SLA, delete an existing SLA, and add a new user. These functions will be discussed in details next. On start up, QoS class attempts to read the user's private key from the .qoskey file from the user's home directory classpath, which will then be displayed in the credentials field. On exit, the program attempts to store the user's private key in the same .qoskey file again.

#### Add SLA

Add SLA function simply calls the QosAddSLA class for creating a blank standard SLA request form.

#### Modify SLA

Modify SLA function is not enabled as it is not complete. However, it is foreseen that it will use many functions similar to that of the Delete SLA function such as QueryRequest, and QueryResults.

#### Delete SLA

Delete SLA function connects the client to the server and search for all the SLA that belong to the current user. It utilises QueryRequest and QueryResults classes in the NRSE package to do the search through the database. The results will then be displayed through the QoDeleteSLA class. If the user is the administrator with the username admin, all the SLA that exists in the database will be displayed.

#### Add User

Add User function simply calls the QosAddUser class for creating a blank standard Add User form. In addition, it contains methods to clear the SLA

form and to set a default SLA format on some fixed fields for facilitating tests easier without retyping the whole form.

### 5.6.2 QosAddSLA class

QosAddSLA class simply display the format of a SLA request form.

#### Send SLS

Send SLS function attempts to send the SLS from the client to the NRSE server to reserve the QoS desired. In addition, it formats all the fields into an appropriate SLS class object.

Even though PGP key generation and authentication are implemented in the program, there are no credential checks currently. The credentials field can contain 'cheat' for overcoming the credential check for now. It should not be left there as a loophole when the application is put to real use.

### 5.6.3 QosDeleteSLA class

QoSDeleteSLA class displays a SLA request one at a time, being navigating by the previous and next buttons. The fields are updates on navigation.

An attempt was made on creating a different format of displaying the QosSeleteSLA form. This approach places all the SLA requests together to be displayed all at once at a glance in a table. Navigation switches to the scrollbars. However, this attempt failed and we focused our attention to other parts of the project to keep up with the planned schedule.

#### Delete

Delete function attempts to delete the SLA request currently being displayed. Deletion involves communication to the NRSE server through BEEP connection and deletion of the request in the database. On successful deletion, the deleted SLA request will not be displayed. When all the SLA requests of the current user are deleted, all the fields will be blank and disabled.

### 5.6.4 QosAddUser class

QosAddUser class display the format of a Add User form.

#### Generate Key

This function generates a set of PGP private and public keys for the new user to be added.

#### Add

This function adds the new user to the database, together with its private and public keys generated. Non-standardise or altered private and public keys will not be accepted. Existing users will also not be allowed to add the same username to the database.

### 5.6.5 Client class

Client class us used to start a client program. It takes the server address, server port, and default configuration from the nrse.properties data file and starts a connection to the NRSE server using BEEP. If no server information is available, the default server settings is used.

Without using the GUI function, this client program can be executed for a simple console QoS application. It accepts user choice of adding, deleting or querying the QoS requests. It also reads from user input for the information to be entered. The default information will be taken in for the SLS request if nothing is entered on the console for simple and efficient testing. It also contains methods to store these information, serialising it for transmission to the NRSE server through BEEP.

### 5.6.6 ClientProfile class

ClientProfile class contains the BEEP profile implementation for NRSE, using the default configuration settings form the Config class.

It contains functions to start and end a BEEP channel. It is also able to listen to a channel and then reply with a response to confirm whether the SLS request, sent through BEEP, has been received successfully by the NRSE server or not.

### 5.6.7 Abandoned classes

There are some abandoned classes like QosMenu, QosLayout, QosQuery classes. Initially, the design of the QoS application was made such that there is a menubar on top of the application which was later removed. QoSQuery class was programmed to handle a general query frame which was redesigned such that both delete and modify SLA functions will perform queries using

programs written in the NRSE package. QosMenu and QosLayout classes was programmed to design the modify and delete frames to be similar with similar methods and layout. Thus, QosModifySLA and QosDeleteSLA class can then inherit from QoSLayout class. This was later removed since the delete SLA screen and modify SLA screen was foreseen to be quite different. QosModifySLA class is not updated and currently the function of modifying an SLA is not completed yet.

## 5.7 Utility classes

### 5.7.1 BeepClient class

BeepClient class does all the actual BEEP connections, connecting and transmitting messages between the client and NRSE server. The BeepClient firstly needs to set up the configuration using the server address and port, config file properties, and the NRSE URI. Otherwise, the default settings of these configurations will be used.

BeepClient deals with the connection of the client to the NRSE server using the configuration provided. After establishing a session with the NRSE, a channel is started to send the request over to the NRSE. A reply is then expected to be provided by the NRSE and is then checked to see that the reply is the same as the request sent. After that, the channel is closed followed by the closing of the session.

After establishing the session, the TLS was initially started for privacy protection. This was then commented out since we do not find it necessary. Furthermore, it had conflicts with the JUnit testing codes we wrote for it. Currently, we return error codes even for local errors which should be done by throwing exceptions instead. Error codes include errors like the reply message is not correct. This part still remains unfixed. The reply message received seems to have an extra byte attached to it which is ignored for now.

### 5.7.2 Config class

Config class generally consists of the configurable options used to operate the BeepClient class. Configuration properties include local and optionally remote NRSE server connection details, certain message properties, database properties, and interface details.

## 5.8 NRSE classes

### 5.8.1 SLS class

SLS class takes in SLS field inputs and creates a SLS object, in a number of different forms because there can be many different formats needed by different parts of the application. This basically means a few constructors are created. Firstly, the SLS information are read up from a XML document and parsed element by element to be stored as an SLS object. There are also functions to serialise the SLS object into an XML document for transmission through BEEP. There are specific methods to create a delete SLA request and modify SLA request in the form of XML document. The modify method is not completed yet.

In addition, SLS class provides authentication functions for generating a set of PGP private and public keys for a single user. It also authenticates a user against his provided private key. Other authentication functions include making a signature to the message to be sent with the related PGP keys, and verifying whether the PGP generated keys are up to standard or not.

### 5.8.2 SLSactivator class

SLSactivator class periodically checks from the database to see if there are any reservation to activate. TCRouter will be called if the results is positive. Notifications checking are also done periodically. Start Qos notification and end Qos notification messages will be sent to the client from the NRSE.

### 5.8.3 NRSE class

An NRSE object represents an actual NRSE server. Running this class creates an instance of NRSE, which then launches a BEEP server using NRSEProfile. All of the functionality of the NRSE is in NRSEProfile. By default, configuration options are loaded from a file named 'nrse.properties' in the classpath, but a different configuration file may be specified on the command line.

This class was originally based on the code for the BEEP Daemon launcher, but we found that to be an order of magnitude more complicated than we needed, because it it supported launching multiple BEEP servers with multiple profiles. We only need one server with one profile.

### 5.8.4 NRSEProfile class

NRSEProfile class is the BEEP profile implementation for NRSE server. It handles the establishment and termination of BEEP connections depending on the configuration from the Config file. After it receives a message from the client, it attempts to send back to the client a reply of the message received. Upon receiving a SLA request, it will coordinate with the database and process the request accordingly. Currently, it processes SLA addition requests, SLA deletion requests, and query requests. On adding a new SLA request, the authentication code checking is skipped by using cheat in it. This is because authentication is implemented but not checked in the application.

### 5.8.5 TCRouter class

TCRouter class sends TC commands to provide for Qos request. Deactivating Qos request method may be useful in certain circumstances but it is not implemented yet. There are methods to find the appropriate network interface used for route to IP, getting the interface name, doing IP lookups for the appropriate interface, and getting the masked IP address depending on the length of its prefix.

### 5.8.6 MyCanvas class and ReservationQuota class

MyCanvas class is useful for a quick glance on all the reservations made and bandwidth used up as well as available bandwidth. To do this, queries are made to the database on all the reservations made and plotted on the screen, updated on every few seconds.

MyCanvas class attempts to read all the Qos reservations from the database and plot the reservations in terms of bandwidth, with the bandwidth plotted vertically in the y-axis and time plotted horizontally in the x-axis. The bandwidth is scaled, such that there is always only 10,000Kbps available at all time, in relatively 10 units.

However MyCanvas class crashes when the start time of a reservation is too early or the end-time of a reservation is too late. ReservationQuota class attempts to overcome this effect by scaling the time such that the earliest start time and the latest end time of all the reservations are taken into account and scaled accordingly. However, scaling is done in terms of seconds and if one reservation lasts for a minute, another reservation lasts for a day, and yet another reservation lasts for many days. This would make the short-lasting reservation that lasts for a minute looked very insignificant and probably not

visible using ReservationQuota class display. This is yet another problem to be solved.

### 5.8.7 IPerf class

IPerf class monitors TCP and UDP packet flows. Currently, there is a TCP flow that attempts to simulate the transmission of the Qos requested data. The other 2 UDP flows attempts to take up the rest of the available bandwidth for the same NRSE. The result of these transmissions are shown on the screen, plotting the three flows with the bandwidth as the y-axis vertically against time along the x-axis horizontally.

### 5.8.8 Notification

Notification class handles the notifications to be sent from the NRSE to the client when commands are executed. The start and end times of the notifications are stored. There are triggers for service violations, user Qos violations, abnormal termination, and administrator interventions. If these triggers are on, notifications will be sent to the client whenever there are any of the triggered events as mentioned. Additionally, there are methods to handle the creation and parsing of the XML documents to form a notification object, and serialising the notification object to form an XML document for transmission through BEEP from the client to the NRSE.

### 5.8.9 CreateTable

CreateTable class simply creates two tables, namely the QoS_rsv and QoS_user, to be utilised by the database class later on.

### 5.8.10 QueryRequest

QueryRequest class reads from an XML document for a query, stores the details into a query object, and then serialises it to an XML document for transmission through BEEP from the client to the NRSE.

### 5.8.11 QueryResults

After processing a database query, the query results will be stored using this QueryResults class. It is able to take an XML document that contains the results of a query , which holds more than one SLA request. These SLA requests are then read and stored as a QueryResults object as a whole.

## 5.9 Database class

The NRSE needs to manage information about SLSs, user and network resources. The DATABASE class is charge of this management. It stores lists of SLSs, user information and network resource reservation schedules, and also offers methods to manage the information.

A relational database management system (RDBMS) is used in order to manage this information. Using the RDBMS, our system can keep the crucial information which affects the system's reliability and return to its normal operating state even if the NRSE is accidentally terminated or suffers hardware failure. We adopted PostgreSQL as a RDBMS for the reasons described later.

The database consists of three tables: QoS_rsv, QoS_user, and ScheduleI. Each table is independent of one another. Only an NRSE administrator can access these tables manually.

In this section, we describe the details of these three tables with an Entity-Relationship diagram for the whole database. Next, we explain Non-Realtime reservation and database transaction. Then, our selection of the RDBMS is justified. Finally, we will briefly explain some issues for the future.

### 5.9.1 QoS_rsv table

The QoS_rsv table is used to store all information on accepted SLAs. It stores received SLA on an "as is" basis. The only modification is that a unique ID value is added to each SLA.

This table has thirty fields shown in table 5.9.1.

The SLSACTIVATOR class and the NOTIFICATION class referenced this table.

The ID fields is the primary key.

#### Check Constraints

SQL allows us to define constraints on columns. A check constraint allows us to specify that the value in a certain column must satisfy an arbitrary expression. Table5.9.1 shows a list of check constraints which are used in the QoS_rsv table.

### 5.9.2 QoS_user table

The QoS_user table is used to authenticate users. The table holds information of user name and user's public PGP key, therefore it has two columns: NAME

and PGPKEY, and NAME is a primary key. Both NAME and PGPKEY are TEXT data type[Pos03]. The TEXT data type is given a detailed explanation in 5.9.7.

### 5.9.3 ScheduleI table

The ScheduleI table is used to check whether a requested QoS is bookable or not. The table stores the amount of remaining bandwidth as it changes over time for each network interface; therefore it has three columns: TIME, BANDWIDTH, and IFACE which means 'network interface'.

The table is initialised with values of the maximum free bandwidth for each network interface, as configured by the administrator. TIME is specified "2003-01-01 00:00:00" as the epoch time. Table5.9.3 shows an initialised ScheduleI table. It shows that the NRSE server administers two network interfaces and the maximum bandwidths are 10000 Kbps for both interfaces.

The following paragraphs explain how the table is updated.

When it is confirmed that enough bandwidth is available to allocate for a requested QoS SLA, the NRSE first adds (or updates) two rows into the ScheduleI table: one has the TIME value equal to start_time in the SLS and the other has the TIME value equal to end_time in the SLS. BANDWIDTH value of the row which has the TIME value equal to start time is calculated by subtracting peak_rate from BANDWIDTH value of the previous row in time order. The same procedure is done for the other row.

Second, the NRSE updates BANDWIDTH values in all rows which have TIME values being between the start_time and the end_time of the requested SLA. New BANDWIDTH values are calculated by subtracting peak_rate from the original BANDWIDTH values.

Table 5.9.3 shows a state of the ScheduleI after one reservation (start_time: 25-08-2003 10:00:00, end_time: 25-08-2003 14:00:00, peak_rate: 2000) was made. If we focus attention on the interface eth1, we see that free bandwidth is 10000 Kbps at the start time, i.e. 2003-01-01 00:00:00. It is reduced by 8000 Kbps at 2003-08-25 10:00:00 and this state lasts until 2003-08-25 14:00:00. Then free bandwidth returns to 10000 Kbps at 2003-08-25 14:00:00.

If another reservation which starts at 2003-08-25 12:00:00 and ends at 2003-08-26 12:00:00 with 3000 Kbps peak_rate on the interface eth1 is made, then the ScheduleI table will become a table shown in 5.9.3.

### 5.9.4 ER model (Entity-Relationship model)

When we designed the database, Entity-Relationship (ER) modelling technique was used to organise the information about NRSE system. Entity-

Relationship diagrams can be used to represent a model, which is then realised as database tables. The E-R diagram is shown in figure 5.9.4. Entities ('user', 'SLS', 'PGP key', etc.) are used to define database structures and attributes. Relationships ('owns') are defined between entities.

We designed a detailed database structure based on the ER diagram. In our actual database, some attributes are expanded and divided into many smaller attributes.

### 5.9.5 Non-Real-time reservation

There are two reservation types in the NRSE: realtime and non-realtime. Some network applications such as video conferencing require a fixed data rate and an appointed start time and end time. On the other hand, some other network applications such as file transfer are tolerant of data rate changes within the appointed start time and end time. This means that realtime transmission is not always required. We may find space for such non-realtime transmissions by changing start time, end time, and transmission rate within the limits of the application's requirement. ([BSCC03])

The DATABASE class first tries to make fixed rate reservation for non-realtime transmission just as in the case of realtime transmissions. If the reservation can be made, then it is done and finished.

If there is not enough network capacity available for fixed rate transmission, then the DATABASE object searches for free space in the ScheduleI table which is sufficient to contain the requested SLS's total amount of transmitting data. If enough space can be found, the database object makes a new reservation with new start time, end time and transmission rate, and then returns the new SLS to the client. If the client cannot accept the new SLS, it must cancel the SLS.

Alternatively, if space cannot be found, the database object returns an error message to the client.

Our strategy for finding space is simple and load balancing. Generally, there are many possible spaces that would be acceptable. However, we do not search all possible spaces. One reason is that even if we list all spaces it is hard to decide which space is best to use. Another reason is that the amount of calculation required is projected to increase rapidly as the number of reservations increases, so our system would not scale well.

In the actual implementation, when a free rectangular space that can contain the total amount of transmitting data is found in the ScheduleI table, we stop searching for other free spaces and set new flow parameters which fit within the gap. If the rectangle found is bigger than the request requires, we set the data rate parameter as small as possible to flatten the data rates.

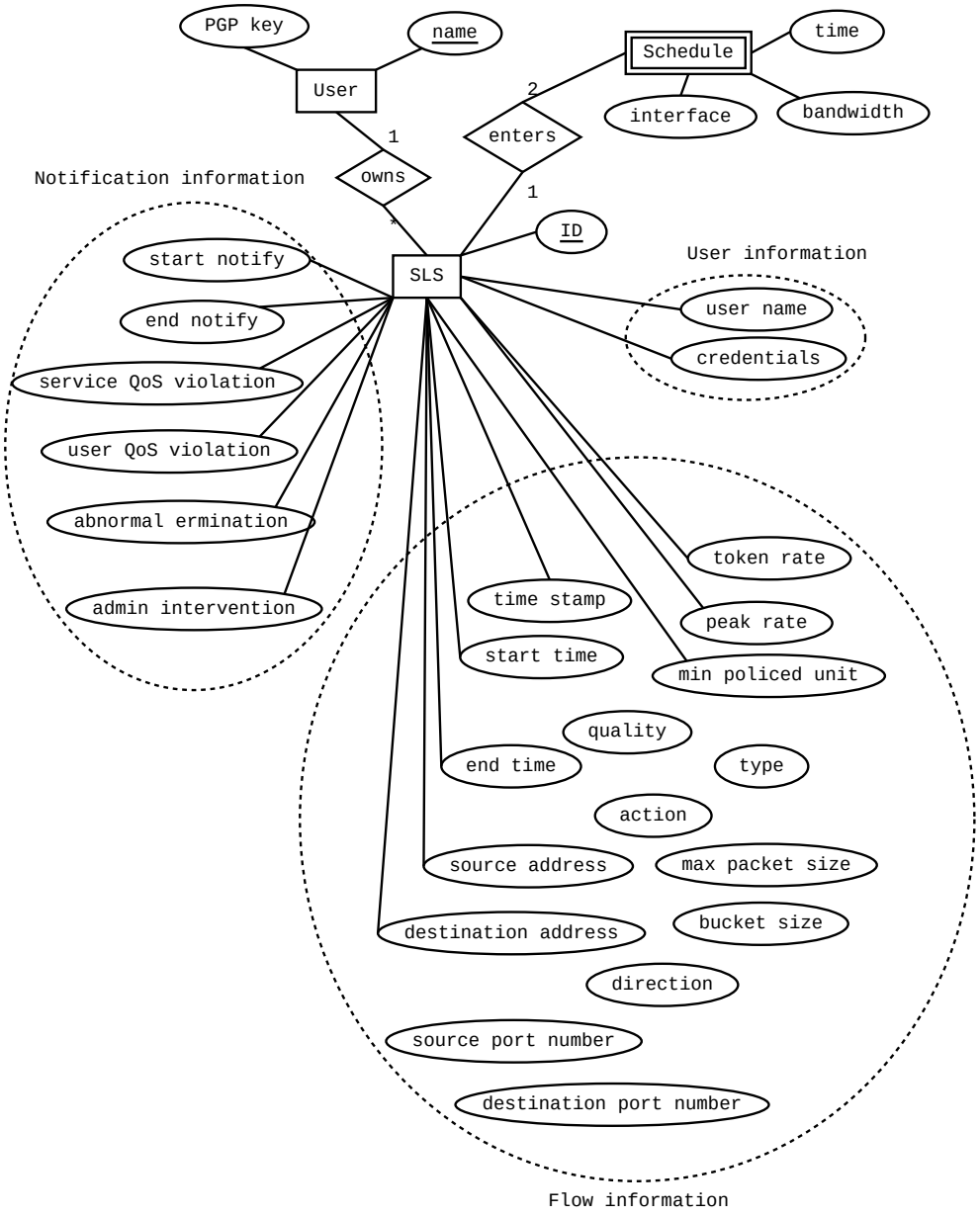


Figure 5.7: Entity-Relationship Diagram for NRSE database

We planed to provide an interface for changing the non-realtime reservation mechanism in order to allow administrators select the best algorithm for their local site policies. However, we regret that we could not complete this because of lack of development time.

### 5.9.6 Selection of Database Management System

There are many options for us when we use a relational database management system (DBMS). For instance, PostgreSQL, MySQL, Oracle, Firebird, mSQL, etc. Great attention should be given to interchangeability, scalability and availability of our system because they enhance the value in the real world. Viewing the database management system from this perspective, widespread availability and compliance with SQL99 standard become the criteria for evaluation. Taking these points into account, we chose PostgreSQL as the DBMS for our project. We discuss this decision in a little more detail in 7.1.

### 5.9.7 PostgreSQL

We tried with care not to use PostgreSQL specific commands. However, some advanced features in PostgreSQL are used for creating a higher quality system. They are basically transposable to other SQL expressions which are compliant with SQL99 standard at the expense of a decline in the quality. These features are enumerated below:

#### Character Types

Although SQL defines two primary character types: **character(n)** and **character varying(n)**, where n is a positive integer, both of these types have limitation in length. Some DBMSs truncate strings that are too long without so much as raising an error. PostgreSQL supports the **text** type which stores strings of any length. It does not require an explicit declared upper limit on the size of the string with no performance penalty ([Pos03]). The type **text** deviates from the SQL99 standard, however, many other DBMS packages with the exception of MySQL have it as well.

#### Network Address Data Types

We use network address data type **inet** for an IPv4 network address. The reason why we adopted the type **inet** rather than fixed length string is that the type **inet** offers input error checking and several specialised operators and

functions ([Pos03]). When we port the DATABASE class to other DBMSs, the type **inet** should be replaced by type **character varying(n)** and some error checking functions should be added.

### 5.9.8 Transaction

Transaction is an essential function especially for a multi-domain QoS reservation to maintain data integrity. Transaction is a unit of interaction with a DBMS and it must be treated in a coherent and reliable way. A local NRSE does not commit a transaction until a remote NRSE returns successful reply. If a remote NRSE does not accept a SLS request, the local NRSE aborts the transaction and reverts data in a database to an earlier state.

Although PostgreSQL provides "Multiversion Concurrency Control" function which maintains data consistency for concurrent data access[HCF97], we used only the least complex functions, that is, lock and rollback, because we placed great value on conformity with the standard.

### 5.9.9 Future work

**Multi-domain issues**

**Non-realtime reservation.** Currently, Multi-domain non-realtime reservations are left unsupported, though single-domain non-realtime reservations are supported. The realisation of the non-real-time reservation is technically possible and is not too difficult. This will be accomplished by changing a return value of the addSLS method, which checks network resource availability and adds a new SLA into the database, so that it returns the modified SLA. The new SLA will be returned to the remote NRSE in a result document. Further negotiation may be necessary if the remote NRSE also needs to modify the SLS.

**Authentication** among NRSEs is one of the essential functions for secure, credible and scalable multi-domain system. It is not a good idea that NRSE authenticates a remote user, otherwise each NRSE will need to deal with a large amount of user information and this will weaken the scalability of our system. Therefore a remote NRSE has an entry in the user table and is authenticated similarly to a local user. This must be configured by the administrator at present, but we would like to add some automation to the process.

**Common issues**

**Deleting and modifying reservations.** Because of the limitation of the Linux Traffic Control (TC), this Database class cannot delete a selected SLA which has already been activated. Once all filters have to be deleted by TC utility and all filters need to be added again except for the filter which should be deleted in order to realise deletion function.

Modifying is not much different than deleting. Only difference is adding the modified new SLA at the last step.

**Accounting.** Information which will be required for accounting such as reservation rate by hour or by bandwidth has not been built in to the database yet.

One idea is that adding a new column into the QoS_user table. The new column holds a sum of charges per user and values of the column are calculated by using a newly created table of charges and QoS_rsv table.

Inter-domain tariff structure will be determined by contracts between domains.

| Column name | data type | description |
|---|---|---|
| ID | INTEGER | unique ID for a SLA |
| TSTAMP | TIMESTAMP | time stamp when the SLA is made |
| START_TIME | TIMESTAMP | transmission start time |
| END_TIME | TIMESTAMP | transmission end time |
| SRC_IPV4 | INET | source IPv4 address |
| DST_IPV4 | INET | destination IPv4 address |
| SRC_PORT | INTEGER | source port number |
| DST_PORT | INTEGER | destination port number |
| PEAK_RATE | INTEGER | packets of a flow which exceeds this limit will be dropped |
| | | |
| BUCKET_SIZE | INTEGER | bucket size of the token bucket |
| MIN_POLICED_UNIT | INTEGER | minimum packet size which is policed |
| MAX_PKT_SIZE | INTEGER | maximum packet size which can be sent |
| | | |
| TOKEN_RATE | INTEGER | token rate of the token bucket |
| QUALITY CHAR | CHECK | EF or best effort |
| ACTION | CHAR | queue management |
| DIRECTION | CHAR | uni- or bi-direction |
| TYPE | CHAR | real-time or non-real-time |
| USER_NAME | TEXT | user name |
| CREDENTIALS | TEXT | user's credentials |
| IFACE_IN | INTEGER | inbound network interface |
| IFACE_OUT | INTEGER | outbound network interface |
| START_NOTIFY | INT | number of seconds before reservation start |
| END_NOTIFY | INT | number of seconds before reservation end |
| SVC_QOS_VIOLATION | BOOL | service QoS violation |
| USER_QOS_VIOLATION | BOOL | user QoS violation |
| ABNORMAL_TERMINATION | BOOL | abnormal termination |
| ADMIN_INTERVENTION | BOOL | administor intervention |
| ACTIVE | BOOLEAN | SLS is activated or not |

Table 5.1: Columns in a QoS_rsv table

| Check Constraints |
|---|
| START_TIME > EPOCH_TIME |
| START_TIME < END_TIME |
| SRC_PORT > 0 AND SRC_PORT < 65535 |
| DST_PORT > 0 AND DST_PORT < 65535 |
| MIN_POLICED_UNIT > 0 |
| MIN_POLICED_UNIT > 0 |
| TOKEN_RATE > 0 |
| DIRECTION IN ('u','b','m') |
| TYPE IN ('r','n') |

Table 5.2: Check Constraints of a QoS_rsv table

| NAME | PGPKEY |
|---|---|
| Toshi Aiyoshi | temporary key |
| Andy Liow | temporary key |

Table 5.3: A sample of a QoS_user table

| TIME | BANDWIDTH | IFACE |
|---|---|---|
| 2003-01-01 00:00:00 | 10000 | eth0 |
| 2003-01-01 00:00:00 | 10000 | eth1 |

Table 5.4: Initial state of a sample ScheduleI table

| TIME | BANDWIDTH | IFACE |
|---|---|---|
| 2003-01-01 00:00:00 | 10000 | eth0 |
| 2003-01-01 00:00:00 | 10000 | eth1 |
| 2003-08-25 10:00:00 | 8000 | eth1 |
| 2003-08-25 14:00:00 | 10000 | eth1 |

Table 5.5: ScheduleI table after making one reservation

| TIME | BANDWIDTH | IFACE |
|---|---|---|
| 2003-01-01 00:00:00 | 10000 | eth0 |
| 2003-01-01 00:00:00 | 10000 | eth1 |
| 2003-08-25 10:00:00 | 8000 | eth1 |
| 2003-08-25 12:00:00 | 5000 | eth1 |
| 2003-08-25 14:00:00 | 7000 | eth1 |
| 2003-08-26 12:00:00 | 10000 | eth1 |

Table 5.6: ScheduleI table after making one reservation

## 5.10 Blocks Extensible Exchange Protocol Core

### 5.10.1 Rationale

BEEP(Blocks Extensible Exchange Protocol Core) is used to exchange messages between NRSE server and the client, as well as between NRSEs. There are two reasons why we choose BEEP. One is that BEEP allows developers to focus on the important aspects of their applications rather than wasting time with the detail of establishing communication channels. The other is that BEEP is now standardised and supports multiple platforms. There are also open source implementations available for many different languages including like Java, C and C++. We used the Beepcore library (version 0.9.07) for Java, available from [Sou02].

### 5.10.2 Use of Beepcore

Beepcore is used to send both client requests (e.g. add SLS, delete SLS) and notifications.

The NRSE server launches a BEEP server which listens for connections from the client. To send a service level specification to the server, a client application will launch the BeepClient class. For the notification, the NRSE client launches the BEEP server and waits for the notification. The NRSE server then sends a notification when the reservation is ready to be activated or the reservation has finished completely.

BEEP provides a framework for the message exchange allowing the user to specify a 'profile' which defines the message syntax and semantics. BEEP has features like transport security (message encryption) and user authentication ([Die99], [Ros02]). The messages can be plain text, and structured with XML.

Three styles of exchange are allowed:

- MSG/RPY for one to one exchange
- MSG/ERR for error report
- MSG/ANS for one to many exchange

A somewhat out-of-date but still informative tutorial that gives an overview of how these exchanges work is [Dum01].

## 5.11 Logging

In the NRSE, we used Jakarta Log4j (see [jak03]) to allow logging. The logfiles will be useful in a production system, e.g. for auditing procedures, as well as diagnosing problems. For us, the logs were an invaluable debugging aid.

- Log4j can be configured at runtime without modifying the application binary and is controlled by the runtime command options or a configuration file. We used the convention of loading the log4j configuration from a file called *log4j.properties*. This file can be anywhere in the classpath. By using the LOGGER.SETLEVEL() method, one can change the level of logging. We can classify the logging information to five different levels of severity:

  1. debug
  2. info
  3. warn
  4. error
  5. fatal

  Debugging information is only of interest to developers, while fatal errors should concern all users.

- Log4j provides an 'appender' object which can print to multiple destinations. For example, the console appender enables us to display the debug messages on the console while file appender enables us to store the messages in a file.

- Log4j also has the PATTERNLAYOUT class. This class enables us to set a prefix message in front of the log message. For example, if you create PATTERNLAYOUT with the options `("%c [%t] %L %x : %m")`, you can store the information category of the logging event, name of the thread, output line number, fixed debug message, debug message correspondingly. (The message which will be shown by %x option can be used in a very similar way to a message in a stack.)

# Chapter 6

# Testing

Testing is an essential part of the XP methodology. We used two complimentary testing strategies: automated unit testing, and manual functional testing.

## 6.1 Unit Tests

Unit tests are automated tests. Whenever a programmer begins to program a new feature, he should first write some code to test whether the new feature works. To aid in this process, we used the JUnit package. This made creating tests very simple.

We created test classes in a separate package. These test classes are subclasses of JUnit classes, and we call JUnit methods to assert that the results of each are as expected. JUnit includes several programs to run the tests. Figure 6.2 shows the graphical tester. Here, the red bar clearly shows one of the tests has failed, and information is provided to enable the programmer to see exactly what went wrong. This failure must be corrected before the code is checked in. Figure 6.1 shows a successful test run.

As figure 6.3 shows, the programmer may choose which test suites to run. After the programmer has succeeded in testing his newly added feature, he should re-run all the test suites to ensure that he has not inadvertently broken some other part of the system. This enables the programmer to confidently refactor code in other parts of the system without worrying that he is introducing additional bugs.

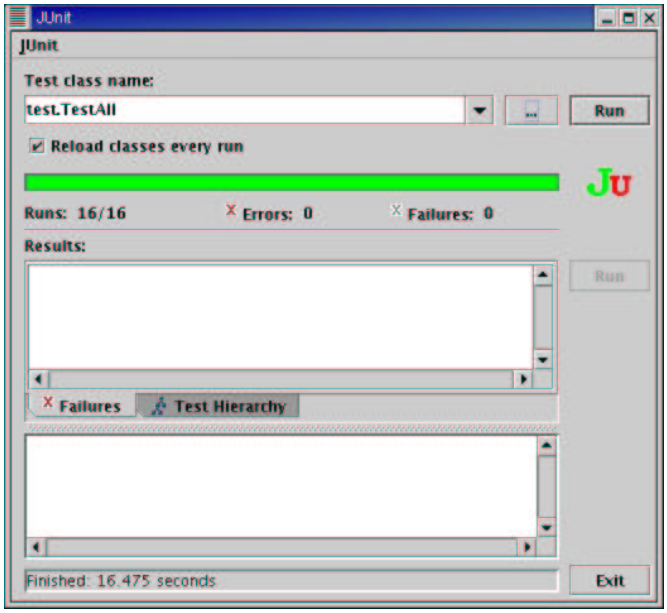


Figure 6.1: Successful JUnit test

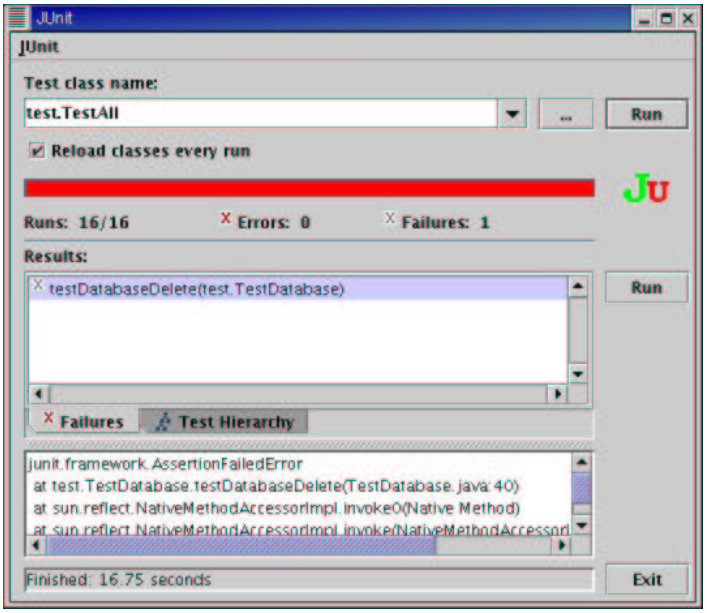


Figure 6.2: Unsuccessful JUnit test

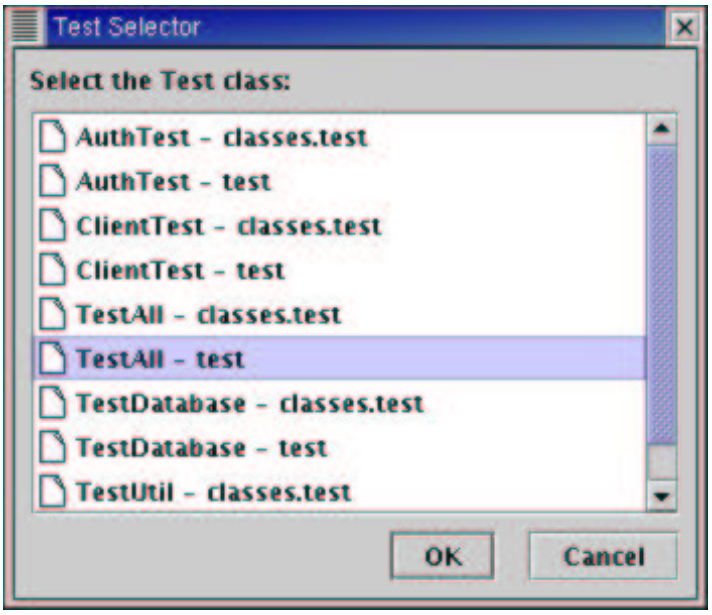


Figure 6.3: JUnit choice of tests

### 6.1.1 Test details

| Suite | Test | Description |
|---|---|---|
| Auth | Sign | Sign SLS and verify signature |
| | Authenticate | Sign SLS and authenticate user from database |
| | Verify | Create and verify good and bad signatures - bad signature must fail |
| Client | SendSLS | Reads SLS parameters (from file) to client which should then create and send SLS to NRSE |
| | Delete | Creates request for deletion and uses client to it send to NRSE |
| Database | Delete | Adds SLS to database, deletes it, checks it has gone |
| | Query | Adds two SLSs to database, then queries for them. |
| | QueryXML | Processes XML query and executes it |
| | Add | Adds some SLSs into database, pulls them out, checks they are the same as before they went in. |
| | Check | Adds lots SLSs into database, some of which exceed bandwidth limits. Check method should notice this. |
| | User | Checks behaviour correct for adding, deleting and querying users. |
| | Non-realtime | Ensures database makes correct changes to SLS to meet a non-realtime request. |
| Notify | Notify | Launches a client and sends it a notification. |
| Util | Clone | Tests that SLS.clone() works |
| | DeleteSLS | Processes deletion request XML and checks against known correct values |
| | ReadSLS | Reads XML format SLS from file and compare to known-correct values. |
| | WriteSLS | Write SLS to XML format file and read it back again. |

### 6.1.2 Test results

All tests passed.

## 6.2 Functional test scenarios

It is very difficult to test the system in the large with unit tests. While it would be possible to write automated tests that make use of the GUI and run on multiple machines at the same time, it is not really worth the time to do so. Therefore we use a human being to test that we have met our major milestones. In a commercial project these tests would be provided by the customer, to ensure the system meets his needs. Similarly, we created these tests in conjunction with our supervisor.

The most important of these milestones are single-domain operation and multi-domain operation. Complete instructions for the human performing these tests can be found in Appendix C.

### 6.2.1 Testbed

A testbed was provided for us, consisting of three routers and three clients, configured as shown in figure 6.4. All machines were running Redhat Linux 9, but the hardware and configuration varied slightly from machine to machine. We did not have physical access to the testbed - all access was via the network, through a secure connection to the gateway machine.

### 6.2.2 Single-domain test

We configured a NRSE to run on Router3, and a client to run on Client3. We used the IPerf tool to monitor network performance. We ran IPerf clients on Client3, which sent packets to IPerf servers running on Router3. We used one TCP flow, one flow of large (1470 byte) UDP packets and one flow of small (40 byte) UDP packets. The UDP flows were designed to completely fill the pipe's 10 Mbps capacity, leaving very little for the TCP, which would never get out of slow start mode.

We used our client software to request a reservation of 5 Mbps. We set the source address to Client3 and the destination address to Router3. We specified the reservation should be activated immediately.

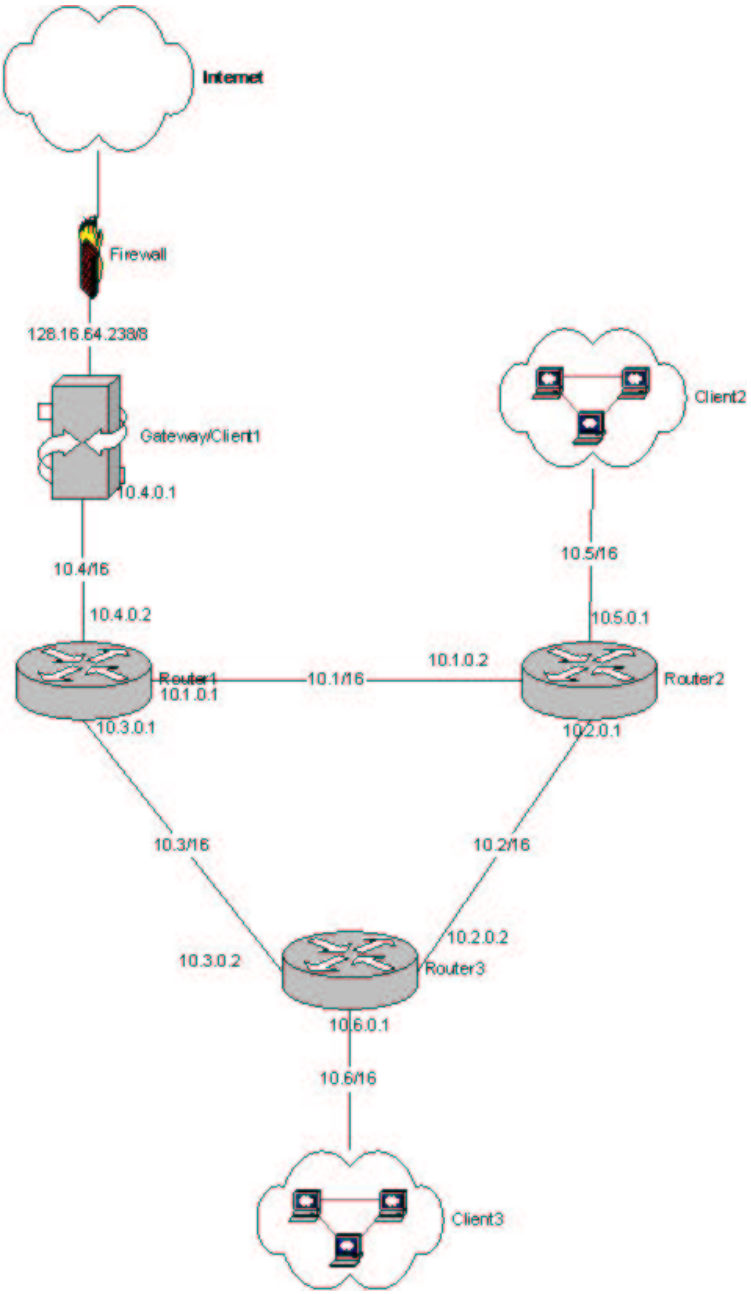


Figure 6.4: Testbed (diagram courtesy of Yangcheng Huang)

**Results**

We wrote a program to plot the bandwidth used by IPerf in realtime. The output can be seen in figure 6.5.

The blue plots represent the bandwidth used by the TCP flow. The other colours are UDP flows. The graph is scaled to 10 Mbps on the Y-axis. Each plot represents one second of time on the X-axis. As can be seen, initially the UDP flows completely swamped the TCP flow. When it made the reservation request, it took the NRSE a few seconds to process it. Then the reservation was activated and the router immediately began dropping UDP packets to give priority to the TCP flow. The test was a success.

In fact slightly more bandwidth was allocated to the TCP flow than we requested, but this is a flaw in Linux's traffic shaping implementation and not in the NRSE.

### 6.2.3 Multi-domain test

The nature of the test was the same as for the single-domain, but this time we used had *two* domains, each containing a router with an NRSE (on the router machine) and a client node.

Client3 wished to make a reservation to guarantee 5 Mbps of bandwidth for a TCP flow to Client2. We ran the same IPerf client and server processes as before, but this time running from Client3 to Client2. All the packets were forwarded through machines in this order:

1. Client3
2. Router3
3. Router1
4. Router2
5. Client2

Thus our TCP flow was actually traversing three different networks.

**Results**

The results, shown in figure 6.6, are very similar to those for the single domain test. After the reservation is made, the TCP flow gets about 5.7 Mbps of bandwidth.

During some of our earlier tests, the plots for the multi-domain test appeared more scattered than they did for the single domain test. This was not

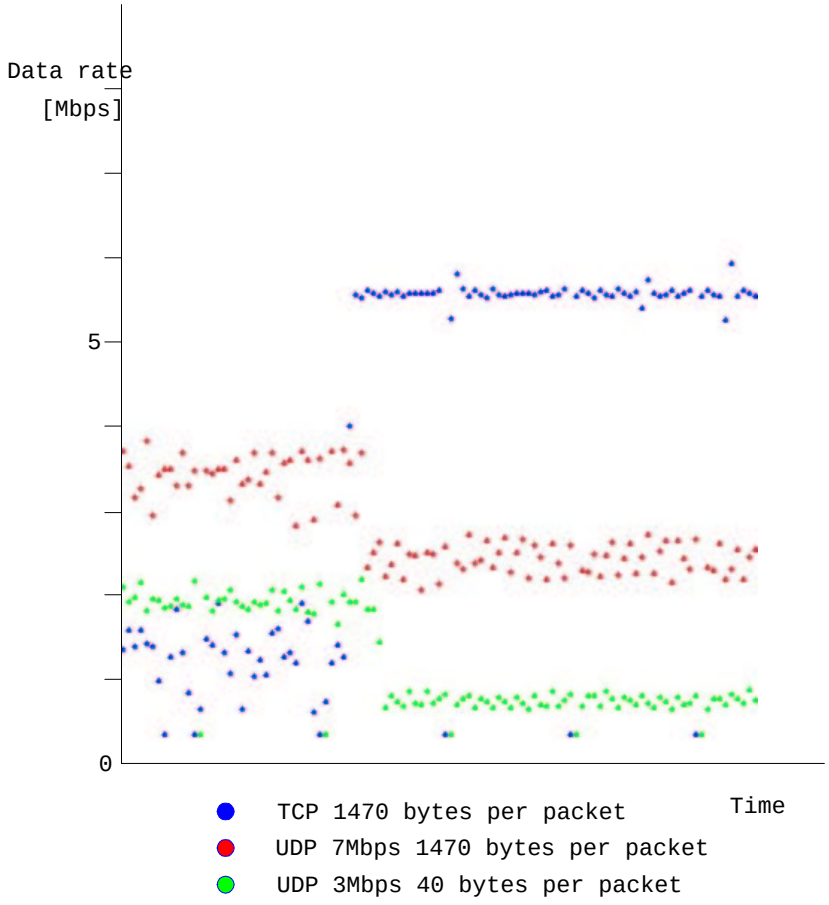


Figure 6.5: Single-domain network performance

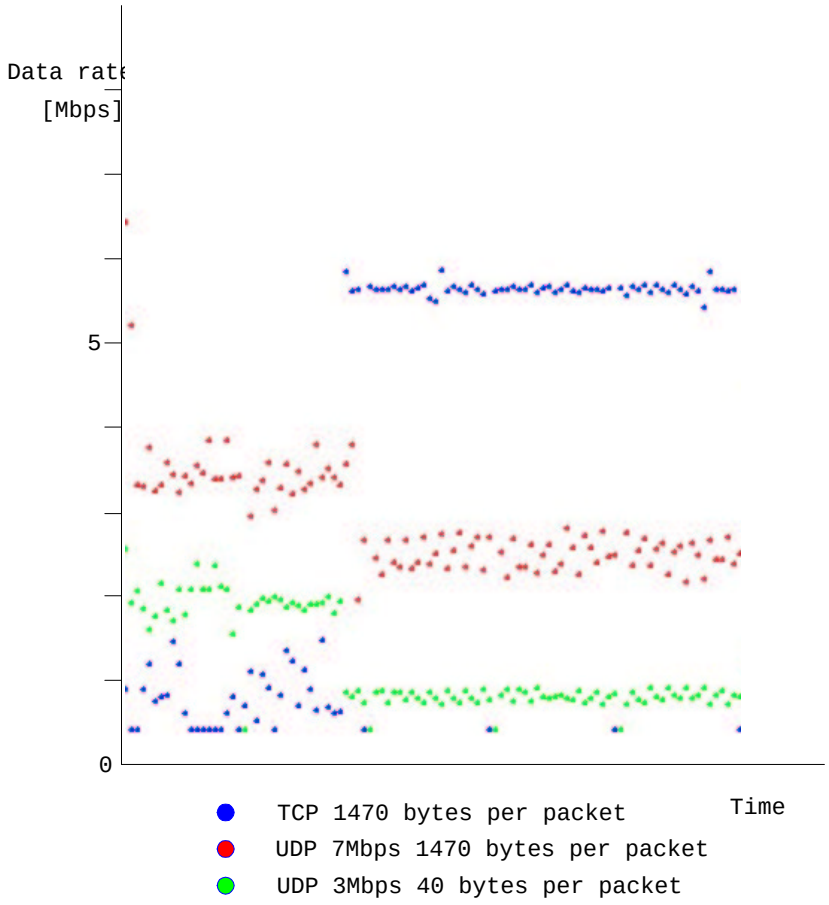


Figure 6.6: Multi-domain network performance

evident towards the end, so we suspect it may have been caused by a misconfiguration in the testbed that was later corrected. (The testbed configuration was not under our control and changed constantly.)

# Chapter 7

# Evaluation

Here we evaluate the design of the system as well as the wisdom of some of our design choices. We also suggest improvements for future development. Further evaluation can be found in our personal reports, and an evaluation of the overall success of the project in in Chapter 8 on page 68.

## 7.1 Database

We wanted our system to be usable by as many people as possible, therefore we could not make it dependent on a proprietary database such as the industry standard Oracle. The most popular Free database is MySQL, and this is what our supervisor suggested we use. However, we had some debates about this because while MySQL has a reputation for being fast, it is also known to lack many of the advanced features of Oracle. We had some prior experience using another Free database, PostgreSQL, which is very full-featured. Therefore we decided to develop using PostgreSQL, but port to MySQL later in the project. Eventually, we would also like to port to Oracle, and Firebird (another Free database which has a small user-base but is fast gaining an impressive reputation and may soon challenge PostgreSQL.)

We took care to avoid using any of PostgreSQL's advanced features, and to keep to the SQL99 standard as far as possible, in order to make it as easy as possible to port to other databases. However, when we did finally attempt to port to MySQL, we found many problems. MySQL is not an ACID compliant database, and is missing many important features such as transactions, views and subselects, all of which are used by our system. (After some work we discovered it is possible to use an alternative MySQL database engine that does support some form of transactions, but the syntax is not standard.) We also had to do extensive work to convert some of our datatypes to the more

limited set available in MySQL.

Therefore we abandoned the port to MySQL. Porting to Oracle, Firebird and mSQL remains a possibility for future work.

## 7.2 Networking

The choice here was between Java's implementation of TCP sockets, or a higher level third party library. BEEP was suggested by our supervisor as a cross platform, peer-to-peer, XML-based framework for authentication communications.

However, in practice we were not greatly impressed by BEEP. The promise of authentication proved, for us, to be an empty one. (See section 7.3.)

The fact that BEEP uses XML internally is irrelevant to the application developer. The messages sent by BEEP are simple plain text. We used that text to contain XML, but BEEP provided no more features to support the sending of XML than is provided by simply TCP sockets.

BEEP appears to have brought as very few advantages over TCP sockets, and cost us several weeks of development time to overcome the lack of documentation.

## 7.3 Authentication

We intended to use BEEP's TLS security feature. This ought to have provided us the verification based on X.509 certificates as well as encryption (if we wanted privacy). Unfortunately, we could find no documentation on how to use this feature. BEEP's entire documentation consists of some very basic Javadoc.[1] There are no tutorials, and no working example to prove that the touted TLS feature is actually implemented.

We considered using X.509 certificates without BEEP. There is some support for them in the Java library. We decided against this, because it would have been like re-inventing the wheel. None of the team has experience of X.509 and we feared it would be necessary to implement an entire security infrastructure on top of X.509 akin to TLS. We planned to investigate this further if there was time later.

For the moment, we decided to implement authentication using PGP message signing. We considered using the Free GnuPG implementation, but we

[1] We are referring here to the Java *implementation* of BEEP, Beepcore. The BEEP protocol itself is well documented, but creating our own BEEP implementation using the specification would have been well beyond the scope of this project.

thought it's easier to use the OpenPGP Java library implementation provided by Cryptix. Some of the team already had experience with this product, and we were confident that we could quickly create a working implementation.[2]

Cryptix has improved a lot since when we used it in the past, and generally worked well. Unfortunately there was a strange incompatibility with Sun's JCE (Java Cryptography Extension) that prevented us from repackaging the Cryptix libraries into a single archive on certain platforms. This problem was worked around by using the original Cryptix packages and by removing Sun's JCE, but it was a bit of an inconvenience. We are not sure whether this was caused by a bug in the JCE, or in Cryptix, or, more likely, whether we were incorrectly signing the archives. Future work could be done to determine this.

Cryptix, like all public key systems, suffers from excessive CPU requirements. A hybrid system would be more practical for a large-scale NRSE. If the requirement to use BEEP is dropped, then we would suggest investigating using a TLS implementation such as OpenSLS.

## 7.4 Testbed

The testbed is shown in figure 6.4 on page 60.

Not having direct control over the testbed caused problems and unnecessary delays. The testbed was setup and maintained by other students who had far less experience with Linux networking than we did. We eventually obtained root access, and from then on we found it quicker to do the sysadmin work ourselves. It was still very inconvenient sharing the testbed with other experiments, however. The design of the testbed was quite different from the design of a real internet. The routing was deliberately convoluted to introduce extra hops, thereby simulating a larger internet than we actually had. This caused unexpected problems - we could not rely on the standard output of the 'route' command to tell us where a packet would actually be routed.

The hardware configuration and software installed varied slightly from machine to machine. This meant it wasn't possible to compile a program on one machine and then simply transfer it to the others - everything had to be re-compiled on each machine. There was so no sharing of files, user logins, or time synchronisation across machines. (Indeed we wasted many

[2]Time management was critical throughout the project, so we preferred to get a working proof-of-concept implementation rather than waste weeks evaluating unfamiliar technologies.

hours looking for a bug that proved to be caused by one of the machine's having a clock that was 7 months slow.)

The greatest problem with the testbed was that it used a private subnet that was not part of our LAN. The only access was via a multi-homed gateway machine. This was because of security concerns, and made testing very cumbersome and difficult.

We recommend that, if possible, programmers are given responsibility for creating a testbed that meets their own needs. On the final day of writing this report, when we came to add in the functional test results, we found the testbed had been reconfigured so that our tests would no longer run. If we had control of the testbed, we not make any changes to a working configuration at such a crucial time. However, In practice, we realise that resource constraints and bureaucracy will rarely allow this.

## 7.5 Performance

Java allowed relatively rapid development, and the large standard library was a great benefit. The performance of Java was a little disappointing though. Setting up a BEEP connection took several seconds, even using the loopback network interface. Executing the cryptographic functions also introduced some delay. Ultimately, however, this is not a huge problem. Users will usually be booking reservations for several hours or days in advance, and so a delay of a few seconds during the booking will not be a problem for them.

Immediately we found Sun's compiler implementation was too slow to use effectively on our old workstations. Therefore we used IBM's open source Jikes compiler instead, which was more than fast enough. We stuck with Sun's JRE, however, because it was already installed on our machines and it was guaranteed to be compatible with all the third party libraries we chose to use. It might be worth considering a switch to use IBM's JRE to improve run-time performance of the NRSE.

The main reason for choosing Java as our platform (other than our existing familiarity with the language) was to enable the NRSE to be run on as many different systems as possible. With hindsight, it may have been possible to achieve similar portability by using C++ with cross-platform libraries such as wxWindows.

# Chapter 8

# Conclusion

We successfully implemented a working prototype of the NRSE. There are still a number of rough edges that should be polished off before the code is 'production ready'. For instance, Java's DOM implementation is rather messy; a higher level, cleaner solution to processing XML would be preferable. Many parts of the system use numerical error codes (modelled after those in HTTP), but we intend to replace these with XML error documents. Indeed, this will be a requirement for negotiating multi-domain non-realtime reservation.

We solved the problem of resource reservation for a single domain for realtime and non-realtime reservations. We also implemented multi-domain reservations for two domains for realtime requests.

Multi-domain non-realtime requests are a simple evolution and would not be difficult to implement given more time.

Most requirements were met. However, we did not implement a keep-alive mechanism, nor notification for policy violations[1]. We also did not have time to do much investigation of how the local NRSE can discover a remote NRSE. At present, remote NRSEs are entered by the administrator. We believe a simple entry in the DNS records should be used to automate the process. (Alternatively, there could be an LDAP database system holding details of NRSEs.) We also recommend that the administration and query features be expanded on.

A more serious problem is that of three or more domains. We ensure there is sufficient bandwidth available for a reservation at the local gateway, and also at the gateway to the destination network. We also mark the packets as EF, to give them priority throughout their journey. However, our packets may be forwarded through several other networks as they traverse the core

[1]We need to investigate whether this is even possible using a Linux router

of Internet. We can not guarantee enough bandwidth is available in all these networks. At present, core networks are usually over provisioned, so this is not a problem. Nevertheless, we suggest this aspect deserves further study.

# Appendix A

# User Manual

This chapter presents the manuals for users.

## A.1 Main Screen

Firstly, execute the NRSE server with the following command.

java nrse.NRSE

Then, executes the client host with the following command.

java client.Qos

NRSE is the main program that runs the NRSE server and Qos is the main program that runs the main program of the client.

The very first client screen that shows is as below.

Enter the user name and password as required, as well as the credentials/ private key of the same user. The server address and port are set to default at the moment, but will be required if they changes.

Then the following options follows, each of them links to a new window with the exceptional Modify SLA option. modify SLA option is not implemented yet and thus is left disabled at the moment.

- Add SLA
- Modify SLA
- Query/Delete
- Add User

## A.2 Add SLA

This screen is illustrated as below.

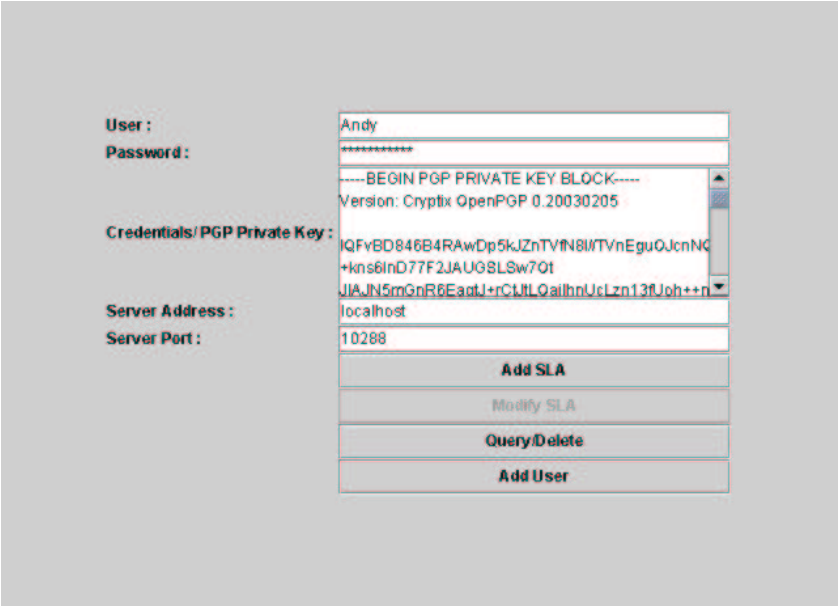


Figure A.1: Client Main Screen

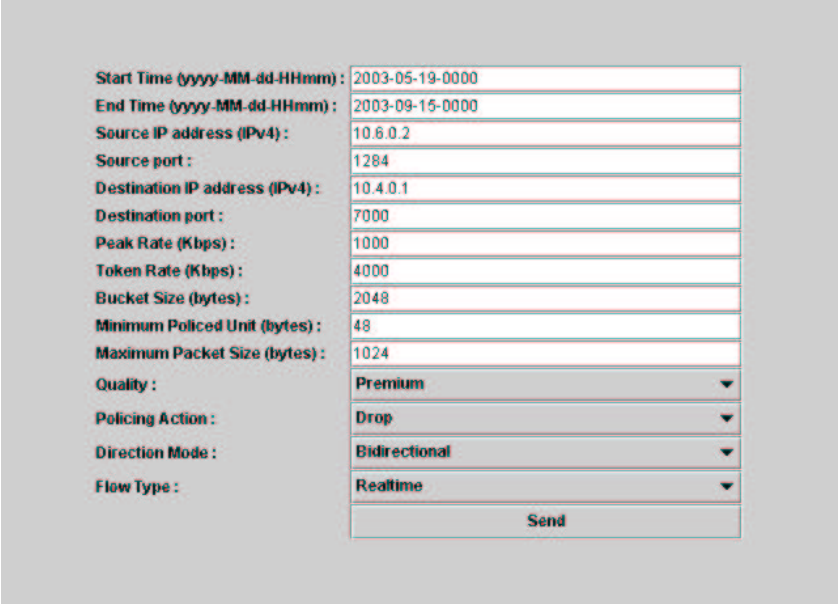


Figure A.2: Add SLA

It simply shows all the fields that are necessary to set up a Qos request. The default values are used currently. Simply enter all the provided fields and click send to send the Qos request. The request will then be sent to the NRSE server.

## A.3 Query/Delete

This screen queries and displays all the Qos requests made by the user referred to on the main screen. These Qos requests can then be modified or deleted. Currently, only the delete function works and the modify option is disabled. Just navigate using the previous and next buttons to the desired Qos request and click delete to delete it.

## A.4 Add User

This option is only applicable for the administrator even though it is not the restriction is not implemented yet. For administrators, please refer to implementation - how to run section on more details. This screen creates a new user account. Enter the new user name and password. Then, click Generate to generate a new set of PGP private and public key. After generating them, click the Add to add the new user with the password and the generated keys.


Figure A.3: Add User

# Appendix B

# Administrator manual

## B.1 Troubleshooting

Cryptix library sometimes does not work if you have Sun's jce.jar in your java libs directory. If you get SecurityExceptions, remove jce.jar. (This *seems* to be because we have not signed our library jar correctly. So you could also fix it by using the original signed jar files from cryptix.org rather than our library bundle.)

## B.2 Installation

The easiest way to install is to copy or download the jar archive, nrse.jar, onto any machine that you wish to act as a router, an NRSE, or a client. Extract the file nrse.properties from the jar file - you will need to edit this file if you do not want the default configuration. See B.5 for details.

### B.2.1 Requirements

- A router running Linux 2.4.
- Beepcore *http://sourceforge.net/projects/beepcore-java*
- JUnit (only for running tests) *http://www.junit.org*
- Xerces *http://xml.apache.org/xerces2-j*
- log4j *http://jakarta.apache.org/log4j*
- Jikes (only to compile) *http://oss.software.ibm.com/developerworks/opensource/jikes/*

- Cryptix *http://www.cryptix.org*
- PostgreSQL database and JDBC driver *http://www.postgresql.org*

For your convenience we have provided a jar archive containing all the necessary libraries. If you do not use this, it will be necessary to install them all separately and add them all to your classpath.

## B.3 Running the NRSE

First, it is necessary to initialise the PostgreSQL database.

```
java -cp .:nrse.jar:libs1.jar nrse.WipeDatabase
```

Then, you may wish to run the test suite:

```
java -cp .:nrse.jar:libs1.jar -Dlog4j.ignoreTCL=true \
junit.swingui.TestRunner test.TestAll
```

To launch the NRSE itself:

```
java -cp .:nrse.jar:libs1.jar nrse.NRSE
```

You may run the NRSE on the same machine that is doing the routing. You *must* run the activator program on the router:

```
java -cp .:nrse.jar:libs1.jar nrse.SLSactivator
```

We also have a couple of very basic, very alpha viewers that are able to display the reservations in the database graphically. These probably need a lot more work before they are useful.

```
java -cp .:nrse.jar:libs1.jar nrse.MyCanvas
java -cp .:nrse.jar:libs1.jar nrse.ReservationQuota
```

## B.4 Running the client

To launch the example graphical client:

```
java -cp .:nrse.jar:libs1.jar  client.Qos
```

For instructions on how to use the client, refer to the manual on page 70.

There is also an older console based client:

```
java -cp .:qos.jar:libs1.jar client.Client
```

## B.5 Configuration

Edit *nrse.properties* and put it somewhere in your classpath. If you wish to use a file with a different name, you can specify that on the launch command-line. To configure logging options, edit the *log4j.properties* file.

| Option | Description | Examples value(s) |
|---|---|---|
| interval | How long interval is needed to search and activate the available reservation | 1000 |
| maxMessageSize | The MAX length of BEEP Message which the client can receive | 10000 |
| port | The port number used by BEEP connection | 10999 |
| server | Host name which is used by BEEP connection | localhost |
| JDBCconnection | Database Server Location and the database name | jdbc:postgresql://URL/DatabaseName [1] |
| JDBCuser | The UserName which is registered to use the database | rsmith |
| JDBCpassword | Password to use the database | none |
| DBMS | Database Software Name | PostgreSQL [2] |
| noOfIface | Number of the Network Interface | 6 |
| bandwidthX | available bandwidth for interface number X | 10000 |
| ifaceNameX | Corresponding network interface name which should be given by the result of ip command. | ethX |
| ifaceDirectionX | The direction of the network packet flow | in/out |
| ifaceRemoteX | Used to check whether the SLS is from remote site or local site. | true/false |
| useRemote | true for single domain, false for multi domain | true/false |
| remoteNRSEserverX | Address of the BEEP Server | 127.0.0.1 |
| remoteNRSEportX | Port Number for the BEEP connection | 10289 |
| operatingSystem | Which test environment you will use. [3] | test/Linux |

[1] In CS Computer, jdbc:postgresql://kennedy.cs.ucl.ac.uk/XXX is used. Kennedy is for the CS department computer and it starts postgresql every morning. In the test bed, jdbc:postgresql://localhost/XXX is used.

[2] MySQL option is no longer used. Reason is in the evaluation section.

[3] SLS activation is not possible on the Solaris machines used for development and testing

# Appendix C

# Demonstration

These instructions can be used by a human tester to replicate our demonstration on our testbed.

## C.1 Before running the program

- copy libs1.jar and release.jar to each computer. libs1.jar contains all jar libraries which are used in our program. release.jar contains all NRSE program which we made.
- run PostgreSQL on Router3 if PostgreSQL isn't yet running. (command : `service postgresql restart`)
- verb|tc-off|.

## C.2 Single-Domain

1. change the nrse.properties useRemote value to false.
2. At router3, wipe out all the data in the database.(command : java -cp libs1.jar:release.jar nrse.WipeDatabase)
3. Run the NRSE server on Router3. (command : java -cp libs1.jar:release.jar nrse.NRSE)
4. Run IPerf Server as a Graphic Mode on router1. (command : java -cp libs1.jar:release.jar nrse.IPerf)
5. Run the iperf-client on Client3. (command : ./iperf-client(run the batch file))

6. Run the SLS Activator on Router3. (command : java -cp libs1.jar:release.jar nrse.SLSactivator)
7. Run the NRSE Client on Client3. (command : java -cp libs1.jar:release.jar client.Qos nrse.properties)
8. Send SLS using GUI. src is 10.3.0.2(Router3), dst is 10.3.0.1(Router1). Reservation time must be the future time.
9. Wait until reservation time.

## C.3 Multi-Domain

According to the user story scenario, data is to be sent from client3 to client2. Client3 send the data to Router3, Router3 will send the data to Router1, and Router1 will forward the data to Router2. Then Client2 will receive the data.

1. change the nrse.properties useRemote value to true.
2. At Router2, wipe out all the data in the database.
3. At Router3, wipe out all the data in the database.
4. Run the NRSE server on Router2.
5. Run the NRSE server on Router3.
6. Run IPerf Server as a Graphic Mode on Client2.
7. Run the SLS Activator on Router2.
8. Run the SLS Activator on Router3.
9. Run the NRSE Client on Client3.
10. Send SLS using GUI from Client3 to Client2.
11. Wait until reservation time.

## C.4 After running the program

- verb|tc-off| to manually remove all TC filters.
- kill iperf process (command : `killall iperf`)

# Appendix D

# Glossary

**AF** - DIFFSERV Assured Forwarding service

**BEEP** - Blocks Extensible Exchange Protocol

**CMM** - Capability Maturity Model

**CVS** - Concurrent Versions System

**DIFFSERV** - Differentiated Services

**EF** - DIFFSERV Expedited Forwarding service

**GRS** - Grid Resource Scheduling

**GARA** - General-purpose Architecture for Reservation and Allocation

**GUI** - Graphical User Interface

**IETF** - Internet Engineering Task Force

**INTSERV** - Integrated Services

**IP** - Internet Protocol

**NRSE** - Network Resource Scheduling Entity

**OO** - Object-Oriented

**PGP** - Pretty Good Privacy

**QoS** - Quality of Service

**RSVP** - ReSerVation Protocol

**RUP** - Rational Unified Process

**SLA** - Service Level Agreement

**SLS** - Service Level Specification

**SQL** - Structured Query Language

**TC** - Traffic Control

**TCP** - Transport Control Protocol

**UDP** - User Datagram Protocol

**UML** - Unified Modelling language

**XML** - eXtensible Markup Language

**XP** - eXtreme Programming

# Appendix E

# XML documents

Here are examples of the XML documents processed by the NRSE (based on those in [BSCC03]. We have not included PGP signatures here.

## E.1 SLS request

```
<?xml version = "1.0"?>

<!-- sla_user_nrse.xml                    -->
<!-- Course: MSc DCNDS 2002/3             -->
<!-- Authors: Richard, Andy, Keiko, Toshi -->
<!-- GRS (GRID Resource Sharing)          -->
<!-- Service Level Agreement              -->
<!-- User - NRSE                          -->

<sla_user_nrse xmlns = "x-schema:sla_user_nrse-schema.xml">

  <!-- Request identification -->
  <id>
    <timestamp>2003-05-19-22080000</timestamp>
    <req_no>1</req_no> <!-- e.g. 32 bit random number -->
  </id>

  <!-- Administrative information -->
  <user_info>
    <user_name>Andy Liow</user_name>
    <user_credentials></user_credentials>
  </user_info>
```

```
<!-- Optional. If this is not present,
     SLA should remain in place until explicitly removed. -->
<time_span>
  <start_time>2003-05-19-0000</start_time>    <end_time>2003-05-20-0000</end_time
</time_span>

<!-- Could also have level 3 or 4 header fields other than these -->
<filter>
  <src_ipv4>128.16.10.1</src_ipv4>
  <src_port>1284</src_port>
  <dst_ipv4>128.16.10.11</dst_ipv4>
  <dst_port>8080</dst_port>
</filter>

<!-- Traffic specifications -->
<tspec>
  <!-- All rates in Kbps -->
  <peak_rate>1000</peak_rate>
  <token_rate>800</token_rate>

  <!-- All rates in bytes -->
  <bucket_size>2048</bucket_size>
  <min_policed_unit>48</min_policed_unit>
  <max_pkt_size>1024</max_pkt_size>
</tspec>

<!-- Service specifications -->
<qos>
  <quality>premium</quality> <!-- 'premium' = EF, 'low' = LBE -->
  <policing>
    <action>drop</action> <!-- For future. "delay" or "remark" possible -->
  </policing>
  <direction_mode>bidirectional</direction_mode> <!-- {uni|bi}directional -->
                                                    <!-- multicast in future -->
  <flow_type>real_time</flow_type> <!-- {real|non_real}time -->
</qos>

<notifications>

  <!-- Multiple instances of this are possible -->
```

```
    <notification_sink>
      <dst_ipv4>127.0.0.1</dst_ipv4>
      <dst_port>4000</dst_port>
    </notification_sink>

    <!-- Optional. Number of seconds before reservation start -->
    <start_notification>1</start_notification>

    <!-- Optional. Number of seconds before reservation end -->
    <end_notification>1</end_notification>

    <notification_flags service_qos_violation = "on"
                        user_qos_violation = "on"
                        abnormal_termination = "on"
                        administrator_intervention = "on"/>

  </notifications>



</sla_user_nrse>
```

## E.2 Deletion request

```
<?xml version = "1.0"?>
<delete>
  <id>
    <req_no>1</req_no>
  </id>
  <user_info>
    <user_name>Andy Liow</user_name>
    <user_credentials></user_credentials>
  </user_info>
</delete>
```

## E.3 Query request

```
<?xml version = "1.0"?>
```

```
<simple_query>
  <user_info>
    <user_name>Andy Liow</user_name>
    <user_credentials></user_credentials>
  </user_info>
</simple_query>
```

# Appendix F

# Miscellany

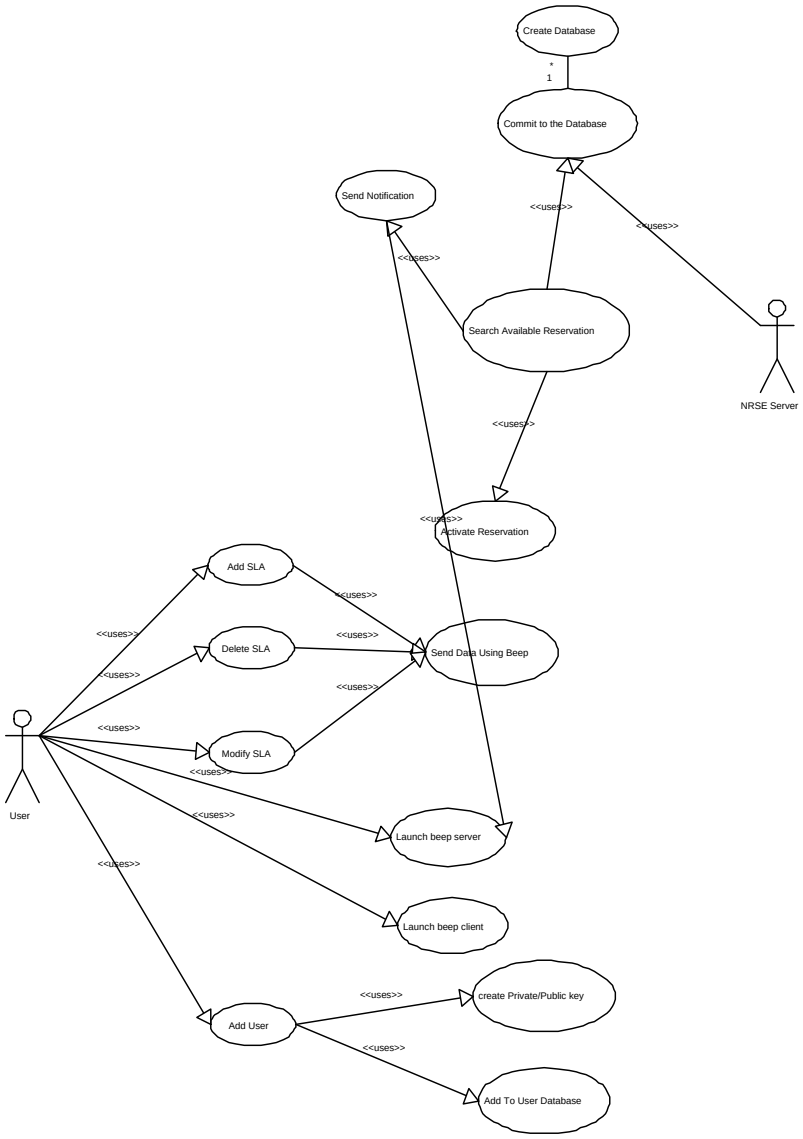


Figure F.1: Early draft of usecase diagram

# Bibliography


[Bec00] Kent Beck. *eXtreme Programming eXplained. Embrace Change.* Addison-Wesley, 2000.

[BSCC03] Saleem Bhatti, Søren-Aksel Sørensen, Peter Clarke, and Jon Crowcroft. Network qos for grid systems. 2003.

[Die99] T. Dierks. *RFC2246.* http://www.ietf.org/rfc/rfc2246.txt?number=2246, 1999.

[Dum01] Edd Dumbill. *Bird's-eye BEEP.* http://www-106.ibm.com/developerworks/xml/library/x-beep/?dwzone=xml, 2001.

[HCF97] Graham Hamilton, Rick Cattell, and Maydene Fisher. *JDBC Database Access with Java. A Tutorial and Annotated Reference.* Addison-Wesley, 1997.

[jak03] jakarta. *Log4j.* http://jakarta.apache.org/log4j/docs/index.html, 1999-2003.

[Jef00] Ron Jeffries. Extreme programming and the capability maturity model. *http://www.xprogramming.com/xpmag/xp_and_cmm.htm*, 2000.

[LC01] Laura Lemay and Rogers Cadenhead. *SAMS Teach Yourself Java 2 in 21 Days Professional Reference Edition Second Edition.* Sams, 2001.

[Pos03] PostgreSQL Global Development Group. *PostgreSQL 7.3 Documentation*, 2003.

[Ray01] Erik T. Ray. *Learning XML.* O'Reilly, 2001.

[Ros01] M. Rose. *FRC3080.* http://www.ietf.org/rfc/rfc3080.txt?number=3080, 2001.

[Ros02] M. Rose. *BEEP*. O'Reilly, 2002.

[Sou02] SourceForge. *Java BEEP Core Java 0.9.07*. http://www.beepcore.org, 2002.