

TECHNISCHE UNIVERSITEIT EINDHOVEN  
Department of Mathematics and Computer Science

MASTER'S THESIS

Interface Selection Layer  
Improving QoS using Interface Pair Selection

by  
C.F. van Antwerpen

Supervisors:  
Dr. J.J. Lukkien  
Dr. ir. P.H.F.M. Verhoeven  
Dr. ir. M.A. van Hartkamp

Eindhoven, April 2005

<b>ABSTRACT .....</b>	<b>4</b>
<b>1 INTRODUCTION.....</b>	<b>5</b>
1.1 OBJECTIVE .....	5
1.2 PROBLEM DESCRIPTION .....	5
1.3 OVERVIEW .....	5
1.4 INTENDED AUDIENCE .....	6
1.5 LIST OF DEFINITIONS .....	6
<b>2 ANALYSIS .....</b>	<b>7</b>
2.1 ENVIRONMENT AND ASSUMPTIONS .....	7
2.2 FOCUS .....	7
2.3 OTHER RESEARCH .....	7
2.3.1 <i>Protocol-based topology detection</i> .....	8
2.3.2 <i>Measurement-based topology detection</i> .....	8
2.4 USE CASES .....	10
2.5 REQUIREMENTS .....	11
2.5.1 <i>Requirements of the conceptual solution</i> .....	11
2.5.2 <i>Requirements of the implementation</i> .....	11
2.6 SOLUTION .....	12
<b>3 MODELS &amp; ALGORITHMS .....</b>	<b>15</b>
3.1 MODELS .....	15
3.1.1 <i>Traffic model</i> .....	15
3.1.2 <i>Path model</i> .....	16
3.2 MEASURING PATH PROPERTIES .....	18
3.2.1 <i>Measuring latency</i> .....	18
3.2.2 <i>Available bandwidth</i> .....	21
3.3 ESTIMATION OF PATH PROPERTIES .....	22
3.3.1 <i>Test network</i> .....	22
3.3.2 <i>Latency</i> .....	23
3.3.3 <i>Available bandwidth</i> .....	34
3.3.4 <i>Conclusion</i> .....	35
<b>4 ARCHITECTURE .....</b>	<b>36</b>
4.1 BERKELEY SOCKET INTERFACE (BSI) .....	36
4.2 HIGH-LEVEL MODEL .....	37
4.3 HANDOVER .....	38
4.4 MAIN COMPONENTS .....	41
4.4.1 <i>Pair Selection Layer (PSL)</i> .....	41
4.4.2 <i>Main Controller (MC)</i> .....	41
4.4.3 <i>Device Handler (DH)</i> .....	41
4.4.4 <i>Property Collector (PC)</i> .....	42
4.4.5 <i>Socket Abstraction Layer (SAL)</i> .....	42
4.4.6 <i>Interface Abstraction Layer (IAL)</i> .....	42
4.5 EXTERNAL COMMUNICATION .....	43
4.5.1 <i>Detection of ISL devices</i> .....	43
4.5.2 <i>Gathering interface information</i> .....	44
4.5.3 <i>Determination of path properties</i> .....	44
4.5.4 <i>Selection of interface pair</i> .....	45
4.5.5 <i>Sending and receiving data</i> .....	46
4.5.6 <i>Synchronisation between ISL devices</i> .....	47
4.6 INTERNAL COMMUNICATION .....	48
4.6.1 <i>Initialization</i> .....	49
4.6.2 <i>Handling messages</i> .....	50
4.6.3 <i>Property determination</i> .....	52
4.6.4 <i>Interface pair selection</i> .....	53
4.6.5 <i>Sending data</i> .....	54

4.6.6	<i>Receiving data</i> .....	55
4.7	CONCLUSION .....	56
<b>5</b>	<b>IMPLEMENTATION</b> .....	<b>58</b>
5.1	LINUX .....	58
5.2	FILE DESCRIPTORS AND SOCKETS .....	58
5.3	DETERMINATION OF THE BEST INTERFACE PAIR .....	59
5.4	COMPONENTS.....	59
5.4.1	<i>Pair Selection Layer (PSL)</i> .....	60
5.4.2	<i>Main Controller (MC)</i> .....	60
5.4.3	<i>Device Handler (DH)</i> .....	61
5.4.4	<i>Property Collector (PC)</i> .....	62
5.4.5	<i>Socket Abstraction Layer (SAL)</i> .....	62
5.4.6	<i>Interface Abstraction Layer (IAL)</i> .....	62
5.5	USING ISL; REPLACING STANDARD BSI FUNCTIONS BY OTHERS WITH THE SAME NAME .....	63
<b>6</b>	<b>EVALUATION</b> .....	<b>65</b>
6.1	INTERFACE PAIR SELECTION (STARTUP TEST) .....	65
6.1.1	<i>Functionality</i> .....	65
6.1.2	<i>Performance</i> .....	65
6.2	SWITCHING INTERFACE PAIRS (CHANGING TEST).....	66
6.2.1	<i>Functionality</i> .....	66
6.2.2	<i>Performance</i> .....	66
6.3	LEGACY DEVICES .....	66
<b>7</b>	<b>FUTURE WORK</b> .....	<b>67</b>
<b>8</b>	<b>CONCLUSION</b> .....	<b>68</b>
<b>9</b>	<b>ACKNOWLEDGEMENTS</b> .....	<b>69</b>
<b>10</b>	<b>REFERENCES</b> .....	<b>70</b>

## Abstract

The Internet is getting larger and larger. Every day more devices get connected to it. But the same goes for the local networks at people's home. There are however two major differences between the infrastructure of the Internet and of the home network. In a home network devices are added but not always replaced and it consists of mainly low-cost unmanaged devices. The result is a chaotic network with congestion problems.

The devices in the home network have multiple interfaces and offer all kinds of services, so the number of connections and the number of paths between devices (peer-to-peer connections between the interfaces of devices) are increasing. Current devices choose a path at random when they want to communicate with another device. To avoid congestion we want to choose which path the device must use. But to make a good choice we need information about the current state of each of the paths to compare the paths. We present a solution to offer this path selection functionality to a device by collecting information about the different paths and using this information to choose the best path.

# 1 Introduction

## 1.1 Objective

This document presents the results of my master's project "Interface Selection Layer" at Philips Research Laboratories together with interesting topics for future research.

## 1.2 Problem description

The Internet is getting larger and larger. Every day more devices get connected to it. But the same goes for the local networks at people's home. There are however two major differences between the infrastructure of cooperate networks (including the network of ISP's and universities) and of the home network. In a home network:

- Devices are added but not always replaced (consumers connect new devices to the current infrastructure and not upgrade old devices with new devices). Devices are replaced when they are broken, but not to let them cooperate with a newly added device.
- Mostly low-cost devices are added: most devices are unmanaged (consumers do not pay extra money for things they do not value, e.g. technical differences between an unmanaged switch and a managed switch).

People just want to connect new devices without performing lots of configuration steps (Plug and play), which results into a chaotic network. If all the devices in the (chaotic) network want to transfer large amounts of information (e.g. video streams), then it is possible that a part of the infrastructure is used by many devices, which in turn can lead to congestion. When displaying video streams, the congestion causes the video to stutter and artefacts may appear. Audio streams are also disturbed and data streams can suffer from a major slow down. Congestion is not wanted, so how can we minimize the chances that congestion appears?

The home network consists of a very diverse infrastructure, to which all kinds of unmanaged devices are connected. These devices have multiple interfaces and offer all kinds of services, so the number of connections and the number of paths between devices (peer-to-peer connections between the interfaces of devices) are increasing. Current devices choose a path (consisting of an interface pair: one interface at the sender and one at the receiver) at random (mostly the last known working path is chosen) when they want to communicate with another devices. To avoid congestion we want to choose which path the device must use. But to make a good choice we need information about the current state of each of the paths to compare the paths. In this document a solution is presented to try to minimize the problem of congestion in a home network by collecting information about the paths, using this information to compare the paths and choosing a path.

## 1.3 Overview

In chapter 2 the congestion problem will be discussed in more detail and a solution is presented to tackle the problem. In chapter 3 the models used to gain a better understanding of the home networking environment are given together with the algorithms that use these models and their validation. Chapter 4 presents the architecture of solution and chapter 5 gives some implementation details. In chapter 6

an evaluation is given. Further research topics and additions to the solution together with the conclusion of the project are given in the remaining chapters.

## 1.4 Intended audience

This document is meant for people that have reasonable background knowledge about computer networks and the terms used in this field. Two good books in this field are “Computer Networks” written by Andrew S. Tanenbaum [1] and the book “Interconnections” written by Radia Perlman [2].

## 1.5 List of definitions

Below is a list of terms with their definitions that are used throughout this document.

Term	Definition
Arrival time	The time that a packet arrives at the sink
Available Bandwidth	Maximum number of bits per second that can still be transferred over a connection
Bandwidth	Number of bits per second
Latency	Time that elapses between start of transmission at source and start of receipt at sink
Legacy device	A device without the Interface Selection Layer
Link	Physical connection between two devices
Loss	The amount of packets that gets lost during a certain interval.
Maximum Bandwidth	Maximum number of bits per second that can be transferred over a connection
Path	One or more physical links that form a connection between two interfaces. There is only one path between two interfaces, but there can be multiple paths between devices.
Serialisation delay	Time needed for a packet to be converted to signals that can be transferred over a link
Sink	The device that receives data
Source	The device that sends data
Transmission time	Time that elapses between start of transmission at source and end of receipt at sink: $transmissiontime = latency + \frac{size}{bandwidth}$
Virtual address	An IP address created and assigned to an existing interface by the Interface Selection Layer to make it possible to seamlessly handover streams.

## **2 Analysis**

In this chapter the congestion problem is analysed in further detail. First the home networking environment is discussed and on which problem this project is focussed. Other research that is done in this field is discussed and a proposal is done to solve the problem. This proposal is explored in the rest of the document.

### **2.1 Environment and assumptions**

Nowadays most homes have at least one personal computer with an Internet connection. Some homes already have two computers that are connected to each other so that they can share resources (e.g. a printer, or an Internet connection). In the future, more and more devices will be connected to each other: besides computers also televisions, stereo's and DVD players. All these devices will be connected to each other using the already available infrastructure devices: simple (and low-cost) relay devices. In general, devices are added to the network and not always removed. We assume that all devices use IP to communicate with each other. The basic IP protocol stack uses best-effort QoS to transfer data, but this is not sufficient for guaranteeing e.g. video streaming without interruptions. So if more and more devices get added, the chance that congestion occurs gets higher, this in turn can have a negative effect on the QoS delivered by the network.

All data in the home networking environment is sent using peer-to-peer connections. This means that all data between devices is sent from interface to interface and not to a broadcast address. Therefore, we will not consider multicast in this document. We also assume that there are no routers within the home network (only one router that connects the home network to the internet), so everything is bridged.

### **2.2 Focus**

The focus of this project is on the improvement of the QoS in the home network: how can we improve the QoS without extensive changes to the already available infrastructure? We focus on the topology of the home network: how is everything connected and what are the properties of the paths between the devices. If the properties of the paths between devices are known, we can compare the paths and offer a better path to the application. By not just choosing a path at random, devices have the possibility to choose the best path and thus avoid hick-ups in audio and video streams.

### **2.3 Other research**

At first we started out with the problem of topology detection. We thought that if we know the topology of a home network, we can use this information to offer feedback to the user about the status of the network (e.g. location of bottlenecks) and to offer better QoS (reroute traffic over better paths). The research on topology detection is divided into two areas: one area focuses on protocol-based and the other on measurement-base topology detection.

If looked at the home network, it is very difficult to get the topology information: to get complete knowledge about the topology it is necessary to detect all relay devices. In a home networking environment the idea about relay devices is that they are

completely transparent, so this makes detecting them very hard. Furthermore there is not much use for the topology information because a path between two interfaces cannot be changed. So actually we are not interested in the complete topology but only in the properties of the (static) paths between devices.

Because at first we started with topology detection and switched later on to measuring path properties, we present in this chapter both research areas. Especially because both areas have some parts in common: some properties of paths can be used to determine the topology of the path (e.g. number of relay-devices).

Most research in the field of topology detection and path property determination focuses on the Internet and not on a home networking environment. These situations differ: in the Internet routers are used to relay traffic between end-points and distances are large, in the home situation bridges are used to relay traffic and distances are in the order of a couple of metres. Because distances in the Internet are much larger, packets need more time (mille seconds instead of micro seconds) to get from one machine to another. So in the Internet latency or transmission time differences are easier to measure because the difference between two paths is not necessarily in the order of a couple of micro seconds.

Because of the difference in relay devices and in distances between devices most research that focuses on the Internet cannot be used in the home networking environment.

### **2.3.1 Protocol-based topology detection**

The aim of the research done in this field is to use already available protocols in a network to discover the topology. Devices in the network already use protocols to communicate with each other. In the literature there are methods that use these protocols to get information about the network topology. Most of these methods are based on the fact that relay devices are not completely transparent and offer management capabilities like SNMP [5]. This is mostly true in a corporate environment or in the Internet, but not in a home networking environment where these relay devices offer no management capabilities. Furthermore, if they do offer management capabilities, mostly this involves extra configuration that can only be done by the more experienced users.

Some research is focused on analyzing traffic by looking at all the different packets that are on the network [6]. However, as seen in [7] this method has many problems; the most important problems are:

- Only detection possible of devices that actually sent traffic
- Only traffic from and to own device (and broadcasts) can be seen in switched networks

Existing solutions analyse also the protocols between routers (e.g. Spanning Tree Protocol or Routing Information Protocol), which makes this research specific for Internet and not applicable in a home networking environment because, as stated before, the relay devices in a home network are completely transparent and do not act as a router but as a bridge.

### **2.3.2 Measurement-based topology detection**

The research done in this field focuses on the detection of shared paths and the measured properties of the paths in order to know which devices share which paths so

that it is possible to derive the topology of the network. In contrary to the research that is protocol based, it does not concern the detection of end-point devices but it concerns the detection of relay-devices on the path [8]. To detect shared paths it is necessary to detect a shared property of two paths: e.g. shared jitter, shared network delay, or shared packet loss.

If we want to measure loss, it is necessary that there is some loss in the network. Depending on the used technology loss can only occur when a link is heavily congested (most wired links) or occurs most of the time (wireless links). The problem with wired networks is that in order to measure the loss, we need to congest the link, which is not wanted because we do not want to interfere with the current data streams. The problem with wireless links is that extra probe traffic can cause more loss than the original situation so again there is interference caused by the probing traffic. It is possible to keep track of all the sent and received packets, but it can take a very long time before information about loss is known (especially in wired networks, because congestion is required).

Two other properties that we want to measure are bandwidth and latency of the links and paths between devices. There are mainly two techniques used to determine these properties: single packet and packet pair techniques. Other techniques are a combination of these two techniques.

Single packet techniques focus on the estimation of link bandwidth and not on the path bandwidths. Each link has a different latency and slower links will take longer to transmit a packet than faster links. Using the formula  $b = \frac{s}{t_{transmission} - t_{latency}}$ , (1) the

bandwidth  $b$  of a link can be calculated if the packet size  $s$ , the link latency  $t_{latency}$  and the transmission time  $t_{transmission}$  are known. The link latency does not depend on the packet size or the link bandwidth: it has a fixed value for a link. If the transmission times of multiple packets with variable sizes are plotted, the bandwidth can be calculated by taking the inverse of the slope of the graph.

To determine the link bandwidth of each link of a path, single packet techniques use the time to live (TTL) field of an IP packet. This value is decremented at every router and if it reaches zero, the router must return an ICMP TTL expired error message. Changing the TTL for each packet, gives a set of round trip times. There are some problems with this method [9], but the main problem for a home networking environment is that only routers change the TTL and bridges do not. Therefore, it is not possible to use such a single packet pair technique to get the bandwidth of each link in a home network.

Packet pair techniques focus on the estimation of the available bandwidth of the path and not on the estimation of the available bandwidth of a link. Each packet on a link experiences a serialisation delay due to the bandwidth of the link. Packet pair techniques measure the difference in arrival times of two packets sent immediately after each other. It is assumed that each device on a path use store-and-forward to forward packets, this means that a packet must first be completely received before it can be sent to the next device on the path. The slowest link is responsible for the spacing between the two packets because the first packet was fully received (and forwarded) while the second packet is still being received. If both packets have equal size  $s$  and the arrival time difference  $d$  between the two packets is known, then the

bandwidth  $b$  can be calculated using  $b = \frac{s}{d}$ , (2). The difference with formula (1) is

that the latency does not have any influence here because both packets have the same latency; this means that the delay between the two packets cancels out the latency, meaning that  $d$  equals  $t_{transmission} - t_{latency}$ :

$$\begin{aligned}
d &= t_{arrival\_time\_p1} - t_{arrival\_time\_p0} \\
&\equiv \{def.arrival\_time\} \\
d &= \left( t_{offset} + t_{latency} + \frac{s_{p0}}{b} + \frac{s_{p1}}{b} \right) - \left( t_{offset} + t_{latency} + \frac{s_{p0}}{b} \right) \\
&\equiv \{s_{p0} == s_{p1}\} \\
d &= \frac{s}{b} \\
&\equiv \{calc.\} \\
d &= t_{latency} + \frac{s}{b} - t_{latency} \\
&\equiv \left\{ def.t_{transmission} = t_{latency} + \frac{s}{b} \right\} \\
d &= t_{transmission} - t_{latency}
\end{aligned}$$

Both techniques have the disadvantage that measurement errors occur because of other traffic on the link or path (cross traffic). Research that is done to filter out this cross traffic focuses on the assumption that cross traffic occurs random. For single packet techniques this means that it is assumed that cross traffic will only increase delays, so if enough packets are sent, one packet will have the minimum delay. This is discussed in [10] and [11]. For packet pair techniques, the most common bandwidth measurement is the actual bandwidth. In [12] statistical methods are discussed to filter out faulty results, but [13] proves that this is not enough. The latter developed a bandwidth estimation methodology, which is implemented in the tool `pathrate`. The same people involved with `pathrate`, also developed the tool `pathload` [15] for determining the available bandwidth of a path. Another tool that uses the same ideas as `pathload` is `pathChirp` [14]. Both tools are discussed in section 3.2.2 and compared in section 3.3.3.

## 2.4 Use Cases

Below we give a few real life examples of situations where a device can offer better performance when the device has the ability to choose a better path when all the properties of the possible paths are available.

- A user has a PDA with two interfaces: a wired and wireless interface. When the user is walking around the house, the PDA uses its wireless interface to communicate with the home network. In general wired connections perform better than wireless connections. So when the user puts the PDA in a cradle, we prefer that all connections that use the wireless interface are transferred to the wired interface without the loss of any of the connections.
- A user has two displays (e.g. a CRT monitor and a plasma television) and wants to watch a video stream from the Internet on one display, and a video stream from a DVD player the other display. Of course the user wants both

display devices to display the video streams without any hick-ups or artefacts. If one (or both) of the display devices have multiple interfaces, possible hick-ups may be prevented if the video streams were sent over different paths to the display devices.

## 2.5 Requirements

If a device wants to offer better network performance to the application, it must first determine the properties of the available paths and then it can use this information to choose the best path for communication with another device. Because in most networks the wired and wireless parts belong to different subnets, we do require that our solution works when a device belongs to multiple subnets. The solution has two kinds of requirements: requirements concerning the solution itself, and requirements concerning the implementation of the solution.

### 2.5.1 Requirements of the conceptual solution

1. A device **must** have the means to select or change an interface pair on the basis of measurement data.
2. The path that meets the application's requirements **must** be chosen and not a worse path.
3. Communication that takes place to determine the best interface pair **must** limit the influence on other traffic on the network to a minimum.
4. Legacy devices **must** be able to work with the new devices and vice versa.
5. It **must** work without modifications of the relay devices.
6. The selection and changing of interface pairs **must** be completely transparent for applications.

### 2.5.2 Requirements of the implementation

7. It **must** use IPv4 for communication with other devices.
8. It **must** run on Linux.
9. It **must** be programmed in C or C++.
10. It **must** not change the application's traffic because of other (future) standards (this means that we are not allowed to encapsulate the traffic in custom made packets or set up a custom tunnel between devices).
11. An application **must** be able to start sending data without a large delay (the response time of the application must remain small).
12. Current applications **must** not need (major) adjustments.
13. It **must** be possible for an application to choose to use the enhancement: the enhancement is optional for the application.
14. It **must** also work in the situation where a device belongs to multiple broadcast domains.
15. It **should** be possible to synchronize all devices to guarantee correct selection of a path.
16. It **should** be extensible to give new applications access to more "advanced functionality".

17. It **should** enforce the requested bit rate on the application (an application is not allowed to send a data stream of a higher bit rate than the application has asked for).

## 2.6 Solution

It seems that topology detection will not offer improved QoS in a home networking environment because the topologies are very small. Furthermore, the paths in the network cannot be changed. This implies that the knowledge of the topology also does not give many opportunities to improve QoS. The only positive effect it can have is that every device has knowledge about the shared paths, but it is very hard to get this knowledge as previous mentioned research has shown.

The proposal is not to focus on topology knowledge, but to look at the possible paths between two end-point devices. Each device has one or more interfaces. These interfaces might be connected directly or indirectly to the interfaces of another device; in theory there could be multiple paths between two interfaces, but in practise each interface combination forms only one path between the two devices. Each path has certain properties (e.g. latency, and bandwidth). If all properties of the paths between two devices are known, the best path can be chosen. These path properties need to be measured and there must be communication between the source and sink so that they both know which path is chosen.

Current APIs focus on connecting devices rather than interfaces. As a result only a single interface is used for a connection and it is not possible to switch between interfaces during a connection. To make it possible to switch between interfaces, we want to make a layer between the application and the operating system so this determination of properties and the switching of interface pairs can be done transparent to the application without diving into the source of the operating system. This layer is called the Interface Selection Layer (or ISL for short). It offers a replacement of the Berkeley Socket Interface (BSI: the API used by UNIX applications to communicate between devices) to the application with the same interface. This way the application needs not be modified. Furthermore all data that is sent and received by the application goes through this layer, which makes it possible to have full control of the sending and receiving process (making it possible to add bit rate control or packet filtering in the future). This makes it possible to use other means to send and receive data besides using only the BSI (see Figure 1). An alternative would be to put the ISL into the OS, but then the ISL is hardly portable to other Operating Systems (or to another kernel version) and it is harder to develop and debug.

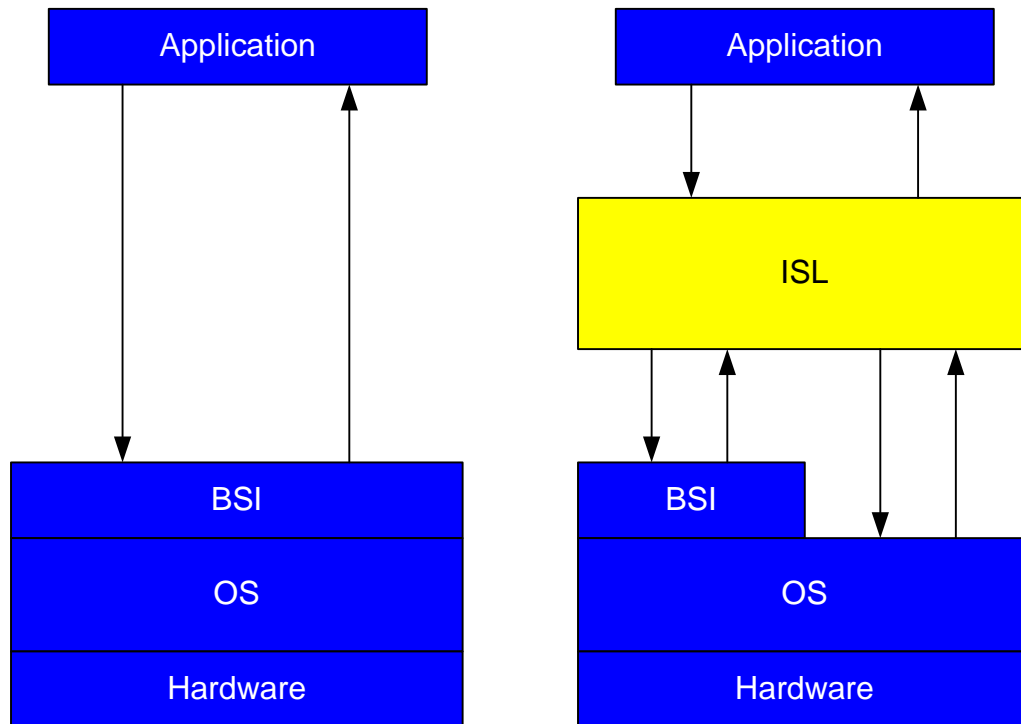


Figure 1. Overview of the network communication of an application with and without ISL.

The greatest challenge was to find methods to determine the properties of each path. A large part of this document covers this. A second problem is how to let devices work together to determine properties and announce the chosen pairs to each other, these are however more implementation issues. If looked at the layers given in Figure 1 the application is not modified because the ISL offers the same interface as the BSI. The ISL determines the available paths using the OS to get information about the own interfaces and the BSI to communicate with other devices to get information about their interfaces. Comparing the properties of the available paths, the ISL chooses the best path and uses the OS to change the local interface. The BSI is used to communicate the choice to the other device. An overview of the different tasks is given in Figure 2. Our main focus will be on the methods used for property determination and on the architecture of the ISL, which we have implemented and tested.

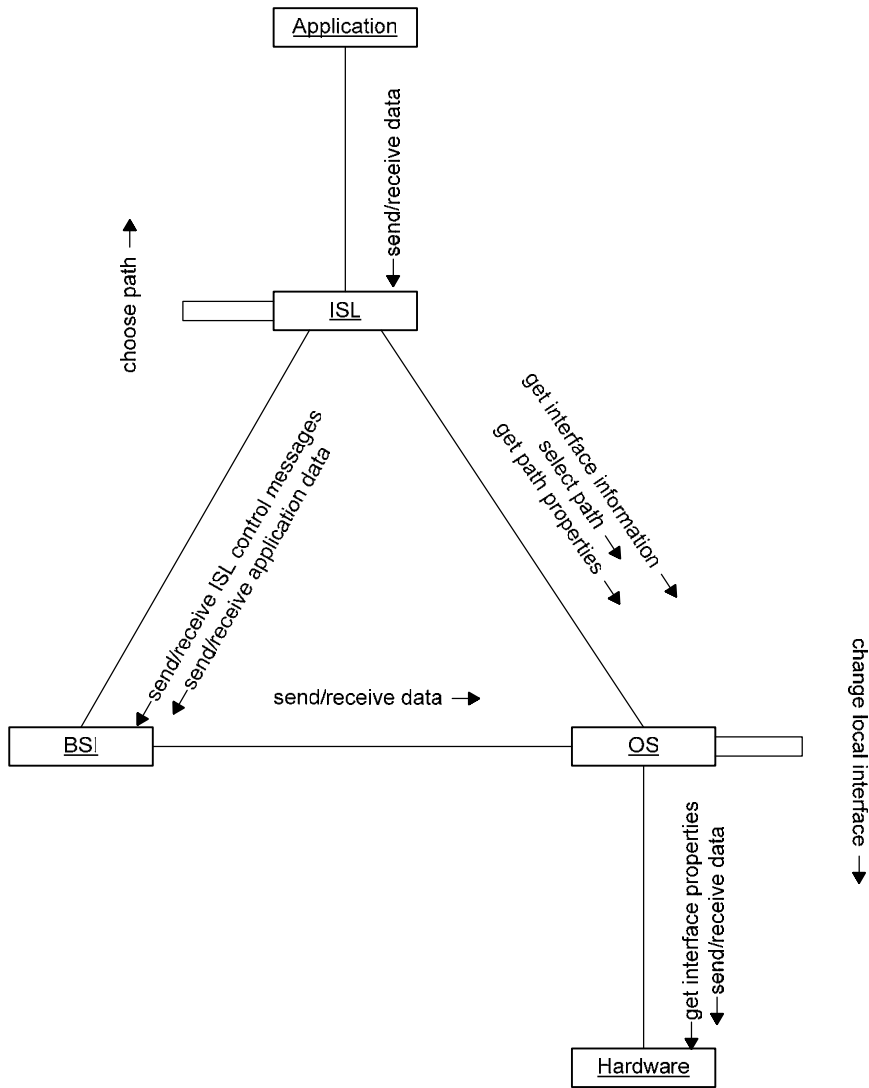


Figure 2. Collaboration between the different layers

### 3 Models & algorithms

#### 3.1 Models

In this section we discuss different models for the traffic in a network and choose one. Furthermore a model for a path is given together with a model of the different path properties.

##### 3.1.1 Traffic model

There are two models to see traffic in a network:

- Packetized model
- Fluid model

The packetized model says that packets of different data streams interleave with each other. The fluid model sees the traffic as different data streams that travel parallel to each other. In the packetized model, packets do not finish later than in the fluid model (see Figure 3).

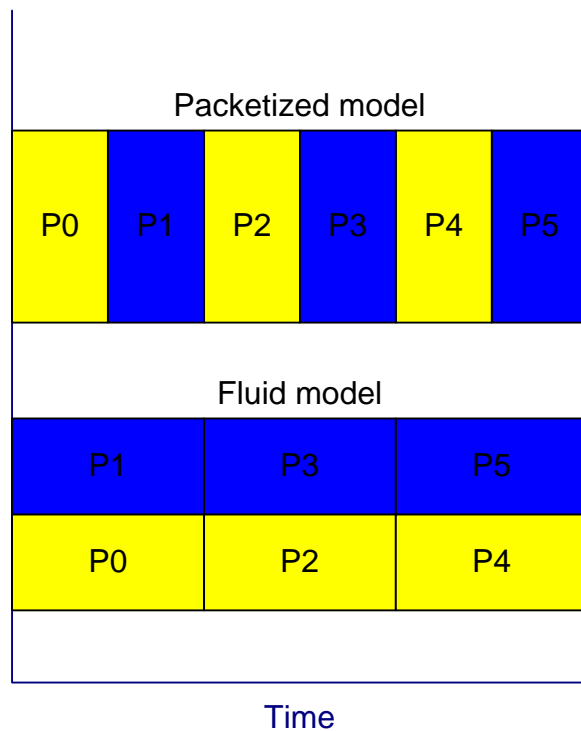


Figure 3. Packetized model versus fluid model

The fluid model is easier to reason about than the packetized model. This is because of the continuous character of the fluid model. At a certain moment in time, both streams can be seen and not only a single packet as in the packetized model. In the fluid model each stream has a private channel with a certain bandwidth. In the packetized model there is an interleaving of streams: only one channel with a certain bandwidth. The packetized model captures more aspects of reality so it can explain behaviour which cannot be explained with the fluid model (especially when looked at

queuing). Therefore we will choose the packetized model for explaining the results of our measurements.

### 3.1.2 Path model

Traffic goes from one device to another device using a path. The relation between the number of links and the number of relay-devices is  $N_{links} = N_{relay} + 1$ . If we look at the relay-devices, we can see three categories of relay-devices.

- Store-and-forward relay-devices
- Cut-through relay-devices
- Repeater relay-devices

A store-and-forward relay-device first needs to receive the entire packet from a link before it is forwarded onto the next link. A cut-through relay-device only needs to receive the address field of a packet before it is able to forward the entire packet, so this kind of relay-device can do its work faster than a store-and-forward relay-device because only the header needs to be received instead of the complete packet before the relay-device can start forwarding the packet. The last category consists of repeater relay-devices. These kinds of devices receive data from one link and forward it to all other links. Repeaters do not receive a packet entirely before it is forwarded; they start forwarding right after the first bit / byte.

In practise, cut-through relay-devices are very rare and not available to the normal consumer. Repeaters are still there but most home networks consist of store-and-forward relay-devices. So we focus on the repeater and the store-and-forward relay-devices.

If we look at a path, it has certain performance properties. The properties that are interesting to know are:

- Latency(path, packet, time): the latency of a packet (of a certain size) on a path at a certain time;
- Total\_Bandwidth(path): the maximum bandwidth of a path;
- Available\_Bandwidth(path, time): the available bandwidth of a path at a certain time (equal to or lower than Total\_Bandwidth(path));
- Cost(path, packet): the amount of money or time it costs to send/receive a packet over a path;
- Loss(path, interval): the percentage of packets that get lost on a path during a certain interval.

We are especially interested in the latency and the available bandwidth, because these properties can be used to check if it is possible to send a stream of data having certain properties. Loss is also an interesting property (e.g. loss profiles for the purpose of control), but the problem here is that loss only occurs in heavily congested networks or if the network has a bad condition (e.g. wireless network with microwave oven nearby, bad cables etc.). This makes it hard to actively measure loss in a not-congested network without congesting the network itself thereby interfering with the current traffic. Furthermore adding measurement traffic to a network with a bad condition can decrease the quality of the network even more. It is however possible to look at all the traffic that is sent and received to determine the loss, which can be used to determine the loss over time. The total bandwidth is nice to know, but it is of no use when one wants to see if it is possible to add a new data stream to the network. Cost is nice, but cannot be measured without knowing the costs per byte / second etc. If the values of the different properties are known they can be used to compare the paths with each other and choose the best path. We believe that latency and the

available bandwidth are the most useful properties for comparing paths with each other, so we focus on these properties.

In the rest of this chapter models are given to determine the latency and transmission time as well as the available bandwidth of a path, in section 3.2 we discuss the algorithms that are based on these models to estimate the values of these two properties.

### 3.1.2.1 Background for measuring latency

If we look at a path, it consists of  $N$  physical links, say  $l_i$  with  $0 \leq i < N$  and  $N \geq 1$  (because there is always at least one link between two connected interfaces). If  $N \geq 2$ , then link  $l_i$  and  $l_{i+1}$  are connected with each other using a relay-device named  $r_i$ . This relay-device can use store-and-forward or it is a repeater. If a packet  $p$  of size  $s$  is sent from device  $A$  to  $B$  that are connected using two links ( $l_0$  and  $l_1$ ) and a relay-device  $r_0$  (using store-and-forward),  $r_0$  first receives the packet completely before it is forwarded using the link  $l_1$  to device  $B$ . This store-and-forward mechanism causes a delay before the packet is received by device  $B$  (see Figure 4). This delay consists of the time the packet is queued in the buffers of the relay-device and the time needed for the relay-device to put the packet on the next link of the path. The first depends on the presence of cross traffic and the operating rate of the relay-device and the second equals the size  $s$  of the packet divided by the bandwidth  $b_i$  of link  $i$  [19].

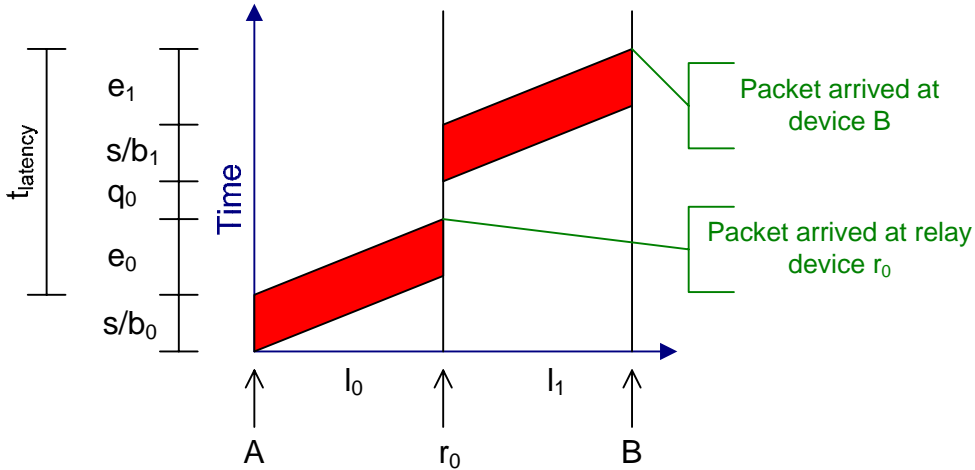


Figure 4. Delay on a path caused by a store-and-forward relay-device

If we look at Figure 4, we also see that there passes a certain amount of time before the first bit is received at the relay-device. This is the link latency  $e$ . If we combine the variables and look at Figure 4 we get the following equation for transmission time:

$$t_{transmission} = \sum_{i=0}^{N-1} \left( \frac{s}{b_i} + e_i \right) + \sum_{i=0}^{N-2} q_i$$

We defined the latency of a packet of size  $s$  on a path as the transmission time of the packet minus the time for the sink to receive the entire packet. Formally, we get:

$$t_{latency} = \sum_{i=0}^{N-1} \left( \frac{s}{b_i} + e_i \right) + \sum_{i=0}^{N-2} q_i - \frac{s}{b_{N-1}}$$

In case there is no cross traffic, the queuing delay  $q$  can be neglected. Also in reality the link latencies  $e_i$  are very small compared to  $\frac{s}{b_i}$ . This means that when two devices are connected directly to each other (without any relay-devices), the path latency

equals 0 seconds. The formula to determine the latency ignores the bandwidth of the last link. As a result, the latency of a path can be different in the other direction. Therefore the latency of the path needs to be measured in the same direction as the data is going to be sent.

### 3.1.2.2 Background for measuring available bandwidth

Each link has a maximum bandwidth, so the maximum bandwidth of a path is the maximum bandwidth of the link with the minimal maximum bandwidth. When a device wants to send data, it has no use for the maximum bandwidth of each path. However the available bandwidth of the path is useful. The available bandwidth of a path equals the available bandwidth of the link with the minimum available bandwidth. Each link  $i$  has an available bandwidth  $b_i$  in bits per second. Between the sending of a packet of size  $s$  at the source and receiving the packet at the sink it takes  $t_i$  seconds. So the available bandwidth of a single link  $i$  equals  $b_i = \frac{s}{t_i}$ . For the entire

path consisting of  $N$  links we get the equation  $b_{path} = \left\langle \downarrow b_i : 0 \leq i < N \wedge b_i = \frac{s}{t_i} : b_i \right\rangle$ . If

a stream is sent over the path, it will be received at the bit rate of the slowest link. Therefore it is not necessary to look at the available bandwidth of each link to estimate the available bandwidth of the entire path; it is enough to only look at the available bandwidth of the entire path.

## 3.2 Measuring path properties

In the previous section we discussed the models on how we see traffic and the properties of a path. In this section algorithms are discussed that are able to estimate the values of the model parameters.

### 3.2.1 Measuring latency

For measuring, we only use arrival time differences between packets to avoid clock synchronization between sender and receiver. A small packet has a smaller transmission time than a large packet: if the size  $s$  of a packet is small, then the latency is also small because  $\frac{s}{b_i}$  is small. So if we precede a large packet with a much

smaller packet, the small packets can be used to announce the start and end of the sending of the large packet to the sink. We use the so-called packet train given in Figure 5.

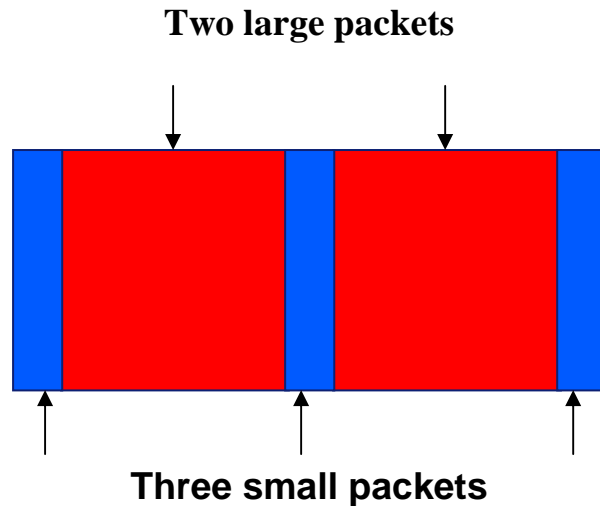


Figure 5. The Big Mac probe

The Big Mac probe (which is a modified sandwich probe [8]) consists of two large packets (each 1500 bytes) interleaved with three small packets (each 75 bytes). The assumption is that the MTU of the network is the same or higher than the size of the large packet. In our case the MTU of the network is 1500. If there is no cross traffic on the path (no queuing delay) between a source  $A$  and sink  $B$ , we get the behaviour of the Big Mac probe on the path displayed in Figure 6.

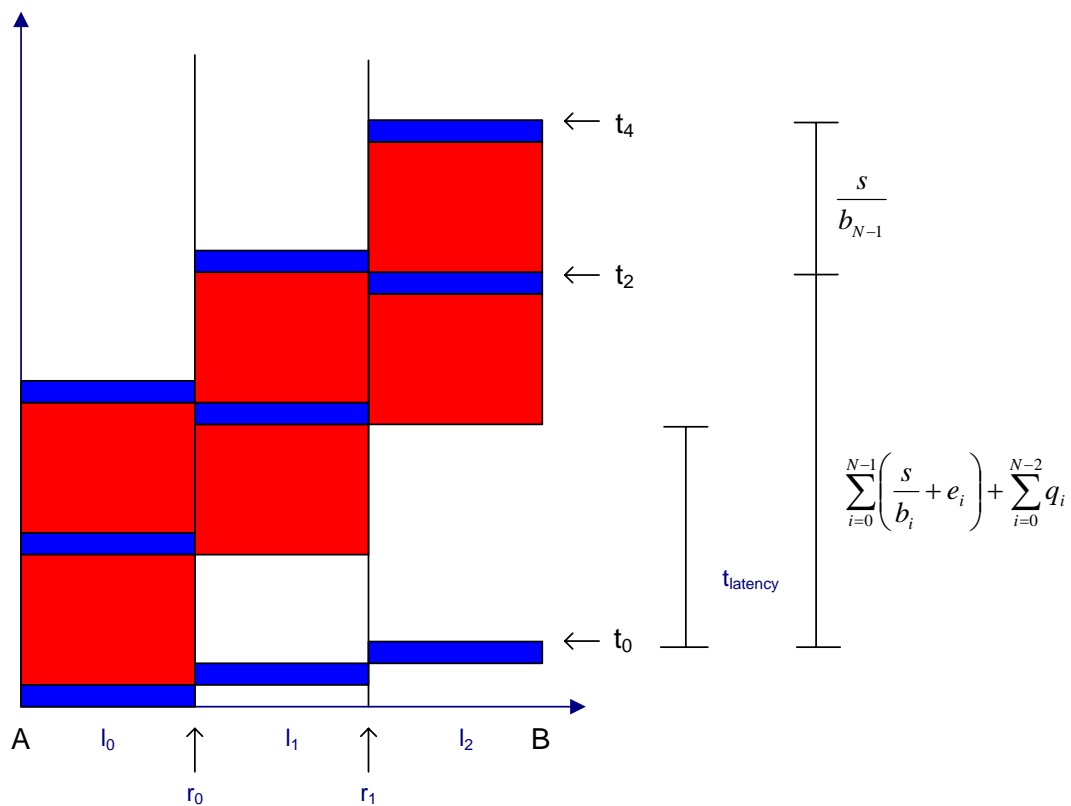


Figure 6. The Big Mac probe on a path ( $N = 3$ ), without cross traffic ( $q = 0$ ).

As seen in Figure 6, the small packet gets queued right after the large packet. So the small packet arrives immediately after the large packet. Between the first small packet and first large packet an extra delay is introduced at each device that uses store-and-forward.

Now we can calculate the path latency of the large packet by using the arrival times  $t_i$  of the small packets:  $t_{latency} = (t_2 - t_0) - (t_4 - t_2)$ .

$$\begin{aligned}
& t_{latency} \\
& = \{def.t_{latency}\} \\
& (t_2 - t_0) - (t_4 - t_2) \\
& = \{def.t_i\} \\
& \left( \left( t_{offset} + 2 * \frac{s_{small}}{b_{N-1}} + \sum_{i=0}^{N-1} \frac{s_{large}}{b_i} \right) - \left( t_{offset} + \sum_{i=0}^{N-1} \frac{s_{small}}{b_i} \right) \right) \\
& - \left( \left( t_{offset} + 3 * \frac{s_{small}}{b_{N-1}} + \sum_{i=0}^{N-1} \frac{s_{large}}{b_i} + \frac{s_{large}}{b_{N-1}} \right) - \left( t_{offset} + 2 * \frac{s_{small}}{b_{N-1}} + \sum_{i=0}^{N-1} \frac{s_{large}}{b_i} \right) \right) \\
& = \{assumption : \langle \forall i : 0 \leq i < N - 1 : b_i == b_{i+1} \rangle\} \\
& (N - 1) \left( \frac{s_{large} - s_{small}}{b} \right)
\end{aligned}$$

The actual latency is a bit higher because the latency of the first small packet is not taken into account. Because the size of the first packet is very small compared to the second (large) packet,  $\frac{s_{small}}{b}$  can be neglected, and thus the latency of the small

packet equals 0 seconds and  $t_{latency} = (N - 1) \frac{s_{large}}{b}$  which equals our model

$$t_{latency} = \sum_{i=0}^{N-1} \left( \frac{s}{b_i} + e_i \right) + \sum_{i=0}^{N-2} q_i - \frac{s}{b_{N-1}}$$

when  $q$  and  $e$  can be neglected. This is the

theoretical latency of the large packet on a path of N links. We show in section 3.3.2 that the end-point devices also increase the latency of the path.

Using the arrival times of the small packets, it is also possible to determine the transmission time of the large packet on the path:  $t_{transmission} = (t_2 - t_0)$  which equals

$$\begin{aligned}
& t_{transmission} \\
& = \{def.t_{transmission} = t_2 - t_0\} \\
& \left( t_{offset} + 2 * \frac{s_{small}}{b_{N-1}} + \sum_{i=0}^{N-1} \frac{s_{large}}{b_i} \right) - \left( t_{offset} + \sum_{i=0}^{N-1} \frac{s_{small}}{b_i} \right) \\
& = \{assumption : \langle \forall i : 0 \leq i < N - 1 : b_i == b_{i+1} \rangle\} \\
& (2 - N) \left( \frac{s_{small}}{b} \right) + N \left( \frac{s_{large}}{b} \right)
\end{aligned}$$

Again  $\frac{s_{small}}{b}$  is very small compared to  $\frac{s_{large}}{b}$ , so  $t_{transmission} = N \left( \frac{s_{large}}{b} \right)$ , which again equals our model  $t_{transmission} = \sum_{i=0}^{N-1} \left( \frac{s}{b_i} + e_i \right) + \sum_{i=0}^{N-2} q_i$  where  $q$  and  $e$  are neglected.

### 3.2.2 Available bandwidth

To measure how much bandwidth is still available, we use the concept of self-induced congestion: if the bit rate of a data stream exceeds the available bandwidth of a path, then the packets of the data stream are queued at the switch or router connected to the link with the lowest available bandwidth. To generate a stream of a certain bit rate, a constant number of equally sized packets is sent. Each so-called packet train has a different bit rate. This way we get packet trains with differing bit rates. At a certain moment the transfer time of each packet will increase because of the extra queuing delay (Figure 7). The value of the bit rate of the packet train when the queuing delay increases is the available bandwidth of the path. Because both the source and sink know about this algorithm, it is only necessary to measure the arrival times of the packets at the sink and look at the relative arrival time differences between the packets. This method avoids clock synchronization between source and sink.

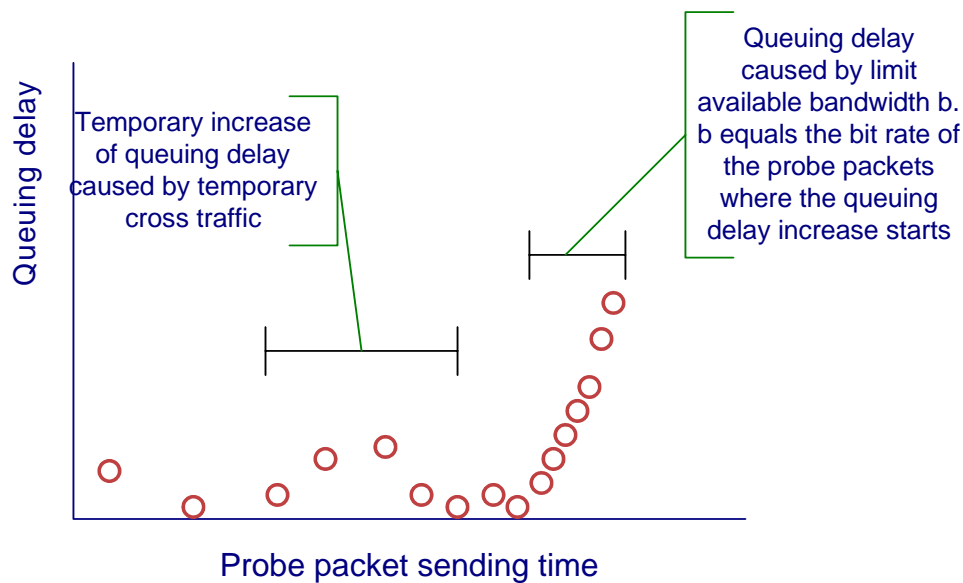


Figure 7. Measured queuing delay of packet trains with increasing bit rate

There are different implementations that use to concept of self-induced congestion to measure the available bandwidth. We will compare two of them: pathChirp [14] and Pathload [15]. PathChirp uses a packet train with exponential increasing bit rate of the packets (Figure 8) to measure the available bandwidth.

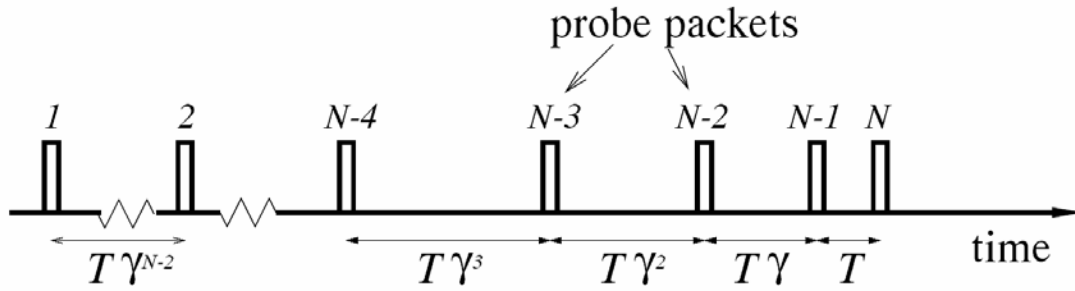


Figure 8. A packet train with exponential decreasing delay between packets (source: [14])

The packets of a packet train get interleaved with the packets that are already on the network. If the bit rate of the packet train exceeds the available bandwidth of the path, the arrival time differences between the packets get larger because of queuing delay in the relay-devices. The packet train is interleaved with other packets in such a way that the bit rate of the packet train is decreased. Comparing different measurements filters out temporary cross traffic. Other (cross traffic) streams will lower the available bandwidth of a path. This kind of cross traffic is constantly present, so it is possible to measure the new value of the available bandwidth.

Pathload does not use a packet train with an exponential increasing bit rate but a packet train with a constant bit rate. Each train has a different delay between packets. Because a train has a constant bit rate, it is easier to filter out temporary cross traffic, context switching and other temporary anomalies in the network compared to pathChirp. But Pathload needs more probing packets before a good estimate can be given. We will compare Pathload and pathChirp in section 3.3.3.

Both algorithms do add extra traffic which causes congestion. But the packet trains only consist of a couple of packets, and the queuing can be detected when the packet train has the bit rate that equals the available bandwidth of a path, so no packet trains of higher bit rates are needed. Therefore, the algorithms minimize the effect that they have on the current traffic.

The measured amount of available bandwidth is only valid as long as no streams are added to or removed from the network. So regular measuring of the available bandwidth is needed or a technique where only one device is allowed to add a stream at a time (after which devices can determine the available bandwidth again, see section 4.5.6). Because we believe that chances are very low that multiple devices try to add new (large) data streams to the network, we do not take the validity of the measurements over time into account. We only require that measurements are repeated periodically so that a snapshot of the network status is updated periodically.

### 3.3 Estimation of path properties

The methods that are discussed in the previous section to determine the path property values have been implemented and tested. In this section the test set up is given together with the results of the tests.

#### 3.3.1 Test network

We used two devices:

- Pentium II 350Mhz with 128MB (Device A)

- Pentium III 550Mhz with 128MB (Device B)

Both devices have Linux as their Operating System and were booted into run level 1 (single user mode) so that all unnecessary services are disabled. These two devices were connected for the latency tests using the following set ups:

- Cross-link cable (100Mbit/s)
- One store-and-forward relay device (one 100Mbit/s switch)
- Two store-and-forward relay devices (two 100Mbit/s switches)
- One repeater (one 10Mbit/s hub)
- Two repeaters (two 10Mbit/s hubs)

The set up with two switches is displayed in Figure 9. This set up is also used to test the available bandwidth algorithms because when two extra devices (Device C and D) are connected to the switches, the cable that connects both switches acts as a shared link with a maximum bandwidth of 100Mbit/s between data streams going from A to B and from C to D.

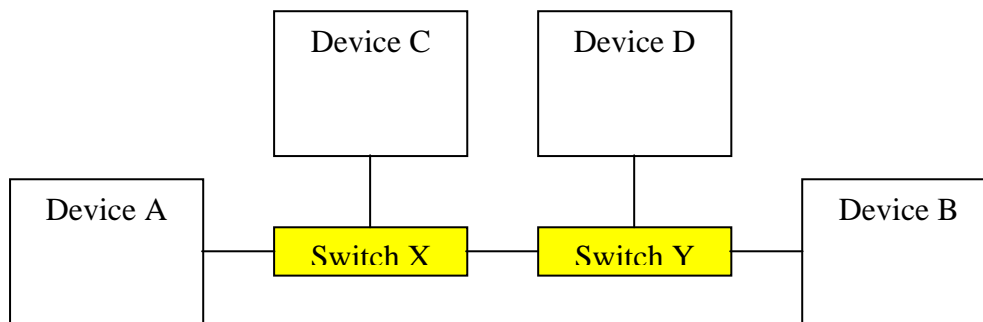


Figure 9. Set up to perform tests with two relay-devices and one shared link.

The tool Iperf [18] is used to generate cross traffic between devices C and D. This cross traffic consists of UDP packets (which fit in a level 2 packet) that are sent at a constant bit rate.

### 3.3.2 Latency

We extended the probe discussed in the previous chapter with two extra large packets (together with two small packets) to show that the all the large packets coming after the first one have the same latency. This probe is given in Figure 10.



Figure 10. A packet train consisting of small and large packets

The small packets are each 75 bytes and the large packets are 1500 bytes. This includes the IP and UDP headers. Because the MTU of the network equals the size of the large packet, there is no extra delay introduced caused by the division and reassembling of the packets.

For each set up we measured the send and arrival times of the packet probe in two directions. Because the path between the two devices is the same in both directions, we can see if the configuration of the device influences the measurements.

### 3.3.2.1 Cross-link cable

The first test we performed is connecting the two devices using one crosslink cable. This means there is no relay-device between the two devices and only one link. Therefore we only expect extra delay caused by the store-and-forward behaviour of the Operating System and NIC of the receiver.

First we look at the arrival times of a single probe to see if there is a clear distinction between the arrival times of the large and small packets (Figure 11).

	P1	P2	P3	P4	P5	P6	P7	P8	P9
A to B	0	189	196	318	325	448	455	578	584
B to A	0	200	221	330	349	460	479	587	606

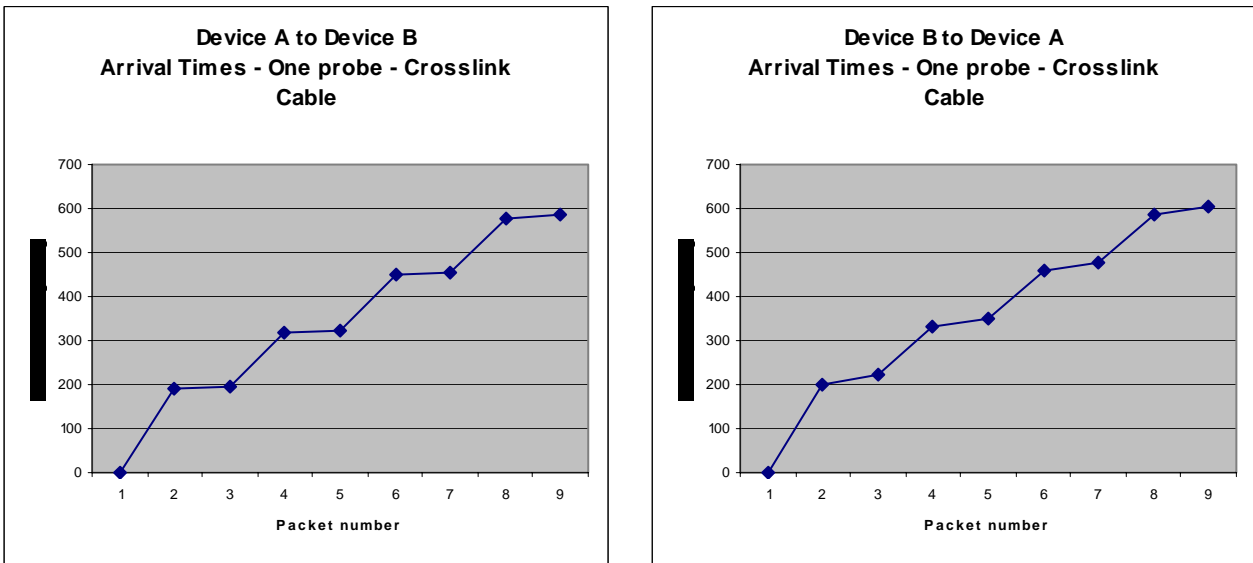


Figure 11. Arrival times of the packet train when there is only one cross-link cable

	A to B	B to A
Mean diff. P3-P2, P5-P4, P7-P6, P9-P8	6.508	19.305
Mean diff. P5-P3, P7-P5, P9-P7	129.643	128.543
Mean diff. P3-P1	195.640	220.860

As seen in the above results, the difference between a pair is constant, although when sending from the slow device to the fast device, the delay difference between the large and small packet is lower than the other way around. This can be explained because device A needs more time to handle a packet than device B (because device A is slower).

With a crosslink cable we expected that P3-P1 equals P5-P3 and that the latency equals 0. The results however show that in both directions P3-P1 is larger than P5-P3. They differ around 66 to 90 microseconds. To determine if this value is independent of the size of the packet, we repeated the Big Mac probe experiment over a cross-link cable with different sized large packets (ranging from 75 bytes to 1500 bytes), measuring from device A to B. Using the arrival times of the small packets we calculated the latency of the large packets. The results are given in Figure 12. We see that if the large packet increases its size, then the latency also increases. We see that the measured latency depends on the size of the packet. We also see that the first few measurements differ from the measurements of the larger packets. This can be

explained looking at the method used to calculate the latency: if the large packet is much larger than the small packet, the impact of the small packet on the latency can be neglected. In this experiment however the large packet is a little larger than or equals the size of the small packet which makes these first few measurements unreliable. The reliable measurements form a straight line that starts at 20 microseconds. Because we have a chain of several packets, it results into pipelining in the transmission. This pipeline needs some time to initialize, which equals the estimated value of 20 microseconds. Furthermore there is some sort of handling delay which depends on the size of the packet: the handling bandwidth. This handling bandwidth equals around 280 Mbit/s and is caused by the handling of the packet by the Operating System and the hardware.

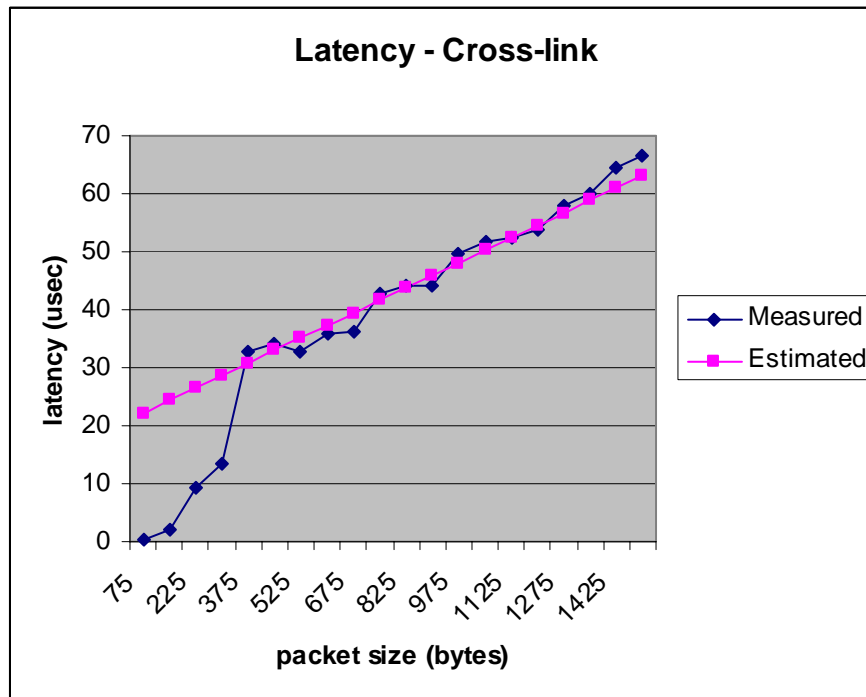


Figure 12. Latency over a cross-link cable using the Big Mac probe with different packet sizes

We extend our model to include the pipeline initialization and the handling bandwidth of a path and simplify it by assuming there is no queuing delay and link latency is negligible to the latency of the entire path (from application layer to application

layer):  $t_{latency} = t_{init} + \frac{s}{b_{handling}} + \sum_{i=0}^{N-1} \left( \frac{s}{b_i} \right) - \frac{s}{b_{N-1}}$  and  $t_{transmission} = t_{init} + \frac{s}{b_{handling}} + \sum_{i=0}^{N-1} \left( \frac{s}{b_i} \right)$ .

So when there is only one (Cross-cable) link:

$$\begin{aligned}
 & t_{latency} \\
 & = \{def. t_{latency}\} \\
 & t_{init} + \frac{s}{b_{handling}} + \sum_{i=0}^{N-1} \left( \frac{s}{b_i} \right) - \frac{s}{b_{N-1}} \\
 & = \{N = 1\} \\
 & t_{init} + \frac{s}{b_{handling}}
 \end{aligned}$$

For measuring from device B to A we see that the measured arrival time differences differ from measuring from A to B. The  $b_{handling}$  is lower here because the arrival time differences between the large and small packet are higher than in the previous situation. When we repeated the latency determination tests between two identical machines, it did not matter which of them the receiver or sender was. The results in both directions were the same. Therefore  $b_{handling}$  and  $t_{init}$  depend on the used hardware.

### 3.3.2.2 One Switch

The second test we performed is connecting the two devices using one 100Mbit/s switch. This means there is one relay-device between the two devices and two links. The relay-device uses store-and-forward, so we expect to see an extra delay caused by the switch. Using our model, we expect a path latency of 183 microseconds when measuring from device A to B.

First we look at the arrival times of a single probe to see if the small packet arrives immediately after the large packet (Figure 13).

	P1	P2	P3	P4	P5	P6	P7	P8	P9
A to B	0	303	309	432	439	562	568	692	698
B to A	0	315	335	444	463	574	593	702	721

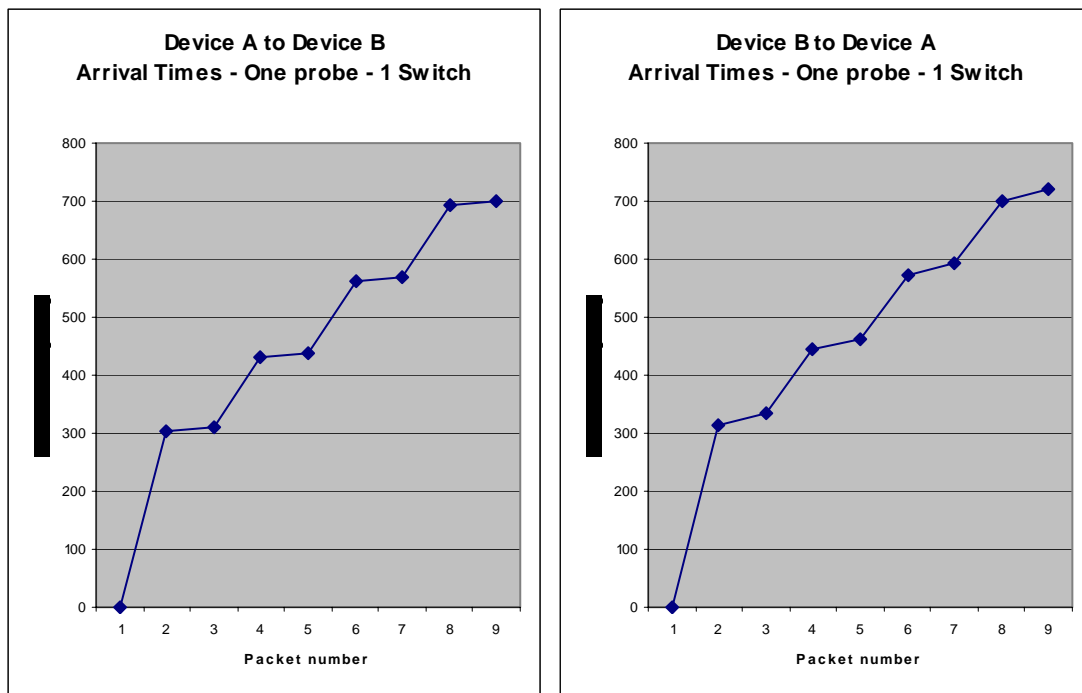


Figure 13. Arrival times of the packet train when there is one switch

We see the same behaviour as with only a crosslink cable. The only difference is the arrival time of P2, which is larger. This is exactly what we expected because the relay-device has to receive the packet completely before it can be forwarded. The second large packet needs the same amount of time to arrive as the second large packet with only a cross-link cable. The large packets after the second large packet also need the same amount of time as the second large packet to be received. This behaviour occurs because the first large packet has 'initialized' the pipeline.

If all arrival times of the packets are plotted, we get the following results:

	<b>A to B</b>	<b>B to A</b>
Mean diff. P3-P2, P5-P4, P7-P6, P9-P8	6.583	19.485
Mean diff. P5-P3, P7-P5, P9-P7	129.740	128.580
Mean diff. P3-P1	309.140	335.010

The mean delay difference between P3 and P2 (and the other combinations of a successive large packet and small packet) equals the mean delay difference of the set up where the two devices are connected using only a crosslink cable. The same goes for the mean delay difference between P5 and P3 (and the other successive delay differences between small packets). The delay difference of P3 and P1 however is much larger, which is caused by the extra store-and-forward delay introduced by the switch. The measured latency from device A to B equals  $309 - 130 = 179 \mu\text{s}$  which differs only a couple of microseconds with the estimated latency of 183 microseconds. Furthermore the difference between the P3-P1 values of the situation with only a cross-link cable ( $220.86 - 195.64 = 25.22 \mu\text{s}$ ) equals the difference in this situation ( $335.01 - 309.14 = 25.87 \mu\text{s}$ ), so only  $b_{\text{handling}}$  and  $t_{\text{init}}$  differ between the two paths (A to B and B to A).

### 3.3.2.3 Two Switches

The third test we performed is connecting the two devices using two 100Mbit/s switches. This means there are two relay-devices between the two devices and three links. The relay-devices use store-and-forward, so we expect to see an extra latency caused by the second switch compared to the situation where there is only one switch. In the test with one switch cable, the measurement corresponded with the expected latency value. Using the model we calculate an expected latency of  $303 \mu\text{s}$ .

First we look at the arrival times of a single probe Figure 14.

	P1	P2	P3	P4	P5	P6	P7	P8	P9
A to B	0	418	424	547	554	677	683	807	813
B to A	0	430	450	559	578	689	708	817	836

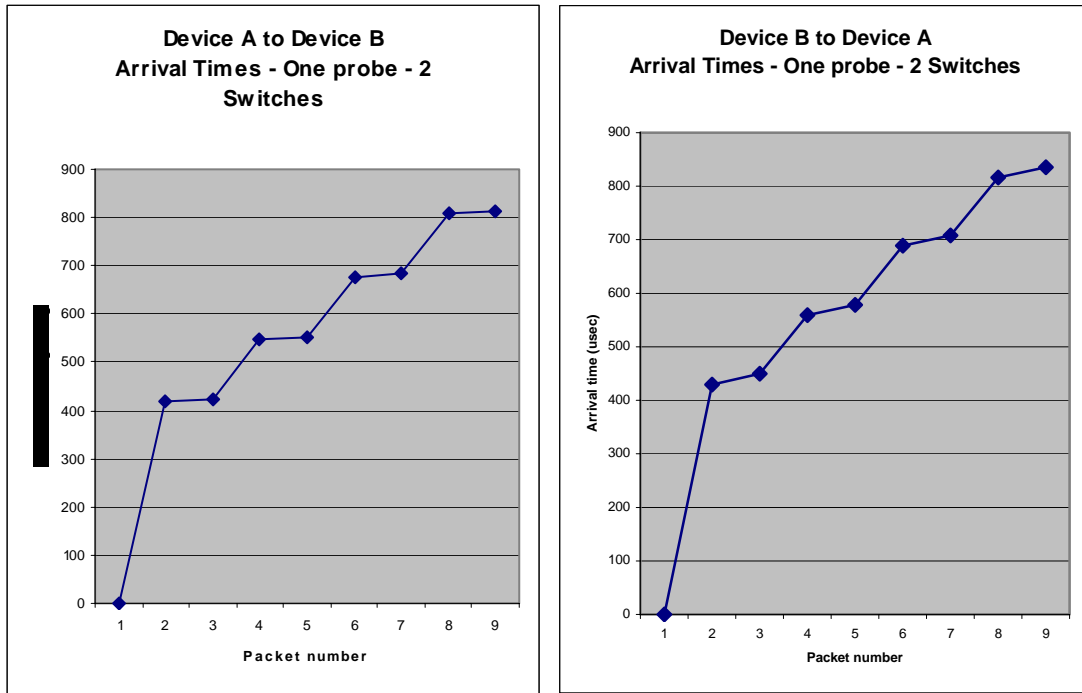


Figure 14. Arrival times of the packet train when there are two switches

Again we see exactly what we expected: the first large packet takes much longer than all the other packets. Furthermore the other large packets have the same delay as the other large packets in case there was only one switch or only a crosslink cable and the differences between A to B and B to A stay the same for the different situations.

	<b>A to B</b>	<b>B to A</b>
Mean diff. P3-P2, P5-P4, P7-P6, P9-P8	6.480	19.498
Mean diff. P5-P3, P7-P5, P9-P7	129.687	128.593
Mean diff. P3-P1	424.390	449.860

Again the first two rows are the same as previous experiments. The measured latency from device A to B is  $424 - 129 = 295 \mu\text{s}$ , which differs less than  $10 \mu\text{s}$  from the estimated value.

We repeated the Big Mac probe experiment over the path with two switches with different sized large packets (ranging from 75 bytes to 1500 bytes), again only measuring from device A to B. The measured latencies are given in Figure 15.

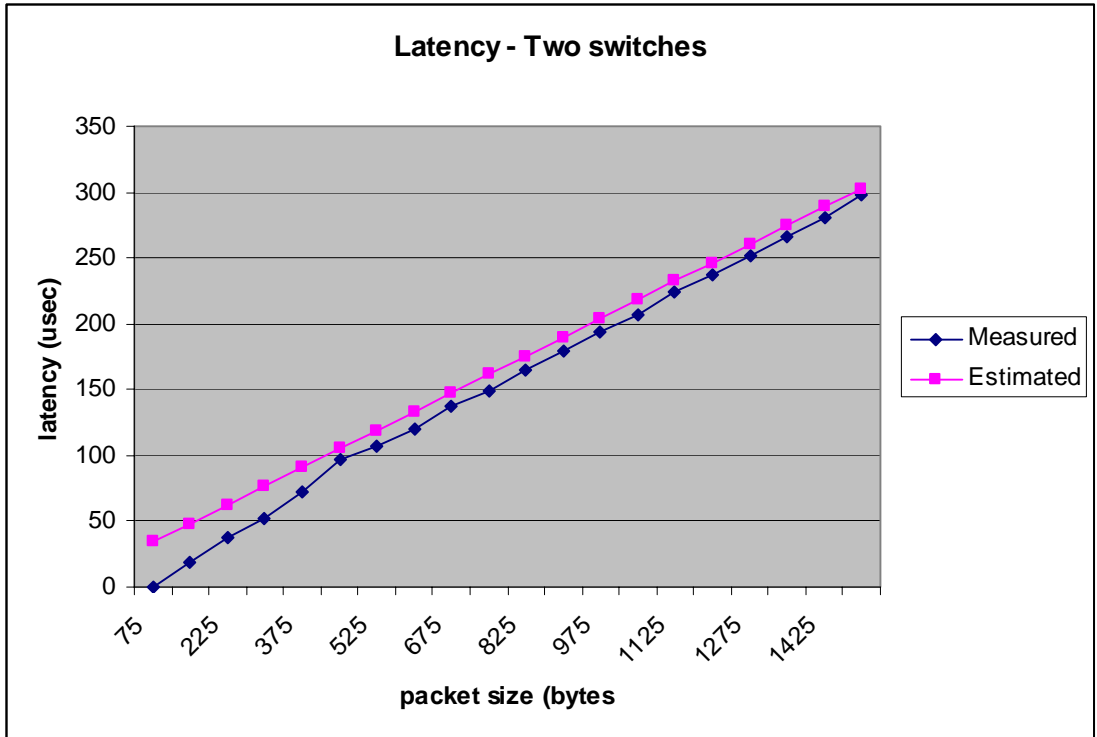


Figure 15. Latency over a path of two switches using the Big Mac probe with different packet sizes

We see the same behaviour as in Figure 12 where the path consisted of only one cross-link cable. Again the first couple of measured values are not accurate and the measured values equal the estimated values differing less than 10  $\mu$ s. If we put all the measured and predicted latency values of the large packet in one graph, we get the graph given in Figure 16.

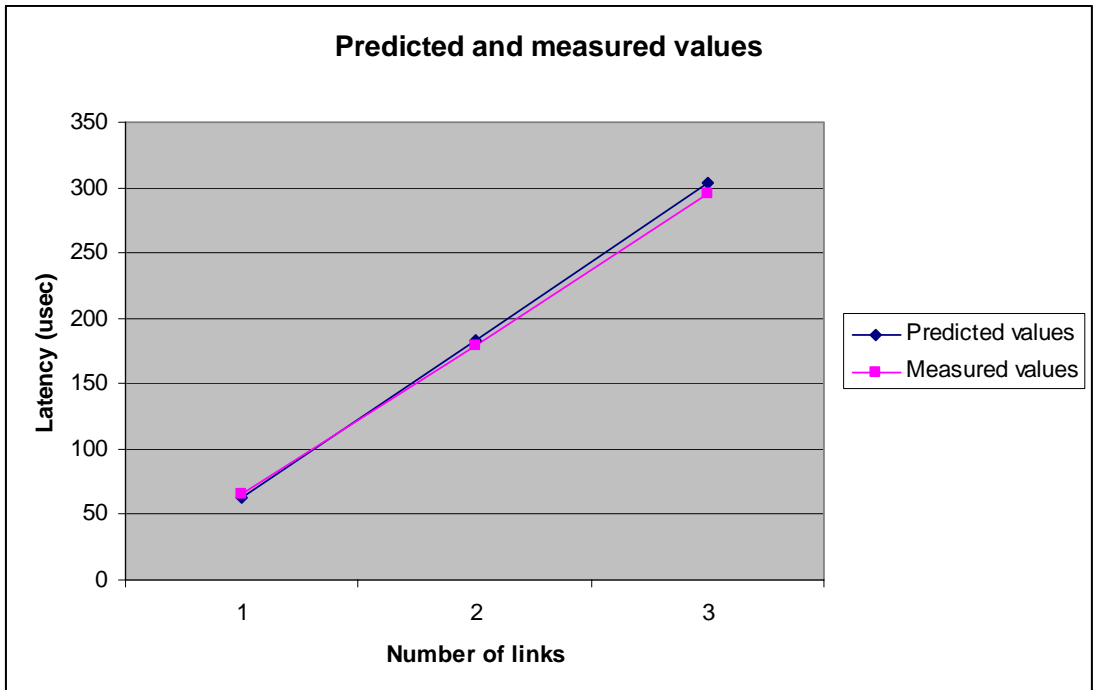


Figure 16. Comparison between measured and predicted latency values

We see that our predicted values almost equal the measured values. So our model is usable for the home network although the results of cross traffic are unpredictable (see section 3.3.2.7).

### 3.3.2.4 One hub

A hub differs from a switch because a hub does not use a store-and-forward mechanism. So we expect that there is no difference between the delay difference of P3 and P1 and the delay difference of P5 and P3. The path between device A and B is seen as a single link (instead of two). The hubs we use only support 10 Mbit/s, so if

we look at our model with  $N = 1$ :  $t_{latency} = t_{init} + \frac{s}{b_{handling}}$  and

$t_{transmission} = t_{init} + \frac{s}{b_{handling}} + \frac{s}{b_0}$ , we can calculate the latency and transmission time of

the large packet ( $t_{init} = 20$ ,  $b_{handling} = 280$ ,  $s = 1500 * 8$ ,  $b_0 = 10$ ):  $t_{latency} = 62 \mu s$  and  $t_{transmission} = 1262 \mu s$ . We see that the expected latency equals the latency of the large packet of the cross-link case. To see if this is true, we first look at the arrival times of the individual packets (Figure 17) of the extended Big Mac probe (3.3.2).

	P1	P2	P3	P4	P5	P6	P7	P8	P9
A to B	0	1253	1301	2554	2602	3855	3904	5157	5205
B to A	0	1339	1359	2639	2659	3940	3959	5240	5259

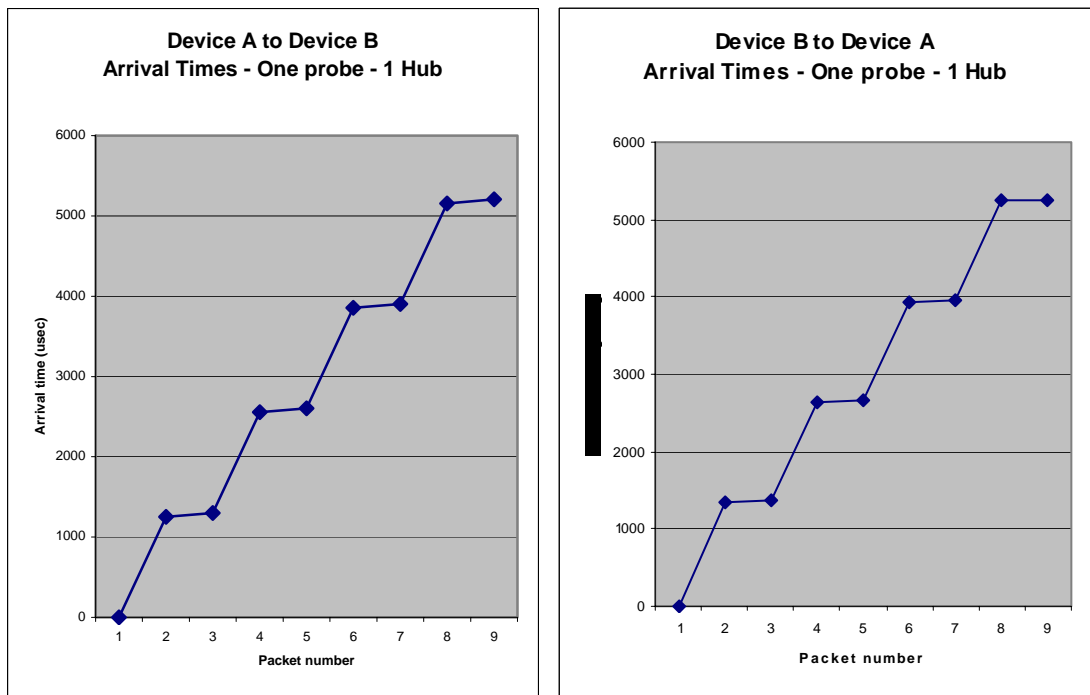


Figure 17. Arrival times of the packet train when there is one hub

As seen in the above graphs, the large packets all have the same transmission time (even the first one), meaning that the measured latency equals  $0 \mu s$ .

	<b>A to B</b>	<b>B to A</b>
Mean diff. P3-P2, P5-P4, P7-P6, P9-P8	48.063	19.438
Mean diff. P5-P3, P7-P5, P9-P7	1301.453	1300.023
Mean diff. P3-P1	1300.650	1359.130

From device A to B the measured latency equals:  $1300 - 1301 = -1 \mu\text{s}$ . Latency however cannot be a negative value, but because measured values are not 100% free from other distortions (e.g. process switching on a device or physical processes) and there is the problem of significant numbers of the measured values we say that the latency equals  $0 \mu\text{s}$ . This is not what we expected because during the tests with a

cross-link cable, we came to the conclusion that  $t_{latency} = t_{init} + \frac{s}{b_{handling}}$ . But if we look

at the estimated transmission time  $t_{transmission} = 1262 \mu\text{s}$ , it seems that the pipeline (consisting of the hardware and operating system on both source and sink) needs to

initialize itself for every packet, adding  $t_{init} + \frac{s}{b_{handling}}$  to the latency of every packet.

When measuring from device B to A the latency does not equal 0, furthermore another strange thing is seen: sending the packets from device A to B shows a larger time difference between P3 and P2 compared to the situation where only switches are used. We expected this value to be the same as in the switch situation (which is the case when sending packets from device B to A). We expect that the differences are not caused by the operating system, but by the hardware because the operating system is not changed between the hub case and the switch case. This means the hardware can influence the measurements: more research in the working of hardware is needed to know how much influence the hardware has and if this influence is predictable.

### 3.3.2.5 Two hubs

Because a hub does not use store-and-forward, the amount of hubs on a path should have no influence on the packet arrival times, because every bit is forwarded without any extra delay. When we repeated the test with two hubs, we saw that there is no significant difference between the results with one hub and with two hubs.

### 3.3.2.6 Number of relay devices

A side effect of the latency measurements using the Big Mac probe is that these measurements show a relation between the number of (store-and-forward) relay devices on a path and the measurement. When all relay devices on a path are equal, then each store-and-forward relay device adds the same amount of latency to the total latency of a packet. If we put the path latency compared to the number of relay devices in a graph we get a straight line as seen in Figure 18.

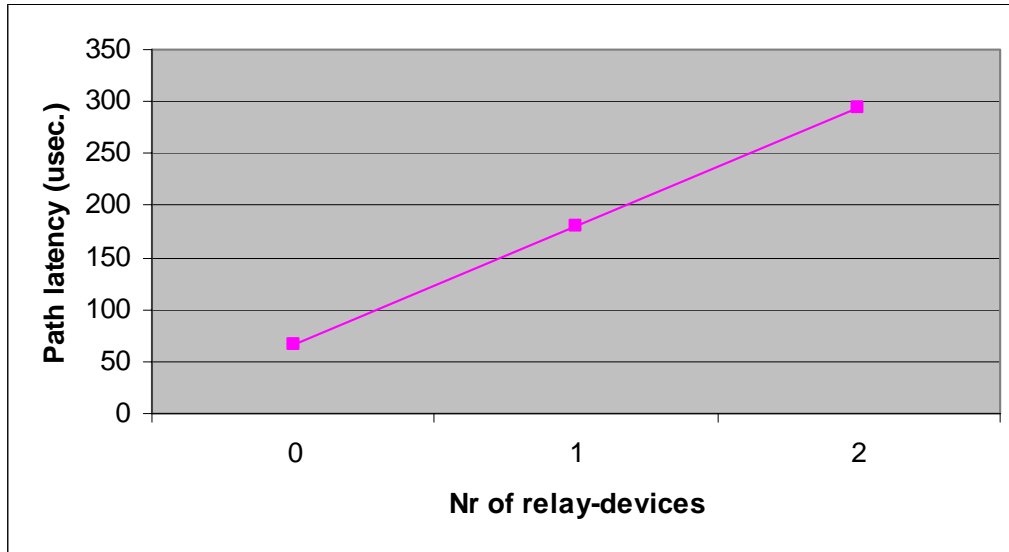


Figure 18. Latency compared to the number of store-and-forward relay devices

Unfortunately, this information can only be used to estimate the number of relay devices when there is no cross traffic on the network, because cross traffic can have a major influence on the measurements (see section 3.3.2.7).

### 3.3.2.7 Cross traffic

Using the same set up as used in 3.3.2.3, it is possible to repeat the latency measurements with the presence of cross traffic. Because of both traffic models, we expect that every packet gets the same extra delay depending on the bit rate of the cross traffic. We expect the cross traffic to decrease the bandwidth of a path (lowering

$b_i$  and  $b_{N-1}$  in the model:  $t_{latency} = t_{init} + \frac{s}{b_{handling}} + \sum_{i=0}^{N-1} \left( \frac{s}{b_i - b_{cross}} \right) - \frac{s}{b_{N-1} - b_{cross}}$  ) and

therefore increasing the latency of the path. The results of the tests with different amounts of cross traffic are presented in Figure 19.

As seen in the graph (Figure 19), the results do not correspond to what we expected. The first couple of packets arrive at the same time no matter how high the cross traffic is. The fluid model is not able to describe this behaviour. The packetized model can explain this result because cross traffic gets interleaved with the Big Mac probe which can cause packets that do get extra delay and packets that do not get the extra delay (Figure 20). The longer the probe is, the higher the probability that one of the packets

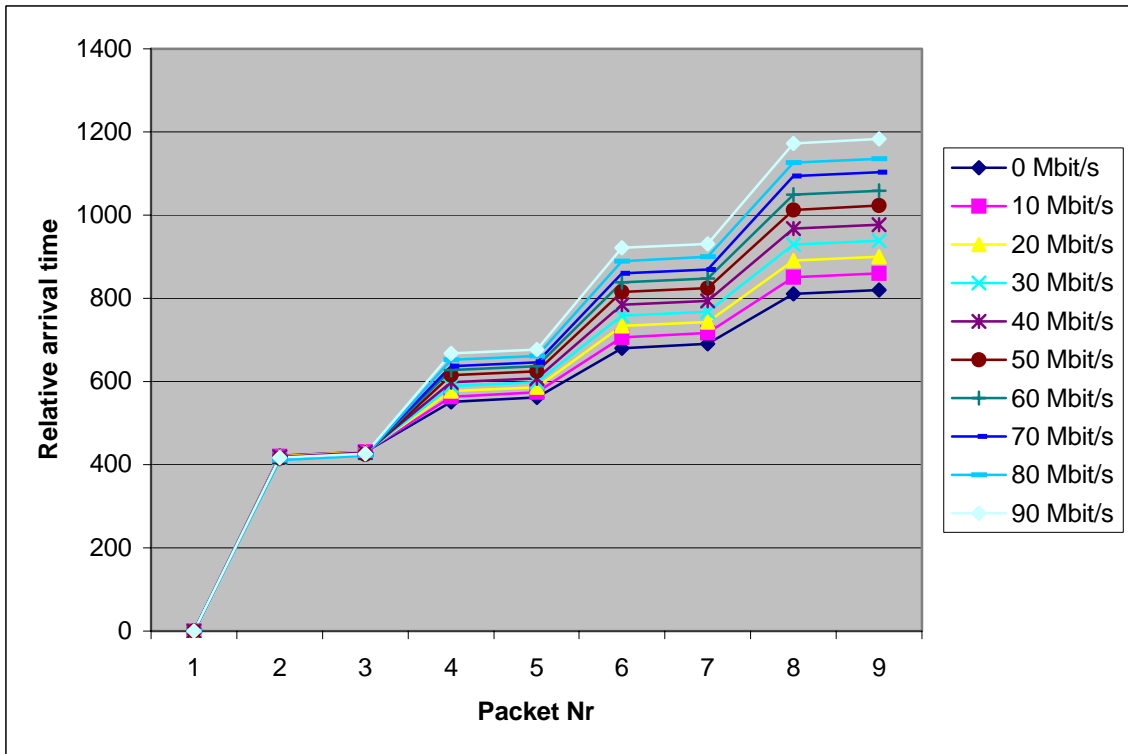


Figure 19. The Big Mac probe on a path with different cross traffic bit rates

in the probe gets an extra delay caused by cross traffic. Because of this, it is not possible to use the latency of the second packet pair to determine the number of relay-devices on the path.

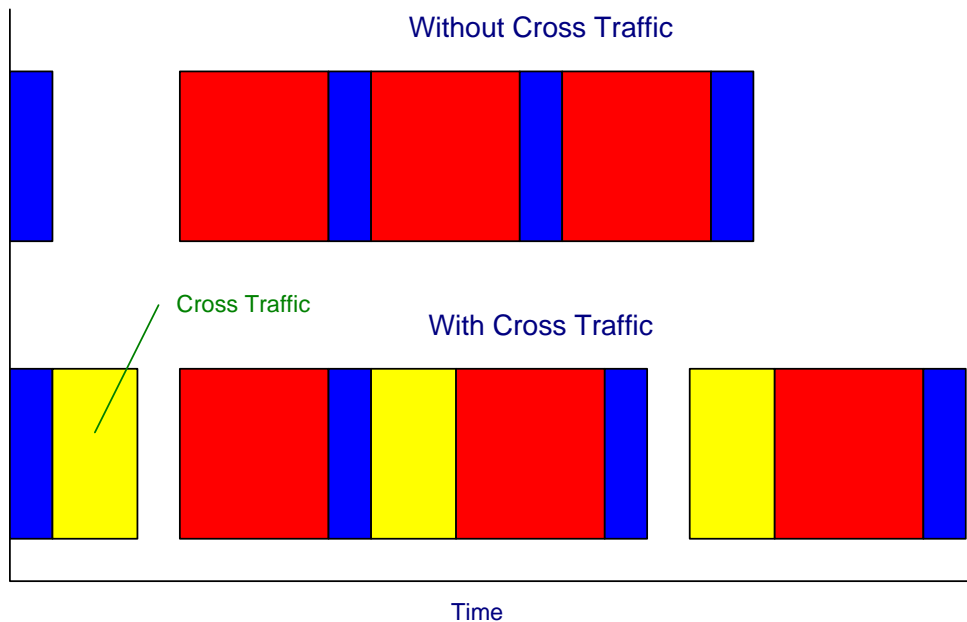


Figure 20. Possible scenario with a Big Mac probe without and with cross traffic

### 3.3.3 Available bandwidth

The set up as given in Figure 9 is used. The bandwidth is measured between device A and B. The link between switch X and Y is the shared link with a maximum bandwidth of 100Mbit/s. Devices C and D are used to generate cross traffic over the shared link. This means that the measured bandwidth between device A and B is 100Mbit/s without any cross traffic.

The cross traffic is generated using the Iperf tool consist of UDP packets. Also pathChirp and Pathload use UDP packets for their probes. All three tools do not use TCP because TCP's congestion control [16] makes it impossible to let an application sent a stream of packets with a predefined bit rate.

Both pathChirp and Pathload only use the relative delays between probe packets as measurement data. The results of pathChirp and Pathload with default options are given in Figure 21. Both tools look if there is any queuing delay between two consecutive packets. If there is any, then the bit rate of the packet is the maximum available bandwidth.

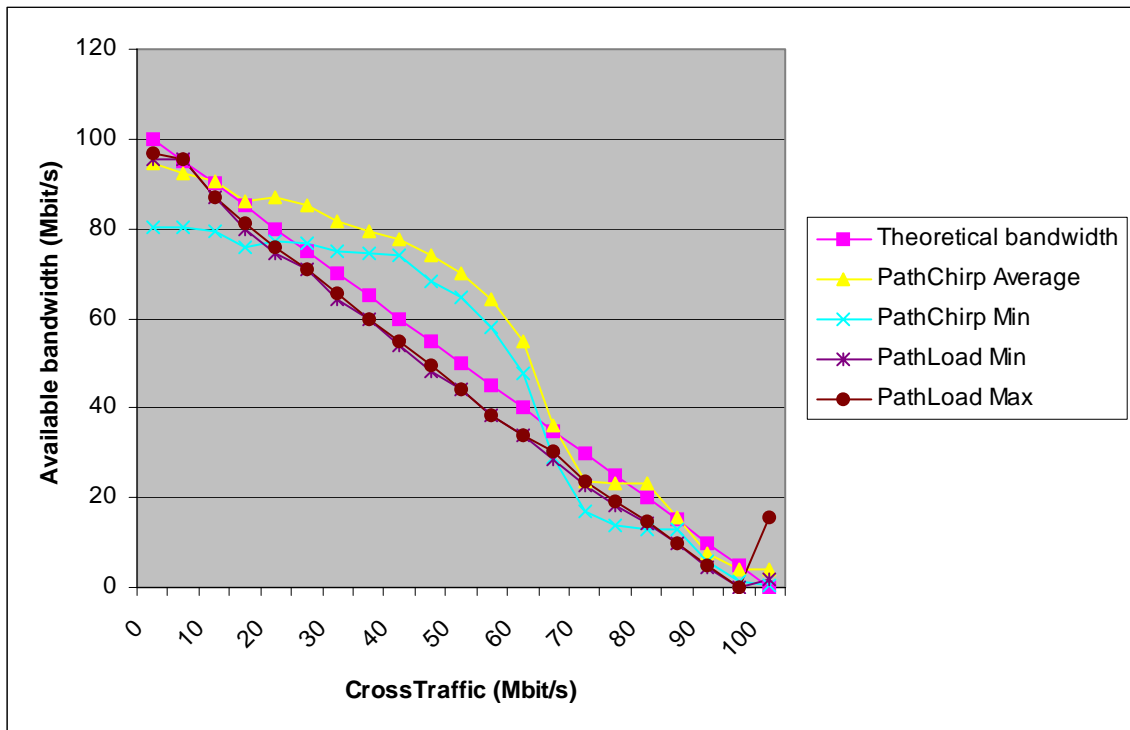


Figure 21. Comparison of pathChirp and Pathload

As seen the measured bandwidth by pathChirp is higher than the theoretical available bandwidth. Four things can explain this:

- The UDP traffic pushes away the other traffic including the Iperf traffic which was intended to decrease available bandwidth
- The packets involved in the measurement could be interleaved with the cross traffic packets. Therefore part of the cross traffic is not noticed in the measurement.
- Cross traffic causes buffering so that some packets do arrive after each other with small delay, which can be interpreted by pathChirp as a high bit rate.
- The speed of the home network is very high, which makes detecting queuing delays caused by the maximum available bandwidth harder because it is only a

difference of microseconds with the situation where there is only temporary cross traffic.

Pathload gives closer estimates although these estimates are mostly lower than the theoretical bandwidth. This is mainly because a packet train consists of packets with the same bit rate that makes it possible to filter out erroneous measurements.

We also tried to use both algorithms to measure the available bandwidth on an IEEE1394 connection between the two devices, but because of Interrupt Coalescence [17] (several packets are received before an interrupt is generated), these algorithms could not be used because the delay between the arrival times of the packets at the application is almost zero.

### **3.3.4 Conclusion**

Latency can be measured using multiple Big Mac probes to filter out erroneous measurements. This value can only be safely used to determine the number of store-and-forward devices on the path as long as there is no cross traffic and the contribution of individual devices is known. Furthermore the value of the latency also depends on the hardware used at the devices: we have seen that two 100 Mbit/s NIC's do have different behaviours. Another thing that is worth mentioning is that the interface adapters also introduce a delay caused by store-and-forward.

Available bandwidth is hard to measure. The most promising techniques temporarily congest a path to cause queuing at the link with the lowest available bandwidth. The congestion itself lasts for a couple of milliseconds, but the time needed to get a good estimate of the available bandwidth can be higher than 10 seconds because the algorithms start with packet train with a low bit rate and increase the bit rate step by step. We compared two algorithms. PathChirp uses fewer packets than Pathload but is therefore less robust against erroneous measurements caused by cross traffic.

Cross traffic has major impacts on measurements because it can cause queuing of packets at relay-devices. Therefore packets that are expected to have a certain arrival time difference can arrive sooner or later because of interleaving cross traffic. If this cross traffic is not taken into account, the wrong conclusions can be made on the bases of the measurements.

We have used Pathload to determine the available bandwidth of a path, because it is more accurate (although it needs more measurement traffic compared to pathChirp to make a good estimate). We also added an additional requirement that in the architecture of ISL there will be possibilities to add more property determination algorithms in the future or replace existing ones as better algorithms became available.

## 4 Architecture

In this chapter the architecture of ISL is discussed. It is explained how applications use the Berkeley Socket Interface, because ISL offers a replacement for this interface. Furthermore a high-level model of the complete ISL is given and each component of the ISL is discussed separately. We end this chapter with a discussion on how the ISL communicates with other devices and how the ISL components communicate with each other to perform its tasks.

### 4.1 Berkeley Socket Interface (BSI)

Current applications use the BSI to communicate over a network. To make it possible to switch between interfaces during a connection, ISL offers the application an API that equals the BSI.

To understand how the ISL works, it is necessary to know how the BSI is used by applications. Depending on the role of the application and the protocol the application wants to use, different function calls need to take place in a certain order (see Figure 22). For a short introduction to programming with the BSI read [20].

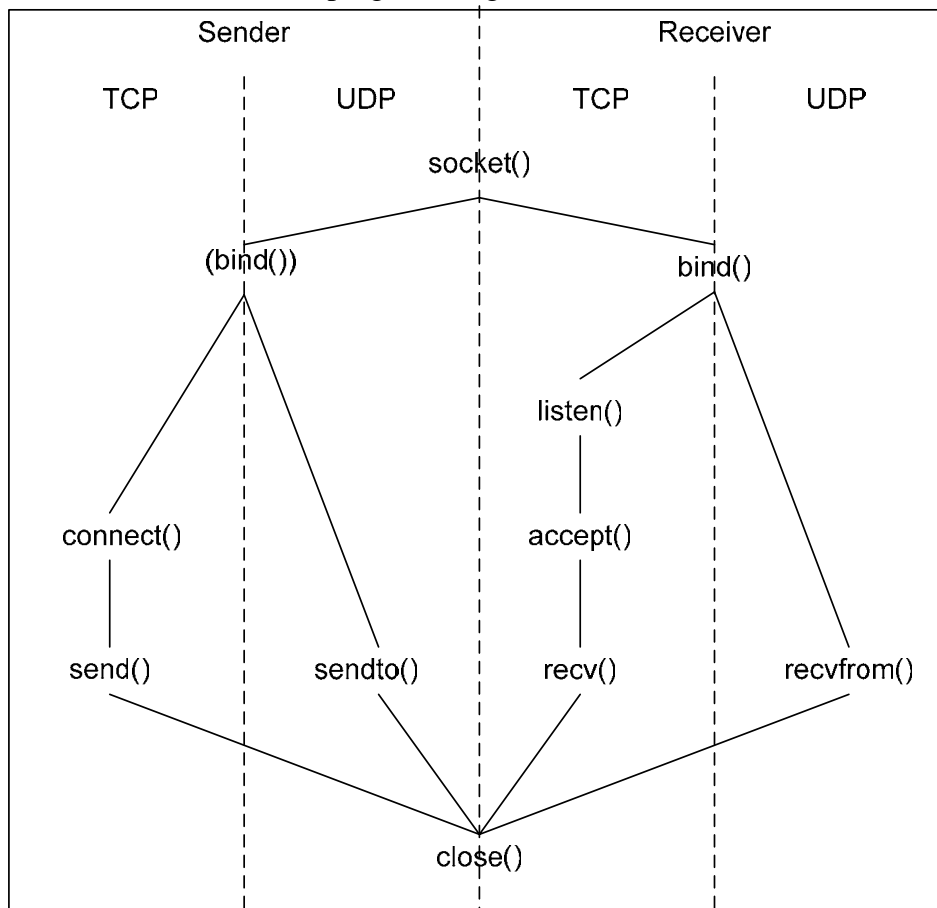


Figure 22. The use of the Berkeley Socket Interface

As seen in Figure 22 there are two roles being identified: a sender and a receiver role. Each of these roles supports the UDP and TCP protocol. The ISL emulates the working of the BSI by intercepting the calls to the BSI functions before they are

passed on to the Operating System. Applications need not to be modified because all calls to the BSI are intercepted by the ISL.

## 4.2 High-level model

As seen in Figure 1 the ISL is placed between the application and the BSI. The ISL also has direct access to the Operating System, which makes it possible for the ISL to use interfaces that cannot be used through the BSI. An alternative would be to put the ISL into the OS, but then the ISL is hardly portable to other Operating Systems.

The ISL itself is divided into a couple of components, each with its own tasks. Putting all the components together, we get the model describing the complete ISL given in Figure 23.

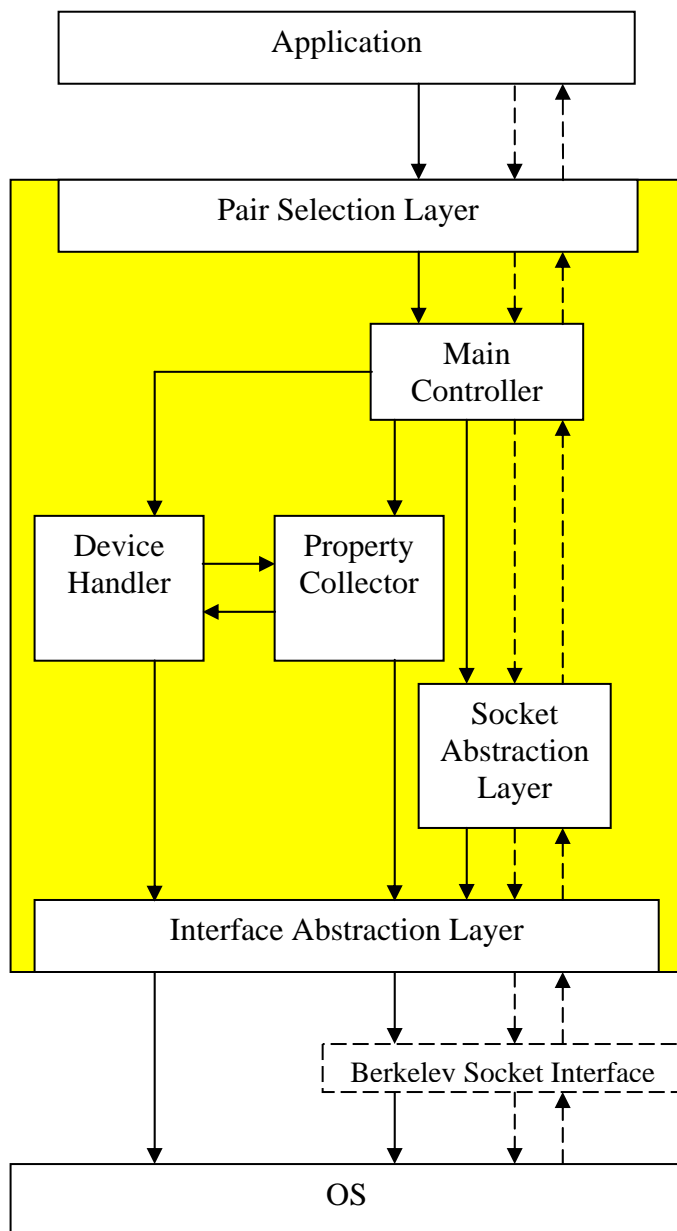


Figure 23. High-level model of the ISL, the solid arrows represent the control flow and the dotted arrows represent the data flow

The first component is the Pair Selection Layer (PSL). The PSL offers the BSI replacement to the application. Second there is the Main Controller (MC) that decides which interface pair to choose for a data stream. To give the ISL the possibility to change the local interface without losing already (by the application) established TCP/IP connections with another device, a layer is introduced that creates virtual addresses and maps them to the real interfaces (this is explained in section 4.3). This layer is called the Socket Abstraction Layer (SAL).

A device can have multiple network-interfaces. These interfaces can be of different types (e.g. Firewire, 802.3, 802.11a/b/g etc.) and each type of interface has its own methods to communicate using its own driver. To abstract from all these different types of interfaces, ISL uses an abstraction layer to talk with the different interfaces: the Interface Abstraction Layer (IAL). The IAL offers an interface that does not differentiate between different types of network-interfaces. Furthermore the IAL offers the functionality of switching between the network-interfaces during a connection. Therefore, it is possible to communicate with multiple interfaces of different types in a generic way.

Two other components are the Property Collector (PC) and the Device Handler (DH). The first is used to collect properties of interface pairs, and the second is used to communicate with other devices that use ISL. All the components are described in detail in the next chapter.

### 4.3 Handover

Whenever a better interface pair is found for a data stream, ISL hands the stream over from the 'old' interface pair to the new pair. This handover must be done seamlessly without loss of data. Meaning that if there is a TCP/IP connection between two ISL devices, it must not be necessary for the two devices to reconnect to each other. The problem of the handover of TCP/IP connections is called the TCP hand-off problem.

We considered various alternatives:

1. Our first idea was to realize a virtual TCP connection between the two devices which was implemented through a real TCP connection. Whenever the connection had to be handed over, the real TCP connection would be broken and a new real TCP connection would be established and the virtual connection would then work with the new real TCP connection. This requires a virtual socket number and a real socket number, which are generally different. Whenever another interface pair was chosen, the real socket was recreated and bound to the chosen interface. The ISL needs to translate the virtual socket number to the corresponding (new) real socket number. The problem with this method is that TCP/IP connections get disconnected and a new connection needs to be established. TCP/IP packets can get lost and the handover is not seamless: TCP/IP packets that are already passed on to the operating system get lost.
2. Another option is to use Mobile IP [24]. But Mobile IP has some disadvantages:
  - The infrastructure of the home network needs to be changed (not only the sender and receiver device but also the relay-devices).
  - All traffic is routed through a home agent and not directly between the sender and receiver (implying that there is always a small part of the

path which cannot be changed: I.e. the receiver cannot actually change because you keep the traffic to the home agent unless the home agent is the server).

3. The method chosen for ISL is a combination of the ideas of J.J. Lukkien, P.H.F.M. Verhoeven and Philips Research Laboratories. We create a ‘virtual interface’ for each socket in the form of an IP address (which we call the virtual address) that we assign to a real interface (with its own IP address, which we call the real address). In other words: one physical interface gets multiple IP addresses (the real address and one or more virtual addresses: Figure 24).

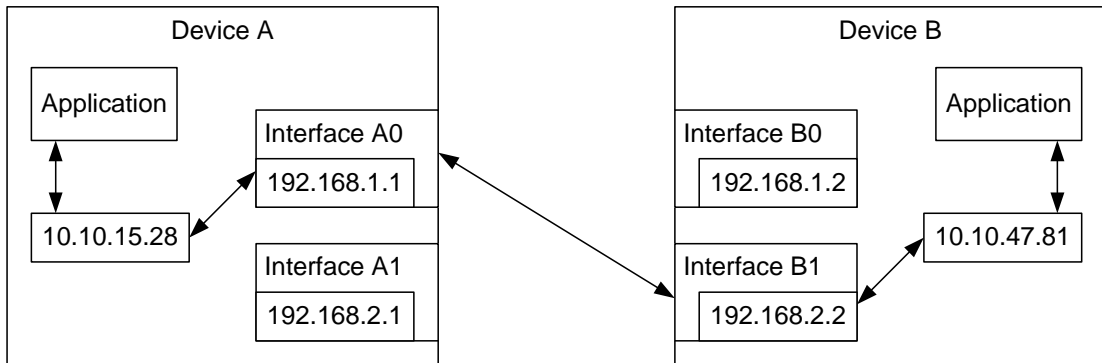


Figure 24. One interface with multiple IP addresses (a virtual and real address)

If another interface pair is chosen by the Main Controller, the virtual address is moved from one real interface to another real interface. Using this method the virtual IP address stays the same, but the MAC address corresponding to the (virtual) IP address changes. Therefore the changing of interfaces is done one level lower (level 2) than the level of TCP/IP (level 3), which in turn make it possible to do seamless handover of TCP/IP connections.

If the MAC address of an IP address changes, all the devices need to be informed of this change. The announcement is done by broadcasting an unsolicited ARP-reply packet containing the virtual IP address and the new MAC address Figure 25. This method of virtual addresses makes it possible to route traffic using a direct path between sender and receiver without changes to relay-devices. A downside of this method is that it is only applicable in a home network where each interface pair is within the same broadcast domain because the virtual addresses are from the same subnet. Also, the change of MAC address could be interpreted by the network as packet spoofing, triggering all kinds of alarms.

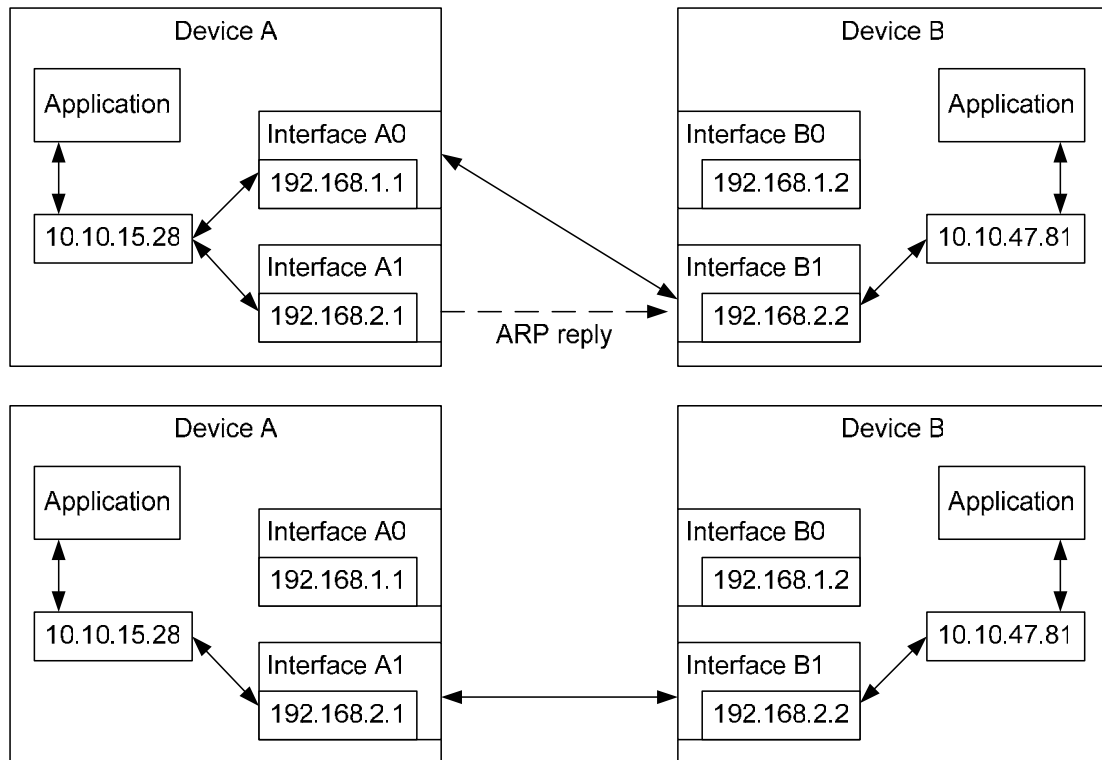


Figure 25. Switching between interfaces, Step1: Device A assigns virtual address to old and new interfaces and sends ARP reply; Step 2: Device B receives reply and updates ARP table, Device A removes virtual address from old interface.

It is also possible to move the real address to another interface, but then the previous interface of the real address is not available anymore because it does not have an IP address. Giving it a new real IP address is not an option, because it is not known which addresses are already in use and suitable for the device. Also DHCP can return the old address which is used by the other interface. Furthermore if the real address is moved to another interface it is not possible to change the interface pair for only one stream at a time: all streams are moved from one interface to the other. With our method we do not have this problem because we use a virtual address for each stream. Using virtual addresses and routing them through the real interfaces makes it impossible to use multiple interfaces for one stream and therefore combining the bandwidth of the paths. Combining interfaces for one TCP/IP connection is only possible if the transport layer is modified because the traffic needs to be divided over multiple paths and recombined at the other end. So another handover method is needed to accomplish this (we suggest a possible method in section chapter 7: using NAT between virtual addresses and real addresses).

In our current implementation the virtual addresses are taken from the range 10.10.x.y with subnet mask 255.255.0.0. The values for x and y are taken at random. This could allow streams to get the same address, but chances are very low. For our proof-of-concept implementation this method suffices.

## 4.4 Main components

In the previous section (4.2) a complete overview of ISL is given. In this chapter each component is discussed separately. Interactions between the different components and between ISL devices are given in sections 4.5 and 4.6.

### 4.4.1 Pair Selection Layer (PSL)

The purpose of the PSL is to offer an interface to the application. This interface contains the same function calls as the Berkeley Socket Interface (See `/usr/include/sys/socket.h` on a UNIX based system for a complete overview of each function with its parameters, or Figure 22 for a quick overview). PSL extends this interface with a couple of new function calls that enable new applications to communicate the properties of the data that they want to send (e.g. the bit rate of the traffic, the maximum allowed latency). Existing applications can be used without any modification to their source code and new applications can easily be written to use the extra functionality ISL has to offer. Read section 5.5 for information about how applications can use the ISL.

### 4.4.2 Main Controller (MC)

The Main Controller keeps track of all data streams that applications send and receive. It knows which interface pair is used for each stream and it chooses another interface pair if a new interface pair offers better performance than the current used path for the stream. A cost function is used to determine the cost of each possible path between the sender and receiver, making it possible to compare paths with each other. The path with the lowest cost is chosen.

The Main Controller only looks at possible interface pairs between devices and not at combining local interfaces together so that multiple interfaces are used for the same stream. Combining interfaces is a problem by itself and therefore left open for future research (see also section 4.3).

The Main Controller asks all the properties of possible interface pairs between the sender and receiver from the Property Collector. Depending on these properties it tries to make the best choice.

When an application wants to receive data, it listens at first on a by the application selected network-interface and waits for an incoming connection or incoming data on that interface. The receiver should listen on the IP address (assigned to a network-interface) selected by the application to let non ISL enabled devices connect or send data to the receiver. It depends on the used handover method if this is possible, see section 4.3.

Because the (receiver) application generally does not know who the sender will be, it is not possible for the receiver to determine the best pair of interfaces between the receiver and the future sender. The sender can determine the best pair of interfaces because it knows to which receiver it wants to connect and send data. Therefore only the sender chooses the best interface pair and not the receiver.

### 4.4.3 Device Handler (DH)

The Device Handler keeps track of which ISL enabled devices are connected to the network and handles all control communication between ISL-enabled devices (see

sections 4.5 and 4.6). The detection of ISL devices is done by listening on the network for broadcast messages sent by ISL devices; therefore it cannot detect non-ISL enabled devices. The ISL is actually a service offered by devices and the detection of ISL enabled devices can be implemented using a service discovery protocol (e.g. Simple Service Discovery Protocol (SSDP) [25]).

The DH has knowledge about the local and remote interfaces (e.g. the address and the type of each interface). The Device Handler sends and receives broadcasts containing the device identifiers and the information about interfaces. This information is given to the Property Collector.

#### **4.4.4 Property Collector (PC)**

The purpose of the Property Collector is to maintain a complete map of all interface pairs together with the properties of the interface pairs of which the device itself is part of. The Main Controller uses the Property Collector to get all properties of each possible pair of interfaces and to keep track of the status of each stream. The Property Collector gets information about the remote devices and their interfaces from the Device Handler. The Property Collector queries the IAL for the properties of the paths where the device itself is part of.

If an application wants to send data to a non-ISL enabled device, the Property Collector knows the receiver is non-ISL enabled because it has no information about that device (otherwise the Property Collector would have received information from the Device Handler). Because all devices need to announce themselves to each other there is some initialization time for each device before they can start sending data. Else it is possible that an ISL device does not know about another ISL device because the announcement information did not yet arrive or was lost. More about these ISL device announcements can be read in section 4.5.1.

We have only looked at property determination methods where both devices use ISL and not at methods for determining path properties between an ISL and a non-ISL enabled device. It should be possible to determine these properties using standard methods (e.g. ping to determine latency) or non-standard one-side only methods (e.g. let all local interfaces try to connect to the remote interface of the non-ISL device, the local interface that sets up the connection the fastest is probably the best interface to connect to the non-ISL device). So it is the responsibility of the Property Collector to get the properties of paths formed with a non-ISL device.

#### **4.4.5 Socket Abstraction Layer (SAL)**

The SAL assigns the socket numbers whenever an application wants to create a socket. The SAL uses the IAL to create a virtual address for each socket and assigns these virtual addresses to a real interface (see section 4.3). The SAL keeps track of all these socket numbers and knows which virtual address belongs to which real interface.

#### **4.4.6 Interface Abstraction Layer (IAL)**

Some types of interfaces offer extra functionality to communicate with interfaces of the same type. A Firewire interface has for example the feature to reserve certain bandwidth for a stream of data in contrast to an 802.3 interface. Also not all interfaces use IP to communicate. That is where the IAL comes in. It gives the ISL the

possibility to talk to all kinds of different interface types without knowing the difference between them. The IAL has knowledge about the local interfaces, their drivers and how to communicate with them. The IAL is used to send and receive data and is used to get the properties of a path. The IAL makes no difference between the sending and receiving of ISL control messages and application data.

## **4.5 External communication**

In this section all communications between ISL-enabled devices is discussed. The external communication is used for the following tasks:

- Detection of ISL devices
- Gathering interface information
- Determination of path properties
- Selection of an interface pair
- Sending and receiving data
- Synchronization between ISL devices

These tasks are discussed in more detail in the subsections of this section.

### **4.5.1 Detection of ISL devices**

There are different methods for service discovery in a network (e.g. SSDP [25] or SLP [26]). Most methods are a combination of unicast and broadcast messages. To keep it simple (because our focus is on the property determination and architecture of ISL), we only use periodic broadcast messages to discover whether a device has ISL or not. The downside of this method is that it adds an extra delay when a new device is added to the network. This extra delay equals the interval at which the broadcast messages are sent. The broadcast can also be used for a kind of keep-alive message on the network and if the broadcast message contains information about the network-interfaces of the device, this information exchange can be done before any application tries to communicate with another device.

Every ISL device sends a broadcast message, containing the identifier of the device, the amount of interfaces together with the properties of the interfaces (e.g. the address and type of each interface). This broadcast is sent on all interfaces that use IPv4. This is done because an ISL device can belong to multiple broadcast domains (see requirements 7 and 14). The name of the device is a Universal Unique Identifier (UUID) that is used for ISL to uniquely identify each device. Therefore ISL knows how many ISL devices are connected to the network and how many interfaces each device has together with the properties of the interfaces.

When a device receives such a message, it checks if it already has knowledge about the existence of the sender. If not, it will add the device to the list of known devices. The device also stores a timestamp of the last received broadcast of each device, so that ISL can detect whether a device is still connected to the network or not. If a device has not received a broadcast from another ISL device for a certain amount of time, it is presumed that the device is not operational or connected to the network anymore. ISL does not keep track on what interfaces the same broadcast message is received. It does however store the number of the interface on which the first broadcast message is received and uses this interface for further ISL communication

with the other device. An ISL device does not send a response back to the sender of the broadcast message.

Before an application can communicate with another device, ISL needs to know if the other device also uses ISL or not. Therefore the initialization time of a newly connected ISL device equals the interval time at which broadcasts are sent.

#### **4.5.2 Gathering interface information**

As discussed in the previous section, the broadcast message sent by all ISL devices contains the properties of their interfaces. Whenever an ISL device receives a broadcast from another ISL device, it knows the type and address of the interfaces of the other device and there is no need for extra communication.

Another option was that a device queries another device about its interfaces using unicast messages. But this would involve extra communication between ISL devices before an application can start its own communication with the other device, which makes this method quite inefficient. The broadcast message that is used for identifying ISL devices is very small, so the information about the interfaces can be added without extra overhead in communication. This information consists of the IP address and the type of the interface (e.g. 802.3, Bluetooth etc.).

#### **4.5.3 Determination of path properties**

The Property Collector periodically collects the properties of the paths of which the device is part of. A property is determined between two devices: the device that initiates the property determination is the sender and the other device is called the receiver. The sender initiates the determination of a path property by sending a message to the receiver telling it which property is going to be determined with which interface pair. The receiver is now able to prepare the local interface that is involved in the path property determination and send a message back when the receiver is ready for the measurement to begin. The complete sequence of communication steps is displayed in Figure 26.

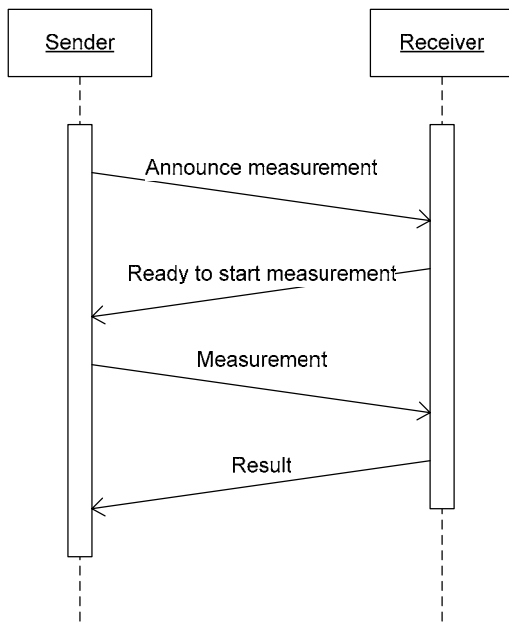


Figure 26. Sequence of communication steps for executing a measurement

The internal communication between the components of the ISL is discussed in section 4.6.

#### 4.5.4 Selection of interface pair

For each interface pair between two devices, a score is calculated using a cost function and the best pair is chosen. This process takes place at the sender; therefore, the receiver does not know which interface pair is chosen by the sender. The sender however does not know to which remote interface the remote virtual address is mapped, so it is not possible to send a simple message to the receiver telling it to change a certain mapping. To let the receiver know which local interface to use, the sender must send a message to the receiver containing the receiver's virtual address (which identifies the stream to the receiver) together with the selected interface (see section 4.6.2).

This message gives the receiver the information needed to change the assignment of the virtual address to the real interface. After the receiver has changed the mapping, it sends an acknowledgement back so that the sender can also change the mapping. See Figure 27 for a sequence diagram describing the interface pair switching process.

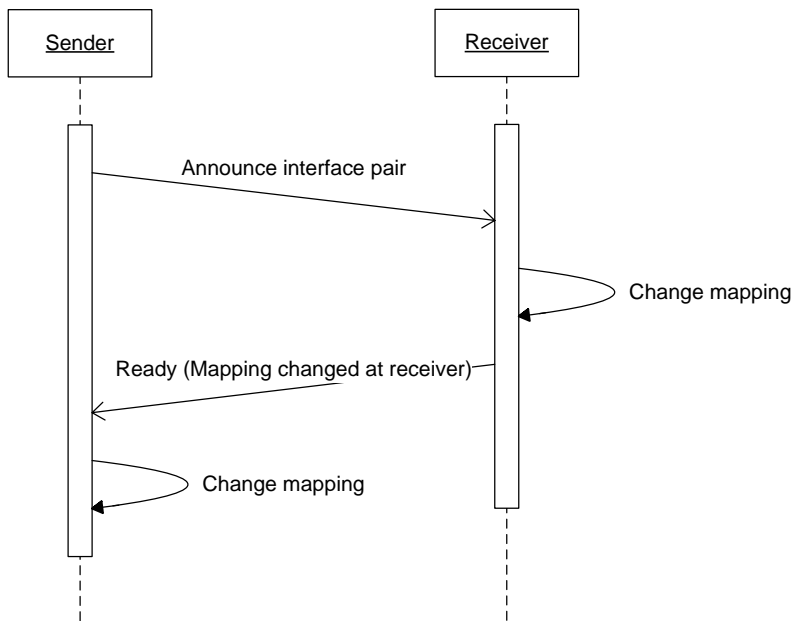


Figure 27. Sequence of communication steps for changing an interface pair

The sender lets the receiver to change the mapping first because if the receiver is not able to change the mapping, the receiver can inform the sender without the TCP/IP connection getting lost. When the receiver has changed the mapping TCP/IP makes sure that packets do not get lost by retransmitting them until the sender has also changed the mapping. UDP packets however do get lost. Another option would be to let ISL block all the traffic, but ISL has no control over all packets that are already passed to the Operating System before the change of the mapping, so this option is not chosen.

#### 4.5.5 Sending and receiving data

The actual sending and receiving of application data is done by the IAL. From the application's point of view, the ISL is transparent and the sending and receiving of data is done through the Berkeley Socket Interface.

To comply with the standards of the different kinds of traffic, no additional header is added to the data (requirement 10). But when an ISL device wants to wait for incoming data, it first creates a virtual address and maps it to a real interface. This mapping is then broadcasted to all other ISL devices, so that they have knowledge of this virtual address. If an application wants to send data to another ISL device, the ISL changes the destination address given by the application to the receiver's virtual address.

The sender also creates a virtual address, but it does not broadcast the information about the virtual address to all other devices because the virtual address will only be used to connect to another (virtual) address and not for receiving incoming connections. See Figure 28 for the sequence of communication steps.

A downside of this method is that it is necessary for a sender to be already connected to the network before the receiver starts listening. Otherwise the sender could miss the

broadcast message containing the virtual interface information and therefore the sender does not have the information needed to connect to the receiver: it simply does not know about the virtual interface's existence. There are two possible solutions to this problem. The first is to add the information about the current existing virtual interfaces to the periodic keep-alive broadcast message. The second is to add some ISL specific protocol between the two ISL devices so that the virtual interface information can be exchanged between them during a set up of a connection. Because we merely want to offer a proof-of-concept implementation we do not address this problem in our implementation.

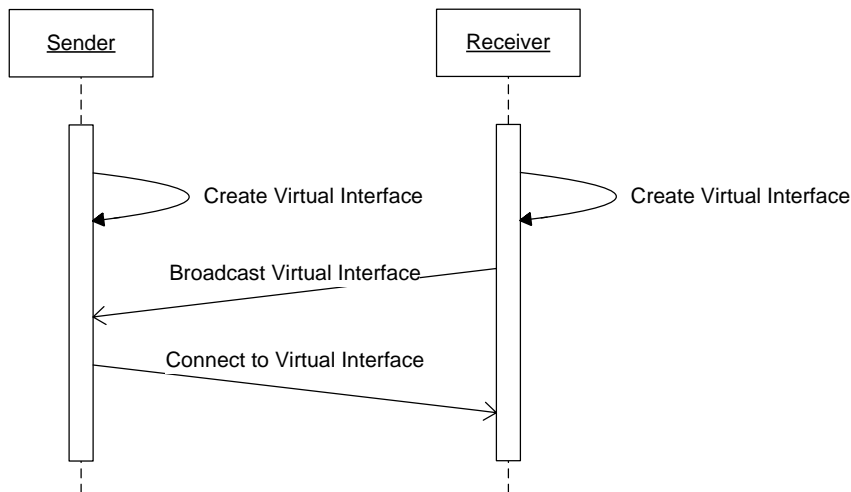


Figure 28. Sequence of communication steps for making a connection between a sender and receiver.

#### 4.5.6 Synchronisation between ISL devices

When a property is determined, the value of the property is a kind of snapshot of the path's condition at a certain moment in time. If a device adds a stream to the network, the measured value could be inconsistent with the current network condition. This could introduce a live lock: consider a device sending a stream over a not-optimal path, so another path is selected, which causes another stream of another device being sent over a not-optimal path. Both ISL devices could keep switching between paths, so we need a method that lets the ISL devices converge to a stable state. To solve this problem, determining properties and setting up a path should together be one transaction that can only be performed by one ISL device and not by multiple ISL devices simultaneously. This however does not solve the situation where a legacy device starts a data stream between the property determination and the starting of a data stream between two ISL devices. In this situation the ISL device has to change the path only once, because the legacy devices do not change their 'selected' path, so the 'live lock'-problem is not an issue here.

Requirement 15 states that it should be possible to synchronize all ISL devices to get a stable state. There are different methods known in the literature to perform an atomic transaction using distributed mutual exclusion [21]. The rest of this section discusses these methods.

There are two mainly two methods: the first method is to let all devices select one device as the coordinator and let each device ask the coordinator for permission to perform a transaction, the second method is to let all devices form a token ring, the device that has the token can do a transaction. Both methods suffer from the problem that not all devices can communicate with each other because devices can belong to multiple broadcast domains (requirement 14). Therefore it is necessary that each broadcast domain has one coordinator or one token.

It is not difficult to elect a coordinator or set up a token ring in one broadcast domain, because each device has a unique ID the bully or ring algorithm can be used for election [21].

There are a few problems when a token ring is used to offer distributed mutual exclusion to perform a transaction:

- The network gets flooded by the tokens because a device only rarely needs to determine properties and change a path.
- A device can only perform a transaction when it has the tokens of each broadcast domain the device belongs to. This can easily result into deadlock when two devices both belonging to two broadcast domains each have one token and want to do a transaction. It can also result into a live lock when there is a time out on how long a device can hold a token and the two devices keep alternating the two tokens.

The coordinator method does not have the first problem. But it does have the second problem: a device needs to have permission from all the coordinators. Now the same problem occurs when two devices belong to the same two broadcast domains and each device only has permission from one of the two coordinators. This can only be solved when there is only one coordinator in the entire network, which is only possible if devices act as a bridge between broadcast domains to let permission requests travel from one broadcast domain to the other broadcast domain.

This is however very sensitive to failure, because when a device crashes (or is disconnected) the network can be split up into two separate broadcast domains that have no bridge between them. However if an election is held each time a network change occurs (e.g. a device is connected or disconnect), then a network change should not be a problem (assuming that algorithms are used to handle pending permission requests).

To guarantee correctness of measurement values at the time a device wants to start a stream, synchronization between the ISL devices is needed. A token ring is not suitable because a device only rarely wants to select another path, so the coordinator method must be used. However because this allows the ISL devices to act as bridges between broadcast domains, it would increase the complexity of our proof-of-concept implementation. Because we only want to prove that it is possible to offer path selection to an application by only looking at the properties of a path, we have not included such a distributed mutual exclusion algorithm in our implementation.

## **4.6 Internal communication**

In section 4.5 we described the communication that takes place between two devices. But inside a device there is also communication between the different components. In this section we look at the internal communication steps that need to be done to perform the following tasks:

- Initialization of the ISL device
- Handling control messages
- Property determination
- Interface pair selection
- Sending application data
- Receiving application data

The internal communication steps are given in the form of sequence diagrams. Each solid arrow represents a function call and the dotted arrows represent a return value. Return values that only report if a function call was successful are not displayed in the sequence diagrams.

#### 4.6.1 Initialization

Before ISL can do anything, it needs to initialize itself. The component Main calls the ‘initialize’ method of the Device Handler, the Property Collector and the SAL. This causes each component to initialize its private variables and creates the necessary threads for the different tasks of each component.

The Property Collector, the SAL and Main only initialize variables and create the necessary threads (more about the different threads in section 5.4). The Device Handler also performs a couple of initialization steps using the other components:

- Get information about the local interfaces
- Start broadcasting presence messages periodically
- Start listening for incoming ISL control messages

To get information about the local interfaces, access to the operating system is required. The IAL has access to the operating system, so the Device Handler asks the IAL for the information about the local interfaces. The Device Handler passes this information on to the Property Collector (Figure 29).

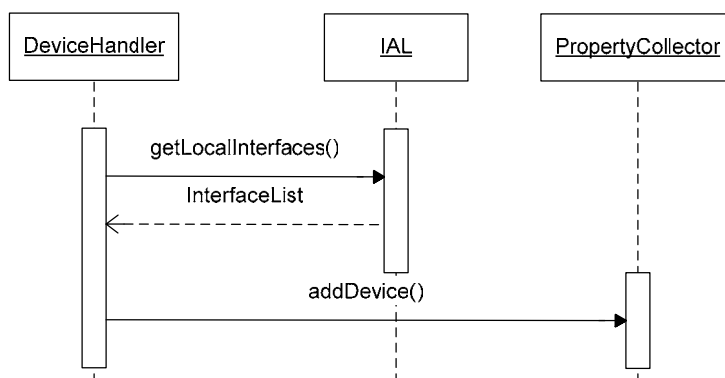


Figure 29. Getting information about the local interfaces

The Device Handler sends periodic broadcasts to distribute its availability to the other devices on the network. The IAL is used to set up a socket and send the broadcast messages (Figure 30), more on the different messages in section 4.6.2.

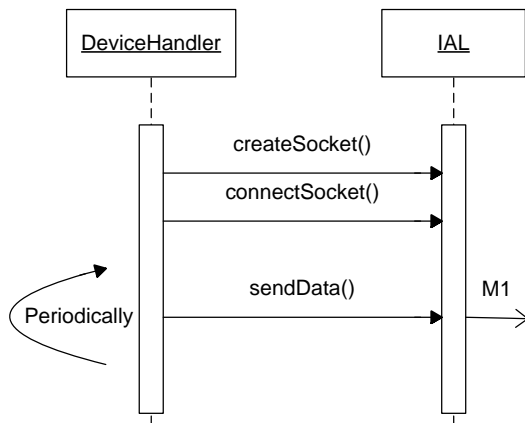


Figure 30. Sending periodic broadcasts

The last initialization step of the Device Handler is to set up a socket for incoming ISL control messages. After an ISL control message is handled by the Device Handler, it waits for another incoming message (Figure 31).

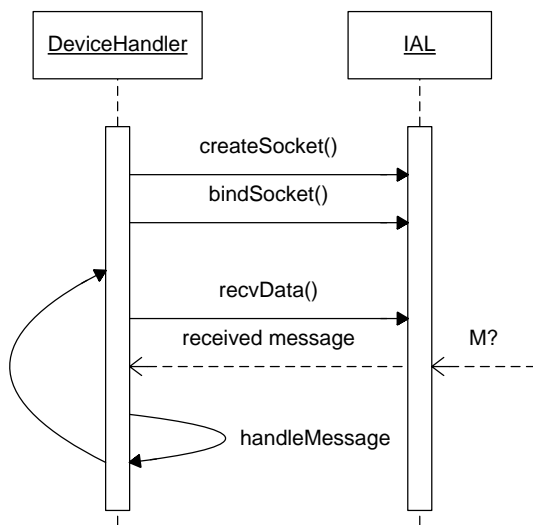


Figure 31. Receiving ISL control messages

#### 4.6.2 Handling messages

The Device Handler receives ISL control messages from other ISL devices. Depending on the type of the message, certain actions need to be performed by ISL. The different ISL control messages are given in following table.

Name	Contains	Description
M1	<ul style="list-style-type: none"> <li>• Device ID</li> <li>• Interface information</li> </ul>	Broadcasted announcement of presence of a device with its interfaces
M2	<ul style="list-style-type: none"> <li>• Real address</li> <li>• Virtual address</li> </ul>	Broadcasted announcement of a newly created virtual address
M3	<ul style="list-style-type: none"> <li>• Virtual address</li> <li>• New local interface</li> </ul>	Announcement of an interface pair
M4	<ul style="list-style-type: none"> <li>• Type of measurement</li> <li>• Remote interface number</li> <li>• Local interface number</li> </ul>	Announcement of a measurement

If the Device Handler receives M1, it passes on the information to the Property Collector so that both the components have the ability to store the information or know that an already known device is still connected to the network.

When message M2 is received, the Device Handler passes this information on to the IAL which can use this information to translate real addresses (from the application) to the virtual addresses (more on this in section 4.6.5 and 4.3).

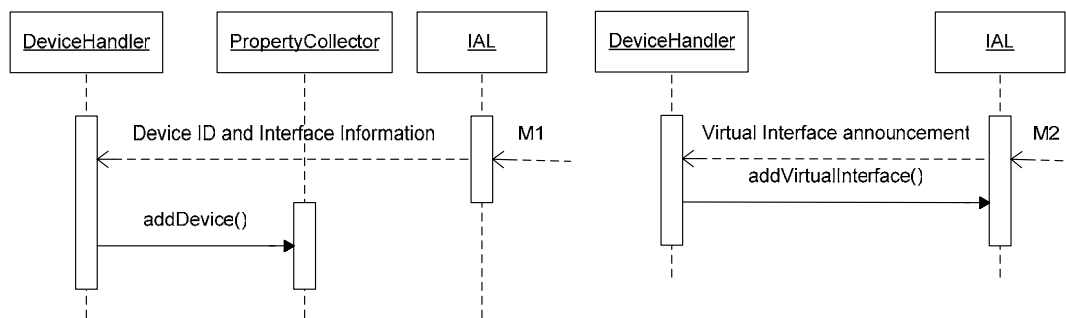


Figure 32. Handling of message M1 (left) and message M2 (right)

If a stream of data is running between two devices, an interface pair is announced to the receiver of the data stream using message M3. The Device Handler of the receiver uses the IAL to move the virtual interface to the newly selected interface. When this is done, it sends a message back to the sender so that the sender can also change his local interface (more on this in 4.6.4).

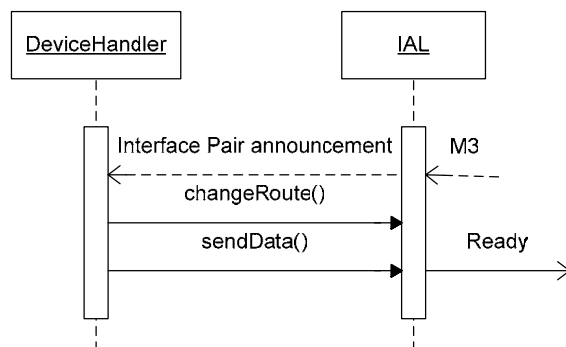


Figure 33. Handling of message M3 (change local interface)

When a device (the sender) wants to determine a property of a path, it sends a measurement announcement to the device on the other end of the path (the receiver). This announcement contains the type of measurement that is going to take place and the name of the interfaces involved in the measurement. The receiver prepares itself for the measurement and sends a message back when it is ready to start the measurement. Because the device receives an announcement, the device knows that it has the receiver role in the measurement. The measurement can now start. This measurement is handled by the IAL. When the measurement is done, the receiver sends a message back containing the result of the measurement (Figure 34). More on the property determination at the receiver side can be found in section 5.4.6.

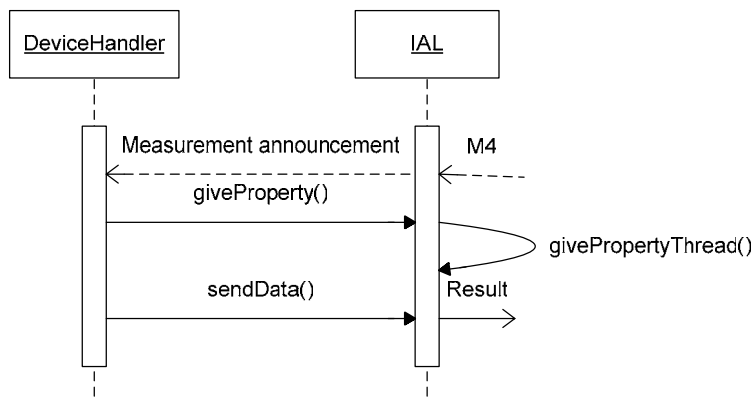


Figure 34. Handling of message M4 (getting ready for a measurement)

Both M3 and M4 messages are handled as a kind of remote procedure call mechanism (RPC [22]) messages. This means that the sender of the messages blocks until the receiver calls the required functions and sends the result back before the sender can continue.

### 4.6.3 Property determination

In section 4.6.2 we discussed how a measurement announcement is handled on the receiver side, here we give the sequence diagram of the sender side. The properties are collected by the Property Collector. The Property Collector gets the properties from the IAL, which determines the properties. The Property Collector keeps a table containing all the detected devices with their interfaces and periodically determines the properties of the paths of which the device itself is part of. The Property Collector tells the Device Handler that it wants to determine a path property. The Device Handler makes sure the other device (the receiver) knows this. The Property Collector calls the IAL to determine the property, and the IAL returns the value of the property that is received from the other device. The Property Collector stores this property value in the graph.



### 4.6.5 Sending data

An application uses the interface of the PSL to send and receive data using the steps given in Figure 22. The PSL translates the BSI calls to calls to Main. Main passes the calls on to the SAL that keeps an administration of all the created sockets and their state. The SAL also creates the necessary virtual addresses for each socket using the IAL.

As seen in Figure 22 there is a difference between the function calls when UDP or TCP is used. First we give the sequence diagram of function calls when an application uses TCP.

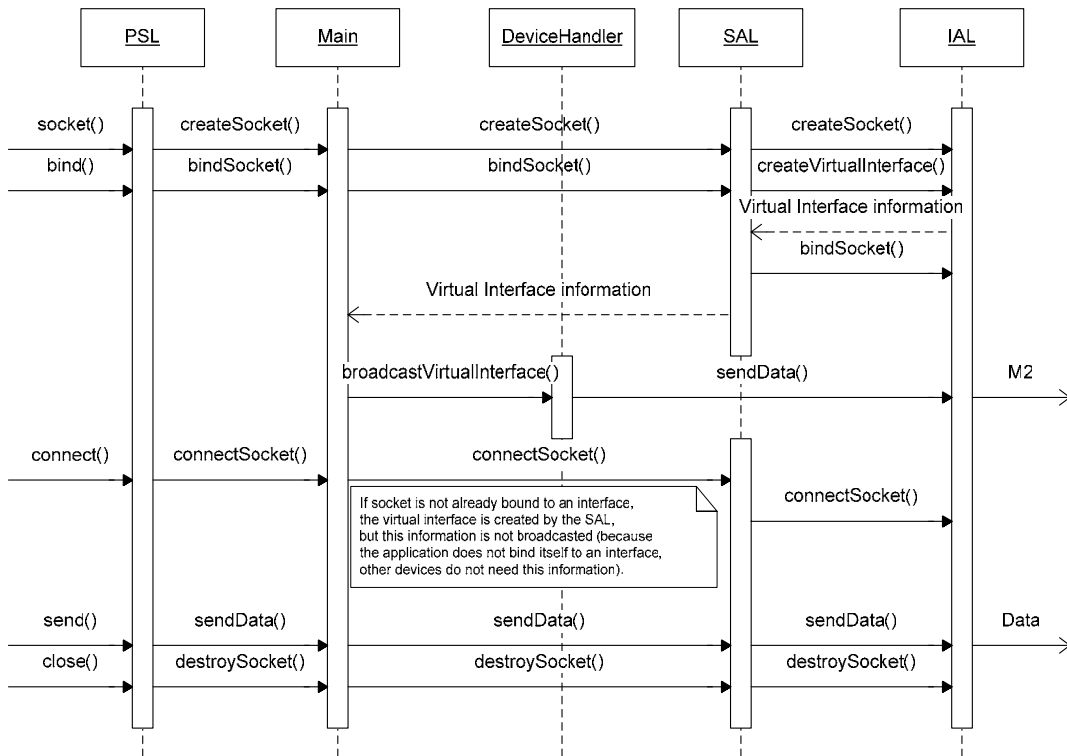


Figure 37. Sender application using TCP to communicate

Because the `bind` call is optional when sending data, the ISL does not broadcast the created virtual address when `bind` is not used.

UDP is connectionless, so the `connect` function of the PSL is not called. It is however possible to have connected UDP sockets (see [20]). So whenever an application uses the `sendto`, we connect to the address that is an argument of `sendto`. This connecting does not add extra overhead because no actual connection is made but only the remote address is stored so that when `send` it used, the operating system knows to which address the data must be sent. This is done to keep the same sequence of function calls as with TCP so that all the components except for the PSL do not make a difference between UDP and TCP traffic.

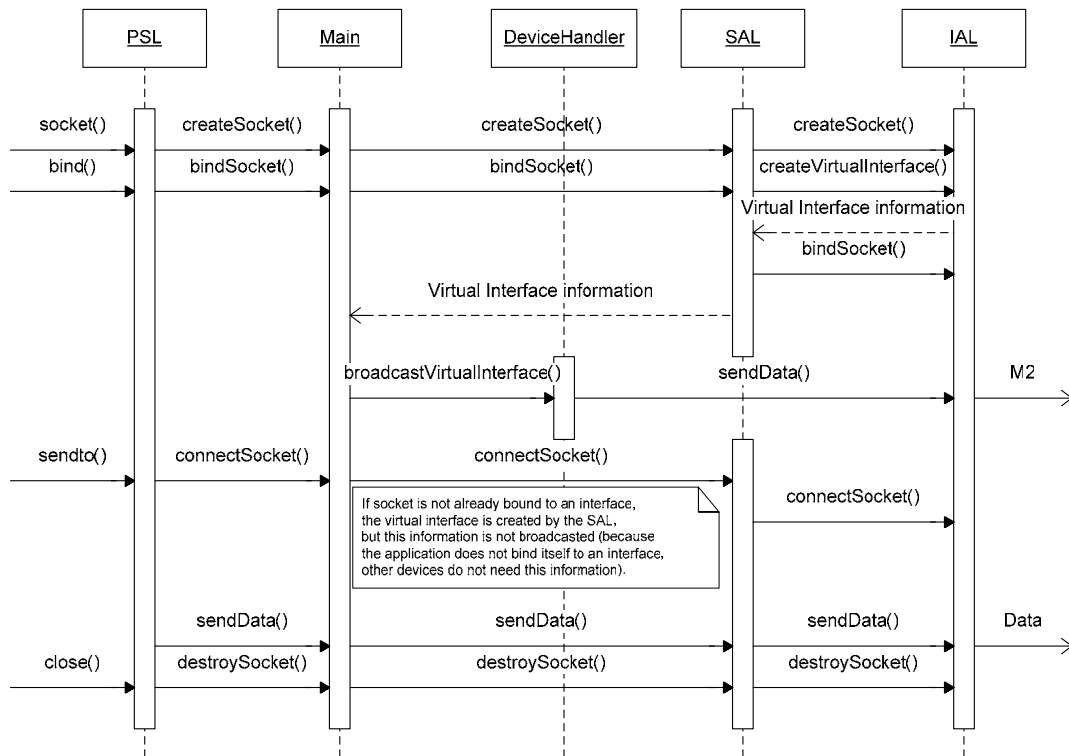


Figure 38. Sender application using UDP to communicate

The IAL has knowledge of all virtual interfaces in the network together with their original addresses (combination of IP address and port number). If a device wants to connect or send data to another device, the IAL translates the address it gets from the application to the corresponding virtual address.

#### 4.6.6 Receiving data

The receiver must always bind to a local address and port number. Therefore `bind` is always called. Applications normally have the possibility to listen to all interfaces (using the BSI to `bind` to `0.0.0.0`) but to have full control of which interfaces are used, we create a virtual address and bind a socket directly to that virtual address. The `create` and `bind` operations are the same as for the sender application given in section 4.6.5. All other operations ripple from the PSL through the Main and SAL to the IAL.

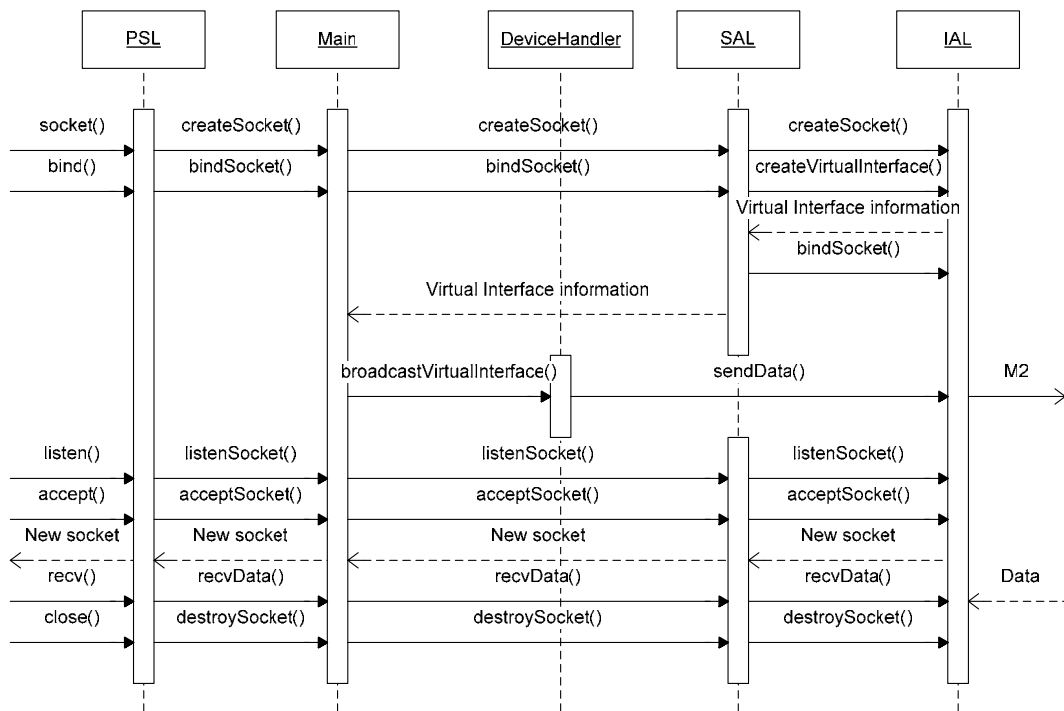


Figure 39. Receiver application using TCP to communicate

The UDP case is described by a more simple sequence diagram because the listen and accept functions are not used here.

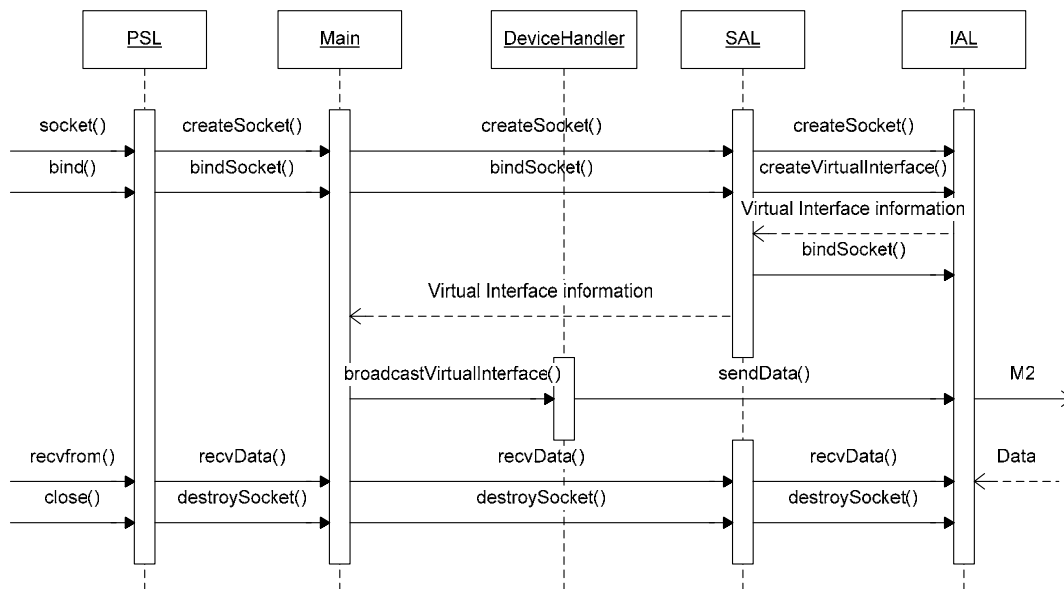


Figure 40. Receiver application using UDP to communicate

## 4.7 Conclusion

Using the architecture as displayed in Figure 23 it is possible to separate the application's tasks from the ISL tasks so that both can do their work separately

(requirements 6 and 11). Furthermore by implementing the ISL as a layer it is easily portable to other kernel versions and easier to develop and debug. Because the ISL offers the Berkeley Socket Interface to the application, applications do not require modifications before they can use the ISL (requirement 12).

The current handover method (4.3) is suitable in a home network situation. The TCP/IP connections are preserved and it is possible to change the path of each stream separately. But the legacy support is not optimal yet because legacy devices have no knowledge about the virtual addresses. An improved method is needed as discussed in chapter 7 to fully support legacy devices.

## 5 Implementation

In this chapter implementation details are discussed about ISL. Information is given about the environment for which ISL is implemented. We explain how the Main Controller chooses an interface pair and how the handover between interface pairs takes place. At the end we discuss implementation details about the different ISL components (e.g. the function of the different threads in each component) and it is explained how an application can use ISL.

### 5.1 Linux

We have chosen to develop ISL in C++ for the Linux operating system. We have chosen for Linux because ISL requires access to low-level parts of an operating system. Also implementation on Linux was a requirement of the project (requirement 8) as other parts of the project were working on Linux. Since the source of Linux is available, that OS is chosen. The techniques that ISL uses could also be applicable to other OSes.

To support the handover of connections between interfaces (see section 4.3), we use the third party utilities ‘`ifconfig`’ (assigning virtual addresses to real interfaces) and ‘`arping`’ [23] (for sending ARP replies). Furthermore the utility Pathload is used to determine the available bandwidth of a path. These tools are called from inside the ISL using the `system` statement, which also has a negative effect on the performance of ISL. Better performance can be given when these third party tools are integrated into the ISL.

The current implementation of ISL is a proof-of-concept implementation, meaning that it is only there to show that our idea of offering interface switching to an application is possible. The performance of the current implementation could be better (see chapter 6). We believe that the ISL should be integrated into the Linux kernel, which will offer a major performance boost to the ISL.

The entire ISL is written in C++ which makes it portable between different OSes (requirement 9). Only parts of the IAL need to be rewritten when it is ported to another OS, because the IAL contains OS specific parts (e.g. the access to the interfaces and the handover between interface pairs).

### 5.2 File descriptors and sockets

The `socket` function call of the Berkeley Socket Interface creates a socket and returns the descriptor of the created socket in the form of a number (which we call the socket number). The socket number is actually a Linux file descriptor. Not only the functions of the BSI can use these socket numbers, but also other functions like file access functions (e.g. `select`, `read`, `write`; see their corresponding manual pages for more info). The ISL only offers replacement functions for the BSI and not for all these other functions (Figure 41) because we only need to intercept the function calls which are used for the actual communication with other devices (e.g. setting the remote and local address, establishing connections etc.).

The ISL creates real sockets for every call to the `socket` function call by the application and returns the actual file descriptor number of the created socket. Therefore, the socket numbers returned by the ISL are backwards compatible with all other functions that use file descriptors.

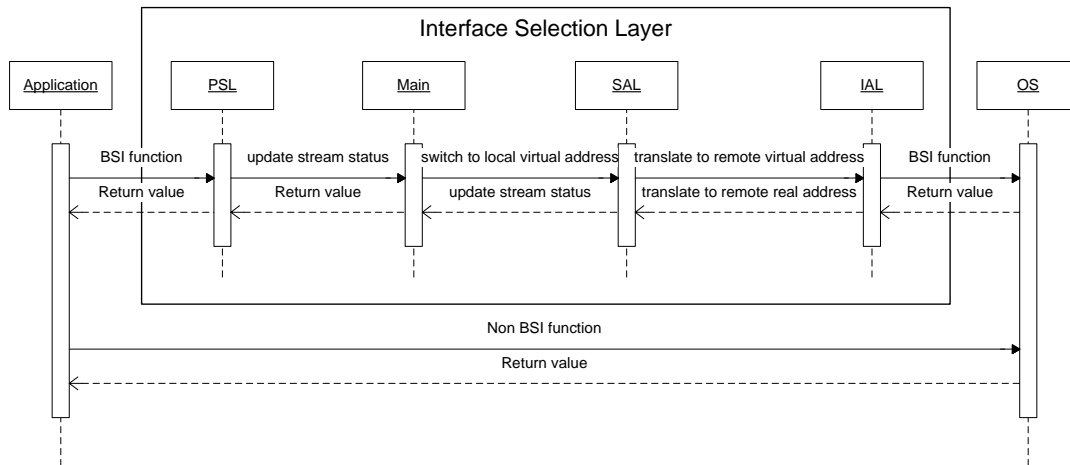


Figure 41. Communication between an application and the OS with ISL

As seen in Figure 41 the ISL only intercepts calls to the Berkeley Socket Interface. All other calls go directly to the operating system. If a BSI call is intercepted, the PSL passes the call on to the Main Controller, which updates the status of a stream (e.g. if it is being created, bound to an interface, sending or receiving data). The Main Controller passes it on to the Socket Abstraction Layer, which (if necessary) creates a virtual address for the socket and binds a created socket to the virtual address. The Interface Abstraction Layer translates the real remote address to the virtual remote address and then establishes connections or sends data to the virtual address using the operating system. If a message is received from a virtual address, the IAL translates the virtual address to the remote address before passing the message to the SAL. The SAL passes the message to the Main Controller so that it can update the status of the stream, and the MC passes it back through the PSL to the application.

### 5.3 Determination of the best interface pair

The Property Collector has a list of properties for each path. It tries to keep this property list up to date by periodically determining the properties of the paths the device is part of. The Main Controller uses the Property Collector to get the properties of the possible paths for the data streams the device itself has initiated. For each active stream the Main Controller periodically compares all possible paths with each other by using a cost function to determine the cost of each path. Because of the limited time for the project, we kept the cost function very simple: the path with the lowest transmission time is the path with the lowest cost and thus this path will be selected.

### 5.4 Components

In section 4.4 we discussed the different tasks of each component. To make ISL more efficient, we used different threads in the components to do certain tasks in parallel. In this section we discuss these threads and other implementation specific details.

One thread is always there: the application's own thread. This thread uses the PSL, Main, SAL and ISL (Figure 42).

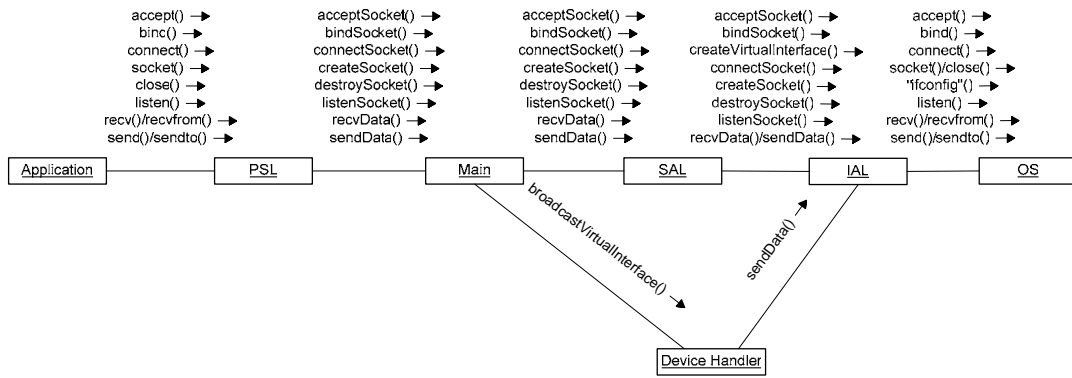


Figure 42. The application's thread

### 5.4.1 Pair Selection Layer (PSL)

The PSL is only used by the application thread; it acts as the interface to the application. It contains the Main Controller as a static global object. The PSL is not an object, but a list of public available functions so that it resembles the BSI.

### 5.4.2 Main Controller (MC)

This object uses the Singleton pattern (Figure 43, [27]) to ensure there is only one Main Controller associated with the application that is using ISL. During tests we found out that each application thread or process instantiates its own ISL. In the future the ISL should be made a singleton per device and not per process or thread.

Singleton
-instance : Singleton
+Instance() : Singleton
-Singleton()

Figure 43. The Singleton pattern

The Main Controller consists of one thread: the stream handler thread. This thread is responsible for keeping track of all running streams by periodically asking all the properties from the Property Collector and selecting the best path for each stream. The new path is announced with the Device Handler. When the new path has been successfully announced, the Main Controller uses the SAL to change the path (Figure 44).

Because the selecting of an interface pair is done in a separate thread (separate from the application's thread), it is possible for the application to immediately start sending and receiving data without a delay needed for the ISL to choose the best interface pair.

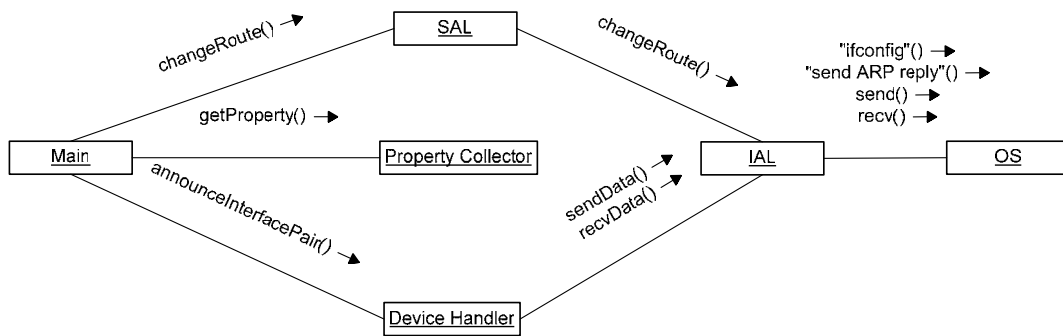


Figure 44. The stream handler thread

### 5.4.3 Device Handler (DH)

The Device Handler consists of two threads: one thread for broadcasting the periodic messages containing the local interface information and device ID, and one thread for receiving and handling ISL control messages.

The first thread uses the IAL to get the interface information and for sending the broadcast message (Figure 45). The second thread uses the IAL to communicate with other ISL devices and the Property Collector to keep the network graph up to date (Figure 46).

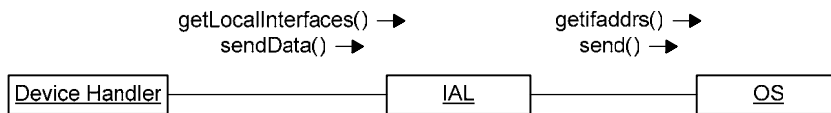


Figure 45. The broadcast thread

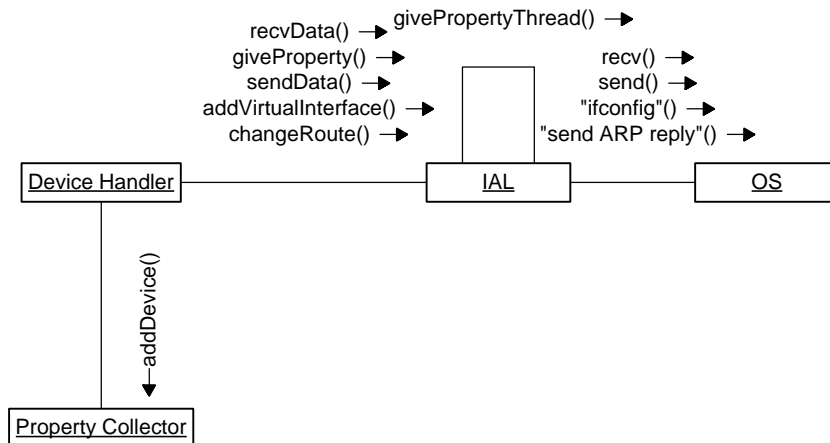


Figure 46. The message handler thread

#### 5.4.4 Property Collector (PC)

The Property Collector consists of one thread, the property handler thread, which is responsible for determining the properties of all the possible paths, making it possible for the Main Controller's thread to choose a path while other properties are being determined. The Device Handler is used to announce a measurement to the other device that is involved in the measurement and the IAL is used for the actual measuring of the property (Figure 47).

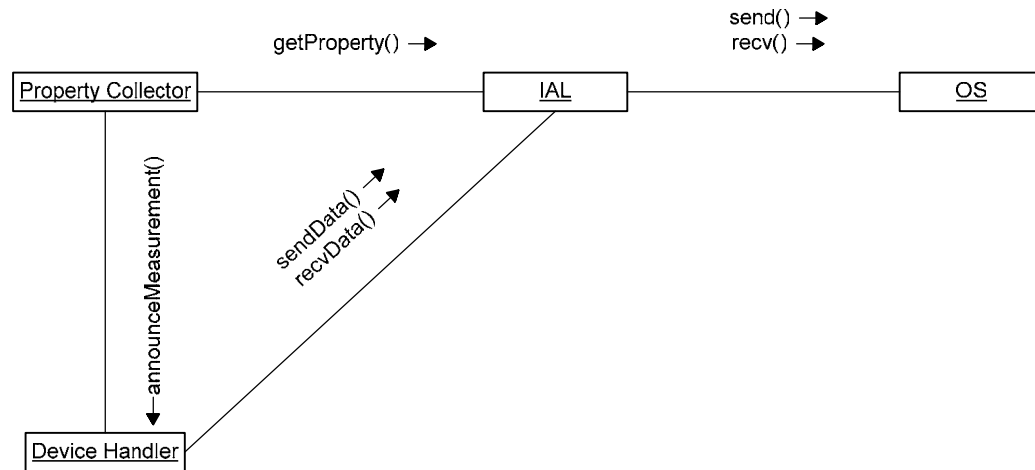


Figure 47. The property handler thread

Measuring properties and comparing paths are done by two separate threads. This allows a fast comparison of paths because paths can be compared with each other with only a couple of properties known. A downside of this performance enhancement is that it is possible for the Main Controller to select a non optimal path because not all properties are determined yet or the determined properties are outdated.

#### 5.4.5 Socket Abstraction Layer (SAL)

The SAL is used by the application's thread for creating virtual interfaces if necessary. The stream handler thread of the Main Controller uses the SAL to change the chosen interface pair for a stream (Figure 44).

#### 5.4.6 Interface Abstraction Layer (IAL)

The IAL is used by all other threads to communicate with other devices or to get information about the own device by accessing the operating system. The IAL also contains the algorithms to perform measurements to determine the path properties. The mediator pattern [27] is used to keep adding new measurement algorithms for new properties simple (Figure 48).

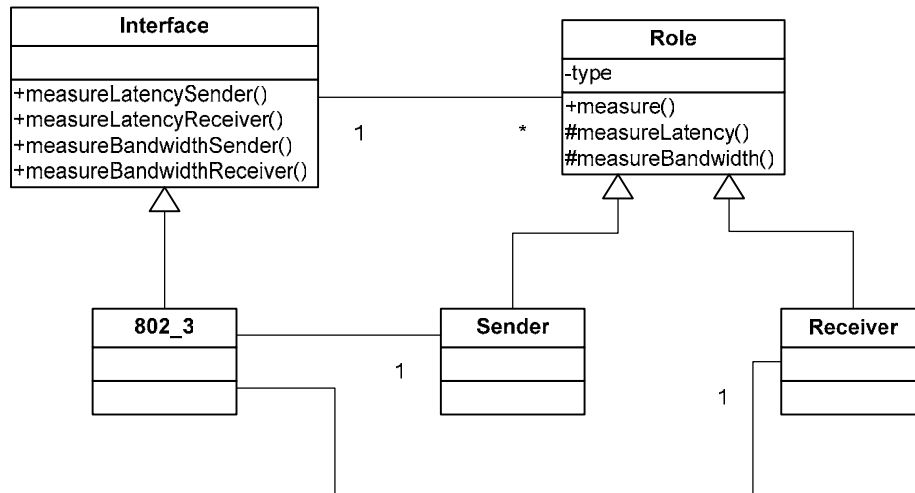


Figure 48. The mediator pattern used in the IAL

The class Interface offers a virtual API for each network type (e.g. 802.3). When the IAL must determine a property (for the Property Collector or the Device Handler) it creates a Sender or Receiver object with an attached Interface object. The IAL knows the type of the interfaces involved in measuring the requested path property, so it knows which Interface object it must attach. If the Property Collector wants to determine a property, the IAL creates a Sender object; else a Receiver object is created. The only method that needs to be called by the IAL is the `measure()` method. The creation of the Role object (Sender or Receiver together with the attachment of the Interface object) determines which protected method is called by the `measure()` method.

When a measurement announcement is received by the DH from another ISL device (4.6.2), the IAL creates a separate thread that waits for the actual measuring to begin. The IAL sends a message back to the sender of the measurement announcement after which the measurement is started by the sender. When the measurement is finished, the thread sends the result of the measurement back.

## 5.5 Using ISL; replacing standard BSI functions by others with the same name

In this section we want to look at how an application can use the ISL. An application must not need changes before it can use the ISL (requirement 12), so we want to avoid source code modifications. For this reason the ISL offers the same interface as the BSI, which causes the problem of name clashes because the standard C library already contains the functions the ISL offers. To avoid source code modifications and the problem of name clashes, we compile the ISL as a shared library and make sure an application makes use of this shared library instead of the BSI. The later is done by setting the environment variable `LD_PRELOAD` to our ISL library (`libISL.so`), meaning that it gets loaded before all other libraries.

To avoid setting this variable manually, we implemented a kind of ‘wrapper application’ do this for you:

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

```

```

int main(int argc, char *argv[])
{
    putenv("LD_PRELOAD=libISL.so");
    execvp(argv[1], &argv[1]);
    return 0;
}

```

The above wrapper program first sets the environment variable and then starts the application with all its parameters (which the wrapper program itself got as its own parameters). This method does not require recompilation of the application's source. ISL should also be extensible to give new applications access to more advanced functionality (requirement 16). This is done by modifying the application's source adding `#include "psl.h"` to the application's header files. When this is done, the new ISL functions can be added to the application's source. The ISL library must still be loaded using the method described earlier in this section.

To avoid name clashes when the ISL itself calls the actual BSI functions, the Dynamic Linking library (`libdl.so`) is used. Else, when the ISL calls a function of the BSI directly, a loop appears because not the corresponding function of the C library is called but the corresponding function of the PSL. To avoid this, we get the function pointer of each BSI function using the following code:

```

int (*_libc_socket)(int domain, int type, int protocol);
void* _libc_handle = NULL;

if ((_libc_handle = dlopen("libc.so.6", RTLD_LAZY)) != NULL)
{
    _libc_socket = (int (*)(int, int, int))dlsym(_libc_handle,
                                                "socket");
}

int libc_socket(int domain, int type, int protocol)
{
    int r = -1;
    r = (*_libc_socket)(domain, type, protocol);
    return r;
}

```

So instead of calling the function `socket`, we call the function `libc_socket` with all its parameters. The `libc_socket` function passes all the parameters to the function pointer of the 'real socket function' (for more info, see the manual page of 'dlsym').

For debugging purposes, we have prefixed all the BSI functions offered by the PSL with `isl_` and added the declarations of the prefixed functions to the `psl.h` header file. Therefore it is not necessary to override the standard BSI using the `LD_PRELOAD` environment variable. The shared ISL library needs to be available to the application (it is enough to put the library in a directory of the library path, e.g. `/usr/lib`). Furthermore the application's source need to have a `#include "psl.h"` instead of the `#include <sys/socket.h>` and all occurrences of BSI functions in the application's source need to be prefixed with `isl_`.

## 6 Evaluation

In this chapter we discuss how we have evaluated the working of the ISL. We test whether ISL selects the best interface pair (the path with the lowest cost, see 5.3) and if a new interface pair is selected when a better interface pair is available. Furthermore the performance and the legacy support of ISL are discussed (e.g. how will ISL operate in a network with non-ISL enabled devices?).

### 6.1 Interface pair selection (startup test)

To test if the best path is selected when ISL is started we have used two devices (Pentium 4's) each with two interfaces (a 100Mbit/s and a 10Mbit/s NIC). Both devices use Linux (Debian) as their Operating System. The complete set up is given in Figure 49.

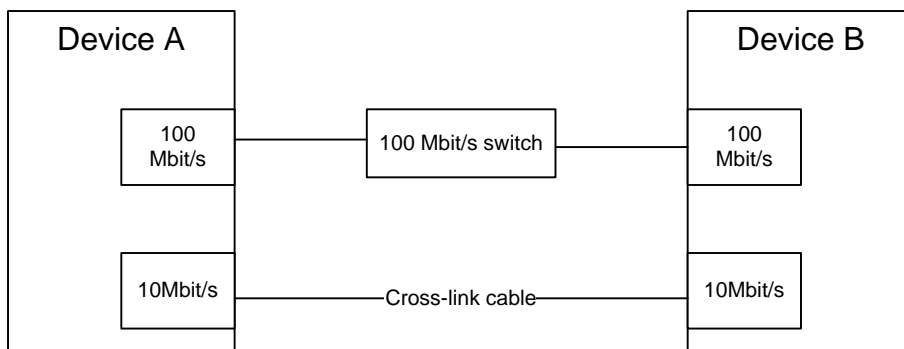


Figure 49. Test set up

A video stream is set up between Device A and Device B using GStreamer [28], using a modified sink and source plug-in for transmission using TCP [29]. We made sure that the video stream uses the 10Mbit/s path when GStreamer is started. All the BSI function calls in these plug-ins were prefixed with `isl_` because it was not possible to use the wrapper program: it resulted into multiple instances of the ISL per device, because GStreamer uses multiple threads. So recompilation of GStreamer's TCP source and sink source code was necessary.

#### 6.1.1 Functionality

We saw that the LEDs on the 100Mbit/s switch are not blinking which implies that the 10Mbit/s path is used instead of the 100Mbit/s path. After a while ISL determines that the 100Mbit/s path has a smaller transmission time and chooses the 100Mbit/s path. After the path switch GStreamer continued streaming the video without a disconnection. We also used Ethereal [30] to verify that indeed the packets were first sent over the 10Mbit/s path and later over the 100Mbit/s path.

#### 6.1.2 Performance

While the video is streaming from Device A to B, we see a lot of hiccups in the video displayed at Device B (each less than a second). When ISL performed the switch between paths, the video stopped until the path switch was completed. The hiccups are caused by the handling of the (ISL) threads by the operating system and because

our implementation is not made for performance (there are pieces of code that are not efficient: lots of memory allocations, de-allocations and – in the end – unnecessary copying of data). The stopping of the video during a path switch is caused because external third party tools are used to perform the switch (so a new process is created for each tool when called and multiple ARP-reply packets – to avoid loss - are sent with an interval of 1 second between packets). This delay during the path switch should be minimized by adding extra buffering and integrating the third party tools with the ISL.

When comparing paths it is not necessary to know the exact values of the properties of each path; only knowing their order is enough to see which path is the best path. Therefore simplifying the property determination algorithms could also give better performance to the ISL because properties are known earlier.

## **6.2 Switching interface pairs (changing test)**

To test if ISL changes its selection when a better path comes available we used the same set up as given in Figure 49. We disconnected the 100Mbit/s path so at first ISL can only select the 10Mbit/s path.

### **6.2.1 Functionality**

While the video is streaming we connected the 100Mbit/s path and saw that in a couple of seconds ISL changes the 10Mbit/s path used by GStreamer to the 100Mbit/s path, which we again verified with Ethereal.

### **6.2.2 Performance**

The same performance issues as in the previous test (6.1) are seen here.

## **6.3 Legacy devices**

When the sender is a legacy device it is not able to connect to an ISL device because of the chosen handover method. An ISL receiver creates a virtual address and assigns this address to a real interface. When an application on an ISL enabled device listens for incoming connections or messages it listens on the virtual address and not on a real address. Because legacy devices have no knowledge of the virtual addresses, they are not able to connect to ISL devices.

When the sender is an ISL device and the receiver a legacy devices, the ISL device is able to connect to the legacy device, because it has no information about a virtual address it connects direct to the real address of the legacy device. The only problem here is that ISL has no extra functionality to offer to improve the quality of the connection (see chapter 7).

A possible solution to let legacy devices connect to ISL devices is to change the handover method. Instead of directly assigning a virtual address to a real interface, it should be possible to change the internal routing on a device so that all the traffic sent to a certain port on a certain address is forwarded to a virtual address and the other way around. So the main idea is to put some kind of NAT between the virtual addresses and the real addresses.

## 7 Future work

During the project, we came up with new ideas that could lead to future improvements for ISL. Also some parts of the current ISL need to be optimized to make it suitable for use in real devices. A list of interesting future work is given below.

**Use standard protocols:** When an ISL device is connected to the network by using a combination of unicast and broadcast messages instead of waiting for all devices to send their periodic broadcast message: replace the current broadcasting method with a standard protocol such as the Simple Service Discovery Protocol (SSDP) [25], (also used in UPnP). This can also minimize the initialization time.

**Implement ISL in the kernel:** Let the ISL replace the Berkeley Socket Interface implementation so that it is part of the kernel and therefore can operate on a lower level. The main advantage that, in practise, it faster than a layer on top of the kernel. This also makes it possible to let the ISL be a singleton per device and not per process or thread.

**Develop and evaluate cost functions to compare paths and select the best path:** Look at comparison algorithms and look at how to compare paths when property values are known so that a cost function is used to determine the best path for each stream.

**Improve legacy support:** Add NAT functionality to ISL (see 6.3), which makes it possible to let ISL be compatible with legacy devices and offer the possibility of using multiple interfaces for one stream.

**Develop one-sided property determination algorithms:** Look for property determination algorithms that also work in a situation where the other device is non-cooperative (e.g. to determine path properties between an ISL and a legacy device).

**Develop measurement algorithms for other types of interfaces:** Evaluate how well the current property determination algorithms are suitable for measuring properties of paths which do not use 802.3 and develop property determination algorithms for these other technologies if the current algorithms are not suitable.

**Add an ISL specific API:** To let applications have more control on which paths the ISL may use or to help ISL by setting the minimum requirements of a path, a ISL specific ISL is needed which newly developed applications can use to make better use of the ISL.

## 8 Conclusion

Looking at the possible interface pairs between devices is a new topic and not yet covered in the literature. Property determination is an already known topic, where already lots of research is done (see section 2.3.2). We have used the already known knowledge of property determination and, if it was necessary, modified it for the home network situation.

We used the Big Mac probe to measure latency and compared different algorithms based on self-induced congestion to measure the available bandwidth of a path. Cross-traffic causes distortions in the measurements and more research is needed to investigate the effect of cross-traffic on the measurements. Also we have only focussed on wired networks and not on wireless networks. Wireless networks will introduce a whole new challenge because of influence from the outside world (microwave ovens and other distortions) so wireless networks is a separate topic. In the end, exact determination of the path properties is impossible but it is not necessary to determine the exact values of the properties, only an estimate is needed to compare paths. We are not interested in small differences between paths, but only in the large differences.

The presented architecture makes it possible for applications to use the ISL without any modifications. Furthermore both the application and the ISL can do their work separately so that the implementation uses multiple threads to improve the performance.

We looked at different handover methods and used a combination of two ideas. The combination works, although it is not possible for legacy devices to connect to an ISL device because the legacy device has no knowledge about the virtual addresses used by the handover method. Furthermore, the data stream is halted until the handover has finished.

This document is a good start in the new topic of interface selection by presenting an overview of different algorithms to measure path properties and a few handover methods which can be used in the presented architecture to offer a device the possibility to select different paths for communication with another device. Interface Selection reduces the congestion of the network by offering a more balanced utilization of the network's infrastructure.

## **9 Acknowledgements**

I would like to thank my supervisors Johan Lukkien, Richard Verhoeven and Michael van Hartskamp for their guidance and encouragement during the course of the project. Although the comments of Johan were strict and somewhat harsh, they were always reasonable and fair and they have proven to be very constructive during the course of my project. Also, I would like to thank the people of the SwA group at Philips Research Laboratories and especially my room mates at Philips (Chavdar, Ruud and Melissa) for their support and the friendly working atmosphere. Furthermore I want to thank Jan Ouwens for his help with setting up GStreamer and his TCP/IP plug-in for GStreamer.

Finally, I would like to thank my parents and my girlfriend Marie-Anne for their continuing support and understanding while I was working on my project.

## 10 References

- [1] Andrew S. Tanenbaum, “Computer Networks – Fourth Edition”, Prentice Hall, ISBN 0130661023
- [2] Radia Perlman, “Interconnections – Second Edition”, Addison Wesley, ISBN 0201634481
- [3] Differentiated Services (diffserv) working group,  
<http://www.ietf.org/html.charters/diffserv-charter.html>
- [4] Integrated Services (intserv) working group,  
<http://www.ietf.org/html.charters/intserv-charter.html>
- [5] Simple Network Management Protocol (SNMP) – Network Management RFCs sorted by topic (see topics about SNMP),  
<http://www.simpleweb.org/ietf/rfcs/rfcbytopic.html>
- [6] David C.M. Wood, Sean S. Coleman, Michael F. Schwartz, “Fremont: A System for Discovering Network Characteristics and Problems”, University of Colorado, 1993 Winter USENIX – January 25-29, 1993 – San Diego, CA
- [7] Michael F. Schwartz, David H. Goldstein, Richard K. Neves, David C.M. Wood, “An Architecture for Discovering and Visualizing Characteristics of Large Internets”, CU-CS-520-91, February 1991, Department of Computer Science, University of Colorado
- [8] Mark Coates, Rui Castro, Robert Nowak, “Maximum Likelihood Network Topology Identification from Edge-based Unicast Measurements”, In Proc. ACM SIGMETRICS 2002 11-20. ACM Press, New York.
- [9] James Curtis, Tony McGregor, “Review of Bandwidth Estimation Techniques”, Department of Computer Science, University of Waikato, Hamilton, New Zealand, <http://wand.cs.waikato.ac.nz/old/wand/publications/bwest-review/>
- [10] V. Jacobson, “Pathchar – A tool to infer characteristics of Internet paths”, Presented at the Mathematical Sciences Research Institute (MSRI);  
<ftp://ftp.ee.lbl.gov/pathchar/>, April 1997.
- [11] Allen B. Downey, “Using Pathchar to estimate Internet link characteristics”, ACM SIGCOMM Computer Communication Review, Volume 29, Issue 4 (October 1999), Pages 241-250, ISSN: 0146-4833.
- [12] Kevin Lai, Mary Baker, “Measuring Bandwidth”, Department of Computer Science, Stanford University,  
<http://mosquitonet.stanford.edu/~laik/projects/nettimer/publications/infocom1999/html/nettimer.html>

- [13] Constantinos Dovrolis, Parameswaran Ramanathan, David Moore, “What do packet dispersion techniques measure?” University of Wisconsin and CAIDA, [www.pathrate.org](http://www.pathrate.org)
- [14] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, Les Cottrell, “pathChirp: Efficient Available Bandwidth Estimation for Network Paths”, Department of Electrical and Computer Engineering – Rice University, SLAC/SCS-Network Monitoring – Stanford University.
- [15] Manish Jain, Constantinos Dovrolis, “Pathload: a measurement tool for end-to-end available bandwidth”, Computer and Information Sciences, University of Delaware, [www.pathrate.org](http://www.pathrate.org)
- [16] M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control”, RFC 2581.
- [17] Ravi Prasad, Manish Jain, Constantinos Dovrolis, “Effects of Interrupt Coalescence on Network Measurements”, College of Computing, Georgia Tech., USA, Passive and Active Measurements (PAM) conference, April 2004.
- [18] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, Kevin Gibbs, “Iperf: The TCP/UDP bandwidth measurement tool”, <http://dast.nlanr.net/projects/Iperf/>
- [19] Kevin Lai, Mary Baker, “Measuring Link Bandwidths Using a Deterministic Model of Packet Delay”, Department of Computer Science, Stanford University.
- [20] Brian “Beej” Hall, “Beej’s Guide to Network Programming – Using Internet Sockets”, <http://www.ecst.csuchico.edu/~beej/guide/net/>
- [21] Andrew S. Tanenbaum, “Modern Operating Systems”, Prentice-Hall International Editions, ISBN 0-13-595752-4.
- [22] Sun Microsystems Inc., “RPC – Remote Procedure Call - Protocol Specification”, RFC 1050.
- [23] Thomas Habets, “Arping v2.05”, <http://www.habets.pp.se/synscan/programs.php?prog=arping>
- [24] IP Routing for Wireless/Mobiles Hosts (mobileip) Charter, <http://www.ietf.org/html.charters/mobileip-charter.html>
- [25] Simple Service Discovery Protocol 1.0 – Operating without an Arbiter, Internet Engineering Task Force, Internet-Draft: [http://www.upnp.org/download/draft\\_cai\\_ssdp\\_v1\\_03.txt](http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt)
- [26] Service Location Protocol (svrloc) Charter, <http://www.ietf.org/html.charters/svrloc-charter.html>
- [27] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns”, Addison-Wesley Professional.

- [28] “GStreamer – Open Source Multimedia Framework”,  
<http://gstreamer.freedesktop.org/>
- [29] Jan Ouwens, “Media Streaming over both Wired and Wireless In-Home Network”,  
<http://www.win.tue.nl/~iradovan/research/afstudeeropdrachten/streaming.htm>
- [30] Ethereal: A Network Protocol Analyzer, <http://www.ethereal.com>