UiO : Department of Informatics
University of Oslo

A Demonstration Scenario for the NorNet Core Multi-Homed Network Testbed

Henrik Vest Simonsen

Master's Thesis Spring 2014

A Demonstration Scenario for the NorNet Core Multi-Homed Network Testbed

Henrik Vest Simonsen

20th May 2014

Abstract

There is extensive on-going research taking place with the goal of implementing transport layer protocols that are able to utilize multi-homing on machines (connection to multiple ISPs). Multi-homing has potential to improve resilience and increase total goodput compared to using a single connection. This would e.g. benefit the trend of more and more services being moved online and into the cloud.

The NorNet Core multi-homed network testbed aims to facilitate this research by offering programmable virtual machines located at multiple different sites across the world, accessible for researchers.

The aim of this project is the design and implementation of a demonstration platform for NorNet Core that enables the testbed to be illustratively demonstrated to potential new users. This is done by offering users interactive experiments, and geographical visualization of various communication scenarios between nodes in the NorNet Core network in real time.

The demonstration platform puts emphasis on demonstrating the unique aspects of NorNet Core compared to other networking testbeds; namely the multi-homed nature and IPv6 support for all of the nodes.

This thesis discusses the technical aspects of designing and implementing the platform, as well as demonstrating the finished result with various demonstration scenarios. The Qt programming framework was chosen as the primary development framework for this application.

Contents

Acronyms

API Application Programming Interface. 8

CMT-SCTP Concurrent Multipath Transfer SCTP. v, 4, 8

DNS Domain Name System. 4

GPL GNU General Public License. 10

GRE Generic Routing Encapsulation. 5

ICMP Internet Control Message Protocol. 18

IDE Integrated Development Environment. 10

IETF Internet Engineering Task Force. 3, 8

IP Internet Protocol. 8

IPv4 Internet Protocol version 4. 5, 7, 18–20

IPv6 Internet Protocol version 6. v, 5, 7, 18–20

ISP Internet Service Provider. 4, 19

LGPL GNU Lesser General Public License. 10, 31

MPTCP Multipath TCP. 3, 4, 6, 8, 21

PLC PlanetLab Central. 6

RR Resource Record. 28

RTT Round-trip time. 17, 37, 40

SCP Secure Copy. 30

SCTP Stream Control Transmission Protocol. 3, 4, 8, 9, 21

SSH Secure Shell. 6, 27, 29, 30, 35, 43, 46

TCP Transmission Control Protocol. 6, 8, 18

UDP User Datagram Protocol. 8

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors at Simula Research Laboratory, Priv.-Doz. Dr. Thomas Dreibholz and Dr. Ernst Gunnar Gran, for providing guidance and inspiration throughout the process of completing this project. My thanks also goes out to my co-supervisors at the University of Oslo, Prof. Stein Gjessing and Prof. Olav Lysne. Finally, I would like to thank my family for the encouragement and support.

Part I

Introduction

Chapter 1

Background

In this chapter I am going to briefly present the background information related to this master project. The two first sections, *Multi-homing* and *NorNet*, are the two most important sections; The section *NorNet* presents the platform I will be working with for this project, and the section *Multi-homing* touches upon the motivation behind NorNet Core. In the final section the problem statement will be presented.

1.1 Multi-homing

The term multi-homing is used to describe computers that are connected to the Internet through more than one connection, having separate IP addresses for each connection. It has potential to improve resilience of network dependant systems: if one connection goes down, another may still be active and usable.

However, the most common transport layer protocols of today (TCP and UDP, see Section 1.4 on page 7) are not taking advantage of multi-homing. Using these single address-to-address transport layer protocols, the handover from one connection to another has to be done on the application layer [1]. Existing programs would have to be heavily customized to deal with multi-homing, which of course is not feasible in most cases.

On the other hand, transport layer protocols that take multi-homing into account, put the work of managing multiple connections behind the scenes and into the kernel, and requires little to no changes to the application layer logic. One such protocol is Stream Control Transmission Protocol (SCTP)(Section 1.4.1). It is a part of the Linux kernel and takes advantage of all available connections to achieve improved resilience to network failure. SCTP is discussed in more detail in Section 1.4.1

Another up-and-coming multi-homing protocol is Multipath TCP (MPTCP) [10]. Its specification was published by the Internet Engineering Task Force (IETF) as an experimental standard in January of 2013. There are several implementations being worked on. It will be further discussed in Section 1.4.3

Another use case of multi-homing is load sharing. It involves using

several connections simultaneously, with the goal of getting an increase in total goodput compared to using a single connection. Load sharing is a part of the MPTCP specification, but it is not a part of SCTP. There is however an extension to SCTP in the works known as Concurrent Multipath Transfer SCTP (CMT-SCTP) [2], that offers load sharing. CMT-SCTP will be briefly covered in Section 1.4.2

In summary, multi-homing has the potential to improve throughput and stability for clients and servers. In fact, every smart phone owner could potentially benefit from an adoption of multi-homing, as it could merge cellular data connection and Wi-Fi connection into a single endpoint.

1.2 About NorNet

NorNet [11] is a distributed network testbed, developed by Simula Research Laboratory. The testbed consists of two branches: NorNet Core and NorNet Edge. NorNet Edge [12] is a platform for doing experiments related to mobile broadband networks, as well as doing measurements on these networks.

This master's project however, is only related to the NorNet Core branch, but I may in some cases refer to NorNet Core simply as NorNet.

NorNet Core was conceived with the main goal in mind of allowing researchers to do research related to multi-homing. This is realized by offering users remote access to multiple sites across Norway (and some worldwide), where each site is connected to more than one Internet Service Provider (ISP). Researchers are able to deploy and run their experiments on these sites.

1.2.1 NorNet Core Architecture

NorNet Core's architecture and software is based on that of PlanetLab[1] [15], which is a global networking testbed with more than 1170 nodes. However, NorNet Core also wants to improve upon the Planetlab architecture. The main improvement is the goal and design of connecting each node in the NorNet Core network to at least two ISPs.

The NorNet Core network currently (20th May 2014) consists of 14 sites [6]. 11 of which are in Norway, and 3 of which are located abroad, in Sweden, Germany and China. Figure 1.1 show's the location of 9 of these nodes in Norway. Each site in NorNet Core consists of a number of nodes. Most of these are *research nodes*. These are nodes that will be running the experiments of the researchers.

Each site also includes a node denoted as the *tunnelbox* [8] (or *Tbox*, figure 1.2). It does tunnelling and works as a Domain Name System (DNS) [14] server. Another node, named the *control node* provides access to the rest of the nodes on the local site(labled *ctrl*, figure 1.2 on page 6), and is for maintenance and monitoring purposes. [11]. Figure 1.2 shows

[1] http://www.planet-lab.org/

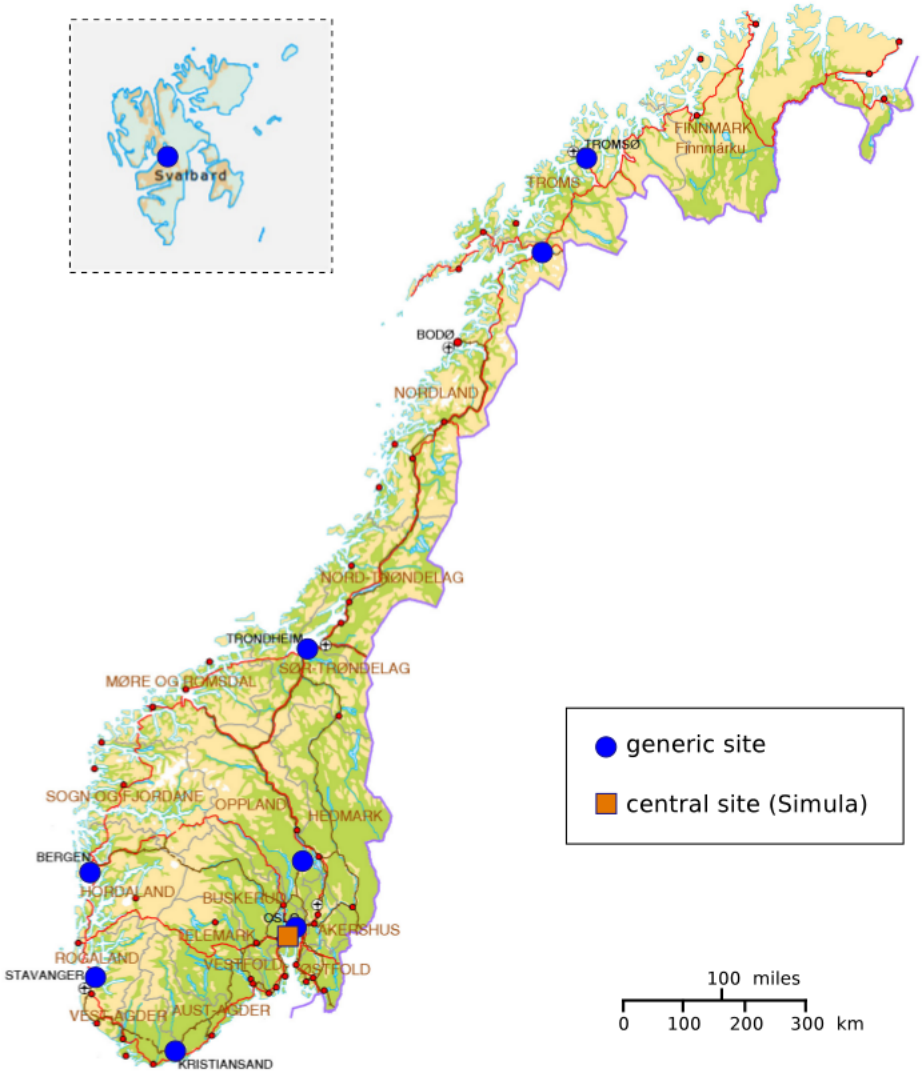


Figure 1.1: Site map of NorNet (NorNet Core, [11])

the internal architecture of a site, and how it is connected to the NorNet network.

NorNet Core supports IPv6 (see Subsection 1.3) for all nodes. For sites where IPv6 is provided by the ISP, IPv6 is tunneled through IPv6. If IPv6 is not supported by an ISP, the NorNet Core system encapsulates the IPv6 packets inside Internet Protocol version 4 (IPv4) packets (which are then tunnelled), by the help of the Generic Routing Encapsulation (GRE) [9] protocol. A routing table is chosen based on the source address.[11][Subsection 4.4] IPv6 support makes NorNet Core a good platform for doing IPv6 related research.

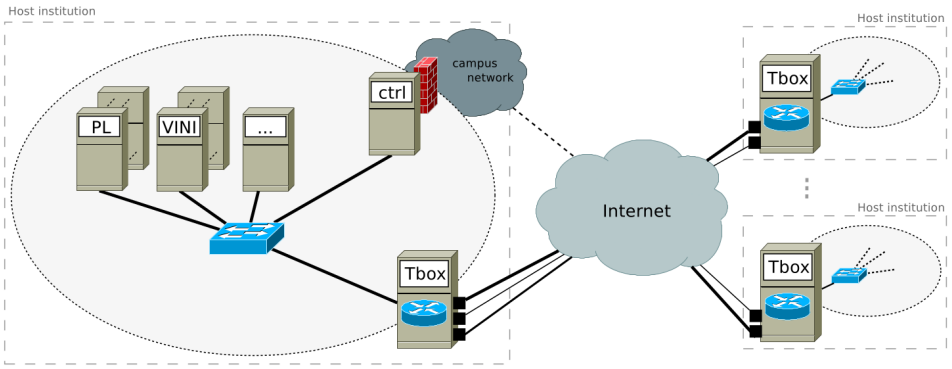


Figure 1.2: Architectural overview on a site. (NorNet Core, [11])

1.2.2 Slices, Slivers and Secure Shell (SSH)

NorNet Core is based on PlanetLab [15], which makes many of its concepts familiar to PlanetLab users. The users will have access to virtual Linux Systems, through the concept of slices. A user with access to slice a slice will be able to log into any of the available nodes, being part of the slice, with an Secure Shell (SSH) [19] command, specifying slice name as well as node name. These individual resources on each node are called slivers. For user verification, the public key of an RSA encryption [17] key set has to be uploaded to the PlanetLab Central (PLC) server through the PLC web interface. The SSH client needs to know the file location of public key's private key counterpart.

Currently (May 2014), users that are not making sliver connections from within the NorNet Core network will have to first remotely log into a server that is available also externally to the NorNet Core network. This server is denoted as the *gatekeeper* [5] server. Each researcher will be provided with SSH access to this server. Once connected, the user is free to connect to NorNet Core slivers from within the gatekeeper.

Although all the slivers on a single node are isolated from each other, they do share the same kernel. This means research based on a custom kernel would require a non PlanetLab based solution. The NorNet Core testbed is based on virtual machines, making it flexible, easy to maintain and quick for deploying alternative setups. However, currently deployment of custom kernels or OS is not automated[8, p. 5].

1.2.3 MPTCP availability

A recent addition to NorNet Core is experimental access to MPTCP on several of the research nodes. Currently, all nodes with an even node index [8] has MPTCP enabled. However, MPTCP will only be available if both of the nodes communicating support MPTCP. If one or both of the nodes do not support MPTCP, normal Transmission Control Protocol (TCP) will be used.

1.3 Internet Protocol version 6 (IPv6)

IPv6 [4] was introduced as a means to solve the limited number of possible addresses in IPv4 and the 32-bit address space. IPv6 uses 128 bit addresses, which gives an address space that is unlikely to run out any time soon. Another improvement of IPv6 is the reduction in the number of fields in the header (7 in IPv6 vs 13 in IPv4). This simplifies parsing of headers at routers. IPv6 also has better support for options, and several of the pruned fields from IPv4 are optional in IPv6. Security and quality of service are two other areas of focus in IPv6.

Although IPv6 has been an Internet standard since 1998, IPv4 is still the far more common protocol for normal users; according to statistics from Google[2], at the current time (May 2014), only around 3 percent of Google users connect through IPv6. However, the curve seems to have an exponential trend. The graph is shown in figure 1.3.

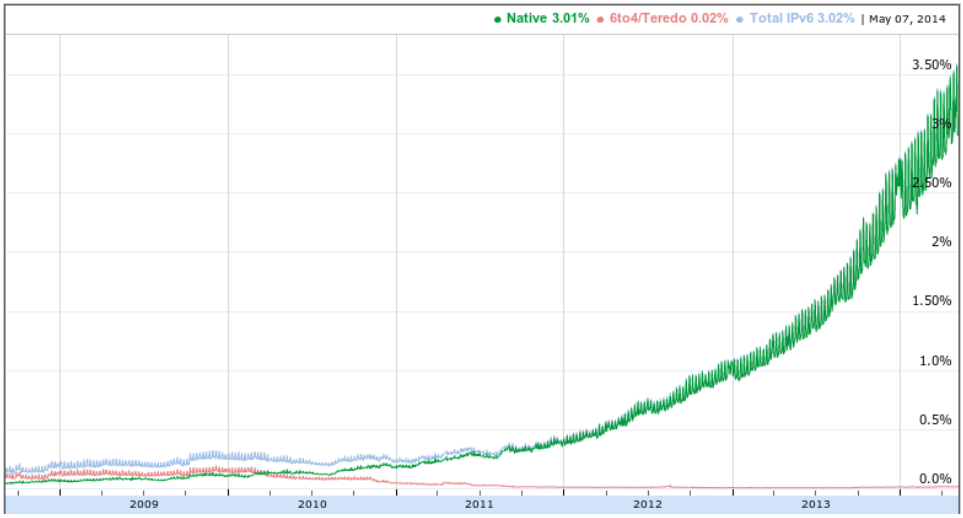


Figure 1.3: The trend of IPv6 over the years.
Google and the Google logo are registered trademarks of Google Inc., used with permission.

The reasons for the slow adoption of IPv6 is that IPv6 is not backwards compatible with IPv4, and middleware is still not able to cope with IPv6 at the same level as IPv4. Tunneling IPv6 through IPv4 networks is currently used at the transitional phase, but it is less than optimal. Still, IPv6 is steadily gaining ground, as IPv4 addresses are depleting, and middleboxes upgraded.

1.4 Existing MultiPath Technologies

The multi-homed nature of NorNet Core gives good opportunities for testing transport layer protocols that can utilize a node being connected to several ISPs. In this section I will briefly introduce the two biggest multi-homed protocols.

[2] https://www.google.com/intl/en/ipv6/statistics.html

1.4.1 Stream Control Transmission Protocol (SCTP)

SCTP [18] is a message-based transport layer protocol. It was conceived by a IETF working group, and the motivation behind it was creating a protocol for telephony signalling over Internet Protocol (IP) networks. They identified 4 main weaknesses with the TCP protocol for this purpose [18].

1. TCP's forces in-order delivery of data. This causes unnecessary delay for applications where sequence of the messages is not important.
2. TCP is stream oriented. Parsing of messages in the data stream requires application level logic.
3. TCP has no multi-homing support, and making it multi-homed would prove difficult because of its limited scope.
4. TCP is vulnerable to denial of service attacks.

The IETF recognized that these changes from TCP could clearly be useful not only for *Public Switched Telephone Network* (PSTN) signalling, but also for a variety of different communication scenarios. Support for SCTP has been implemented in the Linux kernel since version 2.6. This includes integration of SCTP with the Socket Application Programming Interface (API) used by User Datagram Protocol (UDP) and TCP.

It is worth noting that in the native SCTP used by the kernel, and specified by IETF, multi-homing is used as a means to improve resilience and stability, not to do load sharing: an alternative path is used only if the the active one goes down.

1.4.2 Concurrent Multipath Transfer SCTP (CMT-SCTP)

There is an IETF draft for an extension to SCTP called CMT-SCTP [13], that is currently being implemented. Its goal is to enable SCTP to be used to increase throughput as well as improve resilience, by enabling load sharing across multiple paths.

1.4.3 Multipath TCP (MPTCP)

MPTCP is a relatively new addition to the multipath world. Its specification was published by the IETF in January of 2013 [10]. The idea behind it was getting the benefits of using multiple paths, while at the same time avoiding defining a whole new transport layer protocol. A new protocol would mean problems with existing applications, as well as middleboxes on the Internet. The extension to TCP is done by using TCP's variable length options field for attaching the extra information necessary for a MultiPath solution. It is a backwards-compatible extension to TCP, allowing existing TCP based programs to utilize MPTCP without any changes: If MPTCP is enabled system wide, multi-homing for a program is available even if the program is only using the standard TCP socket API calls. This is an advantage over SCTP, which requires linking with an extended socket API.

There are already quite a few MPTCP implementations. One of note is the open source Linux implementation from UCLouvain.[3]

1.4.4 Existing tools for measuring

There are some Linux tools available for measuring point to point network performance that could be relevant to this project. One such tool is iperf [4]. It works by setting up a server listening for connections on one node, and connect to it on another node, specifying which protocol to use, how much data to send or how many seconds to send for, and how often it should report the results. To accommodate MPTCP and SCTP, it would have to be extended(I think there is already an extension for SCTP). It would also have to extended it if I wish to send the intermediate results, live, over TCP.

Another option is to redirect the output to an intermediate program, that parses the information and sends it through TCP. Another possibility is netperfmeter[5][7], which supports SCTP, but the output seems to be harder to parse than that of iperf.

1.5 Existing Planetlab based tools

PlanetLab is an extensively used testbed, and there exists a lot of Open Source software for it, for managing it and running experiments. These programs could possibly be made to work with the NorNet Core testbed without too much refactoring [6]. I am going to list two of these program, as they are closely related to NorNet Core, and also this project.

1.5.1 Plush with Nebula GUI

Plush[7] was developed to ease the deployment and running of applications on large scale distributed systems. The Nebula GUI gives a map view of nodes.

1.5.2 PlanetLab Experiment Manager

PlanetLab Experiment Manager[8] should also be able to give a GUI overview over nodes, and the options to deploy, run and monitor applications across multiple nodes.

1.6 Qt Framework

Because Qt was chosen as the development platform in this project, this section will briefly discuss some of its features.

[3] http://mptcp.info.ucl.ac.be/
[4] http://sourceforge.net/projects/iperf/
[5] http://www.iem.uni-due.de/~dreibh/netperfmeter/
[6] http://www.planet-lab.org/tool
[7] http://plush.cs.williams.edu/
[8] http://www.cs.washington.edu/research/networking/cplane/

Qt[9] is a programming framework maintained by Nokia. It is open source and under GNU General Public License (GPL) and GNU Lesser General Public License (LGPL) license. It aims to offer its users a cross platform C++ programming environment, through system specific compilation.

As an example, this means that unlike with a Java program, a Qt program compiled for windows will not work on a Linux based system (and the other way around). The compiled Qt file is a binary file that does not require any special run-time environment to be executed. It only requires that it is executed on the system it was compiled for.

That being said, Qt programs do contain several Qt library dependencies. These need to either be available on the target computer, or alternatively, be statically compiled into the binary file.

To deal with the problems related to cross platform support, Qt introduces replacement classes for several cross platform problem areas such a I/O and networking. It also introduces improved versions of classes, as well as as many useful new classes. A big portion of Qt's library is related to GUI: There are classes for all kinds of GUI elements such as buttons, text areas, scrollbars, web pages, etc.

Qt also offers a fully fledged cross platform Integrated Development Environment (IDE), which is programmed using the Qt framework itself. It simplifies the process of creating the user interface, by allowing developers to interactively place the elements on a window canvas. Qt also introduces novel programming language constructs such as the signal and slots mechanism, which will be briefly introduced in Section 1.6.1.

1.6.1 Signals and Slots

Signals and slots is a Qt specific mechanism gives call-back functionality between objects. For one object to call a method of another object, the requirement is that the signal method being emitted from the first object matches the slot method in the second object. Additionally, these two methods needs to be connected together, which can be done externally to both objects in this way:

```
QObject::connect(object1, SIGNAL(newData(int)),
        object2, SLOT(handleNewData(int)));
```

In the above example, if object1 *emits* the signal `newData`, with an `int` variable, this int variable will be received by object2 through object2's own `handleNewData` method. Any number methods from both the same object and different objects can be connect to a single signal. It enables the programmer to completely decouple classes, while at the same time allowing for the call-back functionality.

In the class declaration of an object, signals are defined under a *signals* section, with the same syntax as a normal *void* method. These signals are emitted (called) when the programmer wishes so, with the use of the 'emit'

[9] http://qt-project.org/

keyword. Any slots connected to this signal will then subsequently be called. Slots are defined in the same way under a *slots* section in the class declaration.

Signals can be connected with slots if their parameters match. The arguments must have the same type, but the slots may have fewer arguments (The rest of the signal's arguments are then ignored). A way to imagine signals and slots is just as normal method calls, where the signal method emitted represent all of the slots connected to this signal.

Most of Qt's own objects offer the signals and slots interface. It gives flexibility, but it is not as quick as doing a callback on a function. For performance dependent software, too many signal and slot connections may bog the system down unnecessarily. For most cases, this should not be a problem.

Another good thing about signals and slots is how they behave with objects in different threads; when a signal is emitted, instead of directly accessing slot of the object in another thread, the request to execute the slot is put in a queue of pending jobs. This ensures a thread safe callback system when working with objects in different threads. Direct calls may be specified, if the programmer wants to deal with thread safety issues manually.

Qt also supports connecting signals to functions (not only methods), including connections to the relatively recently introduced anonymous lambda functions in C++11.

1.7 Problem statement

The goal of the thesis project is to create a program that is able to demonstrate how the NorNet multi-homed distributed testbed works in action. It requires understanding of the NorNet Core system on a detailed level, as well as knowledge about the current research related to multi-homed networking.

Through this work I wish to clarify the following question:

1. How can the NorNet Core Distributed Network testbed be demonstrated most efficiently?
2. What are the technical difficulties, and how can they be solved?

Some other personal objectives through the work with this thesis:

1. Familiarizing myself further with C++ and Linux
2. Learning how to use the Qt framework

Part II

The project

Planning the project

Finishing this project requires work in several stages. One important stage is defining requirements for the system. At least a few of these should be defined before starting programming. Then there' is the design stage, going into the depth of how the system should work on a technical level. The remaining stage is the actual implementation.

For the purpose of presenting the project, I am going to organize the presentation of my results into the following chapters:

- Requirements: In the Requirements chapter I am going to discuss what the system needs to do from a user perspective, with a foundation in the problem statement.
- Design: In the Design chapter I am going describe the technical aspects of how to design a system that meets the requirements. These description will include challenges along the way, and the thought-process behind design choices.
- The final two chapters will describe the GUI and the source code respectively.
- A Demonstration scenario: -

Chapter 2

Requirements

In this chapter I am going to discuss the requirements I based my demonstration program on. This chapter is not going to go into the requirement details in its entirety, but rather focus on the key requirements, and some discussions related to coming up with these requirements. It can be considered a discussion and documentation on what the program should do (as opposed to how to do it) from a user's perspective.

The basic requirement of the program is to demonstrate that NorNet Core consists of a number of nodes, geographically separated, each with several ISP connections, that are able to communicate with each through these ISP connections using both IPv4 and IPv6.

Since NorNet Core is all about communication between nodes, it is important that the demonstration scenario is able to do some form of communication. This communication between nodes will be referred to as *experiments*. What kind of communication this should be will be discussed in Section 2.1

Additionally, to enable the user to easily interpret the results of the experiments, the results needs to be presented in a suitable format. This topic will be covered Section 2.3.

The purpose of the software is to show the functionalities of NorNet Core in an illustrative way. I want to do this in a way that makes it clear what NorNet Core is about for people with little or no IT background. The main goal is making a framework that can be used in the context of presenting the NorNet Core testbed to an audience. It should also be a live demonstration, connecting to nodes and doing the experiments in real time. Table 2.1 lists some of the basic requirements.

2.1 Demonstration Scenarios

My initial thoughts on demonstration scenarios was scripted events. However, a more useful and flexible approach would be giving the user the power to interact with the NorNet nodes as he/she wishes.

My idea is based on two different measurement factors:

1. Latency - The Round-trip time (RTT) of packets. The time it takes

packets to travel from one host to another, and then back. This will be measured in milliseconds(ms).

2. Maximum throughput (or Bandwidth) - The maximum number Megabits per second (Mbps) that can be send from one host to another.

Requirement	**Details**
Represent the nodes geographically	Drawing the location of nodes on the map
Interaction with the map	Being able to pan the map, and zoom in and out to a degree that gives the user a good overview
Interaction with nodes	Interacting with nodes to see information about them (ISP connections, etc).
Experiments between nodes	Allow nodes to communicate and show the communication in real time.
Easy deployment of software on the nodes	NorNet Core consists of a great number of nodes, deployment of required software on the nodes should be as automatic as possible.
Persistence of important user data	Persisting data such as sliver information and NorNet Core connection credentials
Presentation of measurement results	Allow the user to view the measurement data in graphs, and allow overlaying of several graphs on top of each other.

Table 2.1: Basic requirements

The approach is that the user can interactively start and stop latency and bandwidth measurements between nodes by interacting with the geographical node-map. I believe these two measurement factors will give users a good idea of what is going on between two nodes. The concepts are also easy to grasp for users, and they could potentially produce interesting results.

The current version of the software realize these measurements by using Internet Control Message Protocol (ICMP) [16] for measuring latency and TCP for measuring the throughput. Additionally, latency and bandwidth experiments can be done by using both IPv4 and IPv6.

By combining measurement data from different nodes, ISP connections, experiment types and IP versions, as well as the geographical location of the nodes, the user will be able to:

- See the effect of heavy network load on latency:
  - Latency the same ISP connection
  - Latency on a different ISP connections

- See how heavy load on one ISP connection affects other ISP connections on the same node.
- Compare performance of different ISPs:
  - Compare bandwidth
  - Compare throughput
- Get an idea of how distance between hosts affects the measurements.

The number of customizable elements give a great number of different demonstration scenarios.

Of course, to be able to easily compare measurement results, the demonstration program offers good support for representing and overlaying result data. This is discussed in Section 2.3

2.2 Representation of Experiments

A running experiment between nodes can be represented by lines on the map. However, it should be possible to do several measurement tests between two nodes, using the same or different addresses.

One step in the right direction is drawing the available local IPs on the map near the location of the nodes (see figure 2.1). Then as long as the source and destination addresses are not identical, the lines will not be overlapping. An example of this can be seen in figure 2.1

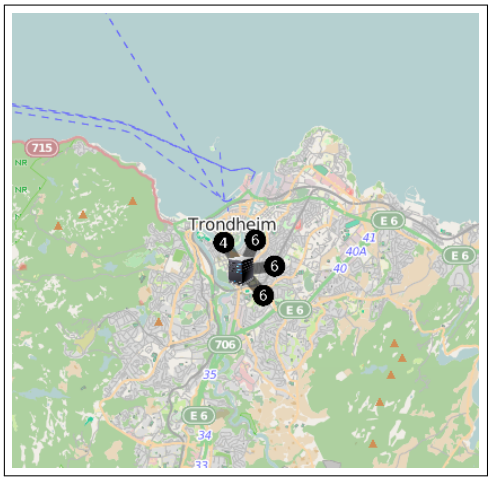


Figure 2.1: Early work - addresses represents IPv4 and IPv6 connections, respectively.

At a later stage of the development process, I realised that the above mentioned way to visualize the available ISP connections for nodes is a bit misleading. IPv4 and IPv6 addresses are tied to ISP connections. Normally, each ISP connection in NorNet Core will offer one IPv4 address, and one IPv6 address.

This relationship between local IP addresses, and indeed the concepts of providers, is important to visualize in the demonstration program, to make it as clear as possible for the users that an IPv4 and IPv6 address pair make up one ISP connection. It is also useful to be aware of ISPs vs addresses when running experiments, as there will most likely be a per ISP connection bottleneck, across both internet protocols.

And this was what I ended up doing. In the demonstration program, the markers clustered around each node marker represents ISP connections, where each ISP connection in most cases has both an IPv4 and IPv6 address associated with it. Interaction with an ISP marker reveals the addresses associated with it, which under normal circumstances should be one IPv4 address and one IPv6 address.

There was still the issue of what to do if the user establishes a second connection between two providers (for example TCP bandwidth test and ping test at the same time). A straight line from provider to provider is not sufficient for visually demonstrating that there are two experiments running between them. I considered the following options:

1. Interaction with the line gives the user information about how many experiments are running, and lets the user choose which connection to view more information about.

2. Visualising several paths between two ISP connections by not drawing a straight line, but rather by drawing a curved or angled line. An example of this is shown in figure 2.2

3. It is also possible to not allow users to start several experiments between two ISP connections.

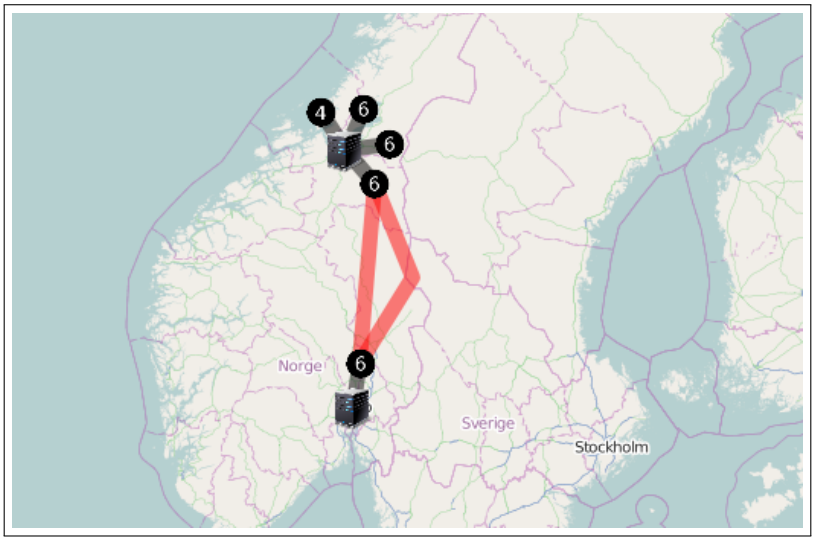


Figure 2.2: Solution if two tests are run in parallell on the same addresses.

The second option of angling the line is decidedly more slick, and it is the option I have implemented for the program.

Despite fixing the multiple experiment problem, there are still possibilities of lines overlapping, making interaction and visual cues from experiments problematic.

To alliviate this, the program could allow the user to interat with the different experiment through a list of running experiment, and from this list the user could get access to the same experiment options as if he/she had interacted with the line on the map. The current implementation includes a list of the last running experiments, but the interaction functionalities have not been added in this version.

2.2.1 Visualisation of incoming data

When there is incoming information about an experiment (for example new ping data, or information about the transfer), there should be some visual cue, relating the information received with the associated connection line. Some possibilities:

1. Showing a pop-up next to the line, with the basic information (for example throughput)
2. Animating the line
3. Changing the color of the line
4. Some other animation from ISP connection marker to ISP connection marker

For the current implementation, the line will get darker for a few milliseconds when new experiment data is received, before returning to it's normal color.

2.2.2 Multi-homing extension

The experiment representation mentioned in Section 2.2 should work well for simple address to address scenarios.

For SCTP or MPTCP however, the mechanisms mentioned are not sufficient. One problem is how we would represent a MPTCP connection on the map. A line between two IP markers is misleading. A possible solution would be a line between IP markers on the same node, and then from the middle of that line, a line going to a remote IP, or to the middle of another line. How about if there are more than 2 local IP's involved in the connection? For example if there's three IP's involved, we could create lines going from all three, meeting somewhere near the node, and then going to the remote address(es).

2.3 Graphical view of the data

Making the measurement data accessible to users was an important requirement of this project. The best way to do it is by drawing the data in graphs. Three possible ways to show the data in graph form:

1. Measurement results on the x axis against time. Measurements received are given a timestamp, and drawn on the timeline. This representation is eseful for getting an over-all, picture of the data received.

2. A frequency distribution table of measurements. - This can be useful for more easily seeing the distribution of the data.

3. A Box plot. It gives an accessible image of the distribution of data.

The current implementation shows the timeline graph, and a box plot representation of the data. A graph based on frequency distribution is not currently implemented.

Chapter 3

Design and Implementation

3.1 The Choice of Development Framework

The software for the actual test scenarios that will be run on the different NorNet Core nodes will be running on the Fedora Core 18 distribution. They require no user interface.

Making lightweight binaries with C or C++ would be easy to deploy, as they would require few dependencies. Higher level languages such as Python or Java would make for a more comfortable development experience, but they would require potentially heavy runtime environment installed on each node, which is impractical.

Things I was looking for when choosing the development language:

- Lightweight installation process on each node, both in terms of installation complexity and file size.
- Access to C libraries
- Efficient coding; plenty of helper libraries.
- Possibility of sharing code between the node program and the demonstration program. This is especially relevant for the network related code, where basic sending, receiving and interpreting messages is relevant for both programs.

3.1.1 Choosing Qt as the development framework

By now it should be no secret that I chose Qt as the development framework for this project. Qt offers myriads of useful functions relevant for this project, although it could be considered bloated for a simple console application.

Unless compiled statically, a Qt executable will have several external Qt library dependencies. Built statically, the size of a very simple executable exceeded 20 megabytes.

Thanks to useful hints online, I was able to reduce the size of the executable to around 5 megabytes by disabling QWebKit and icu libraries at the configuration step of the Qt compilation. The file size for simple

console applications could possibly be further reduced by disabling other libraries, but I was satisfied with a 5 MiB file. The configurations used are listed in Section 6.3

The Qt framework brought a lot of value to the programming project. A few of the relevant features are listed below:

- *What you see is what you get* (WYSIWYG) GUI creation.
- Useful helper classes for:
  - Networking
  - Regex search - For example useful for extracting relevant data from DNS replies.
  - Testing - A library for doing unit testing

It was based on these reasons, and some personal ones (I like to get more familiar with C++) that I chose to use Qt as my development platform.

3.1.2 Alternative choices

Java would be a another cross platform choice, requiring less management of code. Another possibility is running it as a web app, which would give the benefits of cross platform and ease of deploying, as most computers will have web browser with JavaScript support. The nodes would have access to the URL of a RESTful web API which would update a database. Ajax calls on the client devices would then get the latest information, and present it in a suitable way to the user.

3.2 Finding the right map API

Finding the right map API to use with Qt was not as straight forward as I had expected. Clearly, I wanted to avoid having the write the code for displaying and zooming on a map, adding markers, and so on.

The question was, what existing geographical drawing system exists for Qt. I was expecting there to be some kind of Map widget for Qt, either in the native library, or through a third party. After a quick search, it turned out that the Qt 5.2.1 desktop version has no built-in widget to draw geographical maps.

I then did a search on third party widget plugins to do this kind of thing.

A plug in (and standalone application) called Marble [1] looked promising, however, it was developed for Qt 4, and I was not able to get it to compile with Qt 5.2.1. In addition, after testing the standalone Marble application, I discovered that animation and panning was not as slick as I had hoped.

Another standalone application with an API library is QGIS. However, the huge download size of over 180 MiB made me write it off. I then tried

[1] http://qt.digia.com/Qt-in-Use/Marble/

an official but currently unreleased (as of May 2014) map drawing library by Qt, called QtLocation [2].

It was an inconvenience to have to build it from source, and not promising that it was still in beta, but if the performance and ease of use would be good, it might be worth it.

It seemed to work smoothly, but there were flickering issues when maximizing a window, and within a few minutes of drawing the first map on screen, the application crashed. This was tested on only one computer, and could be a computer specific issue.

Turning to JavaScript

What I ended up with, was using Qt's QWebView widget (which allows the user to view html pages in a Qt application), along with the a JavaScript mapping library called Leaflet [3].

I initially started with the Google maps API, as google maps is what I use for my daily map viewing needs, and I naively expected it to be the most stable, flexible and transparent API. However, running through Qt and QWebView, I experienced flickering when panning the map, as well as a general slow-down, compared to when using a browser.

The Leaflet JavaScript library worked much better, giving a smooth experience. Another good thing about Leaflet is that it is Open Source, which in the context of a scripting language means access to original non-obfuscated code, with documentation and comments. This was important for me to be able to freely examine and possibly extend the source, something that turned out to be necessary.

There were some issues with Leaflet discovering the application as running from a mobile browser, and as a result disabling certain features, such as loading the map while panning and keeping tiles in memory. This was easily fixed by editing the Leaflet source.

Leaflet can easily be customized to use different tile servers, however, licensing issues should be taken into account when making the tile server choice. During the development phase, I used a tile server run by OpenStreetMaps [4], however I switched to the MapQuest tile service before release. The MapQuest free Community Edition license offers (currently) unlimited map access [5] for commercial and non-commercial application alike. It requires registration of an application key, which is used with Leaflet. Currently, my personal application key is hard-coded in the `main.js` source. Of course, it can easily be changed if there is a need for it.

3.2.1 Working with Leaflet and QWebView

Using a JavaScript library within a C++ centric Qt project does involve some extra work. However, JavaScript is very well integrated within the

[2] https://qt.gitorious.org/qt/qtlocation

[3] http://leafletjs.com/

[4] OpenStreetMaps offers free world maps. http://www.openstreetmap.org

[5] http://developer.mapquest.com/web/tools/getting-started/terms-overview

Qt environment, due to Qt's own declarative mark-up language named QML[6], which is based on JavaScript. Setting up a JavaScript enabled site with Qt and QWebKit is done exactly the same way as it is done with a regular browser. For my project there is an html file, a JavaScript file containing the interface to Leaflet, and a JavaScript file with the Leaflet library itself. By adding these files as resources, Qt will bake them into the executable.

JavaScript to C++ interface

Accessing C++ classes from JavaScript can be done by first registering a QObject derived C++ class to the QWebFrame by using the QWebFrame method addToJavaScriptWindowObject. Q_INVOKABLE methods(which include both signals and slots) of this JavaScript registered QObject may then be called from within JavaScript in a regular fashion. This functionality is in the Qt Documentation referred to as the *QtWebKit Bridge*[7].

C++ to JavaScript interface

JavaScript can be executed from the QWebFrame instance by using the method QWebFrame::evaluateJavaScript, as shown in the example in the following listing

```
evaluateJavaScript(QString("addStr('%1');").arg(str));
```

3.3 Drawing on the map

Drawing the various markers and lines on the map represented some problems, which I will discuss in this section.

One issue was that the ISP connection of a node needs to be drawn a fixed distance away from the node marker, no matter what the zoom level on the map is. This is necessary to be able to view all ISP connection for nodes clearly.

This was achieved by setting the latitude and longitude position of the ISP connection marker to that of the node marker it belongs to, and then using the *anchor* property of the marker icon in Leaflet to offset it. The notion of anchors is used for several of the GEO mapping APIs, and it's mainly used if the programmer doesn't want the marker to be centered on the geolocation.

However, there was no function in Leaflet for offsetting start or endpoints of a line in pixels from a latitude and longitude position. A small extension to Leaflet had to coded in order to get this functionality. It was also necessary to extend Leaflet to be able to draw lines that do not overlap. The implementation is however crude, and does not move the

[6] http://qt-project.org/doc/qt-4.8/qml-tutorial.html
[7] http://qt-project.org/doc/qt-4.8/qtwebkit-bridge.html

lines correctly while zooming in or out. More time and insight into the Leaflet source will make this a quick fix for future versions.

3.4 Node to Node communication

I decided on using iperf on the backend on each node for doing bandwidth experiments. When the node program is started, it start two sessions of iperf: One for accepting IPv4 connections, and one for accepting IPv6 connections. ping and ping6 are used in the backend for latency testing.

Another option here would be programming custom bandwidth testing programs, but it would require time spent that would be better spend on other areas.

3.5 Node to GUI communication

There are several approaches to the communication between the nodes and the GUI application. NorNet Core users already have access to their slivers through SSH, this interface could potentially also be used for the demonstration system.

Using the shell access for running experiments means the programs would have to be executed through the terminal, and then parse the output of the SSH into useful data. Experiments would just be programs executed locally on the node. The benefits of this method is that I wouldn't have to implement any additional software for communication between the nodes and the GUI application. This makes deployment easier as well. Probably the biggest advantage of this method is the time saved implementing the client and communication protocol for the nodes. This time could instead be spent on polishing the GUI. However, this would mean I would have to rely on text parsing, and this would have to be done through two SSH jumps. In the end I decided to write my own program, which would be deployed and run on each node.

3.6 Application preferences and persistence

Persistence of program data is important to make it easy to use the demonstration program. In this context, I denote preferences to mean options and values that are set before connection is made to nodes; there is no support in my program for persisting any experiment related results.

At the very least the preference data should include the gatekeeper user name, and a list of nodes that should be connect to. All program preferences for the GUI should also be made persistent, so that the user does not have to set the same preferences again and again when exiting and starting up the program.

Persisting information can be achieved by using Qt's QSettings class, which offers a convenient, cross platform way to persist program settings through an associative array style interface.

Arrays of data, such as the information about each sliver, is more easily stored in a separate file that is loaded at runtime and saved when there's new data.

Other things that should be a preference, as well as persistent are:

- Url location of node program to be downloaded to each node
- The gatekeeper host name
- Ports to connect to
- Whether to connect through a gatekeeper at all (if the program is being run on machine in the NorNet Core network)

3.7 Filtering out IP addresses

Not all possible addresses for each node needs to, or should be, shown on the map (for example loopback addresses should not be shown). NorNet Core IPV6 addresses start with 2001:700:4100, and IPv4 starts with 10 [11, Subsection 4.2]. This can be used to filter out unwanted addresses. However, each node will report the full list of IP addresses, and the filtering will be done in the demonstration software. For a future update, it is possible to allow for addresses to be filtered out to be a user option, in case NorNet changes its addressing scheme.

3.8 Extracting the location of sites

The NorNet Core system has included the geographical location of sites and slivers on the DNS servers' *LOC* [3][8, Section F] Resource Record (RR). This information can be can be extracted by doing a DNS lookup on the site name or sliver name. This has to be done from within the NorNet Core network. An example using the Linux tool dig follows:

```
dig loc srl-nndemo.bymarka.ntnu.nornet +noall +answer
```

The above command will show only the *LOC* RR of the DNS query response, to a query for the *LOC* RR of host name *srl-nndemo.bymarka.ntnu.nornet.* The output is as follows:

```
srl-nndemo.bymarka.ntnu.nornet.  86400 IN CNAME
srl-nndemo.bymarka.uninett.ntnu.nornet.
srl-nndemo.bymarka.uninett.ntnu.nornet.  86400
IN LOC 63 25 4.800 N 10 24 5.760 E 50.00m 5m 20m 20m
```

From this output, the geographical numbers of interest are:

1. 63 25 4.8 N
2. 10 24 5.760 E

These numbers can easily be extracted with the help of a Regular Expression search.

3.9 Deployment

One of the requirement goals from the start was being able to automatically deploy and run the necessary software on the different nodes.

This needs to be done using the provided shell access to the slivers. The easiest way to do this is by running the program ssh as a new process from the demonstration software. There is however an issue of the possible password(s) that should be entered. This is further discussed in Section 3.9.1

3.9.1 Multi-hop SSH

To get access to the slivers through SSH, an RSA private key is required, where the public key is uploaded and activated for the slivers. This private key may or may not be password protected. But this access may only be granted if the connection is made from within the *gatekeeper* server. Access to this server is also done through SSH, and this access may also be password protected. To sum up, to get access to a sliver through SSH, we first SSH into the gatekeeper, and from there SSH to the sliver we want access to.

My first approach to automate this multi-hop SSH connection involved parsing the output from the process running the SSH instance, and sending the password(s) back to it as standard input at the appropriate time(s).

Then to make things easier, and more robust, I decided to put a restriction on the user on how his/her SSH access information is kept; I make the assumption that no password needs to be entered during the operation of the demonstration program.

The user can arrange for this by caching the required passwords in an *authentication agent*. This is a program that keeps the private keys with the associated passwords cached, and through agent forwarding shares this cache with the remote SSH instance, so that no password has to be entered upon connection. On Linux-based systems, adding a key to the authentication agent can be done by using the command *ssh-add*, which will normally add key to the *ssh-agent* authentication agent. There is a program for windows called *Pageant*[8] that can be used for the same purpose.

SSH Agent forwarding

Using the agent forwarding functionality also allows me to keep all the necessary user authentication information central. Keys along with the passwords can be loaded into the SSH agent on the computer running the demonstration software, and the agent can be forwarded through to the gatekeeper, and then again to the sliver. For the Linux *ssh* client, this is done by adding an -A option as an *ssh* argument. It is the same option for Windows and *PLink*.

[8] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

3.9.2 Remotely downloading files

Once SSH access is established, deployment of the node software can commence. There are several ways this can be done. Below are three options I considered:

- Using Secure Copy (SCP) - SCP allows file transfer through the SSH protocol. This should be initiated from the demo application.
- wget - Using the Linux tool *wget* to download the executable - or package - from a web server.
- yum install - Installing the program through yum, which is the package manager for Fedora. This requires the software to be properly packaged and accessible through a repository server.

SCP would be the easiest solution, but when doing a multi-hop SSH, it does not work out of the box. It can be made working by forwarding ports on the gatekeeper, or adding entries in .ssh/config. But this would have to be done for every sliver we would want to connect with. SCP was therefore discarded.

The two other solutions are based on making the software available for download on a third instance server. With the ssh program for Linux (and Plink for windows), it is possible to specify one command that should be executed after the SSH connection is established. The command will then be executed once the connection is established, and when the program has finished running, the SSH connection will close.

This method is used by the current implementation to set up each node.

The command wget to fetch the install script, and then from within the script wget to download the node program(denoted nodeprog, and yum install to download the dependencies of the node program. Details about the script are described in Section 3.9.3.

3.9.3 Deployment Script

A simple bash script has been written to deal with deployment on the nodes. It does the following tasks:

1. It checks if node program (nodeprog) is already running. If it is, the rest of the script is not executed.
2. If nodeprog is not running, it checks whether nodeprog exists, and if it does, it compares the nodeprog version(by running *./nodeprog -v* with the version supplied as argument to the script. If the version checks out, the script executes nodeprog. If the version is not compatible, the script will attempt to download the newest version.
3. If nodeprog was not found, it downloads and installs the dependencies of nodeprog by using yum install. These are the packages bind-utils (for dig), iperf and ping. Lastly, it downloads nodeprog itself using wget, from the url given as input argument to the script.

The install script does three important jobs:

1. It makes sure only one instance is running at the same time,
2. It makes sure a node is always up to date, by re-downloading nodeprog when necessary
3. It makes sure all dependencies are installed

The demonstration program will automatically deploy and executed the script on each node by using SSH to get access to the sliver. However, this will only be done if it is necessary; if there is already an IPv6 address associated with a node(from previous connections), the demonstration application will first attempt to do a normal TCP connection, in the hopes that nodeprog is already installed and running.(Since fetching the IP is a part of the deployment process). If there is no successful connection after a few seconds, the demonstration program will start the deployment on the node.

The correct arguments will be supplied to the script based on user preference(the URL of the node program (nodeprog) and install script) and hard-coded data (the current version of the program)

3.10 Discussion on building

When using a shared library build of Qt, the compiled executable will have many Qt library dependencies. Unless the correct version of Qt is already installed on the target system, Qt either needs to be installed or, the necessary libraries needs to be copied along with the executable. Figuring out which libraries are needed can be done with the help of the *ldd* utility for Linux based systems, or Dependency Walker[9] for Windows.

To simplify the deployment of node program(nodeprog), I have relied on building it statically, and then ending up with a self contained executable. The standard Qt install only allows for linking with shared libraries; to be able to statically compile the Qt libraries into the executable, Qt has to be statically built from source. Static compilation requires the software to be licensed under LGPL, which is not an issue for this pen source project.

Static compilation would also be convenient for the demonstration application, but it would likely result in an bloated executable (based on forum rumours, in the excess 100 of MiB), which would be excessively big for a single executable, especially considering that the necessary Qt libraries may already be installed on the target computer.

The other problem with using static Qt libs with WebKit enabled is the compilation process: I made a few attempts, but every time the compilation would ultimately end with a build error. This could of course be sorted out by further trials and studying.

[9] http://www.dependencywalker.com/

Chapter 4

The Demonstration Program

In this chapter I will present various screen caps from the demonstration program, designed and implemented as part of this thesis, and explain possible user interactions. Lastly, I will present some possible demonstration scenarios.

4.1 Settings things up

Before the user can make any connections, the proper settings must be set. The settings window is opened by clicking on the settings icon in the toolbar(figure 4.4).

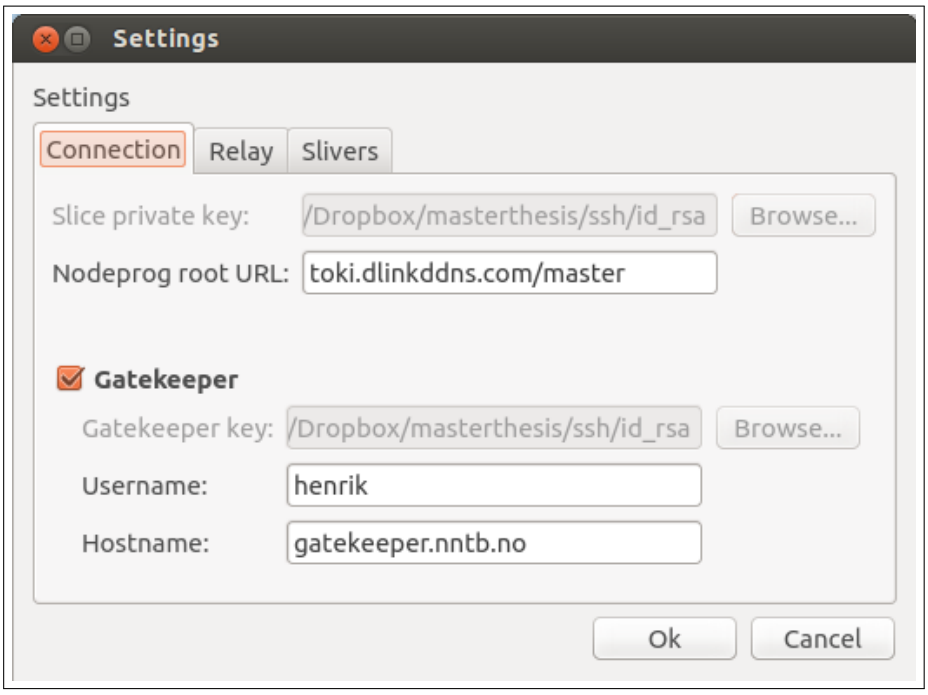


Figure 4.1: Gatekeeper settings

Under the *Connection* tab, figure 4.1, the user needs to specify from which URL the program that should be installed on each node (as well

as the install script) can node can be found. This is done in the *Nodeprog root URL* field.

Currently, to get access to the NorNet Core nodes, the user needs to connect through a gatekeeper server. In this case, both the user name and the host name fields needs to be specified. If the *gatekeeper* check-box is not checked, the program will try to connect directly to the nodes, which will most likely fail, as it's not possible to get SSH access to nodes externally to NorNet Core network.

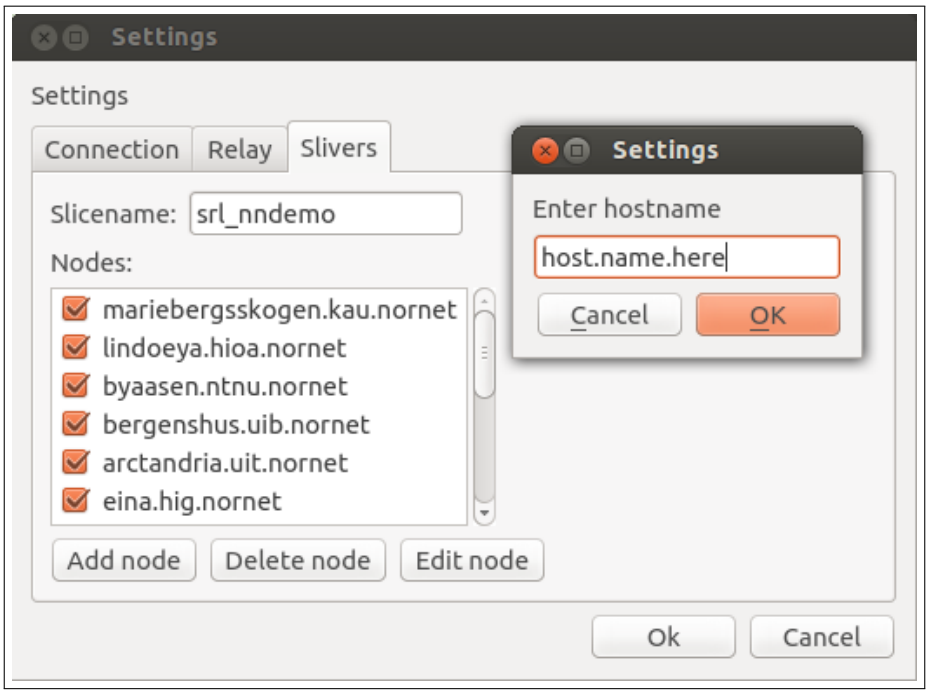


Figure 4.2: Sliver settings

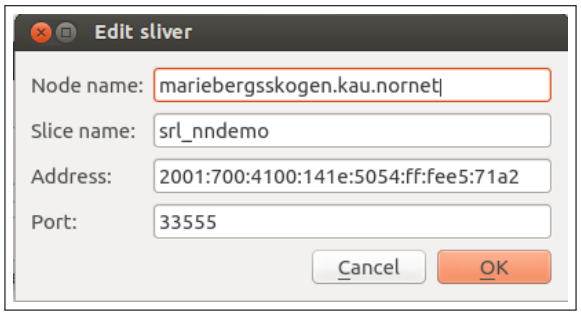


Figure 4.3: Editing sliver connection information

In the Slivers tab, figure 4.2, users has to add the slice name they wish to connect with, as well as all the host names of the nodes that they want the program to connect to. Normally, only the host name will be sufficient information, however. It will select the default port to connect to, and automatically fetch the IP using SSH, as described in Section 5.2. In the event that the IP address is not correctly fetched, the user can enter the

IP address to connect to manually. This is done by selecting a node, and clicking *Edit node* (see figure 4.3).

4.2 Making the connection

After the settings are properly configured, the user can click on the connect icon from the toolbar to start connecting to the nodes (figure 4.4). This may take some time, especially if a node is not already set up with the `nodeprog`(see Section 5) program. A node marker should show up on the map in anywhere from 1 to 20 seconds. If it takes longer than that, there is most likely a problem with setting up a node. This could be related to *yum* not being properly installed on the sliver, missing DNS entries, missing IPv4 addresses, or because the address used for the connection is not valid. A log of what went wrong (and right) can be found in the text file `nodeprog.out` on the node in question. That is assuming the demonstration program was able to get SSH access.

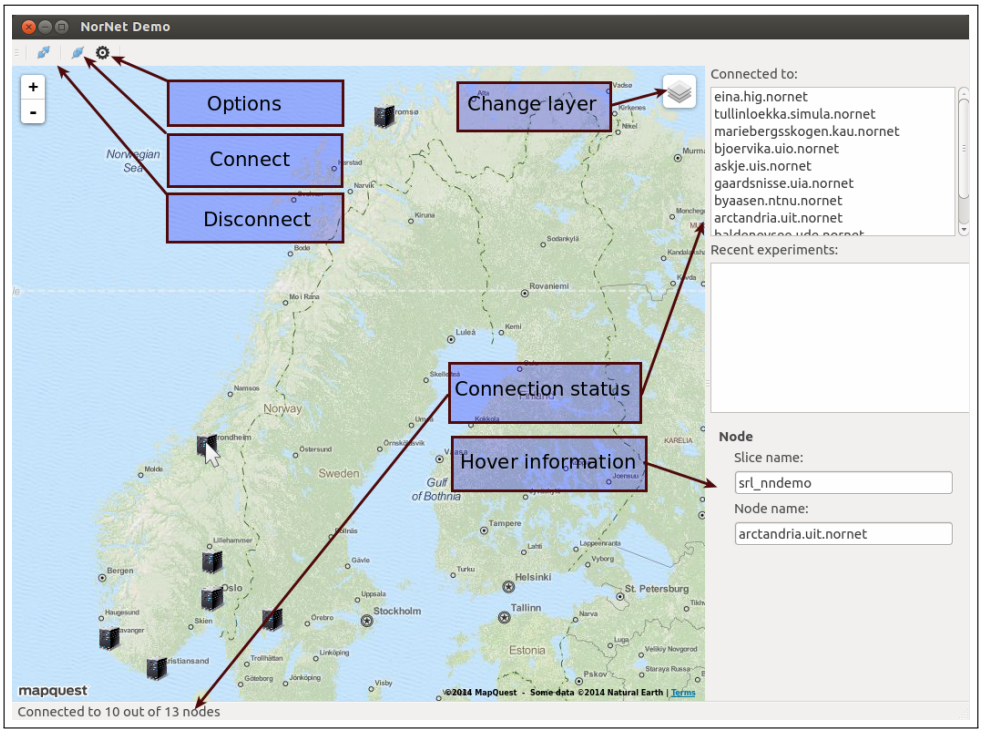


Figure 4.4: Map with GUI annotations

As connection with nodes are established, the marker appear on that map, and the host name of the node is added to the *Connected to* list. Currently the list has no further functionalities beyond giving the user information about the open connections. A nice feature to add in the future would be letting the map automatically pan to the node selected in the list.

4.3 Interacting with nodes

After the connection is successful the user can start interacting with the nodes. Clicking on a node will reveal all of its ISP connections, and hovering over nodes and its ISP connection will show information about these entities. For nodes, this information includes:

- Name of the slice
- Name of the host

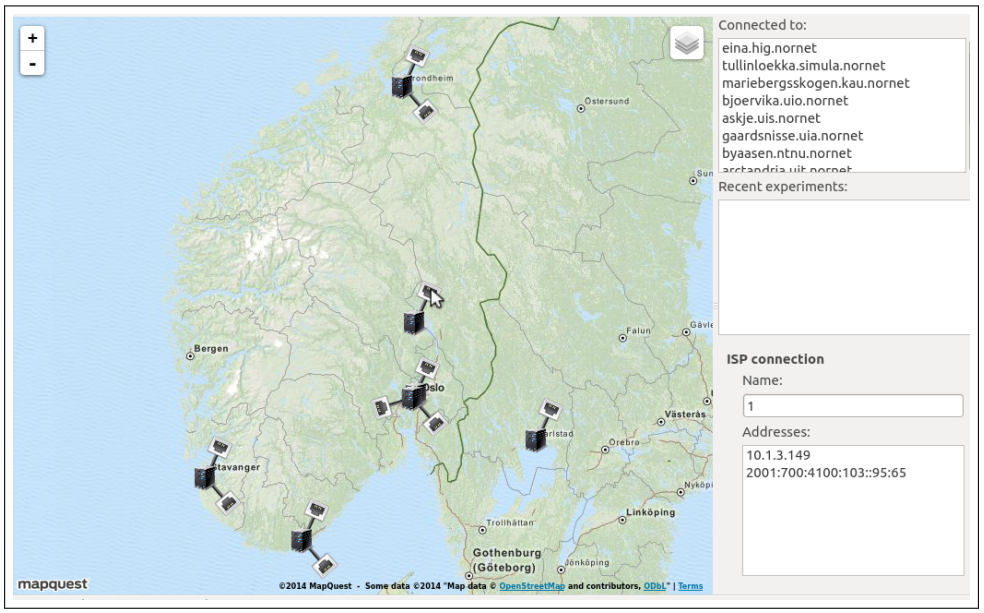


Figure 4.5: ISP connections drawn for nodes

For an ISP connection, hovering reveals:

- The index of this provider.
- All of the available local addresses tied to this connection.

Figure 4.5 shows an example of hovering over an ISP connection, as well as the ISP provider representation on the map.

The reason ISP connections are not all drawn by default is to avoid cluttering the screen with too many lines. This is especially problematic if the user zooms out too much.

Currently, there is no mentioning of the ISP names of the different connections, only the provider index. This is a feature that could be added in the future.

4.4 Starting an experiment

An experiment is started left clicking on an ISP connection, and then consecutively clicking on any other ISP connection. The experiment will

be running from the first one selected and communicate with the second one selected. When two ISP connections have been consecutively clicked, the program prompts the user for what kind of experiment to start. This prompt is shown in figure 4.6. Here the user can choose between a Ping test and a TCP bandwidth test. Both of these experiments can be done over IPv4 and IPv6, given that both IPv4 and IPv6 are available for both ISP connections chosen. The user should also set how long the experiment should run. The experiments can however be stopped at any time.

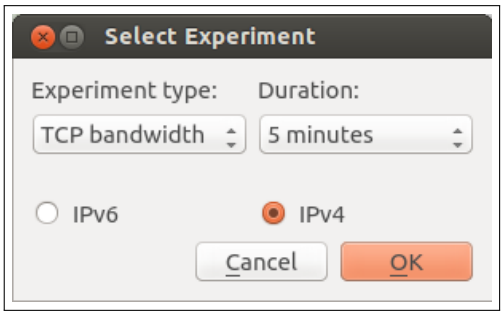


Figure 4.6: Experiment selection

A line will be established between the provider markers involved. The line is interactable, and will flash distinctly for each new incoming measurement related to this experiment. This makes it easy to see whether an experiment is active or not.

In most cases, if an experiment is not successful, due to node related problems, the node will report this back, and the connection line will disappear. There is however a chance of a line not getting any status message, but still not disappearing. This could happen if the connection the node is trying to make is not responding. The user can easily recognize this by checking if the line is flashing or not. If there's no incoming measurements data, but the measurement is still active, a manual removal of the line can be done, by right clicking on the line and selecting *remove*.

4.5 Experiment interaction and graphs

Experiment lines can be interacted with. This can be done in three ways.

1. By Hovering - This shows basic information about the connection, such as the experiment type, internet protocol used, and the source and destination IP addresses.

2. Left clicking - Left clicking will open a new window, and start drawing a graph of the incoming data. For a TCP connection, the incoming data is the bandwidth, in Mbps. For a latency experiment, the data is RTT, in milliseconds.

3. Right clicking - Right clicking gives the user the option of either removing the experiment, or viewing the measurements data in existing window, or viewing it in a new one (same as left clicking).

Figure figure 4.7 shows hovering interaction with an active experiment. At the right hand side of the figure, the related experiment information can be viewed. The *Recent experiments* list has currently no other value other than listing the recent experiments. In a future update, it should be made interactable, to enable the user to view past experiment data.

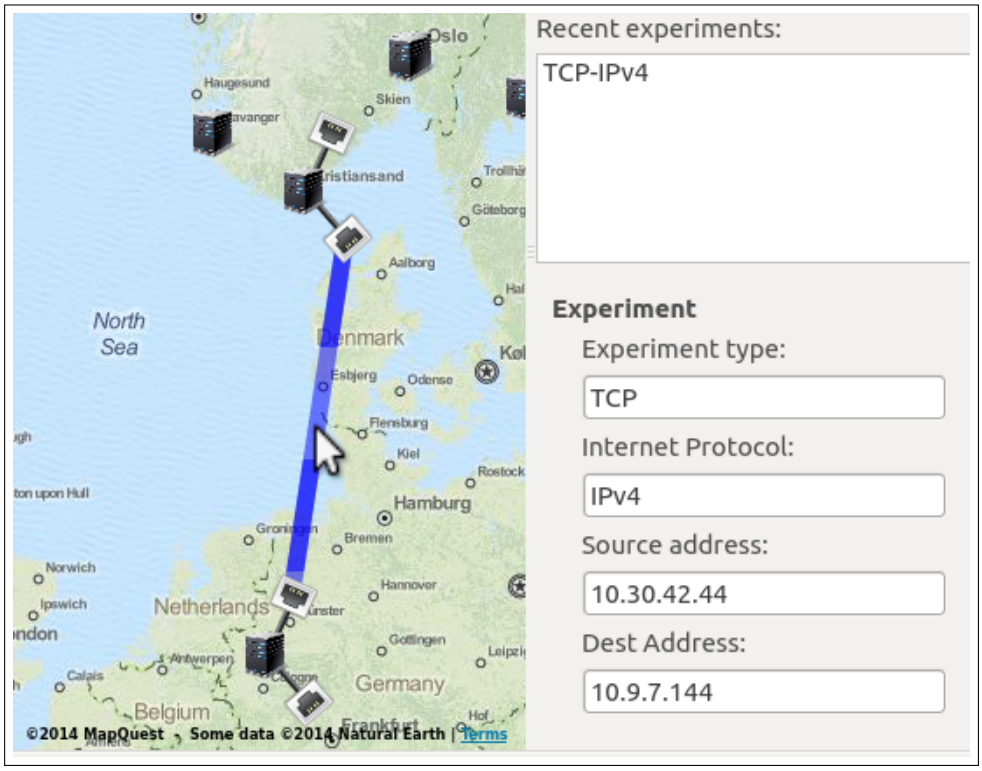


Figure 4.7: A line representing an active experiment

After left clicking on the experiment line, a new plot window will open for this experiment. Figure 4.8 shows the latency of an IPv6 packet travelling from Norway to China and back. Clicking on an experiment from the list in the plot window shows the same information about the experiment as seen in the map window.

As new data is added, the graphs are automatically updates, and the x and y axis are scaled to fit all the measurements. This may be unwanted in some cases when studying the graph; clicking on the *Auto resize* button in the corner will disable this feature. The graph can be zoomed and panned respectively by using the mouse wheel, and dragging in the graph area.

If the user at a later time wants to add other measurements to this graph, it can be done by right clicking on an experiment line, selecting *Add to graph*, and choosing the relevant graph. Graph windows can be renamed to make things more organized when several windows are involved.

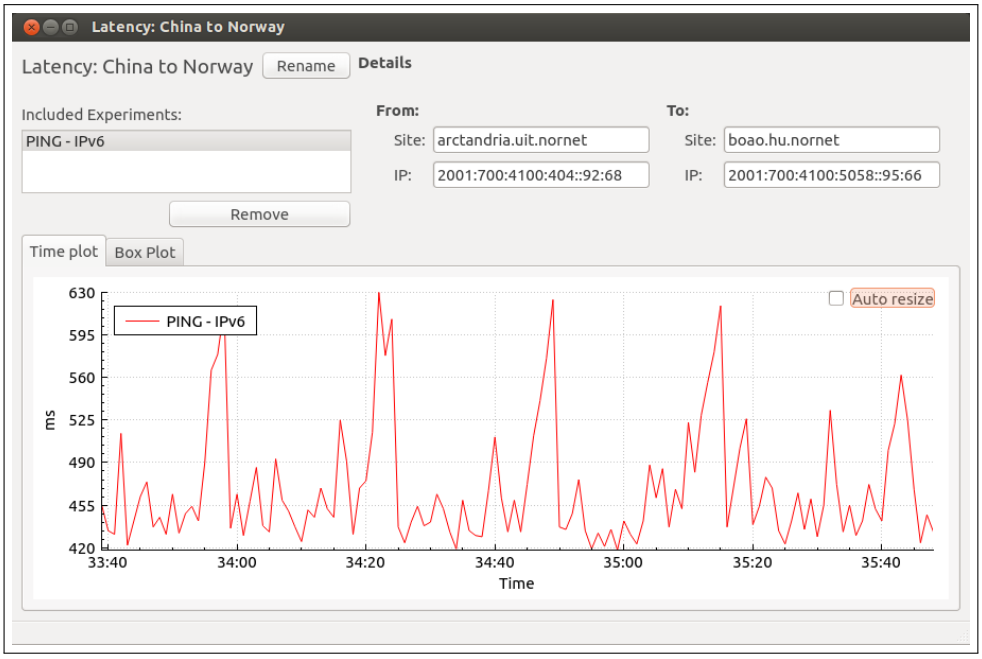


Figure 4.8: IPv6 ping results from Norway to China

4.5.1 A multi-homing scenario

Figure 4.9 show 3 throughput graphs overlaid in the same plot in a multi-homing scenario between *byaasen.ntnu.nornet* and *arctandria.uit.nornet*.

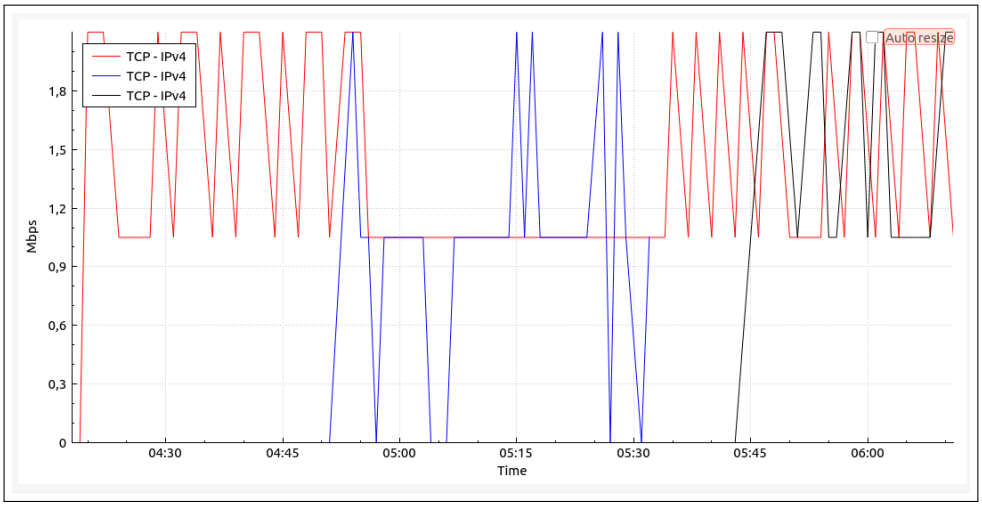


Figure 4.9: A multi-homing scenario

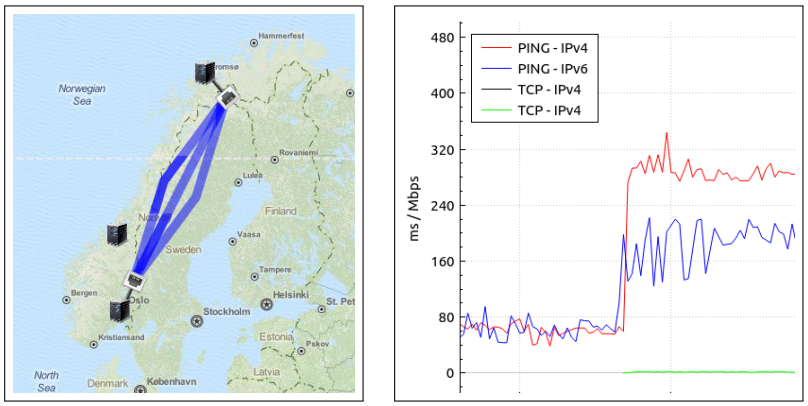
The graph itself does not look too pretty; it is most likely because of the low bandwidth, and because iperf is doing some rounding of the numbers. However, it still shows some clear results: In this plot, the red and blue coloured graphs are TCP bandwidth tests on the same ISP provider pair, and the black connection is using different ISP providers. When the red connection is running by itself, it reaches up to 2 Mbps. When the blue

one is added, the throughput of the red connection is reduced by half, and the total bandwidth is more or less the same. The blue connection is then stopped, and the black connection is started, which is a connection between the same nodes, but using different ISP providers. Judging by the graph, the black connection, using a different ISP providers, has no effect on the red one. This is a typical case where load sharing would definitely be of use to increase the rather slow 2 Mbps connections, and get a total throughput of up to 4 Mbps.

4.5.2 Heavy Network Load and Round Trip Time(RTT)

Figure 4.10b shows the effect heavy network load can have on the RTT of packets. This experiment is done between the nodes *tullinloekka.simula.no* and *kongsbakken.uit.nornet*, and using only one ISP connection on each node, as shown in figure 4.10a. The red and blue lines represents the RTT of IPv4 and IPv6 respectively.

At the beginning of the scenario, the RTT of both IPv4 and IPv6 appears to be more or less similar. However, after a TCP bandwidth test between the same two ISP connections is started, both RTT times are dramatically increased. Interestingly enough, the IPv6 RTT is significantly shorter than that of IPv4.



(a) Geographical view (b) Graph view

Figure 4.10: Heavy load and RTT

Chapter 5

The Demonstration System

This chapter presents implementation details of the NorNet Core Demonstration Framework. First, it is necessary to introduce some terminology for the different parts of the system.

The demonstration program itself, that gives the user the GUI and map overview, will be referred to as `nornetdemo`. It's main GUI class is `DemoGui`. `DemoGui` uses an instance of a class called `DemoCore` to do the actual communicating with the nodes. The software that is going to be running on the nodes, will be referred to as `nodeprog`. `nordnetdemo` and `nodeprog` are the only executables in the system.

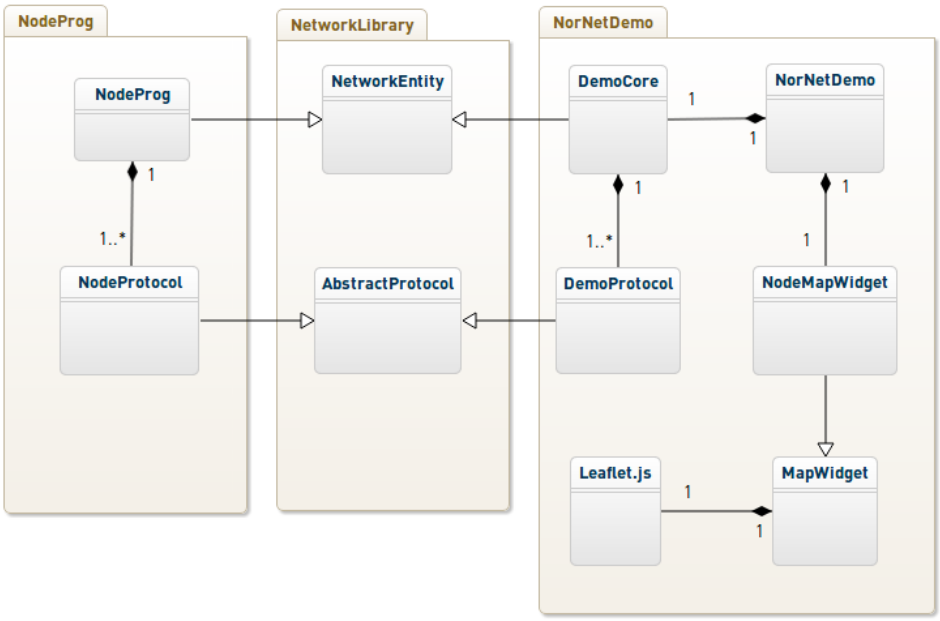


Figure 5.1: Class diagram of the Demonstration System

Both `DemoCore` and `NodeProg` are derived from the class `NetworkEntity` (figure 5.1), which provides functionalities for listening on TCP ports and making TCP connections. When a new connection is established, it will start what I have called the *handshake protocol*. After the handshake is

accepted, each side hands the TCP socket over to another protocol, which deals with the meat of the communication. There are currently 4 protocols. These are organized as the classes:

- NodeProtocol
- DemoProtocol
- HandShakeProtocol
- RelayProtocol.

They are all derived from the AbstractProtocol class.

The AbstractProtocol class provides functionality for serializing a message and attaching a header to it by using the sendMessage method. It will also accept new data through the newData slot. The newData slot reads the fixed sized header. The header contains information about the size of the message, and the type of message that will be received. If there are enough bytes available to be read, the newData slot will call the virtual method handleMessage. handleMessage is the only method that needs to be implemented by derived classes of AbstractProtocol.

It is in the handleMessage method that the message object is created from the datastream, and the content of the message is examined, reacted to, and in most cases responded to. The reactions are usually in the form of *Tasks*. The three currently implemented tasks are:

- PingTask - Starting a new ping or ping6 instance
- TransferTask - Starting a transfer test with another node
- NodeInfoTask - Collecting information about the node's IP addresses and extracting the node location from the DNS server.

Each network tasks are derived from AbstractTask, which offers functions for stopping a task. Each derived class needs to implement this stopping method.

5.1 Messages

All the messages are derived from the class AbstractMessage, and they are required to implement the functions serialize, read and getType. The method serialize is used to generate a byte representation of the message object. This is used by AbstractProtocol::sendMessage when it is converting the message into a byte array.

5.2 Connecting to nodes

When the user clicks on the connect button, the list of slivers is fetched from the SliceManager class. The list is then passed on to an instance of theDemoCore class through the method DemoCore::connectToSlivers.

DemoCore will go through the list of slivers, and do its best to establish a connection with each one. If the IP address is not known, it has to be extracted from the node first. This is done by first establishing an SSH connection and then executing the following command:

```
ip addr show eth0 | grep 'inet6 2001:700' | awk 'NR==1'
```

ip addr show outputs information about IP addresses on interface *eth0*, grep filters out invalid addresses, and awk limits the result to only 1 line. This could most likely be done in a more robust way, and the method makes the following assumption.

1. The output format of ip addr show will not change in the future, and will be the same on every node in the NorNet Core network.
2. The valid IP addresses will always be found on interface *eth0*
3. The address prefix of the NorNet nodes does not change from 2001:700:4100

Clearly, it is not the most robust way to extract an IPv6 address, and thus it will be improved in a future version.

Chapter 6

Setting up the system

6.1 Compiling the software

To be able to compile the program suite successfully, Qt 5.2.0 or later is required. The sources can be downloaded from http://github.com/henrikvs/nornetdemo.

The easiest way to build all 3 programs (nornetdemo, nodeprog and relayprog) is to complete the following 3 steps from the root directory of the project:

1. *qmake* - Generates the makefile.
2. *make* - Builds the programs
3. *make install* - Deploys the programs into the *bins* folder in the root directory of the project.

Additionally, to avoid cluttering up the source folders with the compiled files, the following method can be applied:

1. *mkdir build && cd build* - Create an empty directory to build into, and cd into it.
2. *qmake ..* - Run qmake from within this directory.
3. *make install* - Same as above

To compile the individual programs a "CONFIG+=<program>" argument can be added to the qmake step, as shown in the example below:

```
qmake "CONFIG += nodeprog demogui"
```

The above command will prepare a Makefile for compiling nodeprog and demogui.

6.2 Preparing the system

The executable nodeprog and the script install.sh found in the bins/nodeprog folder after compilation, need to be made available on a web server in the same directory. The URL of this directory has to be entered in the demonstration application, in the *nodeprog root URL* field of the settings window (see figure 4.1). Of course, other information needs to be entered as well, such as the gatekeeper user name, the host name of the gatekeeper, the slice name and which nodes the program should connect to. Before clicking on the connect button, there are two things to consider:

- That all the keys(usually two) needed for the connection are loaded into the SSH authentication agent. The OS may or may not prompt for passwords if this is not done in advance.
- That IPv6 is available on your computer - The NorNet nodes will only accept IPv6 connections from outside the NorNet Core network. If IPv6 is not available from the ISP where the demonstration application will be running from, software tunnelling solutions can be used to obtain an IPv6 connection. An easy option is to install the Linux program miredo. It worked well during the development of this program.

6.3 Compiling Qt statically

The process I used to compile Qt 5.2.1 statically:

```
git clone https://git.gitorious.org/qt/qt5.git qt5
cd qt5
git checkout 5.2.1
./init-repositoy --no-webkit
./configure -opensource -release -nomake examples -
   nomake tests -no-icu -prefix=<directory-of-choice>
./make -j4
./make install
```

It's important to also make sure the dependencies of the build process are installed. Further detailed building instructions and building dependencies should be read on the Qt website[1]

6.4 Tools used

A quick mentioning of the various tools used.

- QCustomPlot [2] - Used for plotting graphs.

[1] http://qt-project.org/wiki/Building_Qt_5_from_Git
[2] http://www.qcustomplot.com/

- Qt [3] - The development framework
- Leaflet [4] - A javascript map drawing API.
- MapQuest [5] - A free to use map tile server
- ping - Used by nodes to ping other nodes on though IPv4.
- ping6 - Used by nodes to ping other nodes on though IPv6.
- iperf - Used to run IPv6 and IPv4 throughput tests.

All the tools mentioned are used within the license agreements

[3] http://qt-project.org/
[4] http://leafletjs.com/
[5] http://www.mapquest.com/

Part III

Conclusion

Chapter 7

Summary

An efficient demonstration of NorNet Core and its capabilities was designed and developed, clearly demonstrating NorNet Core's most important aspects. This was done by drawing the NorNet Core nodes on a geographical map, including each site's available ISP connections as separate markers connected to the node marker. This makes it very clear that NorNet Core supports multi-homing. The program allows for communication between individual ISP connections, with several experiments between pairs of ISP connections.The experiments currently supported are latency and bandwidth tests, both working over IPv6 and IPv4.

A great deal of work was put into allowing the user to be able to comfortably compare measurement results of the different experiments. This was realized by using graphs, and allowing users to overlay graphs from different measurements. Finally, the demonstration system was put to the test on various networking scenarios.

Chapter 8

Future work

There are always improvements and features that can be added to software, such as bug fixes, missing features, etc. Addtionally, the NorNet Core testbed itself is still in the start-up phase, and at it keeps growing and changing, the demonstration program may need to be patched to accomedate for these changes.

Below is a list of some of useful features that could be added for a future versions of the program.

- Sorting out representation of several nodes drawn on the same location. Each site has several nodes, with identical geographical location. The current demonstration program will draw all the nodes on top of each other
- Drawing tunnelling hops.
- Importing list of sites from the PLC
- Showing names of the ISP providers for each ISP connection.
- Other experiment types:
  - MPTCP
  - UDP
  - CMT-SCTP
- Improve graphics
- SSH terminal access integrated in the program
- Installation status of nodes
- Different kinds of graphs
- Exporting results
- Separating Mbps and Ping to separate axis's

Bibliography

[1] I. 7498-1. '"Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model"'. In: 2000.

[2] P. Amer, M. Becke, T. Dreibholz, N. Ekiz, J. Iyengar, P. Natarajan, R. Stewart and M. Tuexen. *Load Sharing for the Stream Control Transmission Protocol (SCTP)*. Internet Draft draft-tuexen-tsvwg-sctp-multipath-08. IETF, Individual Submission, 19th Mar. 2014. URL: https://tools.ietf.org/id/draft-tuexen-tsvwg-sctp-multipath-08.txt.

[3] C. Davis, P. Vixie, T. Goodwin and I. Dickinson. *A Means for Expressing Location Information in the Domain Name System*. RFC 1876 (Experimental). Internet Engineering Task Force, Jan. 1996. URL: http://www.ietf.org/rfc/rfc1876.txt.

[4] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946. Internet Engineering Task Force, Dec. 1998. URL: http://www.ietf.org/rfc/rfc2460.txt.

[5] T. Dreibholz. 'The NorNet Core Testbed - An Experiment Tutorial'. In: *Proceedings of the 1st International NorNet Users Workshop (NNUW-1)*. Fornebu, Akershus/Norway, 19th Sept. 2013. URL: https://simula.no/publications/Simula.simula.2130/simula_pdf_file.

[6] T. Dreibholz. *The NorNet Testbed for Multi-Homed Systems – Introduction and Status*. Invited Talk at Princeton University, Department of Computer Science. Princeton, New Jersey/U.S.A., 8th May 2014. URL: https://www.simula.no/publications/Simula.simula.2730/simula_pdf_file.

[7] T. Dreibholz, M. Becke, H. Adhari and E. P. Rathgeb. 'Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain'. In: *Proceedings of the 19th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. ISBN 978-953-290-027-9. Hvar/Croatia, 16th Sept. 2011, pp. 1–6. ISBN: 978-953-290-027-9. URL: https://www.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/SoftCOM2011.pdf.

[8] T. Dreibholz and E. G. Gran. 'Design and Implementation of the NorNet Core Research Testbed for Multi-Homed Systems'. In: *Proceedings of the 3nd International Workshop on Protocols and Applications with Multi-Homing Support (PAMS)*. ISBN 978-0-7695-4952-1. Barcelona, Catalonia/Spain, 27th Mar. 2013, pp. 1094–1100.

ISBN: 978-0-7695-4952-1. DOI: 10.1109/WAINA.2013.71. URL: https://simula.no/publications/threfereedinproceedingsreference.2012-12-20.7643198512/simula_pdf_file.

[9] D. Farinacci, T. Li, S. Hanks, D. Meyer and P. Traina. *Generic Routing Encapsulation (GRE)*. RFC 2784 (Proposed Standard). Updated by RFC 2890. Internet Engineering Task Force, Mar. 2000. URL: http://www.ietf.org/rfc/rfc2784.txt.

[10] A. Ford, C. Raiciu, M. Handley and O. Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824 (Experimental). Internet Engineering Task Force, Jan. 2013. URL: http://www.ietf.org/rfc/rfc6824.txt.

[11] E. G. Gran, T. Dreibholz and A. Kvalbein. 'NorNet Core – A Multi-Homed Research Testbed'. In: *Computer Networks, Special Issue on Future Internet Testbeds* 61 (14th Mar. 2014). ISSN 1389-1286, pp. 75–87. ISSN: 1389-1286. DOI: 10.1016/j.bjp.2013.12.035. URL: https://simula.no/publications/Simula.simula.2236/simula_pdf_file.

[12] A. Kvalbein, D. Baltrūnas, K. R. Evensen, J. Xiang, A. Elmokashfi and S. Ferlin-Oliveira. 'The NorNet Edge Platform for Mobile Broadband Measurements'. In: *Computer Networks, Special Issue on Future Internet Testbeds* 61 (14th Mar. 2014). ISSN 1389-1286, pp. 88–101. ISSN: 1389-1286. DOI: 10.1016/j.bjp.2013.12.036. URL: https://simula.no/publications/Simula.simula.2434/simula_pdf_file.

[13] M. Becke, T. Dreibholz, University of Duisburg-Essen, J. Iyengar, Franklin and Marshall College, P. Natarajan, Cisco Systems, M. Tuexen and Muenster Univ. of Applied Sciences. *Load Sharing for the Stream Control Transmission Protocol (SCTP)*. July 2010. URL: http://tools.ietf.org/html/draft-ietf-tsvwg-sctpsocket-15 (visited on 27/11/2013).

[14] P. Mockapetris. *Domain names - concepts and facilities*. RFC 1034 (INTERNET STANDARD). Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936. Internet Engineering Task Force, Nov. 1987. URL: http://www.ietf.org/rfc/rfc1034.txt.

[15] L. Peterson and T. Roscoe. 'The Design Principles of PlanetLab'. In: *Operating Systems Review* 40.1 (Jan. 2006). ISSN 0163-5980, pp. 11–16. ISSN: 0163-5980. DOI: 10.1145/1113361.1113367. URL: https://www.planet-lab.org/files/pdn/PDN-04-021/pdn-04-021.pdf.

[16] J. Postel. *Internet Control Message Protocol*. RFC 792 (INTERNET STANDARD). Updated by RFCs 950, 4884, 6633, 6918. Internet Engineering Task Force, Sept. 1981. URL: http://www.ietf.org/rfc/rfc792.txt.

[17] R. Rivest, A. Shamir and L. Adleman. 'A Method for Obtaining Digital Signatures and Public-Key Cryptosystems'. In: *Communications of the ACM* 21 (1978), pp. 120–126.

[18] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960. ISSN 2070-1721. IETF, Sept. 2007. URL: https://www.rfc-editor.org/rfc/rfc4960.txt.

[19] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251 (Proposed Standard). Internet Engineering Task Force, Jan. 2006. URL: http://www.ietf.org/rfc/rfc4251.txt.