University of Oslo
Department of Informatics

QoS-Aware Remote Bindings in IP Based Mobility Management

Kim R. Bredesen
kimbre@ifi.uio.no

Master Thesis

31st January 2006

Abstract

QoS and mobility is a field that still has some performance issues. QoS degrade when IP based nodes become mobile. This thesis tries to make IP based networks as good as cellular networks, when it comes to mobility.

We focus on solving the added complexity introduced by QoS in mobility management in the middleware layer, inside a remote binding.

The main problem states, that we should find out how a remote binding can adapt and enhance QoS-parameters that perform poorly in IP based mobility management.

The methods used are testing and prototyping. We test existing mobility management software (Mobile IP based) and design, implement and test a QoS-Aware Remote Binding, named MobiBind. A custom traffic generator, KGen, was developed as part of this thesis, in order to performe testing.

In the test of Mobile IP, we found that the QoS parameters; packet loss and seamlessness underperformed. We also found that hand-overs in IP based mobility takes about one second to complete.

MobiBind did manage to increase performance of both packet loss and seamlessness in mobility management. This did however increase the delay (round trip time).

We found that the main problem in mobility management is the hand-overs, and the connection breakage of about one second. This introduces QoS degeneration that in our Mobile IP tests showed up in the QoS parameters packet loss and seamlessness. MobiBind increased performance of these QoS parameters, at the cost of delay. This indicates that there is a tradeoff between the different QoS parameters, and that total QoS might be difficult to increase unless the underlying problem is fixed, in this case; the connection breakage. MobiBind enables application developers to control which QoS parameters mobility management impacts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Several Quality of Service (QoS) demanding applications are frequently used today. Applications like streaming and conversational traffic applications, which include Voice Over IP (VoIP), place great demands on the network in order to give a stable service to the user. Enabling networks to provide complex QoS has been the focus of much research. Elements in the network that in some way manage QoS are, in this thesis, said to be QoS-Aware.

In the 80's big computers shrunk to portable laptops, and today lightweight laptops with several types of communication abilities are common. Wired communication like Local Area Network (LAN), and wireless communication like Wireless LAN (WLAN), General Packet Radio Services (GPRS) and Third-Generation wireless (3G) are commonly installed on laptops. All these network connections make it possible for the laptops to become mobile, maintaining at least one of the network connections.

Mobility management is the field that manages the added complexity that comes when the laptops becomes mobile. The major difference, compared to fixed systems, is the hand-over; the process of changing networks. When changing networks, also possibly changing link technology, QoS parameters that are critical for QoS demanding applications may change dramatically, this is also referred to as varying network conditions. The way networks are built makes it impossible to maintain a connection in this mobile environment. Mobility management has until now focused its efforts on providing basic connectivity.

Users want to use QoS demanding applications not just when they are stationary, but also when they are mobile, this pushes for an integration of QoS into mobility management.

Increased complexity is added to mobility management when considering the other type of mobile nodes, i.e., Personal Digital Assistants (PDAs). PDAs are small devices that have a lower amount of Random Access Memory (RAM), and a smaller Central Processing Unit (CPU) than laptops. PDAs have become powerful enough to be equiped with WLAN and other wireless receivers, but are not as powerful as laptops. Battery capacity of these devices is also lower.

The problem that this thesis focuses on is how to handle QoS in mobility management. Streaming traffic requires seamless data delivery, but how can that be provided when a hand-over causes connection breakage, and violates QoS requirements? How can we take advantage of the different links that are used? These are all questions that stem from the question: How can QoS be handled in mobility management?

Or if related to cellular networks: How can IP based networks handle mobility as efficiently at cellular networks do today?

Middleware is a software layer that lies between the resources and the application. Middleware takes responsibilities from the application, to make the job of the application developer easier.

Middleware can be used to connect two computers. In component-based middleware this is often referred to as connecting two components that are located in different address spaces. A connection between two such components is called a Remote Binding. This remote binding can be viewed as a distributed component, e.g. being located in both address spaces of the components that are using it. This thesis researches the possibility of a remote binding that can manage both QoS and mobility.

1.2 Problem Statement

This thesis has the following problem statement:

> *How can a QoS-Aware Remote Binding, implemented in a distributed component, adapt to the varying network conditions to enhance QoS-parameters that perform poorly in IP Based Mobility Management.*

This main problem statement aims at solving the problem of added QoS complexity in mobility management, inside the middleware.

In order help answer the problem statement, it was divided into three sub-problems:

1. Which QoS parameters underperform in mobility management compared to fixed networks?

2. What would be required of a QoS-Aware Remote Binding from an application developer's point of view?

3. Is it possible to design and implement a QoS-Aware Remote Binding that enhance QoS-parameters performance?

Sub-problem one examines existing mobility solutions to reveal which specific QoS parameters that underperform. Sub-problem two outlines a requirement specification of a QoS-Aware Remote Binding based on the findings from the first sub-problem. Sub-problem three tests an implementation and provides proof that a QoS-Aware Remote Binding can enhance QoS-parameter performance.

Solving these three sub-problems will result in a comprehensive solution to the thesis problem statement.

1.3 Scope

This thesis focuses on IPv4-based networks.

If the middleware does not have bindings that handle QoS and mobility, the application developer would have to design a QoS-Aware binding for each new application, every time. This is both complex and error prone. It is therefore far better that the QoS management software is implemented by an QoS-expert, so for this reason we give the middleware the responibility for QoS.

Although there are several ways to handle the added complexity of mobility management, this thesis focuses on the middleware approach.

Further, this thesis focuses on the mobility management scenario where one computer is mobile and the others are stationary.

1.4 Method

The thesis uses the following methods:

- Test bed to test existing Mobile IP software and the proposed QoS-Aware binding. The test bed is described in Chapter 3.
- Timeframed literature study to find related work and other comparable results to this thesis' Mobil IP test.
- Prototyping to validate the presented QoS-Aware Remote binding, called MobiBind.

Work Breakdown Structure

The test bed is used to test Mobile IP and to answer partly the question stated in sub-problem one. This test and other comparable test results found in the literature study will be analyzed in the Mobile IP test chapter, Chapter 4, to provide a full answer to sub-problem one. Comparing the results from the Mobile IP tests with other test results will verify the identification of the correct under-performing QoS parameters.

The Mobile IP test comes before specifying any requirements of a QoS-Aware Remote Binding. The Mobile IP test's goal is to identify under performing QoS parameters, but the test also functions as a benchmark for a QoS-Aware Remote Binding. The later implemented QoS-Aware Remote Binding, MobiBind, is tested in the same lab, and the results of the two tests can be compared.

Sub-problem two focuses on the application developer's needs. Those needs are addressed by specifying requirements in Chapter 5.

To verify that the requirements are met, prototyping is performed. A QoS-Aware Remote Binding, MobiBind, is designed and implemented. To answer the third sub-problem relating to the performance of such a QoS-Aware Remote Binding, MobiBind is finally tested in the test lab. The test results of MobiBind can then be compared to the original Mobile IP test and analysis, found in the Mobile IP tests Chapter, chapter 4. This final analysis will enable an answer to sub-problem three.

The design was performed using an agile iterative process. The design was developed side by side with the prototype, testing out different design options. Several iterations, with several solution proposals were engineered. The design chapter only outlines the final design of MobiBind, and any references to "tests" in the design chapter, Chapter 6, are to the iterative testing performed.

The test bed was used frequently during development and testing of MobiBind. Sadly one key element of the test lab, the laptop named QuA1 died during the final stage of the MobiBind test. The replacement of the laptop was outside our control, and was not done. Some quantification tests, among them the test to quantify the overhead the remote binding produced, could not be performed. MobiBind was still comprehensively tested, but some results could not be quantified.

Method Motivation

The motivation for combining three research methods was that by starting out by testing existing technologies, like Birdstep's Mobile IP software the thesis grounds itself in the real world. The literature study enables the thesis to develop ideas inline with the state-of-the-art. The prototyping enables validation of the ideas presented, making this thesis a part of the state-of-the-art.

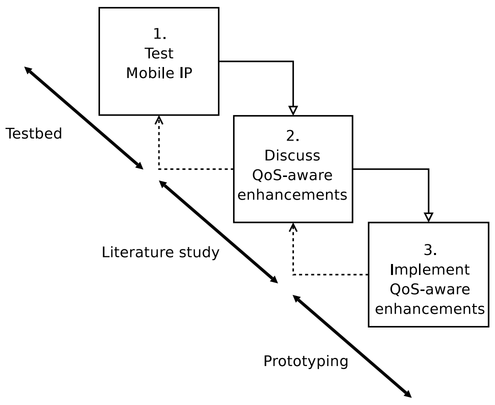

Figure 1.1: Research Method Integration

1.5 Summary of Results

This thesis identifies, by testing, QoS parameters that underperform in the current standard Mobile IP, compared to a fixed environment. QoS-parameters like round trip time and packet loss increase with the use of Mobile IP. Mobile IP does not provide seamless data connectivity either. This thesis identifies several key requirements for a QoS-Aware Remote Binding, the most important of which are reducing packet loss and providing seamless data connectivity. This thesis also designs, tests and implements a QoS-Aware Remote Binding called MobiBind. Through testing we found that MobiBind meets the requirement of seamless data connectivity and reduces packet loss; in part by the use of the harvest mechanism suggested in this thesis.

1.6 Thesis Structure

The thesis is divided into the following chapters:

Chapter 2 gives a background and description of the various technologies relevant to this thesis, and their related work.

Chapter 3 describes the test bed and the traffic generator designed for testing Mobil IP and MobiBind. This chapter can be seen as an extension of the method described in Section 1.4.

Chapter 4 describes the tests performed on Mobile IP, as part of sub-problem one. Results and findings of this test are also analysed.

Chapter 5 discusses and specifies requirements for a QoS-Aware Remote Binding. The requirements are partly based on the Mobile IP tests.

Chapter 6 discusses the design of MobiBind, a QoS-Aware Remote Binding based on the requirements specified in Chapter 5.

Chapter 7 describes the implementation of MobiBind.

Chapter 8 presents tests of MobiBind, and analyses the results and findings of the tests performed.

Chapter 9 evaluates MobiBind, the implemented QoS-Aware Remote binding, against the requirements. It also evaluates whether the research questions studied in this thesis has been answered.

Chapter 10 concludes this thesis and points out suggested further work.

Chapter 2

Background and Related Work

The QoS-Aware Remote binding spans several research domains. In order to fully understand all the discussions around this binding some knowledge about each research domain is needed. This chapter provides the relevant background needed from the different research domains.

Section 2.1 gives an overview of the challenges and existing solutions for mobility management. Section 2.2 follows up with Mobile IP, an IETF proposal to a solution for mobility management in IP based networks. Middleware is discussed in Section 2.3. Section 2.4 highlights important issues related to QoS and traffic classes. Finally, remote bindings are discussed in Section 2.5.

Related work of this thesis is presented in the relevant sections.

2.1 Mobility Management

Mobility is a field that spans several domains; mobile computers as well as cellular phones and many types of network technologies (e.g., global system for mobile communication (GSM), GPRS, 3G and IP-based networks). The largest data network in the world, the Internet, must also solve mobility.

This section looks into session mobility in a general way, starting with a background to the mobility problem. Then Section 2.1.2 goes into the mobility problem in more detail. Lastly, possible solutions to the mobility problem are discussed in Section 2.1.3.

2.1.1 Mobility Background

With the dramatic increase in memory and CPU capacity, combined with longer battery capacity and lighter laptops, it has become more conveni-

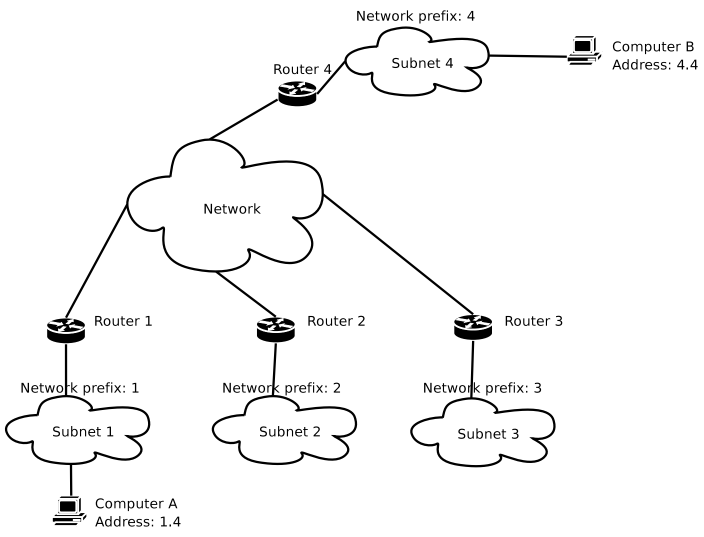

Figure 2.1: Fixed Network

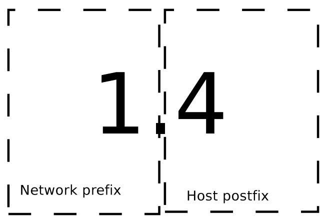

Figure 2.2: Network Address

ent to move the laptops around. Due to certain restrictions in the way most networks are constructed, this becomes a problem.

Generally, networks are divided into sub-networks. One of the reasons for doing this is to cut the size of the routing tables. Instead of remembering each node's address, routers only remember which network the node is attached to.

Figure 2.1 shows a typical network scenario: Two computers that want to communicate across a large network. The computers are located on different sub-networks and hence the traffic has to be forwarded by routers. All computers in the network have an address that is constructed of a host part and a network part. The host part, called host postfix, uniquely identifies the node within the sub-network. The network part, called network prefix, uniquely identifies the sub-network. When the two are combined the address becomes unique for the entire network. Computer A has in Figure 2.1 the address 1.4. As Figure 2.2 shows the

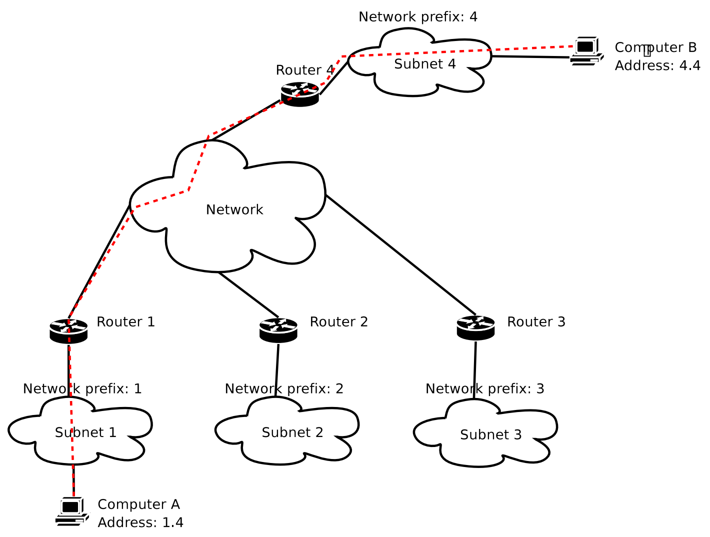

Figure 2.3: Data Packet Through Fixed Network

network prefix is 1 and the host postfix of computer A's address is 4.

When computer B wants to send a data packet to computer A, it marks the data packet with computer A's address and sends it to router 4. When router 4 gets the packet it looks only at the data packet's network prefix. Based on that the router decides to send the packet to router 1. Router 1 sees that the packet is meant for sub-network 1, and sends it out on that sub-network, were computer A receives it (see Figure 2.3).

Keeping the same address when moving between sub-networks would cause the packets to arrive at the wrong sub-network. This is the basis for the entire mobility problem, explained in detail in the following section.

2.1.2 The Mobility Problem

Mobility is introduced if one of the computers, let's say computer A, starts to move from one sub-network to another. The mobility definition used in this thesis is [33]:

> A node's ability to change its point of attachment while maintaining all existing communications.

If computer A changes its attachment point from sub-network 1 to sub-network 2, computer B's packets would not be forwarded correctly by the routers. Therefore mobility is not achieved, as illustrated in Figure 2.4. Mobility is not achieved because Computer B still uses the same address for computer A. Since the routers only look at the network prefix,

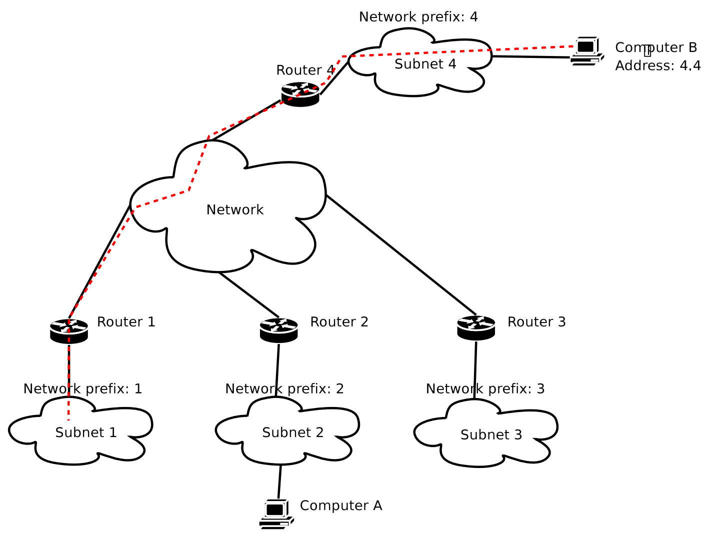

Figure 2.4: Mobility Problem in Finex Network

and hence the packets are still routed to sub-network 1. The packets are therefore lost in sub-network 1 and never reach computer A.

In the field of mobility, computer B, the computer that tries to send data to a mobile node, is often called Corresponding Node. Computer A, that moves, is called Mobile Node. The process of changing sub-networks is called hand-over.

Mobility can be divided into macro and micro mobility. Micro mobility is when the computer moves within one single administrative domain, and macro mobility is when computers move globally or to another domain. The possibility for an entire sub-network to change its point of attachment is called network mobility and mobility within ad-hoc networks is called ad-hoc mobility.

Mobility may introduce gaps in the network connection, where the node is not connected to any network. If a node wants to move from Ethernet (802.3) to Ethernet using only one network card a gap is introduced when the user switches Ethernet cables. In this gap there obviously can be no network connection, since no cable is connected to the computer. This type of hand-over is called hard hand-over. Hard hand-over means that the node disconnects from the current network before it connects to a new one. If a computer has both the old and the new network available when performing the hand-over it is called a soft hand-over. A soft hand-over has the potential to be performed seamlessly, e.g., the user does not notice the hand-over.

2.1.3 Mobility Solutions

Mobility is a complex issue that can be solved several places in the OSI reference model. The link, network, or application layer may be places to solve the mobility problem. Wireless ATM [3] provides mobility support at the link layer, Mobile IP at the network layer. An application layer approach to solving mobility is also possible, but would perhaps not be a wise choice since each application wold have to become mobility aware, and this approach would not be backward compatible with old applications. A network layer solution, on the other hand, gives backward compatibility since the applications would not have to become mobility aware.

Examples of mobility management that are widely used are found in GSM, GPRS, and universal mobile telecommunications system (UMTS). Mobility management in the telecommunications industry is solved, this thesis however focus on IP based networks, like the Internet. Solution proposals for enabling mobility in IP based networks include Mobile IP [46], Hawaii [28], Cellular IP [43], and host based routing.

ALICE [24] is a mobility management solution that enables mobility in CORBA. This approach has no QoS-Awareness.

Host based routing enables the routers to route on the basis of the entire IP address, instead of just the network prefix. This approach would require that all routers have an entry in their routing table for all Mobile Nodes. While this works in small networks, it is not scalable. The routing tables would become huge, and require much memory. Lookup time would increase and a high number of update messages would have to be sent.

IETF has proposed a mobility solution for IPv4 based networks called Mobile IP, which several WLAN operators are on the brink of providing to their subscribers. Furthermore, for 3G mobile communications systems IP version 6 (IPv6) with Mobile IP support has been specified for IP multimedia [18]. The technology used for the IPv6 solution is based upon the IETF's mobility solution for IPv4. Hence, this thesis does not cover IPv6 mobility management. For details of mobility management in IPv6, see [26].

The implications with regards to QoS and mobility will be discussed in Section 2.4.5.

2.2 Mobile IP

Mobile IP works by making it appear to the rest of the network that the mobile node is always addressable in its home network.

The technology used for Mobile IP IPv4 is defined in a series of Re-

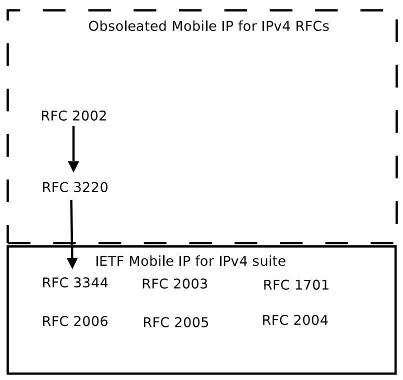

Figure 2.5: Mobile IP RFC History

quest For Comments (RFC). Figure 2.5 shows which ones are valid today, and the previous RFCs. RFC 3344 is the current RFC specifying the main functionality of Mobile IP, the others specify helping technologies including tunneling and management capabilities.

2.2.1 Mobile IP Details

Mobile IP solves the mobility problem by adding some new network elements without requiring changes to existing elements, except the Mobile Node itself. The new elements are the Home Agent and the Foreign Agent. The Mobile Node (computer A in the Figures) is also extended with new functionality. Computer B, the corresponding node, does not require any changes, as it is not aware of the fact that computer A is mobile.

The Mobile Node has a home network; usually the network in which the Mobile Node is most often located, or where the Mobile Node belongs. The home network has to have a Home Agent. This is also were the Mobile Node receives its home IP address from. In Figure 2.6 sub-network 1 is computer A's home network.

The Home Agent may be implemented in a router, for example router 1 in Figure 2.6, or it may be implemented as a computer inside a home network, for example sub-network 1 in the figure. The Home Agent is responsible for forwarding data packets to the Mobile Node when it is connected to a foreign network.

The home network may also be virtual. In Figure 2.6 this could be done by implementing the Home Agent in router 1, and making sub-network 1 virtual. When the home network is virtual the Mobile Node would never be home, and the Home Agent would always operate and tunnel packets to the Mobile Node.

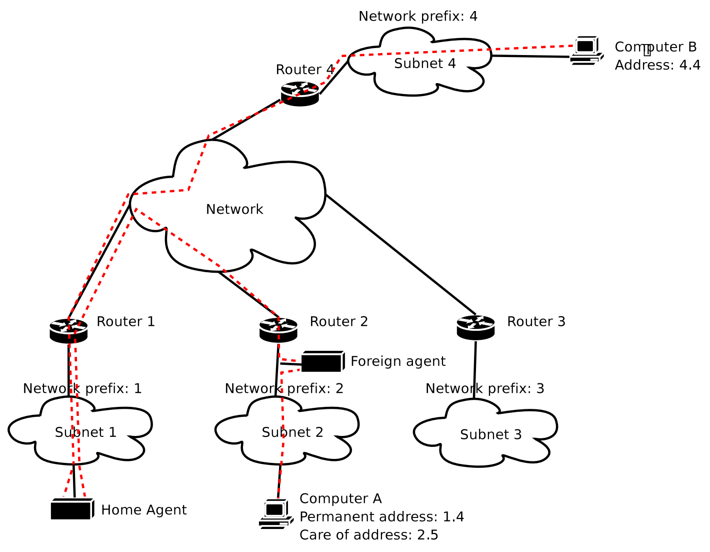

Figure 2.6: Mobility in Mobile IP

When the Mobile Node moves from sub-net 1 to sub-net 2, it connects to the Foreign Agent located in that network. Sub-network 2 would then be a foreign network to the Mobile Node. The Foreign Agent's job is to allocate a temporary care-of IP address to the Mobile Node while it is located in the foreign network. The Foreign Agent also helps the Mobile Node with registration to the Home Agent, and serves as a de-tunneling point for the Home Agent.

A Foreign Agent may serve several foreign networks. All it needs is a connection or route to the network. If a router is a Foreign Agent for more than one foreign network the care-of-address received by a Mobile Node may not match the network prefix of the sub-network on which it is currently located. The care-of-address may have the network prefix of another sub-network which the router is a Foreign Agent to.

Some tests of Mobile IP have been performed. Espen Sagen [29] focuses on testing end-to-end performance during hand-over. Koucheryavy Y. et al. [19] tests WLAN under different conditions.

Service Advertisement

When a Mobile Node wants to attach to a foreign network it needs to discover a care-of-address for that network. This is done using modified Internet Control Message Protocol (ICMP) messages. The messages include available care-of-addresses along with other control information. The same messages are used by the Home Agent and the Foreign Agent.

The Foreign Agent sends these messages to advertise its services, a Mobile Node uses them to determine which Foreign Agents it has contact with.

A Mobile Node may also receive a care-of-address of a foreign network by itself, typically using Dynamic Host Configuration Protocol (DHCP). When a care-of-address is received by the Mobile Node alone, it is called a co-located care-of-address.

Registration

Once a Mobile Node is located in a foreign network and has received a care-of-address from a Foreign Agent, it sends a registration request to the Foreign Agent. The request contains the following information:

- Mobile Node's home address
- care-of-address
- registration lifetime desired
- encapsulation desired
- unforgeable replay protected authentication

Unless the request contains errors the Foreign Agent usually accepts the request and keeps track of it. At the same time it sends the request to the Home Agent. When the Home Agent receives the registration request it can start to forward packets to the Mobile Node's care-of-address. The Home Agent replies with a registration reply message, also containing authentication.

Tunneling

The Mobile Node receives a care-of-address from the Foreign Agent. The care-of-address is an address that is valid in its current sub-network. On behalf of the Mobile Node the Foreign Agent sends the care-of-address to the Home Agent.

If a corresponding node wants to send a data packet to the Mobile Node, while the Mobile Node is in a foreign network, it can still use the same original (home) address of the Mobile Node. The packet it sent from the corresponding node to the mobile node's home network. In the home network the Home Agent intercepts the packets going to the Mobile Node's home address. It then tunnels the datagram using the Mobile Nodes care-of-address. The Foreign Agent de-tunnels the packet and delivers it to the Mobile Node. This is shown in Figure 2.6.

There are several tunneling mechanisms available, the most common is IP-within-IP and is described in RFC2003. Other alternatives are Minimal Encapsulation and Generic Routing Encapsulation (GRE).

Using this technology the Mobile Node may change sub-networks while maintaining ongoing communications, thereby achieving mobility.

Route Optimisation

In Figure 2.7 the Mobile Node is shown to be able to deliver packets the shortest way by the Mobile Nodes default router, the Foreign Agent. This is because when the Mobile Node sends packets it addresses the corresponding node directly, and does not use its care-of-address as sender, but the home address. The correspondent node, on the other hand, delivers packets to the home address, so all packets are sent through the home network and the Home Agent. This is called triangle routing. It represents a routing asymmetry that is potentially annoying to the mobile users; the asymmetry caused by triangular routing can cause some protocols to underperform. It also sets up the Home Agent as a single point of failure.

Since the beginning of the Mobile IP design this weakness has been recognised, and attempts to remedy the problem basically consists of delivering the care-of-address of the Mobile Node to the correspondents. So that corresponding nodes does not have to send packets through the Home Agent. Unfortunately all realistic solutions proposed so far have the disadvantage that it either requires change in the corresponding node to keep state about the Mobile Node, the Mobile Node's care-of-address, or change is required in selected routers to keep the same state information.

2.2.2 Mobile IP Security

Security is implemented in several aspects. Mobile IP is designed so that the corresponding node does not know where the Mobile Node is located. This design decision was based partially on a security concern; the corresponding node should not be able to track the movements of the Mobile Node.

Another security issue is that of stealing traffic from the Mobile Node. This could be done by sending a false registration request to the Home Agent. This would enable an intruder to redirect traffic that should go to the Mobile Node anywhere the attacker pleases. In short all communication in Mobile IP needs to be verified so that an intruder may not send false requests or acknowledgements. This can be done by a Diffie-Hellman key exchange or a private / public security association. Security is not the focus in this thesis, so for further details on security handling in Mobile IP, please see [33].

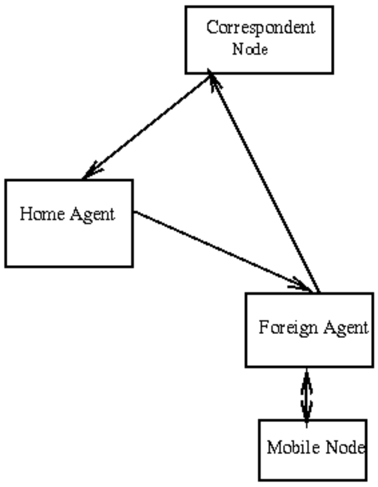

Figure 2.7: Triangle Routing

2.2.3 Reverse Tunneling

Mobile IP was designed for use with classical routing, where data packets are routed on the basis on the destination address, and nothing else. Classical routing is still the main way of routing packets, but firewalls and Ingress Filtering do not operate like classical routing.

Ingress Filtering [10] is a relatively new technique that limits access to the Internet. This is done by only allowing packets to go out of an administrative domain if the packets have a source address that exists within that domain. Thus Ingress Filtering breaks one of the traditional assumptions of IP routing. However Ingress Filtering has found support due to its ability to help prevent spoofing attacks.

Mobile IP uses its home address as source when it sends packets from a foreign network and they will be stopped if Ingress Filtering is in use at the foreign network. The solution proposed for this is reverse tunneling. Reverse tunneling works so that the Mobile Node tunnels the packets to the Home Agent, thereby in effect turning Mobile IP in to a virtual network. This approach also makes the Home Agent the single point of failure, both ways in the communication between the Mobile Node and a correspondent node.

2.2.4 Mobile IP Without Foreign Agents

The Mobile Node may be configured to function without a Foreign Agent in a foreign network. The tasks normally performed by the Foreign

Agent are then performed by the Mobile Node itself. This includes de-tunneling, registration request and IP address assignation. The IP address that the Mobile Node used to get from the Foreign Agent is now given by a DHCP server in the sub-network.

This approach has both advantages and disadvantages. The advantage is that the Mobile Node does not need Foreign Agents to connect to a foreign network. This is useful in today's Internet, as few networks have a Foreign Agent. A disadvantage is that the hand-over may not go seamlessly due to (1) delays in the communication to the DHCP server and gracious Address Resolution Protocol (ARP), and (2) that packets sent to the old foreign network are lost in that network before the Home Agent starts to re-send packets to the correct care-of-address. A Foreign Agent could collect these packets and send them to the new foreign network at the Mobile Node's request.

2.2.5 Summary

The first problem of mobility was to provide connectivity between the moving computers, technologies like Mobile IP has enabled that connectivity. The problem of enabling more complex services stands next in line. Complex applications often require more than just the basic connectivity, real time connectivity add requirements. One way of managing these complex requirements is to introduce middleware as a place to resolve the complexity. The transparency of middleware can help to provide better service. Middleware is discussed in the following section.

2.3 Middleware

2.3.1 Introduction

Traditionally middleware was used as software that facilitated remote database access and system transactions. More recently the term has come to be associated with distributed platforms.

This thesis defines middleware as software designed to manage the complexity and heterogeneity of distributed systems and thereby provide a simpler programming environment for application developers. Network, hardware, OS, and programming language heterogeneity are some examples of where complexity is added in distributed systems. Distributed systems are defined by [6] as: Systems in which hardware and software localised in a network of computers communicate and coordinate their actions only by sending messages to each other (from [6]).

A way of managing the complexity of distributed systems is by hiding heterogeneous properties of the system, making the distributed system look like a black box. The term black box is used to illustrate that the middleware provides functionality, where the implementation is hidden inside the middleware. Hiding in middleware is called transparency and a list of different types of transparencies are defined by [5] and [6]:

Access transparency enables local and remote resources to be accessed using identical operations.

Location transparency enables resources to be accessed without knowledge of their location.

Concurrency transparency enables several process to operate concurrently using shared resources without interference between them.

Replication transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of their replicas by users or application programmers.

Failure transparency enables the concealment of faults, allowing users and application programmers to complete their tasks despite the failure of hardware or software components.

Mobility transparency allows the movement of resources and clients within a system without effecting the operation of users or programs.

Performance transparency allows the system to be reconfigured to improve performance as load varies.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

The middleware software lies above the OS, extending its services and masking heterogeneity. Further, the middleware software lies under the application so it can be used by the applications. Typical services that middleware provides are the ability for programs to communicate even if they are made in different programming languages, e.g., the Common Objects Request Broker Architecture (CORBA) and Microsoft's Distributed Component Object Model (DCOM).

Figure 2.8 illustrates the usefulness of middleware, and what challenges distributed application developers face. The goal is to make a simple client - server application. Without the middleware, Figure 2.8a, the application designer would have to figure out how socket creation is done on both the client and server. He would also need to design a protocol to be used between the client and server program. Figure 2.8b

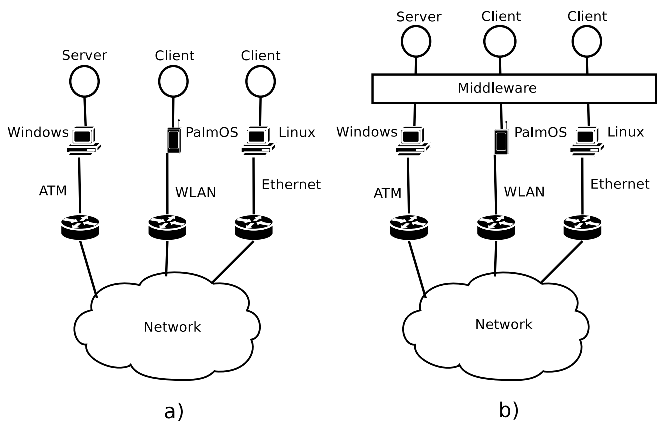

Figure 2.8: Example of Middleware

illustrates the same scenario with the use of middleware. The heterogeneity of the computers and network are transparent and the application designer does not see it as a set of computers, but as one middleware platform. Thus the application can call operations between the client and server as they were local calls i.e., access transparency, and would not need to worry about socket setup or protocol design.

In addition to simplifying the programming models, the middleware offers services like database access, transaction control, persistence and security, making the application more light weight. These added services may be explicitly requested by the application, shown in Figure 2.9b, enabling the application to use the services when necessary. This approach may be error prone; an application designer may forget to call security services in some code were it should be called, exposing a potential back door to the system. To overcome this problem another way of using the middleware services exist, called implicit services. The implicit approach, shown in Figure 2.9a, intercepts messages to and from the application, making sure that all the messages goes through, e.g., security handling.

2.3.2 Component Based Middleware

The server application in Figure 2.9 is an application that is designed as a black box. There is no way of knowing how it operates and how it communicates with its clients. This means that if a third party would like to add anything to a client or server it would not be possible, only the party that created the application would have that knowledge. The limitations of this black box approach has led to a new, component based design, which is more open.

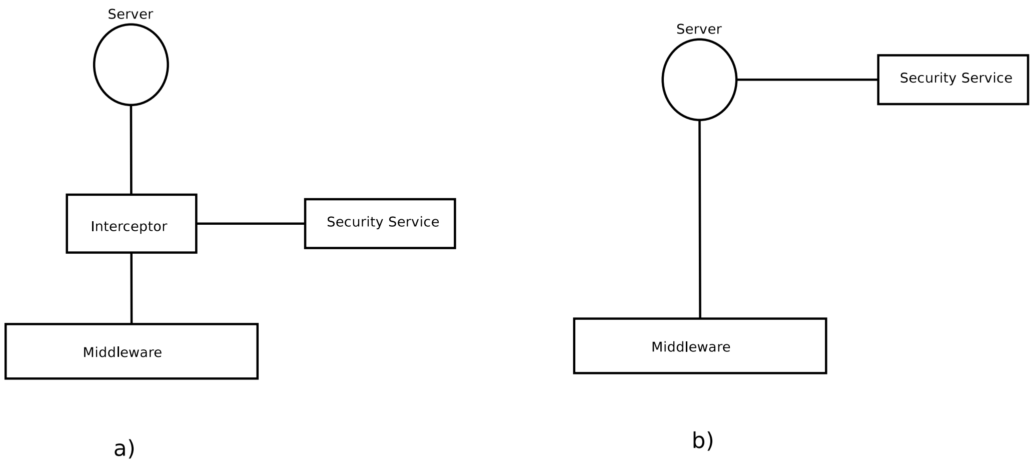

Figure 2.9: a) Implicit Middleware and b) Explicit Middleware

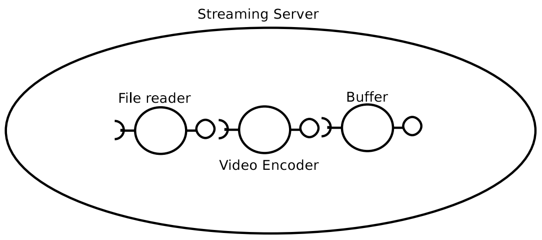

Figure 2.10: Middleware Components

The definition of a component used in this thesis is as follows [36]:

> *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

The component based approach would allow a server to be constructed using components developed by third parties. A video streaming application could be composed as shown in Figure 2.10. The components could also be changed if new and better technologies became available in some areas. The component based approach can also be called a plug and play approach. Developing is done by plugging existing components together. Components may also consist of several smaller components, making it possible to view the component at different granularities. Components constructed in this way are often referred to as composite components.

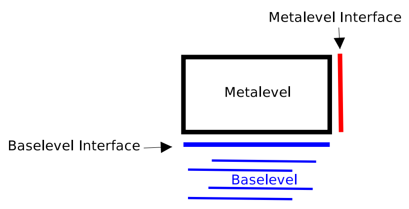

Figure 2.11: Meta- and Base-Level

Examples of middleware platforms for component based applications are Enterprise Java Beans (EJB) [21], Microsoft .NET [23], and CORBA Component Model (CCM) [15].

2.3.3 Reflective and Adaptive Middleware

Reflection and adaptation make the middleware self-aware. Reflection enables the middleware to inspect its internal state and environment. Adaption enables the middleware to change its behaviour and functionality.

To enable this self-awareness, the middleware consists of two levels, referred to as meta- and base-level, see Figure 2.11. The base-level is where the actual working components are located. The meta-level is a high-level representation of the base-level. The meta-level is also called a self representation. The two levels are causally connected. This means that alterations made in the meta-level is absorbed down to the base level, and vice versa. A system could also adapt on the basis on the reflection without any external involvement, making the system self-adaptive. The meta-level contains information of the system and methods for changing the system's behaviour. Only the meta-level is available for reflection and adaptation.

Examples of reflective and adaptive middleware systems are OpenORB[13], OpenCorba[42], QuA[2], DynamicTAO [9], and MADAM [7].

OpenORB provides a meta-level interface of different types to enable reflection based on this meta level interface. OpenCorba has the same meta-level notion, with a focus on providing reflection and dynamic adaptability in CORBA. DynamicTAO provides a reflective ORB, that focuses on handling extended resource and security concerns in the mobile domain. Finally, MADAM provides support for dynamically adaptive applications in the mobile domain.

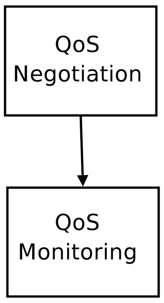

Figure 2.12: Basic QoS

2.4 Quality of Service

First QoS is discussed with focus on traditional network QoS, then the added complexities of mobility management in QoS is discussed.

2.4.1 Introduction to Quality of Service

QoS is defined in many ways, but the definition that will be used here is modified from Vogel et al. [45]:

> *QoS represents the set of those quantitative and qualitative characteristics of a system that are necessary to achieve the required functionality of an application.*

As networks have gone from homogeneous to heterogeneous and applications have become increasingly complex, the QoS requirements of the application have changed. An old QoS requirement is that of preserved payload content. This means that what the sender sends on one side of the network is the same as the receiver receives. This is a basic QoS requirement that is embedded in the TCP/IP protocol. Today QoS requirements are more complex.

QoS is experienced by the user, so it is the user that specifies the QoS requirements. If the user wants to see a streaming video the user may wish to see it in colour and in full screen. These requirements needs to be mapped down to something that the underlying system can understand. The system needs to know which system resources it needs to reserve in order to make the user's requirements possible. In this example the system may need to make sure a colour screen is available and that enough network bandwidth is available to sustain a full-screen stream. Computer resources like the CPU would also need to be reserved. If the system can accommodate these requirements, the resources are reserved and the playback of the stream can begin. The system needs some way of

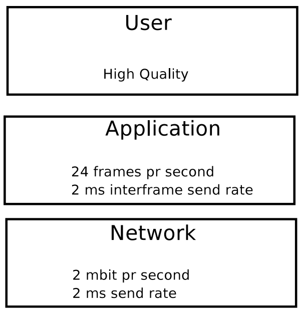

Figure 2.13: QoS Parameters and the OSI Model

controlling the application's use of the resources. If the application uses more than it originally specified, it may effect other applications that are using the shared resource. Monitoring of the application's use of the recourses is therefore necessary. Figure 2.12 illustrates this process.

If the system finds, by monitoring, that one of the applications are using more than it is allowed to, it has to take action, either by reducing the application to best effort or by forcing renegotiation of QoS.

[8] has a QoS-Aware solution that depends on RSVP. The solution is built on Java-RMI and aims to make Java-RMI time predictable. The solution requires that the network supports RSVP and resource reservation. The solution is not mobility enabled.

2.4.2 QoS Parameters

Today several ways of solving the QoS problem and several ways of specifying the QoS parameters exist. Some QoS parameters are well known, like delay, jitter, throughput, data-packet size and packet loss. Furthermore, QoS parameters are specified in different ways according to where in the OSI model they are used. User's QoS parameters are often based on vague descriptions of quality. A user may say that she wants to see a video in high quality. If we assume that QoS is managed on the network layer, "high quality" must be mapped down to parameters that the network can understand. This has to be done via the application layer. So from high quality the application decides that 24 frames per second need to be sent, with 2 ms delay between the frames. The application calculates the necessary throughput required for sending the frames across the network and asks the network layer QoS management service to reserve the throughput and maximum delay. The QoS parameter defined by the user as "high quality" has now been mapped down to the network QoS layer. Figure 2.13 illustrates this.

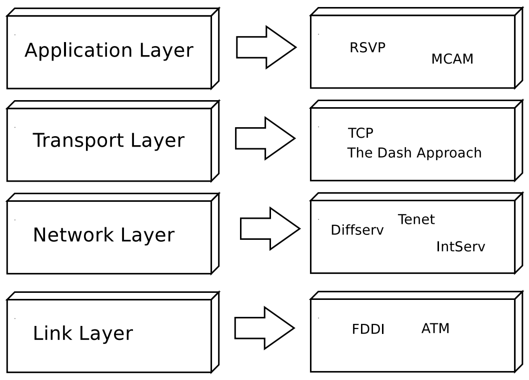

Figure 2.14: QoS Protocols and the OSI Model

2.4.3 QoS and the OSI Model

The QoS problem can be solved at different places in the OSI model. Figure 2.14 shows some of the protocols that have been constructed at different layers in the model. Solving QoS at different layers has different implications, which are discussed below.

Link Layer

Token ring based networks can guarantee maximum delay and reserve resources based on the token, while Ethernet is non-deterministic and can not guarantee QoS.

QoS managed at the link layer is only valid for one link. Solving QoS here has its limitations. In a homogeneous network, QoS could be reserved for the entire network using link layer QoS management. Large networks are often heterogeneous and QoS would be complex to set up. If some portions of the network were based on Ethernet it would be impossible.

Network and Transport Layer

Solving QoS at the network and transport layers allow for QoS management for entire networks, even if the networks are heterogeneous. In the pre-multimedia age, data transport mainly focused on fairness and uncorrupted delivery. Continuous media have quite different communication needs. Continuous media requires that streams be sent and delivered steadily. This forces the network to provide guaranteed delay and throughput.

Application Layer

QoS management at the application layer assumes scalable media. The application adjusts the media according to available resources. This approach gives full freedom to the application developer, but also greatly increase the application's complexity.

2.4.4 Renegotiation

Applications may change their QoS requirements, or a user may decide to go from bad quality video to high quality. This change would require more throughput and, therefore, a renegotiation of the QoS parameters. Renegotiation adds an additional step to QoS management, illustrated in Figure 2.15.

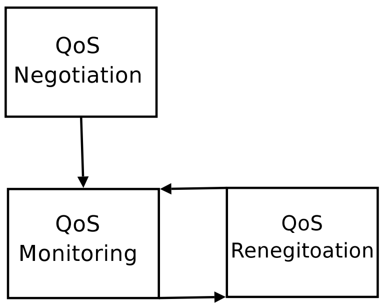

Figure 2.15: QoS Renegotiation

2.4.5 QoS and Mobility

The QoS described up to this point is what we can call traditional network QoS. The renegotiation described focuses on the user. This definition of QoS is still valid, but additional problems arise when mobility is introduced. If QoS management operates on a network and nodes that are mobile, the QoS management mechanism needs to be aware of this in order to guarantee QoS regardless of the mobile setting. This is where this thesis focuses its research.

The first problem for guaranteeing QoS in a mobile enabled network is that during a hand-over the link is usually severed, creating a non seamless environment. Guaranteeing QoS for some complex traffic classes will therefore not be possible under the hand-over.

The hand-over is initiated because of the need to go over from an old link to a new link. Not only is the hand-over process itself difficult, when it comes to QoS, but the new link may also cause problems. If a

Traffic class	*Conversational class*	*Streaming class*	*Interactive class*	*Background class*
Fundamental characteristics	Preserve time relation(variation) between information entities of the stream. Conversational pattern.	Preserve time relation (variation) between information entities of the stream	Request response pattern. Preserve payload content	Destination is not expecting the data within a certain time. Preserve payload content
Example of the application	Voice	Streaming video	Web browsing	background downloads of emails

Table 2.1: UMTS QoS Classes

Mobile Node has some QoS guarantees when connected to a LAN link. When the Mobile Node changes link to perhaps GPRS some of those QoS guarantees might not be kept any more, due to the restrictions of a GPRS link.

Changing links and hand-overs needs to be handled either by involving the end user or solving it in the middleware. A method for involving the end user is by allowing renegotiation to be initiated not only by the user, but also when a new link is taken in use. This might ultimately involve the user, if the renegotiation ends up with lower QoS, the user might prefer ways to adapt to the new limitations.

2.4.6 Traffic Classes

Applications have different QoS requirements. Some require low delays, while others require high throughput. QoS traffic classes categorise applications and their traffic so that instead of talking about specific application QoS characteristics, one can generalise and group applications into traffic classes. This thesis applies the traffic classes specified by 3G Partnership Project (3GPP). 3GPP has specified four classes: Conversational, interactive, streaming and background. The main distinguishing factor between these classes is how delay sensitive the applications are. Conversation defines traffic that is very delay sensitive while the background class is the most delay insensitive traffic class. Table 2.1 lists fundamental characteristics and examples of each traffic class. For further details please see the 3GPP Quality of Service concepts and architecture specification [1].

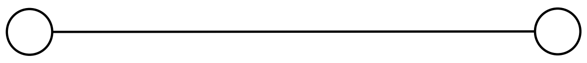

Figure 2.16: Implicit Binding

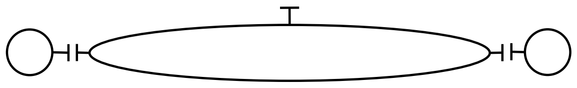

Figure 2.17: Explicit Binding

2.5 Remote Bindings

2.5.1 Introduction to Remote Bindings

In the middleware scenario, a way to connect components is needed. This functionality is provided by bindings. Bindings enable communication between components. The bindings can be divided into two main categories:

- Local Bindings
- Remote Bindings

Local bindings take two components that exist in the same address space and enable communication between them. In its simplest form, a local binding is a binding that uses simple references to enable communication.

Remote Bindings take two components that exist in different address spaces and enable communication between them. This is a more complex procedure, since simple reference addressing is not an option. Usually it takes a complex structure that includes information on which address space the two components are to hold information that the remote binding needs.

While a local binding and an address reference is small, a remote binding might be large; in Orbix [17] the size of a remote binding is 350 bytes.

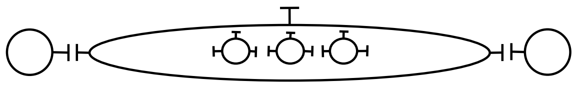

Figure 2.18: Open Binding

The problem of remote bindings can be solved in several ways. Some basic requirements of remote bindings are discussed below. The three types of binding; implicit, explicit and Open are discussed. OpenORB [13] and ReMMoc [14], implementations that focus on these bindings, are also discussed. The binding types are taken from [11]. More information can be found in the book of Gordon Blair and Jean-Bernard Stefani [4].

2.5.2 Implicit Remote Binding

The fist generation bindings were Implicit bindings. Implicit bindings are used in CORBA and use a black box approach. This is illustrated in Figure 2.16. The two components, located on different address spaces, have a connection to each other, but the connection cannot be accessed in any way. This is in accordance with the transparency principle in middleware. The two components do not need to know that the communication is remote, as the binding is supposed to handle all the communication.

2.5.3 Explicit Remote Binding

An explicit binding is addressable for the components. A typical remote binding is illustrated in Figure 2.17. The Explicit remote binding has a interface that the data producing components connect to, and a controlling interface, that are used to control and communicate with the remote binding. There are two important things with explicit bindings:

- The act of creating a binding can subsume static QoS management functions and other static operations.
- The controlling interface can be used for dynamic QoS management.

The controlling interface allows for inspection and adaption.

The explicit remote binding is both visible and controllable. This enables more control of the explicit bindings than implicit bindings.

Implementing a Explicit Remote Binding

The explicit remote binding is, like all remote bindings, represented in two address spaces. When implementing a remote binding, the implementation also has to be represented in both address spaces. Implementing the controlling interface, the one on top of the explicit remote binding in Figure 2.17, requires special attention due to the fact that the remote binding is implemented in both address spaces. If just one interface is available, it has to be implemented on one side, illustrated in figure

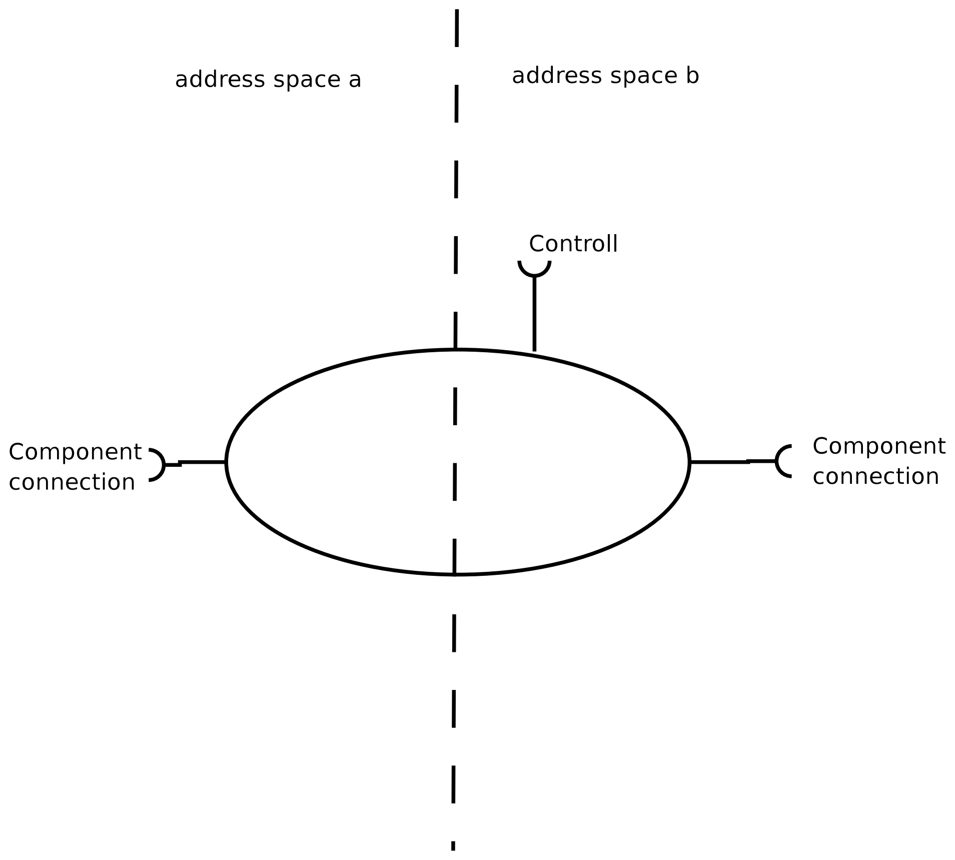

Figure 2.19: One Controlling Interface

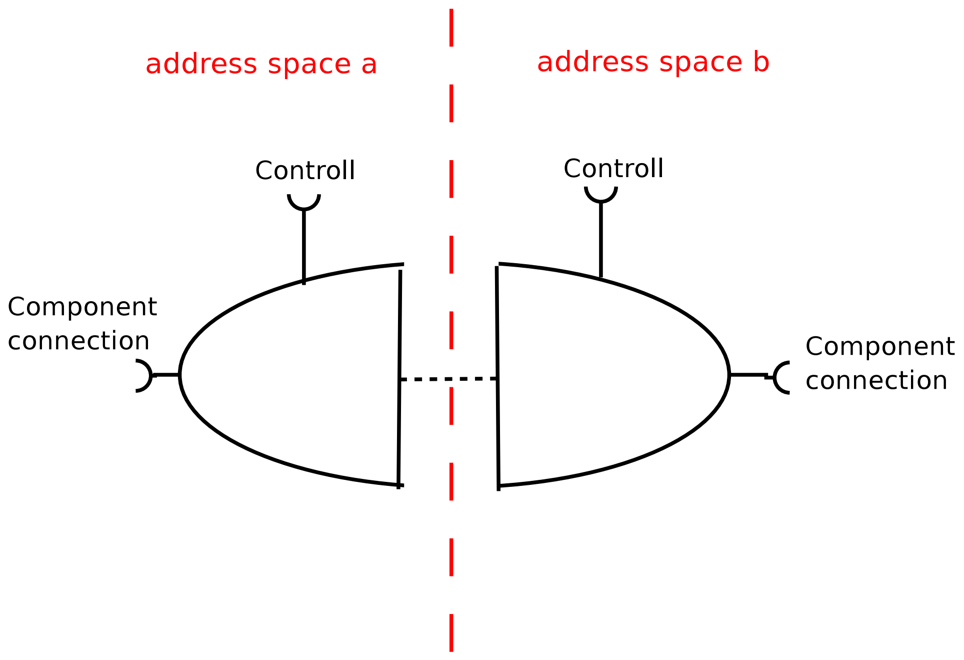

Figure 2.20: Two Controlling Interfaces

2.19. Another possibility is to have the controlling interface represented in both address spaces, illustrated in Figure 2.20.

Implementing just one controlling interface allows just the one side to have access to it. Implementing two allows both side to control the binding.

There exist other ways of handling the controlling interface problem. The above solution describes a solution that handles the problem inside the remote binding. The problem can also be handled outside the remote binding. A RMI type of access mechanism can be implemented to make several access one interface, or a receptacles, described in [22]. The Receptacles catch all the calls to the controlling interface and redirects them to the one controlling interface.

2.5.4 Open Remote Binding

Open bindings build on explicit bindings, but allow for more adaption and inspection. The binding not only has data and controlling interfaces, but the objects inside the binding have the same type of interfaces. The inside structure of the Open Remote Binding is also available through an object graph. The inside structure can be manipulated through the open remote bindings controlling interface. The inside objects can also be reached through the controlling interface, and be manipulated in the same way. Figure 2.18 illustrates an open remote binding.

The Components inside the Open Remote Binding may be specialised to operate in a special setting, e.g., using a UDP binding in the centre, with a MPEG-2 decoder on both sides of that UDP binding. In other situations a lossless decoder will be preferable. To be able to provide a open remote binding for different situations, a Binding Factory was created. The Binding Factory constructs the Open Remote Binding based on some requirements. For instance, lossy or lossless decoding may be set as a parameter when requesting a remote binding from the Binding Factory. The Binding Factory will then take available components and construct an Open Remote Binding that best suits the requirements. The newly created Open Remote Binding is then given back to caller.

There may also exist a hierarchy of Binding Factories. The application may call a high level Binding Factory to construct a Remote Binding with some requirements. The high level Binding Factory might then call other low-level Binding Factories to get components that it can use in its component graph.

Different applications may want to control different levels of the OSI model. Some applications might just want a Open Remote Binding that handles low level things like connectivity, and control other high levels in the application, others might want the Open Remote Binding to handle buffers, encoders and so on. The Open Remote Binding's structure reflect this. The Open Remote Binding is often built with the lower portions of the OSI model in the centre of the binding. Other, more high level components, then encapsulate the "basic" component to provide a more complex service. This building allows for both the applications and Binding Factories to construct a Open Remote Binding that satisfies the requirements of the using components.

2.5.5 Remote Bindings and Mobility

Remote bindings in conjunction with mobility have been the area of much research. The research has focused on providing connectivity despite mobility.

ReMMoc[14] is a binding framework that uses some of the principles described above in a mobile setting. This thesis operates in the same problem domain. ReMMoc focuses on the different technologies used in different domains. ReMMoc solves this problem by having an external entity (inside the framework, but outside the binding) that can change the type of technology used in the binding, depending on the technology used in the current domain.

Mobiware [25] is a project that researches QoS and Mobility. The project focuses on ATM, not IP. The project also tries to address the problem at the network layer, and not the middleware layer, like this thesis.

However not much effort has been done in making QoS a part of the Remote Binding. Adaptive and reflective operations in a QoS enabled remote binding in a mobile setting needs more research.

2.6 Summary

This chapter presents areas that have been much researched; mobility, Mobile IP, middleware, QoS and remote bindings. Several research projects that dealt with the various research areas were also presented. CCM is a component based middleware solution, openORB provides reflection, wireless ATM and ALICE provides mobility. The main problem however is that none of these projects have combined the field of IP based mobility management and QoS-Awareness in a middleware setting. This thesis focuses its efforts on just that area.

Chapter 3

General Test Configuration

The method in Section 1.4 states that Mobile IP will be tested, and that prototyping is used to validate MobiBind. To do this, a test lab is required. This chapter discusses the lab used for the testing and prototyping. The tools used in the testing of Mobile IP and the prototyping and testing of MobiBind are also discussed here. The actual tests are discussed in Chapter 4 and Chapter 8. This chapter only discusses the general lab setup and the tools used.

3.1 Motivation

The main driving force when setting up the mobility laboratory was the ability to test existing software for existing networks. Hence, the test lab use equipment that is commercially available and the lab is configured to resemble a normal network. A normal network is not specially configured for a specific task, but rather configured to perform well on a general basis.

The reason for this motivation is that the gap between the performance of peak setups, like the one described by Aladdin Saleh [30] and a normal setup in today's Internet or private networks is large.

3.2 Equipment and Network

The lab configuration, illustrated in Figure 3.1, includes LAN and WLAN equipment, Linux and Windows based workstations, a laptop with a WLAN- and LAN-card together with Mobile IP software.

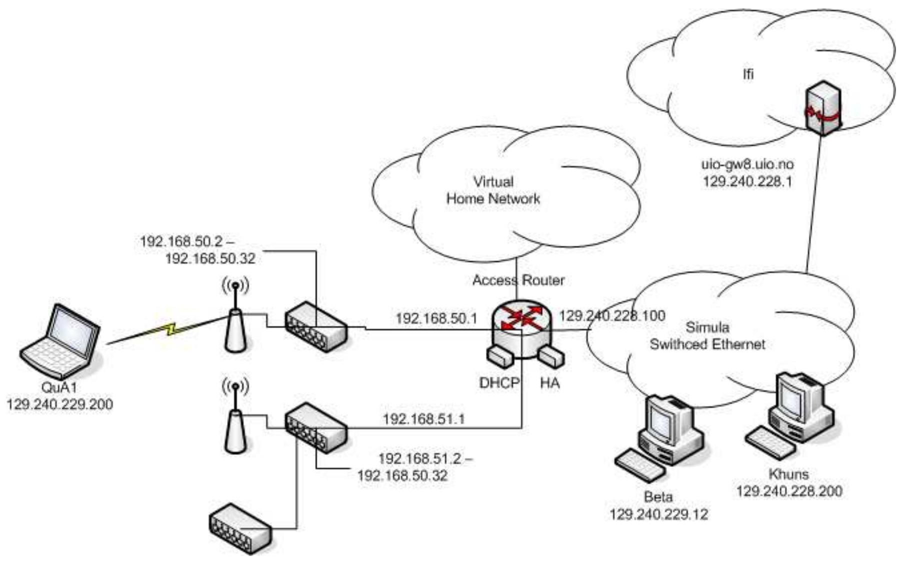

Figure 3.1: Test Configuration

Correspondent Nodes

The correspondent nodes used in the tests were the PCs Beta and Khuns. They were used to send data to the Mobile Node.

Beta

Beta is a Windows computer used in testing throughput with the Windows based tool Iperf.

Khuns

Khuns is the primary test PC, running the Linux operating system and the server side of the KGen traffic generator, see Section 3.5 for more details on KGen.

Mobile Node

QuA1 was the laptop used as Mobile Node. It is running Windows and has both an 10 mbit/s Ethernet card and a 802.11b WLAN card installed. The Mobile Node was also running Birdstep's Mobile IP client software.

Access Router

The Access Router is a Cisco 3200 Mobile IP capable router. This is the router that all traffic passes through. Cisco's Home Agent software operates on the Access Router.

Home Network

The Home network was set up as a virtual home network, so during the tests the Mobile Node was never in its home network, but always in a foreign network.

Network

The networks Simula Switched Ethernet and Ifi, are shared networks connected to the Internet. The network on the other side of the access router, were the Mobile Node operates, was dedicated only to the Mobility laboratory. The Internet connectivity provided some background traffic, making the networks even more realistic.

3.3 Network Transfer Rates

WLAN speed

The 802.11b WLAN has a max transfer rate of 11 mbit/s. The protocol is designed to auto adjust its transfer speed from 11 mbit/s to 2 mbit/s depending on network conditions. This feature was disabled and the speed fixed to 11 mbit/s. The reason for this is discussed in Section 3.6.

LAN speed

The LAN between the Access Router and QuA1 laptop has a speed of 10 mbit/s. The LAN behind the Access Router to the corresponding nodes has a speed of 100 mbit/s. This assures that any bottlenecks would be between the Access Router and the QuA1 laptop.

3.4 Mobile IP Implementation

The Mobile IP software used in this thesis was from Birdstep Technologies and Cisco. Birdstep's software was used in the Mobile Node, while Cisco's software was used in the access router.

Birdstep Mobile IP 2.0

Birdstep Mobile IP 2.0 is a client software designed to operate on the mobile node. It was installed on the QuA1 laptop. Information on the product can be found in the user guide [39], the administrators guide [40], the diagnostic guide [37], and the release notes [38]. Birdstep Mobile IP 2.0 runs on Windows, and was installed on the QuA1 laptop.

Cisco Home Agent

The software from Cisco operated in the access router and was used as the Home Agent. Chapter one of the Configuration guide [35] gives a introduction to the software.

3.5 Data Traffic Generation

In order to perform the test, we not only need the test lab and Mobile IP, but also something to test the technology with. Some kind of data needs to be sent during the tests.

To generate data traffic one could use different applications (e.g., Real Player server) together with a packet analyser (e.g,, Ethereal) and then perform an analysis of the packet data to make some statistics. This will give accuracy on the link layer since the packet analyser reads packets from the link card. Delays encountered from the link layer up to the application layer would not be seen. Because this thesis focuses on the upper layers, however, this approach was not chosen.

Another approach is to use an emulator for the data traffic generated by applications. Several software packages are available. Some packages also incorporates a statistical component, making it possible to read the network jitter and other statistics directly from the program, taking away the need for a independent statistical program. An example of such a program is GenSyn [16] [47]. GenSyn is a Java based program used to generate synthetic Internet traffic. Other traffic analysers, like Argus and Multi Router Traffic Grapher (MRTG) are described and compared in [32].

To be able to tailor the experiments and later test the remote binding, a custom traffic generator was developed as a part of this thesis. The traffic generator was named KGen and is a Java, command line based, traffic generator with statistical abilities. The software produces graphs and data records of the transmissions, giving the user the ability to further post-process the data. Having the source code of the traffic generator tool also helped in connecting the QoS-Aware Remote Binding to KGen.

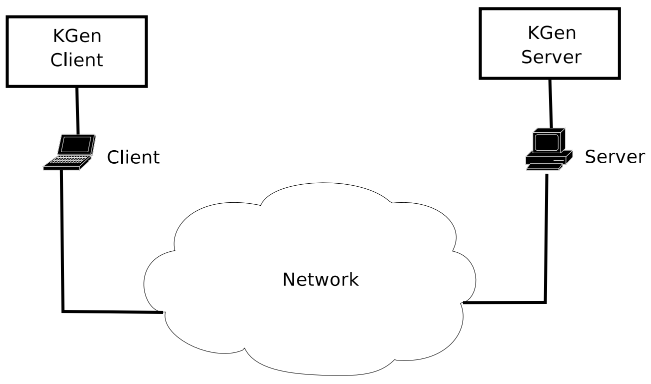

Figure 3.2: KGen

KGen

KGen consists of a client and a server program, illustrated in Figure 3.2. A brief overview of the two programs are given. For a more detailed look, including the source code, at KGen, please go to the enclosed CD and the directory KGen.

The KGen client is a piece of software that receives packets from the KGen Server, and sends a reply back to the KGen server. Each received packet has a serial number, the serial number is reported back to the KGen server in the reply. The reply may be 6 bytes or more.

The KGen server sends packets to the KGen client. The packet contains a serial number together with packet data. The packet may be 6 bytes or more. The server also has a feature that allows the packets to grow in size. It is also possible to set sending rate and packet size. Statistics like round trip time, jitter, and packet loss are produced.

Due to KGen being constructed in Java, KGen comes with some Java specific limitations. The main limitation is that in some scenarios, the granularity of the packet scheduling mechanism is on the order of 20 ms. This limitation is mentioned in several of the competing tools that are Java based. When trying to send packets with a inter packet gap less than 20 ms KGen groups the packets, sending several at once. This gives accurate throughput, but smaller inter packet gap. This limitation does not effect the conclusion of the tests conducted in this thesis. Still it needs to be noted that the granularity of the results has to be seen with this limitation in mind.

Iperf

Another test tool used was Iperf. This tool gives max throughout by pushing as much data as possible through a network connection during a defined time period. Iperf is a part of the NLANR software package [41].

Iperf was used in the bandwidth tests. Iperf has no QoS-Awareness and pushes max data, through a TCP connection, from one side to the other. This test tool can be used to find the actual max data-sending rate for each link.

3.6 Performance Tuning

While setting up the lab, we made several decisions that have an effect on test results. To be able to reproduce the experiments these decisions are discussed here.

Gratuitous ARP

One problem was gratuitous ARP. This is described in [34] and a known problem to the researchers at Birdstep. It is basically a test to see if the given IP belongs to somebody else. The problem occurs when the Mobile Node connects to a new foreign network. This is what happens:

The Mobile Node sends a DHCP request for a new IP address, and a DHCP server in the foreign network replies with a IP address valid for the foreign network. Before the Mobile Node can use the received IP address it has to make sure that the IP address it not used anywhere else. It does this by sending out ARP messages, asking if anyone already has the received address. It does this several times, and waits to see if there is a response. The number of times that is sends out the ARP message depends on the operating system (OS), but between 4 and 10 times with a waiting time of 200 ms between each ARP message is common. This dramatically increases the hand-over time.

A possibility to counter this effect is by reducing the number gratuitous ARP messages, or by disabling it. This can be done in a test setup, but since the normal Internet settings use gratuitous ARP, it was left on.

A solution to this problem proposed by [44] is that the DHCP server performs the gratuitous ARP messaging on its available IP addresses, so that the mobile node does not need to do it.

Noise

Noise from other wireless senders was also experienced in the lab. Since the mobility laboratory is located in a building with other firms that have their own wireless networks, this interference is unavoidable. Furthermore, since interference is inevitable between commercial wireless APs, the noise experienced in the lab makes our tests even more realistic.

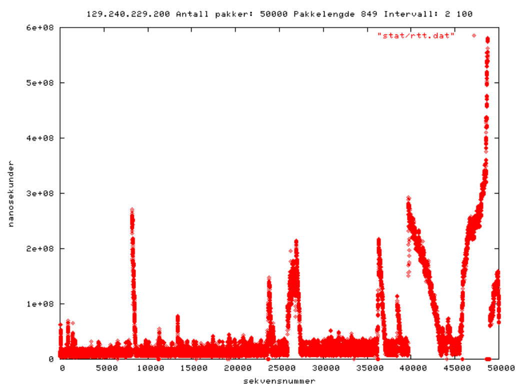

Figure 3.3: Uncontrolled Throughput Test

WLAN speed

Initial tests of the WLAN connections showed that during high bandwidth utilisation the WLAN AP and card got confused, and interpreted the traffic as heavy noise and errors. This caused the WLAN's AP to set different speeds on the connection, from 11 to 6 to 2 mbit/s. This behaviour is caused by the protocol that is designed to adapt to varying network conditions. The protocol did not function well, and was therefore disabled in the mobility laboratory. The speed was set to 11 mbit/s fixed. Figure 3.3 illustrates what happened when the adaptation protocol was in use. In the figure throughput is increased to find a saturation point of the link. But instead of giving a clear indication when maximum throughput is reached, by increasing packet loss, the link gives high packet loss and high RTT from the start of the test (when the throughout is low). Figure 3.4 shows a test where adaptation was disabled. The link has no or low packet loss and RTT up until a point when the RTT and packet load explodes (followed by almost total packet loss). This is a clear indication of the saturation point of the link.

Hard and Soft Hand-over

In the test we always performed hand-overs as soft hand-overs. As noted in the background chapter, soft hand-overs have both old and new links available when performing a hand-over. In theory this makes it possible to perform hand-overs seamlessly.

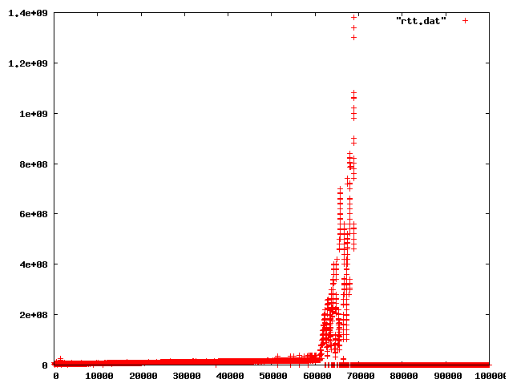

Figure 3.4: Controlled Throughput Test

Foreign Agent

A Foreign Agent was not included in the test configuration. The Mobile Node instead allocates a co-located care-of address, as described in Section 2.2.

Firewall Tunneling

The Mobile Node tunnels all packets through the Home Agent. This eliminates triangular routing.

Other Tweaks

Several other methods for tuning the wireless network exists, e.g., adjusting beacon interval, and request to send threshold suggested in [12]. This would ultimately lead to better hand-over times, but since this thesis attempts to test hand-overs in a normal setting, the attributes were left to default values.

Chapter 4

Performance Test of Mobile IP

This chapter uses the general test bed and utilities described in the previous chapter to test Mobile IP. Testing Mobile IP is the fist step in specifying QoS-Aware Remote Binding requirements and relates to sub-problem one; identifying underperforming QoS parameters. These tests are designed to find out how the basic Mobile IP implementation behaves and if any QoS-parameters in Mobile IP under-perform. There is a particular focus on Streaming and Conversational traffic requirements.

The tests are described in Section 4.1. Results and findings are presented in Section 4.2. The tests performed here are then compared to findings from other researchers. This comparison is performed in Section 4.3. Finally a short summary is presented in Section 4.4.

4.1 Test Description

The Mobile IP test is performed at this early stage because we want to find areas in which a QoS-Aware Remote Binding can enhance performance. Another reason to test Mobile IP now is that the results found in this test, can be compared with the test of MobiBind later.

As described earlier one of the challenges with mobility management is the hand-over between different links in which data are sent. As a result, QoS becomes harder to handle in mobility management. The tests of Mobile IP focus on the relevant QoS parameters for the Conversational and Streaming traffic classes, both of which are defined in Section 2.4.6. The challenges in providing QoS today rests on these two most QoS demanding traffic classes. The basic requirement of mobile connectivity, keeping the nodes connected despite mobility, is solved in previous work. This mobile connectivity is sometimes all that is required for the "lower level" traffic classes.

Figure 4.1 shows the basic test setup: The tests and the QoS parameters that were tested. The first five QoS parameters of the figure

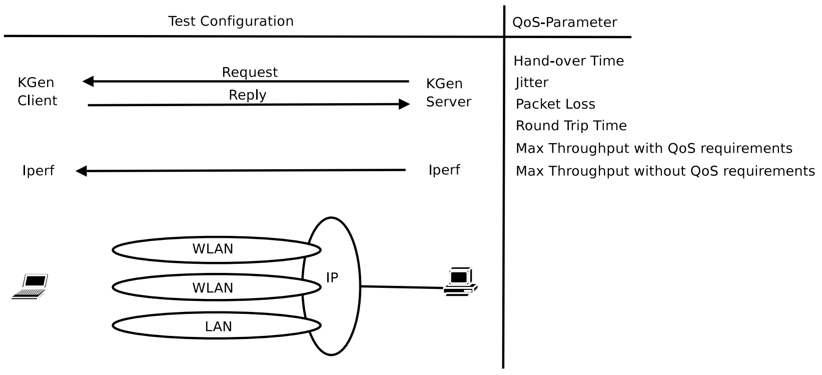

Figure 4.1: Test Setup

were tested using KGen, the last QoS parameter was tested using Iperf. KGen was used to emulate both Conversational and Streaming traffic. To emulate Streaming traffic we used RealPlayer Server data traffic pattern, whose pattern is described in detail by Tianbo Kuang et al. in [20]. The RealPlayer traffic emulated has a throughput of 1609 kbit/s one way(from the server), with only acks travelling the other way (to the server). 1609 kbit/s represents a high quality video stream. The RealPlayer Server encodes one video frame at the time, and sends it to the sender. One video frame is typically too large to be fitted inside one datagram, so several datagrams are created when a video frame is encoded. This encoding scheme produces a bursty traffic behaviour. This bursty behaviour is also described in [20]. The burstyness of RealPlayer was also emulated with KGen.

The Conversational traffic was emulated using a VoIP traffic pattern. One of the differences between Conversational traffic pattern and streaming is that the conversational stream is the same size in both directions. G711 is a voice encoder and decoder that is used in VoIP. G711 produces 64 kbit/s in each direction, giving a total throughput of 128 kbit/s. 128 kbit/s represents a high quality VoIP stream. G711 emulation with KGen was used to emulate the Conversational traffic class.

Two different throughput tests were performed. Iperf was used to test max throughput without any QoS requirements. KGen was used to test max throughput with QoS requirements. The difference between the two max throughput tests (with and without QoS requirements) are described in Section 4.1.4.

Table 4.1 presents the traffic patterns that were used to test the different QoS Parameters.

The tests of Mobile IP were conducted several times on different days. Many hours were spent in the lab; both to perform preliminary investig-

QoS Parameter	Traffic Pattern
Jitter	Conversational
Packet Loss	Conversational and Streaming
Round Trip Time	Conversational and Streaming
Max Throughput	Max offered load
Max Throughput	Max offered load with QoS demands
Hand-over Time	Conversational

Table 4.1: Traffic Patterns for Testing of QoS Parameters

ations of Mobile IP (in order to decide test scope and to design efficient tests) and to perform the actual tests. Testing on different days was especially important in WLAN. WLAN uses a shared medium, air, that in addition to being shared may experience changing atmospheric interference on different days.

Each test ran for 60 seconds, using either 3000 packets for the Conversational traffic class, or 12000 packets for the Streaming traffic class. The throughput tests used special tests describes in Section 4.1.4. In Conversational traffic a packet was sent every 20 ms. The Streaming class sent 4 packets every 20 ms.

4.1.1 Jitter

The first parameter tested was jitter. Many modern applications are jitter sensitive, like Conversational and Streaming traffic. The applications need to have a steady stream of data in order to perform a high quality playback. This is why jitter is an important QoS parameter and should be included in the investigation.

Jitter was tested on both LAN and WLAN, using KGen and the conversational traffic, with a sending rate of 20 ms.

4.1.2 Packet Loss

Another important QoS parameter is packet loss. Packet loss causes TCP streams to reduce their sending rate, regardless of the reason for the packet loss. TCP also resends lost packets, possibly increasing delivery time. When faced with packet loss, UDP streams do not reduce sending rate or resend lost packets. If packets are lost when using UDP, the application simply receives an incomplete data set. Streaming and Conversational traffic can typically manage some gaps in the data set without a noticeable reduction of user quality. But if the packet loss gets too high, and thus the gaps in the data set (when using UDP), user quality will become compromised. It is therefore important to have low packet loss.

Packet loss was tested both for WLAN and LAN using both Conversational and Streaming traffic. Packet loss in hand-overs was also tested.

4.1.3 Round Trip Time

Another important QoS parameter is Round Trip Time (RTT). Streaming traffic does not have very strict RTT requirements, but a large RTT means that the user needs to wait longer before the playback can begin. In order for Conversational traffic to give good quality to the user, the RTT needs to be below 200 ms [30]. 200 ms is the limit of what the user can hear, so the user will not notice any quality change. However, if the RTT is over 200 ms, the user may notice a delay when having a conversation. This delay gets worse the higher the RTT becomes.

RTT is tested for both WLAN and LAN, and for Streaming and Conversational traffic.

4.1.4 Max Throughput

The max throughput of the link limits the number of streams a link can support.

During the preliminary testing we found that when a link had maximum traffic load, QoS for Streaming and Conversational traffic was violated. Max throughput was therefore tested in two different ways: One test, performed with Iperf, tested the max throughput without paying attention to any QoS requirements. Another tested max throughput while paying attention to QoS requirements. The difference between the two tests is that max throughput without any QoS attention only tries to push as much data as possible through the link. The max throughput test with QoS attention tries to push as much data as possible, while not compromising QoS. In this test QoS was specified as meaning a high RTT and packet loss.

Max throughput with QoS was performed with KGen. A Special incremental feature built into KGen was used. KGen incrementally increases the kbit/s while monitoring packet loss and RTT. When the kbit/s reached a certain point, the RTT and packet loss exponentially increased, reaching a defined threshold. The kbit/s that KGen was sending at the time the threshold was reached, was said to be the max throughput with QoS. The tests ran several times until a reliable throughput could be found.

Max throughput without QoS was performed by sending as many packets as possible over the connection. KGen was not suitable for this, so Iperf was used for this test. Iperf does not take into consideration QoS

Handover Scenario	From	To
1	WLAN	WLAN
2	LAN	WLAN
3	WLAN	LAN

Table 4.2: Hand-over Scenarios Tested

and only tries to maximise the throughput, regardless of RTT or packet loss.

Both tests were ran on WLAN and LAN.

4.1.5 Hand-over Time

Seamlessness is an important QoS parameter in Conversational and Streaming traffic. Any connection breakage would result in bad service to the user. A video might stop playing, and similar quality degeneration would be noticed in a conversation.

Hand-over is a place that the data flow would potentially be broken. When the node switches networks it might take some time before the new link (with new route to the Mobile Node) is operational.

The last parameter tested was therefore hand-over time.

This test measures how long the connection between a correspondent node and Mobile Node is down when a hand-over is performed.

The tests were performed on hand-overs from and to both LAN and WLAN. Table 4.2 displays the three different hand-over scenarios that were tested.

4.2 Test Results and Findings

This section presents key results and findings of the Mobile IP tests described above. Selected results based on the complete test set are presented here. For complete test results refer to Appendix A. The results and findings presented here focus on answering sub-problem one, but also shed light on findings that can help to specify requirements for the design of a QoS-Aware Remote Binding.

4.2.1 Jitter

The jitter test results, presented in table 4.3, show that LAN had significantly lower jitter than WLAN. The LAN results also had a lower standard deviation, meaning more stable numbers.

	Max Jitter (ms)	Average Jitter (ms)	Standard Deviation (ms)
WLAN	105	0,619	0,076
LAN	13,109	0,1	0,011

Table 4.3: Jitter Test Results

The jitter was relatively small, with average jitter below 1 ms. Max jitter on WLAN is probably the only value that is high enough to cause quality degeneration.

4.2.2 Packet Loss

	Max Packet Loss (%)	Average Packet Loss (%)	Standard Deviation (packets)
WLAN	1,3 x 10^{-2}	1,1 x 10^{-3}	26,66
LAN	3,3 x 10^{-4}	2,2 x 10^{-6}	0,13

Table 4.4: Packet Loss Test Results

The packet loss was close to zero in LAN. WLAN had an erratic pattern that can be partly explained in the wireless medium, its standard deviation were also relatively high. LAN packet loss was so low that it was difficult to measure. The test showed a big difference between packet loss in LAN and WLAN.

Packet loss was also measured during hand-over. 61 packets were lost when going from WLAN to WLAN. When going from LAN to WLAN the packet loss was 49. The packet loss was also 49 when going from WLAN to LAN.

4.2.3 Round Trip Time

	Conversation (ms)	StDev (ms)	Streaming (ms)	StDev (ms)
WLAN	4,06	0,41	10,33	1.58
LAN	1,26	0,03	4,09	0.03

Table 4.5: RTT Test Results

Table 4.5 shows that WLAN uses more time than LAN to send the same amount of data back and forth. In fact, WLAN uses about 400% more time in the Conversation traffic class, and about 200 % more time in the Streaming traffic class. This could be because a Twisted Pair (TP)

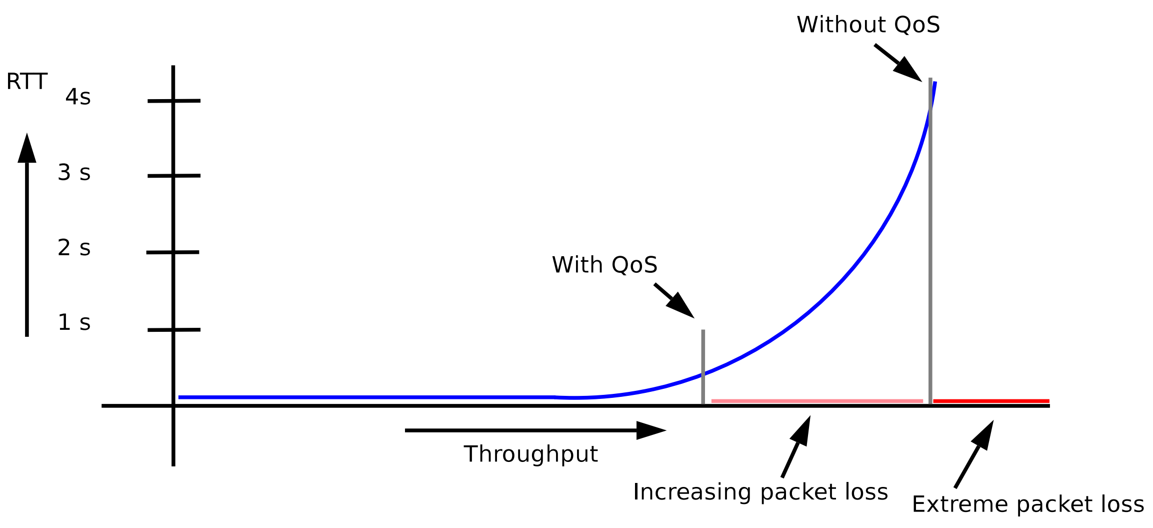

Figure 4.2: WLAN Max Throughput Test

cable is faster than air, or it might be because Ethernet tries to send packets and sees if it is successful, while WLAN checks to see if the link is available before sending.

It is also only natural that there is a difference in RTT between Conversational and Streaming traffic. Streaming traffic has larger and a higher number of packets than Conversational traffic.

In WLAN it was noticed that the RTT increased as the condition of the link worsened.

4.2.4 Max Throughput

	Throughput QoS(kbit/s)	Throughput no QoS(kbit/s)	StDev (kbit/s)
WLAN	3768	5488	51
LAN	8791	8811	36

Table 4.6: Max Throughput Test Results

The results of the max throughput test, with and without QoS, is displayed in Table 4.6.

Despite the fact that WLAN has a higher theoretical bandwidth (11 mbit/s) than LAN (10 mbit/s), both tests showed that LAN outperformed WLAN.

In WLAN there was a big difference between how much throughput the link could have with and without QoS. Figure 4.2 illustrates what happens between QoS max throughput and no QoS max throughput. After the max throughput with QoS has been reached, the link's RTT and packet loss starts to increase. At the max throughput without QoS, the link is experiencing high RTT and high packet loss. The difference between QoS and no QoS max throughout is 1720 kbit/s, or about 31 %. So for real-time sensitive applications or traffic classes, a WLAN link should not have a higher throughput than 3768 kbit/s. Used for Background traffic, however, the WLAN link could have a throughput of 5488 kbit/s.

Even more interesting was the difference between QoS and non QoS max throughout on LAN. The throughput test found that there was almost no difference in throughout with or without QoS requirements in LAN. In WLAN, however, there was a significant decrease in throughput when QoS requirements had to be met. This shows a clear difference between LAN and WLAN.

4.2.5 Hand-over Time

From -> To	Hand-over time (ms)	StDev (ms)
LAN -> WLAN	976	18
WLAN -> LAN	976	8
WLAN -> WLAN	1229	31

Table 4.7: Hand-over Time Test Results

Table 4.7 presents the hand-over times found through testing. The hand-over times were found to be very stable in the technologies we tested. A hand-over from or to LAN takes 976 ms. A hand-over from WLAN to WLAN takes a little longer; 1229 ms. Seamlessness is lost for about one second during each hand-over. The measured hand-over time suggests that Conversational traffic, particularly VoIP, might be difficult to support during hand-overs, due to the limit of 200 ms with regards to user perceived delay.

4.3 Analysis and Comparison

This sections further discusses the test results and findings. Test results and findings from other researchers are also discussed and compared to this thesis results. Some areas that should be addressed in the requirement specification are also highlighted.

The test of Mobile IP showed that jitter was low when using Mobile IP. Only in a few circumstances was jitter a problem. Therefore, jitter handling should probably not be the main focus when we specify the QoS-Aware Remote Binding requirements.

[19] also presents results of a WLAN test lab. They found that jitter does not become a problem until the link becomes very weak. The jitter results from this thesis test was also stable, with a low standard deviation. So jitter should not be view as a problem for the rest of this thesis.

Packet loss was also tested. The tests showed that LAN was very stable when it came to packet loss; almost no packets were lost. WLAN had a much higher packet loss rate, and a much higher standard deviation.

Viewing only this thesis packet loss test we can conclude that there is a big difference between LAN and WLAN when it comes to packet loss. WLAN loses a lot of packets, while LAN is very stable and hardly no packets are lost.

Packet loss for hand-overs was also very high. Depending on the technologies involved, either 59 or 63 consecutive packets were lost. This is a very high number, that spans from mobility management, and the requirement specification of a QoS-Aware Remote Binding should address the problem of high packet loss.

The RTT measured was lower for Conversational traffic than Streaming traffic. This was to be expected, since more data was transported in the streaming tests. It is more interesting to note that the RTT for the same traffic is greatly different in LAN and WLAN. LAN has a lower RTT than WLAN in both test scenarios. This is a big difference that can be used if one needs to decide, from looking at the RTT, which type of link the remote binding is currently using.

Max throughout was measured with, and without QoS requirements. These tests revealed a difference between LAN and WLAN. While LAN had almost the same throughput with and without RTT QoS requirements, WLAN showed a big difference. About 8,8 mbit/s could be sent through the LAN link of a theoretical 10 mbit/s, regardless of having QoS requirements or not. The WLAN link, despite a theoretical bandwidth of 11 kbit/s, could only send 3,7 mbit/s through the link for QoS sensitive traffic and 5.5 mbit/s for non QoS traffic. When trying to send more data than 3,7 mbit/s on the wireless link, the link responded with higher RTT and higher packet loss.

The hand-over time measured in the tests was very stable, taking about one second. This is an unacceptably long time for both streaming and conversational traffic. Hand-over times was also measured by [29]. They also found that the hand-over time took about one second. The hand-over time should therefore be addressed in the requirement specification of the QoS-Aware Remote Binding.

4.4 Test Summary

The findings and observations discussed in this chapter show that there is a big differences between how LAN and WLAN links behave, creating a need for an adaptive remote binding. The tests also show that hand-overs cause both packet loss and connection breakage. Results from this chapter will be used in the requirement specification of the QoS-Aware Remote Binding in the following chapter.

Chapter 5

QoS-Aware Remote Binding Requirements

To answer the second sub-problem, that relates to the requirements of a QoS-Aware Remote Binding. The tests of Mobile IP serve as a foundation. This thesis takes the notion of QoS into the remote binding requirements. The requirements are divided into two categories, first we have the general requirements, that are requirements for a remote binding, secondly the QoS remote binding requirements in the mobile domain is discussed and specified. The tests of Mobile IP from Section 4 are used as a foundation when specifying the requirements in this chapter, and especially in Section 5.2.

The requirements contain the words must and should. The words are defined as follows:

Must or the adjective "required", means that the item is an absolute requirement of the specification.

Should or the adjective "recommended", means that, in some circumstances, valid reasons may exist to ignore them, but the full implications must be understood and carefully weighed before choosing a different course.

5.1 General Requirements

The general requirements of the remote binding are the requirements that enable it to operate in a mobile setting. There are four general requirements.

5.1.1 Mobile Connectivity

A local or remote binding in component bases middleware enables connectivity between components. The QoS-Aware Remote Binding should do this, but also extend this connectivity to cover a Mobile Node, i.e. one of the components exists in an address space in a computer that is mobile.

REQUIREMENT The remote binding must connect two mobile components together.

5.1.2 Mobility Management

To enable the Mobile Node to change APs. i.e., hand-over LAN to WLAN and WLAN to WLAN, the binding must encapsulate a Mobile IP client.

REQUIREMENT The remote binding must implement Mobile IP for mobility management.

5.1.3 Two-Way Connectivity

The binding should support generic data traffic, and traffic might be two-way: Telephony is an example of such an application. To be able to handle applications like voice and other bidirectional applications, the binding should be bi-directional.

REQUIREMENT The remote binding should provide a two-way connection.

5.1.4 Explicit Remote Binding

The remote binding needs to be adaptable, and the design of an explicit remote binding (discussed in Section 2.5) provides this, in the form of connecting and controlling interfaces.

REQUIREMENT The remote binding should be designed as an explicit remote binding.

5.2 QoS Requirements

The general requirements specifies a remote binding that has the ability to function in a mobile domain. From our test of the QoS parameters, we concluded, that hand-over and link changes have a major impact on QoS. Therefore, the QoS-Aware Remote Binding must be designed to handle and share information about hand-over and link changes.

5.2.1 QoS Monitoring

The link change comes as a consequence of the hand-overs. In mobile computing, the link may change drastically after each hand-over.

When the link changes, the QoS characteristics change. It is important that the remote binding is aware of this and in some way uses this information. The information can also be useful for the using components in their effort to provide the best service possible.

The remote binding should therefore provide QoS monitoring of key QoS parameters. This network QoS information should be available both to be used in the binding for adaption, and to the using components.

The QoS monitoring needs to be aware of the fact of the link change that occurs after a hand-over, and should strive to provide as accurate information as possible after a hand-over.

REQUIREMENT The remote binding must provide QoS monitoring.

5.2.2 Hand-over Aware

Special to the mobile domain is the hand-overs. It is important that the remote binding is aware of this and can understand the implications that a hand-over yields.

As stated earlier the one of the main operational differences for a remote binding in a mobile domain is hand-over. It is therefore natural that this area is were the remote binding needs to focus on.

As stated in Chapter 4, the Mobile IP tests showed that QoS characteristics like high packet loss and loss of connectivity was what happened when a hand-over occurred. These problems are addressed in the following requirements.

Packet Loss

As discussed in the Mobile IP tests packet loss, especially in hand-overs, is high. It is therefore important that the remote binding minimises or eliminates packet loss during hand-over.

REQUIREMENT The remote binding must minimise packet loss during hand-over.

Data-flow

The second difference that the Mobile IP tests showed, were that when a hand-over occurred the link went down for about one second. It is important for applications to have a seamless data-flow. Therefore, the binding should provide this.

REQUIREMENT The remote binding should maintain data flow during hand-over.

Monitoring

The using components are aware of the explicit remote binding and might also be aware that the link is a mobile link. The components might want information about the hand-over. The main attribute for a hand-over is the hand-over time. There might also be interesting to know when a hand-over is occurring. This information should be available to the components.

REQUIREMENT The remote binding should provide hand-over monitoring.

5.3 Summary of Requirements

The requirements that were discussed and specified in this chapter aims at providing the best possible operation for components in a mobile environment. The following requirements were specified:

1. The remote binding must connect two mobile components together.
2. The remote binding must implement Mobile IP for mobility management.
3. The remote binding should provide a two-way connection.
4. The remote binding should be designed as an Explicit remote binding.
5. The remote binding must provide QoS monitoring.
6. The remote binding must minimise packet loss during hand-over.
7. The remote binding should maintain data flow during hand-over.
8. The remote binding should provide hand-over monitoring.

Chapter 6

QoS-Aware Remote Binding Design: MobiBind

This thesis not only tests Mobile IP and specifies requirements for a QoS-Aware Remote Bindings in Mobility Management; it also designs and implements a specific QoS-Aware Remote Binding. The designed and implemented QoS-Aware Remote binding for mobility management is named MobiBind. MobiBind addresses the issues from both the Mobile IP tests and the requirement chapter.

This chapter starts with a discussion on the abstraction level for MobiBind in Section 6.1. This discussion refers to the OSI model, open and explicit bindings, binding factories and nested bindings.

After the abstraction level of MobiBind has been discussed, Section 6.2 map the requirements from Chapter 5 to the functionality of MobiBind. This section also discusses the main reasons for the chosen functionality.

The design of MobiBind is then presented; in Section 6.3 the structural design and in Section 6.4, the functional design. As noted in the method the design and implementation of MobiBind was an agile and iterative process. The end design is presented in this chapter together with interesting design decisions. Any references to testing in this chapter refer to the agile and iterative process.

At the end of this chapter example of use is presented. The examples illustrate typical uses of MobiBind.

6.1 Abstraction Level

The first design decision that requires our attention is the abstraction level of MobiBind. The abstraction level efficiently sets the scope of the explicit binding.

The abstraction level of the binding can be looked upon as deciding at which layer of the OSI model the binding should be designed at. There are two main ways of doing this. A remote binding can be designed either at a low level, or at a high level. The low level would cause the explicit binding to get responsibilities of the network layer. A higher level of abstraction would increase the explicit binding's responsibilities, towards the application's responsibilities.

6.1.1 Low level

If an explicit binding is implemented at the network layer the remote binding would typically have communication interfaces that support packets. The remote binding would just receive and deliver packets to the binding clients (the components that are connected to the remote binding through the communication interfaces). This packet based communication enables the remote binding to focus on delivering the packets, and operations related to this. Compression and decompression is then typically something that is done outside the explicit remote binding, maybe by the client and server applications. This makes the remote binding less complex, and adds complexity to the client/server. The compression and decompression is something that needs to be done between the client and server (sending a raw stream through the network is simply a waste of resources). So the question is where to place the complexity of compression and decompression, not whether to implement it or not.

At this low layer of abstraction the remote binding would have no knowledge of the application specific stream that the packets are made from. This makes the remote binding unable to know which, if any, packets are important and should be given priority.

The remote binding cannot manipulate the packet stream at this low level, but changing network conditions could easily require change in the stream in some way. Since the stream is compressed and decompressed from a stream to packets outside the explicit remote binding, information about the changes in the network should be available to components that are responsible for converting the stream to packets, and probably also the component that are producing the stream.

At this low level the explicit remote binding would concentrate adaption at the lower layers, i.e., providing best possible network QoS to the binding components.

6.1.2 High Level

Designing a explicit remote binding at a high level would enable the explicit remote binding to receive and deliver application specific streams, not “generic” packets. At this high level the explicit remote binding

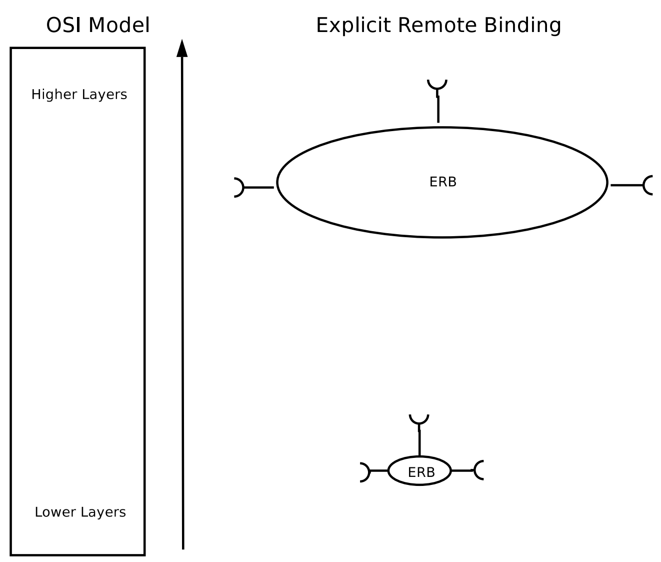

Figure 6.1: Explicit Remote Bindings and the OSI Model

knows what type of stream it is currently managing and could make decisions based on both the stream type, and the network conditions. When the explicit remote binding is faced with low throughput, it could apply a stream specific compression algorithm that would require less throughput from the network.

The explicit remote binding would at this high level know the stream it is presented with. The more knowledge the remote binding would get about the stream, the better would the decision about compression be. The explicit remote binding would require to hold a large library of compression algorithms in order to manage different network conditions and different streams. This would increase the size of the binding. Added complexity to choose appropriate compression algorithms would also have to be included in the explicit remote binding.

Figure 6.1 illustrates the added complexity problem. Explicit remote bindings are "black boxes" and can not be opened and manipulated. So in order to handle increased responsibilities the explicit remote binding would have to grow in size, e.g., by containing many different compression algorithms.

6.1.3 Discussion

There are two arguments that can be applied when choosing abstraction level for MobiBind. First, the end-to-end argument by Saltzer et al. [31]. Secondly the "one shoe does not fit all" argument.

This thesis aims at enhancing QoS performance. The problem statement does not state any specific traffic or application type that the remote binding should be created for. If the explicit remote binding is implemented at a high level, support for "all" types of streams should be included. The binding would then grow to be very large. The well known argument of "one shoe does not fit all" would be applicable to that scenario: A large remote binding would not be able to run on a PDA because of memory limitations.

The end-to-end argument by Saltzer states that if you have the choice of implementing functionality on a high or low software level, the higher level should always be chosen. The only exception from the rule is that if some performance gain is given by implementing the functionality at the lower layer.

The lower software layer is in this discussion the explicit remote binding, and the higher layer is the client and server that are producing and consuming the stream. The functionality is the compression and decompression. Placing the functionality, i.e., the compression, in the binding would be non coherent to the end-to-end argument. So according to the end-to-end argument a performance gain should be the only reason for placing the compression inside the explicit remote binding. The performance gained by this, as described above, is the added value of managing the stream inside the explicit remote binding. The binding sees both the stream and the network and therefore make decisions based on both sources of information. This management could, however, easily be done outside the explicit remote binding if the binding components could access information about the network. So to conform with the end-to-end argument the compression should be placed outside the explicit remote binding. The components performing the compression should instead access the link aware information, as stated in the requirements, to adapt the stream.

Another way to view the explicit remote binding is inside a nested open binding component. A open component is a way of building up the OSI layers. The open design enables the components to change the inside components. The open binding would then not need to contain all possible compression algorithms, but the compression algorithm could be changed depending on the type of stream.

The low level explicit remote binding could then be used as a base, when constructing a higher level nested open remote binding. This principle is illustrated in Figure 6.2.

A way of creating the open binding is by a binding factory. The binding factory is supplied with some parameters, in this case it could be stream type and some notion of what quality the user appreciates. The binding factory would then take the explicit remote binding, and connect appropriate compression and decompression components together

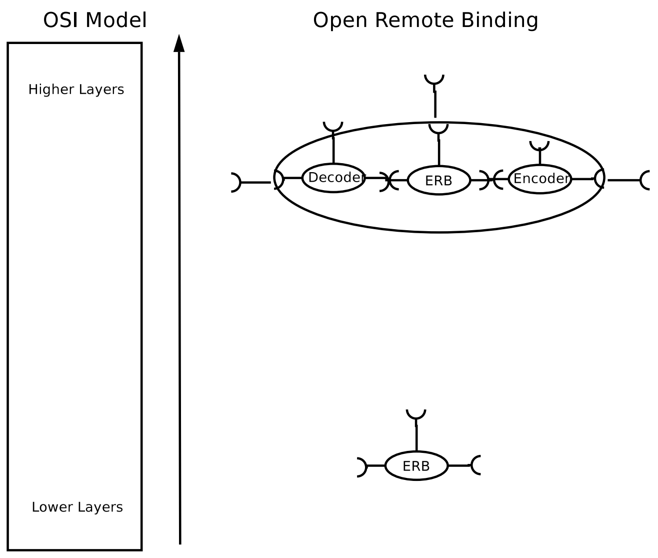

Figure 6.2: Open Remote Bindings and the OSI Model

with other needed components to create a stream and application specific open binding. It is important to note that a high level nested open binding would only be required to contain one pair of compression and decompression algorithms. The explicit remote binding at the same level would need to contain all possible compression and decompression algorithms. So an open binding at a high level would be smaller in size than an explicit remote binding at the same level.

The problem statement does not specify a specific traffic type or application; only enhance QoS, so MobiBind should be designed as a low level explicit binding, that can be used by a binding factory to create open nested bindings. An important feature would be to enable the binding components to see relevant information that will enable them to perform high level adaptation.

6.2 Mapping Requirements to Functionality

Table 6.1 lists the requirements from Chapter 5. The requirements are mapped down to functionality and sections where they are discussed and designed.

The three main functions of MobiBind is the Harvest mechanism, hand-over detection algorithm and link-awareness. The Harvest mechanism's main goal is to provide seamless connectivity during hand-overs. The harvest mechanism also plays a part in minimising packet loss. Another important function of MobiBind is the hand-over detection al-

Requirements	**Functionality / Section**
The remote binding must connect two mobile components together.	Basic Remote Binding Design 6.3.1
The remote binding should provide a two-way connection.	Inner Binding Structure 6.3.2
The remote binding must implement Mobile IP for mobility management.	Basic Remote Binding Design 6.3.1
The remote binding should be designed as an Explicit remote binding.	Basic Remote Binding Design 6.3.1
The remote binding must provide QoS monitoring.	Link Awareness 6.4.3
The remote binding must minimise packet loss during hand-over.	Hand-over Detection Algorithm 6.4.1& Harvest Mechanism 6.4.2
The remote binding should maintain data flow during hand-overs.	Harvest Mechanism 6.4.2
The remote binding should provide hand-over monitoring.	Hand-over Detection Algorithm 6.4.1

Table 6.1: Mapping Requirements to Functionality

gorithm. This algorithm enables MobiBind to provide hand-over monitoring and a efficient hand-over detection algorithm helps prevent packet loss. The link-awareness functionality enables QoS-Monitoring, and also helps the remote binding to provide QoS information to the binding controlling components (the components that are using the controlling interface of MobiBind).

6.3 Structural Design

This section presents the structure of MobiBind. The structure of MobiBind is given before the functional design discussion to better relate the functional discussion to the actual structure of MobiBind.

Section 6.3.1 discusses the basic explicit remote binding design, then a look inside MobiBind is provided in Section 6.3.2. A section about Network communication is then presented in Section 6.3.4. Lastly, the state machine of MobiBind is presented.

The structural elements of MobiBind will be used when the functional design is presented, in Section 6.4.

6.3.1 Basic Remote Binding Design

Requirement four states that the QoS-Aware Remote Binding should be designed as an explicit binding. The scope of the explicit binding is already discussed, but the QoS-Aware Remote binding is also distributed, i.e., located in both address spaces, so design decisions have to be made her as well.

Requirement one and two states that the remote binding should use Mobile IP and connect two mobile components together. So a Mobile IP implementation should be included in the design of MobiBind. This Mobile IP implementation should lie in the middle of the remote binding to provide mobile network connectivity.

There are several possibilities regarding what functionality should be included at each side of Mobile IP in MobiBind. The majority of the functionality can be implemented on one side of Mobile IP, or a more balanced design could be chosen, with equal functionality on each side of the Mobile IP implementation.

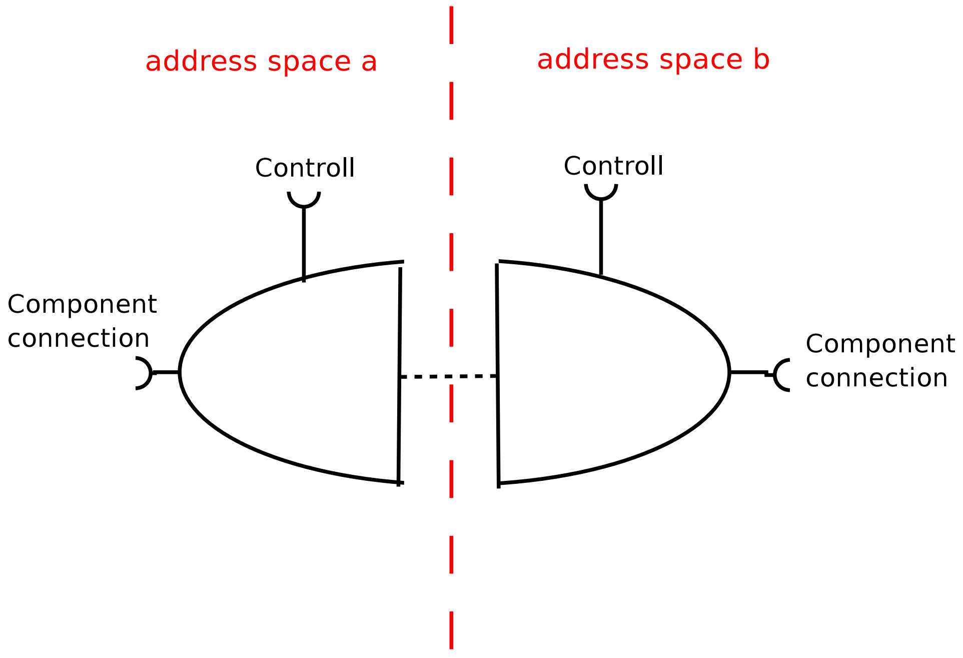

Figure 6.3: Basic MobiBind Design

If a binding were to be designed with major functionality on one side, signalling would be required to notify the less functional side of any decisions. This might be difficult in Mobility Management, since communication is down during hand-over. It is impossible for one side of the remote binding to notify the other side that a hand-over is occurring, when the network link is down. Because of this MobiBind is implemented with equal functionality on both sides of the Mobile IP implementation. This is illustrated in Figure 6.3. The two sides should also be able to both monitor and make independent decisions. Both sides of MobiBind has its own state, more on this in Section 6.3.5.

Figure 6.3 also has a controlling interface for both the sides of MobiBind. This means that MobiBind manages the controlling interfaces itself, and not externally as discussed as an possibility in Section 2.5.3. This comes as a natural decision when MobiBind has two independent sides. Both sides are equal, and there is no reason why one side should have a controlling interface, and the other not.

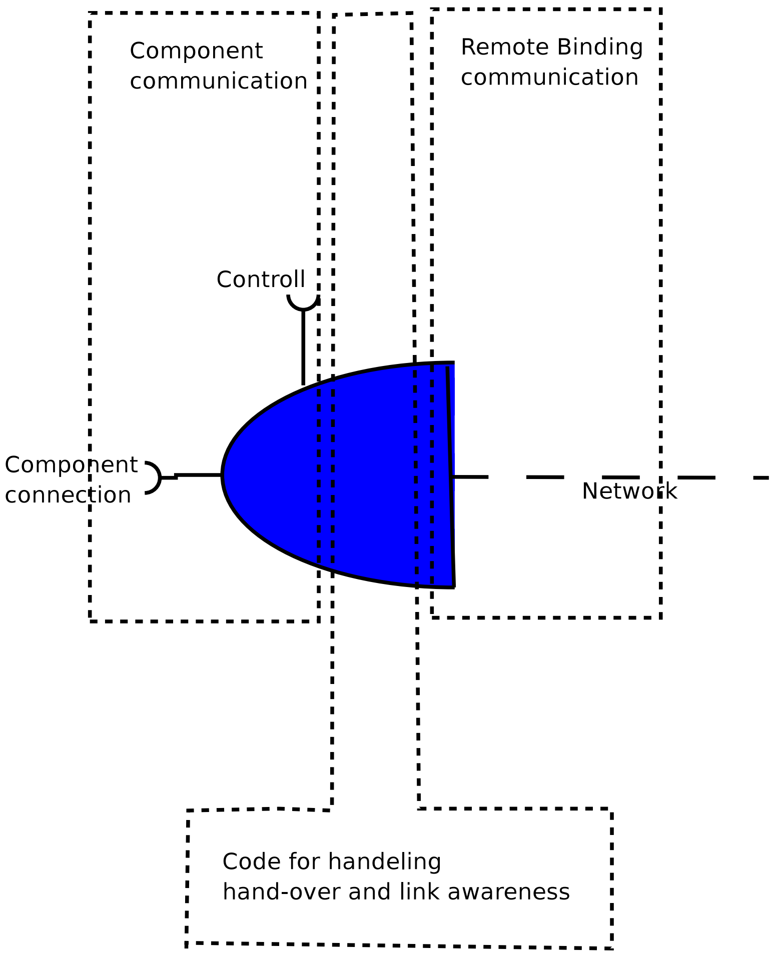
Component communication
Remote Binding communication
Controll
Component connection
Network
Code for handeling hand-over and link awareness

Figure 6.4: Logical View of MobiBind

6.3.2 Inner Binding Structure

This section describes the objects within MobiBind and how they are connected.

The inner structure of MobiBind allows for fulfilment of requirement three, two-way connection. The structure also makes it possible to implement the other functionality and assign responsibilities to the inside objects of MobiBind, to fulfil the remaining QoS-Aware Remote Binding requirements.

As stated earlier the remote binding is made of two independent and equal sides. Figure 6.4 shows one of these sides. First, the side needs to communicate with the connected component, secondly, it needs to support link and hand-over awareness and finally it needs to communicate with the other part through the network.

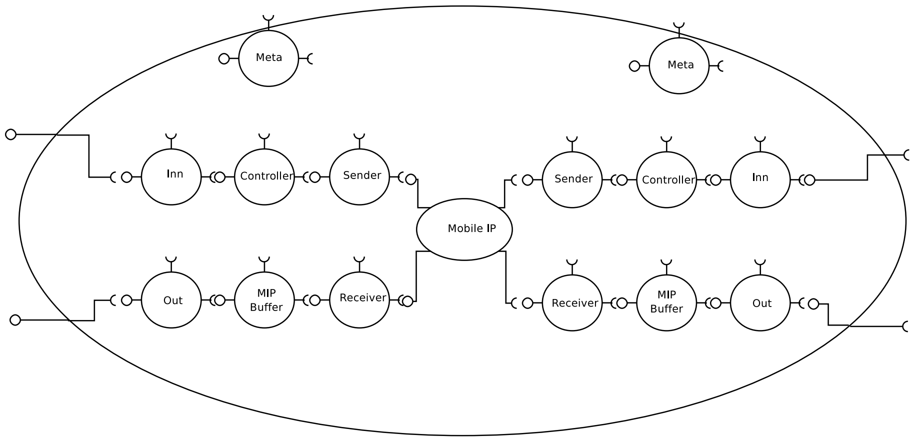

Figure 6.5: Inside MobiBind

Figure 6.5 shows both sides of MobiBind. In the middle lies Mobile IP, that is responsible for sending the UDP packets to and from the two sides, more on network communication in Section 6.3.4. The controlling interface goes through the *Meta* object. The design conforms to the logical design of the sides, described above. The connections shown in this figure are only the main relationships, other connections exist that are not shown in this figure. Here is a list of the objects different responsibilities:

Inn The connection point for the binding component. Used by the binding components to send packets to the receiving component. Takes the binding component's data packet and makes a MobiBind packet of it.

Controller Is responsible for link awareness and also contains a "sender side" buffer. The *Controller* also plays a part in fast send.

Sender Sender communicates with Mobile IP and sends MobiBind packets through Mobile IP. It also encapsulates the MobiBind packets inside a UDP packet. The *Sender* also plays a part in hand-over detection.

Receiver Receives packets from Mobile IP and de-capsulates the MobiBind packets. Depending on the type of packet, it sends the packet to either the *Controller* or *Mobile IP (MIP) buffer*. Notifies the *Controller* that an ack has arrived. The *Receiver* is also responsible for hand-over detection.

MIP Buffer Intelligent buffer to be used during hand-over, and adaptation to future hand-overs. This buffer plays a key role in the harvest mechanism.

Out Removes the data-packet from the MobiBind packet and delivers it to the receiving component.

Meta Holds key information like link awareness information and hand-over information. May also be used to alter the remote binding's behaviour (use *MIP buffer* or not) and to access link aware information. Responsible for starting the entire component. Manages state changes.

Figure 6.6 shows the class diagram for the Remote Binding. The figure shows important classes and methods that implement the functionality of MobiBind.

6.3.3 Creation, Initialisation, and Destruction

Creation and Initialisation

The creation and initialisation of MobiBind is done via a call to it. *Meta* then creates and initialises MobiBind based on the parameters given to *Meta*. *Meta* then provides a way for the binding component to connect to the communication interfaces.

Destruction

The destruction of MobiBind is done, via the controlling interface, to *Meta*. *Meta* purges the buffers and deallocates the resources used by the binding.

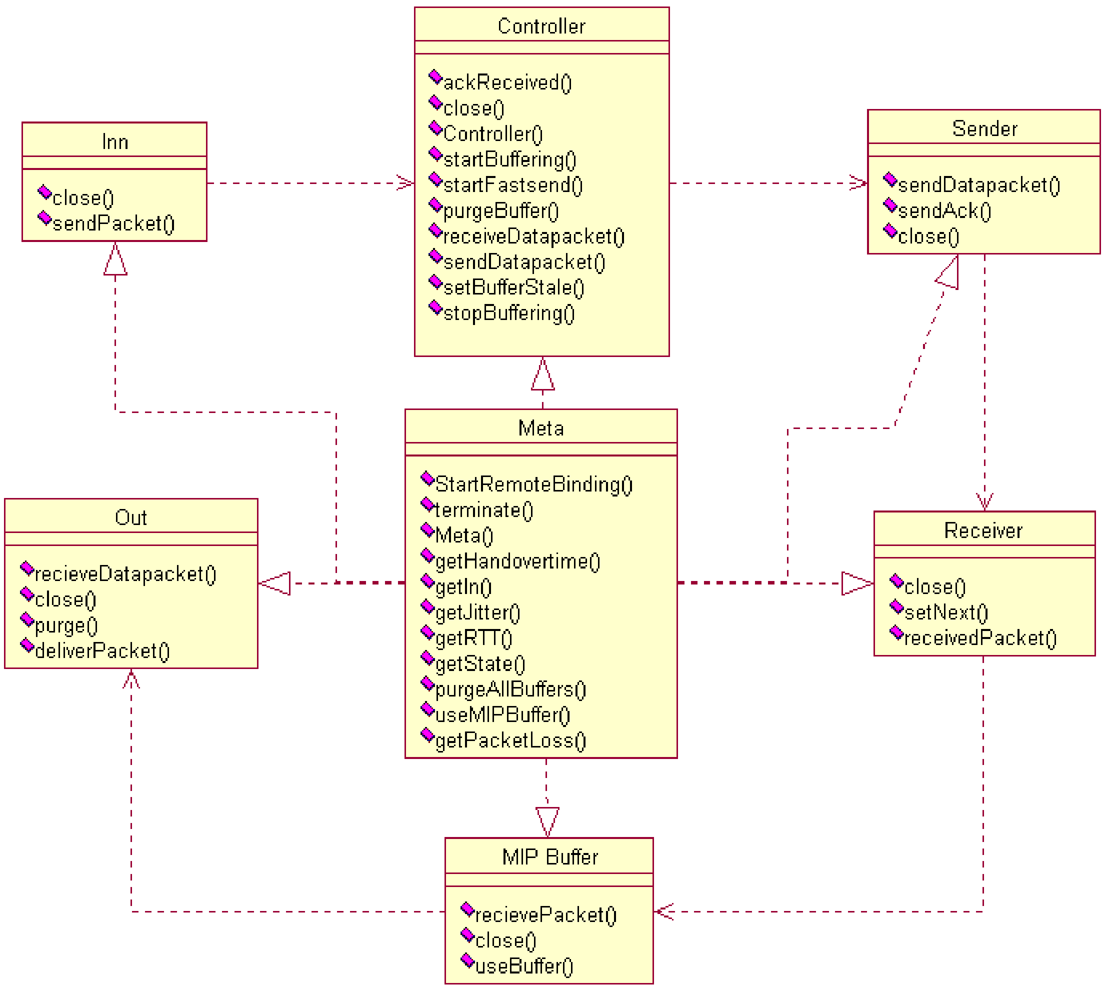

Figure 6.6: MobiBind Classes

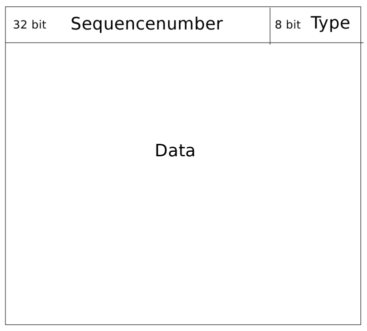

Figure 6.7: MobiBind Packet Format

6.3.4 Network Communication

MobiBind is implemented as two equal sides. All that connects them together is the communication through Mobile IP. A way for MobiBind to communicate through this connection is therefore needed. This section presents the packet format used by MobiBind.

In MobiBind the communication is sent over UDP, the simplest transportation protocol available in the IP stack. UDP was chosen over TCP because TCP adds features, like re-sending, that adds complexity and delay. By choosing UDP we can chose the added complexity, and avoid any complexity overhead. UDP also allow for lossy transmission. MobiBind defines its own packet format that is sent through this UDP connection. The packet format is designed to be very data efficient, meaning that the packet format should generate minimum overhead. The packet-format is shown in Figure 6.7. The packet contains the following fields:

- Sequence number
- Type
- Data

Sequence number

The Sequence number is used for keeping track of the packets and enabling acks. The acks are the basis for link awareness.

Type

MobiBind can send several packet types, so a type field is required to define the packet content. The four types are:

- Data
- Harvest
- Ack
- Signal

Both data and harvest is used to send data (that is generated by the binding components), where harvest is a special mark used during fasts end and in the harvest mechanism. Ack is used by the *Receiver* to notify the *Sender* that a data or harvest packet was received. Signal is used for signalling between the two sides of the component.

Signals include ping and pong used in hand-over detection and a purge signal, which job is to empty buffers, typically at the end of a transmission.

Data

The data field of the packet is where the actual data is stored. In a data-type-packet this will be the data received by the binding components.

6.3.5 State Machine

The state diagram of MobiBind is shown in figure 6.8. A brief overview of the different states is given below. The state machine is used by the entire binding, but especially by the harvest mechanism, which is described in Section 6.4.2.

Transmitting Normal operation. MobiBind only monitors for hand-overs in this state, and calculates link statistics (used in link awareness).

Hand-over Hand-over in progress and the link is down. All incoming data is buffered in the *Controller*. The *MIP buffer* is used to maintain seamless data flow.

Fast send Recovery period after hand-over. The buffered data, in the *Controller*, is sent over with harvest marker set, to the *MIP buffer*. Fast-send lasts until all packets in the sending buffer has been sent to the receiving side.

This state machine represents the remote binding, and the current state is held in *Meta*. The remote binding is made up of two independent parts. The two parts have their own *Meta* and therefore their own state.

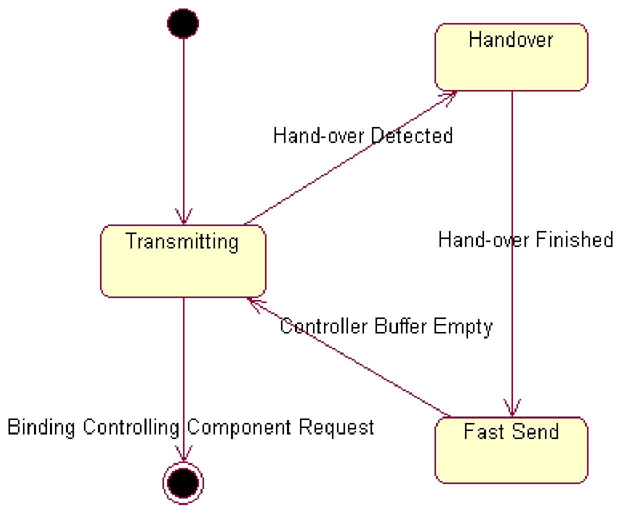

Figure 6.8: MobiBind State Diagram

State change

The state is kept in *Meta*. All objects inside the remote binding can notify *Meta* of a state change. There are different components that notifies about different state changes. It is the *Receiver* that alerts about the transaction from Transmitting to hand-over. This is where the hand-over detection algorithm lies. Hand-over to fasts end is also the *Receiver*, this is the object that knows when the hand-over is finished. Fasts end to transmitting is the responsibility of the *Controller*. Fasts end lasts until the *Controller's* buffer is empty, and that is something the *Controller* knows first.

When *Meta* is alerted of a state change it notifies the components that needs to alter behaviour. The current state is always available through *Meta's* `getState()`method. This is also something that is available through the controlling interface.

Because the two sides of the component are independent of one another, both sides of the component has its own state. While these should always be in synchronisation, it is possible for MobiBind as a whole to be out of synchronisation. This will not cause the component to crash, but may result in "none optimal performance", such as an increase in buffering and packet loss. The state change is mainly caused by a hand-over, therefore the de-synchronisation is caused by the two sides not detecting a, or wrongly detecting a hand-over. The problem of de-synchronisation and hand-over detection is the first area of the functional design, and are discussed next.

6.4 Functional Design

The functional design in this section gives the functionality needed to support the remaining requirements. Hand-over detection discusses how to detect a hand-over. The harvest mechanism provides seamless data flow and reduces packet loss. Link awareness is a function that enables the binding controlling components to access link information.

6.4.1 Hand-over Detection Algorithm

Hand-over detection is important because the requirement specifies that the QoS-Aware Remote Binding should be hand-over aware. It is also important to detect hand-overs when trying to reduce packet loss. The Hand-over time that this algorithm produces needs to be correct in order to secure seamless data flow, used by the harvest mechanism and available through the controlling interface.

The hand-over detection algorithm is also the algorithm that allows the state machine to change from transmitting to hand-over, and from hand-over to fast send. This implies two distinct operations in the hand-over detection algorithm: The first part is to detect the hand-over, the second to detect when the hand-over has ended.

There are two fundamentally different ways of detecting a hand-over: first, with the help of the underlying system, called notifications and second, without the help of the underlying system. Both approaches are discussed below.

Notifications

The remote binding may not itself detect the hand-over, but rather just get a notification when a hand-over is occurring. This notification could be given by mobility enabling software like Birdstep's Mobile IP client or by lower OSI layers. For instance the network card could give a notification when the LAN cable is disconnected. This would enable MobiBind to be very accurate in detecting hand-overs, and thus loosing fewer packets .If the notifications were given from the physical layer, like the different link cards, MobiBind would in some way have to detect all available cards and subscribe to notifications from each one. The mobility enabling software would perhaps be a better location to receive these notifications from. MobiBind would then only subscribe to notifications from that software. The problem with this approach is that currently no support in Birdstep's Mobile IP client or the lower layers for such notifications. This makes notifications not possible at this time.

Alternative Detection Algorithms

Notifications would enable the QoS-Aware Remote Binding to get notifications from the outside. This is not currently possible, so a way of detecting hand-overs based on indications that are observable to MobiBind is needed. We will first discuss some alternative detection methods, before the chosen design is presented and discussed.

Marker
One way to detect a hand-over is for the two sides of the remote binding to send a marker between them, when the marker it not returned, the link is down. The remote binding can then start to take the time of the hand-over. A problem here is to know when to re-send the marker, since there should only be one marker. And also, who sends it.

Packet loss
The remote binding could monitor all packets sent by the binding components, and require acknowledgements. If a defined number of packets are lost, it interprets that as a hand-over in progress. This approach would work well when the mobile node operates on a LAN link. The LAN link has very low packet loss and if a number of consecutive packets are lost, it is most likely a hand-over. On bad links, like WLAN, however the packets may be lost or delayed during normal operations, making the remote binding wrongly assume hand-over. This approach also requires that there is a steady stream of packets, because the remote binding only monitor the packets sent. If for instance three packets are sent within a few milliseconds on an unstable link they may be lost, but the link may not be down.

Another problem here is that if only one side produces data, only one side will detect the hand-over. This will create an unbalance between the two sides of the remote binding.

Timer
A timer may be used to regularly detect hand-overs. The timer may run a detection algorithm with a specified delay. The drawback is that this polling strategy will only detect hand-overs in the timer interval, giving less accuracy. The timer may also need to gather link information for each run. This data gathering may also take time.

Monitoring

The component may use special packets to send from one to another, when they no longer receive those packets it means that a hand-over is in progress. There are several ways to perform the monitoring. They may each send a stream to the other, or they may send pings to one another, that the other side replies to with a pong. An algorithm has to be used to be able to decide when a hand-over occurs. Either packet loss or timer could be used. The disadvantage of this approach is the extra traffic that the monitoring generates. On low bandwidth links this could become a problem.

There is also a possibility to monitor data-packets, and not generate own ping packets. This would require that regular data-packets are sent regularly, to actually detect hand-overs. When testing this it was shown to not produce good results. To reduce the number of pings sent, pings may only be sent when there are no data packets to send.

In MobiBind

Notifications from Mobil IP would be the best way to detect a hand-over. This option is not available today, and, thus; MobiBind was implemented with packet loss and monitoring. Preliminary testing showed that monitoring gave the best result in hand-over detection. Therefore, this was used to detect hand-overs. The time before a packet is marked as lost depends on the current RTT of the link (as described in Section 6.4.3). MobiBind has the RTT of the current link and the value of the RTT is used when deciding how long MobiBind should wait for a answer (pong) from the request (ping). This is an example of self adaptability.

The packet types used in monitoring are special signal packets called ping, with the answer from the receiver; pong. One side sends out pings, and receives pongs from the other as long as the link is up. When a hand-over occurs the pongs will stop and the *Receiver*, which notifies *Meta* about the hand-over. The end of hand-over is also an important function of the hand-over detection algorithm. The ping packets are always sent, and when the *Receiver* starts to receive pongs again (when in hand-over) it knows that the hand-over has ended. The number of pongs the *Receiver* needs to get before declaring end of hand-over is controlled by a parameter.

Securing Correct Hand-over Time

As described earlier the hand-over time should be a prediction of how long the next hand-over takes. The hand-over time is also available to the binding controlling components (through the controlling interface and *Meta*. It may be used for adaptation to hand-overs outside MobiBind. To do this we use the time of the last hand-over to predict the next one. When MobiBind starts it assumes that a hand-over takes 1000 ms. 1000 ms is assumed because this is what the test of Mobile IP showed. MobiBind monitors the actual time spent in each hand-over state, and uses this information to calculate a new hand-over time. The stored hand-over time is adjusted to reflect the hand-over-time just experienced. To stop the hand-over time from going up and down to fast, since wrong hand-over detections may cause very small hand-over-times (false positives), we need some kind of limitation. The hand-over time may be restrained by minimum and max values. However, this is not recommended since hand-over time may vary greatly depending on technology used to enable mobility. A better approach is to only allow the new hand-over time to adjust the old one by a fraction. This fraction is defined in the remote binding and may be set using parameters. The optimal fraction is not researched and therefore not known, but may depend on link properties and the number of false positives. Currently it is set to 5 %, which is a value that worked well in our test scenario. The formula looks like this:

```
If:
 new measured HT -/+ currentHT > currentHT*0.05 ->
               newHT=currentHT +/- currentHT*0.05
Else:
 newHT=new measured HT
```

False Positives and False Negatives

Two ways that the hand-over detection algorithm may fault is by producing false negatives and false positives. False positives and negatives have consequences both for the hand-over time and the later described harvest mechanism and link awareness. The consequences for the harvest mechanism and link awareness are discussed in detail in their respective sections.

False negatives is were the detection algorithm believes that a hand-over is not occurring when it is in fact occurring. This will not have implications to seamless data flow, since the pings and pongs are received (and thus the connection is not broken). It will however have consequences for the link Awareness. The link awareness, described later, is aware that link QoS may change drastically after a hand-over (imagine going from LAN to a bad WLAN link) so the QoS-Awareness has special functionality to handle this, but this require the hand-over detection algorithm to notice the hand-over. More on the consequences of false negatives in the QoS-Awareness in its section.

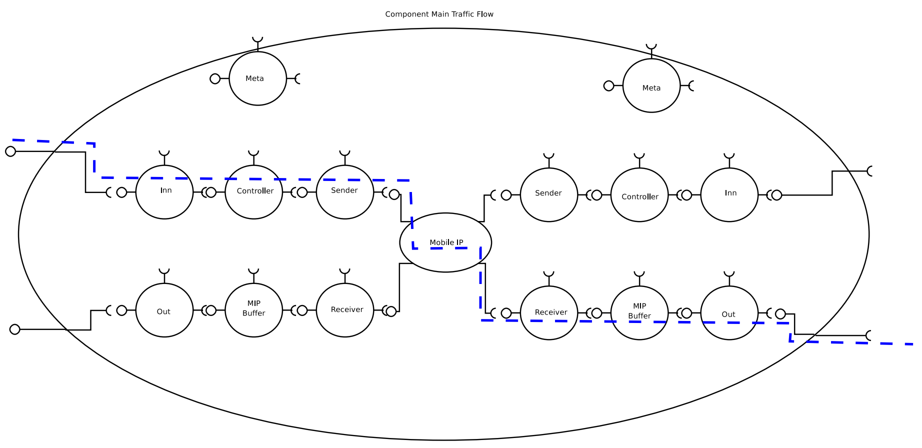

Figure 6.9: Normal Operation Before Adaption

False positives, when a hand-over is falsely flagged, might cause the hand-over time to be set very small, but as described above there are mechanism to limit the implications of this (the fraction). False positives will also effect the harvest mechanism, but not reduce QoS performance of MobiBind. False positives effects on the harvest mechanism are described later.

6.4.2 Harvest Mechanism

MobiBind is required to provide seamless data flow and reduce packet loss. Both these requirements are addressed in the harvest mechanism. The harvest mechanism uses a sender side buffer to reduce packet loss, and a receiving side buffer to provide seamless data flow. How this is done and which design decisions that are taken is described below.

Introduction

The central mechanism for the adaptation of MobiBind is called harvest, and is a way for the binding, in a generic way, to know how much data that needs to be buffered; in order to have seamless data delivery during hand-over. Harvest is a mechanism that effects every state of MobiBind. The main idea is to buffer the data that comes from the sender, while in the state of hand-over, and push those packets over to the receiving side *MIP buffer*. The receiving side will then be able to use the *MIP buffer* during the next hand-over, masking the effects of the hand-over. The two main objects involved is Controller on the sending side, and the *MIP buffer* on the receiving side.

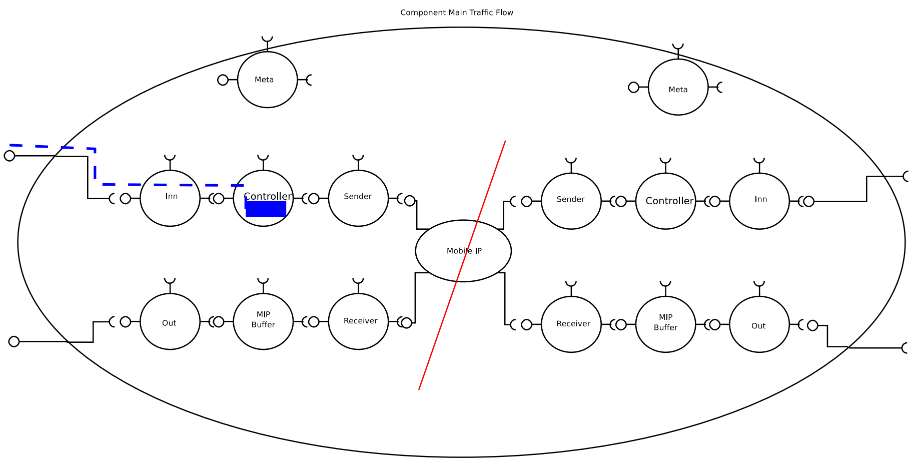

Figure 6.10: Hand-over in MobiBind

Since the hand-over detection algorithm is based on indications, false positives and false negatives might occur, and the harvest mechanism should be able to manage this.

The harvest mechanism has three modes of operation:

First is the normal mode, i.e., state transmitting. When a packet is sent in this state the packet goes straight through MobiBind and to the receiver. This is illustrated in Figure 6.9.

Secondly, when a hand-over is detected and the link is down the *Controller* starts to buffer all incoming packets, as illustrated in Figure 6.10. This causes the buffer to contain data needed for maintaining the next hand-over seamlessly. The first hand-over will incur a stop in the packet stream, as the remote binding uses this first hand-over to adapt.

Third, when the link has come up again, and the hand-over is finished, fast send begins. This is illustrated in Figure 6.11. In state fast send the data buffered in the *Controller* object are sent, at a high transfer rate to the *MIP buffer*. Fast send finishes when the entire buffer has been moved. The packets that are sent when moving the buffer is marked as harvest-type, causing them to remain in the *MIP buffer* until needed.

After the three states have been completed for the first time MobiBind has adapted, it can now seamlessly support hand-overs, since enough data now is in the *MIP buffer* on the receiving side, as shown in Figure 6.12. The next time a hand-over occurs the binding will save all packets received in the *Controller*, and at the same time use the *MIP buffer*. This causes the hand-over to appear seamless, illustrated in Figure 6.13.

The harvest mechanism manages false positives from the hand-over detection well. If the mechanism has adapted, the false positives will not have any noticeable effect to the binding components. All that happens

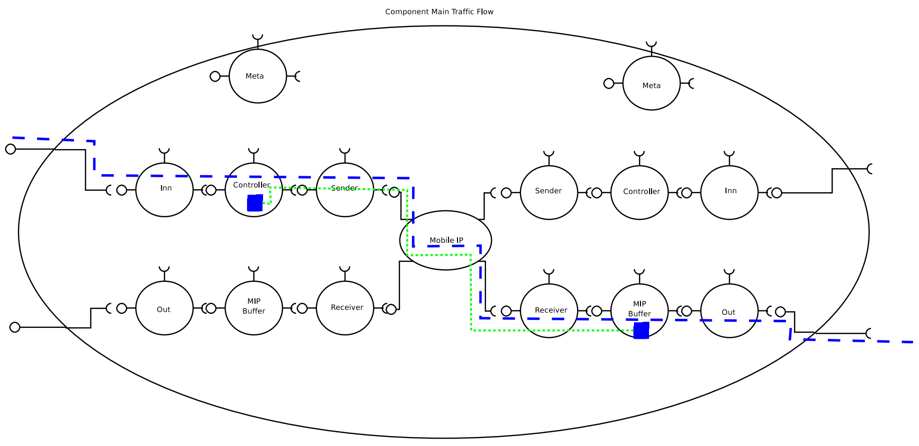

Figure 6.11: Fast Send in MobiBind

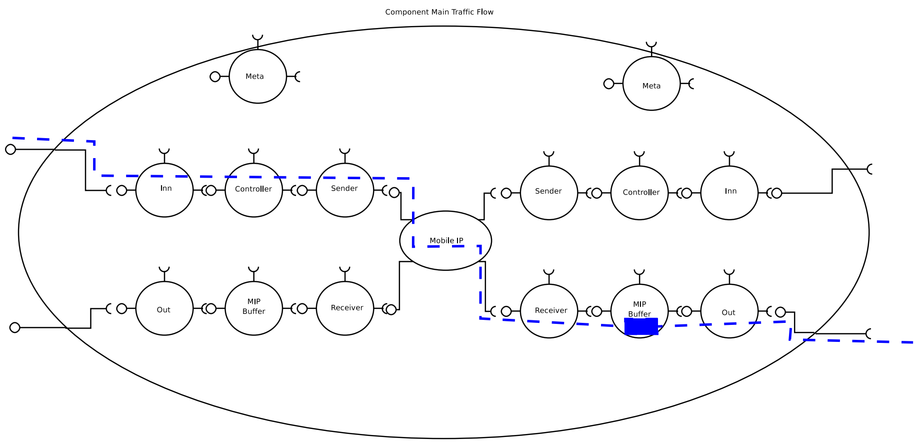

Figure 6.12: Normal Operation After Adaption

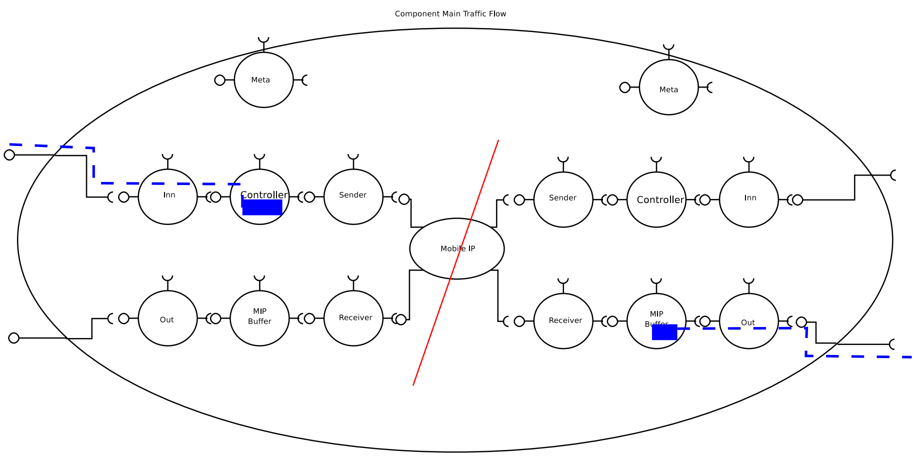

Figure 6.13: Hand-over After Adaption

is that the *MIP buffer* is used, emptied and refilled without any QoS degeneration, loss of seamlessness or packet loss. In fact, a False positive could be started to control when MobiBind should adapt to hand-overs.

False negatives will only happen if a link changes, without any connectivity loss, as stated earlier. And as long as the link is up, there is no problem for the harvest mechanism.

The frame of mind when it comes to the buffers is that the application should decide how much to buffer. So during hand-over all data is buffered, and then that amount is used during a later hand-over. If no data is received during the hand-over MobiBind assumes that the binding component did not have any thing to send. If data was collected, the packets are spaced evenly out from the *MIP buffer* during the next hand-over. Another way of doing this would be to specify a “packet space description” through the controlling interface, to provide information on how the *MIP buffer* should be used while in the hand-over.

Supporting low RTT traffic

MobiBind is designed to work at a low level and for no special traffic in mind. The harvest mechanism and the buffers it uses cause the RTT to increase when providing seamless data flow. In order to support traffic that is RTT sensitive MobiBind can change its inside structure. It can remove the *MIP buffer*, so that RTT is kept low, but packet loss is still kept to a minimum.

Meta can by a call through the controlling interface take out the *MIP Buffer* from the packet delivery stream. This is shown in Figure 6.14. The figure shows MobiBind with one MIP buffer taken out. This adapt-

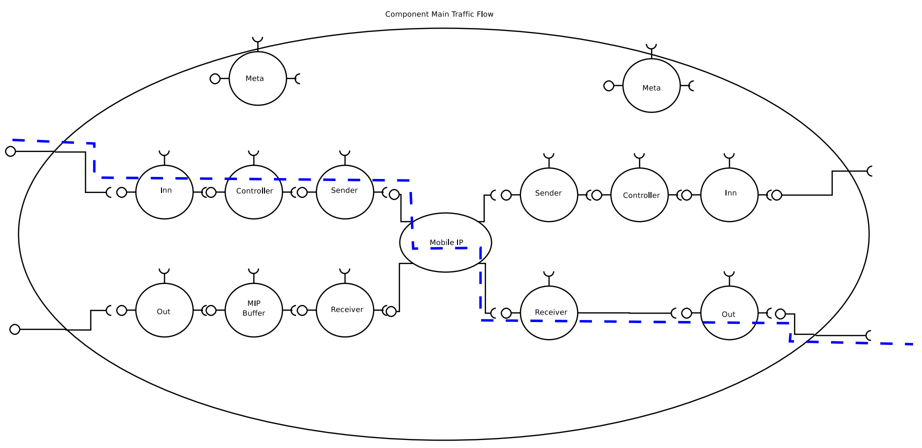

Figure 6.14: MobiBind Without MIP Buffer

ation can be done on both *MIP buffers*, or just the one, providing great flexibility.

When MobiBind receives a request to remove the *MIP buffer* it takes away the requested *MIP buffer* and connect the *Receiver* directly to *Out*. This allows both data packets and harvest packets to go directly from the *Receiver*, through *Out*, to the binding component. Figure 6.15 illustrated this principle during fast send. The packets that are sent from the *Controller* are not buffered at the receiving side, as normal, but sent directly to the binding component.

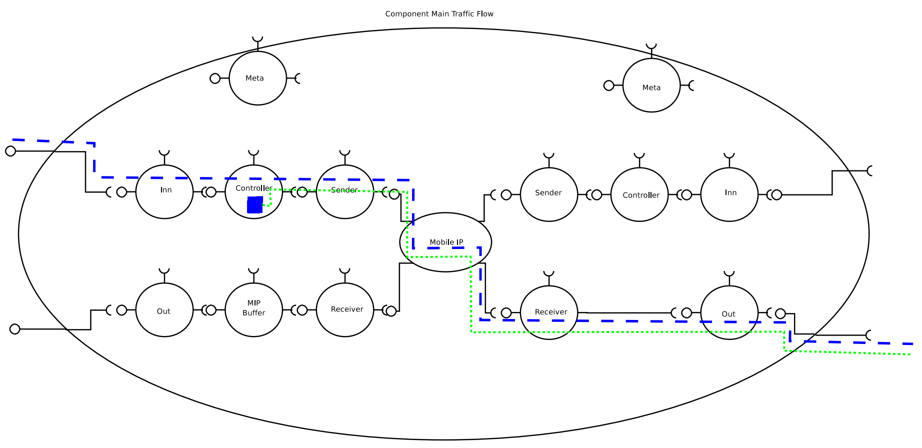

Figure 6.15: MobiBind Without MIP Buffer in Fast Send

Purging

The *MIP buffer* will after a hand-over contain a number of packets. In the transmitting state the MIP buffer size will not change; it only sends one packet out, for every data packet it receives. When the sender reaches the end of the stream the last portion of the streams data packets will remain in the MIP buffer. Purging addresses this problem.

The size of the *MIP buffer* does not change during the transmission state. This is normal behaviour. The packets in the buffer are going to be used to provide seamless connectivity during the next hand-over. At the end of the transmission however this is not wished behaviour. The last portion of the stream will remain in the MIP buffer.

This is avoided by adding a way for the binding controlling components to notify MobiBind that the end of the stream is reached. This is done via a call through the controlling interface called `purgeAllBuffers()`. The call goes to *Meta* and involves the *Receiver*, *Controller* and *MIP buffer*.

When *Meta* receives a request to empty buffers, it sends a request to all the *MIP buffers* to purge their buffers. The call effect the entire distributed binding; the remote *MIP buffer* is reached though MobiBind's signalling packets.

6.4.3 Link Aware

As stated by the requirements the QoS-Aware Remote Binding should support QoS monitoring. Link awareness is the functionality that provides this. The QoS parameters of the current link should be made available to MobiBind's inner objects and also to the binding controlling components. This information might help applications or other components to make decisions.

The lower layers does not support notification about current QoS of a link, so MobiBind must gather this information by it self. The *Controller* holds information about each packet sent, and receives an ack to each data packet from the other side of MobiBind. This information forms the base of the calculation, which is described later.

Characteristics

Which QoS parameters does the binding controlling components and MobiBind need information about, and which is it possible to implement?

All the normal characteristics, those that were tested in the Mobile IP tests, are also interesting here; RTT, jitter and bandwidth and packet loss.

Jitter and RTT can be monitored by the ack mechanism and the *Controller* object. The arrival time of the acks, together with sent time of the packets give both RTT and jitter. MobiBind uses the RTT in the hand-over detection algorithm. This is actually an excellent way of adapting. The RTT will vary, and therefore the hand-over detection algorithm also. This causes fewer wrong hand-over detections, as discussed in the hand-over detection algorithm, Section 6.4.1.

Bandwidth could have been used for the sending component, i.e. a video streaming server, to choose the correct codec. It would be more complex to measure bandwidth, since the entire link would have to be flooded, and that would not be good for QoS performance in a shared medium. As an alternative bandwidth could be fetched from the network link card. This is not currently available, so therefore the bandwidth is not available in MobiBind.

Bandwidth and saturation tests could be provided by the binding as tests. The binding controlling components could request the tests, knowing that it could not transmit at the same time. This is not implemented; the binding may not have sole ownership of the link, and others may suffer because of these tests. Also, the total bandwidth would not be something that can be controlled by a component in the middleware, other network entities may also produce uncontrollable throughput on to a shared medium.

Packet loss is not currently used inside MobiBind, but binding controlling components may benefit from the parameter when choosing whether to send small or large packets. The packet loss are also measured with the ack system. No ack means that the packet is lost.

These basic characteristics just described could be combined to support a higher level of link description. In the Mobil IP tests it was seen that there were significantly different RTT, packet loss etc on a LAN link and a WLAN link. One could have used this information to calculate if one currently was on a LAN or WLAN link. A computer can of course move to other links than LAN and WLAN, so this might limit the usefulness of this higher level description. One advantage of this is to possibly be able to predict the maximum throughput of the current link (by knowing each link's maximum throughput). To design a binding that could efficiently separate between different high level links, test of technologies like GPRS and 3G would need to be performed, and information that distinguishes these technologies from each other should be known to the remote binding. It is also possible that this approach would not be plausible when the number of link technologies is high and the distinguishing factors are low. At this point in time no such information is available, so this feature is currently not implemented.

Today MobiBind supports jitter, RTT and packet loss QoS parameters. They can be access through the controlling interface.

Calculation

As noted above the calculation of the QoS parameters is done by comparing the sent time of the MobiBind data-packet with the time the ack was received. This is all done in the *Controller*. The *Controller* records when the packet was sent, and the receiver informs the *Controller* that the ack was received. Calculation of link awareness and the characteristics is done in the *Controller*.

The calculation is only performed when MobiBind is in the transmitting state, and only if there is sufficient number of packets sent on the link. Update time of the link characteristics and number of packets needed to perform calculation can be adjusted through parameters.

During hand-over the packets sent on the old link is not used for statistics, as they do not represent the new link. These packets are marked as stale and are not used to calculate link characteristics. This prevents old QoS information to be use on the new link

False positives from the hand-over detection algorithm will cause a small interruption in the link awareness, because all packets are marked stale. The QoS information will be available again when enough packets are sent (after the false positive).

False negatives will cause the link awareness to use link information from the old link on the new link. This may cause binding controlling components to get wrong link awareness information when querying.

6.5 Example of Use

MobiBind is a versatile QoS-Aware Remote Binding. It is designed at a low level of the OSI model to be used as a part of the middleware. It can either be used by advanced components alone, or as part of a nested open binding, to provide a complete service simple components.

This section presents two example of use of MobiBind. Example one in Section 6.5.1 presents a use of MobiBind together with a set of advanced video consumer and producers. Example two in Section 6.5.2 presents a MobiBind together with simple components, and highlight that MobiBind can be used for adaption.

6.5.1 Example 1

Figure 6.16 shows MobiBind together in a video streaming example. With an advanced video producer, and an advanced video consumer. The video consumer is in a mobile address space. The video stream is encoded by the advanced video producer and decoded by the advanced video consumer. All the two video components need MobiBind for, is

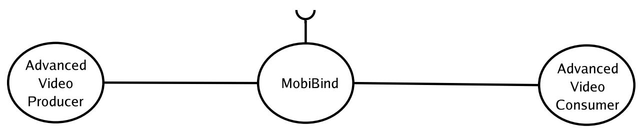

Figure 6.16: Example: MobiBind and Advanced Components

to provide a seamless connection in the mobile setting. The video server further uses the controlling interface of MobiBind to get link information. This information allows the Advanced video producer to adapt and possibly change the decompression algorithm.

6.5.2 Example 2

The second example is also a video streaming example, but with simple video producer and consumer. The simple components are only capable of producing and consuming raw video streams, so more functionality is required to connect the two components across a network. The simple components need to use the middleware in an more advanced way to create a binding between the two. Figure 6.17 shows how the simple video producer uses a binding factory to create a lightweight nested open binding. First, in A the simple video producer requests a binding from a binding factory. The request contains a requirement for mobility and high quality video streaming. The binding factory will then go to the library of available components to create a binding that satisfies the simple video producer's request. In B the binding factory finds MobiBind that enables mobile, seamless, connectivity. The binding factory uses MobiBind as a base. But the simple video producer and consumer does not have any video compression So the binding factory creates a nested open binding that contains high quality MPEG2 compression and MobiBind. In C the video producer and consumer uses this newly created nested open binding. The nested open binding can be self adaptive, or the middleware might adapt the binding if needed, as illustrated in D; By using the controlling interface of MobiBind the nested open binding can read link awareness information. If for instance the link becomes bad and heavy jitter is reported by MobiBind, the nested open binding may adapt by inserting a jitter buffer. This adaption is illustrated in E.

6.6 Summary

This chapter presents a comprehensive design of a QoS-Aware Remote Binding, called MobiBind. MobiBind is designed at a low level in the

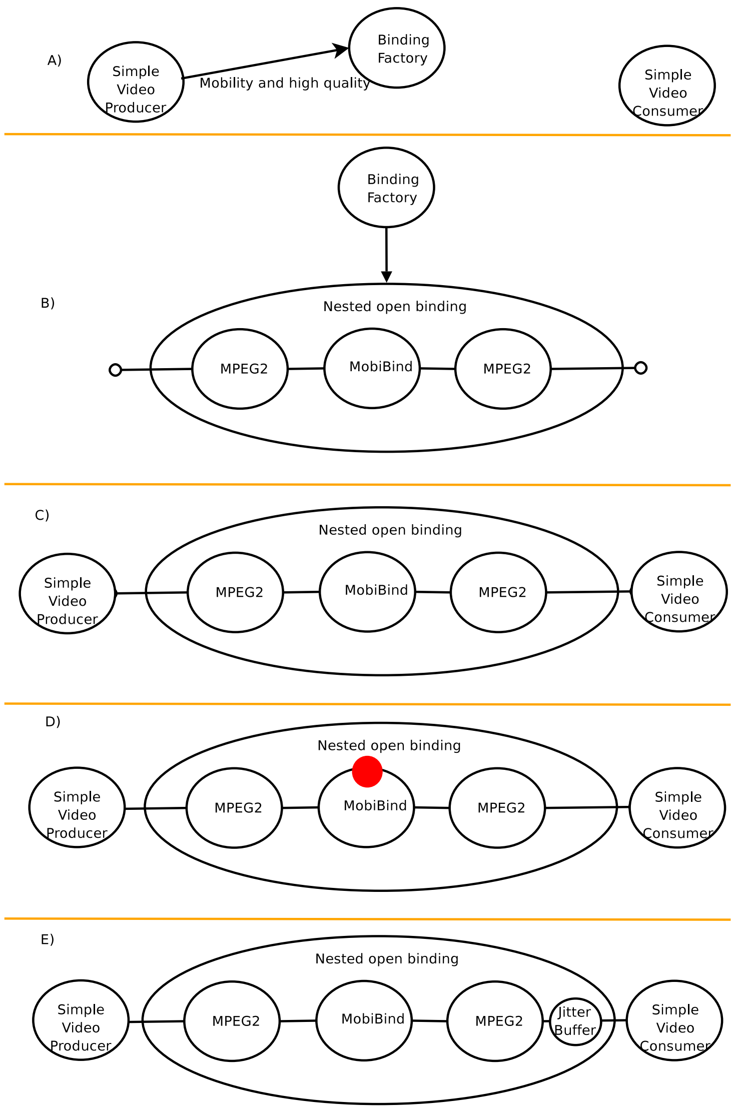

Figure 6.17: Example: MobiBind and Simple Components

OSI model. This enables MobiBind to be used directly by advanced components, as showed in example one, or together with other components inside a nested open binding, which example two describes a scenario of.

Key functionality as hand-over detection, harvest and link awareness assures that the requirements from Chapter 5 are fulfilled.

Chapter 7

MobiBind Implementation

In order to test the proposed design it needs to be implemented. This chapter presents the implementation of MobiBind. The chapter starts out with a discussion of the chosen implementation language. Then continues with a presentation of implemented key functionality.

7.1 Implementation Language

When choosing a programming language to implement MobiBind in there was several considerations. Firstly, the programming language should be a good language to prototype in; It should be high level and give good feedback to the programmer. It should also be object oriented, since the design is object oriented.

Secondly, the QuA project, which this thesis is a part of, is implemented in Java and Smalltalk. So for MobiBind to be compatible with this project it needs to be implemented in either of the two languages.

Having both the tests tool and the remote binding implemented in the same language gave an advantage in making to two applications easy to join. The tests tool KGen was already implemented in Java, making it possible to integrate MobiBind's interfaces into it. Java SE 5.0 was chosen because it met our requirements.

7.2 MobiBind Code Presentation

This section presents the implementation of five key aspects: parameter values, hand-over detection, link awareness, harvest mechanism, and state transitions. The objective is to give a better understanding of the proposed design of MobiBind; the QoS-Aware Remote Binding. The Java source code presented here is enhanced for easier understanding. To view the complete implementation, refer to the enclosed CD.

Parameter	**Value**	**Designation**
FASTSEND_GAP	20	ms
PACKET_HANDOVERTHREASHOLD	8	packets
CONTROLLER_BUFFERSIZE	12000	packets
MAX_HANDOVER_CHANGE	5	%
LINKAWARE_MINIMUM	10	packets
LINK_DETECTION_INTERVAL	500	ms
MAX_SEND_RATE	20	ms
HANDOVERSTREAM_GAP	20	ms
PINGS_AFTER_HANDOVER	10	packets

Table 7.1: Parameters in MobiBind

7.2.1 Utility Class and Parameters

When implementing the binding some code was repeatedly used. This was especially true for code having to with taking the MobiBind packet in and out of the UDP packet. This code was separated out to a utility class called Util.

Static parameters were also placed in the utility class. To make it easy to test out new parameters values, and to ensure that both sides of the binding have the same parameter values. Different values were tested for the parameters to find values that performed well in the forthcoming tests.

Table 7.1 lists of the parameters included in Util.

When implementing the MobiBind we found that during purge we had to have some limitation on how fast the buffers were sent through the network, if the packets were sent to fast heavy packet loss was encountered. The parameter MAX_SEND_RATE sets the max send rate during purge.

7.2.2 Hand-over Detection Algorithm

There are two object within MobiBind that enables hand-over detection; Receiver and Sender. The implemented algorithm uses, as the design noted, a ping and pong monitoring scheme. The Sender is a ping generator, while the Receiver both reflect pings and analyses pongs.

Figure 7.1 presents a illustration of the hand-over detection algorithm.

Sender

The Sender sends a Util.SIGNAL packet with Util.PING_INT signal. The method was implemented as a thread in an inner class of the Sender

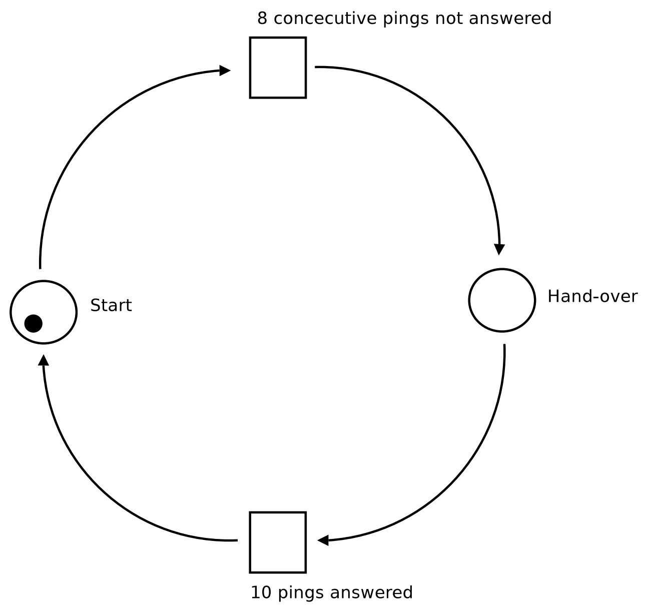

Figure 7.1: Hand-over Detection Algorithm

class. The internal class looks like this:

```
class Hand-overStream extends TimerTask implements Runnable {
    public void run() {
        Datapacket dp = new Datapacket(new byte[1]);
        dp.setType(Util.SIGNAL);
        dp.setSequeceNumber(Util.PING_INT);
        sendDatapacket(dp);
    }
}
```

The parameter Util.HANDOVERSTREAM_GAP controls how often the thread is run, currently set to 20 ms. Every time the thread is run it creates a new data-packet, sets the packet type, and the PING signal type. It then sends the packet.

Receiver

The Receiver's job in the hand-over detection is two folded, first, when it receives a ping it sends back a pong. If the Receiver receives a pong, it passes it on to the hand-over detection monitor (called ping-monitor), implemented as a inner class of the Receiver. The code executed when receiving a signal packet is as follows:

```
if (datapacket.getType() == Util.SIGNAL) {
   if (datapacket.getSequencenumber() == Util.PING_INT) {
     sendPong();
   }
   if(datapacket.getSequencenumber() == Util.PONG_INT) {
     pingmonitor.pongReceived();
   }
   if (datapacket.getSequencenumber() == Util.PURGE_INT) {
     controller.purgeBuffer();
     next.purge();
   }
}
```

The ping-monitor registers the time the pong was received, and uses this information when the monitor is run. The monitor is executed at the same interval as the Sender's ping interval.

```
  if (pong_not_received_time > maximum_allowed_waiting_time) {
    if (meta.getState() == Util.TRANSMITTING) {
      pingsAfterHand-over = 0;
      System.out.println("PingMonitor: Hand-over detected");
      meta.setState(Util.HANDOVER);
    }
  } else if (meta.getState() == Util.HANDOVER) {
    if (pingsAfterHand-over > Util.PINGS_AFTER_HANDOVER) {
      System.out.println("PingMonitor: Hand-over ended");
      meta.setState(Util.FASTSEND);
    }
  }
```

The code section showed above is executed when the connection is up. It checks whether pongs has been received within the maximum allowed waiting time. If not, and the state is transmitting, the state is changed to hand-over. If a packet is received, it may change state from hand-over to fast send, this change is controlled by the number of pongs received after the hand-over was initiated. This increment count is conducted each time a pong is received. In addition to the code above the Receiver checks if the connection between the two parts is up yet, before performing the above code.

The method below shows is the method that is responsible for receiving the pongs. It notes the time of arrival of the pong, and also counts them,

```
  private void pongReceived() {
    addToLastreceived();
    if (buffer[bufferLastReceived] == null) {
      buffer[bufferLastReceived] = new BufferLine();
    }
    buffer[bufferLastReceived].setSent(System.nanoTime());
    if(meta.getState() == Util.HANDOVER){
      pingsAfterHand-over++;
      System.out.println("pong received while in hand-over nr:"
      + pingsAfterHand-over);
    }
  }
```

7.2.3 Link Awareness

The link awareness algorithm measures the QoS parameters; packet loss, RTT and jitter, and is implemented as an inner class in the Controller class. It uses the packet sent and ack received times gathered by the binding and stored in the Controller's buffer. The Controller's buffer is directly available to the LinkDetection class because it is design as an inner class.

The class executes in intervals defined by the parameter Util.LINK_DETECTION_INTERVAL. If there are enough packets, and the information is not stale (it is marked as stale after a hand-over). QoS calculation is performed and the information is given to the meta object.

Below is an exert of the LinkDetection class.

```
class LinkDetection extends TimerTask implements Runnable {
  public void run() {
    int index = bufferAkkIndex;
    if (index < Util.LINKAWARE_MINIMUM) {
      return;
    }

    //check if enough packets to peform link detection.
    if (buffer[index - Util.LINKAWARE_MINIMUM] == null ||
    buffer[index - Util.LINKAWARE_MINIMUM].stale == true) {
      meta.setrtt( -1);
      meta.setjitter( -1);
      meta.setpacketloss( -1);

    } else {
      // RTT
      calculateRTT();
      meta.setrtt(rtt);

      // Jitter
      calculcateJitter();
      meta.setjitter(jitter);

      // Packet loss
      calculcatePacketLoss();
      meta.setPacketLoss(packetloss);
    }
  }
}
```

7.2.4 Harvest Mechanism

The harvest mechanism is the main mechanism for handling the hand-over. Firstly, the Controller, that is responsible for sending additional packets during fast send, and marking them with the harvest-packet-type. Secondly, the MIP Buffer that is responsible for harvesting the extra packets, and using them during the hand-overs.

Controller

The fast send operation sends all the packets in the controllers buffer, using the parameter Util.FASTSEND_GAP as a bandwidth limit. Finally it sets the state back to transmitting.

The fast send algorithm, implemented in an inner class of the Controller, is shown here:

```
class FastSend extends Thread {
  public void run() {
    // calculate how many packets to send
    int numberofpackets = bufferInputIndex - bufferOutputIndex;
    // send the packets
    for (int i = 0; i < numberofpackets; i++) {
      sendDatapacket(true);
      try {
        Thread.sleep(Util.FASTSEND_GAP);
      } catch (InterruptedException ex) {
        ex.printStackTrace();
      }
    }
    meta.setState(Util.TRANSMITTING);
  }
}
```

MIP Buffer

The MIP Buffer has two main functions during hand-over, first receive harvest packets during fast send, second use packets during hand-over. While in fast send the packets are received by the following method:

```
  public void receiveDatapacket(Datapacket datapacket) {

    //store in buffer
    buffer[bufferInIndex] = datapacket;
    updateBufferInindex();

    //if data, send out one from buffer
    if (datapacket.getType() == Util.DATA) {
      sendPacket();
    } else if (datapacket.getType() == Util.AKK) {
      System.out.println("Error: ack came to MIPBuffer");
    }else if (datapacket.getType() == Util.HARVEST){
      //if harvest do nothing
      //nop
    }
  }
```

This method stores the incoming packet in the buffer, and if the incoming packet was a harvest packet, no further action is taken. This causes the buffer to increase.

In the hand-over state, the MIP Buffer gets a message from the meta object to start using its buffer, to provide seamless data flow. The inner class BufferUser is activated by the MIP Buffer. The class BufferUser gets the number of packets to use and the interval gap. A way to space out the packets from the MIP buffer is needed. The number of packets is simply the total number of buffered packets in the MIP Buffer, the interval gap used is actually an example of self adaptation. The interval gap is calculated by taking the expected hand-over-time (the MIP Buffer queries the meta object for this) and divides that time by the number of packets that is in the buffer. This ensures a steady flow of packets during the entire hand-over.

```
  class BufferUser extends Thread {
    int gap = 0;
    int number = 0;
```

```
        BufferUser(int sendgap, int number) {
          this.gap = sendgap;
          this.number = number;
        }

        public void run() {
          System.out.println(
          "component.MIPBuffer.BufferUser: Starting to use buffer");
          for (int i = 0; i < number; i++) {
            try {
              sendPacket();
              Thread.sleep(gap);
            } catch (InterruptedException ex) {
              ex.printStackTrace();
            }
          }
          System.out.println(
          "component.MIPBuffer.BufferUser:" +
          "buffer used (stopping to use buffer)");

        }

    }
```

7.2.5 State Change

The design from the previous chapter presents a state machine. Different objects in the remote binding are responsible for detecting different state transitions. All state transitions goes through meta. The method used for changing states, implemented in the meta object is shown here:

```
  public void setState(short state) {

    // Change state
    this.state = state;

    //implement state change
    if (this.state == Util.HANDOVER) {
      setHand-overStartTime();
      numberofhand-overs++;
      controller.startBuffering();
      mipbuffer.useBuffer();
      // set link information stale
      controller.setBufferStale();
    } else if (this.state == Util.FASTSEND) {
      controller.stopBuffering();
      calculateNewHand-overtime();
      controller.startFastSend();
    } else if (this.state == Util.TRANSMITTING) {
    .
    .
    .
    } else {
      System.out.println("component.Meta: Unknown state");
    }
  }
```

Is here clearly shown that meta holds the state and also informs the other objects of the state change, by telling the objects what to do. Object can always check the current state by using the getState() method.

7.3 Implementation Summary

This chapter has highlighted some key functionality of the MobiBind implementation. The implementation is used together with KGen to test the proposed design in the next chapter.

Chapter 8

MobiBind Test

This chapter discusses the tests performed on MobiBind. The tests of MobiBind have two objectives: Validate the functionality of the proposed design, and compare the QoS characteristics of the remote binding against the raw Mobile IP solution. First the tests configuration is presented. This is followed by modifications done to KGen in order for KGen to use MobiBind. Then follows the description of the tests performed. The test are divided into basic and advanced tests. Finally, the tests results are presented.

8.1 Test Configuration

The test configuration used here was the same as for the Mobile IP tests. The general test configuration, with its strengths and limitations, is discussed in Chapter 3. MobiBind was used together with KGen. This enabled efficient testing. Since KGen were also made specifically for this thesis, we had full freedom in getting out tests results wherever needed.

8.2 KGen Modification

The generator KGen used to test Mobile IP in Chapter 4 was also used to test MobiBind. By using the same tests tool the test results could be compared. As MobiBind is an explicit remote binding, and has special interfaces that the using components need to connect to, it was necessary to do some small rewrites of KGen.

In the Mobile IP tests, KGen used a UDP socket to send and receive the data packets. The major modification to KGen was to remove the UDP socket and instead use MobiBind. This did not require big alterations; the initialisation of the UDP socket was removed, and replaced by the initialisation of MobiBind. The code for sending packets through the UDP socket in the original KGen, looked like this:

```
datasocket.send(packet);
```

This was changed to use MobiBind's interface, and the send method. The new code line, in the modified KGen, looked like this:

```
in.sendPacket(databuffer);
```

The last change,was accessing the controlling interface of MobiBind. The modified version of KGen can be found on the enclosed CD.

8.3 Test Description

The tests have two objectives, first to compare the performance of MobiBind with the basic Mobile IP implementation. This is done in the basic tests section. Then, to validate the advanced functionality like adaption, harvest mechanism, link monitoring and hand-over detection algorithm. This is discussed under advanced testing.

8.3.1 Basic Testing

The basic tests were performed in the same way as the Mobile IP tests from chapter 4. This enabled us to compare the performance of MobiBind with the basic Mobile IP implementation. Tests were performed on several days and to stable findings could be extrapolated. No hand-overs were performed here, this was done in the advanced testing. The absence of hand-overs allowed us to control MobiBind for false positives.

8.3.2 Advanced Testing

The advanced tests aimed at testing the advanced, new, functionality provided by MobiBind. The harvest mechanism was tested, to see if it actually worked according to its design goals. The harvest mechanism was designed to adapt to hand-overs and provide seamless data connectivity. Purging at the end of transmission was also tested. The same was MobiBind with and without the MIP Buffer. The link awareness was tested by comparing the link statistics with the one KGen produced. The stale function in link awareness, that stops the link awareness calculation from using data from an old link after a hand-over, was also tested. The hand-over detection algorithm was also tested. This was done by producing a hand-over, and checking whether MobiBind detected the hand-over. The algorithm was also checked by running MobiBind on a bad link (large packet loss, large jitter), without any hand-overs, to see if a hand-over was falsely detected.

8.4 Test Results

8.4.1 Basic Tests

The basic tests showed that MobiBind has the same basic tests results in the basic tests, like jitter, RTT and packet loss, as the raw Mobile IP implementation, tested in Chapter 4. RTT was about 5% higher than on the raw Mobile IP implementation. This is probably due to the extra overhead in CPU time of the remote binding, and the extra network overhead due to the remote binding packets. MobiBind adds a 5 byte header on each packet. MobiBind was stable and performed well.

On bad WLAN links MobiBind did sometimes faulty detect a hand-over. This was of course not the actual hand-over, since no hand-overs were performed during the basic testing. The false hand-over detection occurred because MobiBind did not receive any pongs for the designated period. MobiBind may have sent or received some data traffic, but the detection algorithm just checks pongs. We discuss this problem in the following section (Advanced Tests).

8.4.2 Advanced Tests

All the implemented functionality worked as expected. The link awareness algorithm regularly performed its calculations, based on the parameter setting. Using only information that was not stale (collected from a previously used link). The link awareness information was available through the controlling interface.

The hand-over detection algorithm continuously checked for hand-over, and also reported to *Meta* when the hand-over was finished. This caused *Meta* to calculate a new expected hand-over time. The new hand-over time did not change more that the maximum fraction allowed per adjustment, controlled by the parameter MAX_HANDOVER_CHANGE. The harvest mechanism did also work as designed. It provided seamless data connectivity, after the initial adaption.

Figure 8.1 shows a typical graph of a test run. When MobiBind starts to transmit the RTT is low, about 5 ms in this test. When MobiBind detects its first hand-over, it adapts. This is an automated function. In the first hand-over, the binding looses connectivity for about one second. This is where MobiBind adapts by increasing the *MIP Buffer*. Chapter 9 discusses an idea on how to make this algorithm better. After the first adaption the RTT is about 1 second. This is due to the packets buffered at

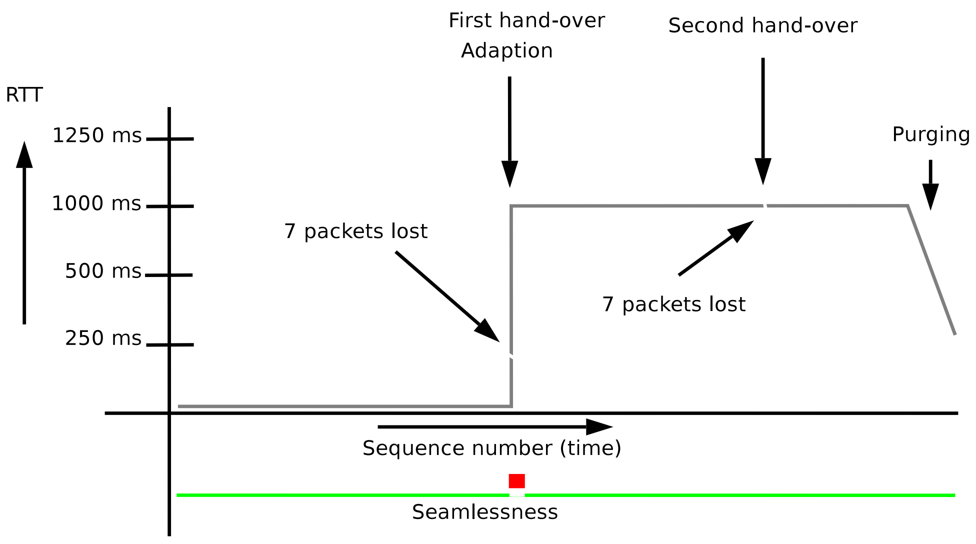

Figure 8.1: Test Run of MobiBind

the sending side *Controller*. The packets are transferred from the *Controller* to the other side's *MIP Buffer* during the state fast send. This transportation does not effect the RTT, as the buffer only changes place, not size. When the second hand-over is detected it uses the *MIP Buffer*, and at the same time stores new packets in the sender side's Controller. The data transmission is now seamless.

The parameters that controls the number of ping/pongs lost before assuming a hand-over and the number of ping/pongs that has to be received before a hand-over is ended was manipulated during the tests. It was found that WLAN is more unstable and easier trigger false hand-over detections than LAN. WLAN required a higher setting of the number of packets both to be lost and received after a hand-over. 7 packets seemed to be a number that worked well on WLAN. 7 Packets also worked on LAN, but LAN did not require 7 packets, since the technology is more stable than WLAN and the number can be adjusted down. The pings were sent every 20 ms, so 7 packets gives a maximum of 140 ms before a hand-over actually happens until it is detected. It also took some time before the hand-over had ended until MobiBind moves away from the hand-over state. This is because the parameter, that controls how many pongs that need to be received before a hand-over is finished, was set to more than one. The value one did not work because it made the remote binding detect that the hand-over was over before it actually had ended. The reason may be that Mobile IP allows for some connectivity before the new link is fully stable, or that an old pong is arriving late. This late finish-detection increases the hand-over time. Generally, the hand-over time was from 0% to 10% higher with the remote binding than the raw Mobile IP implementation.

The hand-over detection algorithm and harvest mechanism decreases the packet loss significantly. During a hand-over in MobiBind 5 to 10 packets were lost, usually 7 (when sending packets every 20 ms). The raw Mobile IP implementation lost about 50 packets. This is a huge improvement in the packet loss parameter for MobiBind. The harvest mechanism does not lose any packets. The packets are lost due to the hand-over detection algorithm that is based on assumptions and therefore has some delay in detecting a hand-over. The parameters for hand-over detection may be adjusted to further minimise packet loss.

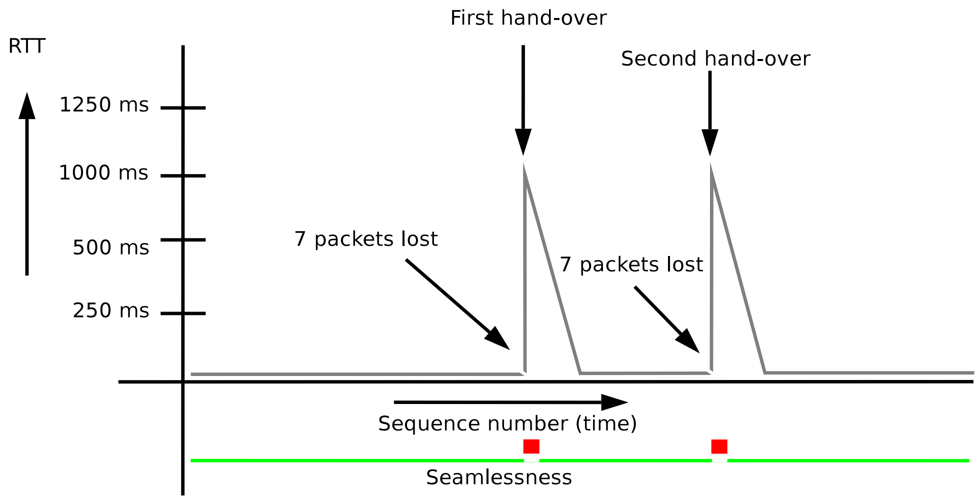

Figure 8.2: MobiBind Without MIP Buffer

Figure 8.2 shows how MobiBind behaved when the *MIP buffer* was taken out to support low RTT traffic. As seen on the figure RTT is kept low, except when the packets are temporarily stored in the *Controller* during hand-over. This function also worked as designed. The length and steepness of the “RTT-tail”, after the hand-over, can be adjusted by the fast send parameter.

8.5 Test Summary

The tests showed that MobiBind did provide seamless connectivity, after the initial adaption. The tests also showed that this connectivity came at an expense of the RTT. Packet loss was also greatly reduced. By turning off the *MIP Buffer* the remote binding reduced packet loss, and at the same time kept the RTT to a minimum.

Link awareness information was calculated and made available through the controlling interface.

During the testing, the implementation was stable and performed well. Some of the parameters may need further adjustment to perform optimal.

Chapter 9

Thesis Assessment

This chapter first assess if the QoS-Aware Remote Binding requirements were met in MobiBind. Then the work performed in this thesis is assessed against the problem statement.

9.1 QoS-Aware Remote Binding Requirements

9.1.1 Requirement 1: Mobile Connectivity

The remote binding must connect two mobile components together.

This requirement is fully met. The basic component structure and the dependencies to Mobile IP ensure this. In the test of MobiBind we successfully connected two "components" together, KGen server and client. They exchange data between each other.

9.1.2 Requirement 2: Mobility Management

The remote binding must implement Mobile IP for mobility management.

This requirement is fully met. The requirement is met by the dependence to Mobile IP. In MobiBind this dependence was realised by Birdstep's Mobile IP client, which worked well together with MobiBind in the tests. Basic mobile connectivity was never a problem.

9.1.3 Requirement 3: Two-way Connectivity

The remote binding should provide a two way connection.

This requirement is fully met. MobiBind's design and implementation provides a receiving and sending interface, as well as a controlling interface to the components using it. All Interfaces can be used concurrently, and provide full two way connectivity.

9.1.4 Requirement 4: Explicit Remote Binding

The remote binding should be designed as an Explicit remote binding.

This requirement is fully met. The MobiBind provides a design that conforms with a explicit remote binding design. Both controlling and sending interfaces are provided. Binding components can connect to the packet based interface directly or MobiBind can be used in a nested open binding. Binding controlling components can get information about current QoS and hand-overs through the controlling interface. MobiBind also supports adaption; through the controlling interface MobiBind can manipulate its internal structure to support low RTT traffic.

9.1.5 Requirement 5: QoS Monitoring

The remote binding must provide QoS monitoring.

This requirement is fully met. MobiBind provides key QoS parameters, like packet loss, RTT and jitter, with the link awareness function. MobiBind also uses some of these characteristics to perform self adaption. The link awareness comes at a cost; additional complexity, internal sequence numbers and acks together with the computation of the QoS-parameter values. The Sequence numbers and acks allow MobiBind to be upgraded easily to provide lossless data transmission. QoS monitoring allows the binding controlling component and middleware to get more information out of the binding, which it can use at it sees fit. Because information used to calculate the QoS-parameter values are set stale after a hand-over, the values become even more reliable.

9.1.6 Requirement 6: Hand-over Aware

The remote binding must minimise packet loss during hand-over.

This requirement is fully met. The harvest mechanism enables MobiBind to limit the packet loss during hand-over. The MobiBind tests show that packet loss is reduced when using MobiBind, compared to the raw Mobil IP implementation. Remaining packet loss is due to a delay in the hand-over detection algorithm. MobiBind has the potential to further limit packet loss during hand-overs, if the hand-over detection functionality is improved, e.g., based on notifications.

9.1.7 Requirement 7: Seamless Data Connection

The remote binding should maintain data flow during hand-over.

This requirement is met. MobiBind provides seamless data flow, it does however provide this only after an initial adaption. This adaption takes place during the first hand-over. This is a natural place to perform the adaption, since the link is down, and the adaption does not degrade current performance, but heightens latter performance, by providing seamless data connection during hand-overs. This means that data flow is not maintained during the first hand-over.

In MobiBind, the adaption can only be triggered by hand-overs, not by the binding controlling components. MobiBind could have provided a method to initiate the adaption. The adaption could then be done at the binding controlling component's convenience, to prevent connection breakage during the first hand-over, this is illustrated in Figure 9.1. In this figure the adaption is requested at a time that is convenient for the component and user (e.g., during non vital communication). Thus, during the first and consecutive hand-overs the connection is seamless. The idea should be research further, to provide even better seamless data connectivity.

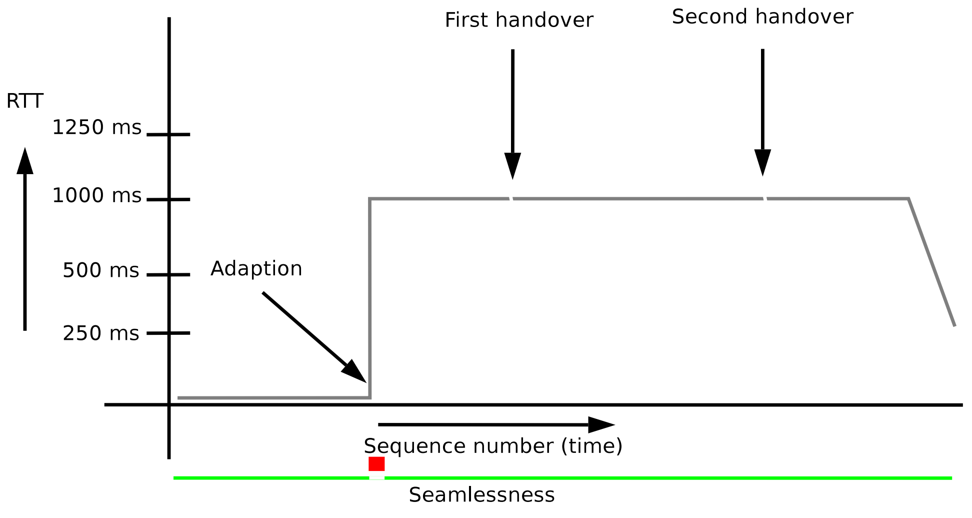

Figure 9.1: Binding Component Initialised Adaption

9.1.8 Requirement 8: Hand-over Monitoring

The remote binding should provide hand-over monitoring.

This requirement is fully met. Hand-over Monitoring is provided by the hand-over detection algorithm. Several alternatives to hand-over detection algorithms as discussed in Section 6.4.1, was implemented to see who functioned best. It was found that the chosen algorithm worked best in our test lab. The Hand-over detection algorithm does still cause some packet loss and increases the hand-over time. This is because the hand-over detection algorithm is implemented using assumptions, not notifications. The hand-over detection algorithm also gives the binding controlling components access to expected hand-over time.

The hand-over detection algorithm would probably function better if it received information from other sources closer to the actual link and Mobile IP technology. As an example: If the link that is currently used is disconnected from the link card, the link cards controlling software could report this directly to MobiBind. MobiBind could then immediately assume hand-over. This would further reduce packet loss, and potentially reduce packet loss to zero. In some cases the Mobile IP software decides to initiate a hand-over because a new preferred link has become available, in the current implementation, the Mobile IP implementation and MobiBind don't share information, so the number of pong-packets decided by the controlling parameter still needs to be lost before a hand-over is detected by MobiBind. If the Mobile IP software could notify MobiBind before the hand-over, no packets would need to be lost in this scenario.

9.2 Thesis Problem Statement

The problem definition was divided in to three sub problems to split the main problem into smaller problems that could be answered separately. First, the sub-problems will be discussed one by one. At the end of this chapter, the main problem is discussed.

9.2.1 Sub-problem 1

Which QoS parameters underperform in mobility management compared to fixed networks?

To find the underperforming QoS parameters we tested Mobile IP in a general test lab. Several of the tests results found were compared to findings from other researches. Several QoS parameters were found to underperform. We found that specific for mobility management is that the QoS parameters packet loss and seamlessness performed poorly. We found differences between LAN and WLAN, which could be used to separate between the two links on a high level. We found interesting max throughput results.

9.2.2 Sub-problem 2

What would be required of a QoS-Aware Remote Binding from an application developer's point of view?

The requirements of a QoS-Aware Remote Binding are discussed in Chapter 5, based on the underperforming QoS parameters. We identified 8 requirements, ranging from mobile connectivity to QoS-monitoring and requirements for seamlessness and reduced packet loss. The requirement specification gives an application developer a powerful tool in providing QoS-aware applications in an mobile environment.

9.2.3 Sub-problem 3

Is it possible to design and implement a QoS-Aware Remote Binding that enhance QoS-parameters performance?

This sub-problem was answered by taking the requirements from Chapter 5 and designing a QoS-Aware Remote Binding that would satisfy these requirements. The designed remote binding, MobiBind, provides enhancements to QoS-parameters in the following functionality:

- Hand-over detection algorithm
- Harvest mechanism
- Link awareness

MobiBind was implemented; all the designed functionality was successfully implemented. The tests of MobiBind showed that it did provide enhancements to QoS-parameters: The enhancements include better seamlessness and lower packet loss.

9.2.4 Main Problem Statement

The problem statement of this thesis was as follows:

> *How can a QoS-Aware Remote Binding, implemented in a distributed component, adapt to the varying network conditions to enhance QoS-parameters that perform poorly in IP Based Mobility Management.*

The three sub-problems helped answer the main problem statement. The QoS-parameters we identified was improved by MobiBind, so in that respect we did succeed. It is important to note that even though packet loss and seamlessness was improved, this came at a cost to other QoS-parameters. We identified through testing that RTT increased when seamlessness was provided. So maximising performance on the identified QoS parameters came at a cost of other parameters. This seems to be a tradeoff, maximising QoS performance on all parameters does not seem possible without fixing the underlying cause of degraded QoS.

We did construct a way for MobiBind to support low RTT traffic, providing only reduced packet loss, not seamlessness. However, as Figure 8.2 shows, RTT still increased, in periods, when providing reduced packet loss during hand-overs. This again illustrates the tradeoff between different QoS parameters. Moreover, the assumption that the total QoS remains the same.

Chapter 10

Conclusion and Further Work

In this chapter, the final conclusion of this thesis is presented. The conclusion emphasises the contributions of this thesis. In Section 10.2, possible further work is presented.

10.1 Conclusion

This thesis set out to find a way that mobility IP based networks could support connectivity in the same efficient way as cellular networks do to day. The main operational feature of those networks is that IP based networks does not handle hand-overs in the same efficient manner that cellular networks do.

The QoS-Aware Remote Binding, MobiBind, designed in this thesis addresses this problem by implementing functionality at the middleware layer. This relatively "high level" approach mainly focused on three things; first on giving binding components access to QoS information, secondly the binding knowing when a hand-over occurs,and finally, to handle those hand-overs in a manner that improve the QoS for the binding components.

In this thesis, a way to handle hand-overs called harvesting was proposed and implemented. In the tests of MobiBind it was shown that the harvest mechanism worked well in providing seamless connectivity, i.e., a steady stream of packets despite that the connection was down during hand-over. This connectivity came at an cost of a higher RTT, indicating a tradeoff.

Because MobiBind was designed to be a general remote binding, we identified that some applications does not require the seamless connectivity, but require low RTT. To accommodate this MobiBind allows the binding controlling component to configure MobiBind so that packet loss are minimised, and RTT kept low.

The link awareness feature enables both the middleware and the binding controlling components to extract QoS information about the current link. This is something that can be very useful when planning or making decisions.

The Harvest mechanism proposed in this thesis is a generic way to handle hand-overs, which works well. The harvest mechanism needs to know when a hand-over is occurring; this is the job of the hand-over detection mechanism. The detection algorithm in MobiBind was not stable during all conditions, some ways of improving stability by making MobiBind more self adaptive was discussed. This might be beneficial, but the fact remains that at this high level, and working with assumptions, there is no easy way of knowing when a hand-over is occurring. A better approach would be to enable notifications from lower layers, like the link layer. Another way is for the Mobile IP software to notify MobiBind of the hand-over.

10.2 Further Work

We conclude the presentation of this master thesis work by looking forward. Unaddressed issues as well as addressed issues that can be further researched are presented below.

Further work can be divided in to three main areas, first there are further research spanning from the current design and implementation of MobiBind, secondly, research into hand-over time, and third notifications to MobiBind.

10.2.1 QoS-Aware Remote Binding

MobiBind is a relatively complex piece of software that aims at providing a general, low level, approach to the hand-over problem. Defining application areas, and producing more application specific solutions, could be one way of evolving the field of QoS-Aware Remote Bindings. Research can then be performed to see if the application specific remote bindings will perform better than this thesis general QoS-Aware Remote Binding.

It is difficult to separate a hand-over from a bad link at this high level. This thesis uses one approach, but research can be conducted into making the hand-over detection better. Both better detection of an actual hand-over and separating between actual hand-overs and false positives and negatives.

Another area of possible future research is to take MobiBind as it is now, and try to adjust the parameters. The parameters are today just broadly tuned. To achieve the best performance MobiBind the parameters needs to be given more attention. One possible way of performing this research is to see whether a generic parameter setting can be found, or if a application or maybe link specific approach is better.

MobiBind is aware of hand-overs, despite of this it still looses some packets during a hand-over. Research could be performed to try to eliminate this packet loss.

On a more practical note, better shutdown and startup procedures need to be found. This could be done by integrating MobiBind better in to the middleware, so that the middleware is in part responsible for initialising and stopping it.

MobiBind has several QoS parameters that it can use for self adaption. This could be researched in more detail. One example of better self adaptability could be to use the the links current packet loss rate in hand-over detection, to better separate between hand-overs and false positives and negatives that are caused by bad links..

10.2.2 Hand-over Time

A big problem of hand-overs in IP based networks is that the hand-over takes a long time to complete. The heterogeneity of the IP based network also makes it more difficult to predict the time the hand-over takes to complete. Further research can be focused on making the hand-over shorter in time.

10.2.3 Notifications

The hand-over detection algorithm in this thesis performs okay, but not perfect. A major part of this problem is that there is no way of knowing exactly when a hand-over is occurring. Because of this, a polling strategy like the one in this thesis, has to be used. A much better approach would be a event-notification pattern from a lower software layer. Research into providing this could be one possible research area.

10.2.4 Application Specific High Level Binding

MobiBind is a generic, low level, explicit remote binding. It is relatively small, but has a lot of basic QoS functionality and improvements. One research path could be to make a high level application specific binding that uses MobiBind. As discussed in 6.1, a generic explicit remote binding could probably not be high level, this however, probably changes with a Application Specific Binding. The possibilities of which should be further researched.

Appendix A

Mobile IP Tests

In this appendix the test data from the performance test of Mobile IP are listed. This is the basis for the numbers presented in Chapter 4. 3000 packets were sent during a conversation test, 12000 packets was sent in a streaming test.

A.1 Jitter

The jitter tests was performed on several days and several times. The traffic sent emulated conversational traffic class, more presicely the G711 codec. Each test ran for 60 seconds. Both WLAN and LAN was tested.

Table A.1 gives the results of the jitter LAN test.

Table A.1: LAN Jitter Test

Test number	**Max jitter (ms)**	**Jitter (ms)**
1	10,155	0,106
2	8,889	0,087
3	15,564	0,110
4	14,433	0,119
5	13,631	0,100
6	14,101	0,103
7	16,588	0,114
8	12,066	0,090
9	12,568	0,103
10	13,734	0,097
11	13,063	0,116
12	12,516	0,094
13	15,902	0,107

Continued on next page

Table A.1 – continued from previous page

Test number	Max jitter (ms)	Jitter (ms)
14	16,384	0,105
15	15,577	0,098
16	10,805	0,090
17	17,297	0,104
18	8,492	0,080
19	8,469	0,073
20	11,951	0,105
Average	13,109	0,100
Standard d.	2,73	0,011

Table A.2 presents the jitter tests performed on WLAN. These test ran on three days. WLAN was more unstable and required more testing to give better results.

Table A.2: WLAN Jitter Test

Test number	Max jitter (ms)	Jitter (ms)
1	102,72	2,152
2	110,165	2,121
3	106,52	2,434
4	32,7	1,747
5	112,466	0,504
6	99,719	0,312
7	120,153	0,391
8	115,227	0,312
9	124,361	0,381
10	129,742	0,336
11	122,038	0,629
12	128,484	0,659
13	29,995	0,456
14	109,673	0,620
15	151,283	0,563
16	129,643	0,659
17	113,896	0,600
18	172,492	0,624
19	113,136	0,656
20	102,834	0,499
21	104,763	0,527

Continued on next page

Table A.2 – continued from previous page

Test number	Max jitter (ms)	Jitter (ms)
22	111,233	0,660
23	105,31	0,693
24	111,371	0,518
25	109,763	0,524
26	113,329	0,525
27	34,546	0,624
28	128,385	0,702
29	122,197	0,673
30	115,946	0,651
Average	105,787	0,619
Standard d.	27,596	0,076

A.2 Packet Loss

Packet loss was tested for WLAN and LAN with both the Conversational traffic class and streaming class.

Table A.3 presents the packet loss tests for LAN. The first 30 tests are performed using the conversational traffic class test, the latter 30 tests are performed using streaming traffic. The tests were performed on 3 different days.

Table A.3: LAN Packet Loss Test

Test number	Packet loss
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0

Continued on next page

Table A.3 – continued from previous page

Test number	Packet loss
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	1
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0

Continued on next page

Table A.3 – continued from previous page

Test number	**Packet loss**
24	0
25	0
26	0
27	0
28	0
29	0
30	0
Average	0,1 packets pr test
Average	2,2 x 10^{-6} %
Standard d.	0,129 packets

Table A.4 presents the test results of the packet loss test performed on WLAN. Both conversational and streaming traffic was tested. The first 30 tests of the table presents the conversational tests, the latter 30 presents the streaming tests.

Table A.4: WLAN Packet Loss Test

Test number	**Packet loss**
1	2
2	4
3	3
4	0
5	4
6	6
7	9
8	4
9	5
10	2
11	0
12	1
13	2
14	0
15	0
16	0
17	0
18	1
19	0
20	1

Continued on next page

Table A.4 – continued from previous page

Test number	Packet loss
21	1
22	0
23	0
24	0
25	2
26	0
27	3
28	0
29	1
30	2
1	10
2	18
3	157
4	17
5	135
6	8
7	24
8	30
9	11
10	25
11	1
12	0
13	0
14	1
15	0
16	0
17	0
18	0
19	0
20	1
21	1
22	0
23	0
24	0
25	3
26	2
27	5
28	0
29	0
30	0

Continued on next page

Table A.4 – continued from previous page

Test number	Packet loss
Average	1,1 packets pr test
Average	1,1 x 10^{-3} %
Standard d.	26,664 packets

A.3 Round Trip Time

The Round Trip Time tests ran on several days, testing both conversational traffic class and streaming traffic class, for both WLAN and LAN.

Table A.5 presents the round trip times for the conversational tests. Table A.6 presents the results for the LAN streaming traffic class tests.

Table A.5: LAN RTT Test; Conversation

Test number	RTT (ms)
1	1,33
2	1,22
3	1,22
4	1,21
5	1,21
6	1,21
7	1,22
8	1,23
9	1,23
10	1,23
11	1,33
12	1,23
13	1,28
14	1,23
15	1,22
16	1,31
17	1,25
18	1,24
19	1,28
20	1,26
21	1,30
22	1,29
23	1,28

Continued on next page

Table A.5 – continued from previous page

Test number	RTT (ms)
24	1,30
25	1,29
26	1,27
27	1,30
28	1,28
29	1,29
30	1,29
Average	1,26
Standard d.	0,03

Table A.6: LAN RTT Test; Streaming

Test number	RTT (ms)
1	4,06
2	4,07
3	4,08
4	4,06
5	4,07
6	4,07
7	4,06
8	4,14
9	4,08
10	4,07
11	4,07
12	4,06
13	4,09
14	4,13
15	4,06
16	4,07
17	4,07
18	4,06
19	4,07
20	4,07
21	4,108
22	4,102
23	4,182
24	4,111
25	4,109

Continued on next page

Table A.6 – continued from previous page

Test number	RTT (ms)
26	4,101
27	4,106
28	4,104
29	4,12
30	4,11
Average	4,09
Standard d.	0,03

RTT for WLAN was also tested for both traffic classes. Table A.7 presents the RTT test for WLAN using conversational traffic. Table A.8 present the WLAN tests using streaming traffic.

Table A.7: WLAN RTT Test; Conversation

Test number	RTT (ms)
1	3,83
2	3,91
3	3,96
4	3,88
5	3,80
6	4,04
7	3,76
8	4,22
9	3,90
10	3,88
11	3,93
12	3,84
13	3,98
14	3,82
15	3,88
16	3,90
17	3,85
18	3,86
19	3,83
20	3,90
21	5,13
22	5,02
23	5,34

Continued on next page

Table A.7 – continued from previous page

Test number	RTT (ms)
24	4,72
25	4,02
26	3,87
27	3,93
28	3,86
29	3,96
30	3,96
Average	4,06
Standard d.	0,41

Table A.8: WLAN RTT Test; Streaming

Test number	RTT (ms)
1	10,69
2	11,71
3	16,92
4	11,58
5	11,39
6	11,6
7	12,54
8	11,72
9	11,08
10	10,82
11	9,55
12	9,64
13	9,47
14	9,55
15	9,50
16	9,44
17	9,49
18	9,47
19	9,44
20	9,50
21	9,47
22	9,478
23	9,655
24	9,404
25	9,50

Continued on next page

Table A.8 – continued from previous page

Test number	RTT (ms)
26	9,48
27	9,549
28	9,476
29	9,422
30	9,485
Average	10,33
Standard d.	1,58

A.4 Max Throughput

Max throughput was tested for both traffic with a RTT QoS requirement and for traffic without any QoS reqirements. The no QoS RTT test is presented in Table A.9. This tests uses IPERF to push as much data as possible through the link. Table A.10 uses KGEN to actively search for the saturation point, where the RTT and packet loss is low.

No RTT QoS-Requirement

Table A.9: Max Throughput Without RTT Reqirement

Test number	WLAN (kbit/s)	LAN (kbit/s)
1	5329	8814
2	5472	8791
3	5529	8797
4	5443	8832
5	5525	8817
6	5572	8867
7	5452	8802
8	5527	8760
9	5453	8783
10	5564	8811
11	5477	8801
12	5547	8866
13	5463	8837
14	5548	8793
15	5409	8837
16	5463	8854
17	5511	8797

Continued on next page

Table A.9 – continued from previous page

Test number	WLAN (kbit/s)	LAN (kbit/s)
18	5455	8848
19	5490	8803
20	5459	8808
21	5444	8827
22	5569	8786
23	5417	8802
24	5542	8738
25	5417	8775
26	5577	8793
27	5455	8802
28	5480	8793
29	5511	8754
30	5500	8896
31	5568	8850
32	5515	8757
33	5540	8774
34	5469	8775
35	5430	8790
36	5554	8851
37	5500	8873
38	5455	8772
39	5416	8762
40	5477	8886
41	5510	8808
42	5422	8791
43	5536	8758
44	5483	8876
45	5479	8783
46	5482	8832
47	5440	8762
48	5557	8812
49	5529	8824
50	5563	8829
51	5435	8799
52	5568	8830
53	5486	8892
54	5546	8819
55	5450	8777
56	5527	8817
57	5485	8831

Continued on next page

Table A.9 – continued from previous page

Test number	WLAN (kbit/s)	LAN (kbit/s)
58	5475	8827
59	5544	8756
60	5433	8804
61	5480	8780
62	5449	8792
63	5452	8850
64	5492	8783
65	5510	8826
66	5564	8824
67	5485	8848
68	5425	8811
69	5468	8851
70	5501	8766
71	5572	8833
72	5533	8791
73	5417	8826
74	5441	8765
75	5442	8837
76	5493	8870
77	5385	8812
78	5493	8858
79	5461	8781
80	5474	8803
Average	5488	8811
Standard d.	51	36

Table A.10: Max Throughput With RTT Requirements

Test number	LAN (kbit/s)	WLAN (kbit/s)
1	8647	3825
2	8701	3693
3	8745	3740
4	8802	3841
5	8489	3491
6	8886	3519
7	8899	3665
8	8759	3828
9	8640	3700

Continued on next page

Table A.10 – continued from previous page

Test number	LAN (kbit/s)	WLAN (kbit/s)
10	8715	3566
11	8913	3860
12	8734	3627
13	8464	3647
14	8882	3765
15	8791	3756
16	8996	3588
17	8897	3760
18	8956	3796
19	8785	3817
20	8551	3797
21	8652	3834
22	8731	3870
23	8730	3643
24	8949	3750
25	8910	3792
26	8636	3746
27	8460	3811
28	8600	3773
29	9002	3824
30	9049	3759
31	8901	3810
32	8759	3813
33	8907	3723
34	8772	3718
35	8669	3755
36	8866	3780
37	8732	3775
38	8880	3801
39	8742	3812
40	8679	3772
41	8641	3583
42	8690	3858
43	8732	3770
44	8691	3721
45	8595	3796
46	8936	3752
47	8988	3840
48	8894	3760
49	9023	3678

Continued on next page

Table A.10 – continued from previous page

Test number	LAN (kbit/s)	WLAN (kbit/s)
50	8633	3863
51	8905	3697
52	8656	3727
53	8517	3765
54	8741	3696
55	8747	3640
56	8869	3755
57	8691	3825
58	8869	3778
59	8745	3789
60	8597	3864
61	8751	3818
62	8786	3751
63	8937	3741
64	8918	3826
65	8784	3762
66	8793	3789
67	8618	3807
68	8748	3747
69	8786	3817
70	8877	3855
71	8872	3676
72	8921	3814
73	8885	3830
74	8920	3665
75	8887	3697
76	8747	3728
77	8908	3820
78	8777	3766
79	8896	3863
80	8922	3796
81	8805	3823
82	8819	3783
83	8636	3768
84	8854	3757
85	8915	3808
86	8998	3835
87	8789	3847
88	8996	3912
89	8708	3703

Continued on next page

Table A.10 – continued from previous page

Test number	LAN (kbit/s)	WLAN (kbit/s)
90	9008	3847
91	8466	3714
92	8887	3700
93	8739	3811
94	8834	3870
95	8644	3771
96	8715	3651
97	8883	3909
98	8858	3950
99	8758	3895
100	9000	3912
Average	8791	3768
Standard d.	136	83

A.5 Hand-over Time

The hand-over test is presented in Table A.11. This test finds out how long the connection is down during a hand-over.

Table A.11: Hand-over Time Test

Test number	LAN -> WLAN (ms)	WLAN -> LAN (ms)	WLAN -> WLAN (ms)
1	960	980	1260
2	960	980	1240
3	960	980	1220
4	960	980	1200
5	980	980	1200
6	980	980	1200
7	960	980	1220
8	960	960	1280
9	980	980	1280
10	1000	980	1220
11	980	980	1260
12	1000	980	1240
13	960	980	1220
14	960	980	1180
15	960	980	1200

Continued on next page

Table A.11 – continued from previous page

Test number	LAN -> WLAN (ms)	WLAN -> LAN (ms)	WLAN -> WLAN (ms)
16	960	980	1200
17	960	960	1220
18	960	980	1280
19	1000	980	1280
20	960	960	1220
21	1000	980	1260
22	1000	980	1240
23	1000	980	1220
24	960	980	1180
25	980	960	1200
26	1000	980	1200
27	960	980	1220
28	960	980	1280
29	960	960	1220
30	1000	960	1220
31	1000	980	1260
32	980	980	1240
33	960	980	1220
34	960	960	1180
35	960	980	1200
36	1000	980	1200
37	980	980	1220
38	1000	980	1280
39	980	960	1280
40	960	980	1220
41	1000	960	1260
42	980	980	1240
43	980	980	1220
44	1000	960	1180
45	1000	980	1200
46	1000	980	1200
47	960	980	1220
48	960	960	1280
49	960	980	1280
50	960	980	1220
51	1000	980	1240
Average	976	976	1229
Stanard d.	18	8	31

Appendix B

Enclosed CD

The enclosed CD contains the following directories:

MobiBind contains the source code of the MobiBind implementation.

KGen contains the source code of KGen, that was used during the Mobile IP testing.

KGen Modified contains the KGen source code, used when testing MobiBind. It is modified to allow it to connect to and use MobiBind.

Documents contain an electronic version of this thesis.

The CD does not contain a runnable example because MobiBind has a dependence to Mobil IP. This dependency was realised with software that is not freely distributed, so it could not be enclosed on the CD, thereby making it impossible to provide a runnable example.

The CD is found
on the very last page

Bibliography

[1] 3GPP. *Universal Mobile Telecommunications System (UMTS); Quality of Service (QoS) concepts and architecture*, ts 23.107 version 5.13.0 release 5 edition.

[2] S. Amundsen, K. Lund, F. Eliassen, and R. Staehli. Qua: Platform-managed qos for components architectures. In *Proceedings of Norwegian Informatics Conference (NIK)*. Tapir, 2004.

[3] Geert Awater and Jan Kruys. Wirleless atm - an overview. *Mobile Notworks and Applications*, 1:235–243, 1996.

[4] Gordon Blair and Jean-Bernard Stefani. *Open Distributed Processing and Multimedia*. Addison-Wesley, 1997.

[5] Cambridge: Architecture Projects Management Ltd. *Advanced Networked Systems Architecture ANSA Reference Manual*, 01 .oo edition.

[6] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems - Consepts and design*. Pearson, third edition, 2001.

[7] Theory of adaptation - specification of the madam core architecture and middleware services. Technical report, EU Project - Sixth Framework Programme, 2005.

[8] M.A. de Miguel. Solutions to make java-rmi time predictable. *Object-Oriented Real-Time Distributed Computing, 2001. ISORC*, pages 379–386, May 2001.

[9] Ping Liu Jina Mao Tomonori Yamane Luiz Claudio MagalhÃ£es Roy H. Campbell Fabio Kon, Manuel RomÃ¡n. *Dynamic Configuration with the dynamicTAO Reflective ORB*, volume 1795. Lecture Notes in Computer Science, Jan 2000.

[10] P Ferguson and D Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. Rfc 2827, Cisco, 2000.

[11] Tom Fitzpatrick, Gordon Blair, Geoff Coulson, Nigel Davies, and Robin P. Supporting adaptive multimedia applications through open bindings. *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDS '98)*, 1998.

[12] Matthew S. Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[13] Geoff Coulson Gordon Blair. The design and implementation of openorb 2. *IEEE Distributed Systems Online*, 2(6), 2001.

[14] Paul Grace, Gordon S. Blair, and Sam Samuel. A reflective framework for discovery and interaction in heterogeneous mobile environments. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(1):2–14, January 2005.

[15] Object Management Group. Corba components. Technical Report 02-06-65, OMG, 2002.

[16] Poul E. Heegaard. Gensyn - a java based generator of synthetic internet traffic linking user behaviour models to real network protocols. ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management, sep 2000.

[17] http://www.iona.ie. The orbix architecture. Technical report, IONA Technologies, 1995.

[18] Juha Korhonen. *Introduction to 3G mobile communications*. Artech House, 2003.

[19] Harju J. Koucheryavy Y., Moltchanov D. Performance evaluation of live video streaming service in 802.11b wlan environment under different load conditions. In *MIPS 2003*, pages 30–41, 2003.

[20] Tianbo Kuang and Carey Williamson. A measurement study of RealMedia streamin traffic. In *SPIE ITCOM*, pages 68–79, 2002.

[21] DeMichiel LG. Enterprise javabeanstm specification. Technical Report 2.1.2002, Sun Microsystems Inc., 2002.

[22] Geoff Coulson Nikos Parlavantzas Michael Clarke, Gordon S. Blair. An efficient component model for the construction of adaptive middleware. In *Lecture Notes in Computer Science*, volume 2218, page 160. Jan 2001.

[23] Microsoft. Overview of the .net framework white paper. Technical report, Micrisoft, 2001.

[24] R. Cunningham M. Haahr and V. Cahill. Supporting corba applications in a mobile environment. In *Proc. 5th Int. Conf Mobile Computing and Networking*, pages 36–47. ACM Press, 1999.

[25] M. Kounavis O. Angin, A. Campbell and R. Liao. The mobiware toolkit: Programmable support for adaptive mobile netwoking. *Personal Communications agazine, Special Issue on Adapting to Network and Client Variability.*, Aug 1998.

[26] Charles E. Perkins and David B. Johnson. Mobility support in ipv6. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 27–37, New York, NY, USA, 1996. ACM Press.

[27] Charles E. Perkins and David B. Johnson. Mobility Support in IPv6. In *Mobile Computing and Networking*, pages 27–37, 1996.

[28] R. Ramjee et al. Ip micro-mobility support using hawaii. Internet draft, July 2000.

[29] Espen Sagen. End-to-end performance during mobile ip handovers. Master's thesis, University of Oslo, 2003.

[30] Aladdin Saleh. Mobile ip performance and interworking architecture in 802.11 wlan/cdma2000 networks. In *CNSR*, 2004.

[31] REED D. P. SALTZER, J. H. and D. D. CLARK. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 1984.

[32] Michael Shuldman. Tools developed for network profiling and traffic analysis. Master's thesis, UiO, 2004.

[33] James D. Solomon. *Mobile IP, The internet unplugged*. Prentice Hall, 1998.

[34] W. R. Stevens. *TCP/IP Illustrated*, volume 3 of *Professional Computing Series*, chapter TCP for Transactions, pages 1–158. Addison-Wesley, Reading, MA, 1999.

[35] Cisco Systems. *Cisco 3200 Series Access Router*, configuration guide edition, August 2003.

[36] Clemens Szyperski, editor. *WCOP'96 Summary in ECOOP'96 Workshop Reader*. dpunkt Verlag, isbn 3-920993-67-5 edition, 1997.

[37] Birdstep Technology. *Birdstep Mobile IP Client Diagnostics Guide*, universal edition release 2.0.4 edition, June 2003.

[38] Birdstep Technology. *Birdstep Mobile IP Client Release Notes*, universal edition release 2.0.4 edition, June 2003.

[39] Birdstep Technology. *Birdstep Mobile IP Client User Guide*, universal edition release 2.0.4 edition, June 2003.

[40] Birdstep Techology. *Birdstep Mobile IP Client Administrator's Guide*, universal edition release 2.0.4 edition, June 2003.

[41] NLANR Tools. http://dast.nlanr.net/npmt/, June 2005.

[42] Ledoux T. Opencorba: a reective open broker. *Reflection'99*, 1616:197–214, 1999.

[43] A. G. Valko. Cellular ip: A new approach to internet host mobility. *Comp Commun Review*, 29(1):42–49, 1999.

[44] Jon-Olov Vatn and Gerald Q. Maguire Jr. The effect of using co-located care-of addresses on macro handover latency. In *14th Nordic Tele-traffic Seminar*, aug 1998.

[45] Vogel. Distributed multimedia and qos: A survey. *IEEE Multimedia 1995*, 1995.

[46] Rfc3344 - ip mobility support for ipv4, 2002.

[47] http://www.item.ntnu.no/ poulh/gensyn/gensyn.html, July 2005.