# Testing the performance of Multipath TCP for connections with limited bandwidth

Network performance analysis

Project Report

Group: 1020

Aalborg University
Electronics and IT

AALBORG UNIVERSITY
STUDENT REPORT

**Electronics and IT**
Aalborg University
http://www.aau.dk

**Title:**
Testing the performance of Multipath TCP for connections with limited bandwidth

**Theme:**
Network performance analysis

**Project Period:**
Spring Semester 2016

**Project Group:**
1020

**Participant(s):**
Kim Ellegaard Jacobsen

**Supervisor(s):**
Tatiana Kozlova Madsen
Karen Elisabeth Egede Nielsen

**Copies:** 2

**Page Numbers:** 113

**Date of Completion:**
June 1, 2016

**Abstract:**

This project analyzes the performance of the protocol called MPTCP. It tries to determine if the protocol is a viable candidate to be deployed in a rural area of Denmark, where the Internet connections are of poor quality with slow throughput, and can aggregate the bandwidth from a DSL connection and a 4G connecting and still be able to be used for interactive streaming purposes. It uses a testbed setup specifically for this project that consists of a Server running Ubuntu 14.04, a gateway that creates 2 emulated DSL connections of 2048 Kbps downstream and 2048 Kbps upstream and a client that uses two interfaces to aggregate the bandwidth. Both the server and the client are running a Linux implementation of the Multipath TCP kernel from [13]. The project concludes that MPTCP is a viable candidate, for deployment in a similar scenario as described in this project, but that further testing is needed in order to be able to fully conclude, if the network connection is suited for running interactive streaming applications, when using MPTCP with heterogeneous paths.

# Preface

This report has been written by a student on Network and Distributed Systems $10^{th}$ semester project. The project was proposed by Tatiana Kozlova Madsen. The purpose of the project was to test bandwidth aggregation of the Multipath TCP when dealing with unreliable connections.

Thanks to my supervisor Tatiana Kozlova Madsen for the time she has invested in guiding this project and providing valuable feedback.
I would also like to thank Karen Elisabeth Egede Nielsen from Tieto for all the feedback and knowledge supplied for this project.

Aalborg University, June 1, 2016

Kim Ellegaard Jacobsen
<kjacob11@student.aau.dk>

# Contents

# Glossary

| | |
|---|---|
| DSL | Digital Subscriber Line. |
| IETF | Internet Engineering Task Force. |
| MPTCP | Multipath Transmission Control Protocol. |
| RTT | Round Trip Time. |
| TCP | Transmission Control Protocol. |

# Chapter 1

# Introduction

When the Transmission Control Protocol (TCP) [27] was first designed in 1974 as a part of the Internet Protocol Suite, the idea revolved around a single device with a single network connection. Since then the times have changed to supply different demands. In 2014 64% of the American adult population [4] owned a smartphone. A smartphone typically supports both a 2G/3G/4G connection as well as a Wi-Fi connection. Even though smartphones have been around for a series of years, there is a very limited usage of a protocol that utilizes the multiple connections, available on the smartphone, to either aggregate bandwidth or as a reliability tool. A typically smartphone works by using either the 2G/3G/4G connection or the Wi-Fi connection [15] and then it uses different techniques for the handover or handoff part. None of the most used smartphone operating systems uses multiple connections simultaneously even though the technology already exists. Seen from the users point of view, it might be beneficial sometimes to use all of the available connections, e.g. when downloading a large file, to decrease the download time.

In a world where network technology is constantly evolving and the requirements for bandwidth, latency and reliability is increasing, it could be beneficial to start using a multipath approach in both smartphones as well as other scenarios where multiple network connections are available. It would make a lot of sense to use it in areas where network connectivity is almost non existent. When looking at a country like Denmark, the network connectivity in the far side of the country is poor or non existent. Typically in these regions the available types of Internet connection will be a DSL connection and a 2G/3G/4G connection. An easy way to ensure greater bandwidth in these regions, would be to aggregate the bandwidth of multiple connections and this can be done by using a multipath approach on the transport

layer e.g. the Multipath TCP (MPTCP).

## 1.1 Motivation scenario

The motivation behind doing this project lies in the rural areas of Denmark, where the coverage on 2G and 3G networks are poor and the DSL connections are slow. This gives some problems for e.g. farmers using their network, to conduct interactive streaming conversations regarding the maintenance and support of the robots they use to produce milk. This has proved to be a somewhat hard task, since the network connectivity in the rural areas often are slow. So instead of using a lot of money on new network technologies such as fiber optics, which is very expensive, because of the distance between the farms can be great. Then it would make more sense to combine already existing technologies for bandwidth aggregation. This can be done in several different ways, and there is already a research project running on Aalborg University, which are trying to deploy software that runs on the application layer and splits the TCP packet stream on multiple network connections in order to get some bandwidth aggregation. However there is already a protocol called Multipath TCP, which can do this on the transport layer and that makes the operation transparent to the application. This project will be about deploying the MPTCP in a scenario similar to what was discussed earlier in this paragraph.

## 1.2 History of Multipath TCP

Currently only a limited usage of the MPTCP implementation can be found. In 2013 the working group behind the MPTCP reported that 5 independent implementations of the MPTCP was made. These were:

**List 1.1:** List of different implementations of MPTCP [24]

1. Linux kernel (reference implementation) from Université catholique de Louvain.
2. Android from Université catholique de Louvain.
3. FreeBSD (IPv4 only) from Swinburne University of Technology.
4. F5 Networks BIG-IP LTM.
5. Citrix Netscaler.
6. Apple iOS 7, released on September 18, 2013 is the first large scale commercial deployment of Multipath TCP.
7. Apple Mac OS X 10.10, released on October 16, 2014.

As seen in the list above there has been made implementations of the MPTCP in both Android and iOS but it is not part of the default setup in either of the mobile operating systems. The most interesting implementation for this project scope will be the Linux kernel implementation made by Université catholique de Louvain.

### 1.2.1 Linux kernel implementation

The Linux kernel implementation of MPTCP is at version 0.90 released 16. September 2015. It has a wide supported range of platforms already including:

**List 1.2:** List of platforms that support the MPTCP Linux kernel implementation [13]

1. Compile it by source
2. Install it from an apt-repository
3. Linux distributions
    - Debian
    - Ubuntu 12.10 and upwards
    - Fedora 19 to 21
    - CentOS 7
    - Gentoo
    - ArchLinux
    - OpenSUSE
4. OpenWRT
5. Android
6. PlanetLab
7. Amazon EC2

As seen in the list above, the MPTCP is already developed and usable for a wide range of different platforms. This list is however only representative of the Linux kernel implementation and other systems such as OSX and iOS also have a version of the protocol that is usable.
The Linux kernel implementation is made from the standard called RFC 6824 [6]. It describes how the protocol works in depth and it is maintained by the Internet Engineering Task Force (IETF). The standard was released in January 2013.

## 1.3 Problem statement

If we consider the farmer scenario, where Internet connections in the rural areas of Denmark are both limited and of very poor quality, a cheap and very likely approach can be a multipath solution as Multipath TCP. So the motivation for using MPTCP in order to fulfill the problem description can be broken into three different parts,

namely:

- MPTCP adds redundancy e.g. if one link fails, the connection will stay active.
- MPTCP reduces congestion. MPTCP makes it possible to steer traffic away from congested links.
- MPTCP can increase efficiency. The protocol can take advantage of additional interfaces e.g. parallel paths.

So this leads us to a problem description which is stated below:

- **Is it possible to setup a multipath scenario using the MPTCP, which consists of multiple interfaces with different characteristics (e.g. 4G and DSL), such that it aggregates the bandwidth and is able to support interactive streaming?**

In this report, the author will analyze the different aspects of a multipath scenario and the technology behind the protocol described in the IETF standard called RFC6824 [6] known as MPTCP. It will also focus on deploying this setup either in a testbed, or in a rural area of Denmark to test if the performance and stability of a multipath setup can be a solution for the scenario description given in section 1.1.

Following this text a short description of each chapter is given, with respect to its content.

- Chapter 2 gives an in depth analysis of the Multipath TCP and the technology behind it. It will also contain a refined problem description.
- Chapter 3 will give an overview of the scenario.
- Chapter 4 will cover the implementation, testbed setup and the test descriptions.
- Chapter 5 will show the results for each test as well as a short analysis and walk through of the results.
- Chapter 6 will be the conclusion and will include future aspects of Multipath TCP.

# Chapter 2

# Analysis

In this chapter the MPTCP will be examined and the key features of the protocol will be explained and described.

## 2.1 Motivation for Multipath TCP

In the 1970s when the TCP was initially designed, the typical setup for a computer was a single device with a single network connection. However when the TCP was designed, it was still an issue that network links could eventually fail and that some sort of feature to decouple the layers would need to be designed. This ended up being split into 2 layers, namely the network-layer and the transport-layer. That is why the protocol today is known as TCP/IP.
Today networks are considered multipath network. Just look at a typical smartphone, which has multiple wireless interfaces. Another example are large data centers, which also have multipath networks within the centers themselves. This can present a problem, when using a protocol as TCP, which is essentially a single-path protocol. The way that TCP works is by binding a connection to a specific IP-address. When the connections is broken it cannot migrate the connection to another interface on the transport layer. There might be some applications that can support migrating a network connection to another interface, but it would require some special software to run on the devices in question.
If TCP would be used for load balancing on multiple paths within some network, it would still be a problem to get the data reordered at the received end, and still maintain a high network throughput. In this case the reordering would be interpreted

as congestion and would eventually slow down the performance.

Today most of the people in western countries own a smartphone with multiple interfaces, but they are still unable migrate TCP connections from one interface as Wi-Fi to an interface with a 3G or 4G connection without the connection stalls or becomes unresponsive for a period of time. This is because the operating system has to detect that an interface has lost connection and then switch to another interface. A way to address these issues is to use Multipath TCP, which is a modification of the normal TCP. It will allow multiple paths to be used simultaneously on a single transport connection. The idea of a multipath approach is not new and was originally proposed by a guy named Christian Huitema in IETF more than 15 years ago. Since then multiple different projects have tried to solve this problem and the current MPTCP standard by IETF draws on the experience from these past projects.

## 2.2 MPTCP Overview

In the design of MPTCP two key requirements are worth mentioning:

- **Application compability:** The applications that already use TCP should be able to use MPTCP with no change.
- **Network compability:** The MPTCP should be able to operate on all Internet paths that already use TCP.

Since most of todays Internet communication include devices such as Network Address Translators, firewalls and other transparent proxies, these types of devices are aware of the TCP connections they handle and they affect these connections in different ways. This is different in plain IP routers, because they will not affect these TCP connections. So one of the biggest challenges of designing the MPTCP, has been to design a protocol, which can traverse these middleboxes safely.

Before the MPTCP will be explained a short overview of how regular TCP works will be given.

### 2.2.1 Regular TCP operation

A basic TCP operation can typically be split into three phases:

- Connection establishment
- Data transfer
- Connection release

### Connection establishment

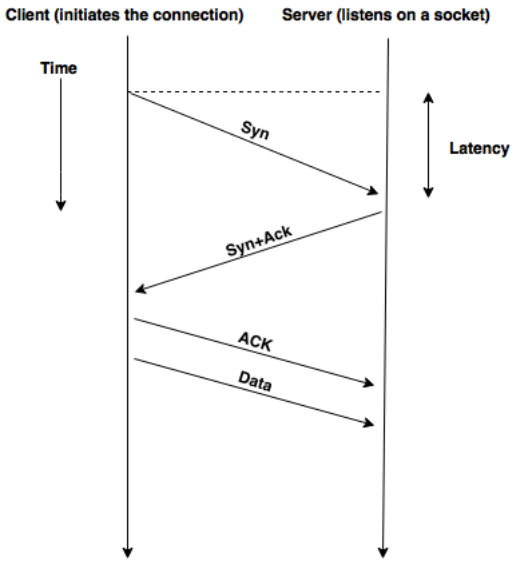


**Figure 2.1:** Overview of a TCP connection and the 3-way handshake[12].

Figure 2.1 shows how a 3-way handshake works for a TCP connection. The client initiates the connection by sending a synchronize (SYN) packet to the port on which the server is listening. The SYN packet includes the source port (the port to which the server answers) and the initial sequence number, which is chosen by the client. It can also include options that are used to negotiate the use of TCP extensions. The server then replies with a SYN+acknowledgment (ACK) packet. This packet includes the serves initial sequence number as well as the options that the server supports. Then the client sends back an ACK packet to the server and the connection is established. All subsequent packets in the connection uses the information gained in the 3-way handshake, namely the IP-addresses and the ports of the client and the server respectfully.

#### Data transfer

When the connection is established the client and the server communicates the data in what is known as segments. The sequence number is used to order the data in the different segments and also used to reorder them, as well as detect losses in the data. Included in the TCP header is a cumulative ACK, which is essentially a number that is used to acknowledge the received data, by informing the sender, what the next byte that is received is expected to be. If data is corrupted or lost there are different techniques used by TCP to retransmit. These will not be explained here because they are not important in understanding how the MPTCP works.

#### Connection release

When the client and the server are done communicating the TCP connection must be closed. This can be done in different ways. The communication can be abruptly closed if either the client or the server sends a Reset (RST) packet. This is not the correct way of closing the connection. If a connection is to be terminated, the correct way to do this is that, the client must send a Finish (FIN) packet to the server. The FIN packets include the sequence numbers of the last transfered byte. The connection is then terminated when the FIN packet has been acknowledged in both directions.

### 2.2.2 Multipath TCP Operation

The following section makes use of [18] as source.

When using MPTCP it will allow multiple subflows to be set up for a single MPTCP session. This works by by setting up an initial subflow for the MPTCP session and this subflow works similar to a regular TCP connection, which is described in subsection 2.2.1. When an initial subflow has been set up for the MPTCP session it is then possible to add additional subflows to the MPTCP session. Each of these subflows will be setup complete with a SYN handshake and it also uses a FIN packet, to mark that the data transfer has been completed. But unlike TCP, the subflows are all added into the same MPTCP session and a data transfer can happen over any of the subflows, given that the subflow has the capacity to handle it.

Before an in depth overview of the MPTCP will be given, a simple scenario where

MPTCP is highly relevant will be explained. A normal smartphone typically has two interfaces namely a 3G/4G connection and a Wi-Fi connection each of which has an IP-address of its own. The user of the smartphone wants to connect to a server with only a single interface. In this scenario, the MPTCP would allow an application to connect to to the server with a single TCP connection from both interfaces. In this case the application does not need to concern it self with what radio interface the smartphone uses to connect to the server. This is all handled by the MPTCP and in a similar scenario, where the server has multiple interfaces the MPTCP would then create subflows between all the interfaces. So if the server has two interfaces and the same goes for the smartphone four different subflows would be created.

When initiating a MPTCP connection, and the device being used it still a smartphone, the 3G interface is used to initiate the connection and it sends a SYN packet to the server. This SYN packet includes the MP_CAPABLE TCP option. This option is to indicate that the smartphone supports MPTCP. Also contained in the SYN packet is a key chosen by the device. The server then replies with a SYN+ACK packet including the MP_CAPABLE option and a key chosen by the server. The smartphone then acknowledges the SYN+ACK by replying with an ACK packet and the MPTCP session is started. Now the smartphone can send data via TCP segments on the 3G path.

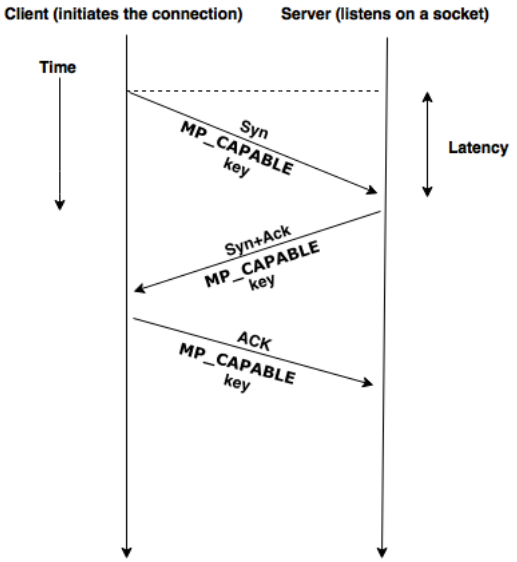


**Figure 2.2:** Overview of a MPTCP connection and the 3-way handshake.

In order for the smartphone to be able to also send TCP segments to the MPTCP

session from its Wi-Fi interface it would have to initiate a new subflow for this to work. By simply sending TCP segments via the Wi-Fi interface to the server, it would cause the TCP segments to be dropped by the Internet Service Provider (ISP), because the source address of the TCP segments would contain the IP-address of the 3G interface. If the smartphone would try to tell the server, that the packets actually belong to the Wi-Fi interface and not the 3G interface, it would cause the packets to be dropped by middleboxes since it is expected that a connection starts with a SYN packet and not data packets.
In order to be able to send data from both the Wi-Fi interface and the 3G interface the MPTCP would have to start a complete handshake for the Wi-Fi connection. This works by sending a SYN packet to the server, which contains the MP_JOIN TCP option. This packet contains all the information such that the server can distinguish the 3G path from the Wi-Fi path. The server responds with a SYN+ACK packet containing the MP_JOIN option and then the smartphone acknowledges. The new subflow is then added to the MPTCP session.

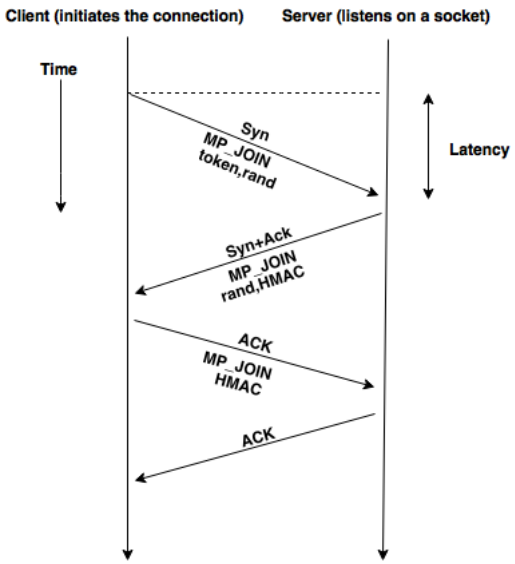


**Figure 2.3:** Overview of a MPTCP connection when adding a new suflow to the session.

An important thing to notice about MPTCP is that subflow can be added and removed though the life span of the MPTCP session. This will not affect the applications using the byte stream. MPTCP also supports adding and removing addresses, which also works when an end-point is located behind a NAT. Consider that the smartphone switches to another Wi-Fi network in the neighborhood. This will give the interface a new IP-address and the MPTCP simply handles this by creating a

new subflow with the informations from the new Wi-Fi network.

### 2.2.3 Paths with different characteristics

Assume that the smartphone has established two subflows, one from the Wi-Fi interface and one from the 3G interface. The smartphone will now be able to send and receive data from both interfaces, but because these two paths will have different characteristics, namely latency, this will cause the data packet to be received out of order. In a regular TCP connection a sequence number is used. The easy solution would be to reuse this sequence number in the subflows as well, but this will cause the middleboxes (e.g. firewalls) to either drop the packets or to try and recover packets by by updating the TCP acknowledgments. This is because they see gaps in the data streams and have no knowledge that the packets are part of a MPTCP session.
In order to fix this issue MPTCP uses its own sequence numbering space. Each segment that is sent through the MPTCP will have two individual sequence numbers. The first being the subflow sequence number inside of the regular TCP header and the other sequence number is a data sequence number stored inside a TCP option. This fix makes sure that each subflow has consecutive sequence numbers and this will not give any problems when traversing different types of middle boxes. The second sequence number, the data sequence number, is simply ignored by older middleboxes and it will then be used by the MPTCP at the received end to reconstruct the data stream, before it is sent to the application that requested it.

### 2.2.4 Congestion control in MPTCP

In a regular TCP connection the congestion controller is one of the most important components of the protocol. This controller makes sure that TCP can adapt its throughput dynamically. This is especially important when communicating via a networks with changing conditions, such as delay and packet loss and congestion. The way it works for TCP is that each sender maintains a congestion window, that describes how may packets the sender can send, without waiting for an acknowledgment. This congestion window is updated dynamically and when there is no congestion, it grows linearly and when a packet loss occurs, the congestion window is then divided by two. If multiple streams are running over the same link, the TCP congestion controller will ensure fairness and the data rate will converge to average for all of the streams.

In order to understand how the MPTCP handles congestion three different goals are defined:

- **First goal:** The congestion controller must ensure fairness to TCP. If several different active subflows of a MPTCP session are running alongside a regular TCP connection, and the link is a bottleneck, the MPTCP should not be able to get more throughput than the TCP connection.
- **Second goal:** The performance of all the active subflows of a MPTCP session should at least be able to obtain the same throughput as a regular TCP connection. This is to ensure that there is incentive to deploy a multipath solution in the first place.
- **Third goal:** When using a MPTCP session with multiple active subflows, the paths used to send data should be the most efficient paths namely the paths that are experiencing less congestion than the others.This third and last goal of the MPTCP congestion controller would ensure load balancing of traffic. When a MPTCP session is used the traffic will be pushed thorough the links, which has the lowest congestion and this will then decrease the amount of traffic on the congested links and increase the traffic on the non congested links. This will in effect cause links to act together as a single larger capacity link that will consist of all the subflows.

The MPTCP achieves these three goals by making simple changes to the congestion controller used by regular TCP connections. In MPTCP each subflow will have its own congestion controller that works exactly as a regular TCP congestion controller. Furthermore when using resource pooling in MPTCP, the congestion controller will allow less congested subflows to increase proportionally compared to the ones that are congested. The increase of the MPTCP across all of its subflows is chosen to be dynamically, and furthermore it is chosen in a way that fulfills goals one and two, as described above.

## 2.3 State-of-the-art

When talking about Multipath TCP, there are a lot of applications for a protocol that can connect multiple interfaces across different technologies. An example would be an Ethernet connection combined with a Wifi or 4G connection, where Multipath TCP is then able to maximize resource usage. This can both be utilized in the form

of bandwidth aggregation, as this project will focus on, or on creating a reliable setup of a network connection, where using the multiple subflows as redundant connections.

To sum up what the bandwidth aggregation is, it aims at maximizing the resource usage of multiple TCP connections. This can be done by evenly splitting the traffic on all the available paths disregarding the characteristics of these paths. If bandwidth aggregation is the only goal of using Multipath TCP, the performance of the network connection will only depend on the throughput. If there is delay or packet loss on some of the paths, this will just translate into a lower throughput at the receiver end. As stated before, if the Multipath TCP is sensitive to latency then head-of-line blocking can become a problem.
To sum up what the reliability aspect of the Multipath TCP is about, it is about the usage of the different available paths, to secure that the network connection is reliable. An example can be a smartphone setup, where there are two available interfaces, namely the WiFi interface and the 4G interface. In this case it would be able to primarily use the interface decided by the user, but it will be able to keep the other interface added as a backup network connection. This backup connection can be used, if the primary interface suddenly loses the network connection e.g. 4G outage. Then the Multipath TCP will be able to handle a handover of the data stream from one interface to the backup interface, without the need for notifying the layers above the transport layer.

In 2013 the Multipath TCP was published by the Internet Engineering Task Force as a experimental standard in RFC 6824 [24] [9]. As of today the Multipath TCP is still not a final standard, so the implementations of the protocol are still limited and the ones that are available has a limited functionality, since there are still details and functions in the protocol, that are not yet done.
The current implementations, as listed in [13], count multiple Linux distributions, Android, PlanetLab and Amazon EC2. Apple has also used Multipath TCP in iOS7 [2]. It is primarily used in their search application called Siri.

Since the experimental standard was released in 2013, there have been a lot of different research about how Multipath TCP should perform. There have been a lot of focus on different congestion control algorithms as well as the scheduler used in Multipath TCP. Both of these functions are crucial for the performance of the Multipath TCP. When looking at different congestion control algorithms for Multipath TCP, they can have different goals. These goals can be to ensure fairness in congestion control when compared to a regular TCP connection. Another goal can be to ensure that the congestion control is able to adapt quickly to network changes. In [8] they do an extensive analysis of the performance of the Multipath TCP. It

focuses on the different congestion control algorithms that are currently available for the Multipath TCP. These congestion control algorithms are Alias Linked Increase Congestion Control (LIA), Oppertunistic Alias Linked Increase Congestion Control (OLIA), Balanced Linked Adaptation Congestion Control (BALIA) and Delay-Based Congestion Control (wVegas). Their conclusion is that MPTCP outperforms normal TCP and has many advantages in different scenarios.
In [5] they do an extensive analysis of the schedulers in Multipath TCP and how these should work. They do an in-depth analysis of the performance of different schedulers namely Round-Robin (RR), Lowest-RTT-First (LowRTT), Retransmission and Penalization (RP) and Bufferbloa Mitigation (BM). Only the first two, RR and LowRTT, are available in the Linux implementation of MPTCP. They conclude that the scheduling in Multipath TCP shoud ideally be done in a way, so the the data arrives in-order at the receiver. This will help minimize the chance of head-of-line blocking as well as any receive window limitations, since the application that receives the data stream, is able to continuously read data from the receive queue. They also state that it would be possible to design such a scheduler, by using estimates of the round trip times (RTT) and the current capacity of the network connection, as they can be maintained by e.g. the Linux kernel.

The current limitations of the Linux implementation of MPTCP is, that the only two available schedulers are Round-Robin and Lowest-RTT-First. The Round-Robin implementation is only an experimental scheduler and it is stated on the Multipath TCP website, that this should only be used for testing purposes. The LowRTT is the default scheduler and since RR is experimental it limits the possible usages of the implementation.

When talking about different usage scenarios there are a few that are more suited than other.
As stated earlier one of the applications of the Multipath TCP would be in a smartphone with two available active network connections as WiFi and 4G. In this scenario the Multipath TCP could both be used for bandwidth aggregation as well as to ensure reliability e.g. when switching from one wireless network to another.
Another scenario where Multipath TCP could be beneficial, is a setup where there are networks of limited bandwidth available. This could be in ares with poor quality network connections e.g. the rural areas of Denmark. Here it could be beneficial to be able to combine the bandwidth from several different available network connection e.g. DSL and 4G. This scenario is also the motivation behind this report.
A third scenario could be the several different network paths inside of large data center. Here it could be beneficial to aggregate multiple paths to be able to utilize the network more effecient and to be able to move data from one server to another

server a lot faster. On [13] it is stated that Multipath TCP was used in an experiment where a bandwidth of 51.8 Gbps was gained. This kind of setup would be beneficial in a data center.

The work done in this project will focus more on the performance of Multipath TCP, when using it for interactive streaming purposes. Since the scenario states that, in the rural areas of Denmark, the farmers are having trouble with not having enough available bandwidth for carrying out support related interactive streaming conversations. This project uses a Multipath TCP testbed to gain results. These results will either show if a Multipath TCP setup will be able to solve the bandwidth limitation problems or not.
The current scenario is that, in these rural areas of Denmark, a maximum bandwidth on a DSL connection is around 2 Mbps downstream and 512 Kbps upstream. This bandwidth is not enough to carry out an interactive streaming support session and therefore it will be examined, if deploying a Multipath TCP setup can be used as a solution to gain more bandwidth. In the current scenario the setup of Multipath TCP will be done using a emulated DSL connection combined with a real 4G connection.

# Chapter 3

# Scenario description

In this chapter an extended overview of the scenario will be given.

As described in the section 1.1, the motivation for using Multipath TCP as a resource pooling of multiple connections, is because of the poor Internet connectivity in the rural areas of Denmark. It all comes down to the connection speed of the Digital Subscriber Line (DSL) that can be provided in these areas. The information provided for this project is that it is only possible to get a DSL line with 2 Mbps downstream and 512 Kbps upstream. This provides a problem for the farmers, that wants to use this DSL connection to do interactive streaming to consultants, regarding the technical setup and support of the robots they use for e.g. milking their cows. With a 2 Mbps / 512 Kbps line there is simply not enough bandwidth to carry out an interactive streaming session with the software, that the consultants use to carry out support or setup of the robots/machines.
This is the motivation to investigate, if resource pooling using the Multipath TCP, will allow enough bandwidth to carry out interactive streaming sessions with the consultants.

Typically when talking about interactive streaming, it has some strict requirements for one way delay, jitter and packet loss. In [11] they talk about VoIP and the requirements to a voice conversation. In [11] they have that the one way delay can not exceed 150 ms, since this would cause noticeable changes in the voice stream. They also say that the delay variances can not exceed 100 ms since the buffers, that are implemented to handle difference in delay, will not be able to correct this behavior from a buffer point of view.
In [25] they recommend a network connection with at least 2 Mbps available band-

width in order to watch streaming media. This is only one way bandwidth and if the traffic type is interactive streaming a network connection with 2 Mbps downstream and upstream is needed. It all depends on the codec that are used to encode the video stream and the voice stream, but having an available bandwidth lower than 2 Mbps is not recommended.

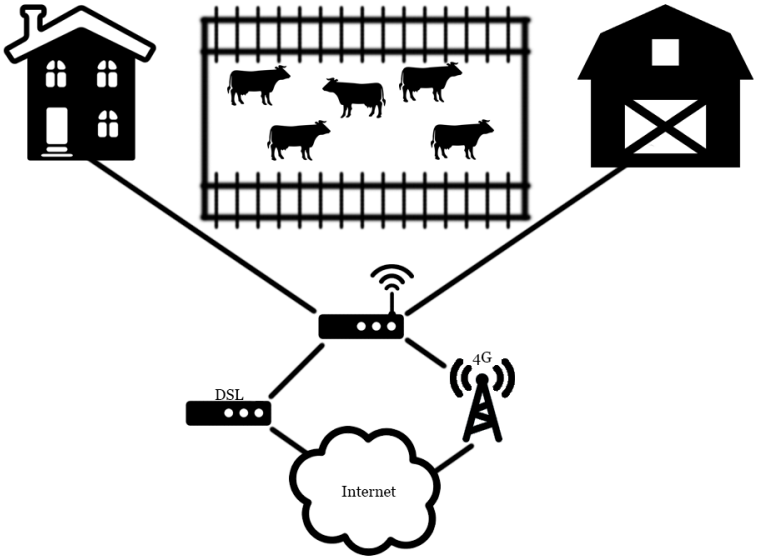


**Figure 3.1:** This figure shows the scenario overview.

**Proxy** The setup shown in Figure 5.1 gives a good indication of what the real life application of the multipath protocol can be. Figure 5.1 shows, on a high level, what kind of setup that is deployed at the farmer and how a proxy setup with a multipath tcp enabled proxy, could help the farmer in terms of bandwidth aggregation and possibly also reliability. This project and the testbed that has been built will only focus on the bandwidth aggregation part, since the current problem for the farmers is that the DSL line they have available, is too slow to handle interactive streaming conversations with the software that they use.
As seen in Figure 5.1, the current usage of multipath TCP would be to setup a proxy at the location of the farmer. This proxy could be either some router capable of running an implementation of the multipath TCP, or a simple gateway that runs the Linux implementation. The idea behind using a proxy for the multipath TCP, is that it is only supported if both end points has MPTCP enabled. So if the farmer wants to use the advantages of the multipath TCP in e.g. his barn or his house he

would just use his normal devices without any modifications. The proxy in this case will be the end point for the MPTCP and the incoming Internet will then get shared to a different interface. This interface is then connected to a router and this router then shares the Internet to the devices at the farmer.

# Chapter 4

# Implementation

## 4.1 Testbed setup

This section will take a closer look on how to get the Multipath TCP kernel up and running, how to configure the routing on the client computer and how to setup a gateway to throttle the network connection such that it emulates a DSL connection.

### 4.1.1 Installing the Multipath TCP kernel

In order to be able to fulfill the scenario described in the section 1.1 some hardware are needed in order to setup a testbed to reproduce the scenario, where the network connection in rural areas of Denmark has access to poor Internet connections such as 2 Mbps downstream and 512 Kbps upstream DSL connection and a variety of 3G and 4G connections. The hardware chosen to build the testbed will be:

- 1 computer running Ubuntu 14.04 LTS 64bit with 2 network cards to function as the client/farmer.
- 1 computer running Ubuntu 14.04 LTS 64bit with a single network card to function as the server or host.
- 1 computer running Ubuntu 14.04 LTS 64bit with 3 network cards.This will function as a gateway and be used to emulate the DSL connection to the client.

The two computers used for both the client and the server will both be running a custom prebuilt kernel running linux 3.18.* and the implementation of the the Multipath TCP[13]. The kernel comes prebuilt and it is installed by getting it directly from the Multipath TCP Apt Repository. Below the commands needed for installing the kernel are shown:

```
1 wget -q -O - http://multipath-tcp.org/mptcp.gpg.key | sudo apt-key add - //The gpg key is added as a
      trusted source
```

The file mptcp.list is created in /etc/apt/sources.list.d/. It will hold this line:

- deb http://multipath-tcp.org/repos/apt/debian trusty main

The the kernel is installed by issuing the following command:

```
1 sudo apt-get update
2 sudo apt-get install linux-mptcp
```

When the process is done, one last step needs to be completed in order to be able to boot the Multipath TCP kernel. The grub bootloader needs to be edited to boot the mptcp kernel. This is done by installing grub-customizer with the apt-get command:

```
1 sudo apt-get install grub-customizer //installs the program
2 sudo grub-customizer //runs the program
```

The the mptcp enabled kernel is moved to the top by using the arrows as shown in Figure 4.1.

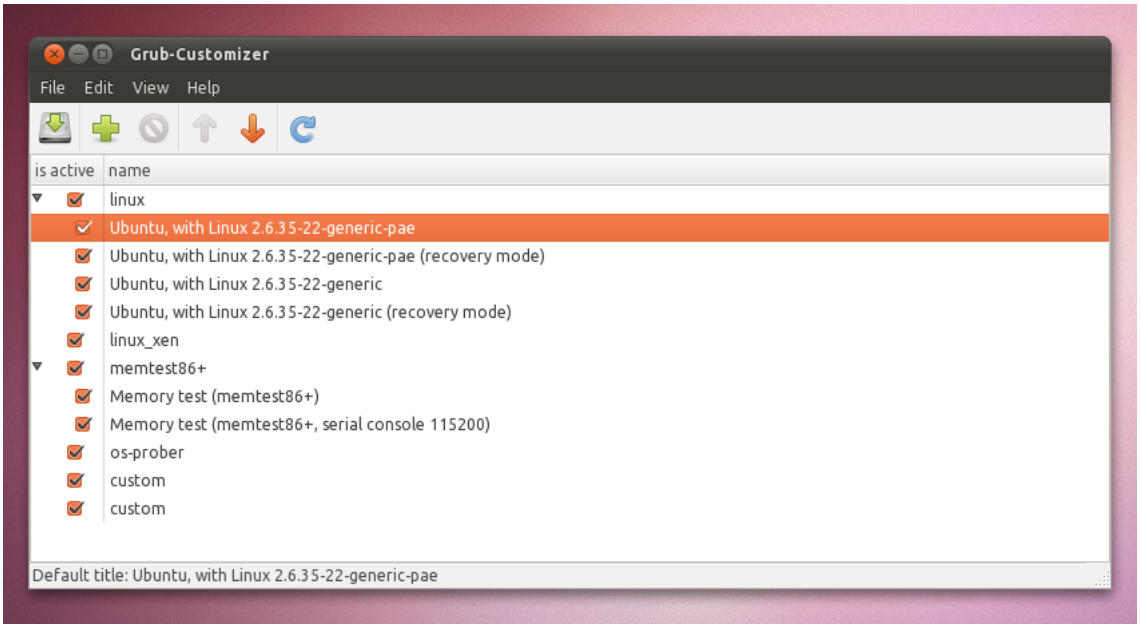


**Figure 4.1:** Overview of the grub customizer program.

The Multipath TCP kernel is now running on both the server and the client computer

### 4.1.2 Setting up the routing tables on the client

In order to be able for the client to simultaneously use multiple subflows, some rules for the routing of both flows is needed. These are setup in a static manner since this is the easiest way to do it, when only dealing with two subflows. It is possible to setup the routing tables automatically and can be done by incorporating scripts found on the multipath-tcp website [13]. It can also be done via different tools, which can be downloaded from github and these tools can be found on the official multipath-tcp website.

The setup is currently being done by running a script, which contains the static routing tables. As seen in 14 it creates two different routing tables for each of the two network interfaces. It also creates a default route, which is used as the primary interface for web browsing, downloading and all other tasks that are not MPTCP enabled.

```
1 # This creates two different routing tables, that we use based on the source-address.
2 ip rule add from 10.42.0.52 table 1
3 ip rule add from 192.168.137.2 table 2
4
5 # Configure the two different routing tables
```

```
6  ip route add 10.0.42.0/24 dev eth0 scope link table 1
7  ip route add default via 10.4.0.1 dev eth0 table 1
8
9  ip route add 192.168.137.0/24 dev eth1 scope link table 2
10 ip route add default via 192.168.137.1 dev eth1 table 2
11
12 # default route for the selection process of normal internet-traffic
13 ip route add default scope global nexthop via 10.0.42.1 dev eth0
```

**Listing 4.1:** Static routing tables for the two subflows of the client.

### 4.1.3 Setup of the Multipath TCP protocol

In this section the different options that are included in the Linux kernel implementation of the Multipath TCP will be explained. It will include descriptions of the different schedulers available in the kernel as well as a description of the different algorithms used for the congestion control.

#### Enabling the mptcp enabled kernel parameter

In order to be able to use the MPTCP it needs to be set as enabled in the kernel. This is not done when installing and booting the kernel and therefore it needs to be done in order to use the multipath option. This is done by changing a flag in the kernel so the MPTCP option is enabled and is shown in 2.

```
1 sudo sysctl -w net.mptcp.mptcp_enabled=''1''
```

**Listing 4.2:** Enabling the MPTCP enabled option in the kernel.

When using the sysctl command it modifies the kernel parameters at runtime, which means that no reboot is needed in order for the changes to take effect. This is also the way the path manager, the scheduler and the congestion control is changed.

The mptcp parameter is now set to 1, which means that it is enabled.

#### Configuring the path-manager

The path manager of the Multipath TCP is used for the creation of new subflows. At compile time different path-managers can be chosen:

- **default:** Does not actively create or announce new subflows. It will stay passive and only accept incoming requests for the creation of new subflows.

- **fullmesh:** Will create a full-mesh of subflows among all of the available subflows. Since the version 0.90 it is possible to create multiple subflows per IP-address and this is controlled with the option /sys/module/mptcp_fullmesh/parameters/num_subflows, which is then set to an integer larger than 1.
- **ndiffports:** It creates X subflows across the same pair of IP-addresses. It is controlled by the num_subflows as described in the fullmesh path-manager.
- **binder:** This path-manager uses Loose Source Routing which is described in Binder: a system to aggregate multiple internet gateways in community networks [14].

### Configuring the scheduler

In order to be able to understand what the scheduler is when discussing multipath networks, an explanation of the problems, that emerges, when different types of schedulers are used to direct the traffic along the different subflows when using MPTCP, will be given.

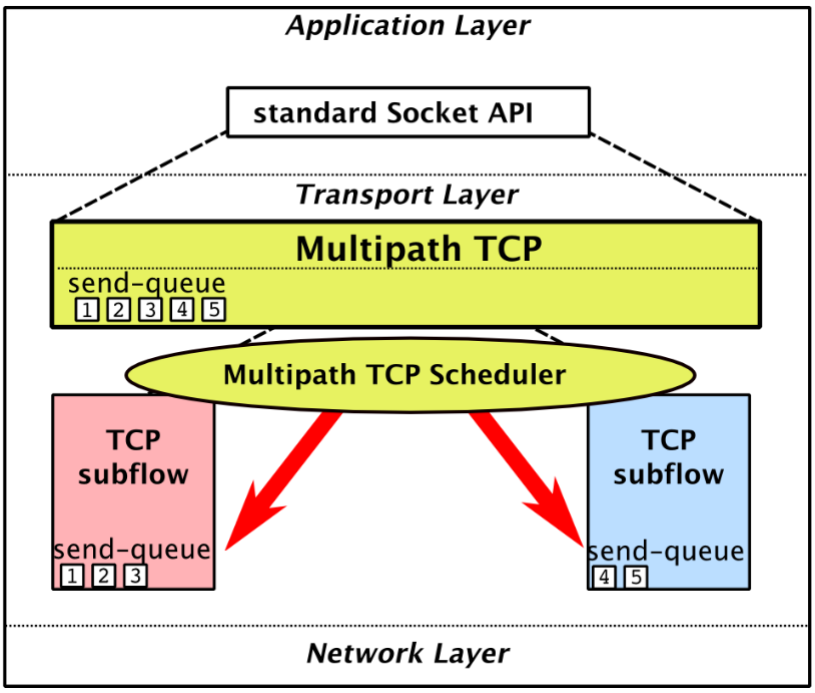


**Figure 4.2:** Overview that shows the task of the scheduler of Multipath TCP.

In Figure 4.2 the task of the scheduler is presented graphically. It shows a scenario, where the MPTCP send queue has 5 different packets ready to be sent via two different subflows. The first 3 packets are sent over the first subflow and the 2 last packets are sent over the second subflow. This is the scheduler that handles this and by using different types of schedulers the throughput might be optimized. The design of a scheduler should take different informations into consideration. In the next paragraph some of the problems that exist with using multiple subflows will be explained.

There are 2 main problem scenarios that needs to be solved by using the correct scheduler.

### Head-of-Line Blocking

As seen in Figure 4.3 the image shows what happens when the two subflows have different delay characteristics aka. Round Trip Times (RTT). In the shown scenario the first subflow has a high delay of 150ms and the second subflow has a low delay of 20ms. This presents a problem when the packets are split up and in this case the first packet are sent via the first subflow and the last 3 packets are sent via the second subflow, which has a much lower delay. The problem arises, when the host are waiting for the first packet to arrive, so it can reassemble the packets and deliver it to the application layer.
The reason why this is a problem, is because Multipath TCP ensures in-order delivery so the packets sent on the low-delay subflow have to wait for the packets sent via the high-delay subflow in order to reorder them. This need to be done to get a full data segment and then deliver it to the application that requested the data. As seen in Figure 4.3 when head-of-line blocking happens, it creates different problems such as blocking the entire session, when waiting for packets sent through the high-delay subflow. This creates high application-level delay since the receiver needs to wait for data from both subflows, in order to reorder it and send it to the application layer. It also creates burstiness, which comes from the fact that in the waiting period no data will be delivered to the application-layer, whereas when the data has arrived large chunks of data segments may be delivered to the application-layer in short period of time.

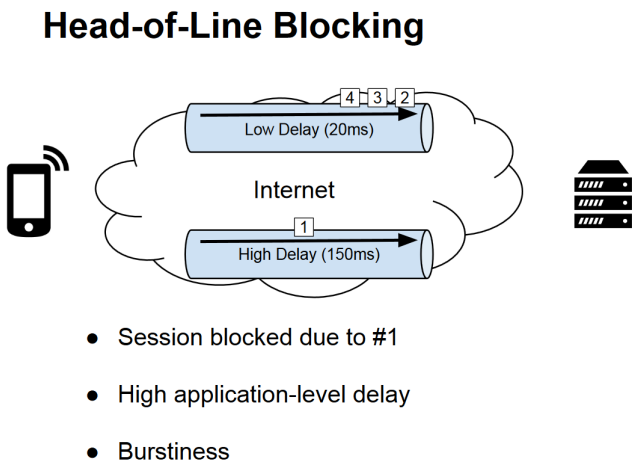


**Figure 4.3:** Graphical overview of the Head-of-Line Blocking problem.

### Receive-window limitations

In regular TCP the TCP stack reserves a certain amount of memory for out-of-order data segments. This memory is used in the event of in-network reordering or packet loss. In Multipath TCP this functionality is implemented across all the different TCP subflows due to the delay differences that can occur. So the receive buffer needs to have enough memory to accommodate out-of-order data segments at the Multipath TCP level. So the size of the buffer is important such that a high goodput can be achieved.
To be able to fully utilize the entire capacity of all the paths, the receiver must have enough buffer space, so that the sender can maintain all of the subflows fully utilized, even if some data segments needs reordering due to the delay characteristics or from packet loss. In [20] a recommendation for the receive buffer size for Multipath TCP is presented:

$$Buffer = \sum_{i}^{n} bw_i \cdot RTT_{max} \cdot 2 \tag{4.1}$$

This recommendation will allow each subflow to send at full speed in the time-interval of the highest RTT among all the subflows, even though packet loss occurs. However this recommendation can not always be fulfilled, since some of the hosts are not able to provide enough buffer space, to utilize the full capacity of all the different subflows. Since this article [20] was written, another approach has emerged,

which suggests incremental changes to the heuristics within the Multipath TCP, by retransmitting segments and then penalizing the slower subflows.

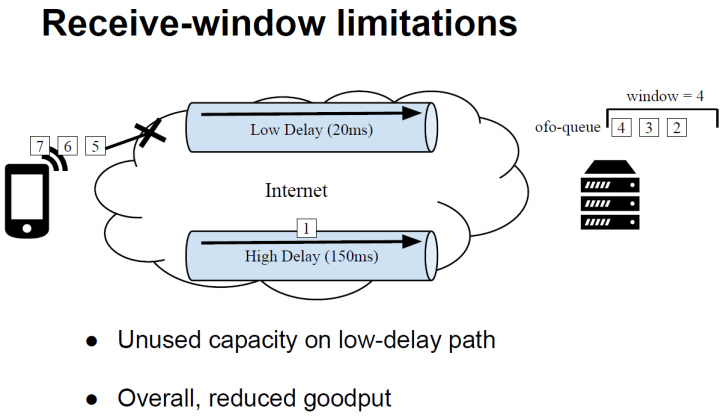


**Figure 4.4:** Graphical overview of the Receive-window limtations.

### 4.1.4 MPTCP Schedulers

In this section the different schedulers that are available in the MPTCP implementation will be explained.

#### Round-Robin scheduler

This scheduler works by selecting the different subflows one at a time in round-robin fashion. This might guarantee, that the capacity of each subflow is utilized fully and this is because the distribution across all subflows becomes equal. However if the data transmission is bulk data, then the scheduler is not functioning in real round robin fashion. This is because that whenever all the subflows have been filled with capacity, the next packets to be transfered are scheduled as soon as capacity becomes available again in the different subflows. This is because the different subflows can have different bandwidth limitations and different delay characteristics, and then some subflows might transfer packets sooner than it should using a real round robin scheduler. This effect is known as the *ack-clock* [21].

#### Lowest-RTT-First (Defalut Scheduler)

It make a lot of sense to schedule data transfers on the paths with the lowest round trip times since it can improve the user experience. It also decreases the application delay which is crucial if the type of service is interactive. The Lowest-RTT-First scheduler works by first sending data on the paths with the lowest delay estimation until the congestion window of the path is filled. Then the algorithm proceeds to send data on the path with the next highest RTT. As stated in subsubsection 4.1.4, whenever all the congestion windows are filled, the scheduling becomes *ack-clocked*. This is because that in the individual subflows capacity is freed up whenever an acknowledgment is received and then the scheduler is able to transmit new data on the path in question.

### 4.1.5 Configuring the congestion control

In this section the different congestion control algorithms that are currently available for the Multipath TCP will be explained.

In Multipath TCP the load is distributed by creating multiple subflows between all of the potential paths from the source to the destination. The easiest way to solve the congestion control in Multipath TCP is to just apply the standard congestion control found in the TCP. However the congestion control in MPTCP is different from regular TCP and simply applying the standard congestion control from TCP, would cause the multipath flows to get a higher bandwidth, when sharing a bottleneck link with a regular TCP connection. This is not be considered fair to the regular TCP and therefore another course of action is needed. Another feature of the Multipath TCP is that, when using multiple subflows it is desirable that the traffic is transfered through the subflows with the least congestion such that resource pooling is achieved, where a group of links acts like one shared link with larger capacity. Any algorithms that control the congestion in Multipath TCP must meet three requirements[8]:

- A multipath connection should perform at least as well as single path TCP would on the best of the available paths.
- A multipath subflow should not take more capacity than a single path TCP would on the same link.
- The multipath flow should move as much traffic as possible to the least congested paths.

In the Multipath TCP release v0.90 a total of four different algorithms for the congestion control is present. These congestion control algorithms are:

- Alias Linked Increase Congestion Control (LIA).
- Opportunistic Alias Linked Congestion Control (OLIA).
- Balanced Linked Adaptation Congestion Control (BALIA).
- Delay-Based Congestion Control (wVegas).

These can all be switched as a kernel parameter at runtime and this is done by issuing the command:

```
1 sudo sysctl -w net.ipv4.tcp_congestion_control='lia'
```

In the following sections an explanation of the different congestion control algorithms included in the MPTCP will be given.

### Alias Linked Increase Congestion Control

The Alias Linked Increase Algorithm (LIA) couples the congestion control algorithms that are active on the different subflows by relating their increase functions and then it automatically readjusts the congestion window. This algorithm is however only applied to the increase part of the congestion avoidance phase. This ensures that the algorithm is fair to a regular TCP connection at a bottleneck link and it also ensures that traffic are moved away from congested paths. In [8] the relation of the increase functions and the subtractive decrease behaviors is described as:

- For each ACK received on subflow $i$, the congestion window $cwnd_i$ is increased by:

$$Min\left\{\frac{\alpha \cdot B_{ack} \cdot M_{SS_i}}{\sum_{i=0}^{n} cwnd_i}, \frac{B_{ack} \cdot M_{SS_i}}{cwnd_i}\right\} \tag{4.2}$$

  Where:

  $\alpha$: A parameter that describes the aggressiveness of the multipath flow.
  $B_{ack}$: Number of acknowledged bytes.
  $M_{SS_i}$: Maximum segment size on subflow $i$.
  n: Total number of subflows.

- For each loss on subflow $i$, decrease $cwnd_i$ by $cwnd_i/2$

In Equation 4.2 it is shown that a minimum of two different calculations is taken. This is calculated in bytes. The first calculation computes the increase value for the multipath subflow and the second calculation is for a regular TCP flow in the same scenario. By taking the minimum of these two calculations, it is ensured that the multipath subflow will not take more capacity than a regular TCP connection on a single path. The alpha value is used to describe the aggressiveness of the multipath subflow and the value is chosen such that the total throughput of the multipath flow is equal to the throughput, that a regular TCP connection would get on the best path available.

As seen in Equation 4.2 the first calculation depends on alpha, maximum segment sizes and the RTTs of the paths. It is not possible to chose a value for alpha, which fulfills the first design goal stated in subsection 4.1.5, such that the desired throughput is achieved at all times. To solve this, alpha is therefore computed by taking the observed behaviors of all the multipath subflows. This is shown in Equation 4.3.

$$\left( \sum_{i=0}^{n} cwnd_i \right) \frac{Max\left\{ \frac{cwnd_i}{RTT_i^2} \right\}}{\left( \sum_{i=0}^{n} \frac{cwnd_i}{RTT_i} \right)^2} \tag{4.3}$$

Where:

$Max\left\{ \frac{cwnd_i}{RTT_i^2} \right\}$: Maximum value of any possible path.
$\sum_{i=0}^{n} \frac{cwnd_i}{RTT_i}$: Summation of all possible values of all paths.

### Opportunistic Alias Linked Congestion Control

In [19] an extensive performance evaluation of the LIA congestion control algorithm was carried out and it was discovered that the LIA algorithm forces a tradeoff between optimal resource pooling and responsiveness. This is a problem since the LIA algorithm is unable to fulfill both goals at the same time and this eventually leads to a fairness problem for regular TCP users.

Opportunistic Alias Linked Congestion Control was introduced as an alternative algorithm, that was capable of solving the problems that LIA presented and as stated above. The algorithm works by linking the increase functions of the congestion window and it uses the same behavior as regular TCP in the event of lost packets. This algorithm is only applied to the increase function of the congestion avoidance. Therefore the slow start algorithm is the same as used in regular TCP, but with a small change, which is used when multiple paths are created.[19] To describe the additive increase behavior[8]:

- For each ACK received on path $i$, increase congestion windows $cwnd_i$ by:

$$\frac{\frac{cwnd_i}{RTT^2}}{\left(\left(\sum_{i=0}^{n} cwnd_i\right) \cdot \left(\frac{cwnd_p}{RTT_p}\right)\right)^2} + \frac{\alpha_i}{cwnd_i} \tag{4.4}$$

  Where:

  $cwnd_p$: Window size of a path p with largest congestion window.
  $RTT_p$: Round trip time of a path p with largest congestion window.
  $\alpha_i$: Adjust parameter for a path i.
  n: Total number of subflows.

- For each loss on subflow $i$, decrease $cwnd_i$ by $cwnd_i/2$

The first term in Equation 4.4 ensures that optimal resource pooling is achieved and furthermore this term is a TCP compatible version, which compensates for difference in the round trip times. The second term is what guarantees that the algorithm has responsiveness and is non flappy.

### Balanced Linked Adaptation Congestion Control

As stated in LIA and OLIA these congestion algorithms suffer from either unfairness to a single path TCP or they are unresponsive to network changes under specific conditions. Among these conditions it is a problem when all paths used for the Multipath TCP has the same round trip times. However a tradeoff between these issues is inevitable and BALIA tries to judiciously balance this tradeoff [1]. BALIA has a good balance between friendliness and responsiveness. In [8] the additive increase and the subtractive decrease behaviors of BALIA is described as:

- For each ACK on path $i$, increase $cwnd_i$ by:

$$\left(\frac{x_i}{RTT_i \cdot (\sum_{i=0}^{n} x_k)^2}\right) \cdot \left(\frac{1+a_i}{2}\right) \cdot \left(\frac{4+a_i}{5}\right) \tag{4.5}$$

  Where: $x_i = \frac{cwnd_i}{RTT_i}$
  $a_i = \frac{Max\{x_k\}}{x_i}$
  n: Total number of subflows.

- For each loss on path $i$, decrease $cwnd_i$ by:

$$\left(\frac{cwnd_i}{2}\right) \cdot Min\{a_i, 1.5\} \tag{4.6}$$

  In the case that there is only one path available then $a_i$ will be 1 and both the increment and decrement formulas will be the same as in the TCP Reno algorithm [26].

### Delay-Based Congestion Control

This delay based congestion algorithm was introduced in [16] and it works, unlike the LIA algorithm which is basked on packet loss events, by taking the packet queuing delay as a congestion signal. Because the wVegas algorithm is a delay based algorithm it is more sensitive to network changes with respect to congestion and therefore it is able to achieve a more timely traffic shifting and a faster convergence compared to e.g. the LIA congestion algorithm. In order for wVegas algorithm to work it must perform these operations after the end of each transmission round[8]:

- For a subflow $i$, calculate the difference between the expected sending rate and actual sending rate.

$$diff_i = \left(\frac{cwnd_i}{base\_RTT_i} - \frac{cwnd_i}{RTT_i}\right) \cdot base\_RTT_i \tag{4.7}$$

  Where $RTT_i$ is the average RTT on the last round on subflow $i$, and $base\_RTT_i$ is the RTT of a subflow $i$ when the path is not congested.

  In the case that the subflow is in the slow start phase and the threshold value called gamma is larger than $diff_i$, then the algorithm goes into the congestion avoidance phase.

- For the congestion avoidance phase it is checked if $diff_i$ is less than the unfairness $a_i$ or not. If $diff_i$ is not less than $a_i$ then the rate must be updated.

$$rate_i = \frac{cwnd_i}{RTT_i} \tag{4.8}$$

$$weight_i = \frac{rate_i}{total\ rate\ of\ all\ i} \tag{4.9}$$

$$a_i = weight_i \cdot total_a \tag{4.10}$$

  If $diff_i$ is larger than $a_i$ then:

$$cwnd_i = cwnd_i - 1\ elsecwnd_i = cwnd_i + 1 \tag{4.11}$$

- For the last task wVegas will try to improve the accuracy of the $base\_RTT_i$ by ensuring that, when the algorithm detects a queuing delay larger than some threshold, the congestion window backs off. By running this task, bottleneck links can decrease the backlogged packets. But not only the flows with queuing delay has a chance to obtain the most accurate propagation delay. This goes for all the flows involved and wVegas tries to calculate the queuing delays as[8]:

$$queue\_delay_i = RTT_i - base\_RTT_i \tag{4.12}$$

  This leads to two cases:

  - current queuing delay $<$ saved $queue\_delay_i$, replace $queue\_delay_i$ with current.
  - current queuing delay $= 2 \cdot queue\_delay_i$ then

$$cwnd_i = (cwnd_i) \cdot (0.5) \cdot \left(\frac{base\_RTT_i}{RTT}\right) \tag{4.13}$$

### 4.1.6 Gateway

The gateway will be used to throttle the network connection to the client. Since the server and the client is running on the same local area network, the delay of the traffic, when passing through the gateway is very low, there will be added delay at the incoming network connection of the gateway. This delay will be set as 40 ms on the outgoing packets from eth0. So the delay will be added when the client sends an ACK to the server. This is perfectly fine, since the data that will be used is the round trip times.
The gateway will use Ubuntu 14.04 and it will use the internal function to share the

Internet connection from eth0 to eth1 and eth2. On both the interfaces, eth0 and eth1, a queue will be created to throttle the bandwidth. This queue will use a token bucket filter with a burst of 2 kB. The burst is explained in subsubsection 4.2.5 and in [10].

## 4.2 Test prerequisites

In this section the prerequisites for setting up the tests and the Multipath TCP will be described.

### 4.2.1 Design base

In this section the information of the design base of the test will be given. It will provide information about the different paths that are setup, in order to test the Multipath TCP bandwidth aggregation.

The design base will consist of 1 regular TCP connection that is limited to 2 Mbps download and 512 Kbps upload. This path will be used to generate the base results, such that when performing the tests with the Multipath TCP and 2 paths, one being the 2 Mbps / 512 Kbps DSL line and the other being the 3G/4G connection, it will either show that using the MPTCP is able to aggregate more bandwidth than the single path.

### 4.2.2 Network metrics

In this section the network metrics that are important for this project will be outlined.

To verify the results of the tests run in this project it is important that some network metrics are agreed upon. This is some basic requirements for avoiding quality of service problems in VoIP conversations [11].

- Ideally there should be no packet loss for VoIP.
- A maximum latency of 150 ms.
- In order for jitter buffers to work correctly there should be no delay variations that exceed 100 ms.

Besides these requirements there will be different bandwidth requirements depending on which audio and video codec that are used for the interactive streaming. But this gives a general idea of what network metrics are important for verifying our tests. So to sum up the network metrics that will be used to evaluate the tests these three will be chosen:

- Bandwidth
- Latency
- Jitter

#### Jitter

Jitter will be calculated from the distribution of ACK RTTs. It will be done by making the assumption that the one way delay is half of the ACK RTT time. This one way delay will then be used to calculate the mean jitter and the variance in the jitter. This is explained further in chapter 5.

### 4.2.3 Path characteristics

In order to carry out the different tests it is important to be able distinguish between homogeneous and heterogeneous paths. The difference between these two types of paths will be given below.

#### Homogeneous paths

When using the term homogeneous paths this is to say something about the characteristics of the paths. Homogeneous means that the paths, that the data traverse, has similar network characteristics. This means that if two homogeneous paths are used, it would be two paths with almost identical bandwidth, latency and jitter. In this project when using more that one homogeneous path it will consists of emulated DSL connections.

#### Heterogeneous paths

Using heterogeneous paths means that, the paths the data traverses, has very different network characteristics. This term can be used, when the Multipath TCP consists of e.g. a DSL connection with a low and consistent latency is then combined with a 4G connection that has a higher and more fluctuating latency.

### 4.2.4 Traffic

In this section the software that will be used to generate the traffic on both the regular TCP and the Multipath setup will be described.

#### Traffic direction

This section will explain how the test will be setup and in which direction the data will flow.

Because the DSL line is emulated by using a gateway, it is throttled at 2048 Kbps downstream and 512 Kbps upstream. However a real ADSL line will not be able to receive with full capacity if the upstream is maxed out. This is because an ADSL line is asymmetric. However the gateway is not able to emulate this behavior so as of now this will not be implemented.

In order to generate the traffic, so it is similar to the traffic from the normal scenario described in chapter 3, the traffic will be split up in two different categories.

- Background traffic
- Foreground traffic
  - General purpose (Surf, Netflix etc.)
  - Interactive streaming

#### Background traffic

In order to generate some background traffic, that will model how normal web surfing traffic looks like, a session where this type of traffic is being used will be recorded with

Wireshark and analyzed in order to model the packet sizes so it can be reproduced. The traffic will either be generated using a piece of software or an application written in c++ or Java that can mimic the burst traffic of a normal web surfing session. Since the background traffic is not the most important part of the project a background traffic model will be assumed.

The assumed model used for the background traffic will be:

- **Downstream (From the server to the client:** 500 Kbps in small bursts. These bursts will be done so that traffic is generated and then there is a small pause of 1-2 seconds.
- **Upstream (From the client to the server:** 100 Kbps in small bursts. These bursts will be done so that traffic is generated and then there is a small pause of 1-2 seconds.

**iperf3** This tool has a lot of options that can be adjusted and tweaked. It is especially relevant because the bandwidth can be limited directly in the application. It can also provide bi-directional traffic and run in both directions.

### Foreground traffic

Since the foreground traffic can be split into two different scenarios, namely being interactive streaming and general purpose, they will be split into two different sections.

**General purpose** The name general purpose is used because this type of traffic can traffic types such as video streaming, web browsing, file download but is not limited to these types. It is in fact just a category that covers what ever traffic that might pass through the network. This kind of traffic does not have any strict requirements when looking at latency or jitter and will mostly depend on the bandwidth of the connection. This is where Multipath TCP can help since it will be able to pool resource from several different active connections simultaneously. If the general purpose is used for e.g. Netflix, YouTube or any other video streaming, there will be some requirements to the jitter and latency.

For the general purpose it is assumed that the network traffic generated will be some streaming video e.g. YouTube, Netflix or another similar service so here the

recommendations made by Netflix for the bandwidth of the Internet connection will be listed. These will be used in the tests when analyzing the results in order to verify if the bandwidth and jitter, obtained in the test, is sufficient to use the network connection for the purpose of video streaming. The information listed is found at [17].

- 0.5 Megabits per second - Required broadband connection speed
- 1.5 Megabits per second - Recommended broadband connection speed
- 3.0 Megabits per second - Recommended for SD quality
- 5.0 Megabits per second - Recommended for HD quality
- 25 Megabits per second - Recommended for Ultra HD quality

**Interactive streaming** In order to find out how interactive streaming traffic looks like we will take a look at the voice codec called G.711. This codec has some basic specifications that will help us in creating a model for the interactive streaming traffic. In [22] we see that G.711 is a standard for voice communication namely telephony. It was developed in 1972 and is also known as Pulse Code Modulation (PCM). It is frequently used as an audio codec in videos as well. It works by sampling audio signals in the range of 300-3400 Hz at a rate of 8000 samples per second. This gives a new audio sample every 0.125 ms. So it has these specifications:

- Bit rate: 64 Kbps
- Frame size: 0.125 ms

If the audio is packetized every 20 ms this will give a packet size of 160 bytes and if we include 40 bytes for the header the voice packet size will be 200 bytes. A packet with this size will then get sent to the send buffer every 20 ms. Since this project is about MPTCP it also makes use of sending segments of approximately 1500 bytes so when the buffer has enough data available it will arrange this in one packet of 1500 bytes and then it will get sent to the client.

This is modeled in iperf3 by writing a small array of 8k bytes to the buffer a number of times. This will ensure that the data available in the buffer, is handled as a steady stream of data to send out, just as if the G.711 codec was used, the only difference being that these writes are larger than when using G.711. This is to make up for the fact that the project is about interactive video streaming and not only audio.

### 4.2.5 Hardware

In this section the setup of the hardware used in the testbed will be explained.

#### Gateway

In this section the gateway setup will be explained.

As per initial results have shown, the first setup used on the gateway included a token bucket filter queue with a queue buffer of 2000 packets. This created a large queue in some of the tests and it can be seen in the round trip time graphs that are created by wireshark. The queue is changed to using a burst of 2000 bytes and a maximum wait of 100ms for a token[10]. This means that the token bucket filter can only have 2 kB tokens that can be available for instantaneously traffic. If more traffic arrives in the buffer the packets will get dropped after a wait period for a token of 100 ms. This setup is done to make the setup reflect a real life setup.

#### Server

As it will show in the throughput graphs for the regular TCP tests the packet sizes captured by wireshark is around 2900 bytes. This happens because because of something called offloading [3]. Offload is when the network interface card (NIC) handles some of the network processing to offload the CPU of some of its processes. This manifests it self in wireshark as packet sizes being 2900 bytes. It is because that the particular NIC can reassemble traffic. Since this has no impact on the tests done the setting will be left alone. This, however, only happens in the regular TCP tests and not the Multipath TCP tests. This can be explained by the underlying kernel implementation that changes some of the network settings in the operating system to make sure that offloading happens in the CPU and not on the NIC. More info on how to turn this function off can be found in [3]

### 4.2.6 Software

In this section the software used in the tests will be described and also the tools used to analyze the data captured in the results.

### Wireshark

Wireshark is an open tool piece of software that is used to analyze e.g. network packets. The tool is cross platform and is similar to the tool called tcpdump. However Wireshark has a GUI, which makes it easy to use and customize, where tools as tcpdump, which is command line based, take more knowledge to master.

Wireshark is an essential tool in order to capture the raw packets from the tests. The information recorded in wireshark will be used to follow the TCP streams created by the Multipath TCP kernel and to extract information used in the analysis of the tests. Wireshark will be used to capture traffic at both endpoints, namely the at the server and at the client.

### Capinfos

Capinfos is a small commandline interface that is supplied along side wireshark and it will be used to generate the statistics of average bit rate and average packet sizes [7].

### iperf3

For performing all tests iperf3 [23] will be used to generate the traffic. In this section the different options used in iperf3 will be listed and described.

Iperf3 is a network testing tool that can create TCP and UDP data streams. These are used to measure the throughput of the network on which the traffic is running. The program is written in C.

```
1  Server or client:
2    -p, --port      #         server port to listen on/connect to
3  Server specific:
4    -s, --server              run in server mode
5  Client specific:
6    -c, --client    <host>    run in client mode, connecting to <host>
7    -b, --bandwidth #[KMG][/#] target bandwidth in bits/sec (0 for unlimited)
8                               (default 1 Mbit/sec for UDP, unlimited for TCP)
9                               (optional slash and packet count for burst mode)
10   -t, --time      #         time in seconds to transmit for (default 10 secs)
11   -w, --window    #[KMG]    TCP window size (socket buffer size)
12   -l, --len       #[KMG]    length of buffer to read or write
13   -N, --nodelay             set TCP no delay, disabling Nagle's Algorithm
```

# Chapter 5

# Tests

In this chapter all the test results for each of the tests will be described and all the relevant graphs that has been produced will be added.

## 5.1 Goals

By conducting all of these tests there are some results that have more importance, for this project, than others. The results that are especially interesting are:

- How does regular TCP compare to Multipath TCP in terms of goodput and jitter?
- When using Multipath TCP how does two homogeneous paths compare to heterogeneous in terms of goodput and jitter?

In order to be able to say something about how one test compares to another all the individual test will be analyzed and explained in the sections respectively. The analysis will reflect upon:

- Throughput of all paths
- RTT of all paths
- Goodput of all paths
- Jitter of all paths

## 5.2 Test setup

In all the tests carried out the traffic direction is only running from the server to the client. The only traffic flowing from the client to the server is the ACKs and the background traffic as explained in subsubsection 4.2.4. When running the tests, the bandwidth limit added on the gateway, will work in both directions and instead of using a setup of a downstream of 2048 Kbps and an upstream of 512 Kbps the effectively setup will be, that the gateway throttles the connection in both directions. So this give a bandwidth limit of 2048 Kbps downstream from the gateway to the client and a 2048 Kbps upstream from the client to the gateway.

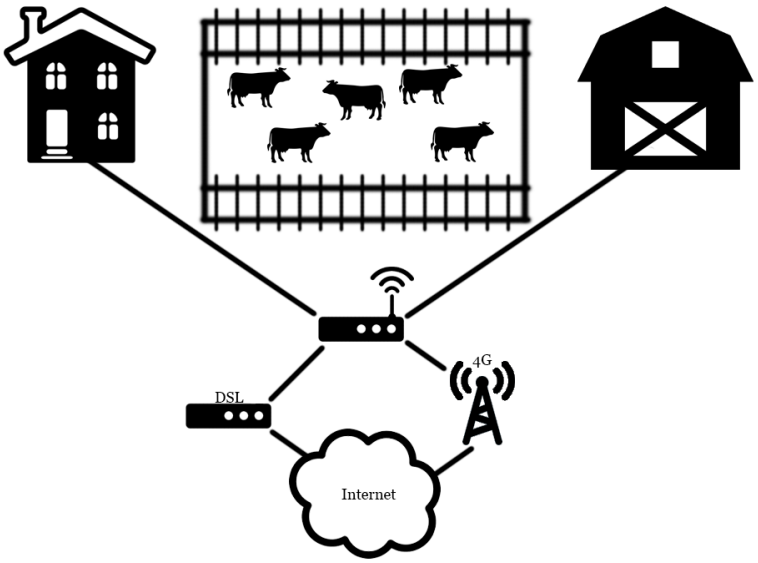


**Figure 5.1:** This figure shows the scenario overview.

Figure 5.1 shows the general principles of the scenario as observed at the farmer. For explaining how the tests was run Figure 5.1 will be split into 3 parts. These parts will be described and explained in the next paragraphs.

To help explain Figure 5.6, Figure 5.4 and Figure 5.59:

- **Server:** The server is connected to the internet by a walled ethernet plug at Aalborg University.

- **Gateway:** The gateway is connected to the internet by walled ethernet (eth0) and is used to throttle and share the incoming network connection to two other network interfaces (eth1 and eth2).
  - **Delay:** A 40 ms delay has been added on the outgoing traffic of eth0. This way the ACKs from the Client are delayed 40 ms.
  - **Queue:** On the figures the Queue is the queue type e.g. token bucket filter, the latency is how long the packets can maximum wait for a token before they are dropped, the burst is the maximum size of the token bucket queue.
- **Client:** The client is connected to the gateway by ethernet cables.
- **Connections:** If no number is added on the connections they are not limited and where there is a number added it is to illustrate what the bandwidth limit is set to. If a number is added the bandwidth limit is done in both downstream and upstream direction.

In Figure 5.6 the setup is done using a single regular TCP connection from the server to the client. In this setup the gateway will be used to throttle the network connection, from the gateway to the client. The gateway will have a token bucket filter queue, where the maximum total size of the tokens cannot exceed 2000 bytes. The packets can wait for up to 100 ms for a token and after this they will be dropped.

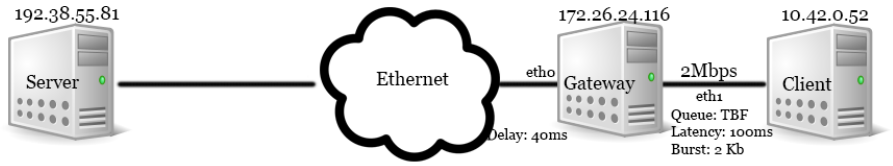


**Figure 5.2:** This figure shows how the test setup for regular TCP is done.

In Figure 5.59 the setup is done by using two multipath TCP connections consisting of emulated DSL connections, that goes from the server to the client through the gateway. As seen on Figure 5.59it is the same queue type, that was used for the regular TCP.

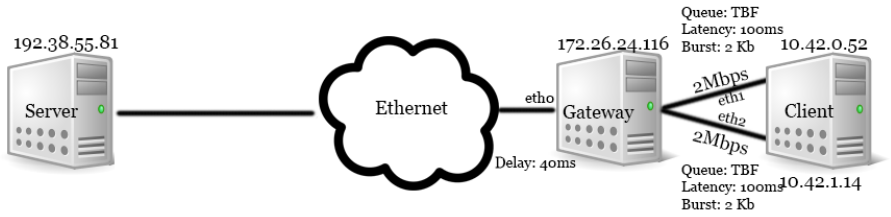


**Figure 5.3:** This figure shows how the test setup for MPTCP with two DSL connections is done.

In Figure 5.4 the setup is done using a single multipath TCP connection consisting of an emulated DSL connection and a real 4G connection using the TDC network. The queue implemented at the gateway, for the emulated DSL connection, is the same as used in the setup of the regular TCP and the multipath TCP scenario with two emulated DSL connections.

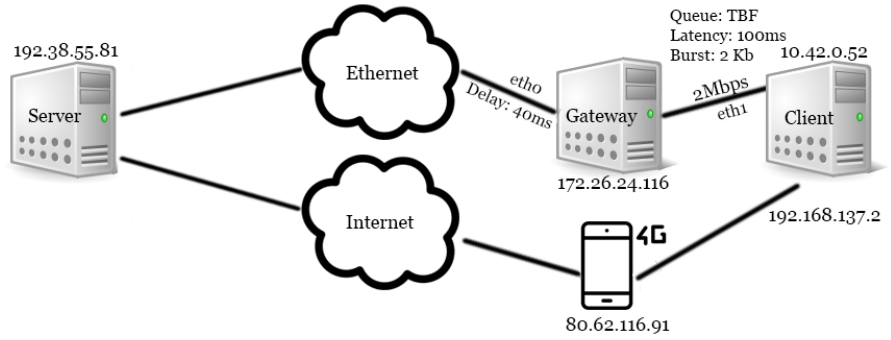


**Figure 5.4:** This figure shows how the test setup for MPTCP with one DSL connection and one 4G connection is done.

### General purpose

In order to be able to determine the throughput when using regular TCP or Multipath TCP the general purpose test will be performed. It will have two different bandwidth limitations configured namely a bandwidth limit of 3 Mbps and a bandwidth limit of 5 Mbps. When testing the regular TCP a 3 Mbps bandwidth will be used in iperf3, in order to not push too much traffic through the queue on the gateway. For the Multipath TCP tests two different limits will be used, namely 3 Mbps and 5 Mbps. The 3 Mbps limit is used to determine how the Multipath TCP performs, when the bandwidth of the generated traffic is below the available bandwidth of the combined

subflows. The 5 Mbps limit is used to examine how the Multipath TCP performs, when the bandwidth of the generated traffic is higher, than the available bandwidth of the combined subflows.

### Interactive streaming

In order to be able to determine the performance of the network connection, when using it for interactive streaming purposes it is important to define what characterizes a good network connection for interactive streaming purposes. In order to be able to have an interactive streaming conversation the first thing to look at will be the audio stream or Voice-over-IP as it is also called. The minimum requirements for a VoIP conversation is that:

- One way delay can be no more than 150 ms including processing delay, reassembly of packets etc [11].
- The variance in jitter should be below 100 ms. This is because the jitter buffers, which are used to compensate for varying delay, are only effective when the jitter is below 100 ms [11].

In order to calculate the jitter a few assumptions is made:

- 1: Since the maximum one-way delay can be 150 ms we assume that when all processing through the layers are done 100 ms - 125 ms is left for the network delay.
- 2: Since only the ACK RTT times are available we assume that the delay is the same in both directions so the ACK RTT time will be divided by 2 to get the one way delay.

In Figure 5.5 it is shown how the ACK RTT values are obtained at the server. Each of these ACK RTTs will be used to calculate the jitter. So the definition of jitter for these tests will be:

$$Jitter = \frac{RTT(i+1)}{2} - \frac{RTT(i)}{2} \tag{5.1}$$

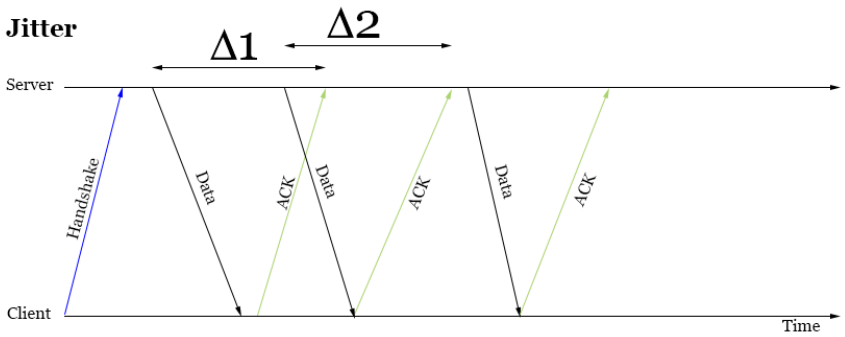


**Figure 5.5:** This figure shows the Delta used to estimate the jitter.

Since the data is extracted from wireshark in order to calculate the jitter, these ACK RTT times is per flow. So when calculating the jitter in the MPTCP tests it is calculated per path and not when the TCP stream is reassembled. It would be better to do an analysis of the jitter when the TCP stream has been reassembled because this is comparable to a regular TCP connection. However since this information is not accessible through wireshark the numbers for each path will be used.

## 5.3 Test description

In this chapter all the different test descriptions will be explained. The tests will be numbered with A1.1, A1.2, A2.1 and so on and so fort. A tests will be for regular TCP, B tests will be for Multipath TCP with homogeneous paths and C tests will be for Multipath TCP with heterogeneous paths.

### 5.3.1 iperf3 server setup

Setup of the server will be done in the same way for all tests. It will use 4 instances of iperf3 running on 4 different ports. This is done because the total amount of connections needed when testing general purpose and interactive streaming plus background traffic running in both directions, it will require 4 active connections and since iperf3 only handles one connection per port, 4 instances are needed.

The server will setup iperf3 by running the following commands in a terminal.

```
1 $ iperf3 -s -p 60020
2 $ iperf3 -s -p 60021
3 $ iperf3 -s -p 60021
```

```
4 $ iperf3 -s -p 60021
```

All tests will be run for 60 seconds unless another time period is specified. The tests will be split up in to three categories. These categories will be:

- **Test A - Regular TCP:** This category will perform tests using a single regular TCP connection.
- **Test B - MPTCP DSL + DSL:** This category will perform tests using two homogeneous paths namely two emulated DSL connections.
- **Test C - MPTCP DSL + 4G:** This category will perform tests using two heterogeneous paths namely one emulated DSL connection and a 4G connection using the TDC network.

A total of 3 tests will be performed for the regular TCP scenario and these are:

- General purpose without background traffic and a 3 Mbps limit in iperf3.
- Interactive streaming without background traffic.
- Interactive streaming with background traffic.

A total of 4 tests will be performed for the MPTCP scenarios and these are:

- General purpose without background traffic and a 3 Mbps limit in iperf3.
- General purpose without background traffic and a 5 Mbps limit in iperf3.
- Interactive streaming without background traffic.
- Interactive streaming with background traffic.

The purpose of the general purpose tests will be to determine the bandwidth so in this case two different bandwidth limits will be used in iperf3.
The purpose of the interactive streaming is to determine whether or not the current network connection is suited for running interactive streaming traffic. This is determined by looking at the jitter.
Only the tests with interactive streaming traffic will be run also with background traffic. This is to determine if the interactive streaming traffic is sensitive to background traffic when using regular TCP and also when using MPTCP.

### 5.3.2 Regular TCP tests

In this test the regular TCP test descriptions will be given.

#### Test A1.1 - General purpose with a 3 Mbps limit

For the general purpose test a terminal is launched on the client and the following command is issued:

```
1 iperf3 -c 192.38.55.81 -p 60020 -b 3M -l 64k -w 128k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

#### Test A2.1 - Interactive streaming without background traffic

For the interactive streaming test a terminal is launched on the client and the following command is issued:

```
1 iperf3 -c 192.38.55.81 -p 60020 -b 1M -l 8k -w 256k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

#### Test A2.2 - Interactive streaming with background traffic

For the interactive streaming test a terminal is launched on the client and the following command is issued:

```
1 iperf3 -c 192.38.55.81 -p 60020 -b 1M -l 8k -w 256k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

The background traffic is generated by launching two terminals on the client and issuing the following commands:

Downstream:

```
iperf3 -c 192.38.55.81 -p 60022 -R -b 0.5M/20 -l 64k -w 128k -N
```

Upstream:

```
iperf3 -c 192.38.55.81 -p 60022 -R -b 0.5M/20 -l 64k -w 128k -N
```

### 5.3.3 Multipath TCP tests - homogeneous paths

In this section the Multipath TCP test descriptions will be given.

#### Test B1.1 - General purpose with a 3 Mbps limit

For the general purpose test a terminal is launched on the client and the following command is issued:

```
iperf3 -c 192.38.55.81 -p 60020 -b 3M -l 64k -w 128k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

#### Test B1.2 - General purpose with a 5 Mbps limit

For the general purpose test a terminal is launched on the client and the following command is issued:

```
iperf3 -c 192.38.55.81 -p 60020 -b 5M -l 64k -w 128k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

#### Test B2.1 - Interactive streaming without background traffic

For the interactive streaming test a terminal is launched on the client and the following command is issued:

```
iperf3 -c 192.38.55.81 -p 60020 -b 1M -l 8k -w 256k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

#### Test B2.2 - Interactive streaming with background traffic

For the interactive streaming test a terminal is launched on the client and the following command is issued:

```
iperf3 -c 192.38.55.81 -p 60020 -b 1M -l 8k -w 256k -N
```

The results of iperf3 will be saved in a text document and the traffic will be captured with wireshark and saved in a dump file for later analysis.

The background traffic is generated by launching two terminals on the client and issuing the following commands: Downstream:

```
iperf3 -c 192.38.55.81 -p 60022 -R -b 0.5M/20 -l 64k -w 128k -N
```

Upstream:

```
iperf3 -c 192.38.55.81 -p 60022 -R -b 0.5M/20 -l 64k -w 128k -N
```

### 5.3.4 Multipath TCP tests - heterogeneous paths

The same tests as for the homogeneous paths will be performed. These will be named C1.1, C1.2, C2.1 and C2.2.

## 5.4 Regular TCP

In this section the results of a total of 3 tests will be shown. It will include different results such as the average bandwidth as reported by wireshark, the throughput graphs for regular TCP extracted by wireshark as well as the ACK RTT graphs from wireshark.

In Figure 5.6 the setup used in the regular TCP test is shown. The traffic flows from the server in the direction of the client. The gateway creates a throttled connection to the client where the downstream from gateway to the client is 2048 Kbps and the upstream from the client to the gateway is 2048 Kbps.

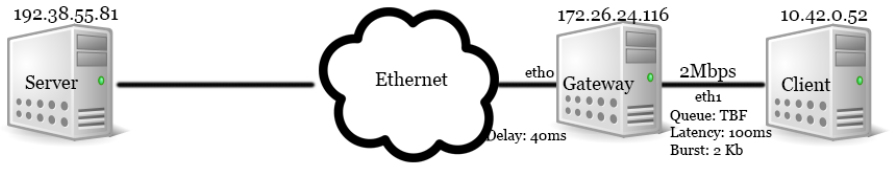


**Figure 5.6:** This figure shows how the test setup for regular TCP is done.

### 5.4.1 Test A1.1 - General purpose (3 Mbps limit) no background traffic

In this test it is expected that the throughput will show no more than a bandwidth of 2 Mbps, since the connection used is a throttled DSL connection with a limit of 2 Mbps. The test will be conducted with a limit in iperf3 of 3 Mbps and this is not to send excessive traffic through the gateway.

#### Throughput graph

As seen in Figure 5.7 the throughput is slightly above 2 Mbps, which is not what was expected from the test and it also corresponds to the average statistics made by capinfos as seen in Listing 5.1. If we take a look at the paragraph 4.2.4 it states that the recommended bandwidth of the Internet connection used for Netflix should be at least 1.5 Mbps and when running this test it has shown a goodput of 1.96Mbps. This should be sufficient for watching streaming videos form Netflix but the quality

will also be poor.

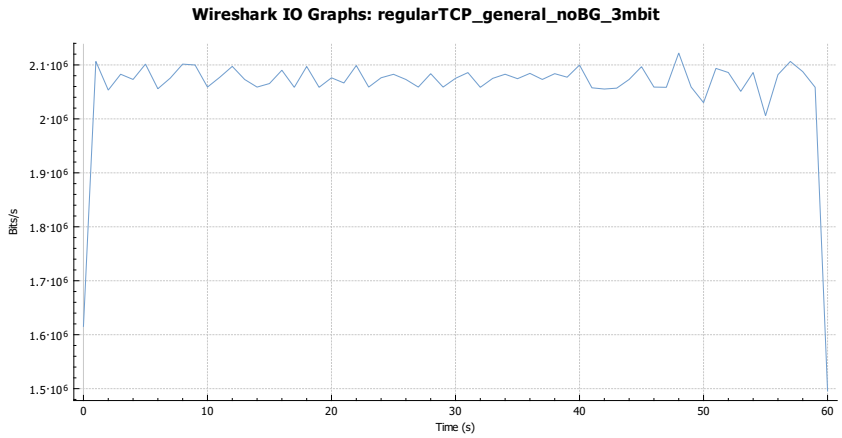


**Figure 5.7:** Graph of the throughput when running general purpose with a 3 Mbps limit without background traffic

## RTT graph

The RTT graph for this test shows, that even though the setup of the gateway has a burst rate of 2 kB and a max wait time for tokens on 100 ms, the RTTs still show somewhat of a queuing tendency. This is displayed in the RTT graph as the packet jumps up and down in RTTs.

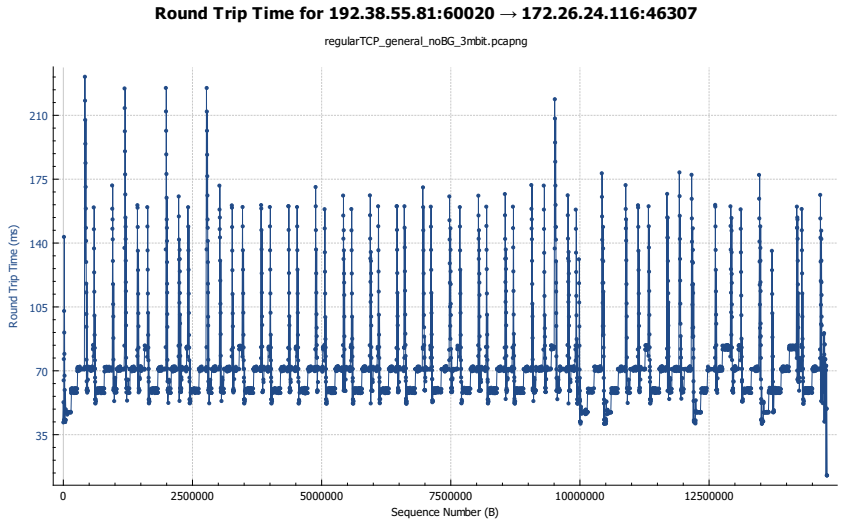


**Figure 5.8:** Graph of the RTT of the TCP stream as reported by wireshark.

### Capinfo

```
Number of packets:   11 k
Data byte rate:      258 kBps
Data bit rate:       2071 kbps
```

**Listing 5.1:** Capinfo for regular TCP general purpose 3 Mbps limit no background traffic.

### Retransmissions

```
62
```

**Listing 5.2:** Retransmissions as reported by wireshark

### Bandwidth from iperf3

As seen in Listing 5.3 the goodput of the test is 1.96 Mbps at the receiver end and if we convert the average throughput as seen in Listing 5.1 we get 2.022 Mbit/sec which is very plausible. The difference between the throughput and the goodput should be the overhead used by the TCP.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[   4]   0.00-60.00  sec  14.1 MBytes  1.97 Mbits/sec   86             sender
[   4]   0.00-60.00  sec  14.0 MBytes  1.96 Mbits/sec                  receiver
```

**Listing 5.3:** iperf3 bandwidth for regular TCP general purpose 3 Mbps limit no background traffic.

So this test shows that when using only one regular TCP connection, that has a downstream bandwidth of 2 Mbps, then the goodput is around 1.96 Mbps. This is sufficient for using e.g. Netflix for streaming videos. It is however the lowest recommended bandwidth required by Netflix as stated in paragraph 4.2.4.

### 5.4.2 Test A2.1 - Interactive streaming (1 Mbps limit) no background traffic

This test is trying to replicate an interactive streaming session on a single regular TCP path. This test is therefore limited to 1 Mbps in iperf3 and this is because we want to catch the behavior of the interactive streaming scenario more than we want to maximize the throughput of the network connection. That is what the general purpose tests is designed for. We want to be able to show, that if the single path TCP has enough capacity, what is the jitter of the network connection and will the connection be suited for interactive streaming conversations.
The expected outcome of the test is that the bandwidth will be around 1 Mbps and that the jitter should be low enough to support an interactive streaming conversation as explained in subsubsection 5.2.

#### Throughput graph

As seen in Figure 5.9 the throughput for this test is very close to 1 Mbps. This is the expected outcome and even though it fluctuates the goodput should be 1 Mbps. The fluctuation of the graph may be a result of the way iperf3 limits the bandwidth.

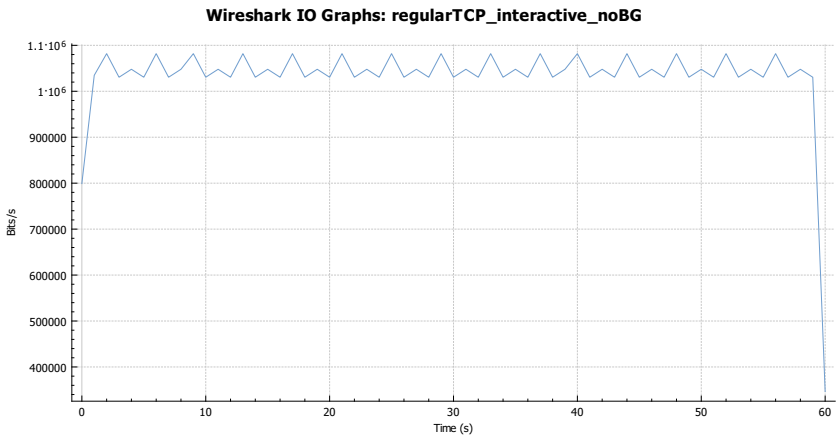


**Figure 5.9:** Graph of the throughput when running interactive streaming with a 1 Mbps limit without background traffic.

### RTT graph

As seen in Figure 5.11 the RTTs show a very fluctuating pattern that can be explained by the queue in the gateway. The RTTs show a sign of queuing discipline since the queue is a token bucket filter the rate of the traffic through it, will depend on the rate that the tokens arrive. The RTT graph shows the same tendency through the entire duration of the test, which aligns with the fact that the traffic is passing through the queue in an even fashion.

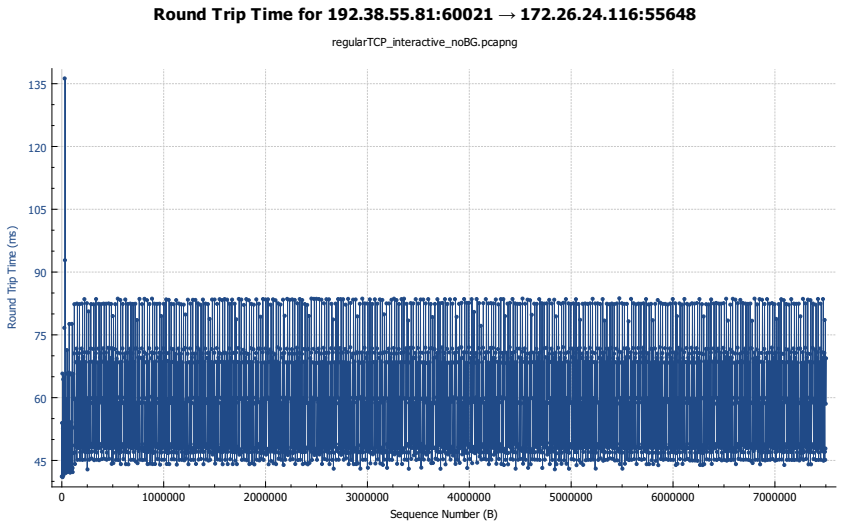


**Figure 5.10:** Graph of the RTT of the TCP stream as reported by wireshark.

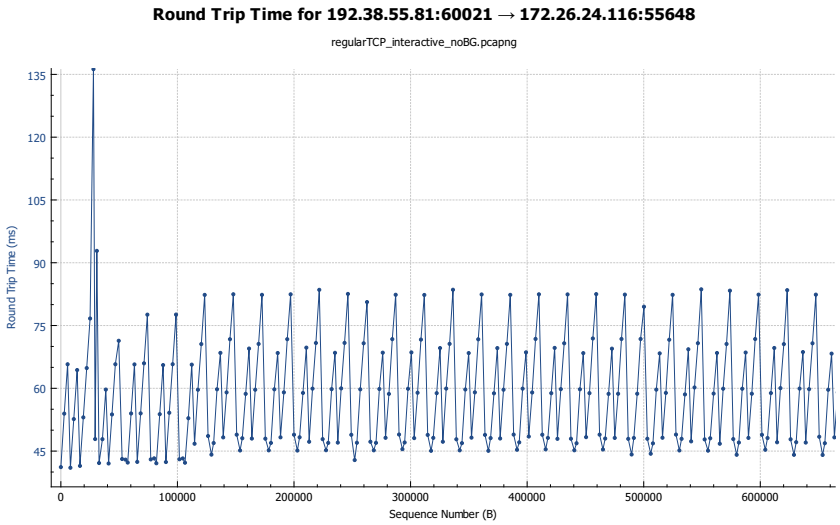


**Figure 5.11:** Graph of the RTT of the TCP stream as reported by wireshark. This is zoomed to show the queuing tendency better.

## Capinfo

```
Number of packets:   5579
Data byte rate:      130 kBps
Data bit rate:       1040 kbps
```

**Listing 5.4:** Capinfo for regular TCP general purpose 3 Mbps limit no background traffic.

## Retransmissions

```
0
```

**Listing 5.5:** Retransmissions as reported by wireshark

## Bandwidth from iperf3

As seen in Listing 5.6 the reported goodput is exactly 1 Mbps, which is the expected goodput. iperf3 has reported two retransmissions and by looking at Figure 5.11 there are two spikes in the beginning of the test that could be these 2 retransmissions.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[   4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec    2             sender
[   4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec                  receiver
```

**Listing 5.6:** iperf3 bandwidth for regular TCP general purpose 3 Mbps limit no background traffic.

### Jitter

To calculate the jitter, the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.2}$$

The mean and variance of the one way delay is:

$$mean(delay) = 29.3901\ ms \tag{5.3}$$

$$var(delay) = 39.9737\ ms \tag{5.4}$$

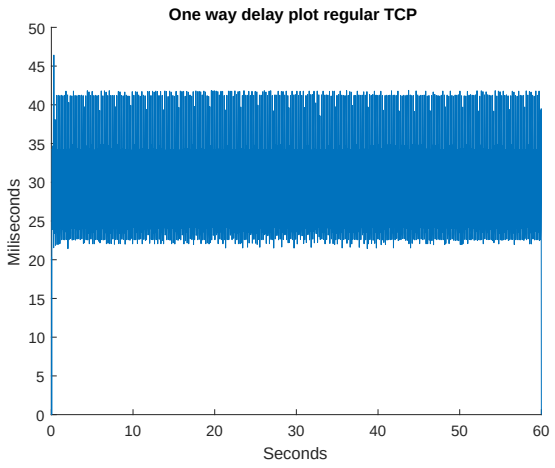


**Figure 5.12:** This is a plot of the one way delay.

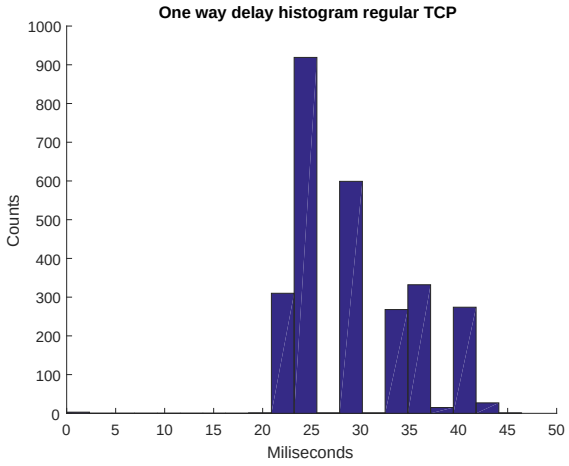


**Figure 5.13:** This is a histogram of the one way delay.

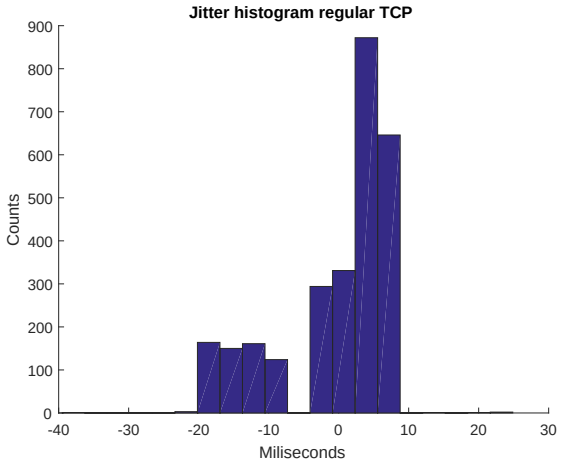


**Figure 5.14:** This is a histogram of the jitter.

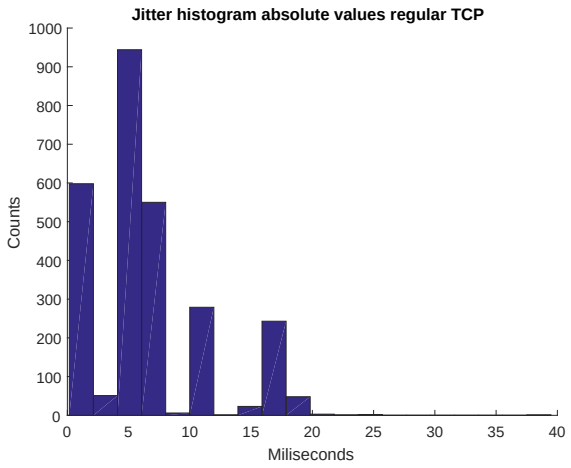


**Figure 5.15:** This is a histogram of the absolute values in the jitter.

The mean and variance of the jitter is:

$$mean(jitter) = 6.5598\ ms \tag{5.5}$$

$$var(jitter) = 22.1216\ ms \tag{5.6}$$

In Figure 5.14 we see that there are quite a few samples with negative jitter but since the mean and the variance of the jitter are relatively small this will only have a small impact on a interactive streaming conversation. The variance of the jitter does not exceed 100 ms and the one way delay never exceeds 50 ms. So when using this setup for a regular TCP connection, it would be possible to use it for interactive streaming purposes as long as the connection has enough bandwidth.

### 5.4.3 Test A2.2 - Interactive streaming (1 Mbps limit) with background traffic

This test is almost the same as test A2.1 but with added background traffic. This is to test if the interactive streaming setup is sensitive to background traffic in the sense of the jitter. The expected outcome is that the goodput will still be 1 Mbps as in the previous test, but the throughput graph should reflect that the data stream gets more fragmented in the presence of background traffic.

#### Throughput graph

Figure 5.16 shows that the throughput of the test is still around 1 Mbps but now the tendency of the graph is much more uneven. The graph shows that the throughput now fluctuates a lot more than in Test A2.1. This must be due to the added background traffic.

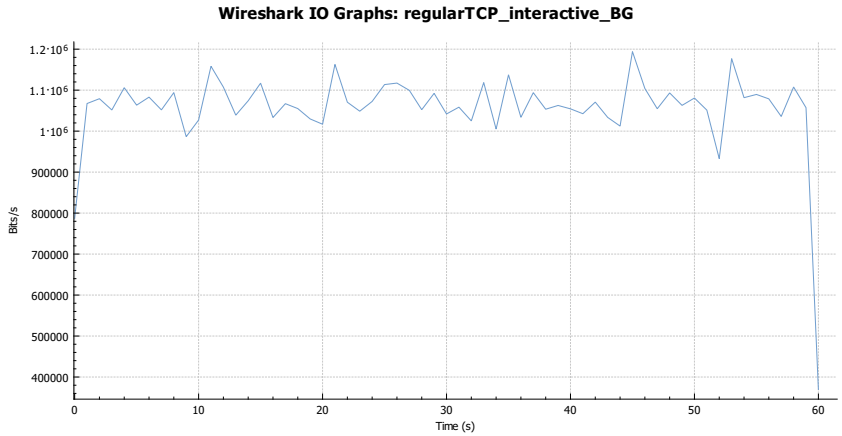


**Figure 5.16:** Graph of the throughput when running interactive streaming with a 1 Mbps limit with background traffic.

#### RTT graph

Figure 5.17 shows that the RTTs of the data stream now is more uneven that in the test with no background traffic. It has a lot more spikes than in Test A2.1 but the base tendency of the graph is still that the RTTs jump from around 40 ms to around 85 ms. If we look at Listing 5.9 iperf3 reports 71 retransmissions. This corresponds to the spikes in the RTT graph so the conclusion here is that when there are large

spikes in the RTT graph it is due to reaching the maximum delay since the queue is being filled and this causes packets to be subjected to the maximum latency when traversing through the queue.

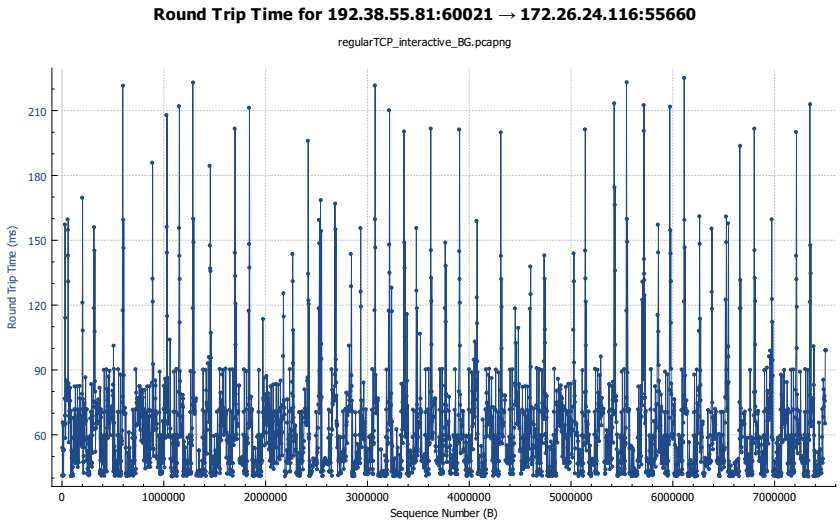


**Figure 5.17:** Graph of the RTT of the TCP stream as reported by wireshark.

### Capinfo

```
Number of packets:   6567
Data byte rate:      132 kBps
Data bit rate:       1061 kbps
```

**Listing 5.7:** Capinfo for regular TCP general purpose 3 Mbps limit no background traffic.

### Retransmissions

```
50
```

**Listing 5.8:** Retransmissions as reported by wireshark

### Bandwidth from iperf3

Listing 5.9 shows a goodput of 1000 Kbps, which is very close to 1 Mbps. Even with the added background traffic the 2 Mbps network connection is still able to handle all the traffic, so the goodput for the interactive streaming is virtually unchanged compared to the test without background traffic.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[   4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec   71             sender
[   4]   0.00-60.00  sec  7.15 MBytes  1000 Kbits/sec                  receiver
```

**Listing 5.9:** iperf3 bandwidth for regular TCP general purpose 3 Mbps limit no background traffic.

### Jitter

To calculate the jitter, the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.7}$$

The mean and variance of the one way delay is:

$$mean(delay) = 30.4546\ ms \tag{5.8}$$

$$var(delay) = 75.3203\ ms \tag{5.9}$$

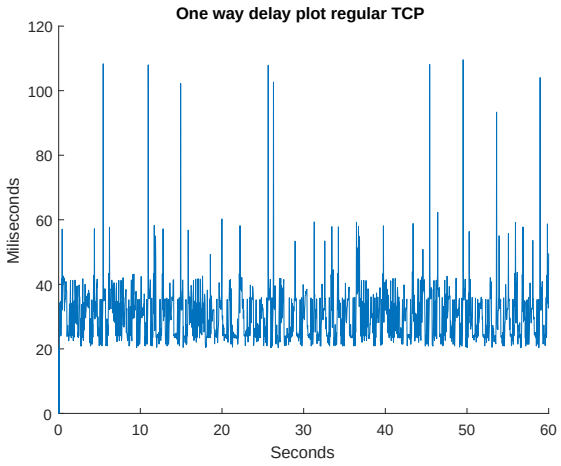


**Figure 5.18:** This is a plot of the one way delay.

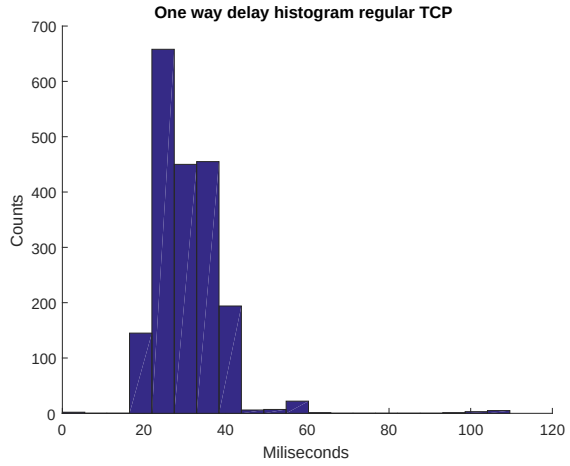


**Figure 5.19:** This is a histogram of the one way delay.

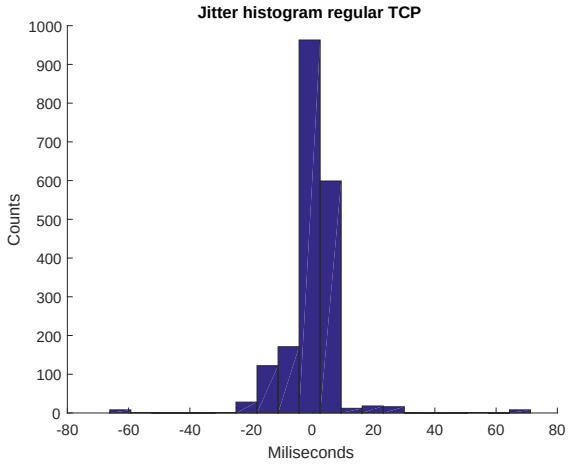


**Figure 5.20:** This is a histogram of the jitter.

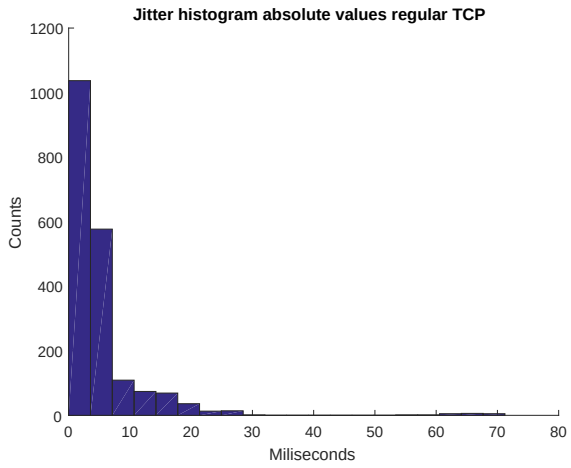


**Figure 5.21:** This is a histogram of the absolute values in the jitter..

The mean and variance of the jitter is:

$$mean(jitter) = 5.1953\ ms \tag{5.10}$$

$$var(jitter) = 58.3802\ ms \tag{5.11}$$

By adding background traffic to the interactive streaming traffic, it shows that in the throughput graph, it becomes much more unstable over the duration of the test. There are a lot more fluctuations than in Test A2.1, where there was no background traffic. In Figure 5.20 we see that the jitter is mostly positive and it has a mean of approximately 5 ms so the mean jitter is actually a bit lower than in the previous test. In Figure 5.18 we see that there are 8 samples which exceed 100 ms. Since we established earlier that, when the one way delay is between 100 ms and 125 ms it starts to become problematic, to have an interactive streaming conversation, since the voice stream will begin to show signs of stutter. In this case since the samples does not occur in a back-to-back form it will only have a little, but noticeable impact on a voice stream. The variance of the jitter when running with background traffic is almost triple of what it was when having no background traffic. So in conclusion the network connection can still be used for interactive streaming purposes as long as there is sufficient bandwidth and that the background traffic is not more intense than used in this test, however the variance of the jitter is sufficiently large to cause some problems e.g. stutter in the audio.

## 5.5 MPTCP homogeneous paths

In this section the results of a total of 3 tests will be shown. It will include different results such as the average bandwidth as reported by wireshark, the throughput graphs for Multipath TCP extracted by wireshark as well as the ACK RTT graphs from wireshark.

In Figure 5.59 the setup used in the Multipath TCP DSL+DSL test is shown. The traffic flows from the server in the direction of the client. The gateway creates two throttled connections to the client where the downstream, on each connection, from gateway to the client is 2048 Kbps, and the upstream from the client to the gateway is 2048 Kbps.

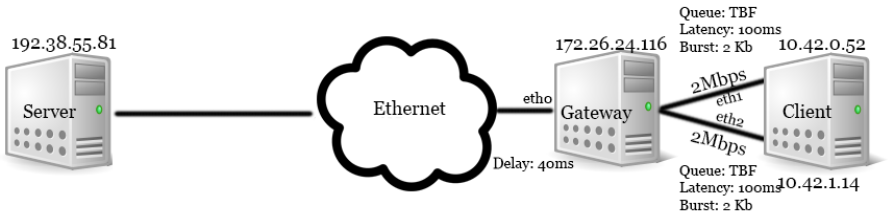


**Figure 5.22:** This figure shows how the test setup for Multipath TCP with 2 DSL connections is done.

### 5.5.1 Test B1.1 - MPTCP DSL+DSL general purpose 3 Mbps limit without background traffic

In this test the Multipath TCP will be used with homogeneous paths namely two throttled DSL connections. The general purpose will be limited to 3 Mbps in iperf3. The expected outcome is that with the default scheduler in the MPTCP the traffic should be split almost evenly between the two subflows, since the default scheduler uses the subflow with the lowest RTT first. It should show that the throughput should be around 3 Mbps and the RTT graphs on each subflow should be somewhat similar.

### Throughput graphs

If we look at Figure 5.23, where the blue line is flow 1 and the green line is flow 2, we see that both flows have almost the same behavior and they are both slightly above 1.5 Mbps. The reason why they are slightly above 1.5 Mpbs is probably because of the added overhead that comes with Multipath TCP.

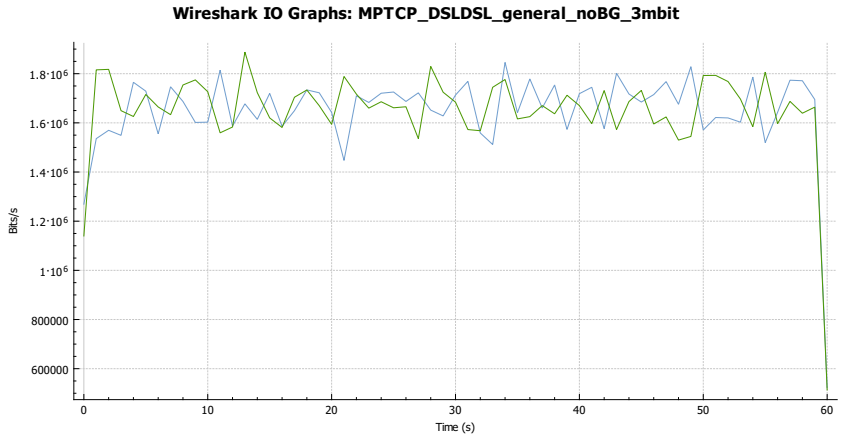


**Figure 5.23:** Graph of the throughput for general purpose with a 3 Mbps limit when running a MPTCP setup with 2 DSL connections. Blue is flow1, green is flow 2.

### RTT graph

If we look at Figure 5.24 and Figure 5.25 we see almost two identical graphs, which was what we expected. The traffic is split almost even between the two subflows, and this will cause the RTT graphs to resemble each other. Now in both the graphs we see a lot of spikes that are as high as what we observed in Test A2.1 and Test A2.2. While the spikes in these two tests was caused by retransmissions and the queue being filled because of this, the spikes in Figure 5.24 and Figure 5.25 are not cause by this. This is confirmed by looking at Listing 5.14, where iperf3 does not report any retransmissions. The spikes in Figure 5.24 and Figure 5.25 can still be caused by the queue and it might also be because of the TSO settings in Linux or a combination of both.

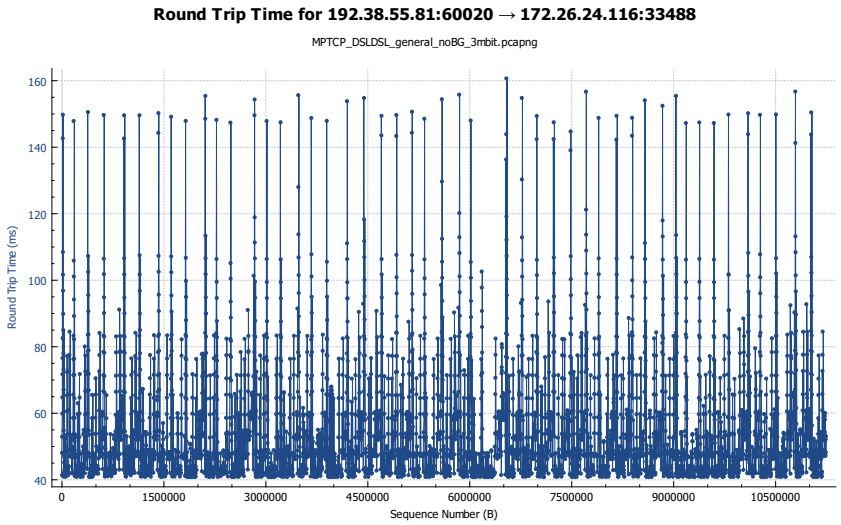


**Figure 5.24:** Graph of the RTT of the TCP stream for general purpose with a 3 Mbps limit as reported by wireshark. This is for flow 1.

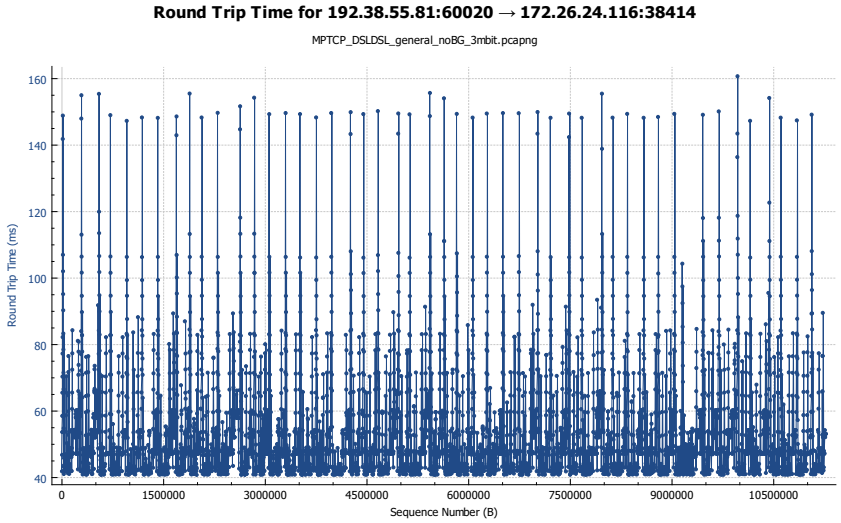


**Figure 5.25:** Graph of the RTT of the TCP stream for general purpose with a 3 Mbps limit as reported by wireshark. This is for flow 2.

## Capinfo

```
1 Number of packets:   15 k
2 Data byte rate:      208 kBps
3 Data bit rate:       1664 kbps
```

**Listing 5.10:** Capinfo for MPTCP DSL+DSL general purpose 3 Mbps limit with no background traffic. This is for flow 1.

```
Number of packets:   15 k
Data byte rate:      208 kBps
Data bit rate:       1669 kbps
```

**Listing 5.11:** Capinfo for MPTCP DSL+DSL general purpose 3 Mbps limit with no background traffic. This is for flow 2.

### Retransmissions

```
45
```

**Listing 5.12:** Retransmissions as reported by wireshark. This is for flow 1.

```
46
```

**Listing 5.13:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

If we take a look at Listing 5.14 we see that the goodput of the test is 3 Mbps, which was the limit used in iperf3. So the Multipath TCP is capable of aggregating the traffic on both of the subflows to achieve a higher throughput than when only using one path. In Listing 5.10 and Listing 5.11 we see that there are only 5 Kbps difference on the throughput on the two subflows, which is also what we expected.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[   4]   0.00-60.00  sec  21.5 MBytes  3.00 Mbits/sec    0             sender
[   4]   0.00-60.00  sec  21.5 MBytes  3.00 Mbits/sec                  receiver
```

**Listing 5.14:** iperf3 bandwidth for MPTCP general purpose 3 Mbps limit no background traffic.

### 5.5.2 Test B1.2 - MPTCP DSL+DSL general purpose 5 Mbps limit without background traffic

In this test iperf3 will be set to a limit of 5 Mbps, which will then generate more traffic than the two combined subflows is able to handle. The expected outcome is that the traffic will still be almost evenly distributed on the two subflows, given that the RTTs of both subflows are identical. The goodput of the test should be very close to 4 Mbps, but it is also expected that some retransmissions might occur, since the queue on the gateway is relatively small. The RTT graphs should be almost identical on both of the subflows, given that the traffic is expected to be distributed evenly on the two subflows.

#### Throughput graphs

If we look at Figure 5.26, where flow 1 is the blue line and flow 2 is the green line, we see that the flows are almost on top of each other, and despite some fluctuations the throughput looks more or less identical, which is because the setup is using homogeneous paths.

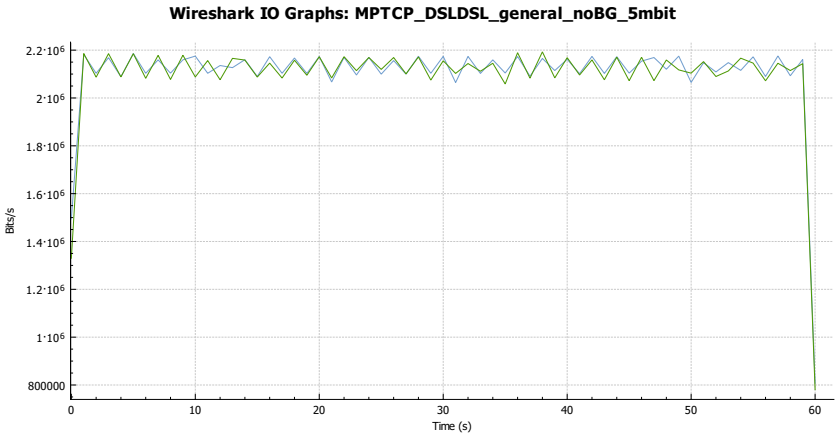


**Figure 5.26:** Graph of the throughput for general purpose with a 5Mbps limit when running a MPTCP setup with 2 DSL connections. Blue is flow1, green is flow 2.

#### RTT graph

Looking at Figure 5.27 and Figure 5.28 shows the same behavior as with the throughput. The two graphs have clear indications that a queue is present at the gateway.

The spikes that occur from 40 ms to 80 ms on the RTT graphs are the packets moving through the queue, where the large spikes is delayed acks. There are retransmissions on both flows and this causes the tendency, where the queue is handling even more packets due to the retransmissions of the ones that was dropped.

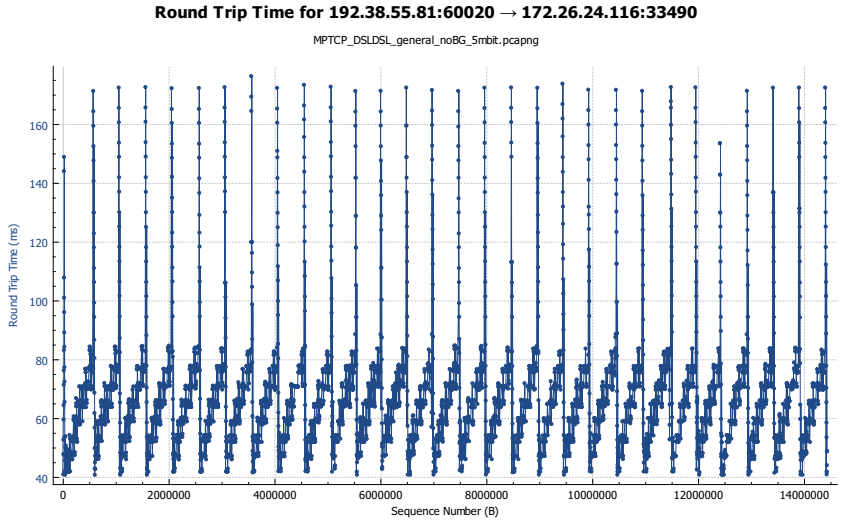


**Figure 5.27:** Graph of the RTT of the TCP stream for general purpose with a 5 Mbps limit as reported by wireshark. This is for flow 1.

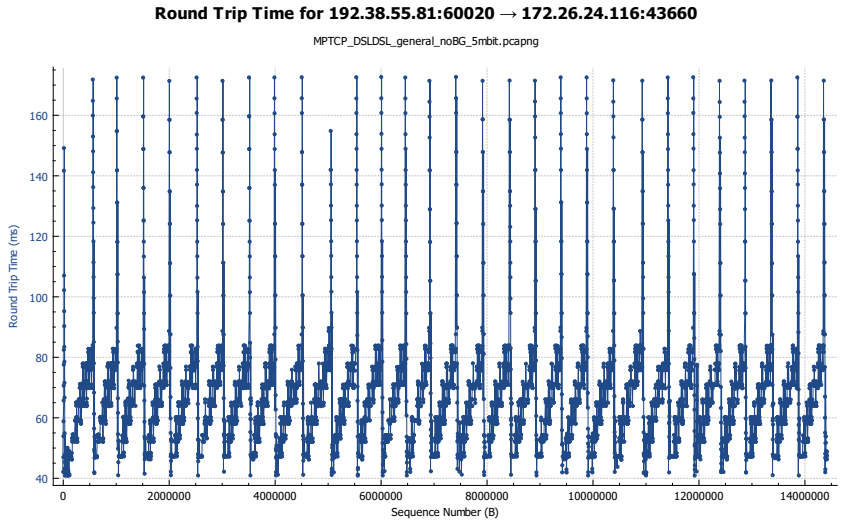


**Figure 5.28:** Graph of the RTT of the TCP stream for general purpose with a 5 Mbps limit as reported by wireshark. This is for flow 2.

### Capinfo

```
Number of packets:   19 k
Data byte rate:      265 kBps
Data bit rate:       2123 kbps
```

**Listing 5.15:** Capinfo for MPTCP DSL+DSL general purpose 5Mbps limit with no background traffic. This is for flow 1.

```
Number of packets:   18 k
Data byte rate:      264 kBps
Data bit rate:       2119 kbps
```

**Listing 5.16:** Capinfo for MPTCP DSL+DSL general purpose 5Mbps limit with no background traffic. This is for flow 2.

### Retransmissions

```
28
```

**Listing 5.17:** Retransmissions as reported by wireshark. This is for flow 1.

```
29
```

**Listing 5.18:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

The goodput listed in Listing 5.19 is close to 4Mbps as we expected. The difference from 4 Mbps is 0.16 Mbps, which might be explained by the extra header added by using the Multipath TCP.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  27.5 MBytes  3.85 Mbits/sec    0             sender
[  4]   0.00-60.00  sec  27.4 MBytes  3.84 Mbits/sec                  receiver
```

**Listing 5.19:** iperf3 bandwidth for MPTCP general purpose 5 Mbps limit no background traffic.

In this test we showed that when generating a higher throughput than the combined subflows can handle, the recorded goodput is very close to 4 Mbps. The traffic is almost evenly split among the two subflows, which is what we expected since this setup consists of two subflows with similar characteristics. The test also show that when using Multipath TCP it is possible to get bandwidth aggregation.

### 5.5.3 Test B2.1 - MPTCP DSL+DSL interactive streaming 1 Mbps limit without background traffic

This test is with the interactive streaming traffic and is to determine what happens to the jitter, when distributing the traffic on two subflows. The expected outcome is that the goodput of the test will be 1 Mbps and that it will be split evenly on the two subflows. The RTT graphs should show similar behavior. The jitter should not be that affected by the traffic being split on two subflows, since the paths are homogeneous and has almost identical RTTs.

#### Throughput graphs

Figure 5.29 shows that the throughput has a larger variance than when running the general purpose traffic. This effect can be due to the scheduler of the Multipath TCP and because the RTTs are almost identical, the scheduler distributes the traffic on either flow 1 or flow 2. Since the RTTs on the two flows change by a few ms all the time this behavior is to be expected.

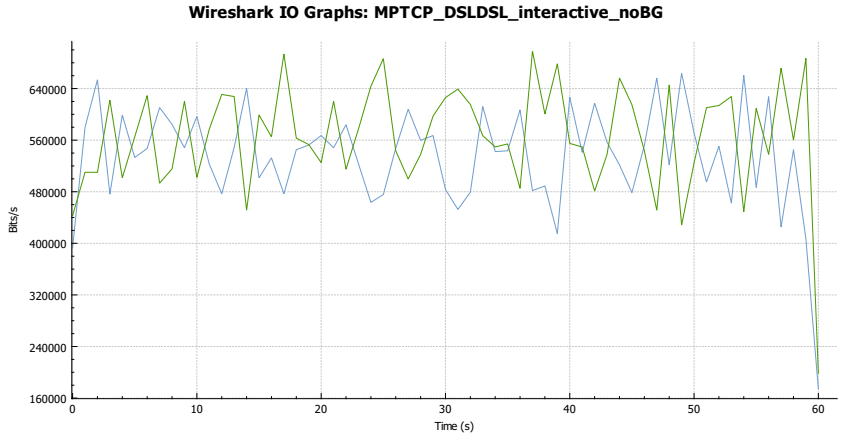


**Figure 5.29:** Graph of the throughput for interactive streaming with a 1 Mbps limit when running a MPTCP setup with 2 DSL connections. Blue is flow1, green is flow 2.

#### RTT graph

If we look at Figure 5.30 and Figure 5.31 we see that they look almost similar. The small jumps from 40 ms to 80 ms is still due to the nature of the queue at the gateway. The large spikes from 40 ms to 280 ms is these delayed ACKs.

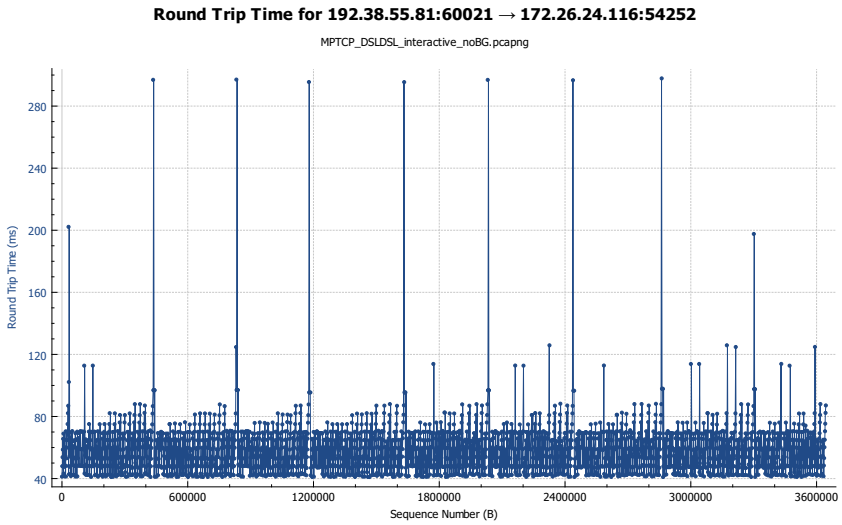


**Figure 5.30:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 1.

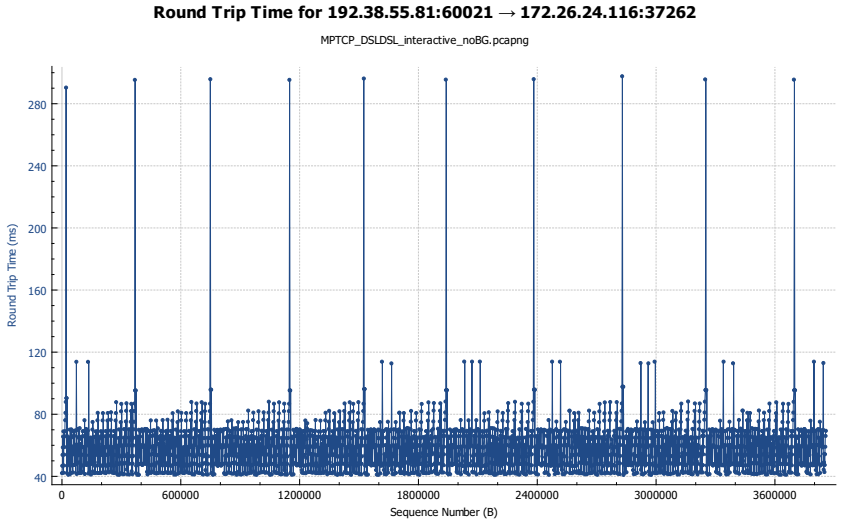


**Figure 5.31:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 2.

## Capinfo

```
Number of packets:   5203
Data byte rate:      67 kBps
Data bit rate:       540 kbps
```

**Listing 5.20:** Capinfo for MPTCP DSL+DSL interactive streaming 1 Mbps limit without background traffic. This is for flow 1.

```
Number of packets:   5457
Data byte rate:      71 kBps
Data bit rate:       572 kbps
```

**Listing 5.21:** Capinfo for MPTCP DSL+DSL interactive streaming 1 Mbps limit without background traffic. This is for flow 2.

### Retransmissions

```
8
```

**Listing 5.22:** Retransmissions as reported by wireshark. This is for flow 1.

```
9
```

**Listing 5.23:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

Listing 5.24 shows that the goodput of the test is 1 Mbps. This aligns with the fact that the two subflows have more than 1 Mbps combined. As we saw in the capinfo statistics the traffic is not evenly distributed but very close to it.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec    0             sender
[  4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec                  receiver
```

**Listing 5.24:** iperf3 bandwidth for MPTCP interactive streaming 1 Mbps limit without background traffic.

### Jitter

To calculate the jitter the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.12}$$

The mean and variance of the one way delay is:

$$mean(delay) = 28.9289\ ms \quad (5.13)$$
$$var(delay) = 39.9714\ ms \quad (5.14)$$

**Figure 5.32:** Mean and variance flow 1.

$$mean(delay) = 29.1133\ ms \quad (5.15)$$
$$var(delay) = 39.9963\ ms \quad (5.16)$$

**Figure 5.33:** Mean and variance flow 2.

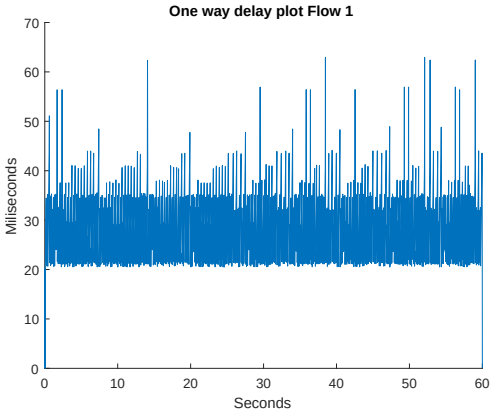


**Figure 5.34:** One way delay plot flow 1.

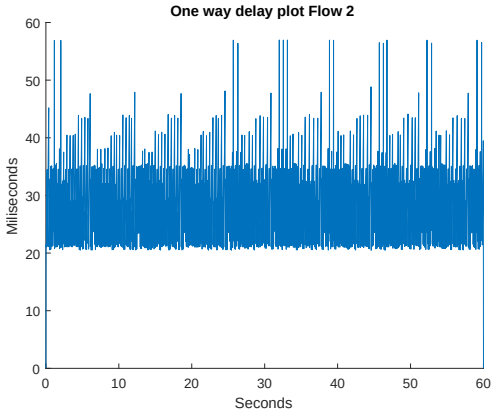


**Figure 5.35:** One way delay plot flow 2.

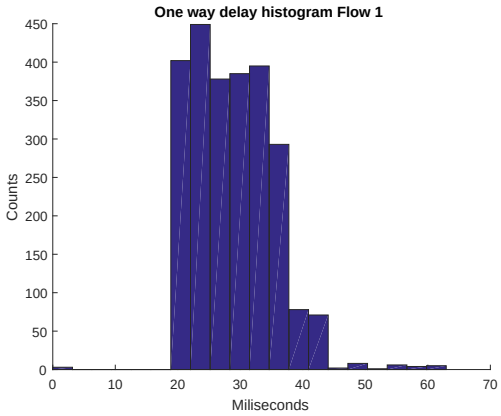


**Figure 5.36:** One way delay histogram flow 1.

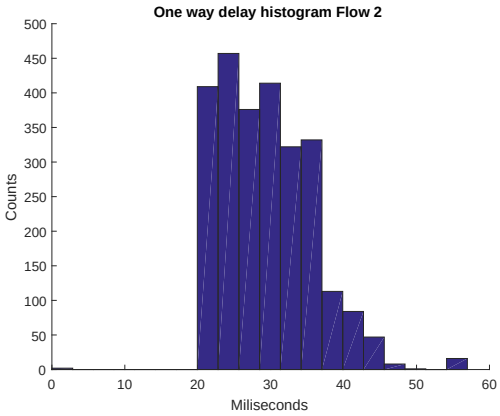


**Figure 5.37:** One way delay histogram flow 2.

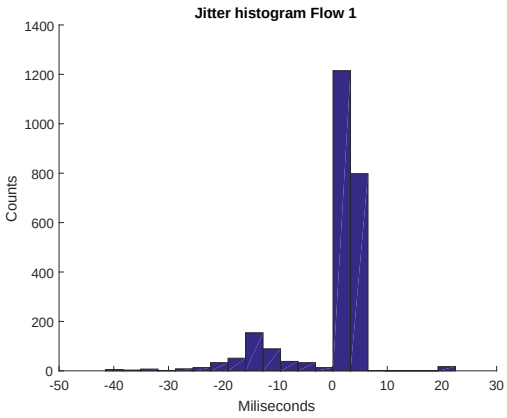


**Figure 5.38:** Jitter histogram flow 1.

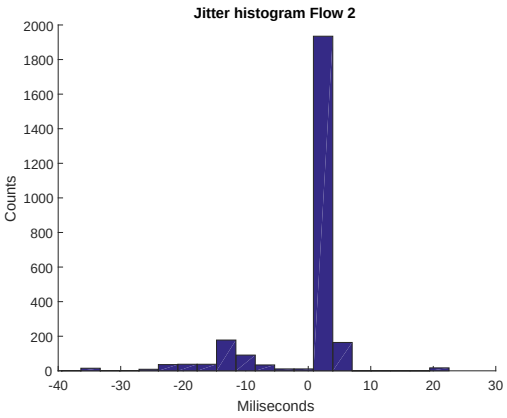


**Figure 5.39:** Jitter histogram flow 2.

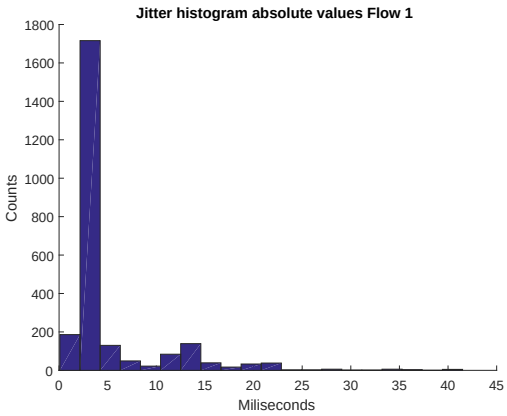


**Figure 5.40:** Abs jitter histogram flow 1.

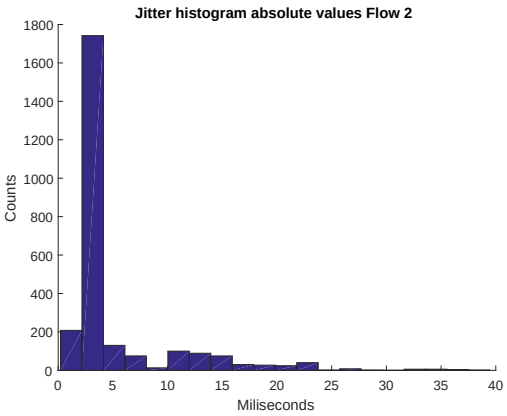


**Figure 5.41:** Abs jitter histogram flow 2.

The mean and variance of the jitter is:

$$mean(jitter) = 5.082\ ms \quad (5.17)$$
$$var(jitter) = 28.535\ ms \quad (5.18)$$

**Figure 5.42:** Mean and variance flow 1.

$$mean(jitter) = 5.180\ ms \quad (5.19)$$
$$var(jitter) = 28.993\ ms \quad (5.20)$$

**Figure 5.43:** Mean and variance flow 2.

If we look at both Figure 5.34 and Figure 5.34 we see that the spikes on both flow 1 and flow 2 are lower than compared to Test A2.1, where the spikes peaked at

approximately 110 ms. Here the maximum one way delay does not exceed 70 ms on either flow 1 or flow 2. If we look at Figure 5.38 and Figure 5.39 the jitter should be more or less the same for both of the subflows. The mean jitter for both paths are approximately 5 ms and the variance is approximately 30 ms. Comparing this with Test A2.1 the jitter is slightly lower when using this Multipath TCP setup with homogeneous paths. The variance is slightly higher than when using the regular TCP connection, but since the spikes in the one way delay never exceeds 70 ms and the variance is below 100 ms, the network can be used for interactive streaming purposes.

### 5.5.4 Test B2.2 - MPTCP DSL+DSL interactive streaming 1 Mbps limit with background traffic

This test will be the same as the previous test just with added background traffic. The expected outcome is that the goodput should still be 1 Mbps and the throughput graphs should look similar to the ones in test Test B2.1. The RTT graphs should show a little more variance than in the test without background traffic, simply because the queue needs to process the background traffic as well.

#### Throughput graphs

Looking at Figure 5.44 we see the expected behavior and the two flows look very similar to the throughput graphs of the previous tests. The throughput graphs show that the traffic is almost evenly distributed on the subflows.

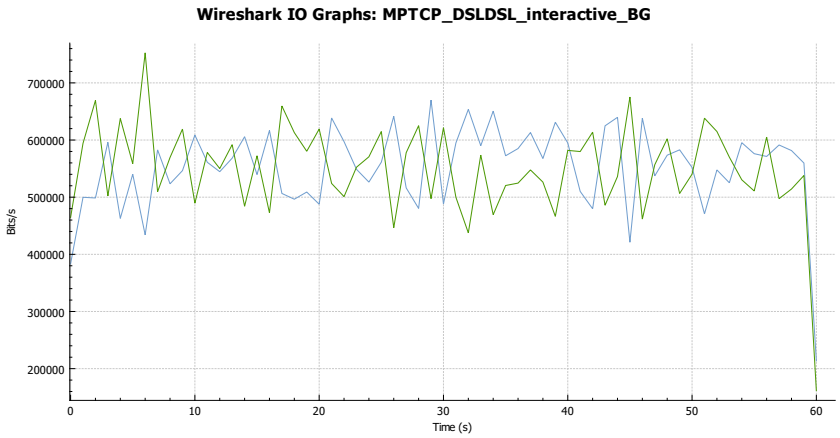


**Figure 5.44:** Graph of the throughput for interactive streaming with a 1 Mbps limit when running a MPTCP setup with 2 DSL connections. Blue is flow1, green is flow 2.

#### RTT graph

When looking at Figure 5.45 and Figure 5.46, they show that there are more variance between the samples compared to the test without background traffic. This is because the queue is handling both the background traffic and the interactive streaming traffic so in some cases the RTT will be slightly higher or lower. The large spikes is still due to the delayed ACKs.

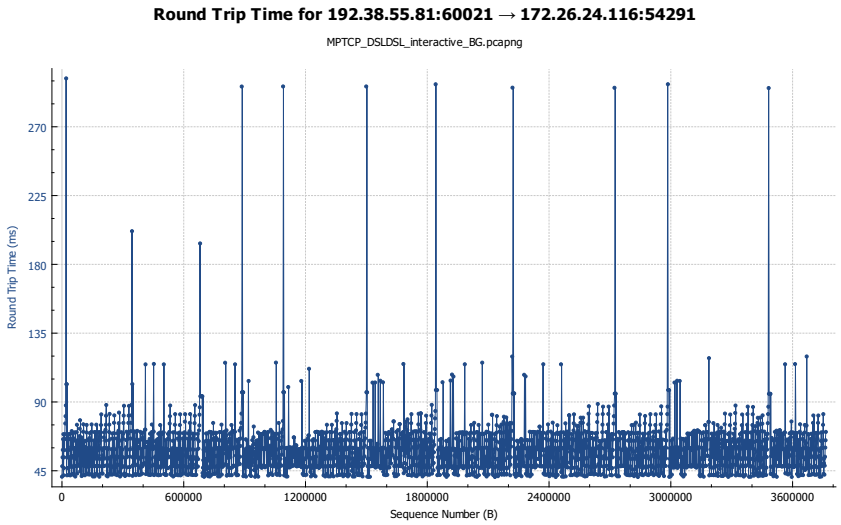


**Figure 5.45:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 1.

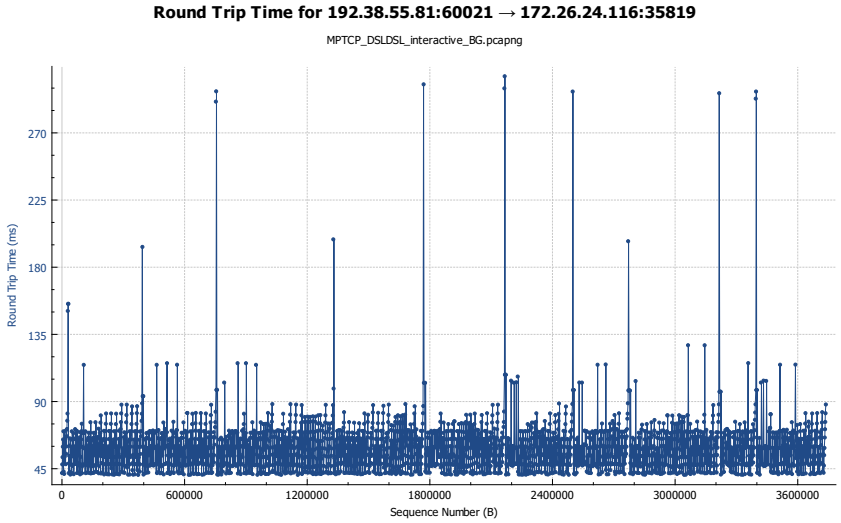


**Figure 5.46:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 2.

## Capinfo

```
Number of packets:   5255
Data byte rate:      69 kBps
Data bit rate:       557 kbps
```

**Listing 5.25:** Capinfo for MPTCP DSL+DSL interactive streaming 1 Mbps limit with background traffic. This is for flow 1.

```
Number of packets:   5349
Data byte rate:      69 kBps
Data bit rate:       555 kbps
```

**Listing 5.26:** Capinfo for MPTCP DSL+DSL interactive streaming 1 Mbps limit with background traffic. This is for flow 2.

### Retransmissions

```
9
```

**Listing 5.27:** Retransmissions as reported by wireshark. This is for flow 1.

```
5
```

**Listing 5.28:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

As see in Listing 5.25 and Listing 5.26 the average data bit rate is only differing with 2 Kbps so the traffic in this test is closer to evenly distributed than it was in the test without background traffic. The difference was 32 Kbps in the previous test so the magnitude of the difference is still very low and it is probaly due to the nature of the scheduler used in Multipath TCP.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[   4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec    0             sender
[   4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec                  receiver
```

**Listing 5.29:** iperf3 bandwidth for MPTCP interactive streaming 1 Mbps limit with background traffic.

### Jitter

To calculate the jitter the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.21}$$

The mean and variance of the one way delay is:

$$mean(delay) = 29.0500\ ms \tag{5.22}$$
$$var(delay) = 41.9535\ ms \tag{5.23}$$

**Figure 5.47:** Mean and variance flow 1.

$$mean(delay) = 29.3430\ ms \tag{5.24}$$
$$var(delay) = 59.5477\ ms \tag{5.25}$$

**Figure 5.48:** Mean and variance flow 2.

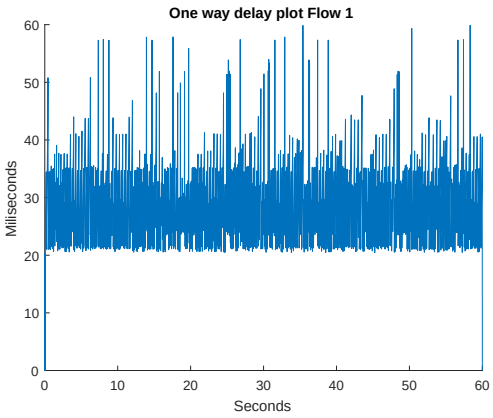


**Figure 5.49:** One way delay plot flow 1.

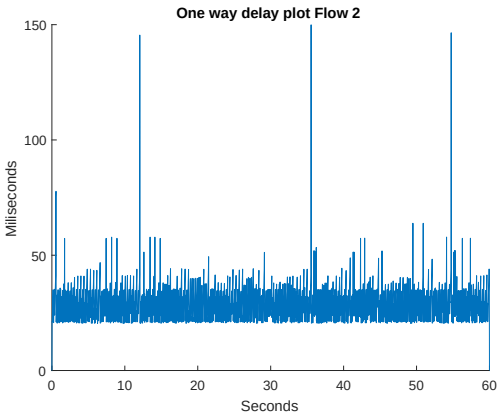


**Figure 5.50:** One way delay plot flow 2.

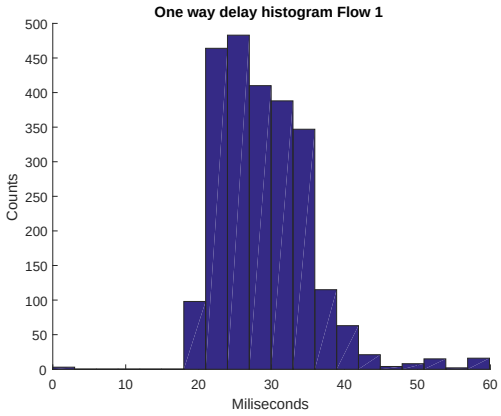


**Figure 5.51:** One way delay histogram flow 1.

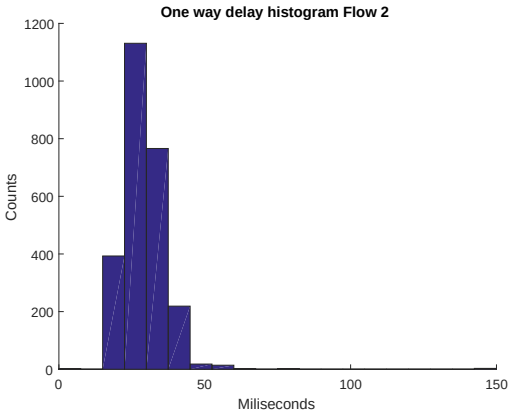


**Figure 5.52:** One way delay histogram flow 2.

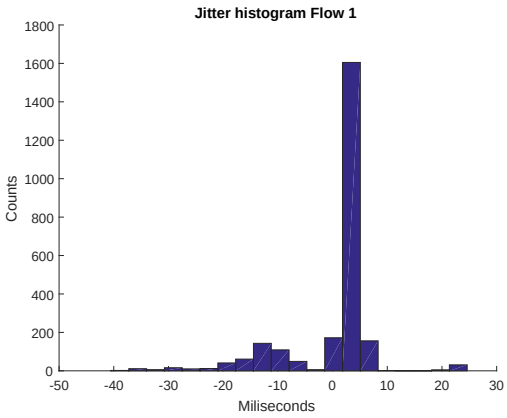


**Figure 5.53:** Jitter histogram flow 1.

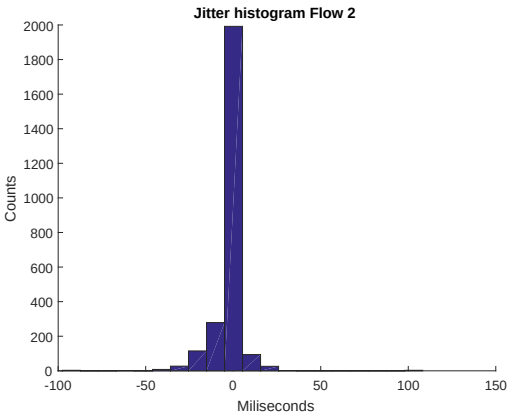


**Figure 5.54:** Jitter histogram flow 2.

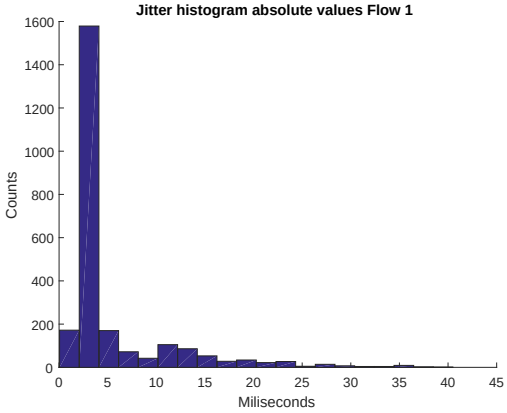


**Figure 5.55:** Abs jitter histogram flow 1.

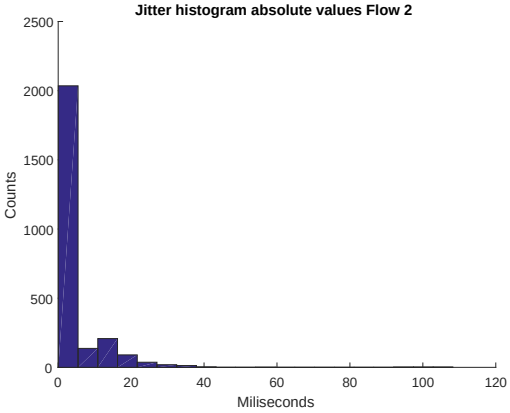


**Figure 5.56:** Abs jitter histogram flow 2.

The mean and variance of the jitter is:

$$mean(jitter) = 5.488\ ms \quad (5.26)$$
$$var(jitter) = 33.777\ ms \quad (5.27)$$

**Figure 5.57:** Mean and variance flow 1.

$$mean(jitter) = 5.459\ ms \quad (5.28)$$
$$var(jitter) = 54.654\ ms \quad (5.29)$$

**Figure 5.58:** Mean and variance flow 2.

Looking at Figure 5.49 and Figure 5.50 we see that the tendency of both graphs are the same except for 3 large spikes on flow 2. These spikes are almost 150 ms

so it would cause noticeable changes in an interactive streaming conversation. The mean jitter is approximately the same for both paths, namely 5.5 ms however the variance in the jitter on flow 2 is almost double of flow 1. This shows that the variance in the jitter is very sensitive to large spikes in the one way delay. So we can conclude that large spikes in the one way delay, even if it is only a few spikes, will give a significantly higher variance in the jitter. Too many of these spikes will make the network connection unusable for interactive streaming purposes. In this case the traffic is split almost evenly between the subflows but when the variance in the jitter on flow 2 is almost 60 ms it cannot be used for interactive streaming purposes. Comparing these results to the previous test this behavior can be caused by the added background traffic. So when using this setup we can conclude that if the network is used for interactive streaming purposes it becomes sensitive to background traffic.

## 5.6 MPTCP heterogeneous paths

In this section the results of a total of 4 tests will be shown. It will include some different results such as the average bandwidth as reported by wireshark, the throughput graphs for Multipath TCP extracted by wireshark as well as the ACK RTT graphs from wireshark.

In Figure 5.59 the setup used in the Multipath TCP DSL + 4G test is shown. The traffic flows from the server in the direction of the client. The gateway creates one throttled connections to the client where the downstream from gateway to the client is 2048 Kbps, and the upstream from the client to the gateway is 2048 Kbps. An android phone is used as a hotspot to share a 4G connection to the client as the second subflow.

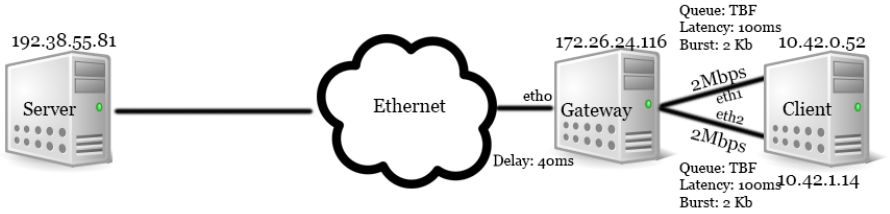


**Figure 5.59:** This figure shows how the test setup for Multipath TCP with 1 DSL connection and 1 4G connection is done.

### 5.6.1 Test C1.1 - MPTCP DSL + 4G general purpose 3 Mbps limit without background traffic

In this test a 3 Mbps limit will be used in iperf3. This is done to determine how the Multipath TCP works, when there is enough capacity on the subflows combined. The expected outcome is that the traffic will be split between the two subflows but it might be an uneven split, since the scheduler in the Multipath TCP takes the subflow with the lowest RTT first until this gets congested.

#### Throughput graphs

As we see in Figure 5.60 the throughput on the 4G flow path, it is the blue line, is fluctuating around 3 Mbps so it seems that the scheduler, in this case, favors the 4G path. This must be because the path has the lowest RTT and does not get congested. At around 44 seconds some of the traffic is routed via the DSL path. This is shown as the green line increasing to 0.6 Mbps. It aligns with the fact that the blue line at the same time drops from approximately 3 Mbps to 2.4 Mbps.

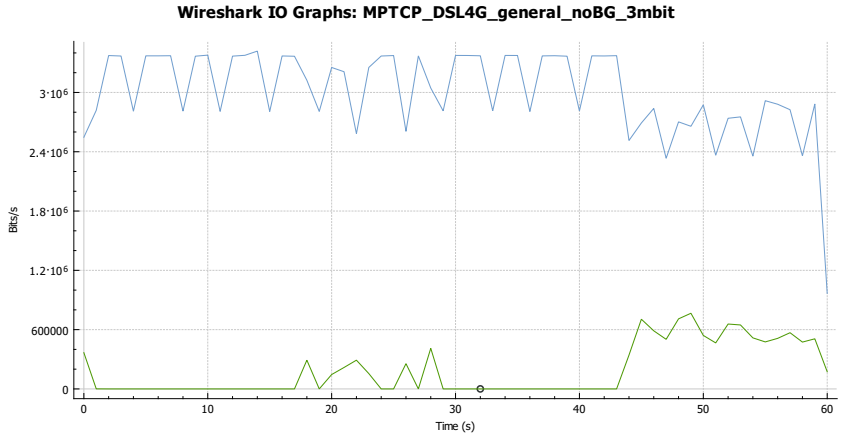


**Figure 5.60:** Graph of the throughput for general purpose with a 3 Mbps limit when running a MPTCP setup with 1 DSL connection and a 4G connection. Blue is the 4G, green is the DSL path.

#### RTT graph

If we look at Figure 5.61 we see the same behavior in the throughput, namely that around 44 seconds the 4G path changes characteristics and then there are traffic getting routed between the DSL interface.

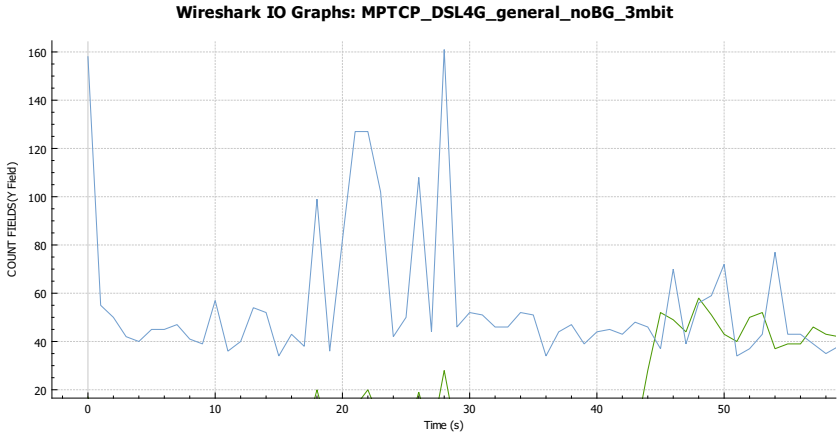


**Figure 5.61:** Graph of the RTT of the TCP stream for general purpose with a 3 Mbps limit as reported by wireshark. This is for flow both flows.

## Capinfo

```
Number of packets:   19 k
Data byte rate:      379 kBps
Data bit rate:       3033 kbps
```

**Listing 5.30:** Capinfo for MPTCP DSL+4G general purpose 3 Mbps limit with no background traffic. This is for flow 1 (4g).

```
Number of packets:   1827
Data byte rate:      23 kBps
Data bit rate:       187 kbps
```

**Listing 5.31:** Capinfo for MPTCP DSL+4G general purpose 3 Mbps limit with no background traffic. This is for flow 2 (dsl).

## Retransmissions

```
2
```

**Listing 5.32:** Retransmissions as reported by wireshark. This is for flow 1.

```
3
```

**Listing 5.33:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

Listing 5.34 shows a goodput of 3 Mbps, which is what was expected since the two subflows have enough capacity to handle the traffic generated.

```
1 [ ID] Interval           Transfer     Bandwidth       Retr
2 [  4]   0.00-60.00  sec  21.5 MBytes  3.01 Mbits/sec    0             sender
3 [  4]   0.00-60.00  sec  21.5 MBytes  3.01 Mbits/sec                  receiver
```

**Listing 5.34:** iperf3 bandwidth for MPTCP general purpose 3 Mbps limit no background traffic.

This test showed, that when using a Multipath TCP setup with heterogeneous paths MPTCP is still able to aggregate the bandwidth. Even though almost all the traffic was transfered via the 4G path, Figure 5.60 still shows that around 44 seconds some of the traffic is routed via the DSL path. This is because the RTT changes on the 4G connection, and the scheduler of the MPTCP then routes traffic through the path with the lowest RTT, which in this case changes to the DSL path.

### 5.6.2 Test C1.2 - MPTCP DSL + 4G general purpose 5 Mbps limit without background traffic

In this test a 5 Mbps limit will be used in iperf3. This is to determine what happens to the goodput when the generated traffic exceeds the capacity that the two homogeneous paths were able to handle. Since the capacity of the 4G link is unknown and it changes over time due to other users and signal strength etc. the 5 Mbps limit is used to be able to compare the results from the tests with the homogeneous paths. The expected outcome is that the traffic will still be routed through the path with the lowest RTT. Taking into consideration the results from the previous test it is expected that most of the traffic will get routed through the 4G path.

#### Throughput graphs

If we look at Figure 5.62 we see that the throughput for the blue line, which is the 4G path, fluctuates around 4.5 Mbps, which aligns with the expectation that most of the traffic would get routed through the 4G path. The green line, which is the DSL path, shows that the throughput for this path is around 1 Mpbs. Looking at Listing 5.35 it shows an average data bit rate of 4353 Kbps for the 4G path and an average data bit rate of 1067 Kbps for the DSL path.

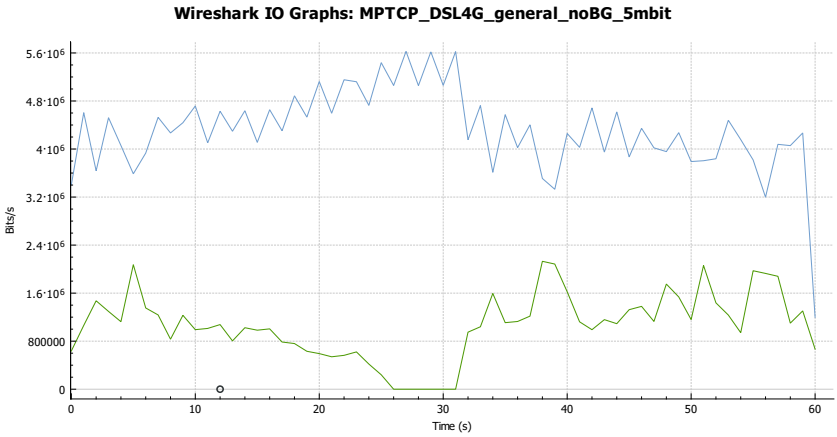


**Figure 5.62:** Graph of the throughput for general purpose with a 5 Mbps limit when running a MPTCP setup with 1 DSL connection and a 4G connection. Blue is the 4G path, green is the DSL path.

### RTT graph

As expected the RTT graphs show the same behavior observed in the previous test, that in some cases the DSL path has a smaller RTT than the 4G path and this causes some of the traffic to be routed via the DSL path. It is however not an even split between the two paths. If we compare Figure 5.63 and Figure 5.64 we see that even though the 4G RTTs fluctuate the DSL path still shows higher RTTs most of the time.

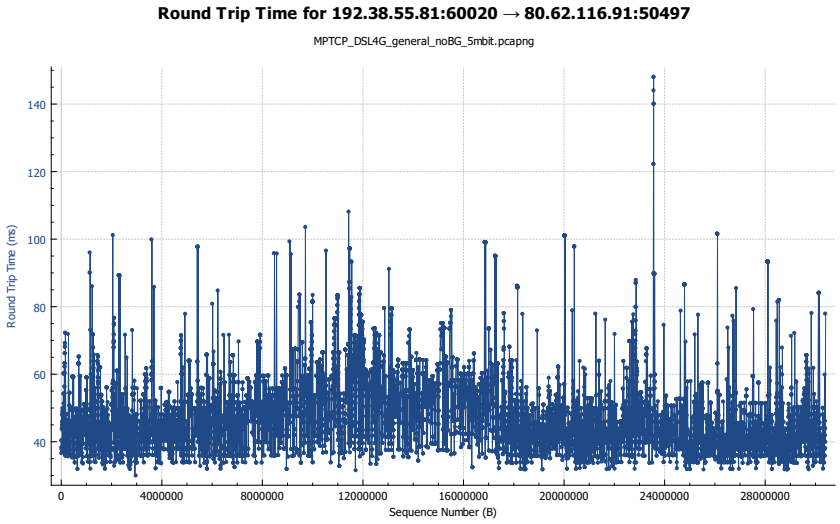


**Figure 5.63:** Graph of the RTT of the TCP stream for general purpose with a 5 Mbps limit as reported by wireshark. This is for flow 1 (4g).

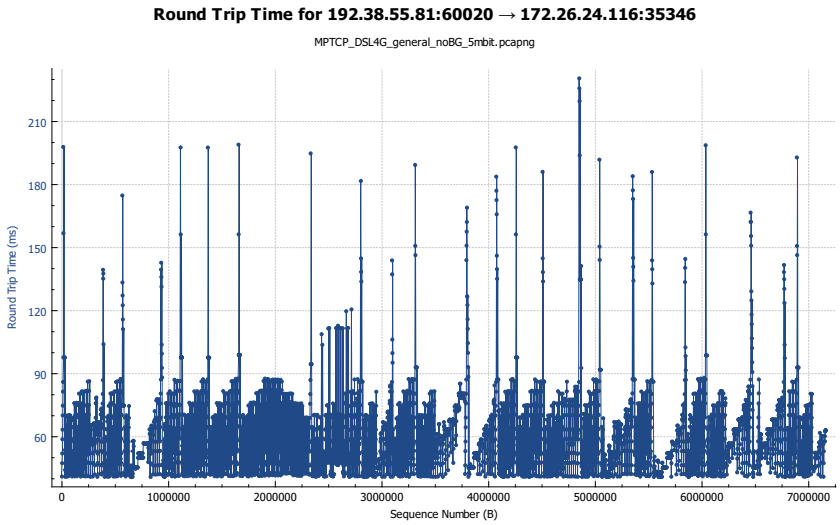


**Figure 5.64:** Graph of the RTT of the TCP stream for general purpose with a 5 Mbps limit as reported by wireshark. This is for flow 2 (dsl).

## Capinfo

```
Number of packets:   29 k
Data byte rate:      544 kBps
Data bit rate:       4353 kbps
```

**Listing 5.35:** Capinfo for MPTCP DSL+4G general purpose 5 Mbps limit with no background traffic. This is for flow 1 (4g).

```
Number of packets:   10 k
Data byte rate:      133 kBps
Data bit rate:       1067 kbps
```

**Listing 5.36:** Capinfo for MPTCP DSL+4G general purpose 5 Mbps limit with no background traffic. This is for flow 2 (dsl).

## Retransmissions

```
9
```

**Listing 5.37:** Retransmissions as reported by wireshark. This is for flow 1.

```
23
```

**Listing 5.38:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

The goodput reported by iperf3 is still 5 Mbps, so the general purpose traffic generated is still lower than the total available capacity of the paths combined.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  35.8 MBytes  5.01 Mbits/sec    0             sender
[  4]   0.00-60.00  sec  35.8 MBytes  5.01 Mbits/sec                  receiver
```

**Listing 5.39:** iperf3 bandwidth for MPTCP general purpose 5Mbps limit no background traffic.

This test showed that the MPTCP is able to aggregate the bandwidth even if the paths have different characteristics. In this case the majority of the traffic was transfered via the 4G path, which is plausible when looking at Test C1.1.

### 5.6.3 Test C2.1 - MPTCP DSL+4G interactive streaming 1 Mbps limit without background traffic

In this test the traffic for interactive streaming will be generated. It is done to determine that impact it has on the jitter when using Multipath TCP on two heterogeneous paths. The expected outcome is that the traffic will be routed via the path with the lowest RTT. Looking at the results of the previous tests with heterogeneous paths it is expected that most of the traffic will get routed via the 4G path. Since the difference in RTT on the two paths are not great, in this case where only 1 Mbps of the capacity will be utilized, it is expected that head-of-line blocking will not have a big impact.

#### Throughput graphs

If we look at Figure 5.65 we clearly see that virtually no traffic is routed via the DSL path. This aligns well with the fact that the 4G path in this test has the lowest RTT and that the path never gets congested, so the scheduler of the Multipath TCP only utilizes the 4G path. In this case it assures that head-of-line blocking will not be a problem.

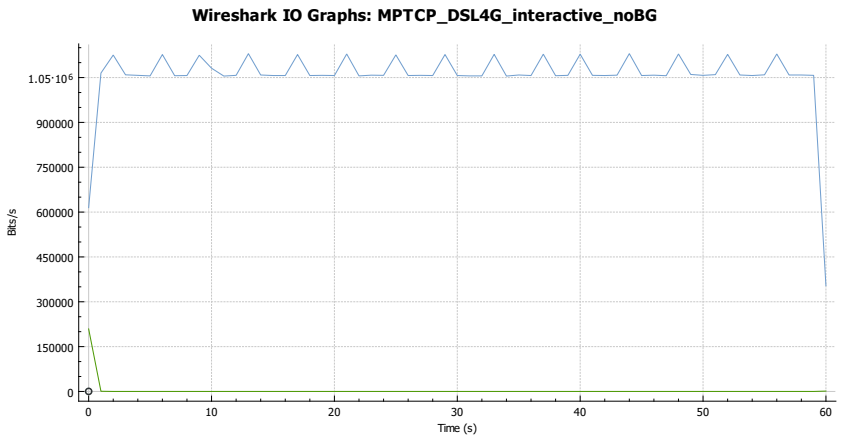


**Figure 5.65:** Graph of the throughput for interactive streaming with a 1 Mbps limit when running a MPTCP setup with 1 DSL connection and a 4G connection. Blue is the 4G path, green is the DSL path.

### RTT graph

Since almost all the traffic gets routed via the 4G interface Figure 5.67 is very limited in data points, but it is worth noticing that the data points that are plotted are higher than the ones in Figure 5.66, so the scheduler from Multipath TCP works as intended.

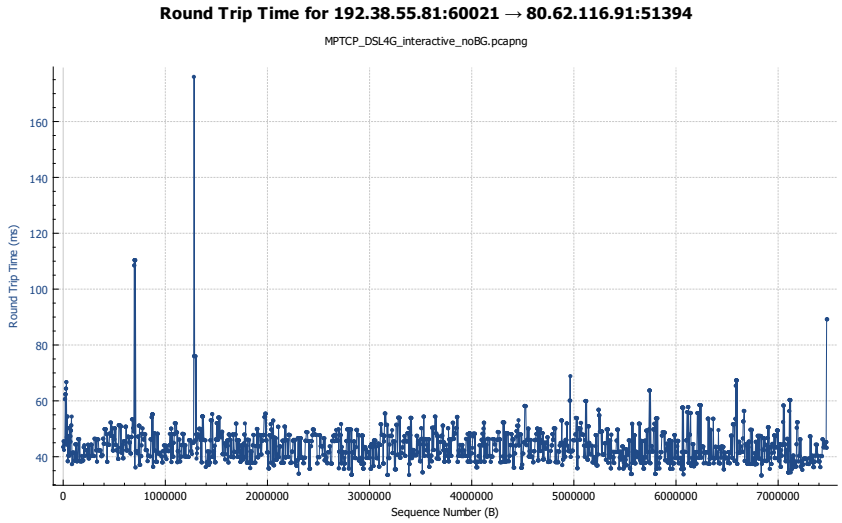


**Figure 5.66:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 1 (4g).

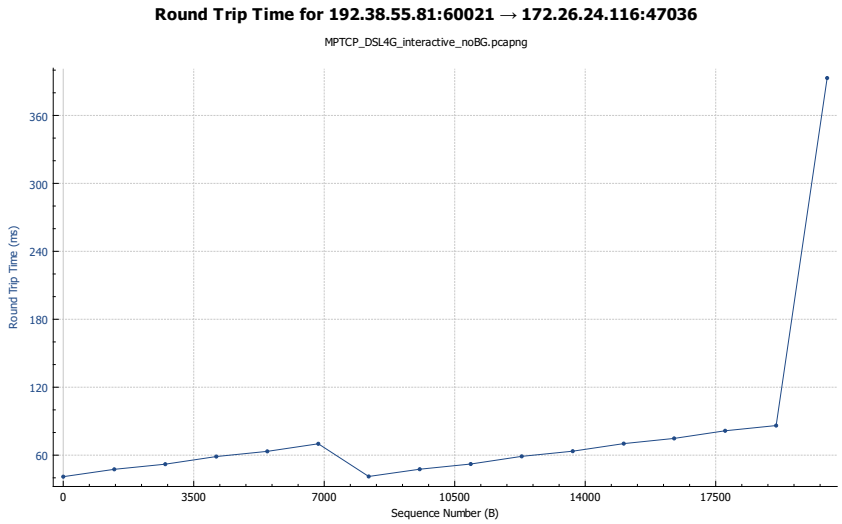


**Figure 5.67:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 2 (dsl).

### Capinfo

```
Number of packets:   6824
Data byte rate:      133 kBps
Data bit rate:       1070 kbps
```

**Listing 5.40:** Capinfo for MPTCP DSL+4G interactive streaming 1 Mbps limit without background traffic. This is for flow 1 (4g).

```
Number of packets:   39
Data byte rate:      438 bytes/s
Data bit rate:       3509 bits/s
```

**Listing 5.41:** Capinfo for MPTCP DSL+4G interactive streaming 1 Mbps limit without background traffic. This is for flow 2 (dsl).

### Retransmissions

```
1
```

**Listing 5.42:** Retransmissions as reported by wireshark. This is for flow 1.

```
0
```

**Listing 5.43:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

The goodput shows 1 Mbps at the server but slightly lower at the client. The magnitude of the difference should not matter.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec    0             sender
[  4]   0.00-60.00  sec  7.15 MBytes   999 Kbits/sec                  receiver
```

**Listing 5.44:** iperf3 bandwidth for MPTCP interactive streaming 1 Mbps limit without background traffic.

### Jitter

To calculate the jitter the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.30}$$

The mean and variance of the one way delay is:

$$mean(delay) = 21.8353\ ms \tag{5.31}$$
$$var(delay) = 10.538\ ms \tag{5.32}$$

**Figure 5.68:** Mean and variance 4G.

$$mean(delay) = 37.1155\ ms \tag{5.33}$$
$$var(delay) = 1579.651\ ms \tag{5.34}$$

**Figure 5.69:** Mean and variance DSL.

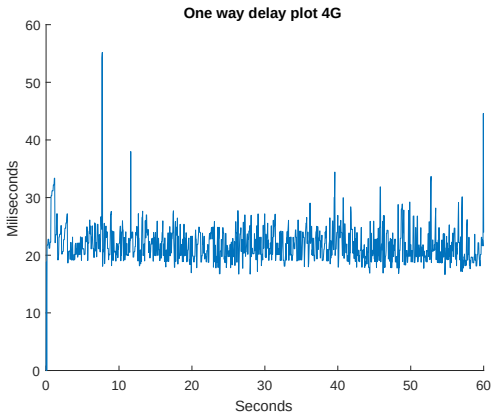


**Figure 5.70:** One way delay plot 4G.

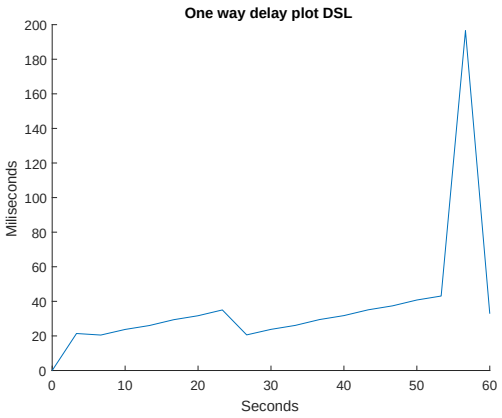


**Figure 5.71:** One way delay plot DSL.

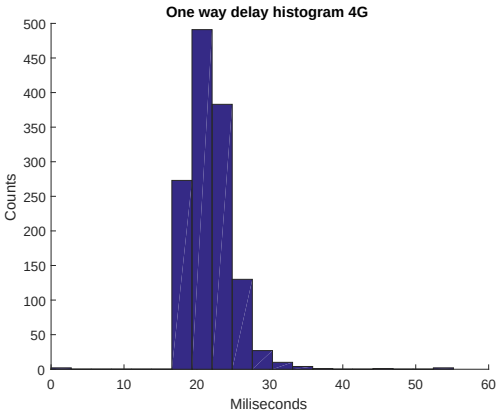


**Figure 5.72:** One way delay histogram 4G.

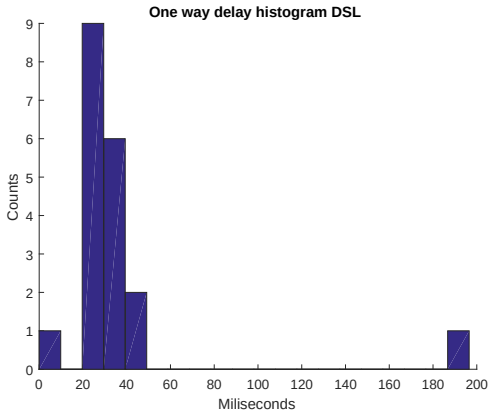


**Figure 5.73:** One way delay histogram DSL.

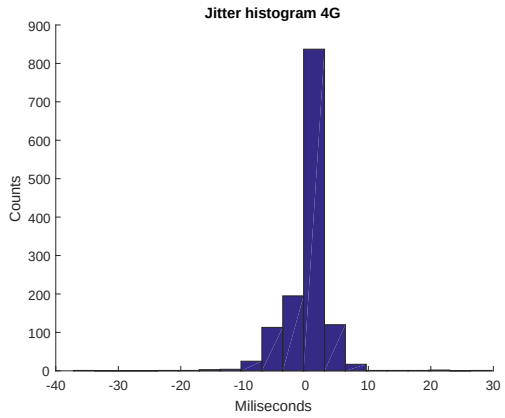


**Figure 5.74:** Jitter histogram 4G.

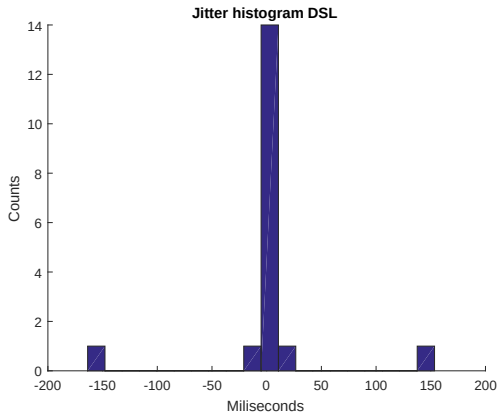


**Figure 5.75:** Jitter histogram DSL.

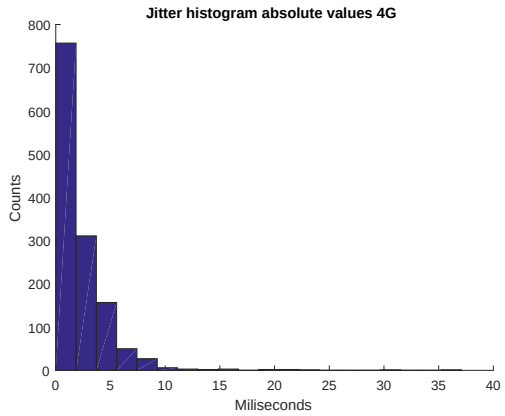


**Figure 5.76:** Abs jitter histogram 4G.

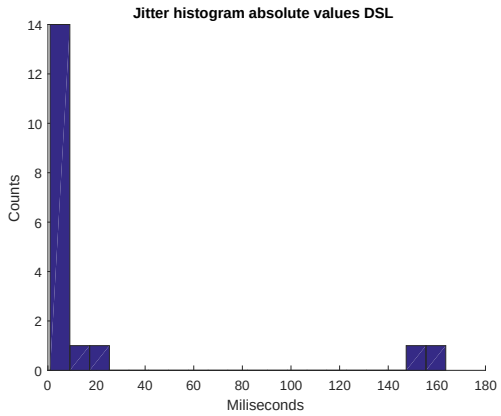


**Figure 5.77:** Abs jitter histogram DSL.

The mean and variance of the jitter is:

$$mean(jitter) = 1.975\ ms \quad (5.35)$$
$$var(jitter) = 7.969\ ms \quad (5.36)$$

**Figure 5.78:** Mean and variance 4G.

$$mean(jitter) = 21.711\ ms \quad (5.37)$$
$$var(jitter) = 2507.139\ ms \quad (5.38)$$

**Figure 5.79:** Mean and variance DSL.

If we look at Figure 5.70 and Figure 5.71 we clearly see that almost all of the traffic is routed via the 4G path. Looking at the capinfo it is seen that only 39 packets was

transfered via the DSL path. So we will focus on the results of the 4G path.
Looking at Figure 5.74 we see that there are both negative and positive jitter but most of the samples are located around 0-5 ms. This is supported by looking at the mean jitter for the 4G path, which is only 2 ms. The variance of the jitter is also very low, when comparing them to both the regular TCP tests but also the MPTCP homogeneous tests. In this case the 4G connection would actually be a very good candidate for running interactive streaming traffic.

### 5.6.4 Test C2.2 - MPTCP DSL+4G interactive streaming 1 Mbps limit with background traffic

This test will be done in the same way as the previous test but with added background traffic. This is to determine how background traffic impacts the jitter and the goodput.
The expected outcome is that the background traffic should have almost no impact on the jitter of the 4G path. It is also expected that the traffic will mainly get routed through the 4G path. The background traffic should have no impact on the goodput since the two subflows have more than enough capacity to handle the 1 Mbps limit ued in iperf3.

#### Throughput graphs

Figure 5.80 shows that the blue line, which is the 4G path, has a throughput of 1 Mbps, so the interactive streaming traffic is getting routed through the 4G path just as in the test without background traffic. The green line, which is the DSL path, shows that virtually no traffic was routed through the DSL path.

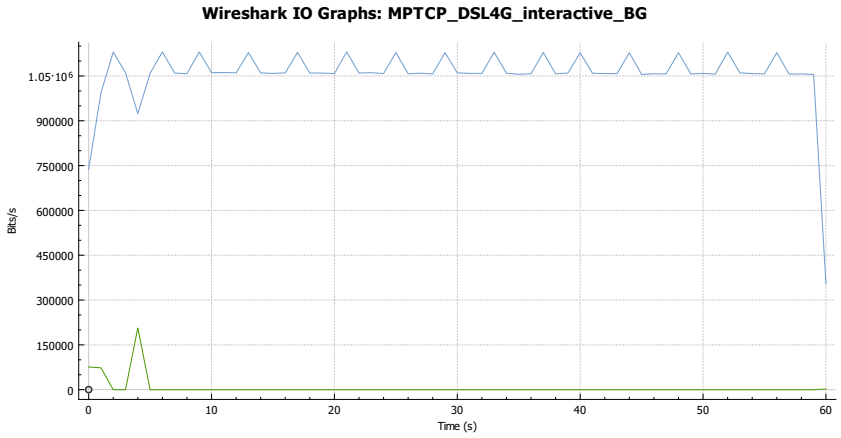


**Figure 5.80:** Graph of the throughput for interactive streaming with a 1 Mbps limit when running a MPTCP setup with 1 DSL connection and a 4G connection. Blue is the 4G path, green is the DSL path.

### RTT graph

Looking at Figure 5.82 shows that in the cases where an ACK RTT is calculated the value is higher than in Figure 5.81, which aligns with the fact that the traffic is routed via the 4G path.

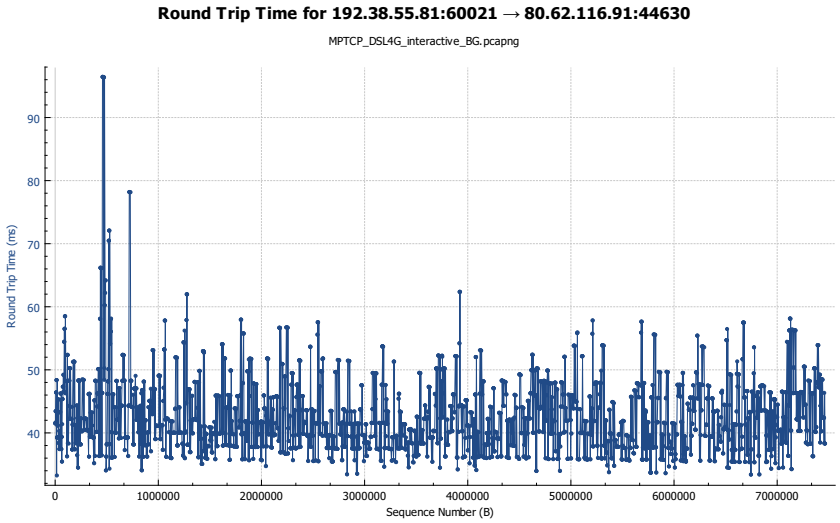


**Figure 5.81:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 1 (4g).

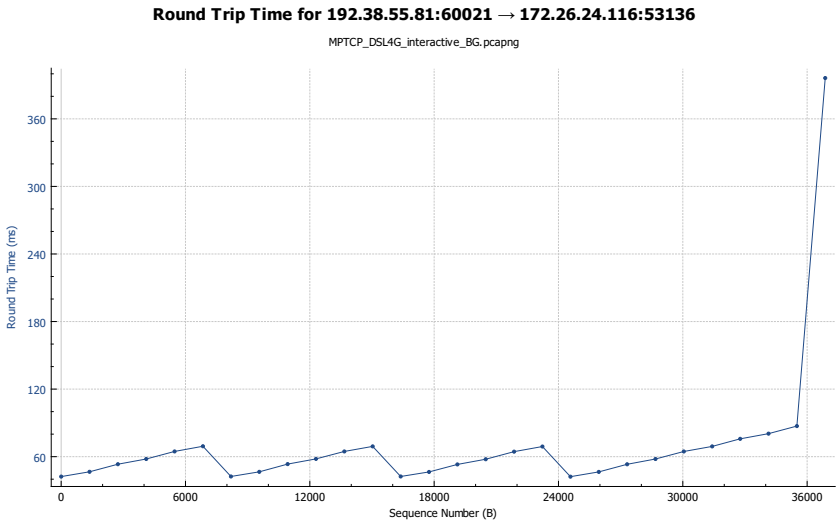


**Figure 5.82:** Graph of the RTT of the TCP stream for interactive streaming with a 1 Mbps limit as reported by wireshark. This is for flow 2 (dsl).

### Capinfo

```
Number of packets:   6967
Data byte rate:      133 kBps
Data bit rate:       1068 kbps
```

**Listing 5.45:** Capinfo for MPTCP DSL+4G interactive streaming 1 Mbps limit with background traffic. This is for flow 1 (4g).

```
Number of packets:   65
Data byte rate:      743 bytes/s
Data bit rate:       5949 bits/s
```

**Listing 5.46:** Capinfo for MPTCP DSL+4G interactive streaming 1 Mbps limit without background traffic. This is for flow 2 (dsl).

### Retransmissions

```
1
```

**Listing 5.47:** Retransmissions as reported by wireshark. This is for flow 1.

```
0
```

**Listing 5.48:** Retransmissions as reported by wireshark. This is for flow 2.

### Bandwidth from iperf3

The goodput form iperf3 shows 1 Mbps as expected. The background traffic has no impact on the goodput in this test.

```
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec    0             sender
[  4]   0.00-60.00  sec  7.16 MBytes  1.00 Mbits/sec                  receiver
```

**Listing 5.49:** iperf3 bandwidth for MPTCP interactive streaming 1 Mbps limit with background traffic.

### Jitter

To calculate the jitter the one way delay will be used by dividing all the values in the RTT distribution by 2. This will be an estimate because we assume that the delay is equal in both directions.

The one way delay will be:

$$delay = \frac{RTT}{2} \tag{5.39}$$

The mean and variance of the one way delay is:

$$mean(delay) = 21.3305\ ms \tag{5.40}$$
$$var(delay) = 11.262\ ms \tag{5.41}$$

**Figure 5.83:** Mean and variance 4G.

$$mean(delay) = 32.7455\ ms \tag{5.42}$$
$$var(delay) = 1000.752\ ms \tag{5.43}$$

**Figure 5.84:** Mean and variance DSL.

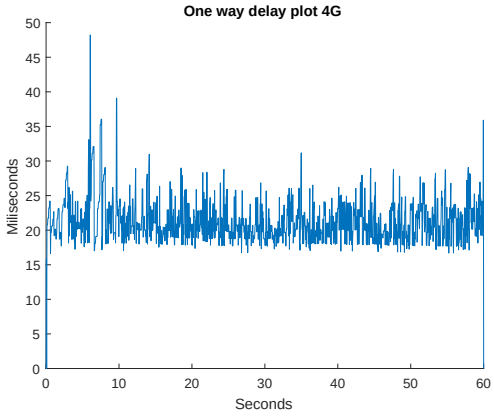


**Figure 5.85:** One way delay plot 4G.

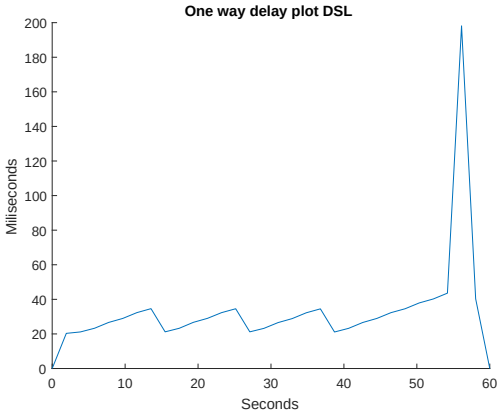


**Figure 5.86:** One way delay plot DSL.

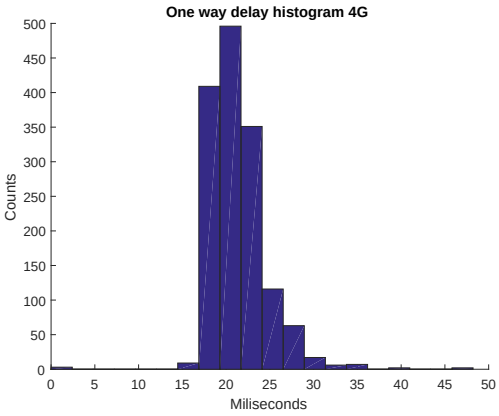


**Figure 5.87:** One way delay histogram 4G.

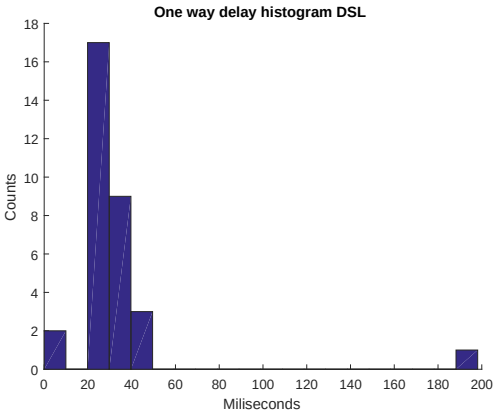


**Figure 5.88:** One way delay histogram DSL.

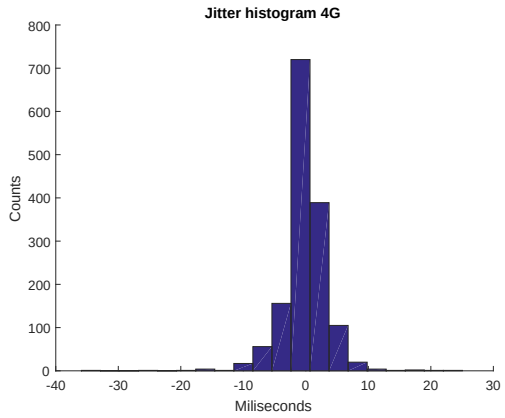


**Figure 5.89:** Jitter histogram 4G.

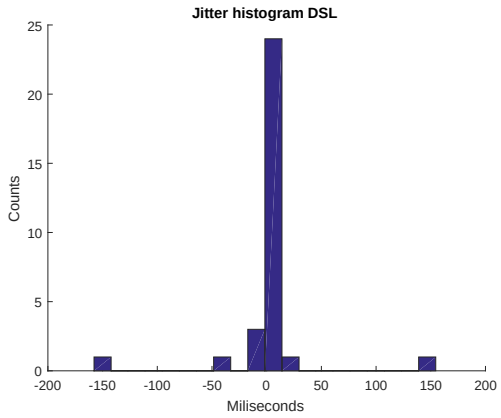


**Figure 5.90:** Jitter histogram DSL.

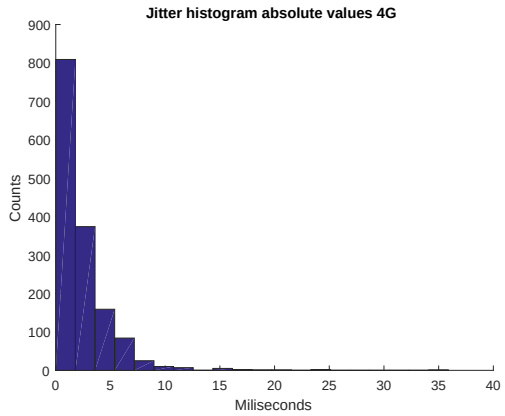


**Figure 5.91:** Abs jitter histogram 4G.

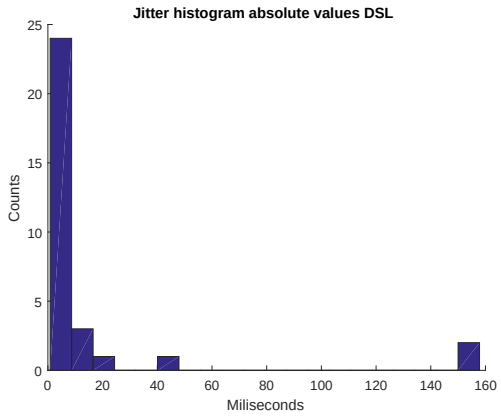


**Figure 5.92:** Abs jitter histogram DSL.

The mean and variance of the jitter is:

$$mean(jitter) = 2.0400\ ms \quad (5.44)$$
$$var(jitter) = 7.753\ ms \quad (5.45)$$

**Figure 5.93:** Mean and variance 4G.

$$mean(jitter) = 15.372\ ms \quad (5.46)$$
$$var(jitter) = 1473.974\ ms \quad (5.47)$$

**Figure 5.94:** Mean and variance DSL.

If we look at Figure 5.85 and Figure 5.86 it has the same behavior as in the previous test. Virtually all the traffic was transfered via the 4G path so we will only focus on

the results of the 4G path.
If we look at Figure 5.89 it looks almost identical to the one in the previous test. This is confirmed by looking at the mean and variance of the jitter, which is almost the same as the previous test. So in this case, where there was added background traffic, the mean and variance in the jitter is almost unchanged. And since the traffic is still routed via the 4G path it can be concluded that with this setup the network is not sensitive to background traffic and can be used for interactive streaming purposes. It can however not be concluded if using a Multipath TCP setup consisting of heterogeneous paths is able to be used for streaming purposes, since it only utilizes one subflow.

# Chapter 6

# Conclusion

When looking at the fact that, in the rural areas of Denmark, the quality of the Internet connections are very slow, there is a demand for a technology, that can help by combining already existing technologies and aggregate the bandwidth. This will be a cheap and smart solution, since it does not depend on installing expensive cables and equipment such as fiber optic solutions, and it can be done solely with some basic hardware and an existing technology like Multipath TCP.
This project tries to uncover if using the Multipath TCP technology will be a possible candidate to ensure that, in the rural areas of Denmark, greater bandwidths of the Internet connections can be reached, by aggregating already existing technologies such as DSL and 4G.

In chapter 1 the motivation for using a multipath approach is given. It explains how Multipath TCP was thought up as well as the history behind the protocol. It also explains how far the development of Multipath TCP has come.

In chapter 2 an extended analysis of how the protocol works is given. It compares a Multipath TCP operation to a regular TCP operation in order to understand the differences between these two protocols. It then proceeds to explain how the performance of Multipath TCP is dependent on the characteristics of the multiple paths used in a Multipath TCP setup, namely a setup with homogeneous paths and a setup with heterogeneous paths.
It also contains a section that explains what the state-of-the-art is, when talking about Multipath TCP and in which direction Multipath TCP is headed. Since Multipath TCP is an experimental standard with the IETF, it still has a long way to go before it will be used in practice.

In chapter 3 a short explanation of how the scenario is in the rural areas of Denmark is given. It describes how a proxy solution can be setup at a location and act as a gateway with an MPTCP enabled kernel in order to achieve bandwidth aggregation.

In chapter 4 an extensive description of how the setup of the testbed is done is given. It also contains a section, where the prerequisites that are needed in order to run the tests, are described. This included descriptions of the hardware, software and also which metrics that will be used to determine the performance achieved in the tests.

In chapter 5 an analysis of the results are given. This analysis will use data from the wireshark dump files recorded while running the tests. In each test a short analysis of the performance is given by looking at the achieved goodput, the ACK RTTs as calculated by wireshark and the calculated jitter from the ACK RTT distribution.

This project has showed that, by using a protocol as the Multipath TCP, bandwidth aggregation can be achieved. It works in cases where the setup consists of homogeneous paths as well as cases where the setup consists of heterogeneous paths. As shown in chapter 5 the Multipath TCP outperforms a single regular TCP connection when looking at the goodput. So it can be concluded that Multipath TCP is a viable candidate for deployment in rural areas where bandwidth aggregation is the goal.
In chapter 5 it was also shown that, when using Multipath TCP, even though the traffic is split on two routes, the network connection is still suited for running interactive streaming conversations. In only a few tests we saw a tendency with spikes over 100 ms, which would cause problems for interactive streaming traffic, due to the jitter buffer not coping well with variations over 100ms. However to be able to definitely conclude that MPTCP will not experience any problems if used in a real life setup, more test would be needed. This is because in the MPTCP heterogeneous tests only the 4G path was utilized and this basically makes the test the same as one single regular TCP connection. So here it is especially hard to compare the jitter results to the other tests.
The jitter calculations made in the MPTCP part of chapter 5 is only done as the jitter per path.The jitter calculated after the TCP stream is reassembled is the correct, but since the values needed for these calculations is not accessible in wireshark, the ACK RTT times was used instead.

## 6.1 Future work

The testbed setup in this project should be able to be deployed as a setup in a real life location. The setup would work as a proxy setup as explained in chapter 3 where the proxy would be MPTCP enabled and then it will not matter what kind of devices that are connected to the proxy e.g. router, switches etc. Then all it would take to establish a MPTCP connection with multipled subflows would be a MPTCP enabled device in the other end of the communication network e.g. a IT consultants device. Since Multipath TCP is still only an experimental standard they way the protocol works might change in the future. As of now the most critical step is to get an implementation of the Multipath TCP distribution where more schedulers are present. The schedulers mentioned in [5], that as of now are not available in the Linux implementation, Retransmission and Penalization (RP) and Bufferbloat Mitigation (BM) would be a nice addition to Multipath TCP, since they bring another approach than the current default scheduler, which works by using the paths with the lowest RTT first.
When looking at the different congestion control algorithms in the current version of Multipath TCP, the different algorithms seem to be sufficient to handle a lot of different scenarios, but it also seems that a congestion control algorithm, that can both be responsive to network changes as well as robust at the same time is possible. Maybe this will be implemented some time in the future.
When talking about congestion control algorithms and Multipath TCP, there are a lot of information about, whether or not a congestion control algorithm is fair or not to regular TCP. The reason for wanting to use a different congestion control algorithm that is different from regular TCP, is that if a TCP connection and a MPTCP connection with multiple subflows competed for the bandwidth the MPTCP would be able to get an unfair advantage since the congestion control algorithm would run on each of the subflows. If we look at an example with 1 regular TCP connection and a MPTCP connection with 2 subflows, all running the same congestion control algorithm as used in regular TCP, the bandwidth, if they share a bottleneck link, would be split evenly 3 ways. So MPTCP would get roughly 67% of the bandwidth and the regular TCP would get 33% of the bandwidth. However if this was to be deployed in a scenario where there are actually need for a higher bandwidth when using MPTCP, as it is the case in the farmer setup, where the MPTCP would be utilized for interactive streaming, using the regular TCP congestion in the MPTCP would actually ensure that the MPTCP traffic would get an unfair share of the bandwidth allocated.

# Bibliography


[1] J. Hwang S. Low A. Walid Q. Peng. *Balanced Linked Adaptation Congestion Control Algorithm for MPTCP*. https://tools.ietf.org/html/draft-walid-mptcp-congestion-control-00. 2015.

[2] Apple. *iOS: Multipath TCP Support in iOS 7*. https://support.apple.com/da-dk/HT201373. 2016.

[3] Anders Broman. *Offloading*. https://wiki.wireshark.org/CaptureSetup/Offloading. 2013.

[4] Pew Research Center. *Mobile Technology Fact Sheet*. http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/. 2016.

[5] Ozgu Alay Olivier Bonaventure Christoph Paasch Simone Ferlin. *Experimental Evaluation of Multipath TCP Schedulers*. http://inl.info.ucl.ac.be/system/files/paper_7.pdf. 2014.

[6] Internet Engineering Task Force. *TCP Extensions for Multipath Operation with Multiple Addresses*. https://tools.ietf.org/pdf/rfc6824.pdf. 2013.

[7] Wireshark Foundation. *Wireshark*. https://www.wireshark.org/docs/man-pages/capinfos.html. 2016.

[8] Mohammad M. N. Hamarsheh Hamzah M A Hijawi. *Performance analysis of multi-path TCP network*. http://aircconline.com/ijcnc/V8N2/8216cnc13.pdf. 2016.

[9] IETF. *TCP Extensions for Multipath Operation with Multiple Addresses*. https://tools.ietf.org/html/rfc6824. 2013.

[10] Alexey N. Kuznetsov. *tbf - Token Bucket Filter*. http://linux.die.net/man/8/tc-tbf. 2016.

[11] VOIP-Info.org LLC. *QoS*. http://www.voip-info.org/wiki/view/QoS. 2016.

[12] Log-Normal. *Analysing network characteristics using JavaScript and the DOM, Part I*. http://www.lognormal.com/blog/2011/11/14/analysing-network-characteristics/. 2012.

[13] Université catholique de Louvain. *MultiPath TCP - Linux Kernel implementation*. http://www.multipath-tcp.org/. 2016.

[14] Mahesh K. Marina Luca Boccassi Marwan M. Fayed. *Binder: a system to aggregate multiple internet gateways in community networks*. http://dl.acm.org/citation.cfm?id=2502894. 2013.

[15] Carlos J. Bernardos M. Isabel Sanchez Antonio de la Oliva. *How does my smartphone manage network connections?* http://www.it.uc3m.es/cjbc/papers/pdf/2016_sanchez_comnet_connectivity_management_smartphones.pdf. 2016.

[16] Enhuan Dong Mingwei Xu Yu Cao. *Delay-based Congestion Control for MPTCP*. https://tools.ietf.org/html/draft-xu-mptcp-congestion-control-01. 2015.

[17] Netflix. *Internet Connection Speed Recommendations*. https://help.netflix.com/en/node/306. 2016.

[18] Costin Raiciu Olivier Bonaventure Mark Handley. *An Overview of Multipath TCP*. http://inl.info.ucl.ac.be/system/files/bonaventure_0.pdf. 2012.

[19] Miroslav Popovic Utkarsh Upadhyay Jean-Yves Le Boudec Ramin Khaliliy Nicolas Gast. *MPTCP is not Pareto-Optimal: Performance Issues and a Possible Solution*. http://conferences.sigcomm.org/co-next/2012/eproceedings/conext/p1.pdf. 2012.

[20] Olivier Bonaventure Sébastien Barré Christoph Paasch. *MultiPath TCP: From Theory to Practice*. http://inl.info.ucl.ac.be/system/files/networking-mptcp.pdf. 2011.

[21] Michael J. Karels Van Jacobson. *Congestion Avoidance and Control*. http://ee.lbl.gov/papers/congavoid.pdf. 1988.

[22] Wikipedia. *G.711*. https://en.wikipedia.org/wiki/G.711. 2016.

[23] Wikipedia. *Iperf*. https://en.wikipedia.org/wiki/Iperf. 2016.

[24] Wikipedia. *Multipath TCP*. https://en.wikipedia.org/wiki/Multipath_TCP. 2016.

[25] Wikipedia. *Streaming media*. https://en.wikipedia.org/wiki/Streaming_media. 2016.

[26] Wikipedia. *TCP congestion control*. https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Tahoe_and_Reno. 2016.

[27] Wikipedia. *Transmission Control Protocol*. https://en.wikipedia.org/wiki/Transmission_Control_Protocol. 2016.