CHALMERS

Optimized Routing within an Ericsson Node

Routing procedures in an Ericsson SGSN

Master of Science Thesis

ELISABET SVENSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Optimized Routing within an Ericsson Node
Routing procedures in an Ericsson SGSN

ELISABET M SVENSSON

Examiner: K.V.S. PRASAD

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

Abstract

In a specific Ericsson telecom equipment (SGSN) there are application boards and router boards. When the routers can not communicate directly with each other, a handover process between them can cause packet loss and unnecessary use of the internal network. This thesis presents a solution to this problem, using routing tables and feedback from routers to applications. The proposed solution is implemented into the node's source code. Performed tests show that the packet loss decrease dramatically and with that, the backplane is used more effectively.

Acknowledgements

The author could not have done this thesis with such good results without the support from my supervisors KVS Prasad at Chalmers and Daniel Nilsson at Ericsson. They have both, in different ways, helped pushed the project forward.
The author would also like to thank family and friends for support and encouragement during this time. Especially big thanks to Iris Duranovic for her love and support as well as proofreading and tips throughout the project.
The author would like to thank Therese Nordqvist and Jonas Hadin for proofreading.

Elisabet Svensson, Göteborg 11/06/09

Contents

Nomenclature

Computer Communication

ABR	**Area Border Router** In the OSPF protocol, the AS is divided into areas. The ABR is the router that connects one area with the others.
AS	**Autonomous System** A small network with its own routing protocol
IGP	**Interior Gateway Protocols** Routing protocols to be used internally in an AS
IS-IS	**Intermediate System To Intermediate System** A routing protocol similar to OSPF
OSPF	**Open Shortest Path First** A routing protocol commonly used in ASs. Based on link state information.
PDU	**Protocol Data Unit** A message format used in IS-IS
RIP	**Routing Information Protocol** One of the first routing protocols used on a more general basis. Based on distance vector information.

SGSN Specifics

AP	**Application Processor** The board in the SGSN that takes care of signaling, such as where the mobile device is and which cell tower it is connected to, what subscriptions it has etc.
DP	**Device Processor** The board in the SGSN that takes care of user payload. It also charges the user for packets sent as well as security issues.

GGSN	**Gateway GPRS Support Node** A node in the GPRS system
GTT	**GSN Testing Tool** A test tool that simulates the environment surrounding SGSN, optionally also the node itself
IBAS	**Interface Board ATM Single-Mode Fiber** A router PIU for both ATM and ETH
PEBv5	**Power and Ethernet Board version 5** The latest version of PEB, that is able to be used as a router
PEB	**Power and Ethernet Board** A PIU that provides power to all boards as well as a internal network in the node
PIU	**Plug-In Unit** A board in the node that is plugged in
SGSN	**Serving GPRS Support Node** The node in the GPRS system this project will focus on

Mobile Communication

BSC	**Base Station Controller** A controller for BTS
BTS	**Base Transceiver Station** Cell tower used for GSM
GPRS	**General Packet Radio Service** The service that makes it possible for a cell phone to connect to internet
GSM	**Global System for Mobile Communications** The second generation(2G) of digital and wireless connections
kpps	**Kilo Packets Per Second** Measurement for packet speed
LTE	**Long Term Evolution** Fourth generation (4G) of digital and wireless connections

NodeB	**Cell tower** Corresponds to the BTS, but can be used for WCDMA as well.
RNC	**Radio Network Controller** A controller for NodeBs
WCDMA	**Wideband Code Division Multiple Access** Third generation (3G) of digital and wireless connections

1. Introduction

Every day millions of users connect to the Internet through their mobile phones. Whether the user wants to check the news, facebook or email, the phone will use something called *General Packet Radio Service* (GPRS). GPRS is a network which will connect the phone to Internet, illustrated in figure 1.1.

The GPRS consists of various elements which together make it possible for phones to use the internet. The GPRS network is not only involved in Internet connections, it is involved in everything a cell phone does except voice calls.

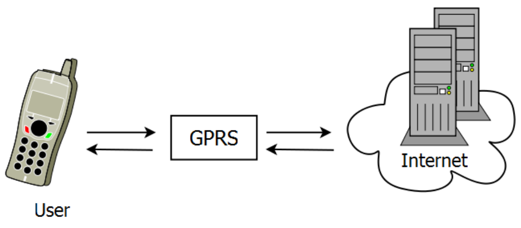


Figure 1.1: *Connection between a mobile phone and Internet is through GPRS.*

When the phone, for instance, wants to browse the web it connects to a cell tower, managed by a controller. The traffic from the mobile phone passes through the cell tower and its controller and then through the two GPRS support nodes, *Serving GPRS Support Node* (SGSN) and *Gateway GPRS Support Node* (GGSN), see figure 1.2. From the GGSN the traffic is sent out on the internet. [7] [8]

Figure 1.3: *An Ericsson SGSN [10]*

This project is about the SGSN, figure 1.3. The node consists of a number of *Plug-In Units* (PIUs). Among these, there are *Application Processors* (APs) and *Device Processors* (DPs) sending and receiving traffic via the node's router. The AP handles the signaling, keeping track of which mobile devices are attached, where they are and which cell towers they are con-

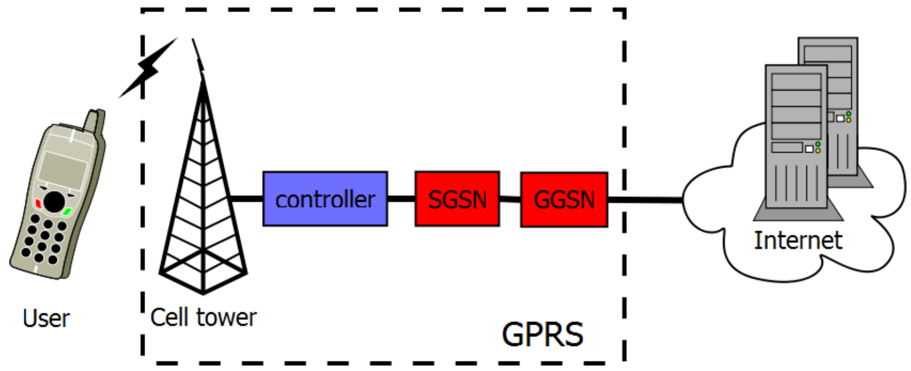


Figure 1.2: *The different components in GPRS.*

nected to. The DP boards take care of the traffic the user generates when he or she surfs the net with the cell phone. The user payload is redirected towards the GGSN and can be monitored in order to charge the user correctly, or when the phone company wants to control the internet accessibility. In this report, the word application will be used to denote both AP and DP.

The SGSN also contains a number of *Power and Ethernet Boards* (PEBs) with two main functions. As the name suggests, a PEB is responsible for providing power and ethernet to the other PIUs. The ethernet connections provided by the PEBs are between the PIUs inside the node, the so called backplane. The backplane makes it possible to send packets within the node.

The routers used in the SGSN is called *Interface Board ATM Single-Mode Fiber* (IBAS) routers. [9] In order to provide resilience and redundancy, there are multiple routers. A node will then have multiple ways to contact the external networks.

Capacity in the SGSN can be increased with more application PIUs. The latest version of PEB (version 5) can function as a router, and the IBAS routers can be replaced by extra application boards. However, by doing so, there is now a number of internal routing problems to consider. This project is about these problems, and how to optimize the routing between application PIUs and PEBs within the node.

The project aims to improve the routing process in terms of speed and packet loss, memory space and internal network traffic. The project is located at Ericsson Lindholmen.

2. Purpose

The purpose of this thesis is to present a solution to the internal routing problems in a SGSN with multiple routing PIUs that are not directly connected to each other.
The project optimizes the internal traffic in the node, with respect to time, memory space and traffic load over the backplane.

The application boards in the SGSN have limited memory space, so the routing procedure should limit the memory use.
Traffic load on the internal network in the SGSN should be limited to avoid a possible flooding situation.

The main focus is the packet loss problem. The memory is today not an immediate problem and the SGSN in itself is designed to avoid any possible flooding situations.

3. Limitations

Optimally all solutions should be tested on a real SGSN in a real GPRS environment. Real nodes require setup and configuration, while a simulated SGSN is preconfigured. Specific hardware problems can be avoided when working in a simulated environment. Due to time limits of this project and a lack of available hardware, no parts of this project will be tested on a real SGSN.
Owing to limitations in the simulated node, measurements might not correspond to a real SGSN. However, this project will mainly focus on the comparison between different solutions.
There is currently not a simulation of PEBv5 in the *GSN Testing Tool* (GTT), but the functionality with route distribution is implemented for IPv6 on the simulated IBAS routers.
The simulated SGSN is not capable of handling heavy traffic load, why no tests can be performed during such conditions.

There are in GTT support for at most two routers, why no testing will be done with more routers than that.

4. Background

As mentioned in the introduction (section 1) the SGSN is one part of GPRS, a system used to make mobile devices able to connect to the Internet. The GPRS system is involved in everything a mobile phone does, except voice calls.
The GPRS consists of various elements with different tasks. We have the cell tower and its controller that makes the radio contact with the mobile devices. There is also the GGSN, responsible for keeping billing information and allocating IP-addresses. The GGSN also has the interface towards the network known as the Internet.
This project, however, is focused on the SGSN. The SGSN takes care of authentication of users, where they are and what traffic they send and receive (called user payload).
Most commonly used in the GPRS network is the routing protocol *Open Shortest Path First* (OSPF). In this report it is presumed that the GPRS always uses OSPF internally.

The SGSN is connected to a variety of different networks, and each router has one routing table for each network. The networks have different tasks, for instance there is one network for the traffic towards GGSN, one for GSM cell tower controllers, one for WCDMA controllers etc.

4.1 The problem at hand

Inside the Ericsson SGSN there are a number of boards. Some of these are applications, either AP or DP, sending and receiving traffic from the outside world via the router board.
In order to provide resiliency, there is more than one router in the SGSN. If one router experiences problems, it may be a loose cable or it needs to be rebooted, the whole node does not have to go out of service since traffic is re-routed through one of the others.
There are two different types of routers, IBAS routers and PEB routers, the architecture in the node is described below for each of the two types.

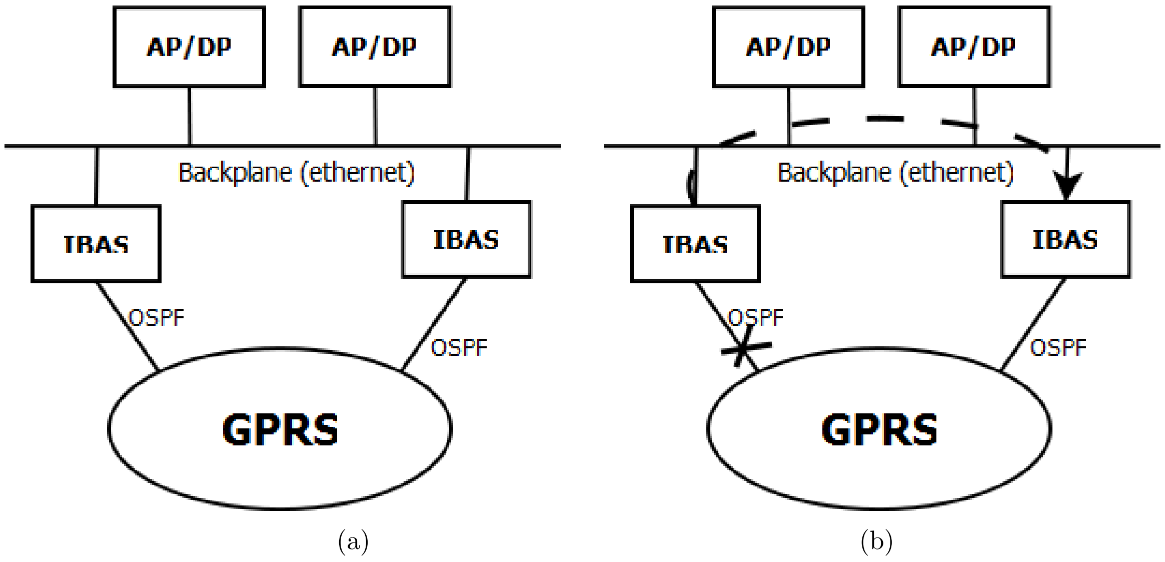


Figure 4.1: *Network sketch for the node's architecture with IBAS routers. The routers are connected through the backplane. If one router is unable to forward a packet, it will be redirected to another router that can.*

4.1.1 Architecture with IBAS routers

When the SGSN has multiple IBAS routers, they are directly connected to each other via the backplane (figure 4.1(a)). The applications send their packet to any of the router PIUs, and they do not know anything about the state of them. If one of the routers is unable to forward packets it can still receive them.

When one router PIU does not succeed in forwarding a packet, it will send the packet over the backplane to another router in hopes that that router is able to forward the packet (figure 4.1(b)).

4.1.2 Architecture with PEB routers

The latest version of PEBs (version 5) can function as routers, and the IBAS routers can then be replaced with application boards in order to increase the total capacity of the SGSN even more.

Each application is connected directly to each of the PIUs, as illustrated in figure 4.2. The PEB routers are not connected to each other through the backplane, thus there is no longer any possibility for one router to forward packets to another.

4.2 Route Distribution

The routing with PEBv5 is today implemented with routing tables and a state variable, the implementation is called route distribution. The routers have, as they used to on IBAS routers, one routing table for each network. A few tricks have been implemented in order to come to terms with the missing redundancy.

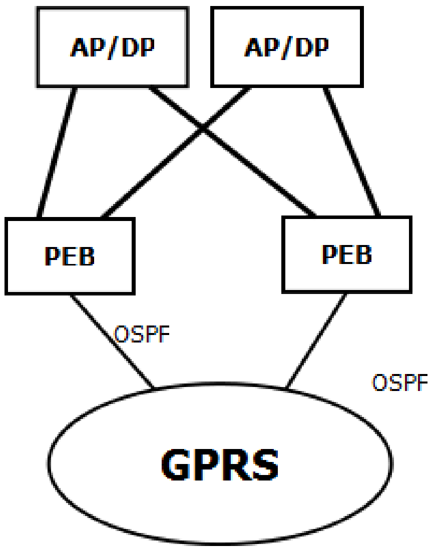


Figure 4.2: *Architecture with PEB routers; routing PIUs not connected through backplane.*

4.2.1 Application boards with routing tables

The application boards have their own forwarding tables, one table for each network on each router. When an application wants to send a packet, it first checks the forwarding table to see if the router is able to forward the packet and then sends the packet. This way the application will always send packets to a router that can forward them. However, there is inevitably a small delay from when the router has updated its table until the application has the updated version (see section below). If a router receives a packet that it can not forward (due to faulty entries in the forwarding table on the application) the packet will be dropped.

4.2.2 Updating application boards' forwarding table

Each router has a state register variable that is set when the routing table is updated. The applications that have a copy of the routing table is said to be subscribing to that network on that router. When the state register variable is set, a notification is sent to all applications. The subscribing applications will send a request to the PEB router, asking for the new table, and the PEB replies with the complete, updated routing table.

When one of the PEB routers adds or deletes an entry in its routing table the following will happen:

1. OSPF updates the router's routing table.
2. A timeout is (re-)started to avoid frequent updates.
3. The state variable changes.
4. All applications get noticed.
5. Subscribing applications request the new forwarding table.
6. The router replies with the updated table.
7. The applications replace the old routing table with the new one.

These steps are repeated every time the routing table in any of the PEB routers is updated.

4.2.3 Problems with the current implementation of route distribution

There are three major problems with the current implementation of route distribution.

4.2.3.1 Packet loss due to the updating procedure

The update procedure takes a long time, mostly due to the implemented delay that is restarted every time the routing table is updated, and the applications' forwarding tables will be old and might contain errors during this time. The applications will continue to send packets to the router according to the old table which might result in packet loss.

4.2.3.2 Network traffic over the backplane

Even though a single entry is updated in the forwarding table, the complete table is sent to the applications. Those big packets plus the high level of packet loss cause unnecessary load on the backplane.

4.2.3.3 Unnecessary use of memory space on application boards

Most of the routes are the same for the routers since they are connected to the same external network, and thus most of the entries in the different routing tables are the same. The applications have one table for each router, even though we expect most of the entries to be identical. The use of memory on applications can be optimized.

4.3 Project: propose new solution

This project is about proposing a better solution for this problem. The three aspects mentioned above (section 4.2.3); packet loss, internal network traffic and memory use is used to measure the success of the project.

The packet loss and backplane traffic part of the solution is implemented in the source code and tested.

5. Scope

- General analysis
  - The project look at these aspects;
    - Adaptation speed/Packet loss
    - Memory space on application boards
    - Traffic load on the internal network
  - A new routing solution is proposed and analyzed.
- Implementation
  - Suggested solution is implemented in the programming language C.
  - All implementation is on top of the original source code.
- Tests
  - The GSN Test Tool (GTT) is used for testing. GTT is a framework for the SGSN and simulates the nodes surrounding it. It can be run on a real or simulated node, but for this project the simulated node is used (see Limitations, section 3, for details).
  - Automatic tests are developed in the programming language Erlang.
- Test anaysis

6. Method

The project was carried out by investigating a proposed improvement to the routing issues discovered. First, this project developed a solution to propose. The aspects packet loss and backplane traffic of the solution was implemented and tested. The memory aspect was not tested, but is analyzed throughly.

6.1 Proposed Solution

Common routing protocols uses different methods to propagate changes in the network. The *Routing Information Protocol* (RIP) adapt to changes by letting each router send out routing state information every 30 seconds to its neighbors. Routing protocols such as OSPF and IS-IS send out routing state information each time the topology in the network has changed. The solution presented in this thesis is a combination of both of these techniques. It works with immediate feedback as well as periodic updates.
The period is set to five minutes (compared to 30 seconds for RIP and 30 minutes for OSPF/IS-IS). [1][2][3]

This is how it works:

- **Route distribution - complete table**
  The complete routing table is distributed periodically and when the node, an application or a router is restarted.

- **Feedback**
  When an application uses a route to send a packet and the router is unable to forward it, the router sends an error message to the application. The application will then consider that route to be invalid until further notice. Only packets sent during the round trip time will get lost. After that the application will send the packets to another router.

- **Forwarding Tables**
  Each application has one forwarding table per network.

This solution will discover negative changes, such as an unplugged cable or other connection errors, really fast. The feedback function makes the applications aware of changes almost instantly.
Positive changes, such as a new network configured or a cable replugged, is not discovered as fast. It takes up to five minutes before the router will send out a notification of its forwarding table's update. This may cause the work load to be unevenly distributed over the routers during this time.

6.2 Implementation

The implementation is inserted in the node's source code. The routing code on the node is complex, so here follows a rather brief description.

6.2.1 Operating system

The different PIUs in the node are of different types and architecture. All APs are Linux based while all DPs have VxWorks. IBAS routers also have VxWorks.
All testing will be done on a simulated node, all simulated nodes have IBAS routers (see section 6.3). Route distribution is implemented on IBAS routers, but only for IPv6. IPv6 support is only implemented on VxWorks, not Linux. Therefore all implementation is done for VxWorks.

```
#ifdef LINUX
    ...code only for linux...
#elif VXWORKS
    ...code only for vxworks...
#endif
```

Figure 6.1: *Code that is only used in Linux or VxWorks, respectively.*

When the code is compiled, certain parts are included or excluded depending on the operating system. With the C functionality of `#define` we can control which code is compiled, see figure 6.1.

In a similar manner we can control if certain code is supposed to only be on routers or only on applications, or only on PEB routers, or only on applications with Linux etc.

6.2.2 Levels of code

The source code in the node is roughly divided into two levels - kernel level and user level. The difference is more significant on boards using Linux than on those using VxWorks.
The code that monitors the routing is on the user level, but the actual routing is done on kernel level. In the original solution, all routing information takes

place on user level through the system called *Connectivity Deamon* (ConnD). This system will receive and send the messages about routing, such as notifications, requests or replies. All packets from GPRS are routed on kernel level by the *Forwarding Engine* (FE).

Let us use an simple metaphor. On the user level we have everything that makes routing possible but does not do any actual routing; packets from GPRS are not handled on user level. Let us call ConnD a postal service. The postal service provides postboxes, employment for postmen, post offices etc.

Postmen are operating on kernel level, regulated by the guidelines set up by the postal service. The postmen are the core of the postal service. Without them, no mail would ever get delivered. On the other hand, without the postal service, the postmen would not know where to deliver the letters.

This is how ConnD and FE interact. FE is the postmen, delivering packets according to the forwarding tables ConnD have set up.

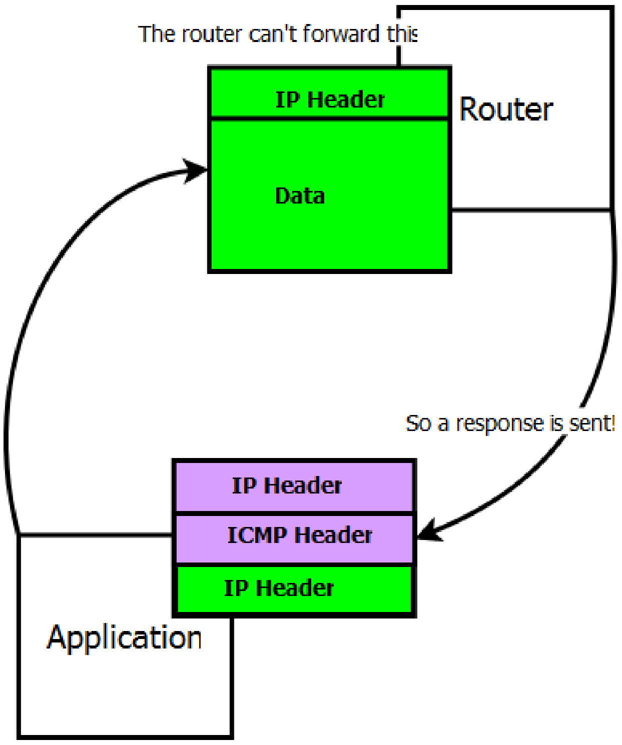


Figure 6.2: *The icmp message is constructed from the original message.*

6.2.3 Proposed solution - implementation

All implemented code is found in appendix B.
The solution proposed in this thesis is only partly implemented in the node source code. Merged forwarding tables are not implemented and the project is mainly focused on changes in FE. When the router is forwarding a packet it does so in FE.
The only changes in ConnD are to disable and change the functionality from the original solution.
The original solution had the applications request the forwarding tables every 30 minutes. In the proposed solution we want the router to send out its

forwarding tables every 5 minutes. This way, all additions to the forwarding table on the routers will take at most 5 minutes to propagate.
If the node or just the router is restarted, the router will function as the original solution for five minutes. The proposed solution is still there and working, giving feedback if a packet is not forwardable.

6.2.3.1 Router's code

When the router discovers a packet from the backplane that it is unable to forward, it constructs an ICMP Destination Unreachable message. The ICMP message has a specific type and code that is not used anywhere else in the node (Destination Unreachable - Address Unreachable). The ICMP message contains information about the router's location in the node as well as the header of the undeliverable packet to make the sender recognize which route it is that is invalid. Figure 6.2 shows a visualization of how the packet is constructed. Note that this message is *only* sent when the packet is received from the backplane.

6.2.3.2 Application's code

When the application receives an Desination Unreachable ICMP message with the code Address Unreachable, it will update its forwarding table. The information in the ICMP message gives the application information about which router is the sender of the ICMP. In the header from the lost packet (included in the ICMP message from the router) the destination address is found. With the application's normal methods to look up addresses in the forwarding table it can find the specific entry in the forwarding table and remove it.

6.3 Testing

The *GSN Test Tool* (GTT) is used to test the implemented solution. GTT simulates the environment surrounding an SGSN, the GPRS network. The tool can also simulate the node itself.
With automatic tests, written in Erlang, and a trace of all packets we can achieve testing that is repeatable. The test implementation can be found in appendix C.

6.3.1 Test environment

There is support for two routers in the simulated environment, but the second router has no default configuration. Everything else in the simulated node is automatically set up.
By default, all traffic is directed through the first router. By configuring a static route for IPv6 traffic we make the applications use that route as well.

6.3.2 Test implementation

The testing will give us an answer to the following questions:

1. How fast is traffic redirected?
2. How many packets are lost?
3. How much traffic is sent over the backplane?

The first two questions are essentially the same, the longer time the hand over process requires the more packets will be lost.

To simulate a OSPF update from the GPRS system we create (and later delete) a static route on one of the routers. The application will then prefer this router over any other. The test will generate traffic from the application to the external network. During the traffic flow the static route is taken down. The application will update its forwarding table and send traffic through another route. After the test, the number of packets sent from the application is compared to the number of packets received by GTT (simulating the outside world) to determine packet loss.

Generation of traffic is done by spawning 500 processes which create one mobile device each. The device is then attached to the SGSN and starts a hand shaking procedure with it. Since there are limitations in the test tool, GTT, it was not possible to generate as much traffic as wished for without making the node crash.

The static route is taken down during the period when the traffic is increased, a few seconds after the test has started.

The traffic rate is calculated by counting the number of packets during 2 seconds, ca 0.5 seconds before the update and 1.5 seconds after, and dividing with two. This way we will get an estimated traffic rate that will help us predict how the algorithm will behave during more extreme conditions.

6.4 Analysis

6.4.1 Minimize Packet Loss

The original and the proposed solution uses different techniques to avoid packet loss. The original solution uses routing tables, while the proposed solution uses direct feedback from the router.

6.4.1.1 Routing Tables

When the router updates its forwarding table, there are numerous steps until the application finds out about it. First of all, there is a delay implemented in order to avoid too frequent updates. After the delay the state variable is set, the notification and requests are sent and finally the forwarding table is sent and updated.
During this time, the applications continue to send packets according to the old, outdated forwarding table. All packets sent to a router that can not forward it, will be lost.

6.4.1.2 Router Feedback

The routers can be designed to give feedback to the applications directly. Whenever the router receives a packet from an application it will try to forward it. If the router does not succeed it will send a message back to the application. This way, the applications can know almost instantly if a router no longer is connected to the desired network. It only takes one round-trip time until the application has found out about the lost network.

6.4.2 Avoid heavy traffic load on the backplane

We want to minimize the traffic sent over the backplane. The traffic that is directed into or out from the node is considered minimum already, we do not want to touch it. However, there is some traffic that is ”unnecessary”. That is the traffic that is supposed to go out of the node to the GPRS network but never reaches outside of the node. Other traffic that might be possible to minimize is the packets sent within the node, containing only routing information.

6.4.2.1 Minimize unnecessary traffic

All packets that are lost are an unnecessary load on the backplane. As we will see in the result section, this is a considerably big part of the traffic we want to minimize (lost packets plus packets with only routing information).

6.4.2.2 Routing traffic

The proposed solution tries to limit the routing information traffic within the node by working with direct feedback from the router to the interested applications. While the original solution sends the complete routing table each time any entry in it has changed, the proposed solution will only generate extra traffic when the entry is actually used by an application.

6.4.2.3 Limit number of packets vs limit size of packets

There are two different viewpoints of this problem; do we rather cut the number of packets sent or the number of bytes? In other words; do we prefer fewer and larger packets or multiple smaller packets? At this point it is natural to expect that the original solution will result in few, large packets and the proposed in more but smaller packets.

6.4.2.4 Backplane traffic in the Original Solution

If the network outside the node is stable, with few OSPF updates, there is not much excessive traffic over the internal network. All packets are sent to a router that is able to forward them. The forwarding tables are correct on all applications.
An unstable network will result in heavier traffic load over the internal network. Undeliverable packets might have to be resent, depending on their protocol (such as TCP or SCTP that uses reliable transmissions).
When the forwarding table is updated on a router, the complete table is sent out to subscribing applications. If the forwarding table is big, this might result in rather large packets.

Each OSPF update will cause extra traffic over the backplane. First of all, there is a notification sent to all applications. Then there is a request sent from subscribing applications, requesting the updated table. The last part is the packet containing the routes, sent from the router to all subscribing applications.

The router sends out notifications to all applications. Subscribing applications will reply with a request for the updated table.When the router receives such requests, it will construct a packet containing the complete forwarding table.

6.4.2.5 Backplane traffic in the Proposed solution

The proposed solution will make the router send an error message for every packet that it can not forward. This may increase the pure routing traffic, but on the other hand the number of undeliverable packets will decrease.

Pure routing messages are sent out, as mentioned, every time a packet is not forwardable. In addition to that, the router will send out the routing table to applications periodically and after a router or application restart. At those times the procedure is the same as for the original solution, with notifications, requests and replies.

6.4.3 Minimize memory requirements

The memory space on the application boards is limited, why we might want to have a routing solution which limits the memory usage.

6.4.3.1 Merged Routing Tables

With the currently implemented solution minimization is possible. Right now each application has one routing table per network and router. These tables can be merged to decrease the memory space needed. Each application would then have one forwarding table per network, in which all routers are present. Since most routes in the different forwarding tables for different routers are the same, the merged table is expected to require less space. The forwarding table needs to contain record of which entries is valid for which routers.

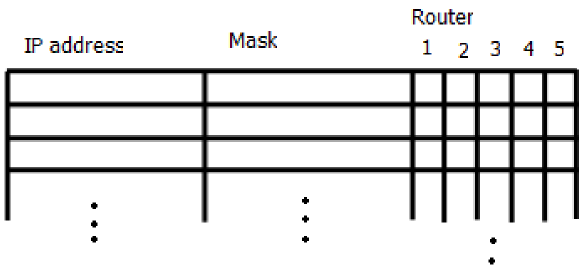


Figure 6.3: *How a forwarding table could look like, with the IP address and mask as well as a list of which routers that has that route.*

For each entry in that merged table there is a binary list with one bit per router, 1 if the entry is valid for the router and 0 if it is not. The binary entry can then be compared to a list with all routers to determine which router has that route, illustrated in figure 6.3. This way there will be no duplicated entries of IP address and masks.

7. Results

7.1 Packet loss analysis

Each test was performed by creating and deleting a static IPv6 route while triggering as much traffic the node could handle without crashing. The application will start using the static route as soon as it finds out about it, and will prefer it over the other router. When the route is taken down, applications will use another route (again).

When the application uses the static route 500 simulated mobile devices is created and attached to the (simulated) SGSN. The test will during these attachments take down the static route, making it unusable. Until the application has updated its forwarding table, it will send traffic towards the router that can not forward it, and none of the packets are received by the GTT board.

The test case is monitoring all traffic sent from the application and the traffic received by the GTT board from the node. The exact time of the forwarding table update (when the static route is taken out of service) is determined by looking at the last packet received by the GTT board from the static route.

The number of *packets per second* (pps) is an average, calculated by counting the number of packet during two seconds, approximately 0.5 seconds before the update and 1.5 seconds after, and dividing with

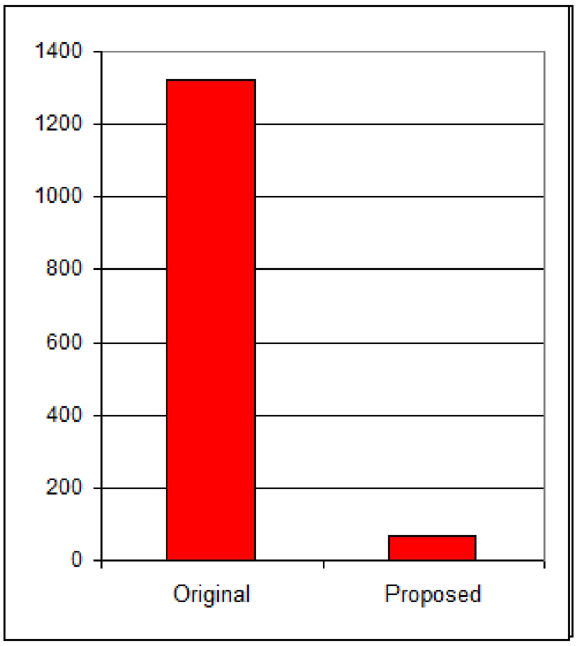


Figure 7.1: *The total sum of packets lost during the ten test cases for original and proposed solution.*

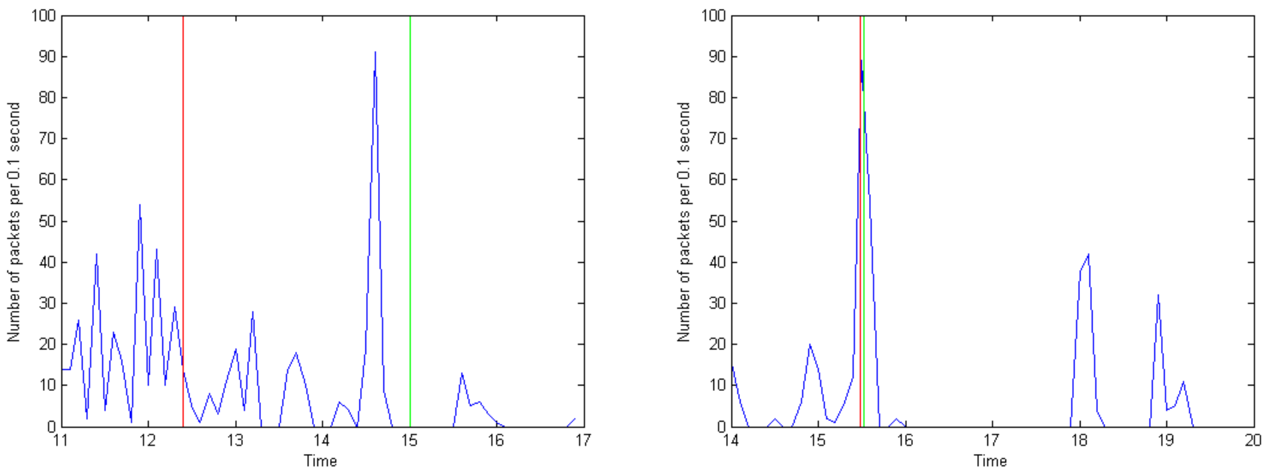


Figure 7.2: *The blue line is the number of packets sent, counted every 0.1 seconds. The vertical red line is the last packet successfully delivered. The green line is the first packet received after the update. All packets in between are lost. The left graph is the results from a test of the original solution and on the right is a graph for the results from the proposed solution.*

two.

The sum of all packets lost during all ten tests is illustrated in figure 7.1 for both the original and proposed solution.

7.1.1 Tests of original solution

The traffic rate during each of the tests are found in appendix A. Figure 7.2 shows an example of a typical test result. The graph's y-axis is number of packets sent per 0.1 seconds (by the application) and the x-axis is time. The last packet received by GTT is marked with a vertical red line. The first packet received after that is marked with a green line, indicating that the application has updated its forwarding table.

All data is compiled in table 7.1. The update time ranges from 1.78 to 3.9 s, but most of the tests show that the update procedure takes between two and three seconds. The packet loss is also varying, from 50 packets up to 185 packets.

Test	Update time	Packets lost	Average pps
Test 1	3.9 s	78	45.5
Test 2	2.16 s	144	109.5
Test 3	2.22 s	174	116.5
Test 4	1.78 s	185	123.5
Test 5	2.26 s	50	35.5
Test 6	2.12 s	182	96.5
Test 7	2.3 s	85	52
Test 8	2.26 s	136	114
Test 9	2.08 s	143	66
Test 10	2.12 s	144	81

Table 7.1: ***All tests:*** *Table for all results from the testing of the original solution*

7.1.2 Tests of proposed solution

As for the original solution, we have ten graphs for the ten tests. One of the graphs are seen in figure 7.2 (all the others are in appendix A). As before, the blue line indicates the number of packets sent every 0.1 second. The vertical red line indicates the last packet received from the static route and the green vertical line indicates the first received by another route. Please note that the scale is not the same on all graphs.

All data is compiled in table 7.2. The update time ranges from 0.04 to 0.98 s. The update time is calculated from the last packet GTT received from the static route until the first packet received from the other router. The traffic intensity is relatively low, and the packets are sent out intermittently. The longer update times measured were due to the application not sending any traffic, not the updating process to be slow.

The proposed solution is dependent on the packets sent, while the original solution will make the application update its forwarding table even though it did not send any packets.

The packet loss varies, but the *worst* test result from the tests of the proposed solution, with 16 packets, is far better than the original solution's *best* result, 50 packets.

Test	Update time	Packets lost	Average pps
Test 1	0.06 s	1	108
Test 2	0.98 s	6	74
Test 3	0.04 s	16	185.5
Test 4	0.32 s	4	75
Test 5	0.04 s	16	53.5
Test 6	0.38 s	9	3200.5
Test 7	0.04 s	7	88
Test 8	0.92 s	7	63.5
Test 9	0.24 s	2	60
Test 10	0.26 s	2	83.5

Table 7.2: ***All tests:*** *Table for all results from the testing of the proposed solution*

7.1.3 Expected results

7.1.3.1 Original Solution

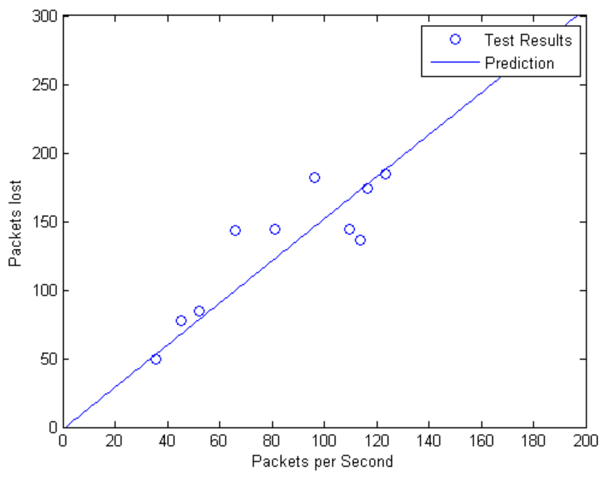


Figure 7.3: *Expected and actual packet loss for original solution for traffic speed 0-120 pps*

Let us assume that we have n applications. We expect each of these to send v packets per second to the route on the forwarding table affected by an update.

In order to avoid too frequent updates there is a delay implemented in the router's code. The delay is here denoted with D. After the timeout, the state variable is set with a message over the backplane. Notifications to all application boards are sent out and the subscribing applications will request the updated forwarding table. The router will respond with the new, updated table. The messages sent over the backplane is thus:

1. Message to set the state variable
2. Notifications to all applications
3. Requests from subscribing boards

4. Reply from router with new table

If each of these takes t seconds to travel between its source and destination, the total time until the application boards have the correct forwarding table is thus (besides computing time):

$$T_{orig} = D + t + t + t + t \tag{7.1}$$

Using the values from table 7.1 we get data points for the traffic rate and packet loss. With the least square method we can find an approximation to a linear relation between the points.
We get the equation:

$$PL_{orig} = v * 1.5336 \tag{7.2}$$

The implemented delay is one second, and a trip over the backplane takes a packet 0.01 seconds. The expected coefficient is thus at least 1.04. The approximation from the least square function does exceed this limit. The expected packet loss for traffic intensity, from 0-120 pps, is shown in figure 7.3, together with the measured values.
The range of traffic 0-120 pps is chosen because that is what the test traffic was measured.

7.1.3.2 Proposed solution

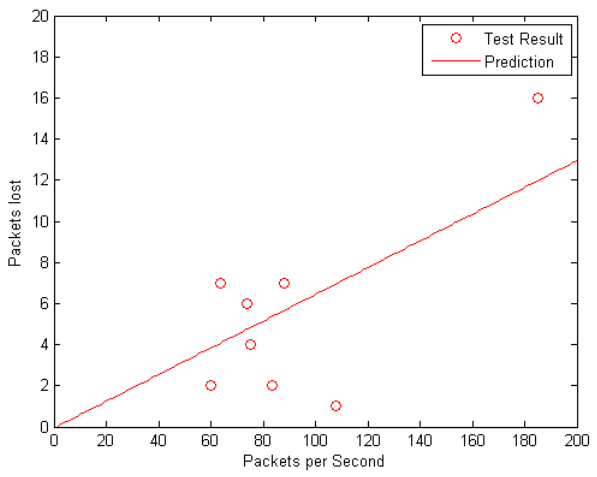


Figure 7.4: *The expected and actual packet loss for traffic 0 - 200 pps*

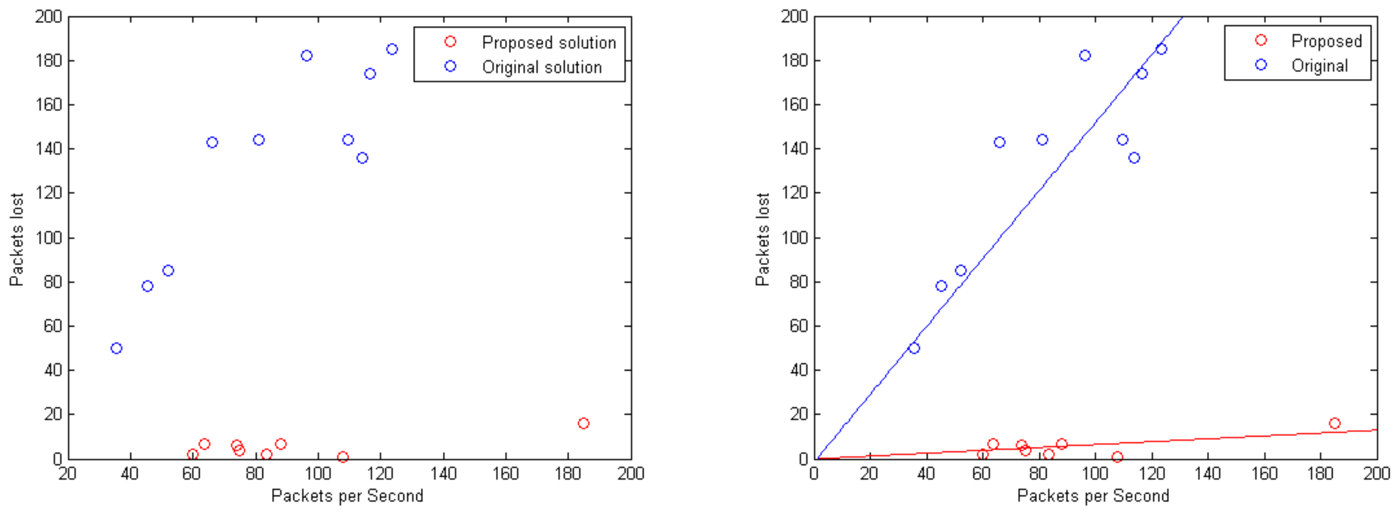
With the proposed solution the router immediately responds with feedback when it can't deliver a packet. Packets sent during the round trip time will be lost. The time frame we are expecting is:

$$T_{prop} = t + t \tag{7.3}$$

As before, a packet takes 0.01 s for one trip over the backplane. The time frame is thus:

$$T_{prop} = 0.02 \tag{7.4}$$

All packets sent during this time frame will be lost. The tests did however show a higher rate of packet loss, except for test number 6. In that test the result should be (according



(a) Test results plotted (b) Test results and linear approximations

Figure 7.5: *Comparison between results from original and proposed solution*

to the equation above) lost 64 packets, but only nine were lost. This is the result of a lot of good luck and is not representative for the algorithm. That data point will be disregarded.

The packet loss is dependent on how much traffic the application sends out during a period of 0.02 seconds. Even though the average traffic rate (during two seconds) is low, the application might send out many packets during one 0.01 s time slot. The traffic is unevenly distributed over time, when we look at the resolution 0.01 s.

We expect that the relation between packet loss and packet speed would be much more linear and accurate with higher traffic rates. We do the least square method to find the best linear relationship among the remaining test results. We find that the best line would be:

$$PL_{prop} = v * 0.065 \tag{7.5}$$

where v is the number of packets per second.

In the graph in figure 7.4 we can see this approximation and the actual test results.

7.1.3.3 Comparison between original and proposed solution

When we compare the test results we see that the proposed solution gives significantly better results. The test results can be seen together in the graph in figure 7.5(a).

Figure 7.5(b) illustrates the test results plus the predictions.

Finally, we can look at the predicted values for traffic rates up to 10 000 pps. It is obvious from figure 7.6 that the proposed solution will result in much lower packet loss, even though it is overly negative.

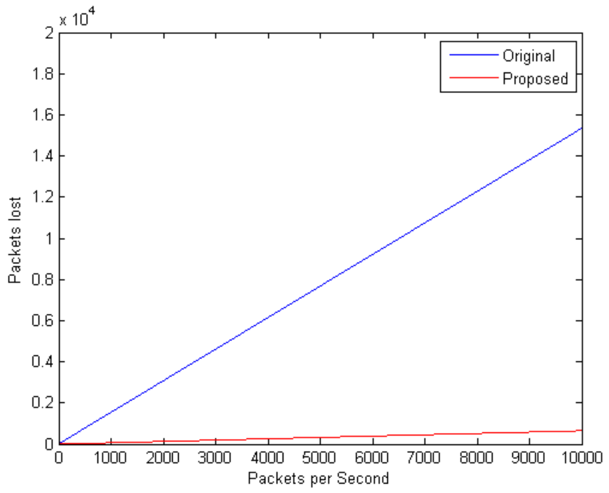


Figure 7.6: *Linear approximations for proposed and original solution for higher traffic rate.*

7.2 Backplane Traffic

The backplane traffic is dependent partly on the number of undeliverable packets and partly on the number of pure routing packets.

We can either just count the packets or also consider the size of the packets. This section will go through the number of packets in the tests followed by a discussion of the size of the packets.

7.2.1 Test results

7.2.1.1 Original Solution

In the original solution, the unnecessary traffic is dominated by the packet loss, see figure 7.7. The monitored traffic is the traffic going in and out of an application. The router does only have one entry in its forwarding table at the beginning of the test, an entry which is removed during the test. If the routing table does not contain any entries, the router will discard the table. All tests resulted in five pure routing messages:

- Notification of an updated table
- Request for the new table

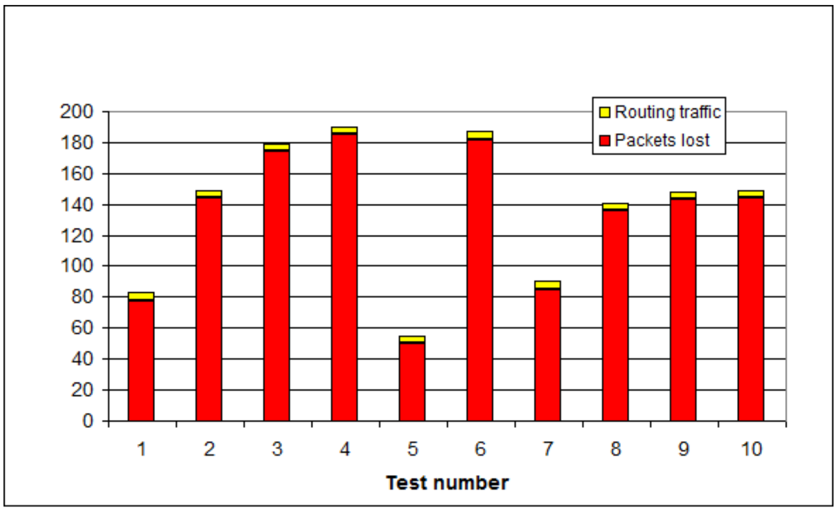


Figure 7.7: *Lost and routing packets - Original solution*

- Router response - it has no table
- Another request for the new table
- Another decline from the router

After these messages, the application knows the router is not to be counted on (for that route).

Test no	1	2	3	4	5	6	7	8	9	10
Packets lost	78	144	174	185	50	182	85	136	143	144
Routing messages	5	5	5	5	5	5	5	5	5	5

Table 7.3: *Unnecessary packets in the original solution*

Test no	1	2	3	4	5	6	7	8	9	10
Packets lost	1	6	16	4	14	9	7	7	2	2
Routing messages	6	11	21	9	6	10	12	12	7	7

Table 7.4: *Unnecessary packets in the proposed solution*

7.2.1.2 Proposed Solution

All tests were performed when the router was newly restarted, why the original routing solution was still in use. This made all pure routing messages that are seen in the tests of the original solution (section 7.2.1.1) are also seen

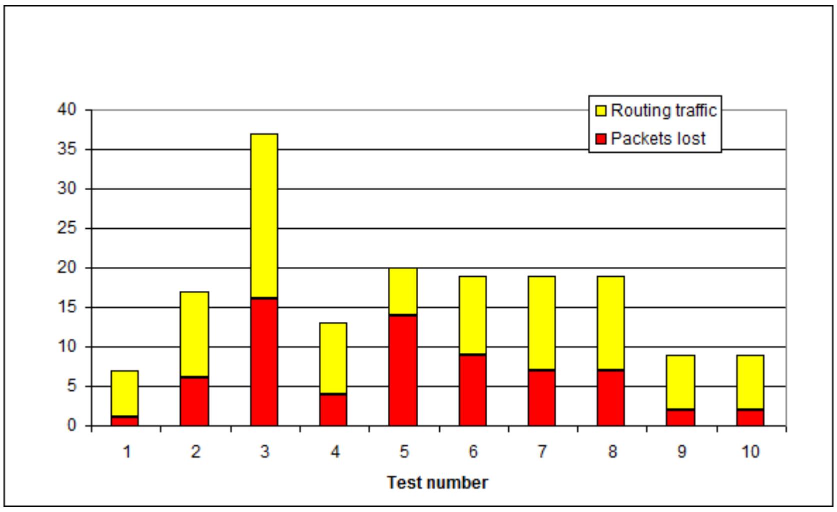
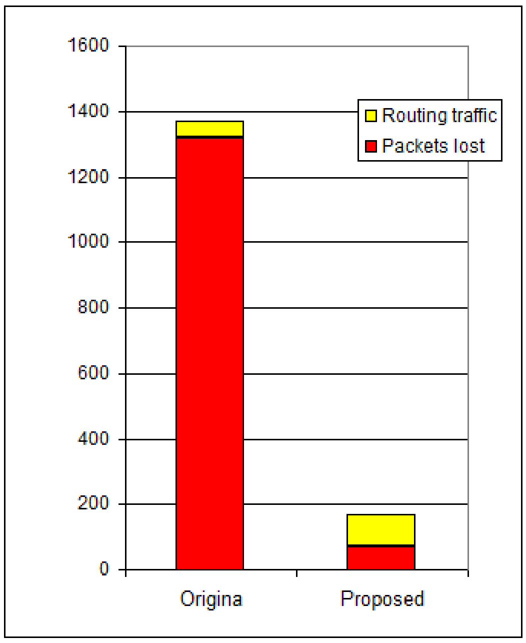


Figure 7.8: *Lost and routing packets - Proposed solution*

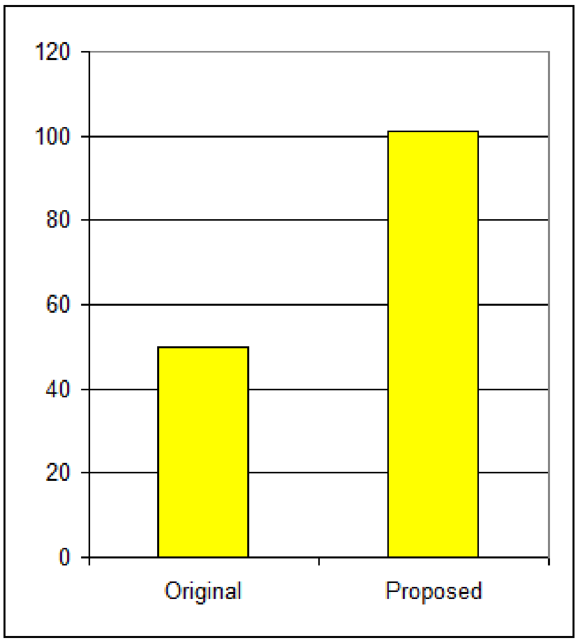
here. In addition to these, there is the ICMP responses to the undeliverable packets, one for each undeliverable packet. The ratio between routing packets and lost packets is illustrated in figure 7.8.

7.2.1.3 Comparison of test results

The sum of the lost packets and the routing packets for all tests is illustrated in figure 7.9(a) for both the original and proposed solution. As we might have expected, the original solution does perform worse, but only due to its large packet loss. In figure 7.9(b) we can see that the pure routing traffic is higher in the proposed solution than in the original. Even though the routing messages that are sent because the recent restart are disregarded, the original solution does result in as many pure routing messages as the proposed.



(a) All routing traffic



(b) Pure routing traffic

Figure 7.9: *Comparison between results from original and proposed solution*

8. Conclusion

Even though the merged memory part was not implemented, the project can be considered a success. The packet loss was significantly better with the proposed solution and with that comes that the backplane did not have to carry a lot of undelivered packets.

8.1 Packet loss

The test results from the proposed solution shows a much smaller time window during which packets sent will be lost. This resulted in fewer packets lost, see figure 8.1.

The maximum number of packets lost was 16 for all tests of the proposed solution. The *minimum* number for the original solution is 50. The *best* result in the original solution is thus more than three times *worse* than the worst result for the proposed solution.

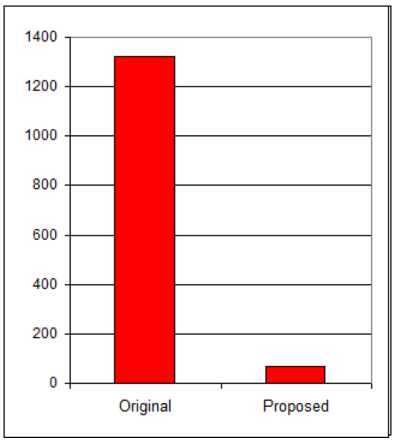


Figure 8.1: *Comparison between original and proposed solution - Packets lost*

The update time will never be shorter than the implemented delay for the original solution while the proposed solution will never have shorter update time than a round trip time of a packet, which is considerable smaller.

8.2 Backplane Traffic

The pure routing traffic is heavier in the proposed solution than in the original solution, if we consider only the number of packets containing pure routing information. The sum of all traffic sent which only contains routing information is seen in figure 8.2.
All packets that are lost contributes to unnecessary load on the backplane.

If these are counted as unnecessary load on the backplane, the proposed solution does perform much better than the original.

8.2.1 Theoretical analysis

The tests might not give a fair picture of the actual traffic situation. There are many things that we need to consider that is not tested:

- Size of the forwarding tables
- Traffic is only calculated for one application
- Tests are too short to cover periodical updates

8.2.1.1 Forwarding tables

When a real node is in use, in a real environment, its router's forwarding tables will contain more than one entry. The tests did only cover the case where the forwarding table only contains one single entry. Therefore it was not interesting to look at the size of the packets sent. However, on a real node, the forwarding table might contain thousands of entries and cause the router's reply to applications to be very big. In the proposed solution all routing messages will only have the size of a standard icmp message.
If all the routes in one network are down, maybe a cable is unplugged, the original solution will only generate one message. The proposed solution will generate one message each time any of the routes are requested.

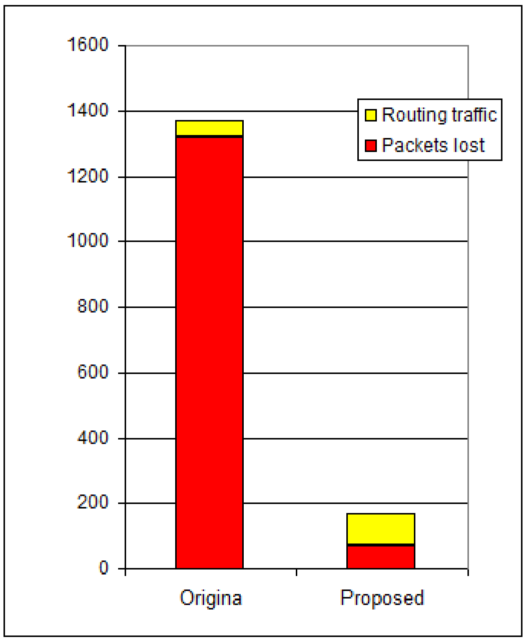


Figure 8.2: *Comparison between original and proposed solution - packet loss and pure routing traffic*

8.2.1.2 Multiple applications

During tests, only a single application was used. In a real environment all applications will send and receive traffic. The pure routing traffic is then proportionally heavier than the measured values in the tests. The expected packet loss is also proportional to the packet loss for a single application.

8.2.1.3 Periodical updates

The tests did not show any sign of periodical updates. In the proposed solution the routers set the state variable every five minutes, announcing their forwarding tables. The original solution does also have periodical updates but the period is 30 minutes. In the original solution the applications request the newest version of the forwarding tables they have.

The routing messages for periodical updates are very few compared to other traffic and can be disregarded in this context.

8.3 Further work

The most obvious next step would be to implement the proposed memory solution with merged tables.
Another step further would be to revise the code together with designers, maybe improve some details that is now overseen. One improvement could be a limit to how many similar ICMP messages the router will send out.
Before the code is a part of the product, it needs to be tested much more throughly. The biggest consideration is that the solution might not be fast enough when the traffic is heavy.

Bibliography

[1] **C. L. Hendrick (1988)** ***Routing Information Protocol*** **RFC 1058**
Document describing the standards to be used when using the routing protocol RIP (version 1).

[2] **J. Moy (1998)** ***OSPF version 2*** **RFC 2328**
Document describing the standards to be used when using the routing protocol OSPF (version 2).

[3] **D. Oran (1990)** ***OSI IS-IS Intra-domain Routing Protocol*** **RFC 1142**
Document describing the standards to be used when using the routing protocol IS-IS (version 1).

[4] **Information Sciences Institute, University of Southern California (1981)** ***Internet Protocol*** **RFC 791**
Document describing the standards for IP version 4

[5] **S. Deering, R. Hinden (1998)** ***Internet Protocol, Version 6 (IPv6)*** **RFC 2460**
Document describing the standards for IP version 6.

[6] **J. Hawkinson, T. Bates (1996)** ***Guidelines for creation, selection, and registration of an Autonomous System (AS)*** **RFC 1930**
Document describing how a autonomous system should be set up, registered and managed.

[7] **Ericsson.com** ***SGSN-MME - Ericsson*** `http://www.ericsson.com/ourportfolio/products/sgsn-mme` **(accessed 2011-03-14)**
Ericsson information about their SGSN-MME.

[8] **Ericsson.com** ***GGSN-MPG - Ericsson*** `http://www.ericsson.com/ourportfolio/products/ggsn-mpg` **(accessed 2011-03-14)**
Ericsson information about their GGSN-MPG

[9] **L. Ekeroth, P-M Hedström (2000)** ***GPRS Support Nodes*** `http://www.ericsson.com/ericsson/corpinfo/publications/review/2000_03/files/2000034.pdf` **(accessed 2011-03-18)**

[10] **DegroupNews** ***Image du dossier : Bouygues Telecom nous ouvre ses portes*** `http://www.degroupnews.com/galerie-D-56-5666.htm` **(accessed 2011-05-26)**

A. Test Results

A.1 Original Solution

Packets Lost	78
Update time	3.9 s
Average pps	20

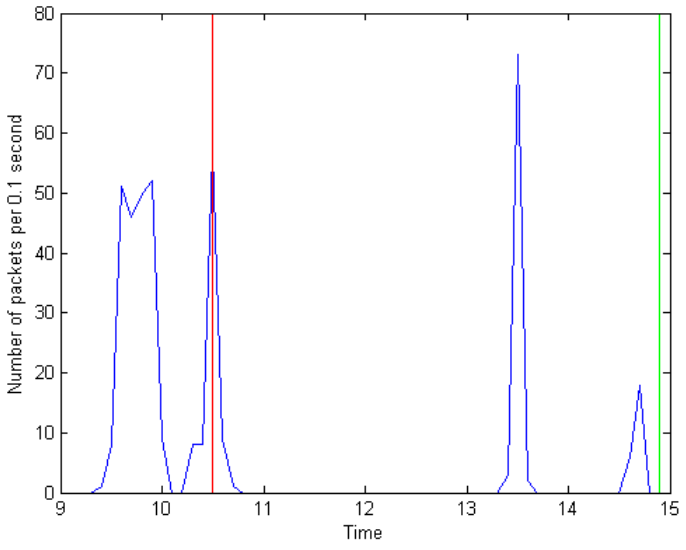


Figure A.1: *Results Test 1*

Packets Lost	144
Update time	2.16 s
Average pps	67

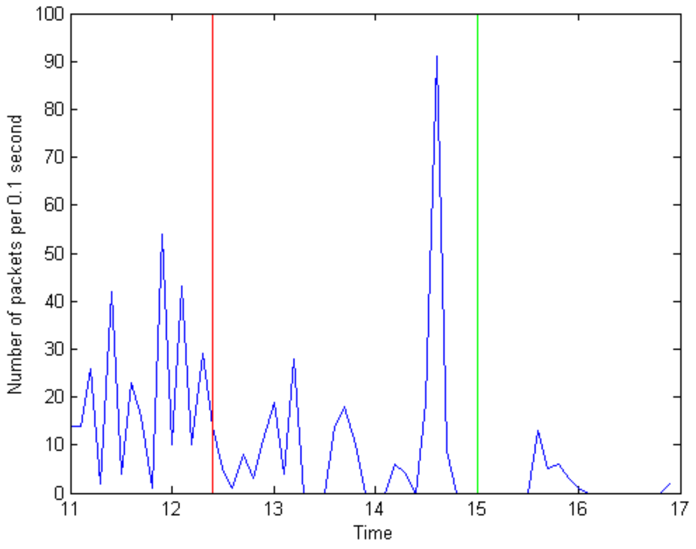


Figure A.2: *Results Test 2*

Packets Lost	174
Update time	2.22 s
Average pps	67

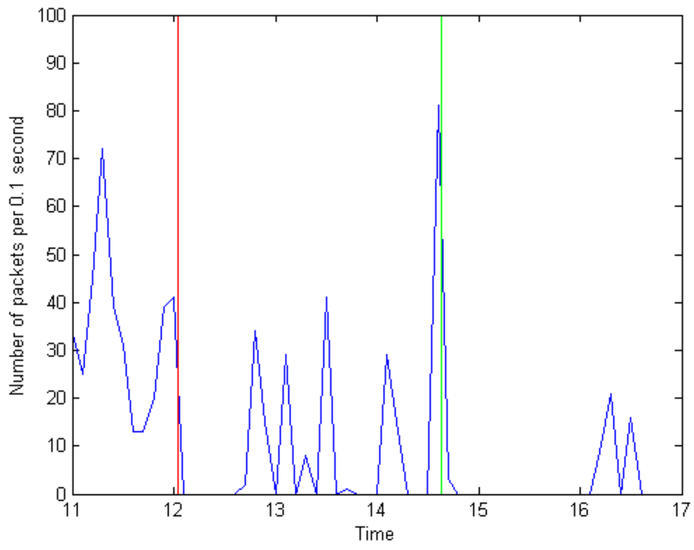


Figure A.3: *Results Test 3*

Packets Lost	104
Update time	1.78 s
Average pps	67

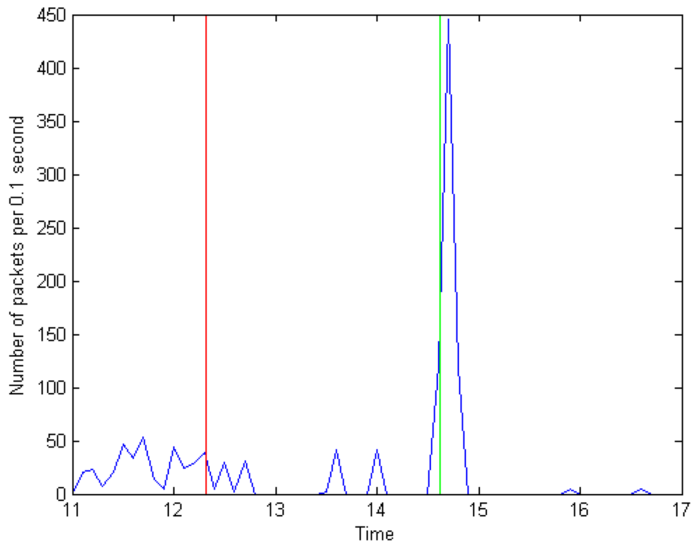


Figure A.4: *Results Test 4*

Packets Lost	50
Update time	2.12 s
Average pps	67

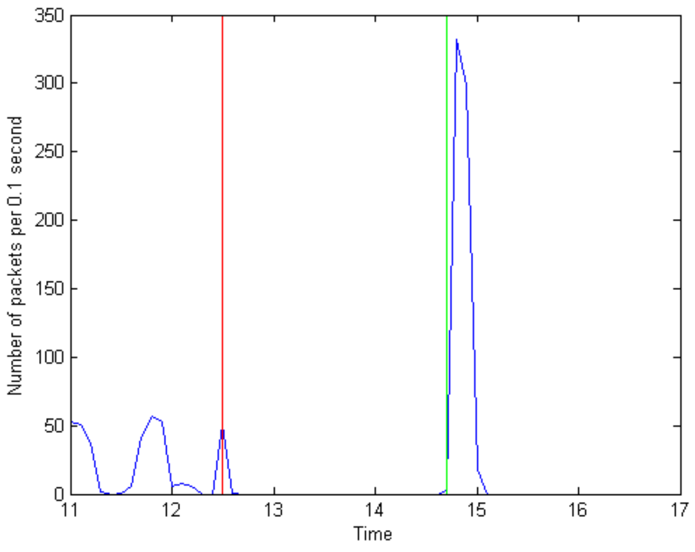


Figure A.5: *Results Test 5*

Packets Lost	182
Update time	2.12 s
Average pps	67

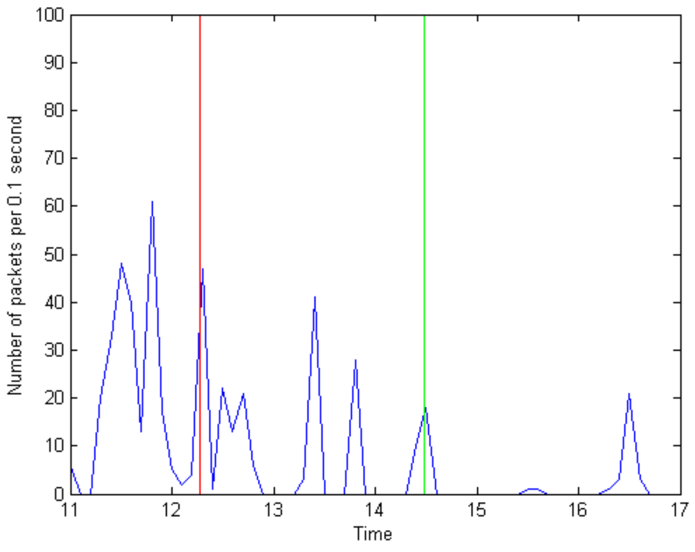


Figure A.6: *Results Test 6*

Packets Lost	85
Update time	2.3 s
Average pps	67

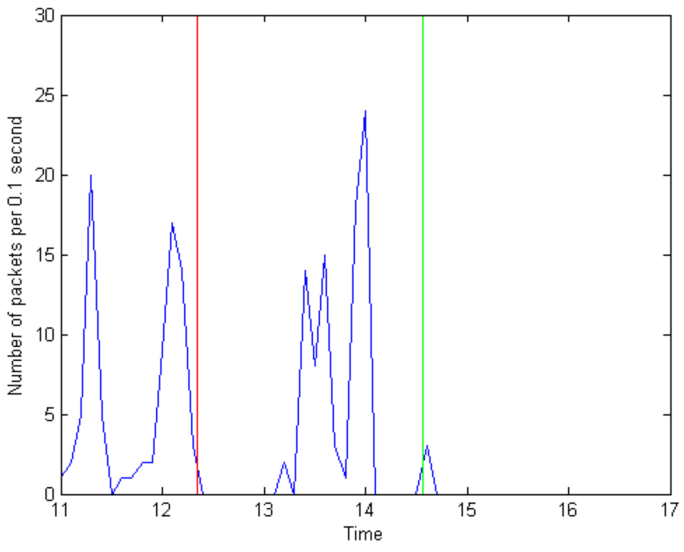


Figure A.7: *Results Test 7*

Packets Lost	136
Update time	2.26 s
Average pps	67

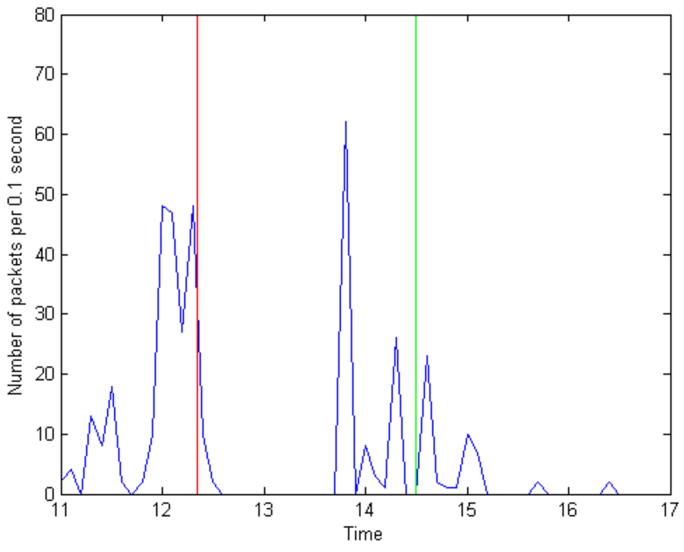


Figure A.8: *Results Test 8*

Packets Lost	143
Update time	2.08 s
Average pps	67

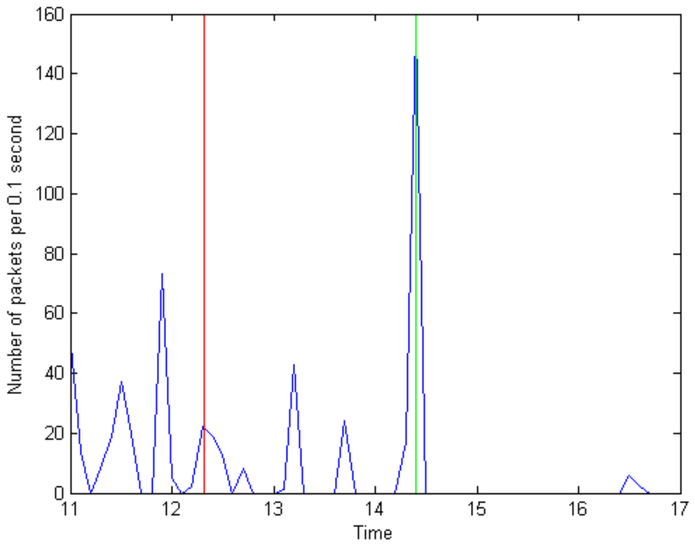


Figure A.9: *Results Test 9*

Packets Lost	144
Update time	2.12 s
Average pps	67

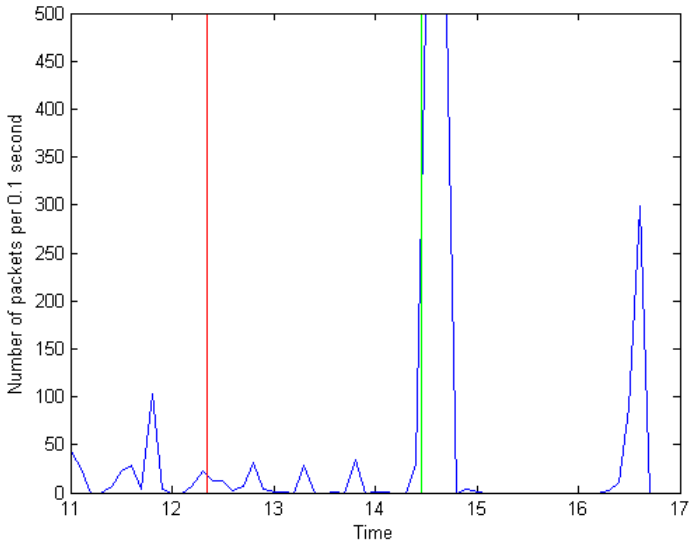


Figure A.10: *Results Test 10*

A.2 Proposed Solution

Packets Lost	1
Update time	0.06 s
Average pps	108

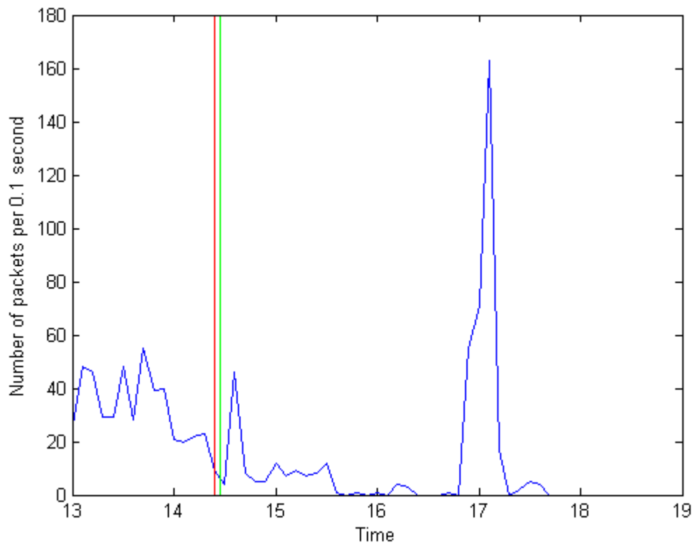


Figure A.11: *Results Test 1 (Proposed)*

Packets Lost	6
Update time	0.98 s
Average pps	74

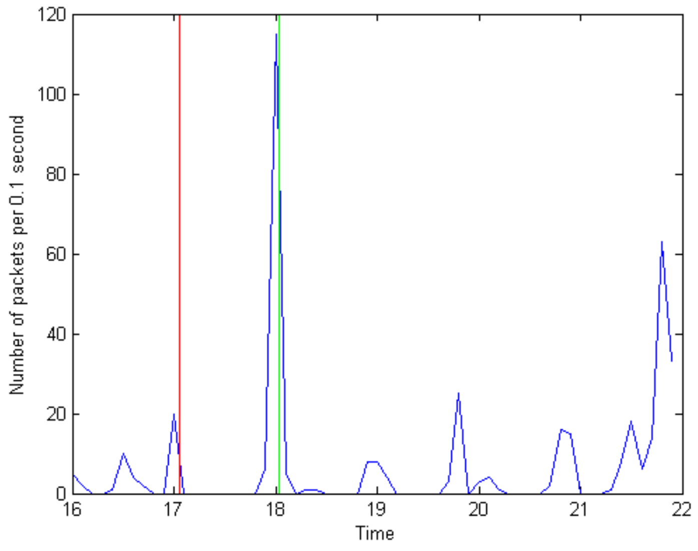


Figure A.12: *Results Test 2 (Proposed)*

Packets Lost	16
Update time	0.04 s
Average pps	185.5

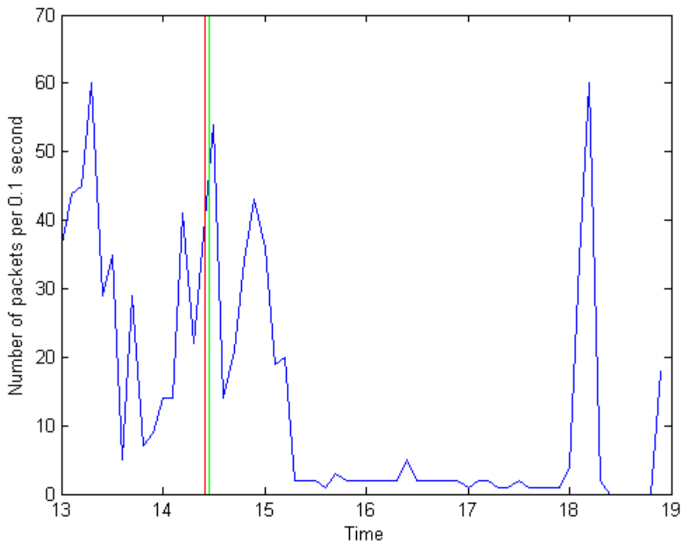


Figure A.13: *Results Test 3 (Proposed)*

Packets Lost	75
Update time	0.32 s
Average pps	75

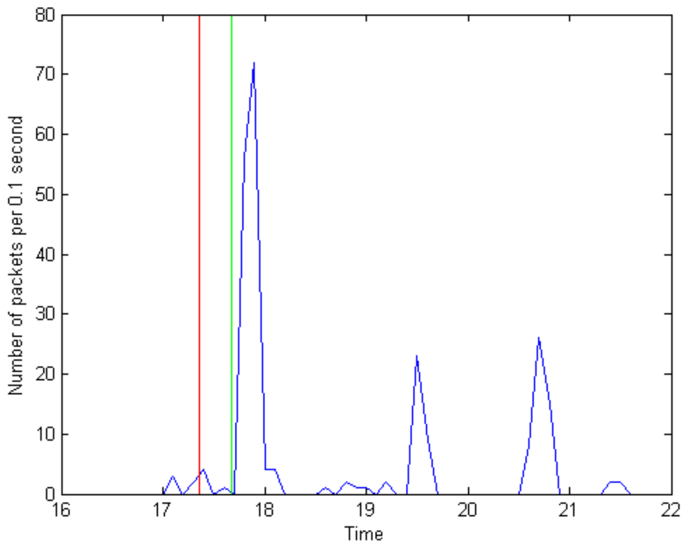


Figure A.14: *Results Test 4 (Proposed)*

Packets Lost	16
Update time	0.04 s
Average pps	53.5

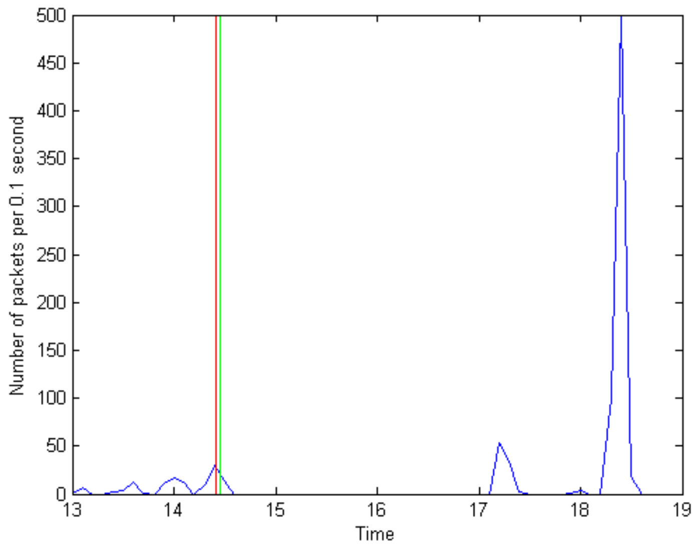


Figure A.15: *Results Test 5 (Proposed)*

Packets Lost	9
Update time	0.24 s
Average pps	3200.5

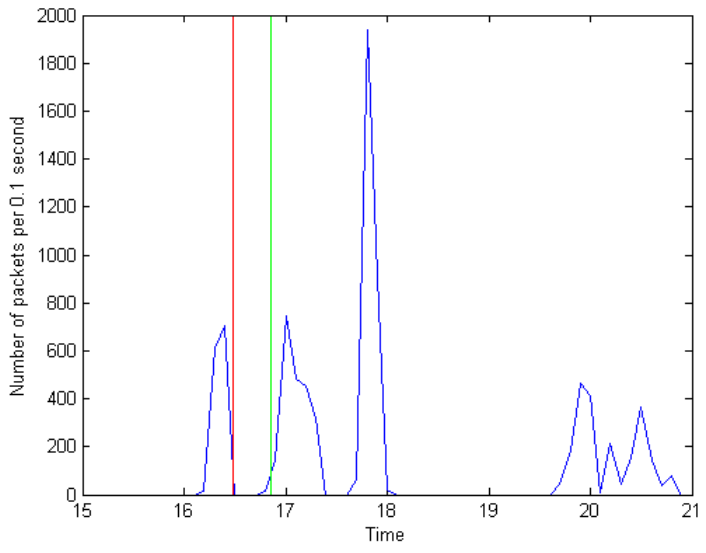


Figure A.16: *Results Test 6 (Proposed)*

Packets Lost	7
Update time	0.04 s
Average pps	88

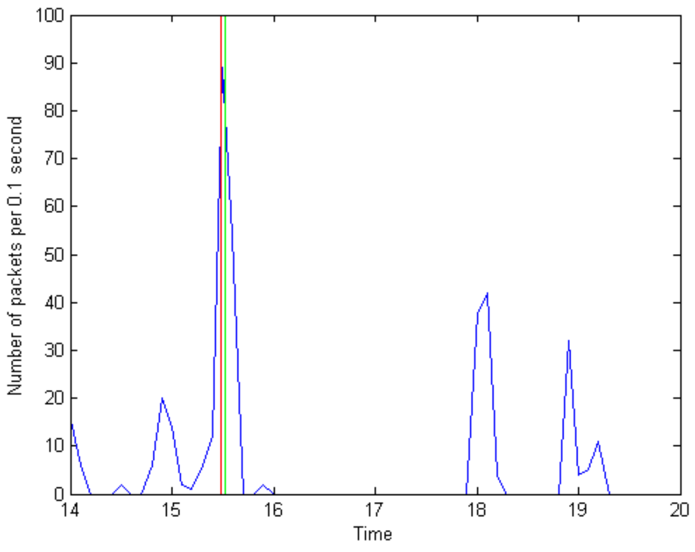


Figure A.17: *Results Test 7 (Proposed)*

Packets Lost	7
Update time	0.92 s
Average pps	63.5

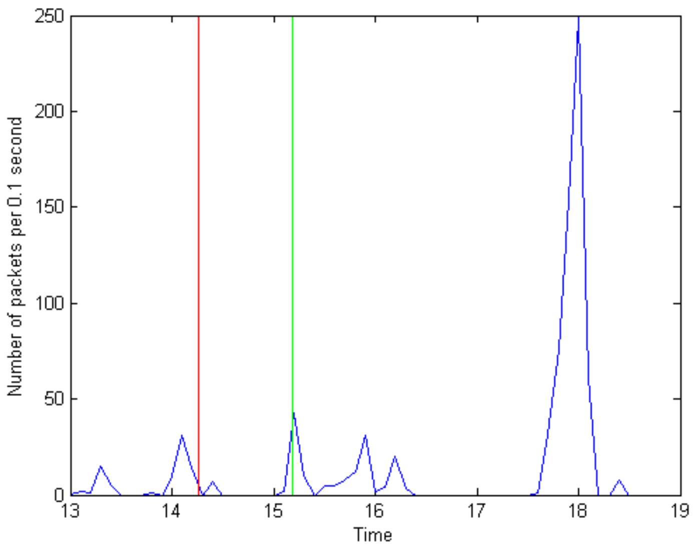


Figure A.18: *Results Test 8 (Proposed)*

Packets Lost	2
Update time	0.24 s
Average pps	60

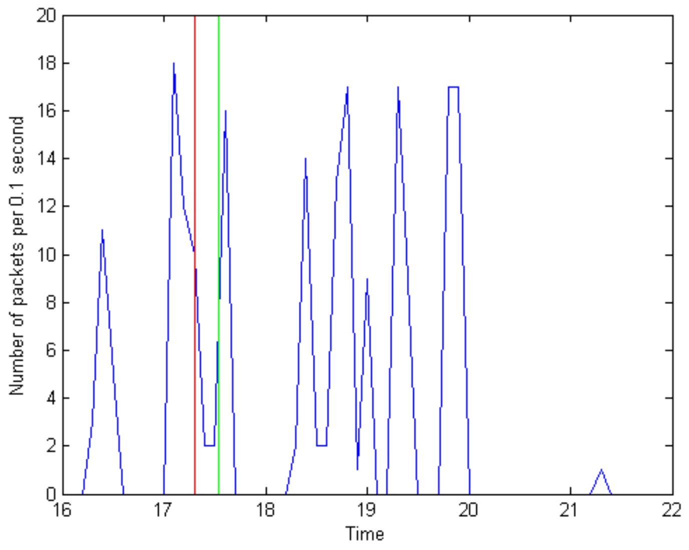


Figure A.19: *Results Test 9 (Proposed)*

Packets Lost	2
Update time	0.26 s
Average pps	83.5

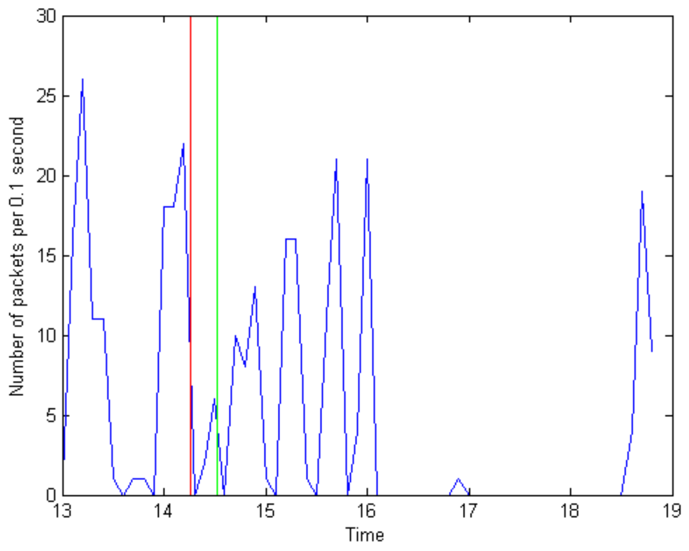


Figure A.20: *Results Test 10 (Proposed)*

B. Source Code

This appendix will present the code parts that was written during this project. [---] indicates omitted lines of code.

B.1 ConnD Changes

```
#define DYNC_RTREQ_TIMEOUT     (5*60) /* cyclic rt req interval: 5 minutes */

[---]


static void ConndTimer_TimeoutHandler(void)
[---]

      /* if the routing table is updated */
      if (g_update_route_table6[i]) {
        /* timer to avoid frequent changes */
        if (tickGet() > g_update_route_table6[i]) {
          rt_InfoShow_v6CreateRoutesLists(i);
          rt_InfoShow_UpdateRtFileIpv6(i);

          /*Route distribution works in RTR only for IPV6*/
          if (!(my_board_type & PEB_BOARD)) {
            /* if the update is successful */
            if (rt_CM_UpdateForwardingTable(i, PROTO_IPV6)) {
                 /* set new timeout*/
      g_dync_route_req_timer[i] = tickGet()/60 + DYNC_RTREQ_TIMEOUT;
      /* set route distribution the first 5 minutes from router start */
      if(tickGet() < 180000)
setSRV = TRUE;
            } else {
              AS_Trace(TRDEBUG, "IPv6:No route distribution change is needed for NW:
            }
          }
          g_update_route_table6[i] = 0;
          rt_InfoShow_v6DeleteRouteLists();
          }
        }

[---]

#ifdef LINUX
  struct timeval current;
  gettimeofday(&current, NULL);
  startTime = current.tv_sec;
#else   /* VxWorks */
  startTime = tickGet()/60;
#endif
  if (my_board_type & ROUTER_BOARD){
/*skip nw is 0, no routes should be download from IB for nw 0 */
    for (i=1; i < MAX_NBR_OF_VPNS; i++) {
      if (g_dync_route_req_timer[i]) {
if (startTime > g_dync_route_req_timer[i]) {
  rt_CM_SetRouteDistributionSRV();
}
g_dync_route_req_timer[i] = startTime + DYNC_RTREQ_TIMEOUT;
AS_Trace(TRDEBUG, "next cyclic route set for VPN %d will occur at %d\n",
 i, g_dync_route_req_timer[i]);
      }
    }
  }
#endif

[---]
```

B.2 FE changes

```
int fe6_internal_input(struct mbuf** m0)

[---]

  if(feBoardTypeRouter()) {  /* Don't do anything on appl. board */

[---]
```

```
    outError = ip6_output (m, NULL, NULL, 0, NULL, NULL);
    if(outError){
      struct mbuf* med;
      Bool handled = FALSE;

      med = fe6_icmp6_addr_unreach(m);

      fe6_inbound_forward(&med,vpn,&handled);
      printf("sent something?\n");
      if(!handled){
m_free(med);
m_free(m);
      }
    }

[---]

  } else if (feBoardTypeApplication()) {
    fixTable(m);
  }


static
struct mbuf* fe6_icmp6_addr_unreach(struct mbuf *m)
{

  struct mbuf *packet = NULL;
  struct ip6_hdr *ip = NULL;
  struct ip6_hdr   *oip  = NULL;
  struct icmp6_hdr *icp  = NULL;
  int icmp_len = 0;
  eq_pos myeq = {2, 9};

  /* take the ip header from the original message */
  if(m != NULL) {
    oip = mtod(m, struct ip6_hdr *);
  }

  /* allocate memory for the packet */
  MGETHDR1(packet, CL_SIZE_256, M_DONTWAIT, MT_HEADER);
  if (packet == NULL) {
    printf("ERROR: Couldn't allocate mbuf\n");
    goto end;
  }


  if(m != NULL) {
    icmp_len += sizeof(struct ip6_hdr);
  }

  /* the size of the packet is the ip header, icmp header,
     ip header again (from the original message) and the position */
  packet->m_len = sizeof(struct ip6_hdr) +
                  sizeof(struct icmp6_hdr) +
                  icmp_len +
                  sizeof(myeq);
  MH_ALIGN(packet, packet->m_len);

  /* Create IPv6 header */
  ip = mtod(packet, struct ip6_hdr *);
  ip->ip6_vfc = IPV6_VERSION; /* = 0x60 */
  ip->ip6_flow = 0;
  ip->ip6_plen = htons(icmp_len + (int)sizeof(struct icmp6_hdr));
  ip->ip6_nxt = IPPROTO_ICMPV6; /* Next header */
  ip->ip6_hlim = 0xFF; /* Hoplimit */

  /* byta plats p src/dst */
  ip ->ip6_dst = oip->ip6_src;
  ip ->ip6_src = oip->ip6_src;


  packet->m_data += (sizeof(struct ip6_hdr));
  packet->m_len -= (sizeof(struct ip6_hdr));

  /* Create ICMPv6 header */

  icp = mtod(packet, struct icmp6_hdr *);

  icp->icmp6_type = ICMP6_DST_UNREACH;
  icp->icmp6_code = ICMP6_DST_UNREACH_ADDR;
  icp->icmp6_cksum = 0;
  icp->icmp6_cksum = in6_cksum(m,
                              IPPROTO_ICMPV6,
                              sizeof(struct ip6_hdr),
                              packet->m_len);

  packet->m_data += sizeof(struct icmp6_hdr);
  packet->m_len -= sizeof(struct icmp6_hdr);


  /* add the original header */
  if(m != NULL) {
    bcopy((caddr_t)oip, (caddr_t)packet->m_data, icmp_len);
  }


  packet->m_data += (sizeof(struct ip6_hdr));
  packet->m_len -= (sizeof(struct ip6_hdr));


  /* add the position of me */
  bcopy((caddr_t)&myeq, (caddr_t)packet->m_data,sizeof(myeq));


  packet->m_data -= (sizeof(struct ip6_hdr) + sizeof(struct icmp6_hdr)) +
                    (sizeof(struct ip6_hdr));
  packet->m_len += (sizeof(struct ip6_hdr) + sizeof(struct icmp6_hdr)) +
                   (sizeof(struct ip6_hdr));
  packet->m_pkthdr.len = packet->m_len;
 end:
  return packet;
}


static int fixTable(struct mbuf* m){
  int              proto = 0;
  struct ip6_hdr*   ip = NULL;
  int returnval = 0;

  ip = mtod(m, struct ip6_hdr *);
  proto = ip->ip6_nxt;
```

```
    if(proto == IPPROTO_ICMPV6){
      struct icmp6_hdr *icp;
      /* struct ip6_hdr *ip; */
      int ip6_hlen = sizeof(struct ip6_hdr);
      int icmp6_hlen = sizeof(struct icmp6_hdr);
      int icmp_type = -1;
      int icmp_code = -1;

      /* find the icmp header */
      m->m_data += ip6_hlen;
      m->m_len  -= ip6_hlen;
      icp = mtod(m , struct icmp6_hdr *);
      icmp_type = icp->icmp6_type;
      icmp_code = icp->icmp6_code;

      /* after the icmp header comes the original header */
      /* the packet the routers couldn't forward */
      m->m_data += icmp6_hlen;
      m->m_len  -= icmp6_hlen;
      ip = mtod(m , struct ip6_hdr *);


      m->m_data += ip6_hlen;
      m->m_len  -= ip6_hlen;


      if((icmp_type == ICMP6_DST_UNREACH)
         && (icmp_code == ICMP6_DST_UNREACH_ADDR)){
        register FeVpn_t* vpn = NULL;
        FeRt6Entry_t* rte = NULL;
        u16 eqid;
        u8 mag, slot;


        /* find vpn for the sent message */
        vpn = feVpnIp6ToVpnLookup(&(ip->ip6_src));
        if (vpn == NULL) {
  return -1;
        }

        /* the data is the position of the sending router  */
```

```
          /* first byte is mag and second is slot */
          mag = *(m->m_data);
          slot = *(m->m_data +1);
          eqid = POSTOEQID(mag,slot,GICPOSITION);

          /* find the entry in the forwarding table */
          rte = adt_RdxTree6_Lookup(vpn->fwd_tbl_in6,&(ip->ip6_src) );
          DeleteRt6EntryAll(vpn->fwd_tbl_in6, rte, eqid);
          returnval = 1;

        }

        /* reposition the pointers */
        m->m_data -= ip6_hlen;
        m->m_len += ip6_hlen;
        m->m_data -= icmp6_hlen;
        m->m_len  += icmp6_hlen;
        m->m_data -= ip6_hlen;
        m->m_len += ip6_hlen;

      }
        return returnval;
    }
```

C. Test implementation

```
tc_004_setup_static_ipv6route(config)->
    ok;
tc_004_setup_static_ipv6route(unconfig)->
    case gtt_testcase:was_successful() of
        yes ->
            ok;
        no ->
            ?CLI_CMD("undo_config_pending")
     end,
    ok;
tc_004_setup_static_ipv6route(exec)->
    ?CLI_MATCH("modify_sctp_profile -pn Pn_2.5 -bundling 0",[]),
    ?CLI_MATCH("create_eth_port -eqp 2.9 -ep 0",[]),
    create_ip_interface("188","elisabet","2088::1",
                                   "S1-MME-VPN1","elsa_gtt","2088::2"),

    create_static_route("S1-MME-VPN1","2027::","2088::2"),

    ?CLI_CMD("check_config"),
    ?CLI_OBM_MATCH_UNDO("activate_config_pending", []),
    ?SLEEP(2000),

    send_packet(exec),
    ok.



test(config)->
    ok;
test(unconfig)->
    case gtt_testcase:was_successful() of
        yes ->
            ok;
        no ->
    gtt_dp_control:send_command_sgsn_ss7("be5TraceStop",[1500]),
    gtt_dp_control:wait_command_response(),

    gtt_dp_control:send_command_gtt_dp("ethTraceStop",[0]),
    gtt_dp_control:wait_command_response()

     end,
    ok;
test(exec)->

    gtt_dp_control:send_command_gtt_dp("ethTraceStart",[1500]),
    gtt_dp_control:wait_command_response(),

    gtt_dp_control:send_command_sgsn_ss7("be5TraceStart",[1500]),
    gtt_dp_control:wait_command_response(),

    ?SLEEP(2000),

    ?CLI_CMD("delete_static_ipv6route -eqp 2.9 -nw S1-MME-VPN1
                          -dip 2027:: -pl 64 -gip 2088::2 "),
    ?CLI_CMD("check_config"),
    spawn(ervkat_esveeli,spawn10,[20]),
    ?SLEEP(2000),
% remove the route
    ?CLI_OBM_MATCH_UNDO("activate_config_pending", []),

% sleep one minute
    ?SLEEP(60000),

    gtt_dp_control:send_command_sgsn_ss7("be5TraceStop",[1500]),
    gtt_dp_control:wait_command_response(),

    gtt_dp_control:send_command_gtt_dp("ethTraceStop",[0]),
    gtt_dp_control:wait_command_response(),

    ok.

hej(0)->
    ok;
hej(N)->
    spawn(ervkat_esveeli,create_and_delete,[]),
        hej(N-1).

    spawn10(0)->
        spawn(ervkat_esveeli,create_and_delete,[]);
    spawn10(20)->
        ?SLEEP(5000),
        spawn10(19);
    spawn10(N)->
        ?SLEEP(700),
        hej(30),
        spawn10(N-1).


    create_and_delete()->
        ?SLEEP(500),
        UeId =  gtt_mme_library:create_ue(?CONF(location_eriksbo)),
        {UeId,SessionId} = gtt_mme_library:attach(UeId),
        ?SLEEP(1000),
        gtt_library:detach({UeId, SessionId}, normal_detach).
```