MultiPath TCP: Hands-On

Author

Gerrie Veerman gerrie.veerman@os3.nl

Supervisor

Ronald van der Pol rvdp@sara.nl

Universiteit van Amsterdam
System & Network Engineering

July 9, 2012

Abstract

This project's aim, is to get more experience with Multipath TCP within SARA. Multipath TCP makes it possible to use multiple paths at the same time. MPTCP wants to become the TCP 2.0 and came recently available for the 2.6 linux kernel. In this project we go over the theory behind MPTCP and its design goals and workings. We talk about how we configured it and how we were able to use it. We created a layer two test topology with LAN and WAN capabilities, both with 1- and 10Gb/s links. We did experiments regarding the following topics: performance, load balancing, robustness and congestion, this within our topology. A couple of scripts have been created to get the desired environment for our experiments.

As a result we can say that MPTCP works well overall and can meet two of its goals concerning: *improve throughput* and *balance congestion*. We are however a bit curious about the *do no harm goal*, in our results MPTCP was a bit unfair compared to TCP. We also noticed that MPTCP behaved strange on link changes. When new connections joined, the old link was not used anymore. Further MPTCP behaves most optimal in a stable environment, especially when the links have the same capacity and have a small RTT difference.

Contents

Preface

I would like to thank my supervisor Ronald van der Pol and Freek Dijkstra from SARA for their great help and support during this project. Without their effort I would not have been able to create the topology and get the project results. I appreciate their critical questions and reviewing of this document.

I also would like to thank SARA for the opportunity to do this project with them and Gerben van Malenstein of SURFnet for providing the wide area links towards Geneve and Chicago.

Besides this I thank Christoph Paasch from the Universite Catholique de Louvain for his help and information regarding the configurations of the topology setup.

1 Introduction

The demand for more bandwidth is constantly increasing, not only large organizations but also home users are using more bandwidth. The AMS-IX gets more traffic throughput every month with a current peak of 1.6 Tb/s [1]. Organizations are increasing their fiber infrastructure dramatically [2]. Also the wireless world is expanding and increasing its infrastructure [3]. To cope with all those bandwidth demands and increasing infrastructure the Internet is in need for new future technologies. One of those promising techniques is called MultiPath TCP (MPTCP), this is a new protocol developed in the IETF, which was partly funded by the Trilogy project of the European Commission [4].

The purpose of MPTCP is the capability to use multiple paths in a network simultaneously. This is obviously not always possible, since not every end-node has multiple network interfaces available. However, some environments usually have multiple paths for failover or robustness. A good example is a datacenter, which currently uses those different paths in a static load balancing manner. The problem with load balancing different flows is that they do not fully utilize the actual maximum bandwidth possible. This load balancing is usually done through hashes of the connections, based on those hashes flows get created. Flows only take the same path, thus when one has a lot of flows one can get great throughput. However e-science applications always have a limited amount of flows and do not fully utilize the different links available. MPTCP should make it possible to fully utilize all available paths and increase bandwidth and robustness. Some call MPTCP already the Transmission Control Protocol (TCP) 2.0. Figure 1 shows how MPTCP can be used.

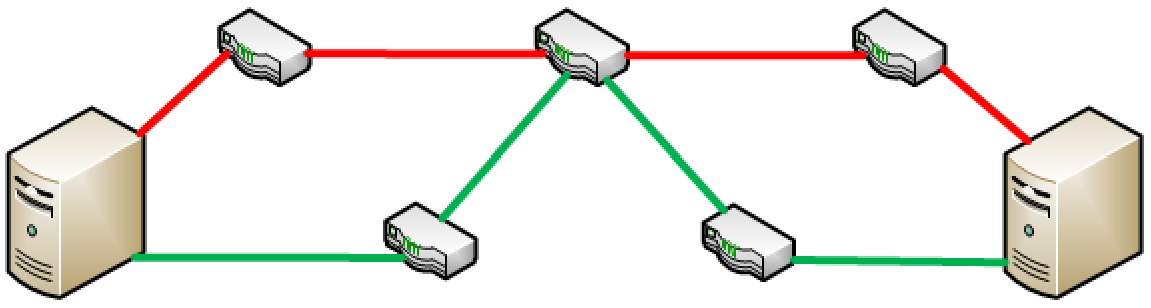


Figure 1: Topology where Multipath TCP is being used

Another promising example is the mobile world. Phones hove multiple interfaces for example 3G and Wifi. With regular TCP it is only possible to use one of those for a connection. The concept of multi-homing in combination with MPTCP makes it possible to use both the 3G and Wifi link at the same time. This has of course its pros and cons but at least MPTCP can make this possible. One of the pros is for example the easy migration of your connection from 3G to Wifi, your phone can use the same connection and does not have to establish a new one. In figure 2 is shown how this looks like from a mobile phone point of view.

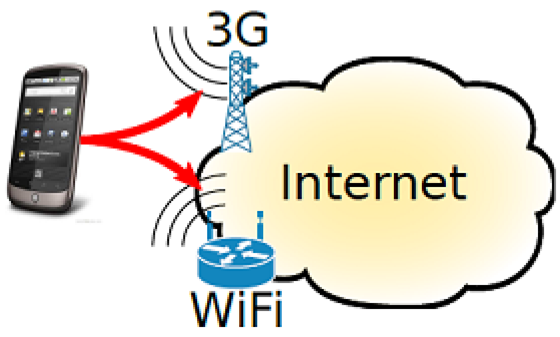


Figure 2: Mobile phone with Multi-Homing [5]

This project aims to be the first hands-on practice for SARA with MPTCP [6]. We want to get more experience with the current MPTCP implementation and see how it operates and performs.

1.1 Related Work, Motivation & Goal

The Internet Engineering Task Force (IETF) has a working group, which is currently working on standardizing MPTCP in different RFCs [7]. The IETF currently has three finalized RFCs 6181, 6182 and 6356. They also have four RFCs in draft, which get updated frequently. One of the parties developing MPTCP is the University College London (UCL) department of computer science is doing a lot of research in MPTCP [8]. They have released some papers and specifications in the field of MPTCP, most of them are about congestion control and fairness with MPTCP. The IP Networking Lab (INL) from the Universite Catholique de Louvain (UCL) created an MPTCP implementation for the Linux Kernel and this is the main test environment for users, testers and developers [9]. A protocol which has some comparable features to MPTCP is Stream Control Transmission Protocol (SCTP), which is explained in section 2.2.1.

We are doing this research for SARA which is currently setting up a large WAN environment in the Global Lambda Integrated Facility (GLIF) network. They want to perform MPTCP and Openflow experiments in this topology. Since MPTCP is relatively new, SARA needs some hands-on experience.

1.2 Research Question

We formulated the following research question for this project:

> *Is the current* MPTCP *implementation a useful technology for e-science data transfers in the GLIF environment*

We identified the following subquestions, which help to answer the main research question:

- How does MPTCP handle failures and bandwidth changes?
- How is the fairness being handled? (congestion)
- Can MPTCP be configured and used easily in layer two networks?
- How is MPTCP working with large RTTs and buffers (LAN vs WAN)
- *Optional:* How well does MPTCP perform compared to GridFTP?

We also note some additional questions that can come up during the project:

- Does MPTCP work when all interfaces are in the same VLAN? (routing)
- Does MPTCP work when both IPv4 and IPv6 are being used at the same time?
- Can MPTCP work with different bandwidths speeds correctly?

1.3 Scope, Time & Approach

We want to look if the main goals MPTCP 'should' provide are really met. Also check which kind of features it has and how well MPTCP performs. We also want to look into two additional subjects: One is about the Wide Area Network (WAN) performance of MPTCP and if it works well with a large RTT, will this give any problems compared to a Local Area Network (LAN) environment? The second is how GridFTP performs compared to MPTCP.

This project has a predetermined fixed duration of four weeks. Due to this small time frame we can't do that much and do not have the time to test/analyze everything extensively. We will first get ourself familiar with the MPTCP protocol and everything that comes with it. After that we want to get some hands-on experience by creating a LAN environment and run the defined tests. Than we move to the WAN environment and can test the same there and in combination.

1.4 Report Structure

This report covers the theory, the research and the outcome of this project. The report is structured as follows:

- *Chapter 1:* **Introduction** – the current chapter, introducing the project, its goals and the research question
- *Chapter 2:* **Theoretical Definitions and MPTCP explained** – providing the required theoretical definitions for this research and explaining the MPTCP protocol
- *Chapter 3:* **Theoretical Research** – examining the MPTCP protocol, describing our topology and running test experiments
- *Chapter 4:* **Results Discussion** – discussing the results of the experiments and compare it with MPTCP its design goals
- *Chapter 5:* **Conclusion** – the research conclusions and project outcome
- **Appendices**

2 Theoretical Definitions

This chapter will give the definitions of the different terms and protocols used in this project, this to get a basic understanding of what this project is about.

2.1 Definitions and Terms

This section covers the general terms that come up in this document.

2.1.1 Congestion Control

Network congestion occurs when too much data is being carried over a physical link and packets are being dropped. When this happens the total amount of bandwidth that can be carried by the link gets less. There are some techniques which make it possible to make efficiently use of all the bandwidth available. Those techniques are implemented in the transport layer of the OSI stack and are referred to as 'congestion control' mechanisms. In figure 3 is shown how congestion control works when a link has two connections running over it. One can see that both links together almost fully utilize the link.

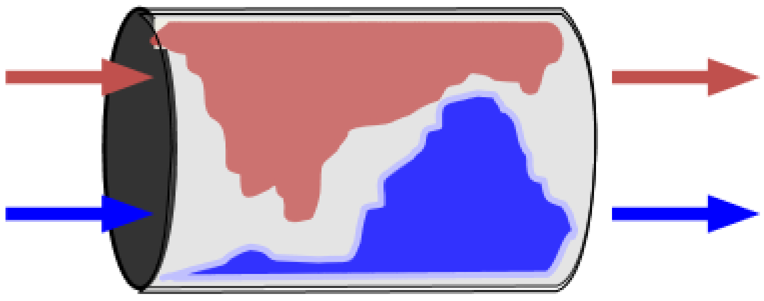


Figure 3: Purpose of congestion control on a link [10]

Congestion control is implemented in the TCP protocol, this since TCP has to guarantee that data gets delivered, compared to User Datagram Protocol (UDP) which does not. Some of the congestion control algorithms used within TCP are Tahoe and New Reno, there are also specific algorithms designed for long connections which have high RTTs. A congestion control algorithm is also being used by MPTCP, the algorithm is almost like the one from TCP but modified to meet MPTCP its goals.

2.1.2 Bandwidth Fairness

The idea of bandwidth fairness is that one connection should not consume more bandwidth than any other connection. If one has for example three connections, which all want to use the full bandwidth it should be shared, so all have them should get around 33% of the link.

Sometimes some kind of traffic is preferred, this is called Quality of Service (QoS). With QoS one can prefer different streams of traffic over another, this way one is less fair to other traffic. However, MPTCP should be fair with its bandwidth usage to other traffic when no QoS is implemented.

2.1.3 Multi-Homing

Multi-Homing is a term that gets used when a device has multiple interfaces that can be used. This for instance, can be a phone as shown in figure 2, but also a laptop or tablet can have multiple interface. In recent years it is becoming more common for end-devices to have multiple connections for example with Wifi and 3G. MPTCP should make it possible to use all available links from those multi-homed devices at the same time with one MPTCP connection.

2.1.4 Round Trip Times

The Round-Trip Time (RTT) is the time it takes for a data packet to be acknowledged. For instance, when a ping is being send the RTT basically means the time it takes to go back and forth over a link. This can usually indicate how close the other device is and how fast one can connect to it. The RTT is used by MPTCP in its congestion algorithm.

2.1.5 MTU and MSS

The Maximum Transmission Unit (MTU) is the size in bytes of what the largest protocol data unit can be. The MTU is the size in which packets get fragmented and send over the link. The usual MTU for Ethernet is 1500, but with higher speeds this can be increased till 9000. This is called a jumbo frame. With a large MTU the CPU has less interrupts on the receiver side, this since it can send larger packets and has less overhead this way. In this project we use jumbo frames when we use 10Gb/s links, this to get the best performance in some of our experiments.

The Maximum Segment Size (MSS) is a parameter used in TCP protocols. It specifies how large a packet can be. Therefor, the headers + MSS should be smaller or equal to the MTU. The default MSS for MPTCP is 1400 but can be modified, which we did to get better performance on high speed links, we increased it to 8900.

2.1.6 Global Lambda Integrated Facility

GLIF is an virtual organization that connects international networks together. This GLIF is used by researchers and students for educational purposes and research. The network consists of optical connections between major cities/universities around the world. A GLIF loop to Chicago and Geneve is being used for this project. Further will SARA be using the GLIF infrastructure with its test-bed topology.

2.2 Explanation of Protocols mentioned

This section covers the protocols that gets talked about in this document.

2.2.1 SCTP

The SCTP protocol uses multiple streams over the same physical link, this in the way that it creates multiple sub-connections over the same link. Those streams can have different priorities, so one can have for example a control and data plane stream. The reason that SCTP is not a real success is due to legacy devices that do not support the protocol. NATs and firewalls don't recognize the SCTP protocol and see it as 'malicious or unknown' traffic. Currently the IETF is working on an extension which should make it possible to use SCTP over multiple paths. MPTCP however, should work on every device that supports TCP. Application also don't have to be changed to let them make use of MPTCP, which is not the case with SCTP.

2.2.2 VLAN

A Virtual LAN (VLAN) is a group of computers which act as being on the same LAN. However, physically they can be connected to different switches in different places. VLANs can also be used to create different LANs while on the same switch. We use VLANs in this project to create different paths, which are needed to make the MPTCP connection possible.

2.2.3 ECMP

Equal-Cost Multi-Path (ECMP) is a protocol that uses next-hop packet forwarding to a single destination, which can be over multiple "best paths". It is used to balance traffic over different paths available, which have the same "Cost". The routing is done based on a hash of a users connection (IP address and port). A routing decision can be made with this hash value. Specific hashes only go through a specific port which creates a "flow", every device that uses ECMP needs to make this decision. Besides using VLANs one can make use of ECMP for their MPTCP routes.

2.2.4 GridFTP

GridFTP is a variant of the popular File Transfer Protocol (FTP) protocol. GridFTP is used by Grid computing applications to transfer large chunks of data fast and reliable. GridFTP has the capability to create multiple sockets over different interfaces, which makes it able to use multiple links just as MPTCP. GridFTP however creates multiple TCP connection streams, while with MPTCP one would have a single MPTCP connection. This project may test if this can help or is better than using GridFTP.

2.3 Multipath TCP Explained

The idea of using multiple paths for TCP came out some time ago. It was first suggested by Christian Huitema in 1995 [8]. It took some time before the idea turned into MPTCP. Around 2006 the first documentation appeared with the term MPTCP, later in 2011 Request For Comments (RFC)s were written for it. In 2011 a paper appeared about the first implementation in the 2.6 Linux kernel [11]. They started working on it in 2009. Mark Handley is one of the members who is looking at the MPTCP protocol.

MPTCP makes it possible to use multiple paths at the same time for the same connection. Since MPTCP is a layer four protocol, it is able to use both IPv4 and IPv6 at the same time. MPTCP does not setup paths itself but depends on routing protocols like IS-IS, OSPF and BGP. It is of course also possible to use other protocols like ECMP or Openflow for this. With the use of MPTCP one 'should' get more robustness and the best congestion control possible when multiple links are being used.

2.3.1 How does MPTCP work?

The MPTCP protocol is actually an extension (option) of the TCP protocol. When both the client and server use the MPTCP capable option it will be used. Since MPTCP only uses options in the TCP protocol all applications are able to use the regular TCP socket Application Programming Interface (API) 'as is'. The API is however extended and MPTCP aware applications can make use of those extensions. Another feature is that MPTCP does not get intercepted by middle-boxes like firewalls, since it just looks like TCP for them. This is one of the main advantages compared to SCTP.

When the server does not support MPTCP the protocol switches over to regular TCP. When more connections are possible MPTCP creates so called TCP subflows and every subflow can be seen as a connection from interface to interface. MPTCP tries to create subflows between all the interfaces it can find and creates a full-mesh. If for example one has two IPs on the same interface it tries to connect them both as a subflow. A high level overview of how MPTCP works is shown in figure 4.

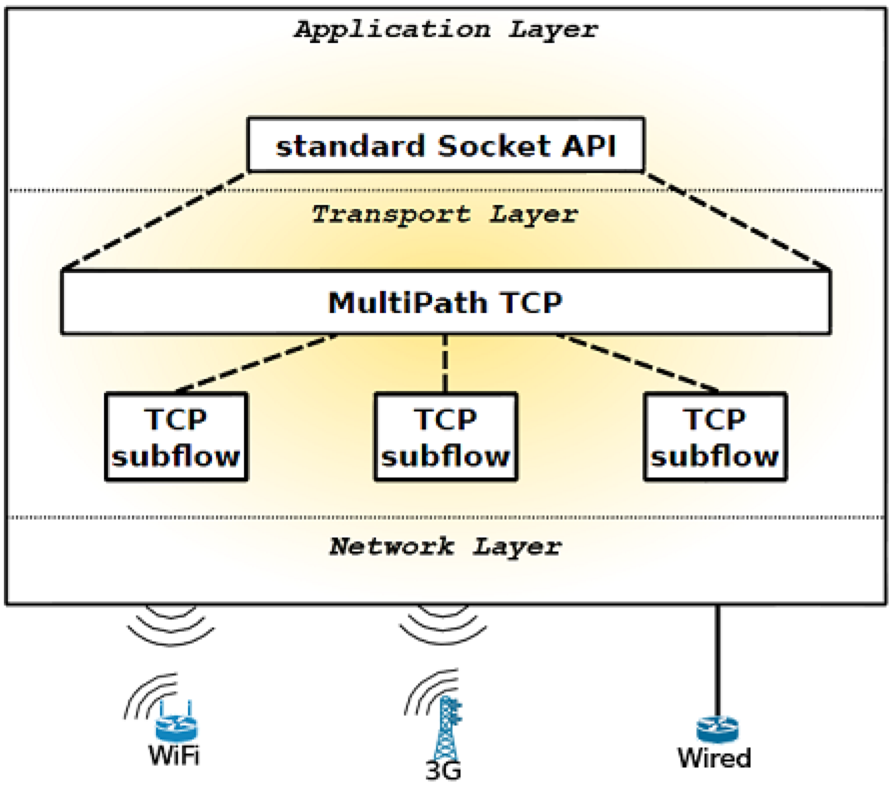


Figure 4: Architectural diagram of MPTCP [5]

2.3.2 MPTCP design Goals

While developing MPTCP, some important goals regarding the protocol were created. Below the three goals in mind for MPTCP are given, they are from RFC 6356 [12]:

1. Improve Throughput: A multipath flow should perform at least as well as a single path flow would on the best of the paths available to it.
2. Do no harm: multipath flow should not take up more capacity from any of the resources shared by its different paths than if it were a single flow using only one of these paths. This guarantees it will not unduly harm other flows.
3. Balance congestion: A multipath flow should move as much traffic as possible off its most congested paths, subject to meeting the first two goals.

All three goals make absolute sense and are very interesting when one combines them. The first rule should make sure, that always the best performance is achieved or the same as TCP would get on its fastest link. The second rule can be seen as the congestion and fairness part. When one has a TCP and MPTCP connection over the same link they should both get 50% of the bandwidth, this even when MPTCP has several subflows over the link. The last rule says that one should always move its traffic to the less congested link, this to get better throughput for all the links overall. It is non-trivial to create an algorithm that follows all three goal. Section 2.3.5 dives into more detail about this.

2.3.3 MPTCP Handshake

This section will cover how the MPTCP handshake works and looks like, the handshake is shown in figure 5 and is written in a Internet draft [13]. First ‘Server A’ sends a SYN packet to ‘Server B’, which is the same as a general TCP connection would do. This packet however also contains a MP_CAPABLE option in the TCP option field. Than ‘Server B’ responds with a SYN/ACK packet also with this MP_CAPABLE option (only when it can use MPTCP). Both packets also contain authentication keys and additional information needed by MPTCP. At last ‘Server A’ responds with a ACK to ‘Server B’ which also contains both keys from the previous packets. Now the first connection (subflow) is setup and the same as TCP is a three way handshake.

Now the initial setup is done and the protocol tries to create additional subflows. Since the implementation is in both servers the same, they can both create additional subflows, however it is recommended the senders only do this. The creation of a subflow is also a SYN, SYN/ACK and ACK handshake but now an additional MP_JOIN option is added. ‘Server A’ also sends with the first SYN ‘Server B’s token’, which is generated with the keys exchanged earlier. This is used to identify and authenticate the first subflow connection. Also a nonce is sent to prevent replay attacks. The address ID sent is to specify the address and make the subflow unique from the others. Because of this address ID the connection is independent from the source address. The flags can be used for more advanced options (for example use it only as back-up link).

When the subflow MP_JOIN SYN is verified by ‘Server B’ it responds with an SYN/ACK. This ACK contains a Message Authentication Code (MAC), a random nonce and the address ID for ‘Server B’. The MAC is used for the authentication of a subflow. After this ‘Server A’ responds with a ACK packet and the MP_JOIN option also with the additional data as with ‘Server B’. After this ‘Server B’ needs to verify this packet with another ACK, since ‘Server A’ has to be sure ‘Server B’ received the packet. For all the authentication technique’s SHA-1 is being used.

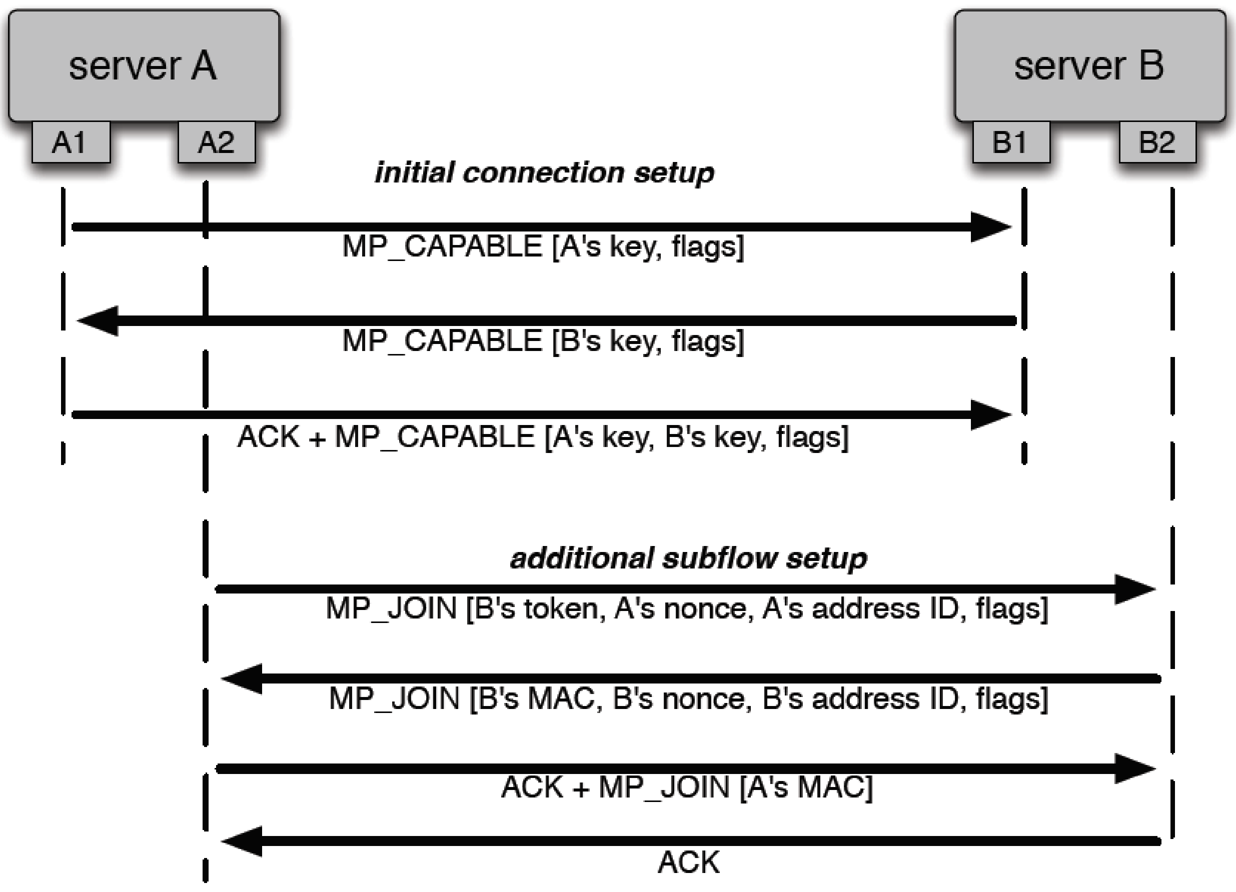

server A
A1
A2
server B
B1
B2
initial connection setup
MP_CAPABLE [A's key, flags]
MP_CAPABLE [B's key, flags]
ACK + MP_CAPABLE [A's key, B's key, flags]
additional subflow setup
MP_JOIN [B's token, A's nonce, A's address ID, flags]
MP_JOIN [B's MAC, B's nonce, B's address ID, flags]
ACK + MP_JOIN [A's MAC]
ACK

Figure 5: MPTCP handshake [14]

2.3.4 MPTCP Path Management

With the handshake explained above in section 2.3.3 the first initial subflow gets created. After this MPTCP makes sure both hosts exchange all IP addresses they have with each other. The sending host tries to create a full-mesh out of all combinations possible. The connections which do not work are being dropped, after a time-out. When you have a interface with both a IPv4 and IPv6 address, MPTCP creates two subflows for that inferface, one with IPv4 the other with IPv6.

MPTCP exchanges IP address with packets who have the subtype *add-address*. After this a *join* subtype is send to create the actual new subflow with those new known addresses. Another nice feature MPTCP has is that it is capable of removing subflows with the subtype *remove-address*. When a host discovers one of its links is broken it can very fast send over a working link that that particular link does not work anymore. This way the other end does not have to wait for time-outs on the broken link and can just drop that subflow. Most of the subtype options are shown in section 3.2.4 in detail.

2.3.5 MPTCP Congestion Algorithm

All reliable transport protocol needs a congestion algorithm. Because MPTCP uses multiple links it is different than TCP and uses a modified algorithm. The idea for MPTCP is to have congestion over multiple links as can be seen in figure 6.

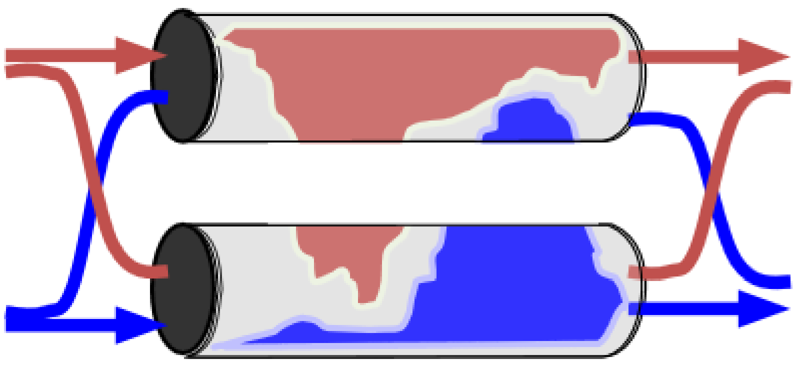


Figure 6: How MPTCP should handle congestion [10]

The congestion algorithm for MPTCP should meet the goals described in section 2.3.2. In chapter three in RFC 6356 is written how the algorithm is constructed and all the different goals are being met [12]. The algorithm is not very different than the TCP congestion algorithm. TCP maintains its congestion window by increasing the window with each ACK (1/cwnd) and dropping it to half when a ACK is dropped. For MPTCP this is no different, however now a window is kept for each subflow. The only thing changed is the increase rule for this window which is shown in equation 1.

The increase rule is based on alpha, which one can see as a constant for now and is explained below. It increases the window by alpha, over the sum of the window sizes of all paths. This gives more increase to the subflows with a larger window, remember that a larger window size means less congestion. The sum of all windows part should meet goal two.

$$cwnd_i = cwnd_i + min\left(\frac{alpha}{cwnd_{tot}}, \frac{1}{cwnd_i}\right) \tag{1}$$

Alpha is the increase parameter shown in equation 2. The first part looks at the congestion window and RTT on all different paths and checks if it is getting better or worse than normal TCP, which meets goal one. The second part of alpha moves traffic from a congested link to a uncongested link and meets goal three.

$$alpha = cwnd_{tot}\frac{max_i(\frac{cwnd_i * mss_i^2}{RTT_i^2})}{(\sum_i \frac{cwnd_i * mss_i}{RTT_i})^2} \tag{2}$$

The decrease formula is almost the same as for TCP but is now used per subflow and is shown in equation 3.

$$\frac{cwnd_i}{2} \tag{3}$$

The algorithm should make sure that MPTCP takes the best path available. This is shown in figure 7, MPTCP should take the shortest and least congested path(s). Choosing the wrong paths can make MPTCP less efficient, it can for example take longer routes or use links which are congested. When MPTCP takes the most optimal paths it would get the best throughput.

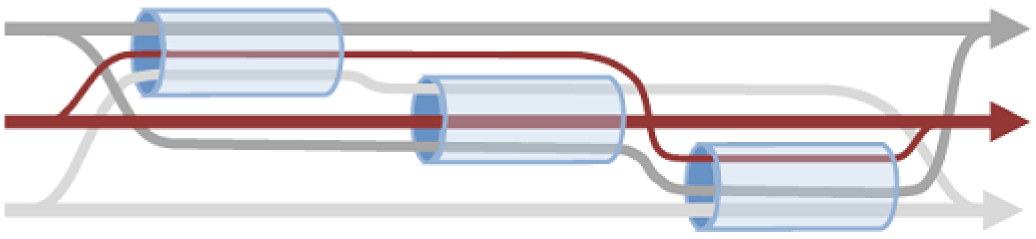


Figure 7: MPTCP path decision taking [11]

2.3.6 MPTCP and buffer sizes

The buffer size needed for MPTCP is a little different than with TCP. In equation 4 is shown how the buffer size for TCP is being calculated. Which is the maximum RTT in seconds multiplied by the maximum the link can carry in bits.

$$Buffersize_{TCP} = RTT_{max} * LinkMax_{bits} \tag{4}$$

For MPTCP, this is a little bit different. Here you need to specify the total maximum bits possible of all links combined. They also note that: "If we want to allow all paths to keep sending while any path is fast retransmitting, the buffer must be doubled" [15]. This ends up in equation 5, shown below. One can notice that you significantly need more buffer space than with TCP was the case. One needs to combine the maximum bits of all links together and multiply that by two plus the RTT.

$$Buffersize_{MPTCP} = RTT_{max} * AllLinksMax_{bits} * 2 \tag{5}$$

If one has for example a RTT of 36ms and two 1Gb/s links, one would get the following formula:

$0.036 * 2000000000 * 2 = 144000000bit = 18\text{MB}$

2.3.7 More about MPTCP

The concept and idea of MPTCP is very interesting. The current kernel version has parameters which one can change to modify the MPTCP implementation, more over this in section 3.2.2. In chapter 3 we look deeper into the technical side of MPTCP. For more information there are two presentations on youtube, which go over the protocol briefly and clear [16][17].We will see in the near future if the protocol will be adopted in the standard protocol stack and if we can really call it the TCP 2.0.

3 Research

This chapter will cover the research part of this project. We will describe how our topology looks like and what we configured for our environment. At the end we describe and give the results of our defined experiments.

3.1 What our Topology looks like

We created a topology with three servers, two switches and multiple 1Gb/s or 10Gb/s links. We have both LAN and WAN links available in our topology. We got two WAN loops to Geneve and one to Chicago. There was no routing in our network. All paths were configured by using various VLANs. Our MPTCP servers have multiple VLANs per interface available. In figure 8 our topology is shown. The dark-blue blocks are our servers. With a combination of all those links we can come up with interesting test setups and see if MPTCP really works well in a real environment. Our experiments can be found in section 3.3.

We have two servers ‘bliksem’ and ‘donder’, which are configured with a MPTCP kernel. They have both a management interface on their eth0, which is being used for as the control plane. For our data plane we have three other 10Gb/s interfaces on both servers. Each interface as either 2 or 3 VLANs attached to it. The server ‘bliksem’ is quite new and has a lot more RAM and CPU than the ‘donder’ server. This caused some problems which are defined later in our report. The technical specifications of those servers can be found in appendix B.

We have another server ‘rainbow’, which is used to generate normal TCP traffic and will be used to cause congestion on a link. Rainbow also has the ability to power cycle ‘donder’ and ‘bliksem’, which was handy since they ‘froze’ a few times. Further our topology contains a switch with 2x 1Gb/s local loops, with this switch we could create a link bottleneck from 10Gb/s to 1Gb/s. We needed this since all our other links were 10Gb/s.

3.2 Setup of the environment

We first got ourself familiar with MPTCP and looked at its theory on how it should function. We came up with some basic ideas and questions, which are defined in section 1.2. Below we define what we configured, experienced, show our topology, formulate our experiments and describe our scripts created.

3.2.1 Configuration of our servers

We had to build the kernel for two of our Ubuntu 11.10 servers to be able to talk MPTCP to each other. We used the guide on the INL website to build the 3.2.0 linux kernel with MPTCP (*http://inl.info.ucl.ac.be/mptcp/* [9]). We had to build the kernels a couple of times to load modules for VLANs and different network cards to be able to work.

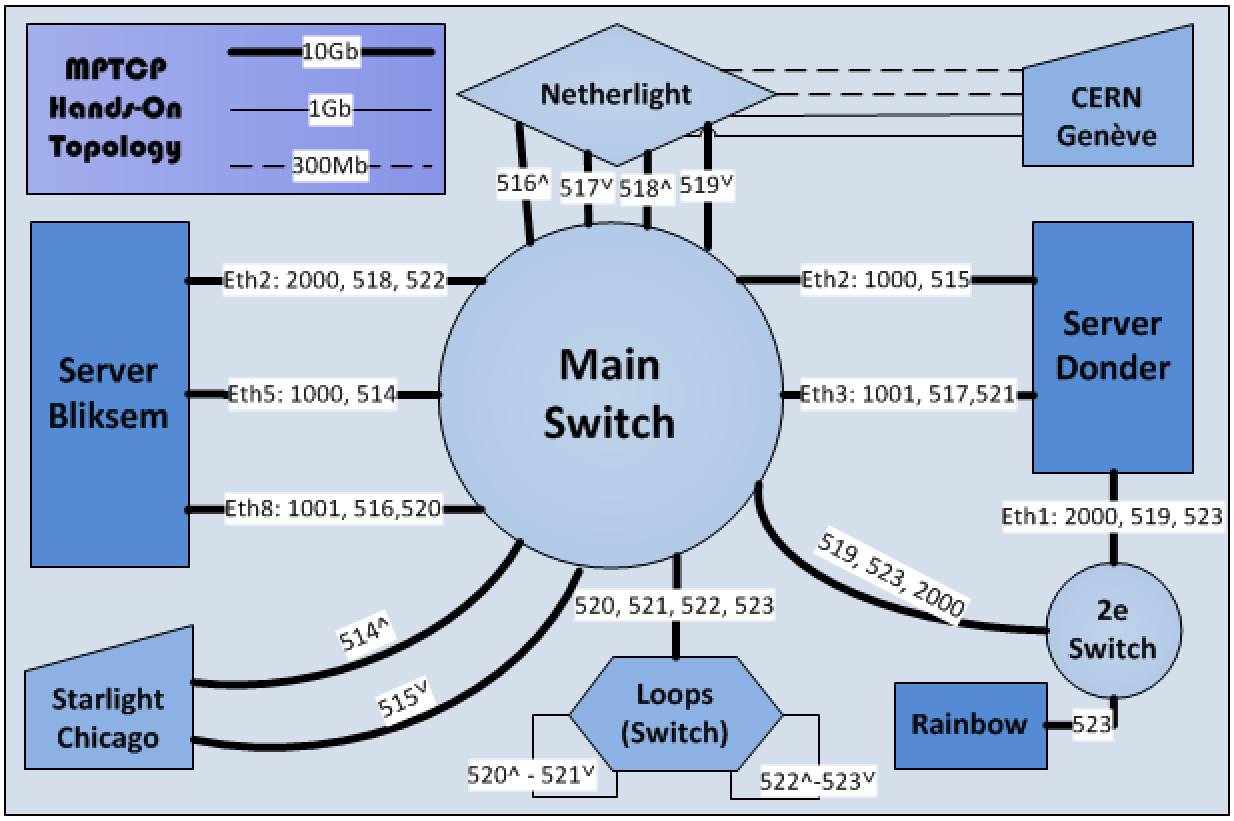


Figure 8: Our test-bed topology used, VLAN numbers are shown

We also compiled Wireshark with the MPTCP patch provided on the INL website on one of our servers, with this we could inspect packet dumps when this was needed. We noticed that the latest Wireshark version (1.8.0) recognizes MPTCP.

For the different paths that should be possible we made use of VLANs, this since it was the easiest way of routing and due to the short time slot for the project. We created a script for this which is more explained in section 3.4. One can have a more complex environment by for example using ECMP or Openflow.

3.2.2 Hands-On

When we had the kernel up and running we had to create different routes first. Our first simple tests with MPTCP worked and we saw that both our interfaces were being used. MPTCP has some parameters build in which can be changed, those can be useful or needed in some situations. With the following command one can configure them: `sysctl -w <item below>`

The following parameters can be changed and the default one is given:

- *net.mptcp.mptcp_ mss=1400* With this parameter, one can change the MSS size, which is needed to reach high speeds for example with 10Gb/s links and greater (max 8900, this since the max MTU= 9000).
- *net.mptcp.mptcp_ ndiffports=1* Defines how many subflows should be created. If it is one a full-mesh is being created otherwise the number of flows specified. When it is zero only the initiated flow will be created.
- *net.mptcp.mptcp_ enabled=1* This enables or disables the MPTCP option in the TCP headers (0=disable, 1=enable).
- *net.mptcp.mptcp_ checksum=1* The checksum parameter for MPTCP can also be disabled to increase performance on high speed links.
- *net.mptcp.mptcp_ debug=0* With this parameter you can enable the debug modus, which can be useful for developers and testers.

Besides the kernel implementation the INL released a special iproute version, which is able to disable specific interfaces for MPTCP. We used this to disable MPTCP for our management interfaces. The git for it can be found here: *git://mptcp.info.ucl.ac.be/iproute2.git.* After installing the needed packages and building/compiling it the iproute folder can be found in the /tmp folder, one can disable a interface with the following command:

```
/tmp/iproute/sbin/ip link set dev <interface-name> multipath off
```

For our experiments to work as they should in theory, we need to increase the buffer sizes in our kernel. Why this is needed should be clear from section 2.3.6. On the website: *http://www.cyberciti.biz/faq/linux-tcp-tuning/* is explained how you can modify the buffer sizes from your kernel. We used the following commands which increased the buffers to 115MByte, which should be sufficient in our case:

```
echo 'net.core.wmem_max=115343360' » /etc/sysctl.conf

echo 'net.core.rmem_max=115343360' » /etc/sysctl.conf

echo 'net.ipv4.tcp_rmem= 10240 87380 115343360' » /etc/sysctl.conf

echo 'net.ipv4.tcp_wmem= 10240 87380 115343360' » /etc/sysctl.conf
```

We must say that the kernel froze some times and we had to reboot it often. Especially in the beginning we had a lot of problems with this. Testing became difficult because scripts got stuck often. The server froze especially often when interfaces went up and down.

3.2.3 Addressing and Routing

This section focuses on the addressing and routing. For MPTCP to be able to work, every subflow should take a different route over a different link or path. There are some possibilities to do this, one can use layer three routing protocol like IS-IS or OSPF, or layer two protocols like ECMP, VLAN's or IEEE 802.1aq. We used different VLANs in this project, for every path we created a different routing table with a default gateway.

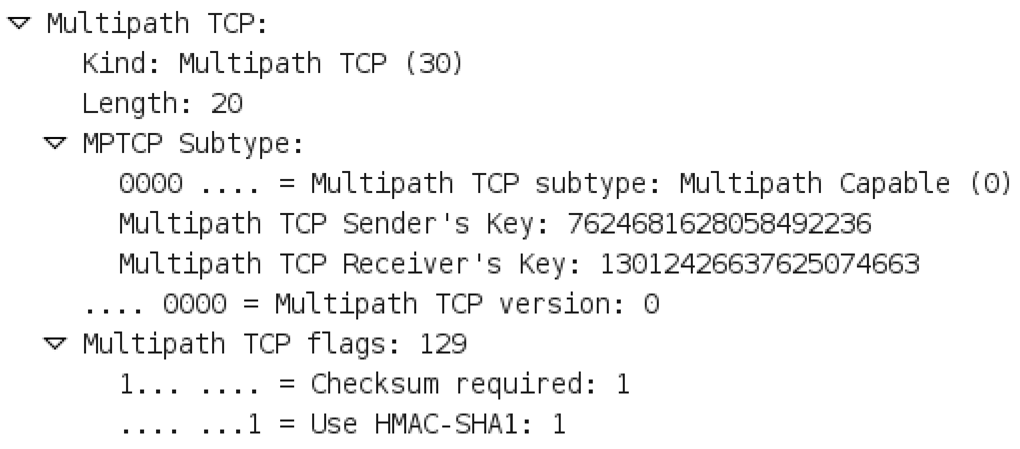
Since MPTCP is a layer four protocol it is capable of handling both IPv4 and IPv6 addresses, even at the same time over different or the same link. For the addressing part we only use IPv4, this because its easier to use and packets have smaller headers. Our servers are multi-homed with 3x a 10Gb/s interface card with in total of 8 VLANs, which means we need 8 IP addresses on both servers. We created scripts which create all VLAN interfaces and configure the interfaces, another script can easily create situation we want like a LAN or WAN environment. Below in table 1 is shown how we addressed all interfaces and show their details.

Host	**Interface**	**IP address**	**Link ID**	**Environment**	**Speed**	**To**
Bliksem x.1	eth0	145.100.37.131	0	MAN	1Gb	Switch
	eth8.1001	192.168.11.1	1	LAN	10Gb	Switch
	eth5.514	192.168.138.2	2	WAN	10Gb	Chicago
	eth5.1000	192.168.12.1	3	LAN	10Gb	Switch
	eth2.2000	192.168.13.1	4	LAN	10Gb	2e Switch
	eth8.520	192.168.21.1	5	LAN	1Gb	Loop Switch
	eth2.522	192.168.22.1	6	LAN	1Gb	Loop Switch
	eth8.516	192.168.23.1	7	WAN	1Gb	Geneve
	eth2.518	192.168.24.1	8	WAN	300Mb	Geneve
Donder x.2	eth0	145.100.37.132	0	MAN	1Gb	Switch
	eth3.1001	192.168.11.2	1	LAN	10Gb	Switch
	eth2.515	192.168.138.18	2	WAN	10Gb	Chicago
	eth2.1000	192.168.12.2	3	LAN	10Gb	Switch
	eth1.2000	192.168.13.2	4	LAN	10Gb	2e Switch
	eth3.521	192.168.21.2	5	LAN	1Gb	Loop Switch
	eth1.523	192.168.22.2	6	LAN	1Gb	Loop Switch
	eth3.517	192.168.23.2	7	WAN	1Gb	Geneve
	eth1.519	192.168.24.2	8	WAN	300Mb	Geneve
Rainbow x.3	em2	192.168.22.3	9	LAN	1Gb	2e Switch

Table 1: Addressing and link information of our Servers

3.2.4 Inspecting Packets

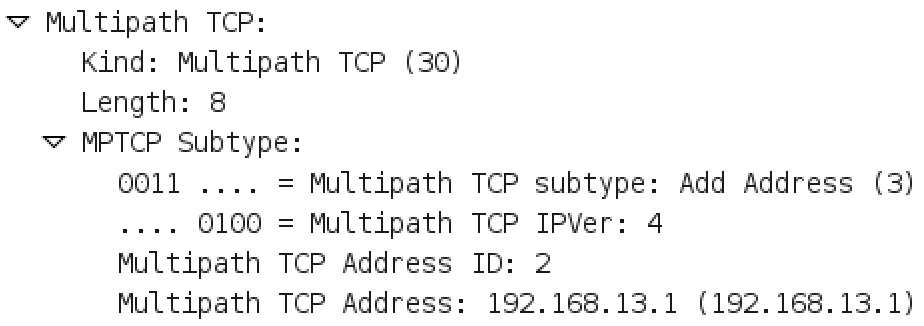
We used the patch available for WireShark to look deeper into the MPTCP packets. We can see all MPTCP attributes in the options of the TCP header. In figure 9 is the subtype: *Multipath Capable* shown, this option is from the sender side and sends both its own and receiver key back to the receiver. This packet verifies that both sides can use MPTCP. The add address subtype explained below can be in the same TCP packet.



```
▽ Multipath TCP:
    Kind: Multipath TCP (30)
    Length: 20
  ▽ MPTCP Subtype:
      0000 .... = Multipath TCP subtype: Multipath Capable (0)
      Multipath TCP Sender's Key: 7624681628058492236
      Multipath TCP Receiver's Key: 13012426637625074663
    .... 0000 = Multipath TCP version: 0
  ▽ Multipath TCP flags: 129
      1... .... = Checksum required: 1
      .... ...1 = Use HMAC-SHA1: 1
```

Figure 9: MPTCP packet with multipath capable subtype

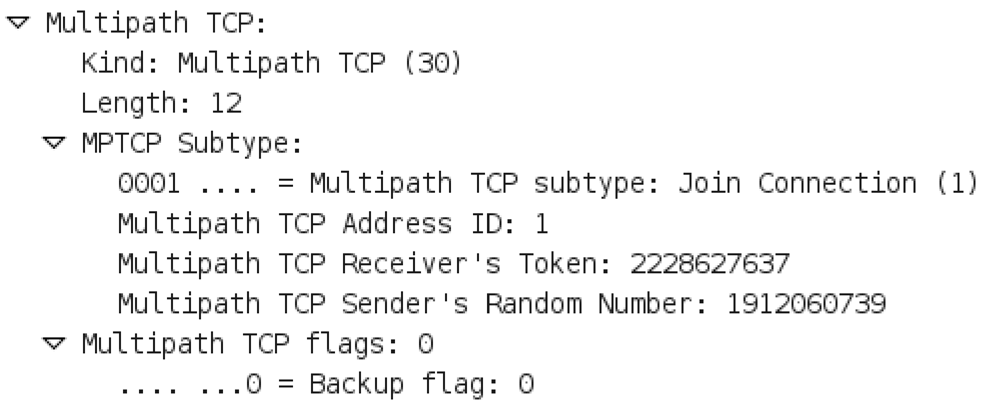
In figure 10 the subtype: *Add Address* is shown. Both the sender and receiver send those packets to each other to show each other which IP addresses they have available. MPTCP then tries to create a whole mess out of the addresses it knows and checks if a connection is possible with the subtype join.



```
▽ Multipath TCP:
    Kind: Multipath TCP (30)
    Length: 8
  ▽ MPTCP Subtype:
      0011 .... = Multipath TCP subtype: Add Address (3)
      .... 0100 = Multipath TCP IPVer: 4
      Multipath TCP Address ID: 2
      Multipath TCP Address: 192.168.13.1 (192.168.13.1)
```

Figure 10: MPTCP packet with the Add Address subtype

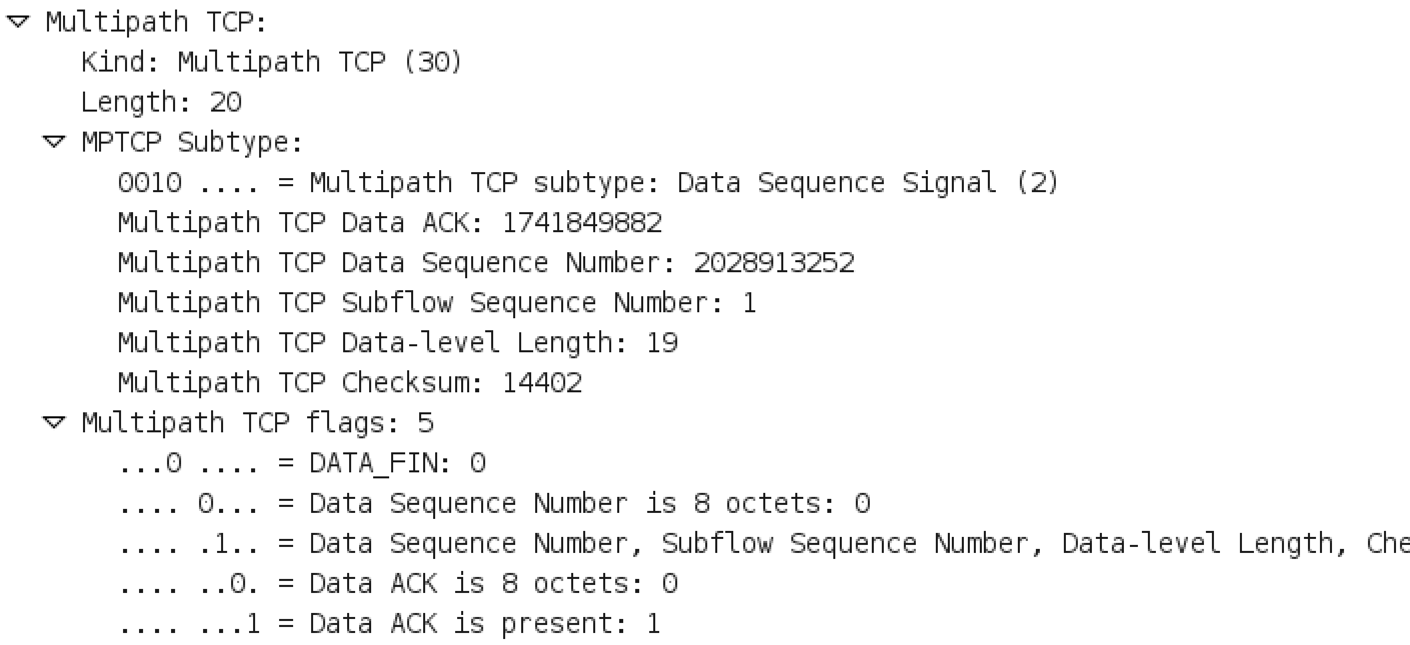
To check if a connection is possible and to create it the MPTCP protocol sends a subtype *Join* option, shown in figure 11. Only the initial sender may send this packet to create additional subflows between the two hosts. When this packet has been successfully arrived by the receiver the subflow is ready.



```
▽ Multipath TCP:
    Kind: Multipath TCP (30)
    Length: 12
  ▽ MPTCP Subtype:
      0001 .... = Multipath TCP subtype: Join Connection (1)
      Multipath TCP Address ID: 1
      Multipath TCP Receiver's Token: 2228627637
      Multipath TCP Sender's Random Number: 1912060739
  ▽ Multipath TCP flags: 0
      .... ...0 = Backup flag: 0
```

Figure 11: MPTCP packet with Join subtype

The subtype *Data Sequence Signal* is a option that contains a packet with the actual data. One can see the ACKed data, the sequence number, the subflow sequence number and the MSS length (data-level) in figure 12.



```
▽ Multipath TCP:
    Kind: Multipath TCP (30)
    Length: 20
  ▽ MPTCP Subtype:
      0010 .... = Multipath TCP subtype: Data Sequence Signal (2)
      Multipath TCP Data ACK: 1741849882
      Multipath TCP Data Sequence Number: 2028913252
      Multipath TCP Subflow Sequence Number: 1
      Multipath TCP Data-level Length: 19
      Multipath TCP Checksum: 14402
  ▽ Multipath TCP flags: 5
      ...0 .... = DATA_FIN: 0
      .... 0... = Data Sequence Number is 8 octets: 0
      .... .1.. = Data Sequence Number, Subflow Sequence Number, Data-level Length, Che
      .... ..0. = Data ACK is 8 octets: 0
      .... ...1 = Data ACK is present: 1
```

Figure 12: MPTCP packet with the Data Sequence Signal subtype

3.3 Experiments and their setups

This section will describe the different experiments we did in our topology and what we think the added value of the experiment is. For every experiment we created a script which also calls our other scripts defined in section 3.4. We verified that we could get the maximum throughput possible with TCP on all links. We need to note that the seconds in the graphs are less than the real tests because the scripts sometimes took a little longer to run.

For our tests we used iperf to measure maximum throughput. All our experiments are performed on our 'bliksem' server to our 'donder' server, we got the best results when the fastest machine acts as sender. Most of our experiments run for 600 seconds to get a good idea how stable it is. It would be a interesting to check how it would function in very long runs in specific environments, but this is something for another project. Some of our experiments, which are based on changes run some longer than 600 seconds to see if the connection is stable and changes on the interfaces take more time to measure properly. We show in our experiments the "best" results of all runs we did for it, since due to time constrains we could not test every experiment extensively.

For most of our experiments we used the following command *iperf -l 64KB -w 3MB $ADDRESS* which seemed sufficient in most LAN environments, for the WAN experiments we needed to increase the buffers as was explained in section 2.3.6. We must note that in iperf the -w options automatically doubles the buffers so if we give it a value of 5MB it turns it into 10MB.

3.3.1 *Experiment A*: Measure throughput on 2x 1Gb/s LAN links

The first experiment is with 2x 1Gb/s Ethernet links. We test if both links are being used and when they are used, if both links are being fully utilized. The experiment runs for 10 minutes and our scripts arrange the addressing, routing and bandwidth monitoring. This test was simple and no advanced options were used.

We ran the test a couple of times and got the following possible scenarios:

- Only one of the links is being used fully
- Both links are being used fully
- In the beginning both links are used later only one
- In the beginning one link is used later both of the links

However after tweaking the parameters into the command *iperf -l 64KB -w 3MB $ADDRESS* we were able to get the same result back every time (more or less at least). Below in figure 13 is shown how it looks like when both links are being fully used. We can verify that it is possible for MPTCP to fully utilizes both links at the same time.

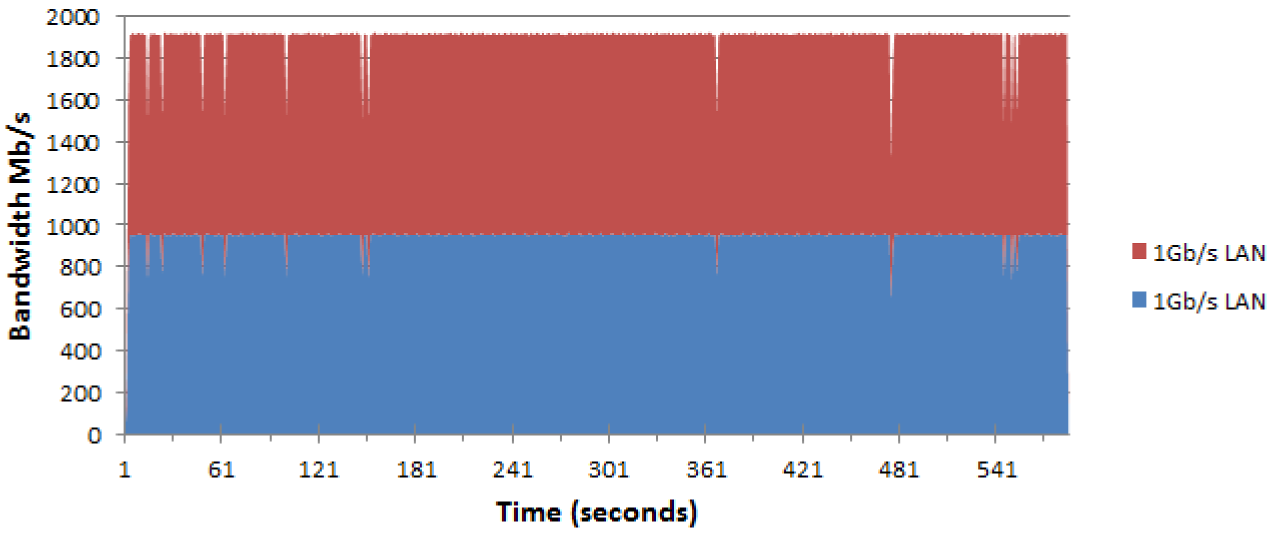


Figure 13: Experiment-A: 2x 1Gb/s link

3.3.2 *Experiment B*: Robustness, 2x 1Gb/s LAN links

In the next experiment we use the same 2x 1Gb/s links from *experiment A*. However this test turns interfaces up and down, this to see how robust MPTCP is and how it responses to changes. The script runs for 900 seconds (15min) and in table 2 is shown what it does on specific time slots.

Time	**Action Taken**	**Max Throughput in Mb/s (±)**	Link Count
0	Only 1Gb link enabled ID1	960	1
120	Second 1Gb link joins ID2	1920	2
420	Link ID1 goes down	960	1
540	Link with ID1 joins again	1920	2
900	Experiment finished	0	0

Table 2: Actions for figure 14

After a couple of runs we can verify the same as in *experiment A* some runs are better than others. Below is the result of the most perfect run, which is shown in figure 14. One can see that the first few minutes everything goes fine. However at time slot 540 something strange happens. Here the first initial link joins again, which was down for a short time frame and after that moment MPTCP only used that link and not the other anymore. In our analyzes we look deeper into this and try to figure out what the problem could be (sections 3.5 and 3.6).

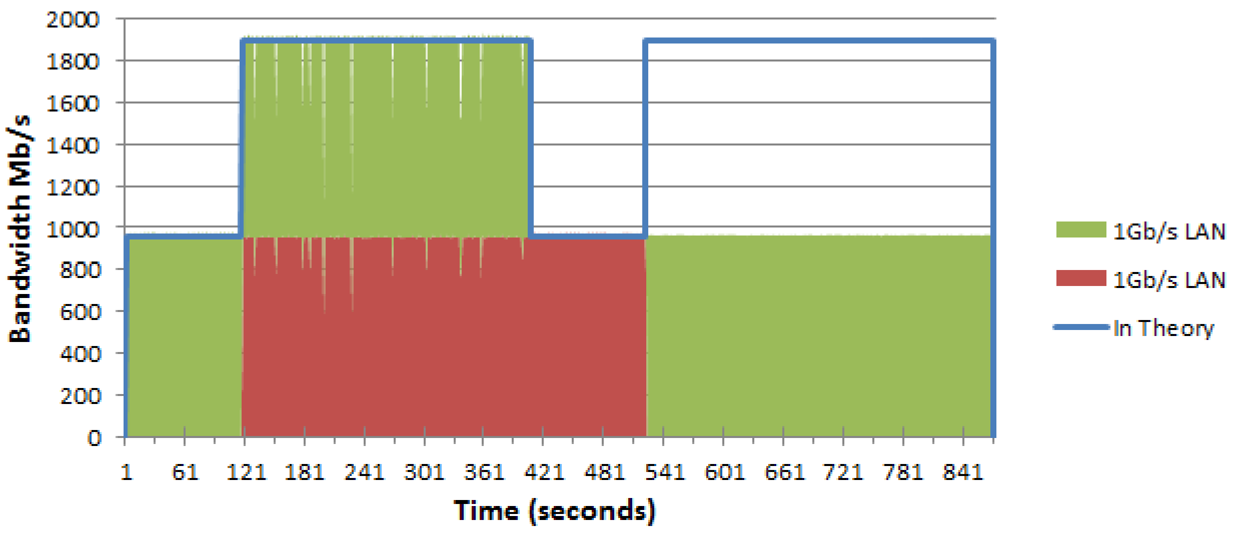


Figure 14: Experiment-B: 2x 1Gb/s link

3.3.3 Experiment C: Throughput on high speeds, 2x 10Gb/s LAN links

This experiment will have faster links, 2x a 10Gb/s link will be used to see if MPTCP functions the same over faster links. This experiment will run for 10 minutes and can be compared with *experiment A*. However, since we use higher speeds we had to increase the MSS to 8900, tuned the kernel buffers [18], disabled the MPTCP checksum and increased the size of the receive/send buffers for iperf to 10MB with the -w option. We increased the buffer even higher but this did not have a more positive effect.

In figure 15 our result shown. We couldn't get higher than around 10Gb/s, we think this is due our server hardware since the receivers CPU was fully used and hardware restrictions of the PCI bus. We see however that one of the links is almost fully used it had a average of 9185 Mb/s.

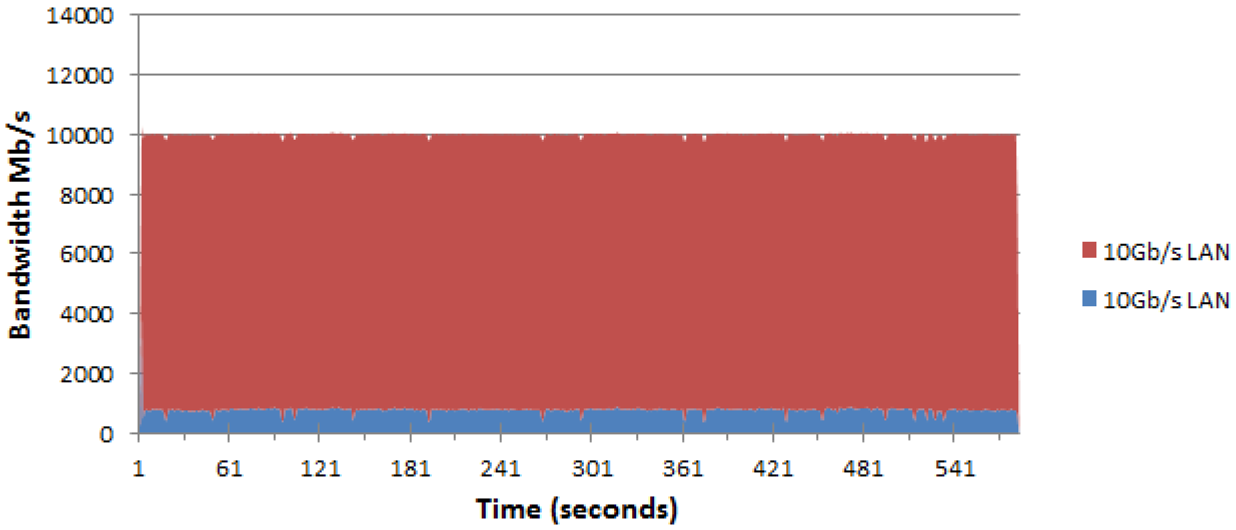


Figure 15: Experiment-C: 2x 10Gb/s link

Besides this we also checked what happened when we opened two iperf sessions at the same time, since we thought that using a single CPU may be the issue. The reason this being faster is that a single iperf session only uses one CPU instead of multiple. When we open two session we should be faster and as can be seen in figure 16 we do get more throughput. The average speed is now 12754 Mb/s, it was also nice to notice that both links got used evenly and got 6393 Mb/s and 6361 Mb/s as throughputs.

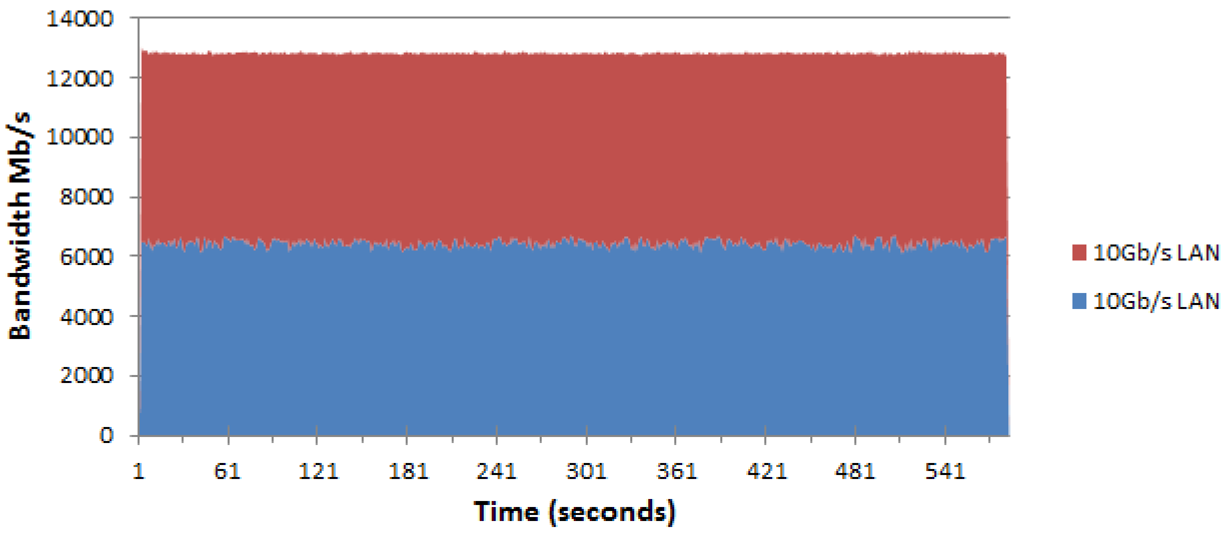


Figure 16: Experiment-C: 2x 10Gb/s link with two iperf sessions

3.3.4 Experiment D: Balancing, 2x 1Gb/s LAN links and 1x 10Gb/s LAN link

This experiment will test how MPTCP behaves with different link speeds available. We test this with both a high and low MSS, to see how MPTCP behaves in both environments. We'll use 2x 1Gb/s and 1x 10Gb/s LAN links with a buffer of 8MB. The tests will run for 10 minutes.

In figure 17 and 18 are two tests with a MSS of 1400 shown. One can see that the results are different and MPTCP uses different paths, this while the exact same run is being used. The run where only the 10Gb/s link is being used is also slightly faster and has a average of 4667 Mb/s, the one where all three links are being used has a throughput of 4438 Mb/s.

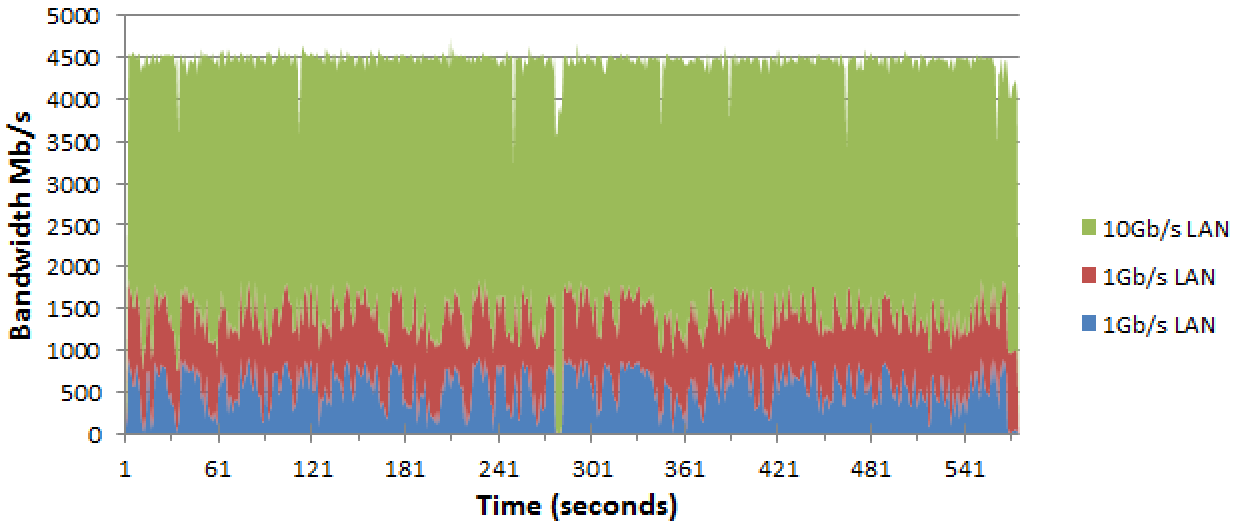


Figure 17: Experiment-D: 2x 1Gb/s, 1x 10Gb/s link, MSS 1400

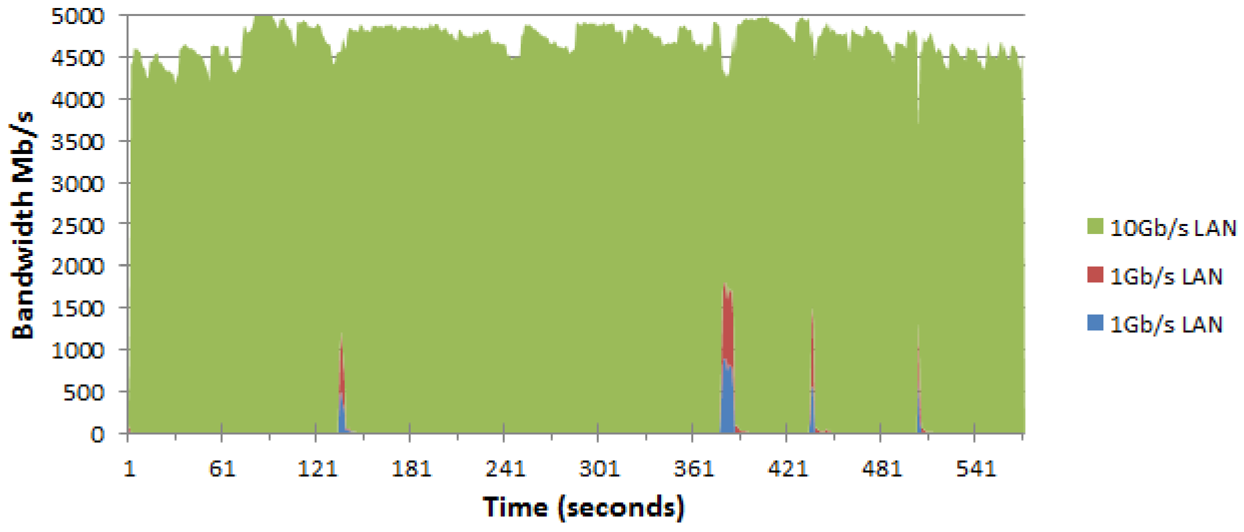


Figure 18: Experiment-D: 2x 1Gb/s, 1x 10Gb/s link, MSS 1400

When we use a MSS of 8900 and high buffers on the 10Gb/s link we get a average throughput of 9540 Mb/s as shown in figure 19. In all repeated test with this setup only one link was being used, the 10Gb/s one. Compared with a low MSS you can say for certain that you must increase your MSS to reach higher speeds. The buffer was increased to 13MB with the -w option.

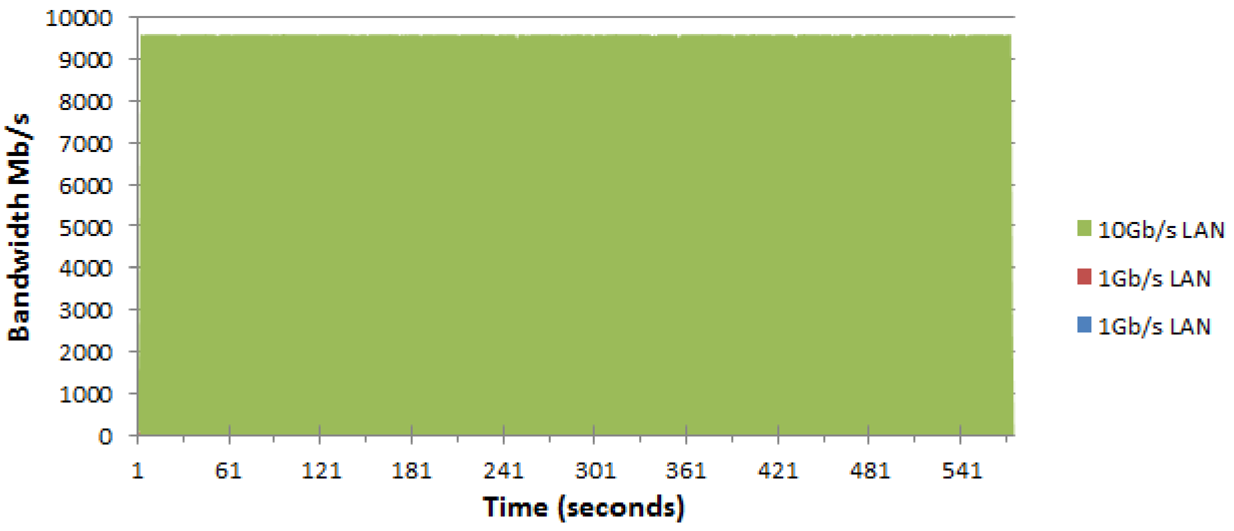


Figure 19: Experiment-D: 2x 1Gb/s, 1x 10Gb/s link, MSS 8900

3.3.5 Experiment E: Robustness and balancing in a LAN environment

Now that we have three interfaces with different bandwidth capabilities it is interesting to see how traffic will flow from one to another and how robust MPTCP is to changes with more links, this experiment can be compared to *experiment B*. We however test this experiment with both a low and high MSS. Below in table 3 is shown what happens at specific time slots, the total run takes 1000 seconds.

Time	**Action Taken**	**Max Throughput in Mb/s (±)**	Link Count
0	Only 1Gb link enabled ID1	960	1
30	Second 1Gb link joins ID2	1920	2
280	Link ID1 goes down	960	1
310	10Gb link joins link ID3	10560	2
610	Link ID2 goes down	9600	1
640	Link ID1 goes up	10560	2
670	Link ID3 goes down	960	1
700	Link ID2 goes up	1920	2
880	Link ID3 goes up	11520	3
1000	Experiment finished	0	0

Table 3: Actions for figures 20 and 21

Below in figure 20 is the result shown with a low MSS. You see the exact same problem coming up as which we saw in *experiment B*, as soon as another link joins the other one may stop working but this is not always the case. We try to answer this in section 3.6.2.

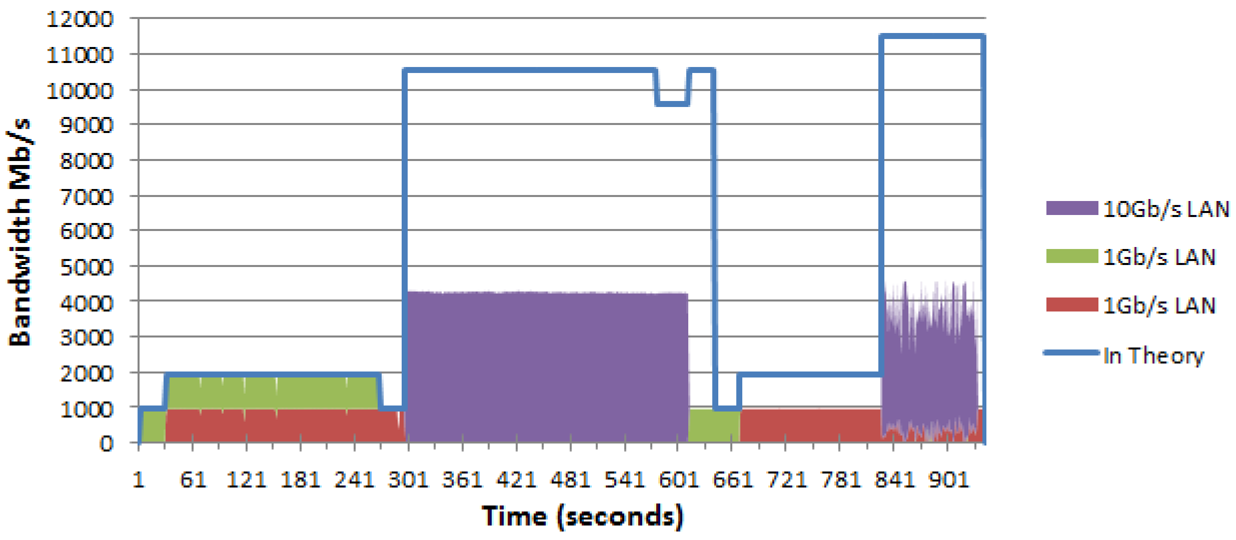


Figure 20: Experiment-E: 2x 1Gb/s, 1x 10Gb/s link, MSS 1400

We can suspect that in the test with a high MSS the exact same problem would show up and that the 10Gb/s link would get more throughput. As can be seen in figure 21 this is true in both cases. We used a 8MB buffer so that the 1Gb/s links also should be used optimal, increasing the buffers higher in both cases did not have any positive effect.

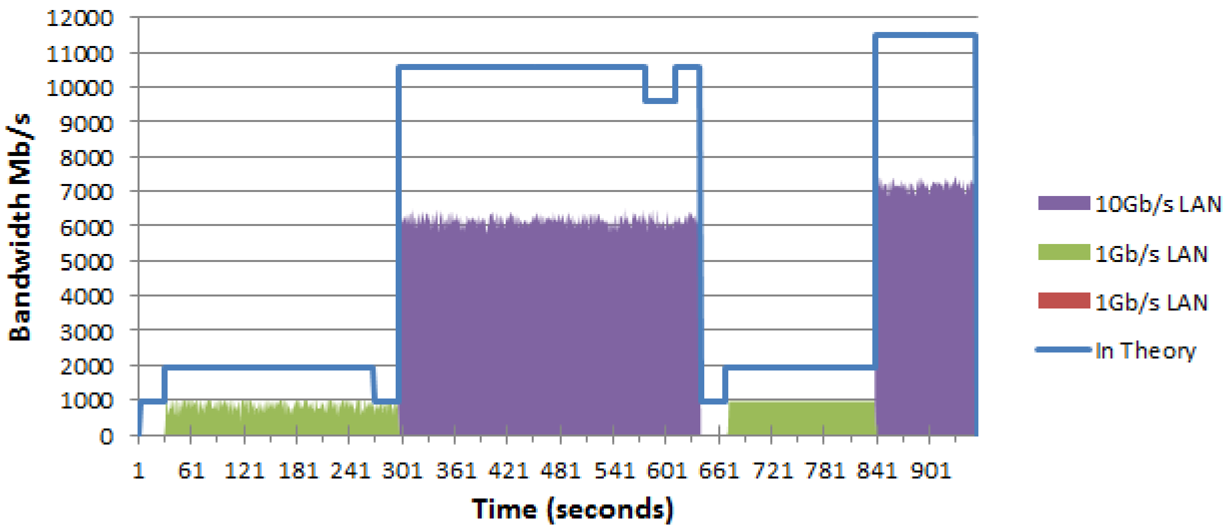


Figure 21: Experiment-E: 2x 1Gb/s, 1x 10Gb/s link, MSS 8900

3.3.6 Experiment F: Throughput in WAN environment, 1x 1Gb/s link and 1x 300Mb/s link

In this experiment we test how well MPTCP performs in a WAN environment. In a WAN environment the RTT is higher and servers need the ability to capture more packets to put them back in the right sequence, this is done by increasing buffer sizes. We will use two loops to Geneve, one has a speed of 300Mb/s and the other of 1Gb/s.

We test this first with different buffer sizes, the result is shown in figure 22. We see that the best total throughput was with a buffer size of 5MB with 955 Mb/s as throughput. The particular result from this experiment with a 5MB buffer size is shown in figure 23. It is however strange to see that the performance drops when the buffer size increases, in theory this would stay stable after some point.

When we calculated the needed buffer size with:

$$Buffersize_{MPTCP} = RTT_{max} * AllLinksMax_{bits} * 2 \tag{6}$$

We get: $0.033 * 1300000000 * 2 = 85800000bit = 10,73$MB. Which more or less corresponds with our graph, since iperf doubles the 5MB which gives us 10MB.

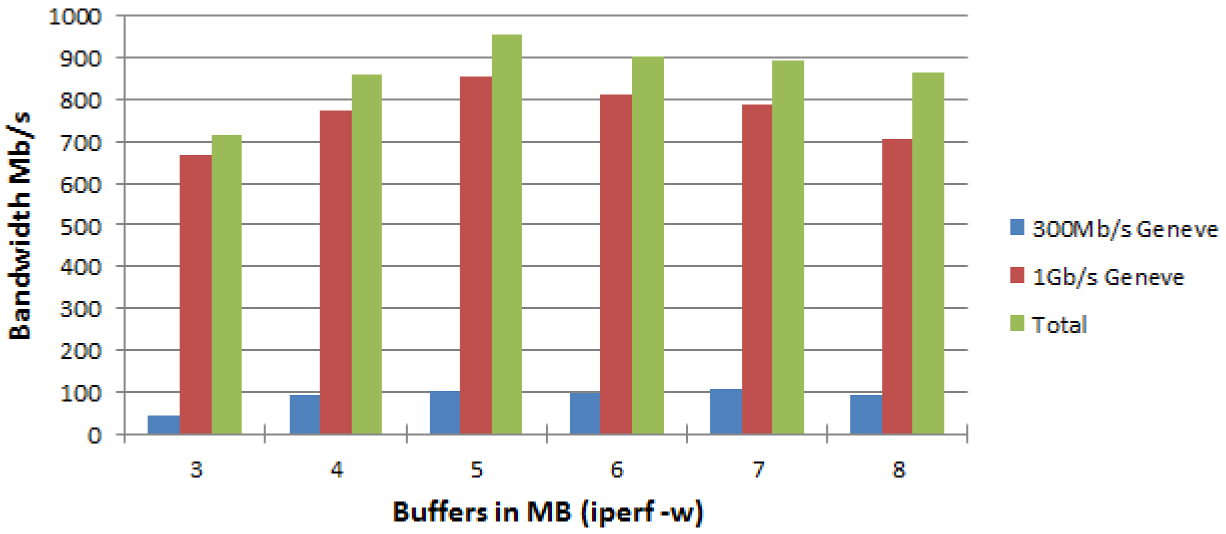


Figure 22: Experiment-F: Sum(1x 1Gb/s + 1x 300Mb/s) WAN loops

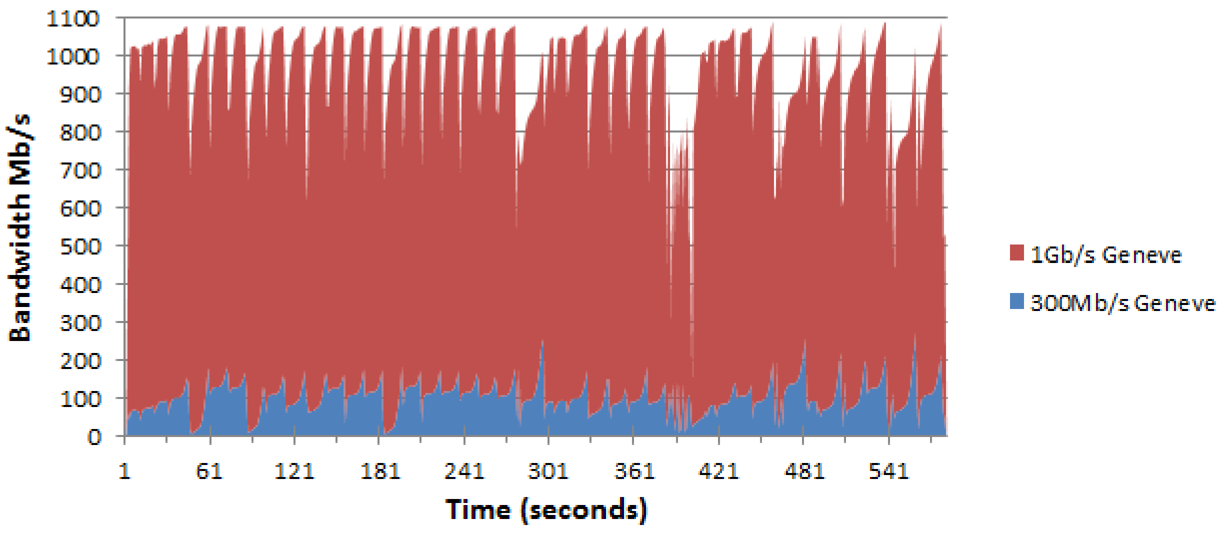


Figure 23: Experiment-F: 1x 1Gb/s and 1x 300Mb/s WAN loops, buffer 5MB

3.3.7 Experiment G: RTT Difference, WAN environment over three links

This is the second WAN environment experiment. Here things get a bit more complex since we at another WAN link. Besides the links from *experiment F* we added a loop to Chicago, this loop has a speed of 10Gb/s. This experiment is interesting since both loops have different RTT response times, which means packets will dramatically arrive out of order. The RTT value for links to Geneve are 33,3ms while for Chicago 202ms, which is a large RTT difference. We expected to get more throughput than *experiment F* at first, this since we use more links. However, our results deviated from these expectations.

After several runs we can say that the 300Mb/s and 1Gb/s link are mainly used when buffers are small 3-9MB. Around 12-15MB the MPTCP takes a different path and takes mostly the 10Gb/s link instead of the other two, however the test with a buffer of 21MB showed some inconsistency. We used a low MSS in this experiment. When we used very high buffers our servers had problems and could not get a stable throughput. The summary result is shown in figure 24.

The total throughput when using a buffer of 6MB was 472Mb/s (shown in figure 25), which is way less than we were using only the two Geneve links, where we got 905Mb/s throughput as was shown in *experiment F*. So using more links is not always faster you need a good balance between the link speeds, MSS, RTT and buffers to get the most optimal throughput.

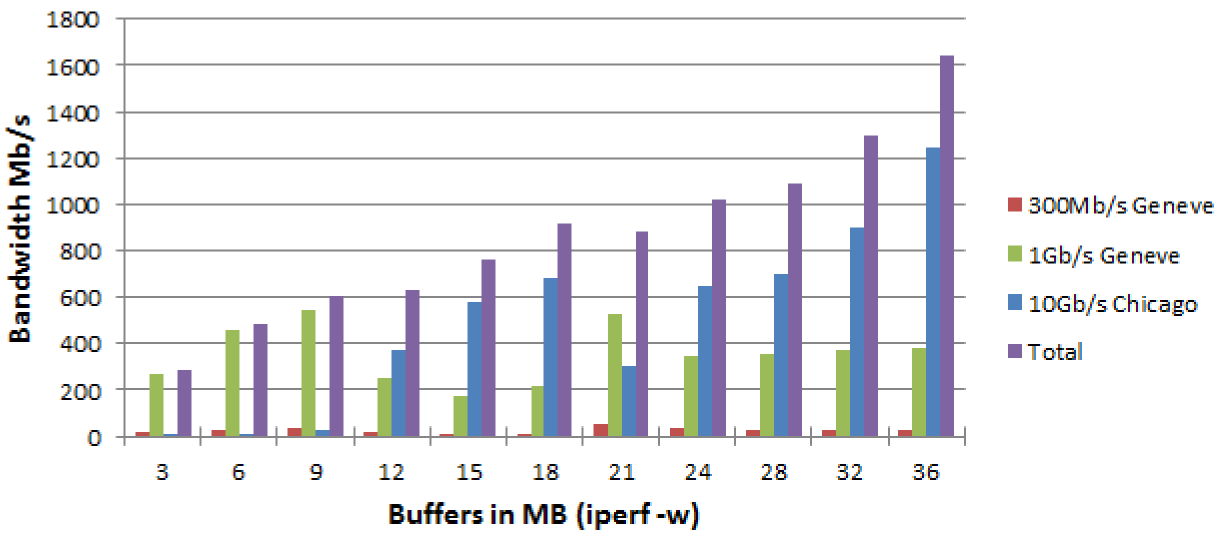


Figure 24: Experiment-G: Summary with different buffer sizes

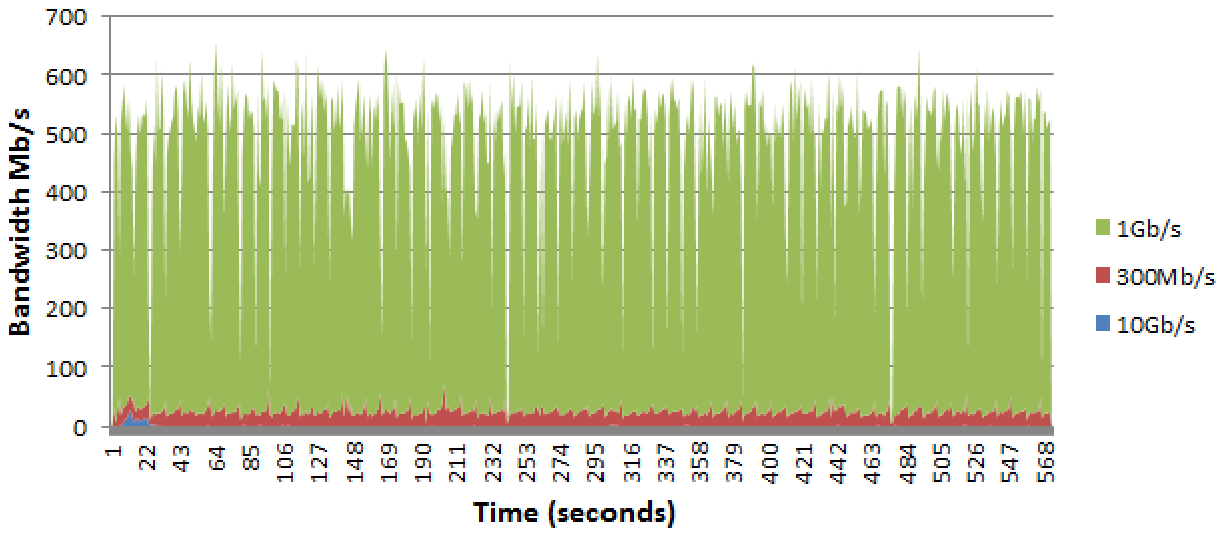


Figure 25: Experiment-G: 2x 1Gb and 1x 300Mb/s WAN loops with different RTTs, buffer 6MB

3.3.8 Experiment H: RTT difference, 1x LAN and WAN link

In this experiment we make both use of a LAN and WAN link at the same time, this to see if a low RTT difference is possible. This experiment can be compared to *experiment G*, we can already suspect from it that it will work well with the Geneve link and worse with the Chicago link, because of the big difference in RTT.

In figure 26 is shown how it looks like when both a 1Gb/s LAN link and a 1Gb/s Geneve link is being used with a buffer of 5MB. One can see that it performs well, in the LAN environment we got a average of 1899Mb/s throughput and in this one 1878Mb/s which is comparable (only a 1% difference). Due to time restrains we could not test the Chicago link.

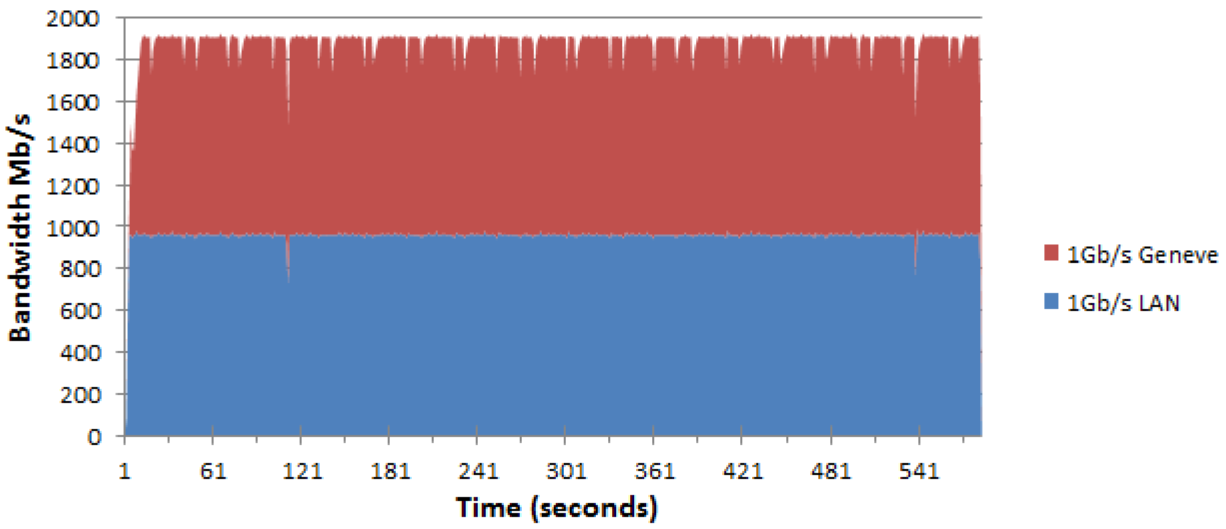


Figure 26: Experiment-H: 2x 1Gb/s one in a LAN other in a WAN environment, buffer 5MB

3.3.9 Experiment I: Fairness, 1x and 2x 1Gb/s link(s)

Our last experiment is for testing how well MPTCP behaves when a link is used at the same time with other traffic. This "other traffic" is in our case another iperf connection which is using TCP. We use our machine 'rainbow' for this, which has a TCP kernel and acts as the receiver for this connection. In theory both links should get half of the link, since MPTCP should be fair to other traffic on the same link.

In figure 27 is shown how MPTCP behaves when only one link is being used for both sessions. One can notice that MPTCP is not really fair for TCP and MPTCP gets a lot more throughput. The actual numbers are 243Mb/s for TCP and 686Mb/s for MPTCP, which is a big difference. In theory they should both get around ±480Mb/s.

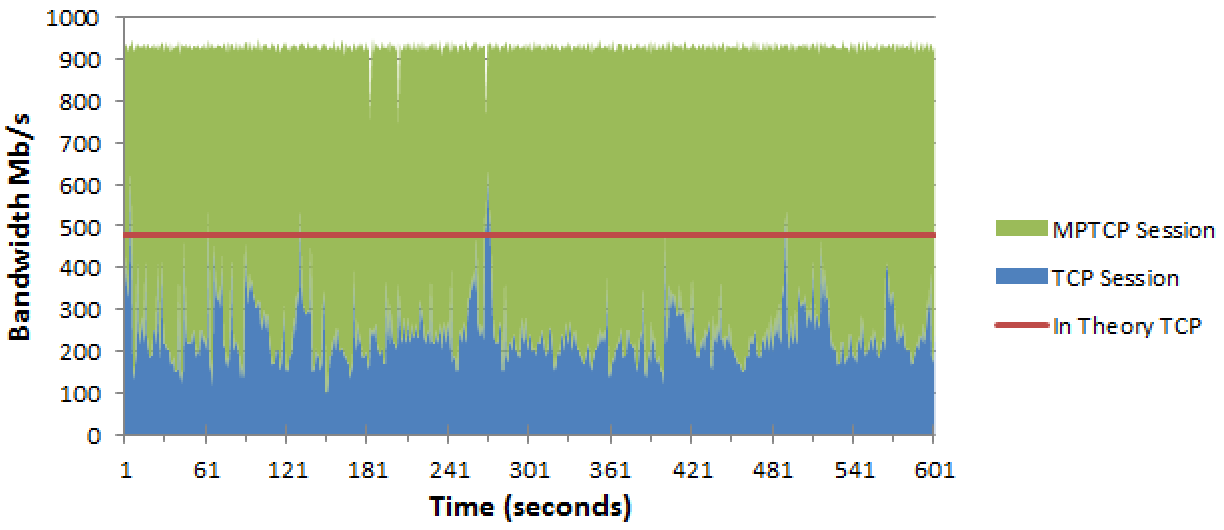


Figure 27: Experiment-I: 1x 1Gb/s link, with one MPTCP and one TCP session

We did this experiment again but now with two times a 1Gb/s link, shown in figure 28. Now MPTCP should use two links while the TCP connection would only use one. One can still conclude that TCP gets way less than it should get the actual numbers are 306Mb/s for TCP and 1468 for MPTCP.

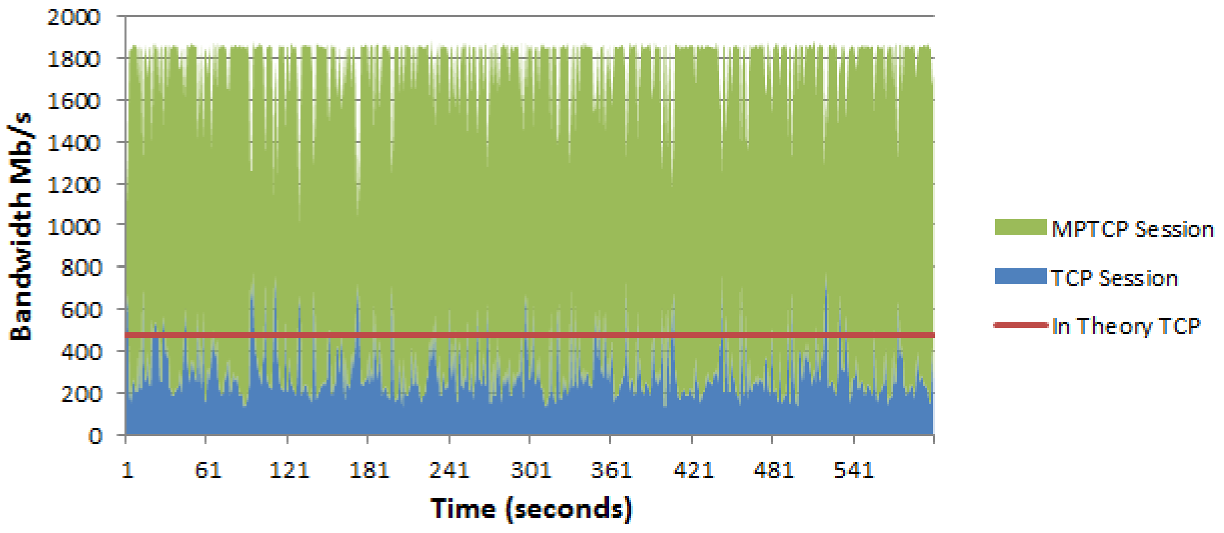


Figure 28: Experiment-I: 2x 1Gb/s link, one which is congested with TCP session

We again did this same experiment with two TCP sessions on both one and two times 1Gb/s. Here MPTCP was a bit more fair to TCP but still 20% less than it should get in theory, this was 40% less when only one link was being used. As a summary we show the results of *experiment I* in figure 29.

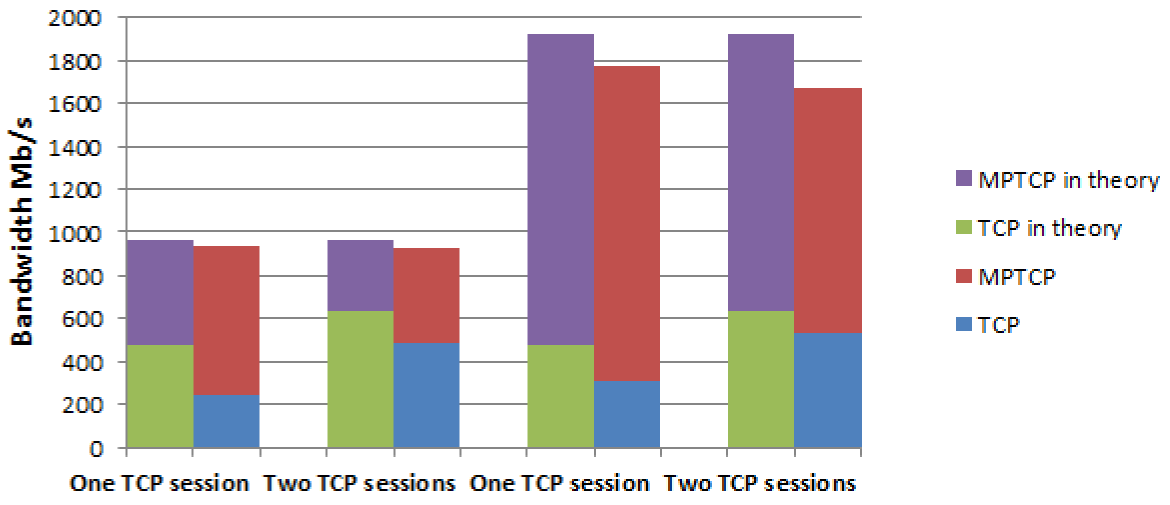


Figure 29: Experiment-I: Summary overview of the congestion experiment

3.4 Scripts created

We created scripts for the following topics within this project:

- Routing Configurations
- Interface states
- Monitoring Bandwidth
- Experiments

Routing Configurations

This script creates all the VLANs needed and configures the interfaces correctly. Besides this it disables the management interface for MPTCP and creates the routing tables, which contain default gateways for the different interfaces and paths needed. The script for our 'bliksem' server can be found in appendix C.

Interfaces States

A second script was needed for easy configuration of the desired situation. The script can easily enable the LAN or WAN interfaces and a user can even specify to enable only the 1Gb/s or 10Gb/s interfaces. One can use the script for example with: *./interfaces blik lan 10* which will enable the 10Gb/s lan interfaces on our server bliksem. The script can be found in appendix D.

Monitoring Bandwidth

To get all the results we needed to read the in and output bits going over the different interfaces. This script can easily monitor any interface specified and puts it into a log file, it can also monitor the RTT to the other side. When we look at the final results we could get the interface details from those logs. One can call the script with: *./monitor eth5.1000 eth2.1001* this would read out those two interfaces every second. The script can be found in appendix E.

Experiment Scripts

For every experiment explained above we created a script. The scripts do not vary a lot from each other. With the other scripts created we could easily change the environment and capture the information we needed. The script for our *experiment I* can be found in appendix F.

3.5 Analysis with MPTCPs Debug function

It is possible to enable the debug mode for MPTCP one can do this with the following command *sysctl -w net.mptcp.mptcp_debug=1*. When it is enabled the kernel will log information in the file /var/log/syslog. We looked at what happened in *experiment B* and checked if anything suspicious could be found. We defined all stages in *experiment B*, they are shown in table 4, we then marked them in the log below it. The first number is the stage the second number is the subflow ID, the number 0 is the main MPTCP process.

Stage	**Meaning**
0	Starting the connection
1	First connection is made
2	New interface goes up
3	Initial interface goes down
4	Initial interface goes up again
5	Closing the connection

Table 4: Stages in experiment B

```
0-0     Jun 26 10:49:51 donder kernel: [  638.534479] net/mptcp/mptcp_ctrl.c:
    mptcp_alloc_mpcb: created mpcb with token 0xc2b4e677
1-1     Jun 26 10:49:51 donder kernel: [  638.534486] net/mptcp/mptcp_ctrl.c:
    mptcp_add_sock: token 0xc2b4e677 pi 1, src_addr:192.168.22.2:5001
    dst_addr:192.168.22.1:34694, cnt_subflows now 1
2-2     Jun 26 10:50:51 donder kernel: [  698.536532] net/mptcp/mptcp_pm.c:
    mptcp_lookup_join:mpcb not found:d17275f0
2-2     Jun 26 10:50:51 donder kernel: [  698.536717] net/mptcp/mptcp_ctrl.c:
    mptcp_add_sock: token 0xc2b4e677 pi 2, src_addr:192.168.21.2:5001
    dst_addr:192.168.21.1:36214, cnt_subflows now 2
3-1     Jun 26 10:52:51 donder kernel: [  818.541432] net/mptcp/mptcp_ctrl.c:
    mptcp_del_sock: Removing subsock tok 0xc2b4e677 pi:1 state 7 is_meta? 0
4-3     Jun 26 10:53:51 donder kernel: [  878.544121] net/mptcp/mptcp_pm.c:
    mptcp_lookup_join:mpcb not found:54cf0d62
4-3     Jun 26 10:53:51 donder kernel: [  878.544336] net/mptcp/mptcp_ctrl.c:
    mptcp_add_sock: token 0xc2b4e677 pi 3, src_addr:192.168.22.2:5001
    dst_addr:192.168.22.1:44803, cnt_subflows now 1
5-0     Jun 26 10:55:51 donder kernel: [  998.523876] net/mptcp/mptcp_ctrl.c:
    mptcp_close: Close of meta_sk with tok 0xc2b4e677
5-3     Jun 26 10:55:51 donder kernel: [  998.524072] net/mptcp/mptcp_ctrl.c:
    mptcp_del_sock: Removing subsock tok 0xc2b4e677 pi:3 state 7 is_meta? 0
5-2     Jun 26 10:55:51 donder kernel: [  998.524132] net/mptcp/mptcp_ctrl.c:
    mptcp_del_sock: Removing subsock tok 0xc2b4e677 pi:2 state 7 is_meta? 0
5-0     Jun 26 10:55:51 donder kernel: [  998.524136] net/mptcp/mptcp_ctrl.c:
    mptcp_release_mpcb: Will free mpcb 0xc2b4e677
```

The connection opens at 0-0. At 1-1 the first subflow is created and at 2-2 the other interface joins where a new subflow gets created. At 3-1 the initial subflow closes. In 4-3 the initial interface is used again but now for a new subflow with ID3. For some reason the total subflow count stays at 1, this while 2 subflows should be available. Finally in 5-0 the connection gets closed again. The log does not give a clear reason why the other flow is not used anymore in *experiment B*.

3.6 Analysis with tcpdump and Wireshark

3.6.1 Analysis of dips in the experiment graphs

We wanted to know what the cause of all the downward spikes were in our graphs. We captured a packet dump of one minute and used the setup of *experiment A*. From those captures we grabbed the throughput and window sizes, we plotted those in figure 30. One can see that the drops of throughput map the drops of the window size, this happened at both interfaces at the same time. We can say that this is most likely happening due to the slow server or buffers overflowing which causes packets to drop.

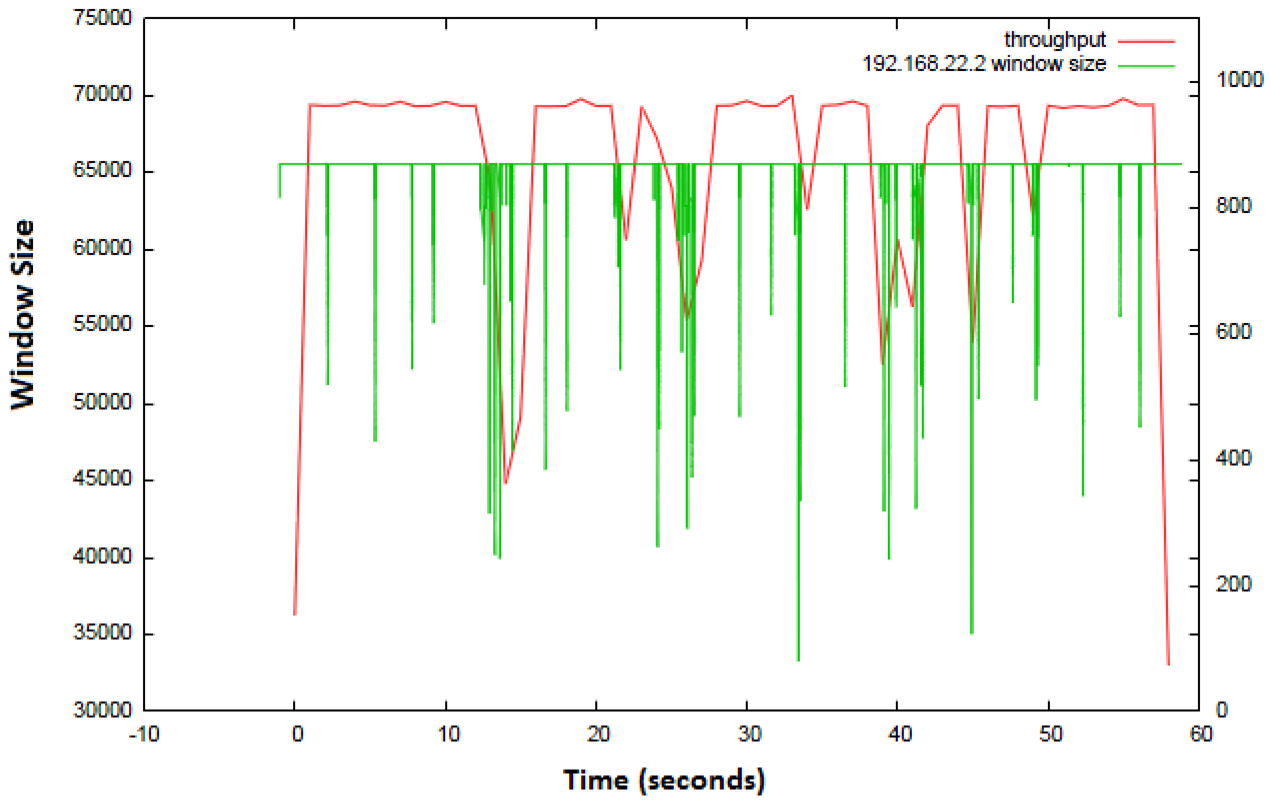


Figure 30: Analysis of the downward spikes in our graphs

3.6.2 Analysis when old interface joins again

We were also really interested in the changes experiments, this since in *experiments B and E* is shown that they do not work well. For those experiments we also looked at the debug logs, just as described in section 3.5.

In Wireshark we analyzed the packets, with the following line you can filter on specific subtypes *tcp.options.mptcp.subtype == 1*. One can change the number to show other subtype packets (subtype numbers can be seen in the images in section 3.2.4). It was a bit unhandy to look at our results since tcpdumps became very large, this even when we captured only the headers. One minute of capturing resulted in a file of 3GB.

We filter the dump on the moment were the 'handover' from one interface to the other takes place, see figure 31. We did this by searching for subtypes numbers 3 and 1, which are the add and join subtypes. We saw that the sender 'bliksem' sends the new available address and a few packets later the join packet is send. After this the other side sends a join ACK package back. Than follow some data packets from the old subflow but they stop very fast and after that point the other interfaces was not used anymore. We filtered on the IP address of that particular interface and we could see that, after that point no MPTCP packets were send over that link anymore. It seems as if that flow just stops.

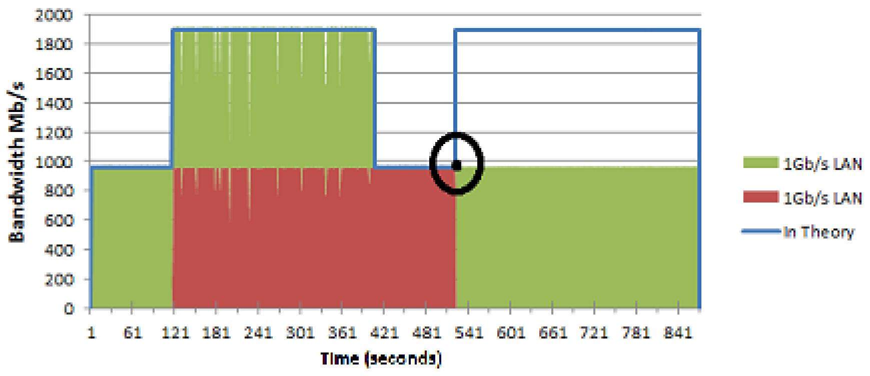


Figure 31: Which part we zoomed in on in *experiment B*

When we analyzed further we saw a very strange behavior of the window sizes of both interfaces. One can see in figure 32, that after a few packets the working interfaces stops and only the other interface is being used. What is really interesting is that the windows size is tremendously large (a socket buffer of 3MB was used). Path 1 is in the beginning still normal with a 65535 window size but stops very fast after that. We can not explain the Path 2 its very first value is also 65535 but very fast increases to a window size of 9427456, at the end I cut off the graph and the window size stayed stable at windows size of 14155776. So due that one of the interfaces gets a very large window size the other one is most likely not use anymore. This is only an indication and we can't say this for certain.

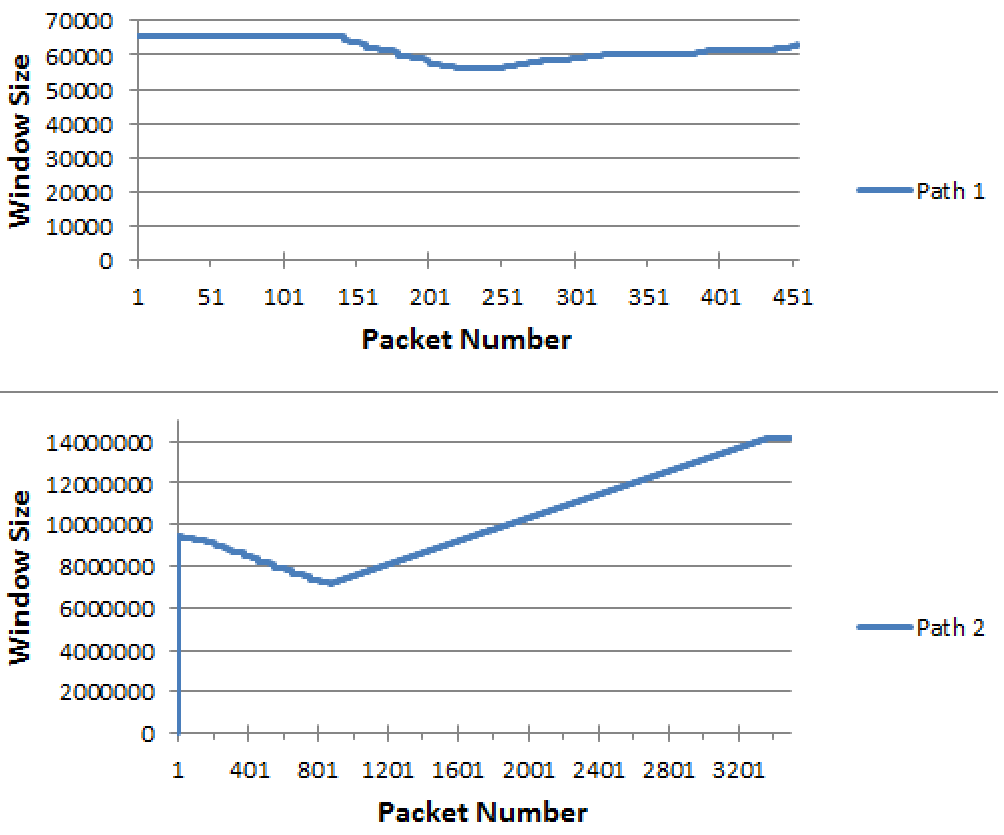


Figure 32: The two interfaces as paths, they both start at the time where both links should be used

4 Results Discussion

This chapter discusses MPTCP in combination with the results we got from our experiments.

4.1 Improve Throughput

From all experiments we can verify that MPTCP is at least as fast as a single TCP session on the best link available. With all experiments one of the links was always fully occupied. In the *experiments A, D, F and G* one can see this well. We must say this only works well when your buffers are big enough to handle the received data from multiple links.

MPTCP makes the best use of all links when they use the same bandwidth speed. This due the changes in the MSS as could be seen in *experiment D*. The 1Gb/s links are not used properly when a high speed link is being used and a high MSS is configured. The other way around is also true that with a low MSS a 10Gb/s link can't get full throughput over it. So in both cases you would not get the maximum throughput possible due to those differences. When you use links with the same maximum throughput you would get the best performance.

Another factor which should be taken into account is the RTT. A high buffer should narrow the negative effect of the difference in RTT. But as can be seen in *experiment G* a big difference in RTT does not work well and does not give maximum throughput. However, with a lower RTT difference it is still possible to get maximum throughput as is shown in *experiment H* .

In the end we couldn't get full throughput with our 2x 10Gb/s links in *experiment C*, this however is most likely due to the servers and not due the MPTCP protocol. Christoph Paash (one of the MPTCP implementers) mentioned by email that he once managed to get 30Gb/s while using MPTCP.

We can say that MPTCP meets its *goal 1: improve throughput* as defined in section 2.3.2. MPTCP gets the best throughput when the same link speeds are being used and the different paths have a low RTT difference. When high speeds are being used one should take the buffers into account, the buffer size needed for MPTCP is way more than TCP as described in section 2.3.6.

4.2 Balancing and Congestion

The balancing with a single MPTCP process is not always evenly balanced. Especially when we were testing the high speed links in *experiment C*, usually only one of the links was fully occupied and the other a little bit. This is however not bad since both links were not congested. One could see that when we use two sessions at the same time MPTCP does his job well and both links get evenly balanced. In *experiment D* the result was changing every time when a low MSS was being used, traffic was balanced over all of the links or only the 10Gb/s one.

In *experiments F and G* is shown that MPTCP can still balance traffic with different RTTs and even when there is a huge bandwidth difference, this is however not always optimal, since using a single link can get more throughput. Also shown in *experiment H* is that even a LAN and WAN link can work well together.

Our last *experiment I* showed that MPTCP is still a little bit unfair for normal TCP traffic. When they were both used at the same time MPTCP got more throughput than it should get in theory.

After our experiments we can say that MPTCP meets its balancing goal very well when the same link speeds are being used and that it meets *goal 3: balance congestion*. This is however a bit less the case when different link speeds are being used. We are also a little bit concerned about *goal 2: do no harm*, since it did some harm in our experiments and TCP got less throughput than it should have gotten. However we must say we could not test everything very extensively.

4.3 Robustness and Changes

Besides the improved throughput and congestion difference in MPTCP it was also interesting to see how robust the protocol was. In *experiment B* we looked at this and can say its behavior is still very strange. When a link goes down and goes up again it gets the whole traffic flow and the other link is not used anymore. We tried to look at what was wrong or happening as described in sections 3.5 and 3.6, but we were not able to say anything for curtain in the short time slot. When we tested MPTCP again against changes in *experiment E* the same problems showed up as in *experiment B*.

But besides those link interfaces going up and down the protocol was robust when the connection was stable. It was still working when a link went down, which means it can be more reliable than TCP already, since this would end your whole connection (when it times out).

4.4 Our Experience

It first took us quite some time to get ourself familiar with MPTCP and get the whole topology up and ready to use. Especially creating the different VLANs and configurations took some time. In the end we were quite happy with the topology we got and could test all our experiments we had in mind. How our topology was setup can be read in the first sections of chapter 3 our experiments are described in section 3.3.

We can say that overall the current MPTCP kernel implementation worked well. Sometimes our servers froze but we can't verify this was due MPTCP or some other factor, we also had some issues with the memory sometimes. We were happy that we had the time and opportunity to look at MPTCP and believe it can perform well in particular cases. We must also say that due the short time frame for this project we could not test everything in such an extend as we wanted.

5 Conclusion

This project aimed to answer the following research question:

> *Is the current* MPTCP *implementation a useful technology for e-science data transfers in the* GLIF *environment?*

During our research, we worked towards answering the main research question by answering the identified subquestions. This section restates the research and summarizes the findings in this project.

5.1 Final Conclusion

Familiarizing ourself with MPTCP and creating the whole test topology took us some time. Also creating the MPTCP environment correctly and configuring interfaces/routing tables was time consuming. But in the end everything turned out quite well, with some problems here and there. We can say that MPTCP worked optimal in our layer two VLAN setup. It is capable of using both IPv4 and IPv6 at the same time. MPTCP could even get maximum throughput when both a LAN and WAN path were being used at the same time.

Looked at the MPTCP protocol side, we can say that it works most optimal in stable environment. When MPTCP uses links with the same speed and little RTT difference it gets the best performance and throughput. For higher speeds one should also take buffer sizes in real consideration (both the kernel and TCP socket buffers), they need to be a lot bigger than with TCP was the case. One should also not forget to increase the MSS size in high speed environments. MPTCP behaves worse when links have large bandwidth- or RTT differences. It also seems that MPTCP has a particular problem when interfaces go down and come up again, we tried to look deeper into this but due to the short time we could not determine the cause.

While researching MPTCP, our experiments showed a little unfairness regarding general TCP traffic, which MPTCP defined as the *do no harm* goal. The other two goals *improve throughput* and *balance congestion* are being well met in our experiments.

When the *e-science data transfers in the* GLIF *environment* is a stable environment, with the same link speeds, buffer sizes and RTTs MPTCP should perform well. However when this environment is less stable, has different link speeds, buffers and RTTs one should really consider if using MPTCP will give any real benefit compared to TCP. This because MPTCP needs higher buffers and uses more CPU. However, when availability is a key factor you can of course use MPTCP, since any of your links can break while the others would still continue functioning.

As a result we can say that MPTCP may work well in a e-science environment, this depending on the actual environment details. We can say that MPTCP can improve throughput and robustness when more than one path is available. We are very curious where and when we will see more of MPTCP in the near future.

5.2 Future Work

Due that MPTCP is still being developed and the short time frame of this project a lot of future work can be done. After this research we got some additional questions which could be looked into further. As an optional question we wanted to look how well MPTCP performs compared to a application like GridFTP, which opens multiple TCP session over different interfaces. We are also curious why MPTCP behaves strange when links go down and up again and why this is happening, we wanted to test this over a longer time frame but did not have time for this. In addition to that our experiments can be run again to verify the results we ended up with. Also will MPTCP be further researched in the GLIF topology currently created by SARA, which this research formed a basis for. We can sum everything up to the following:

- More advanced analyzing and testing of MPTCP
- Testing against other projects like GridFTP
- Run our experiment again to verify the results
- Further investigate the tuning of parameters
- Research in the GLIF test-bed topology within SARA

A Acronyms

API Application Programming Interface

ECMP Equal-Cost Multi-Path

FTP File Transfer Protocol

GLIF Global Lambda Integrated Facility

IETF Internet Engineering Task Force

INL IP Networking Lab

LAN Local Area Network

MPTCP MultiPath TCP

MSS Maximum Segment Size

MTU Maximum Transmission Unit

QoS Quality of Service

RFC Request For Comments

RTT Round-Trip Time

SCTP Stream Control Transmission Protocol

TCP Transmission Control Protocol

UCL Universite Catholique de Louvain

UCL University College London

UDP User Datagram Protocol

UvA University of Amsterdam

VLAN Virtual LAN

WAN Wide Area Network

B Specifications of our servers

Bliksem:

Supermicro X8DAH+-F

2x Intel Xeon X5677 3.73 GHz (8 cores total)

24GB DDR3 1333 MHz

2x Myricom 10G-PCIE-8B 2-port 10GE NIC

1x Intel 82598EB 2-port 10GE NIC

Donder:

Dell Precision T3400

2x Intel Core2 Duo E6550 2.33 GHz (4 cores total)

2GB DDR2 667 MHz

2x Myricom 10G-PCIE-8B 2-port 10GE NIC

1x Intel 82598EB 2-port 10GE NIC

C Script: Routing Configurations

```
`/home/gerrie/iproute/sbin/ip link set dev eth0 multipath off`
`/home/gerrie/iproute/sbin/ip link set dev eth1 multipath off`

echo "[I]Increasing␣Buffers"
#echo 'net.core.wmem_max=115343360' >> /etc/sysctl.conf
#echo 'net.core.rmem_max=115343360' >> /etc/sysctl.conf
#echo 'net.ipv4.tcp_rmem= 10240 87380 115343360' >> /etc/sysctl.conf
#echo 'net.ipv4.tcp_wmem= 10240 87380 115343360' >> /etc/sysctl.conf

echo "[I]Creating␣VLANs"
`vconfig add eth8 1001`
`vconfig add eth5 514`
`vconfig add eth5 1000`
`vconfig add eth2 2000`
`vconfig add eth8 520`
`vconfig add eth2 522`
`vconfig add eth8 516`
`vconfig add eth2 518`
echo "[I]Closing␣unused␣interfaces"
`ifconfig eth3 down`
echo "[I]Adding␣IPaddresses"
`ifconfig eth8.1001 192.168.11.1`
`ifconfig eth5.514 192.168.138.2`
`ifconfig eth5.1000 192.168.12.1`
`ifconfig eth2.2000 192.168.13.1`
`ifconfig eth8.520 192.168.21.1`
`ifconfig eth2.522 192.168.22.1`
`ifconfig eth8.516 192.168.23.1`
`ifconfig eth2.518 192.168.24.1`

echo "[I]Creating␣routing␣tables"
`ip rule add from 192.168.11.1 table 1`
`ip rule add from 192.168.138.2 table 2`
`ip rule add from 192.168.12.1 table 3`
`ip rule add from 192.168.13.1 table 4`
`ip rule add from 145.100.37.131 table 5`
`ip rule add from 192.168.21.1 table 6`
`ip rule add from 192.168.22.1 table 7`
`ip rule add from 192.168.23.1 table 8`
`ip rule add from 192.168.24.1 table 9`

echo "[I]Creating␣routes"
`ip route add 192.168.11.0/24 dev eth8.1001 scope link table 1`
`ip route add default via 192.168.11.1 dev eth8.1001 table 1`

`ip route add 192.168.138.0/24 dev eth5.514 scope link table 2`
`ip route add default via 192.168.138.2 dev eth5.514 table 2`

`ip route add 192.168.12.0/24 dev eth5.1000 scope link table 3`
`ip route add default via 192.168.12.1 dev eth5.1000 table 3`

```

```
`ip route add 192.168.13.0/24 dev eth2.2000 scope link table 4`
`ip route add default via 192.168.13.1 dev eth2.2000 table 4`

`ip route add 145.100.37.131/27 dev eth0 scope link table 5`
`ip route add default via 192.168.37.131 dev eth0 table 5`

`ip route add 192.168.21.0/24 dev eth8.520 scope link table 6`
`ip route add default via 192.168.21.1 dev eth8.520 table 6`

`ip route add 192.168.22.0/24 dev eth2.522 scope link table 7`
`ip route add default via 192.168.22.1 dev eth2.522 table 7`

`ip route add 192.168.23.0/24 dev eth8.516 scope link table 8`
`ip route add default via 192.168.23.1 dev eth8.516 table 8`

`ip route add 192.168.24.0/24 dev eth2.518 scope link table 9`
`ip route add default via 192.168.24.1 dev eth2.518 table 9`

`ip route add default scope global nexthop via 145.100.37.131 dev eth0`

a=`ip route`
echo "$a"
```

D Script: Interfaces Environments

```
#!/bin/bash

HOST=$1
ENV=$2
SPEED=$3
NUMBER=$4

 if [ "$HOST" == "blik" ]; then
        echo "[I]BLIKSEM"
        `ifconfig eth8.1001 down`
        `ifconfig eth8.516 down`
        `ifconfig eth8.520 down`
        `ifconfig eth5.1000 down`
        `ifconfig eth5.514 down`
        `ifconfig eth2.2000 down`
        `ifconfig eth2.518 down`
        `ifconfig eth2.522 down`
        sleep 2
        `ifconfig eth2 down`
        `ifconfig eth5 down`
        `ifconfig eth8 down`
        sleep 5
        if [ "$ENV" == "lan" ]; then
        echo "Enabling LAN interfaces"
        `ifconfig eth2 up`
        `ifconfig eth5 up`
        `ifconfig eth8 up`
        sleep 10
        `ifconfig eth8.1001 up`
        `ifconfig eth8.520 up`
        `ifconfig eth5.1000 up`
        #`ifconfig eth2.2000 up`
        `ifconfig eth2.522 up`
        `ifconfig eth8.516 down`
        `ifconfig eth5.514 down`
        `ifconfig eth2.518 down`
        sleep 10
                if [ "$SPEED" == "10" ]; then
                echo "Only Enabling 10GB interfaces"
                `ifconfig eth8.520 down`
                `ifconfig eth2.522 down`
                        if [ "$NUMBER" == "2" ]; then
                        echo "Only the real working two GB links"
                        `ifconfig eth2.2000 down`
                        sleep 3
                        `ifconfig eth2 down`
                        fi
                elif [ "$SPEED" == "1" ]; then
                echo "Only Enabling 1GB interfaces"
                `ifconfig eth8.1001 down`
                `ifconfig eth5.1000 down`
                `ifconfig eth2.2000 down`
                else
                echo "No Speed Specified"
                fi
        elif [ "$ENV" == "wan" ]; then
        echo "Enabling WAN interfaces"
        `ifconfig eth2 up`
        `ifconfig eth5 up`
        `ifconfig eth8 up`
        sleep 10
        `ifconfig eth8.516 up`
        `ifconfig eth5.514 up`
        `ifconfig eth2.518 up`
        `ifconfig eth8.1001 down`
        `ifconfig eth8.520 down`
        `ifconfig eth5.1000 down`
        `ifconfig eth2.2000 down`
        `ifconfig eth2.522 down`
                        if [ "$SPEED" == "2" ]; then
                        echo "Only the Geneve links"
                        `ifconfig eth5.514 down`
                        sleep 3
                        `ifconfig eth5 down`
                        fi
        elif [ "$ENV" == "all" ]; then
        echo "Enabling all interfaces"
        `ifconfig eth2 up`
        `ifconfig eth5 up`
        `ifconfig eth8 up`
        sleep 10
        `ifconfig eth8.1001 up`
        `ifconfig eth8.520 up`
        `ifconfig eth5.1000 up`
        #`ifconfig eth2.2000 up`
        `ifconfig eth2.522 up`
        `ifconfig eth8.516 up`
        `ifconfig eth5.514 up`
        `ifconfig eth2.518 up`
        sleep 10
                if [ "$SPEED" == "10" ]; then
                echo "Only Enabling 10GB interfaces"
                `ifconfig eth8.520 down`
                `ifconfig eth2.522 down`
                `ifconfig eth8.516 down`
                `ifconfig eth2.518 down`
                elif [ "$SPEED" == "1" ]; then
                echo "Only Enabling 1GB interfaces"
                `ifconfig eth8.1001 down`
                `ifconfig eth5.1000 down`
                `ifconfig eth2.2000 down`
                `ifconfig eth5.514 down`
                else
                echo "No Speed Specified"
                fi
        else
        echo "No Environment given"
        fi
elif [ "$HOST" == "don" ]; then
        echo "[I]DONDER"
        `ifconfig eth2.1000 down`
        `ifconfig eth2.515 down`
        `ifconfig eth3.1001 down`
        `ifconfig eth3.517 down`
        `ifconfig eth3.521 down`
        `ifconfig eth1.2000 down`
        `ifconfig eth1.519 down`
        `ifconfig eth1.523 down`
        `ifconfig eth1 down`
        `ifconfig eth2 down`
        `ifconfig eth3 down`
        if [ "$ENV" == "lan" ]; then
        echo "Enabling LAN interfaces"
        `ifconfig eth1 up`
        `ifconfig eth2 up`
        `ifconfig eth3 up`
        #`ifconfig eth1.2000 up`
        `ifconfig eth2.1000 up`
        `ifconfig eth3.1001 up`
        `ifconfig eth1.523 up`
        `ifconfig eth3.521 up`
        `ifconfig eth2.515 down`
        `ifconfig eth3.517 down`
        `ifconfig eth1.519 down`
                if [ "$SPEED" == "10" ]; then
                echo "Only Enabling 10GB interfaces"
                `ifconfig eth1.523 down`
                `ifconfig eth3.521 down`
                        if [ "$NUMBER" == "2" ]; then
                        echo "Only the real two GB links"
                        `ifconfig eth1.2000 down`
                        fi
                elif [ "$SPEED" == "1" ]; then
                echo "Only Enabling 1GB interfaces"
                `ifconfig eth2 down`
                `ifconfig eth1.2000 down`
                `ifconfig eth2.1000 down`
                `ifconfig eth3.1001 down`
                else
                echo "No Speed Specified"
                fi
        elif [ "$ENV" == "wan" ]; then
        echo "Enabling WAN interfaces"
        `ifconfig eth1 up`
        `ifconfig eth2 up`
        `ifconfig eth3 up`
        `ifconfig eth2.515 up`
        `ifconfig eth3.517 up`
        `ifconfig eth1.519 up`
        `ifconfig eth2.1000 down`
        `ifconfig eth3.1001 down`
        `ifconfig eth3.521 down`
        `ifconfig eth1.2000 down`
        `ifconfig eth1.523 down`
                        if [ "$SPEED" == "2" ]; then
                        echo "Only the Geneve links"
                        `ifconfig eth2.515 down`
                        sleep 3
                        `ifconfig eth2 down`
                        fi
        elif [ "$ENV" == "all" ]; then
        echo "Enabling all interfaces"
        `ifconfig eth1 up`
        `ifconfig eth2 up`
        `ifconfig eth3 up`
        `ifconfig eth2.1000 up`
        `ifconfig eth2.515 up`
        `ifconfig eth3.517 up`
        `ifconfig eth3.1001 up`
        `ifconfig eth3.521 up`
        #`ifconfig eth1.2000 up`
        `ifconfig eth1.519 up`
        `ifconfig eth1.523 up`

                if [ "$SPEED" == "10" ]; then
                echo "Only Enabling 10GB interfaces"
                `ifconfig eth3.517 down`
                `ifconfig eth3.521 down`
                `ifconfig eth1.519 down`
                `ifconfig eth1.523 down`
                elif [ "$SPEED" == "1" ]; then
                echo "Only Enabling 1GB interfaces"
                `ifconfig eth2.1000 down`
                `ifconfig eth2.515 down`
                `ifconfig eth3.1001 down`
                `ifconfig eth1.2000 down`
                else
                echo "No Speed Specified"
                fi
        else
        echo "No Environment given"
        fi
else
 echo "No Host machine specified"
exit
fi

echo "[I]Sleep for five seconds"
sleep 5
echo "`ifconfig -s -a | grep BMRU`"
```

E Script: Monitoring Interfaces

```

#!/bin/bash
if [ -z "$1" ]; then
        echo
        echo usage: $0 network-interface
        echo
        echo e.g. $0 Interface-name
        echo
        exit
fi

speed=1
INT1=$1
INT2=$2
INT3=$3
INT4=$4

if [ "$INT1" == "eth3.1001" ]; then
        HOST1=192.168.11.1
elif [ "$INT1" == "eth2.515" ]; then
        HOST1=192.168.138.2
elif [ "$INT1" == "eth2.1000" ]; then
        HOST1=192.168.12.1
elif [ "$INT1" == "eth1.2000" ]; then
        HOST1=192.168.13.1
elif [ "$INT1" == "eth3.517" ]; then
        HOST1=192.168.23.1
elif [ "$INT1" == "eth3.521" ]; then
        HOST1=192.168.21.1
elif [ "$INT1" == "eth1.519" ]; then
        HOST1=192.168.24.1
elif [ "$INT1" == "eth1.523" ]; then
        HOST1=192.168.22.1
elif [ "$INT1" == "eth2.1001" ]; then
        HOST1=192.168.11.2
elif [ "$INT1" == "eth5.514" ]; then
        HOST1=192.168.138.18
elif [ "$INT1" == "eth5.1000" ]; then
        HOST1=192.168.12.2
elif [ "$INT1" == "eth8.2000" ]; then
        HOST1=192.168.13.2
elif [ "$INT1" == "eth2.516" ]; then
        HOST1=192.168.23.2
elif [ "$INT1" == "eth2.520" ]; then
        HOST1=192.168.21.2
elif [ "$INT1" == "eth8.518" ]; then
        HOST1=192.168.24.2
elif [ "$INT1" == "eth8.522" ]; then
        HOST1=192.168.22.2
else
        echo "Interface one given: Unknown"
```

```
fi
if [ "$INT2" == "eth3.1001" ]; then
        HOST2=192.168.11.1
elif [ "$INT2" == "eth2.515" ]; then
        HOST2=192.168.138.2
elif [ "$INT2" == "eth2.1000" ]; then
        HOST2=192.168.12.1
elif [ "$INT2" == "eth1.2000" ]; then
        HOST2=192.168.13.1
elif [ "$INT2" == "eth3.517" ]; then
        HOST2=192.168.23.1
elif [ "$INT2" == "eth3.521" ]; then
        HOST2=192.168.21.1
elif [ "$INT2" == "eth1.519" ]; then
        HOST2=192.168.24.1
elif [ "$INT2" == "eth1.523" ]; then
        HOST2=192.168.22.1
elif [ "$INT2" == "eth2.1001" ]; then
        HOST2=192.168.11.2
elif [ "$INT2" == "eth5.514" ]; then
        HOST2=192.168.138.18
elif [ "$INT2" == "eth5.1000" ]; then
        HOST2=192.168.12.2
elif [ "$INT2" == "eth8.2000" ]; then
        HOST2=192.168.13.2
elif [ "$INT2" == "eth2.516" ]; then
        HOST2=192.168.23.2
elif [ "$INT2" == "eth2.520" ]; then
        HOST2=192.168.21.2
elif [ "$INT2" == "eth8.518" ]; then
        HOST2=192.168.24.2
elif [ "$INT2" == "eth8.522" ]; then
        HOST2=192.168.22.2
else
        echo "Interface␣two␣given:␣Unknown"
fi
if [ "$INT3" == "eth3.1001" ]; then
        HOST3=192.168.11.1
elif [ "$INT3" == "eth2.515" ]; then
        HOST3=192.168.138.2
elif [ "$INT3" == "eth2.1000" ]; then
        HOST3=192.168.12.1
elif [ "$INT3" == "eth1.2000" ]; then
        HOST3=192.168.13.1
elif [ "$INT3" == "eth3.517" ]; then
        HOST3=192.168.23.1
elif [ "$INT3" == "eth3.521" ]; then
        HOST3=192.168.21.1
elif [ "$INT3" == "eth1.519" ]; then
        HOST3=192.168.24.1
elif [ "$INT3" == "eth1.523" ]; then
        HOST3=192.168.22.1
elif [ "$INT3" == "eth2.1001" ]; then
        HOST3=192.168.11.2
```

```
elif [ "$INT3" == "eth5.514" ]; then
        HOST3=192.168.138.18
elif [ "$INT3" == "eth5.1000" ]; then
        HOST3=192.168.12.2
elif [ "$INT3" == "eth8.2000" ]; then
        HOST3=192.168.13.2
elif [ "$INT3" == "eth2.516" ]; then
        HOST3=192.168.23.2
elif [ "$INT3" == "eth2.520" ]; then
        HOST3=192.168.21.2
elif [ "$INT3" == "eth8.518" ]; then
        HOST3=192.168.24.2
elif [ "$INT3" == "eth8.522" ]; then
        HOST3=192.168.22.2
else
        echo "Interface three given: Unknown"
fi
if [ "$INT4" == "eth3.1001" ]; then
        HOST4=192.168.11.1
elif [ "$INT4" == "eth2.515" ]; then
        HOST4=192.168.138.2
elif [ "$INT4" == "eth2.1000" ]; then
        HOST4=192.168.12.1
elif [ "$INT4" == "eth1.2000" ]; then
        HOST4=192.168.13.1
elif [ "$INT4" == "eth3.517" ]; then
        HOST4=192.168.23.1
elif [ "$INT4" == "eth3.521" ]; then
        HOST4=192.168.21.1
elif [ "$INT4" == "eth1.519" ]; then
        HOST4=192.168.24.1
elif [ "$INT4" == "eth1.523" ]; then
        HOST4=192.168.22.1
elif [ "$INT4" == "eth2.1001" ]; then
        HOST4=192.168.11.2
elif [ "$INT4" == "eth5.514" ]; then
        HOST4=192.168.138.18
elif [ "$INT4" == "eth5.1000" ]; then
        HOST4=192.168.12.2
elif [ "$INT4" == "eth8.2000" ]; then
        HOST4=192.168.13.2
elif [ "$INT4" == "eth2.516" ]; then
        HOST4=192.168.23.2
elif [ "$INT4" == "eth2.520" ]; then
        HOST4=192.168.21.2
elif [ "$INT4" == "eth8.518" ]; then
        HOST4=192.168.24.2
elif [ "$INT4" == "eth8.522" ]; then
        HOST4=192.168.22.2
else
        echo "Interface four given: Unknown"
fi

while true
```

```
do

        R1=`cat /sys/class/net/$INT1/statistics/rx_bytes`
        T1=`cat /sys/class/net/$INT1/statistics/tx_bytes`
        if [ -z "$2" ]; then
        echo -n ""
        else
        R2=`cat /sys/class/net/$INT2/statistics/rx_bytes`
        T2=`cat /sys/class/net/$INT2/statistics/tx_bytes`
        fi
        if [ -z "$3" ]; then
        echo -n ""
        else
        R3=`cat /sys/class/net/$INT3/statistics/rx_bytes`
        T3=`cat /sys/class/net/$INT3/statistics/tx_bytes`
        fi
        if [ -z "$4" ]; then
        echo -n ""
        else
        R4=`cat /sys/class/net/$INT4/statistics/rx_bytes`
        T4=`cat /sys/class/net/$INT4/statistics/tx_bytes`
        fi

        sleep $speed

        R11=`cat /sys/class/net/$INT1/statistics/rx_bytes`
        T11=`cat /sys/class/net/$INT1/statistics/tx_bytes`
        if [ -z "$2" ]; then
        echo -n ""
        else
        R12=`cat /sys/class/net/$INT2/statistics/rx_bytes`
        T12=`cat /sys/class/net/$INT2/statistics/tx_bytes`
        fi
        if [ -z "$3" ]; then
        echo -n ""
        else
        R13=`cat /sys/class/net/$INT3/statistics/rx_bytes`
        T13=`cat /sys/class/net/$INT3/statistics/tx_bytes`
        fi
        if [ -z "$4" ]; then
        echo -n ""
        else
        R14=`cat /sys/class/net/$INT4/statistics/rx_bytes`
        T14=`cat /sys/class/net/$INT4/statistics/tx_bytes`
        fi

        echo `date`
        snelheid_T1=`expr $T11 - $T1`
        snelheid_R1=`expr $R11 - $R1`
        gem_T1=`expr $snelheid_T1 / $speed`
        gem_R1=`expr $snelheid_R1 / $speed`
        #KBPS_T1=`echo "scale=2; $gem_T1 / 1024" | bc`
        #KBPS_R1=`echo "scale=2; $gem_R1 / 1024" | bc`
        #MBPS_T1=`echo "scale=2; $KBPS_T1 / 1024" | bc`
        #MBPS_R1=`echo "scale=2; $KBPS_R1 / 1024" | bc`
        #TotKB_1=`echo "scale=2; $KBPS_T1 + $KBPS_R1" | bc`
        #TotMB_1=`echo "scale=2; $MBPS_T1 + $MBPS_R1" | bc`
        perf_1=`echo "scale=2; $gem_T1 / 131072" | bc`
        perf_11=`echo "scale=2; $gem_R1 / 131072" | bc`
        #perf_T1=`echo "scale=2; $TotMB_1 * 8" | bc`
        perf_T1=`echo "scale=2; $perf_1 + $perf_11" | bc`
#       rtt1=`ping -I $INT1 $HOST1 -c 1 -i 0.01 | cut -d "/" -f 5 | tail -n 1 | cut -f 1`
        #echo -e "$INT1= \t tx: $KBPS_T1 kb/s $MBPS_T1 mb/s \t rx: $KBPS_R1 kb/s $MBPS_R1 mb/s \t Totaal: $TotKB_1"
        echo -e "$INT1= \t iperf: tx $perf_1 rx $perf_11 tot $perf_T1 RTT: $rtt1"

        if [ -z "$2" ]; then
        echo -n ""
        else
        snelheid_T2=`expr $T12 - $T2`
        snelheid_R2=`expr $R12 - $R2`
                gem_T2=`expr $snelheid_T2 / $speed`
        gem_R2=`expr $snelheid_R2 / $speed`
        perf_2=`echo "scale=2; $gem_T2 / 131072" | bc`
        perf_22=`echo "scale=2; $gem_R2 / 131072" | bc`
        #perf_T2=`echo "scale=2; $TotMB_2 * 8" | bc`
        perf_T2=`echo "scale=2; $perf_2 + $perf_22" | bc`
        echo -e "$INT2= \t iperf: tx $perf_2 rx $perf_22 tot $perf_T2 RTT: $rtt2"
        fi

        if [ -z "$3" ]; then
        echo -n ""
        else
        snelheid_T3=`expr $T13 - $T3`
        snelheid_R3=`expr $R13 - $R3`
        gem_T3=`expr $snelheid_T3 / $speed`
        gem_R3=`expr $snelheid_R3 / $speed`
        perf_3=`echo "scale=2; $gem_T3 / 131072" | bc`
        perf_33=`echo "scale=2; $gem_R3 / 131072" | bc`
        perf_T3=`echo "scale=2; $perf_3 + $perf_33" | bc`
        echo -e "$INT3= \t iperf: tx $perf_3 rx $perf_33 tot $perf_T3 RTT: $rtt3"
        fi

        if [ -z "$4" ]; then
        echo -n ""
        else
        snelheid_T4=`expr $T14 - $T4`
        snelheid_R4=`expr $R14 - $R4`
        gem_T4=`expr $snelheid_T4 / $speed`
        gem_R4=`expr $snelheid_R4 / $speed`
        perf_4=`echo "scale=2; $gem_T4 / 131072" | bc`
        perf_44=`echo "scale=2; $gem_R4 / 131072" | bc`
        perf_T4=`echo "scale=2; $perf_4 + $perf_44" | bc`
        echo -e "$INT4= \t iperf: tx $perf_4 rx $perf_44 tot $perf_T4 RTT: $rtt4"
        fi
done
```

F Script: Experiment I

```

#!/bin/bash

ADDRESS=192.168.22.2
EXPERIMENT=14B3-ID6
UITLEZEN="eth8.520␣eth2.522"    #LAN 1
BUFFER=3MB
MSS=1400
RUNTIME=600

echo "Starten␣van␣experiment␣$EXPERIMENT"
echo "———————————————————————"

echo "[L]Local␣interfaces␣goed␣zetten"

sysctl -w net.mptcp.mptcp_mss=$MSS
sysctl -w net.mptcp.mptcp_checksum=0
#/home/gerrie/Desktop/interfaces blik lan 1
#/home/gerrie/Desktop/interfaces don lan

echo "[R]Remote␣routing␣goed␣zetten"

ssh root@145.100.37.132 sudo sysctl -w net.mptcp.mptcp_mss=$MSS
ssh root@145.100.37.132 sudo sysctl -w net.mptcp.mptcp_checksum=0
#ssh root@145.100.37.132 sudo /home/gerrie/Desktop/interfaces don lan 1

for a in {1,2,3,9,16,21}
do
date
echo "[R]Remote␣iperf␣Starten"

#ssh root@145.100.37.131 sudo killall -9 nuttcp
#ssh root@145.100.37.131 sudo killall -9 nuttcp
#ssh root@145.100.37.131 sudo nuttcp -S
#ssh root@145.100.37.131 sudo nuttcp -S -r -l3MB -n3MB #LAN 1

ssh root@145.100.37.132 sudo killall -9 iperf
ssh root@145.100.37.132 sudo killall -9 iperf
ssh root@145.100.37.132 sudo iperf -s -l 64KB -w $BUFFER -p 5001 -P $a &
#ssh root@145.100.37.132 sudo iperf -s -l 64KB -w $BUFFER -p 5002 -P $a &

#ssh root@145.100.37.132 sudo iperf -s &

sleep 5

echo "[L]Uitlezen␣starten"
killall -9 uitlezen-donder
/home/gerrie/Desktop/uitlezen-donder $UITLEZEN > experiment$EXPERIMENT.$a.txt &

sleep 5

```

```
echo "[L] Start Local Test"
#killall -9 nuttcp
#killall -9 nuttcp
 killall -9 iperf
 killall -9 iperf

echo "Parameters: [S] -t 600 -w $BUFFER -l 64KB -P $a [R] -s -w $BUFFER -l 64KB -P $a
     " >> experiment$EXPERIMENT.txt
echo -n "Thread $a:" >> experiment$EXPERIMENT.txt
#nuttcp -t -T1200 -N$a $ADDRESS >> experiment$EXPERIMENT.txt &    #LAN 1
#nuttcp -t -T30 -w200000 -ws200000 -wb200000 -l1100000000 -n100000000 -N10 $ADDRESS
#nuttcp -t -T30 -l1100000000 -n100000000 -N10 $ADDRESS
#nuttcp -T20 $ADDRESS

#iperf -i 5 -t $RUNTIME -l 500KB -c $ADDRESS >> experiment$EXPERIMENT.txt #LAN 1
iperf -i 1 -t $RUNTIME -l 64KB -w $BUFFER -p 5001 -P $a -c 192.168.22.3 >>
     experimentRAIN$EXPERIMENT.txt &
iperf -i 1 -t $RUNTIME -l 64KB -w $BUFFER -p 5001 -P $a -c $ADDRESS >>
     experiment$EXPERIMENT.txt &

sleep $RUNTIME

sleep 5

 killall -9 uitlezen-donder
INFO=`cat experiment$EXPERIMENT.txt | tail -n 1`
echo "[I] Experiment $EXPERIMENT: Thread $a finished"
echo "$INFO"
done
```

G Bibliography

[1] “Ams-ix traffic.” http://www.ams-ix.net/statistics/.

[2] L. van Wyk, “Players hard at work putting fibre infrastructure in place,” 2012. http://www.interconnectionworld.com/index/display/wire-news-display/1667926957.html/.

[3] T. Simonite, “The great bandwidth brawl,” 2012. http://mashable.com/2012/05/30/the-great-bandwidth-brawl/.

[4] “Trilogy project.” http://www.trilogy-project.org/.

[5] S. Barre, C. Paasch, and O. Bonaventure, “Multipath tcp: From theory to practice,” 2011. http://inl.info.ucl.ac.be/system/files/presentation_0.pdf.

[6] “Sara.” https://www.sara.nl/.

[7] “Ietf workgroup.” http://datatracker.ietf.org/wg/mptcp/charter/.

[8] “Ucl-london.” http://nrg.cs.ucl.ac.uk/mptcp/.

[9] “Inl.” http://mptcp.info.ucl.ac.be/pmwiki.php?n=Main.HomePage.

[10] C. Raiciu and C. Paasch, “Multipath tcp,” 2012. https://docs.google.com/presentation/d/1KR2UuoODrkGuzeAYiTmR6j4NKD7vmOnWLhAdNX1Y1D4/edit?pli=1#slide=id.p16.

[11] C. Raiciu, D. Wischik, A. Greenhalgh, and M. Handley, “Design, implementation and evaluation of congestion control for multipath tcp,” 2011. http://dl.acm.org/citation.cfm?id=1972457.1972468.

[12] D. Wischik, C. Raiciu, and M. Handley, “Rfc 6356: Coupled congestion control for multipath transport protocols,” 2011. http://www.rfc-editor.org/rfc/pdfrfc/rfc6356.txt.pdf.

[13] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “Tcp extensions for multipath operation with multiple addresses,” 2012. http://tools.ietf.org/pdf/draft-ietf-mptcp-multiaddressed-09.pdf.

[14] R. van der Pol, S. Boele, F. Dijkstra, A. Barczyky, G. van Malensteinz, J. H. Chenx, and J. Mambretti, “Multipathing with mptcp and openflow,” 2012.

[15] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, “How hard can it be? designing and implementing a deployable multipath tcp,” 2012. http://inl.info.ucl.ac.be/system/files/nsdi12-final125.pdf.

[16] C. Raiciu and C. Paasch, “Multipath tcp,” 2012. http://www.youtube.com/watch?v=02nBaaIoFWU.

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," 2011. http://www.youtube.com/watch?v=iqx1cShqH6I.

[18] V. Gite, "Linux tune network stack (buffers size) to increase networking performance," 2009. http://www.cyberciti.biz/faq/linux-tcp-tuning/.