M. Tech. 2nd Semester Mini-Project Report

On

Experimental Evaluation of Different Schedulers in Out-of-Tree Implementation of MPTCP

Nishi Sahu

(222IS014)

Guide

Mohit P. Tahiliani

Department of Computer Science and Engineering,

NITK, Surathkal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

August, 2023

DECLARATION

I hereby declare that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Experimental Evaluation of Different Schedulers in Out-of-Tree Implementation of Multipath TCP** which is being submitted to the National Institute of Technology Karnataka, Surathkal, in partial fulfilment of the requirements for the award of the Degree of **Master of Technology in Computer Science and Information Security** in the Department of **Computer Science and Engineering**, is a bonafide report of the work carried out by me. The material contained in this Report has not been submitted to any University or Institution for the award of any degree.

Nishi Sahu
222IS014
Department of Computer Science and Engineering
NITK, Surathkal

Place: NITK, Surathkal.
Date: 20-04-23

CERTIFICATE

This is to certify that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Experimental Evaluation of Different Schedulers in Out-of-Tree Implementation of MPTCP** submitted by **Nishi Sahu**, (Roll Number: 222IS014) as the record of the work carried out by her, is accepted as the M. Tech. 2nd Semester Mini-Project Report submission in partial fulfilment of the requirements for the award of degree of **Master of Technology in Computer Science and Information Security** in the Department of **Computer Science and Engineering**.

Guide

Dr. Mohit P. Tahiliani

Department of Computer Science and Engineering

NITK, Surathkal

Chairman - DPGC

Dr. Manu Basavaraju

Department of Computer Science and Engineering

NITK, Surathkal

Abstract

Multi-path TCP (MPTCP) is designed to provide resource utilization through multi-path transfer of data packets to utilise network name spaces simultaneously. One of the key goal of MPTCP is to preserve fair resource sharing with regular TCP at network bottlenecks. Multi-path TCP was proposed for utilizing the feature of multi-homed devices where a device is able to connect with more than one device like both cellular data and WiFi simultaneously. Due to this the packets follow different paths during the data transfer and it might result in head-of-blocking and path heterogeneity problem. So to avoid these problems researchers proposed several path scheduling algorithms so that the packet reach in-order at the destination. The main aim of this report is to make comparison between different existing scheduling algorithms. The evaluation covers different performance metrics namely throughput, RTT, and buffer size to compute the performance of existing widely deployed scheduling algorithms.

Keywords: Multipath-TCP, Path Scheduler, Scheduling Algorithm, Network Namespaces.

Contents

List of Figures **iv**

1 Introduction **1**

2 Literature survey **2**

2.1 Overview of Linux Network Namespaces 2

2.2 Overview of MPTCP 2

2.3 Packet Schedulers in MPTCP 3

2.3.1 Lowest-RTT-FIRST 4

2.3.2 Delay-Aware Packet Scheduler 4

2.3.3 Round-Robin Scheduler 4

2.3.4 Blocking Estimation based MPTCP Scheduler 5

3 Problem Description **6**

3.1 Problem Statement 6

3.2 Objectives 6

4 Work Done **7**

4.1 Experimental Setup 7

4.1.1 Compiling the MPTCP Linux Kernel 7

4.1.2 Creating the Testbed using Linux Network Namespaces 9

4.2 Results 10

4.2.1 Low Rtt(default) Scheduler 10

4.2.2 Round Robin Scheduler 11

4.2.3 BLEST Scheduler 12

5 Conclusions and Future Work **15**

Bibliography **16**

List of Figures

2.1 Simple MPTCP Scheduler 3
4.1 MPTCP Kernel not loaded into the memory in normal mode 9
4.2 MPTCP Kernel not loaded into the memory in recovery mode 9
4.3 MPTCP linux kernel compilation 9
4.4 Topology 10
4.5 Output for Low RTT scheduler 11
4.6 Throughput analysis for Low RTT scheduler 11
4.7 Throughput for Round robin scheduler 12
4.8 Throughput for Round robin scheduler 12
4.9 Throughput for Blest scheduler 13
4.10 Throughput for Blest scheduler 13

1 Introduction

Multipath TCP (MPTCP) is an extension to the traditional Transmission Control Protocol (TCP) that allows a single TCP connection to use multiple network paths simultaneously. The key motivation behind the MPTCP is the trend toward providing trusted and reliable connectivity in the future Internet. The research in context focuses on faster downloads, lower data transfer costs and seamless switching between different interfaces, particularly the wireless ones such as wifi and cellular networks.

MPTCP is a modified version of the TCP protocol which enables applications that use TCP to utilize multiple available paths. This means that an MPTCP host can take advantage of various IP/port addresses to reach a destination host, and data streams can be distributed among these paths using subflows. Each subflow acts as an independent TCP flow, and they can be added or removed dynamically without ending the MPTCP connection.

When multiple paths exist between the end-hosts, it is the responsibility of the MPTCP scheduler to decide which path to use when transferring data. After establishing the connection, the MPTCP scheduler segments the data in the sender buffer and selects the best available subflow for each packet. A subflow is considered available when the three-way handshake is completed. The scheduler policy is used to select the best path from all currently available subflows.

MPTCP has two implementations in the Linux kernel: Out-of-tree and Upstream implementation. MPTCP (out-of-tree) became available in the Linux kernel in 2013. Since then, several multi-path schedulers have been added for different purposes. The scheduling algorithm is designed to distribute data packets on multiple paths based on each path's congestion window size controlled by the congestion control algorithm. Each scheduler can be selected based on the application and system requirements. The path scheduling algorithm is crucial for us to study MPTCP.

2 Literature survey

2.1 Overview of Linux Network Namespaces

Linux network namespaces are available in the *iproute2* package of the Linux kernel. It allows us to create isolated instances of the network stack on a single host. Each network namespaces have their own set of network devices, IP addresses, routing tables, and firewall rules, which are independent of the other network namespaces on the system. Network namespaces allows the user to create multiple virtual instances of the TCP/IP stack in the same machine. With the help of linux network namespaces we are able to create virtual test beds and perform network emulation for testing and comparing the performance of different network protocols

Since network namespaces virtualize the network stack, setting up a complex topology is quick, and collecting network statistics is simpler because network namespaces are isolated from each other. To enable communication between different network namespaces there exist virtual ethernet devices (veth). It is also possible to influence the network traffic between network namespaces using Linux traffic control (tc). These features make them ideal for network emulation. Hence, Linux network namespaces are widely used for creation and experimental evaluation of network protocols. These tests are necessary to gain a practical grasp of how different networking algorithms like congestion algorithms will behave in the real world. So, network namespaces provide a lightweight, cost-effective and scalable alternative to physical systems.

2.2 Overview of MPTCP

Multipath TCP (MPTCP) is an extension to the traditional Transmission Control Protocol (TCP) that allows a single TCP connection to use multiple network paths simultaneously, providing several benefits such as increased throughput, improved resilience to network failures, and better resource utilization. In traditional TCP, a connection is established between two endpoints, and all data is transmitted over a single path between those endpoints. However, with MPTCP, multiple paths can be used at the same time allowing for data to be split across multiple network paths. This can result in faster transfer rates and more efficient use of available bandwidth.

MPTCP achieves this by introducing a new subflow concept, which allows multiple subflows to exist within a single TCP connection. Each subflow corresponds to a separate path between the endpoints, and all subflows share a common congestion control mechanism to ensure fair sharing of network resources. MPTCP also includes mechanisms for path management, which allows for the addition and removal of network paths to a connection dynamically. This makes MPTCP particularly useful in environments with multiple network interfaces, such as mobile devices that can switch between Wi-Fi and cellular networks. Overall, MPTCP provides a flexible and efficient way to use multiple network paths simultaneously, improving the performance and resilience of TCP-based applications.

2.3 Packet Schedulers in MPTCP

In the case of MPTCP, the endpoints are connected via multiple paths, so the packets need to be scheduled on multiple subflows. The MPTCP scheduler is the algorithm which selects the path to be used for transmitting a packet. The path selection (routing) at the network layer is based on the network condition without considering the endpoint and application requirements. However, the MPTCP scheduler can use the information about paths (gathered by end-hosts) to efficiently distribute the packets on various paths/subflows. Fig. 2.1 shows the working of a simple MPTCP scheduler. The application sends the data to the output queue. The scheduling algorithm decides the path to be used for transmitting the data.

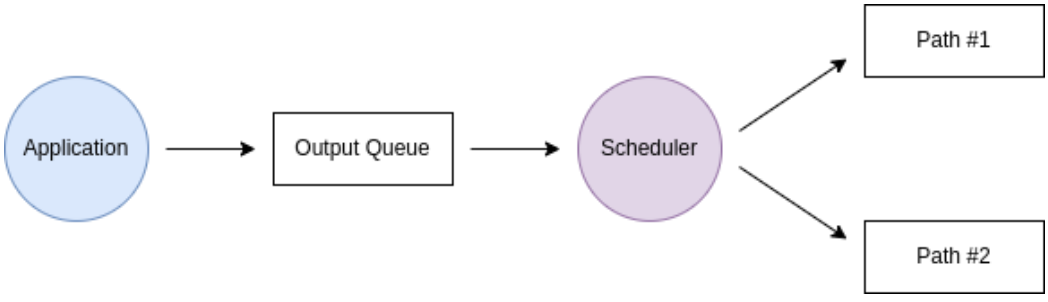

Figure 2.1: Simple MPTCP Scheduler

2.3.1 Lowest-RTT-FIRST

The LowRTT algorithm is the default scheduler, which first sends the packet on the subflow with the lowest RTT, until its congestion window is filled with packet. Then, the packet is sent on the subflow with the next higher RTT. The LowRTT algorithm makes the sub-flow with good path quality bear more data and has a certain load balancing effect. This is better than the RR algorithm, but neither algorithm considers the packet ordering. When the path difference is large, the path utilization will decrease.

The LRT scheduler in the MPTCP kernel assigns a priority value to each MPTCP subflow based on the estimated RTT of that subflow. The subflows with lower estimated RTTs are given higher priority and are scheduled first. This ensures that MPTCP packets are sent over the subflow with the lowest RTT, which reduces the overall RTT of the connection. Overall, the LRT scheduler in the MPTCP kernel can significantly improve the performance of MPTCP-based applications that require low-latency communication by reducing the overall RTT of the connection.

2.3.2 Delay-Aware Packet Scheduler

The DAPS algorithm aims to reduce the blocking time of the receive buffer. In the existing network, the link is asymmetric (i.e., each path has different delays, different capacity, different congestion windows), so the packets that can be scheduled for each path is different. The DAPS algorithm considers the difference of the path. It uses the ratio of the forward transmission delay (RTT /2) and the size of the congestion window (CWND) to determine the amount of data that each path should send next time, so that the packets arrive in order, and reducing Head-Of-Line Blocking.

2.3.3 Round-Robin Scheduler

The Round Robin (RR) algorithm is a simple scheduling mechanism. There is no priority between subflows in the RR algorithm, which selects subflows in a round-robin fashion.When using the RR scheduler, the MPTCP kernel sends packets over each available subflow in a round-robin fashion. This means that packets are sent over each subflow in turn, starting from the first subflow and moving on to the next

subflow in sequence. When all subflows have been used, the cycle starts again from the first subflow.

The RR scheduler in MPTCP can be useful in situations where all subflows have similar characteristics, such as similar RTT and available bandwidth. By evenly distributing traffic across all subflows, the RR scheduler can help to maximize the utilization of all available network resources. However, the RR scheduler may not be the best choice when subflows have different characteristics, as it may result in suboptimal performance.

2.3.4 Blocking Estimation based MPTCP Scheduler

The BLEST scheduler is a path manager in the Multipath TCP (MPTCP) kernel that is designed to maximize the available bandwidth across multiple subflows while minimizing packet reordering and latency. BLEST stands for "Bandwidth and Latency Estimation-based Subflow schedulers using Thresholds". It is a complex path manager that uses various algorithms to estimate the available bandwidth and latency of each subflow and dynamically adjust the scheduling of packets to optimize the overall performance of the MPTCP connection.

The BLEST scheduler in MPTCP consists of several sub-schedulers that work together to achieve the desired performance. These sub-schedulers include:

- **Bandwidth Scheduler**: This sub-scheduler estimates the available bandwidth of each subflow by measuring the sending rate and the amount of data in flight. It then allocates packets to each subflow based on its estimated available bandwidth.

- **Latency Scheduler**: This sub-scheduler estimates the latency of each subflow by measuring the round-trip time (RTT) and packet delay variation (PDV). It prioritizes packets on subflows with lower estimated latency to minimize reordering and improve performance.

- **Congestion Control**: This sub-scheduler implements congestion control mechanisms to prevent network congestion and ensure fair sharing of network resources among different connections.

3 Problem Description

3.1 Problem Statement

To evaluate and compare the performance of different path schedulers available in the out-of-tree implementation of Multipath TCP.

3.2 Objectives

- To gain an in-depth understanding of the working of Multipath TCP and its components.
- To gain an in-depth understanding of the different Scheduling algorithms of Multipath TCP.
- To emulate Multipath TCP using Linux network namespaces and utilities.
- To perform experiment and collect statistics by configuring different path schedulers for MPTCP.

4 Work Done

4.1 Experimental Setup

For creating the experiment, we can create a physical testbed or we can create a virtual testbed using Virtual machines. However, in our experiments we have used Linux network namespace for creating the virtual testbed since they are a cost-effective and scalable alternative to physical testbeds or VMs. We have used an Ubuntu 16.04 machine with Linux kernel version 4.19.2.34 having the out-of-tree implementation of Multipath TCP. For generating the dummy TCP traffic we have used the *iperf3* utility and the link attributes like bandwidth and delay have been configured using the Linux traffic control subsystem (*tc*).

4.1.1 Compiling the MPTCP Linux Kernel

The MPTCP out-of-tree implementation is present in the Linux kernel version 4.19.2.34. So for performing the experiment we first have to compile and install the MPTCP Linux kernel on the UBuntu 16.04 machine. Compiling the Linux kernel refers to the process of building the kernel source code into a binary executable that can be loaded into memory and executed by the computer's hardware. The kernel is the core component of the operating system that manages system resources and provides services to user applications.Compiling the kernel involves several steps, including configuring the kernel options, compiling the source code, and creating a bootable kernel image. The process can vary depending on the system architecture, hardware configuration, and kernel version. The following steps were followed in the Linux kernel compilation process.

- Download desired kernel from the MPTCP github repository [10].
- Update system and install necessary packages.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install git fakeroot build-essential
sudo apt-get install xz-utils libssl-dev bc flex
```

- Copy existing configuration from current kernel.

```
cp /boot/config-(uname -r) .config
```

- Edit configurations using GUI menu.

```
make menuconfig
```

- Ubuntu configuration file has full debugging which makes an enormous kernel and takes twice as long to compile. So we can disable debug_info to speed up the compilation process.

```
scripts/config --disable DEBUG_INFO
```

- We also need to override certificate checking using the following commands:

```
scripts/config --disable SYSTEM_TRUSTED_KEYS
scripts/config --disable SYSTEM_REVOCATION_KEYS
```

- To build Kernel and its modules run the following command:

```
time make -j (nproc)
```

- Next we install the necessary modules and then install the kernel.

```
time sudo make modules_install
sudo make install
```

Problems faced during Compilation process:

Initially we faced an error during the compilation process as shown in Figure 4.1. The error was related to loading the kernel into the memory. It appears that the memory block from where it is trying to load the OS is failing.

Problem Resolution:

The issue was that the latest ubuntu versions >= 20.04 are not compatible with the MPTCP kernel. So we setup anpther version of ubuntu 16.04 and try compiling the MPTCP linux kernel by downloading kernel-4.19.234.mptcp.tar.gz. following the above mentioned steps of kernel compilation and finally the mptcp kernel was successfully installed in the Ubuntu 16.04.

We used the following command to verify whether the Linux kernel is compiled successfully:

```
sysctl net.mptcp.mptcp_enabled
```

The output shown in Figure 4.3 indicates that the MPTCP Linux kernel has been successfully compiled and installed.

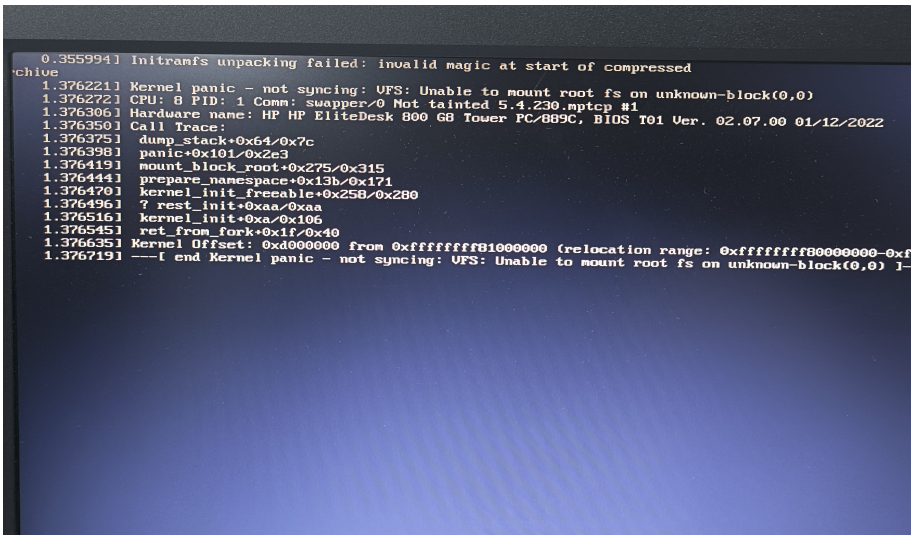

Figure 4.1: MPTCP Kernel not loaded into the memory in normal mode

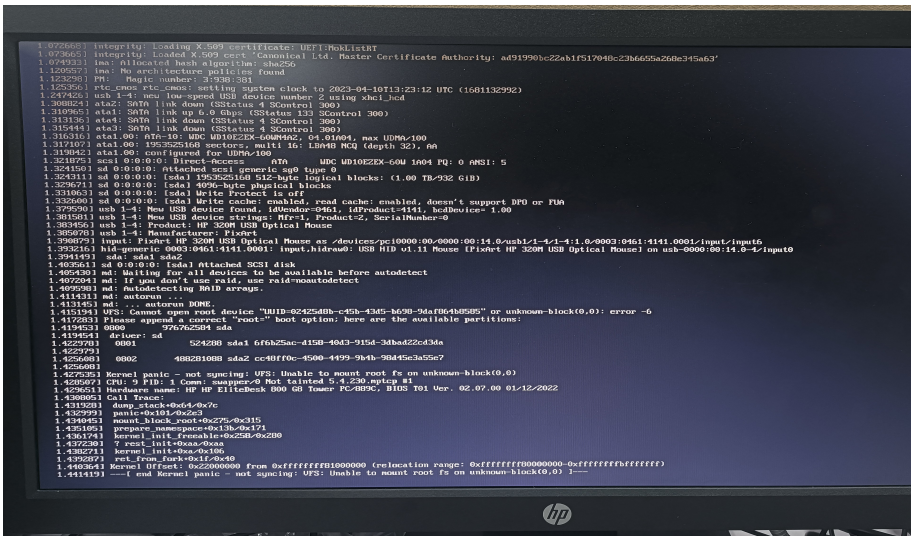

Figure 4.2: MPTCP Kernel not loaded into the memory in recovery mode

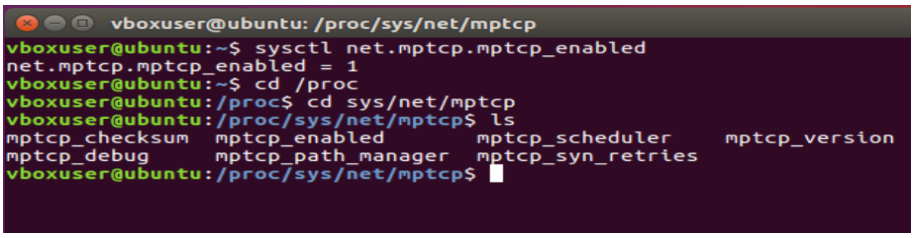

Figure 4.3: MPTCP linux kernel compilation

4.1.2 Creating the Testbed using Linux Network Namespaces

We have used the topology shown in Figure 4.4 for performing the experiments.

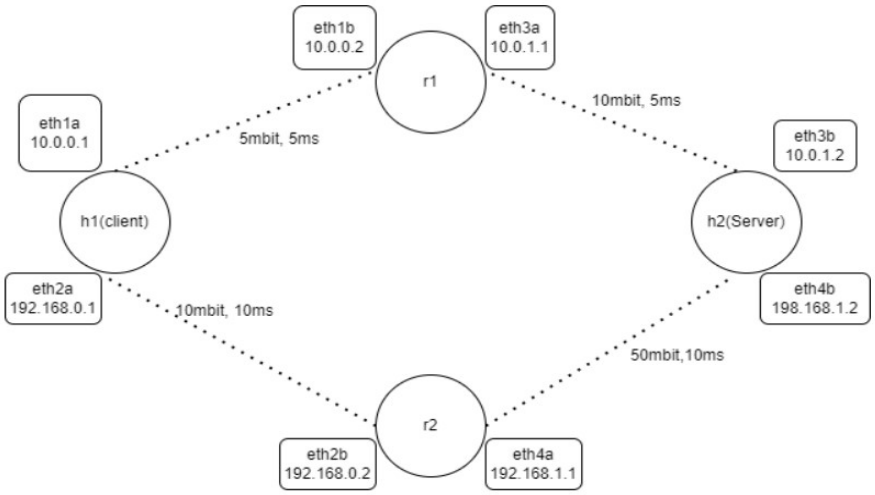

Figure 4.4: Topology

In this topology, we created two hosts h1 and h2 and two routers connected with virtual ethernet pairs. The host h2 is the server and h1 acts as the client. The host h1 is connected with router r1 and r2, and the host h2 is also connected with router r1 and r2 using the virtual ethernet pairs and the script to create this topology is present in the Appendix A.

4.2 Results

We evaluate the performance of the different scheduler algorithms based on the virtual MPTCP Linux experimental platform with the above mentioned topology.

4.2.1 Low Rtt(default) Scheduler

The LowRTT algorithm is the default scheduler, which first sends the packet on the subflow with the lowest RTT, until its congestion window is filled with packet. We configure the LowRTT scheduler and path manager as full mesh using the following commands:

```
sysctl net.mptcp.mptcp_scheduler=default
sysctl net.mptcp.mptcp_path_manager=fullmesh
```

On running the iperf for 10 seconds we obtained the output shown in Figure 4.5. The graph showing the variation in throughput is shown in Figure 4.6.

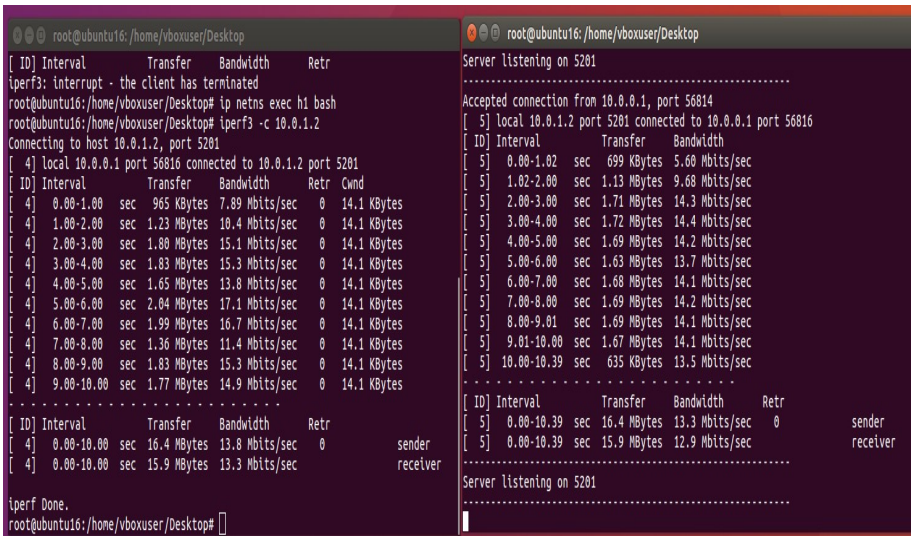

Figure 4.5: Output for Low RTT scheduler

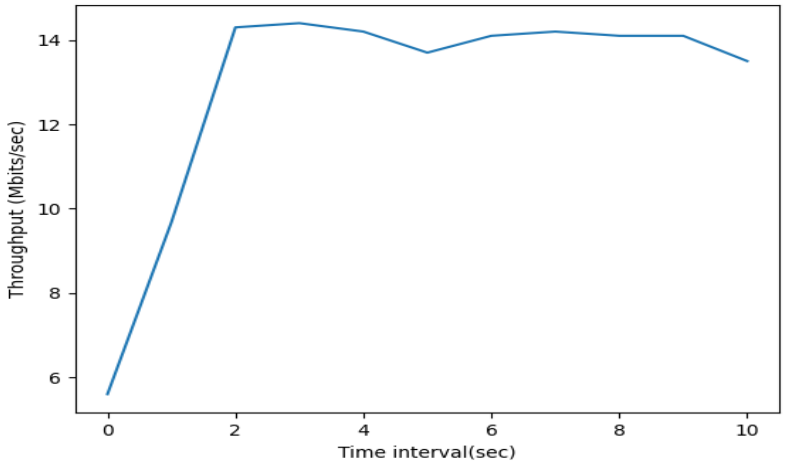

Figure 4.6: Throughput analysis for Low RTT scheduler

4.2.2 Round Robin Scheduler

A Round Robin scheduler is a type of scheduling algorithm that cycles through a list of elements, allocating each element an equal share of resources in a circular fashion. We configure the Round Robin scheduler and path manager as full mesh using the following commands:

```
sysctl net.mptcp.mptcp_scheduler=roundrobin
sysctl net.mptcp.mptcp_path_manager=fullmesh
```

On running the iperf for 10 seconds we obtained the output shown in Figure 4.7.

The graph showing the variation in throughput is shown in Figure 4.8.

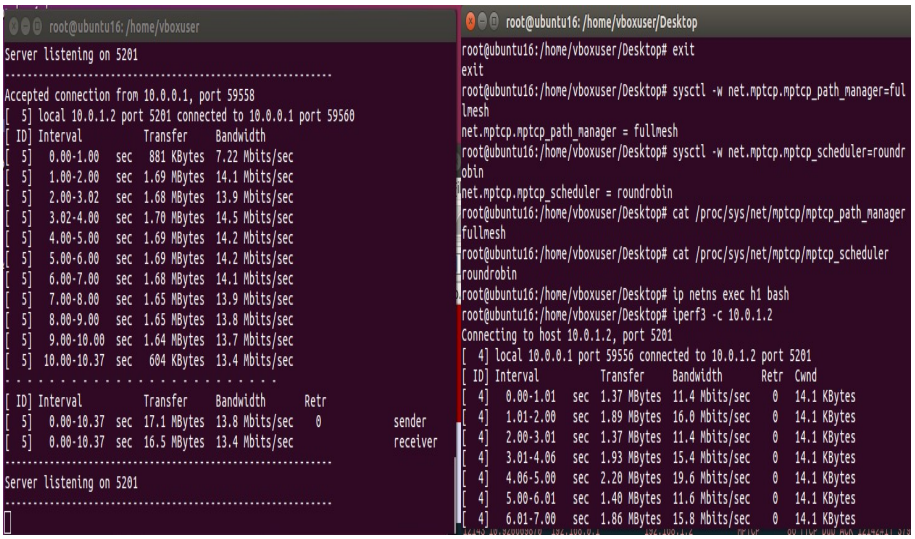

Figure 4.7: Throughput for Round robin scheduler

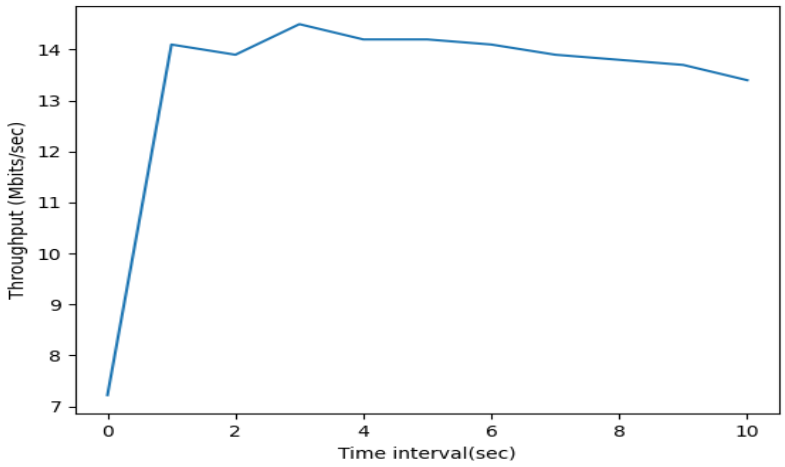

Figure 4.8: Throughput for Round robin scheduler

4.2.3 BLEST Scheduler

The Blest Scheduler is a specific implementation of MPTCP that is designed to improve the performance of MPTCP by optimizing the way it chooses which path to use for each packet. We configure the Blest scheduler and path manager as full mesh using the following commands:

```
sysctl net.mptcp.mptcp_scheduler=blest
sysctl net.mptcp.mptcp_path_manager=fullmesh
```

On running the iperf for 10 seconds we obtained the output shown in Figure 4.9. The graph showing the variation in throughput is shown in Figure 4.10.

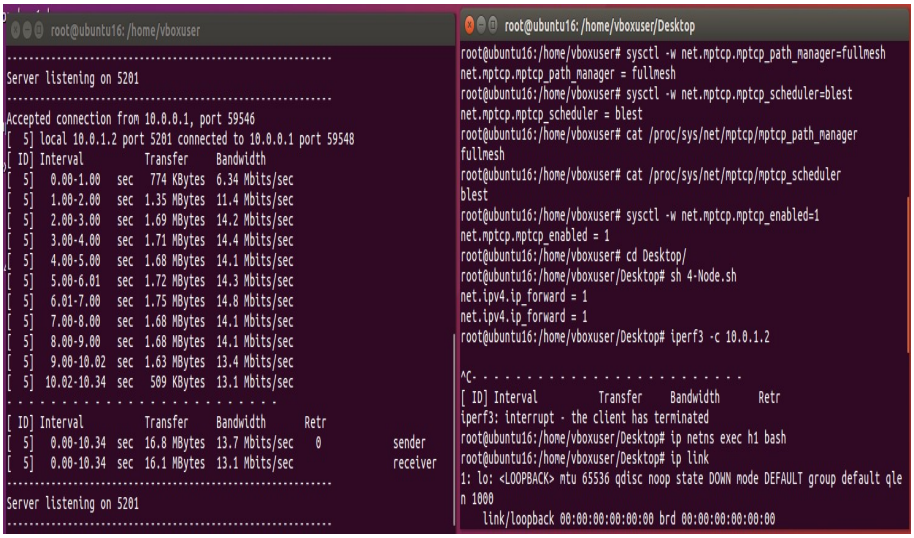

Figure 4.9: Throughput for Blest scheduler

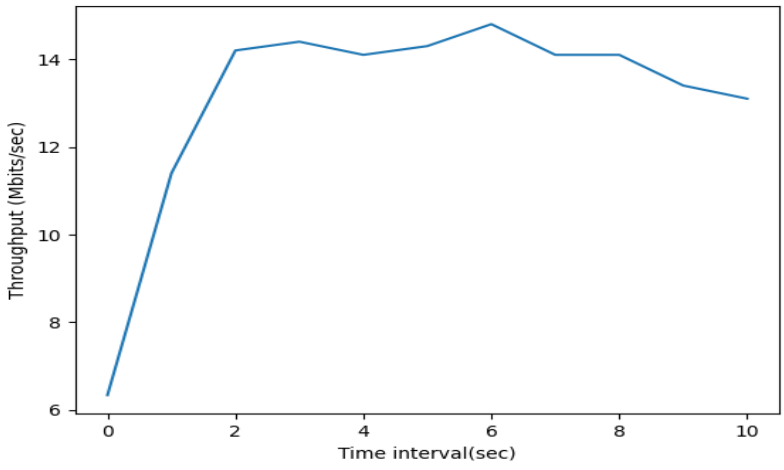

Figure 4.10: Throughput for Blest scheduler

Average Throughput:

Table 1 shows the average throughput for different types of MPTCP scheduler. The Low-RTT scheduler gave the best results, followed by Round Robin scheduler.

Scheduler name	Average Throughput(Mbits/sec)
Round Robin	13.8
Blest	13.1
Low Rtt	14.8

Table 1: Average throughput

5 Conclusions and Future Work

In this report, we have given an overview of the experimental analysis of different schedulers. We have discussed the different types of MPTCP scheduler . Additionally, we performed the comparitive analysis of several MPTCP schedulers in the Linux Kernel using network namespaces for setting up the testbed. IN this work, we have performed the evaluation based on the throughput variation. As part of the future work we plan to perform the experimental analysis considering a more performance metrics like RTT, buffersize and file size.

Bibliography

[1] G. Liu et al., "3-D-MIMO with massive antennas paves the Way to 5G enhanced mobile broadband: From system design to field trials," IEEE J. Sel. Areas Commun., vol. 35, no. 6, pp. 1222–1233, Jun. 2017.

[2] M. Hashemi, C. E. Koksal, and N. B. Shroff, "Out-of-band millimeter wave beamforming and communications to achieve low latency and high energy efficiency in 5G systems," IEEE Trans. Commun., vol. 66, no. 2, pp. 875–888, Feb. 2018.

[3] U. Chunduri et al., Considerations for MPTCP Operation in 5G, docu- ment draft-defoy-mptcp-considerations-for-5g-00, 2007.

[4] C. Lee, S. Song, H. Cho, G. Lim, and J.-M. Chung, "Optimal multipath TCP offloading over 5G NR and LTE networks," IEEE Wireless Commun. Lett., to be published.

[5] J. Zeng, Y. Cao, F. Ke, M. Huang, G. Zhang, and W. Lu, "Performance evaluation of secure multipath retransmission mechanism in next gener- ation heterogeneous communication systems," IET Netw., vol. 7, no. 2, pp. 61–67, 2018.

[6] K. Xue et al., "DPSAF: Forward prediction based dynamic packet schedul- ing and adjusting with feedback for multipath TCP in lossy heterogeneous networks," IEEE Trans. Veh. Technol., vol. 67, no. 2, pp. 1521–1534, Feb. 2018.

[7] H. Huang, "Weight-based data scheduling algorithm designing for MPTCP," Comput. Eng. Softw., vol. 37, no. 2, pp. 77–80, Feb. 2016.

[8] C. Raiciu, M. Handley, and D. Wischik, Coupled Congestion Control for Multipath Transport Protocols, document RFC 6356, 2011.

[9] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, imple- mentation and evaluation of congestion control for multipath TCP," in Proc. Usenix Conf. Netw. Syst. Design Implement., 2011, pp. 99–112.

[10] MPTCP out-of-tree Github repository: `https://github.com/multipath-tcp/mptcp/releases/download/v0.95.2/kernel-4.19.234.mptcp.tar.gz`

Appendix

A. Script for Creating the Topology

```
ip netns add h1
ip netns add h2
ip netns add r1
ip netns add r2
Disable the reverse path filtering
sysctl -w net.ipv4.conf.all.rp_filter=0
Enable MPTCP on both the network namespaces
Create two virtual ethernet (veth) pairs
ip link add eth1a netns h1 type veth peer name eth1b netns r1
ip link add eth2a netns h1 type veth peer name eth2b netns r2
ip link add eth3a netns r1 type veth peer name eth3b netns h2
ip link add eth4a netns r2 type veth peer name eth4b netns h2
Assign IP address to each interface on h1
ip netns exec h1 ip address add 10.0.0.1/24 dev eth1a
ip netns exec h1 ip address add 192.168.0.1/24 dev eth2a
Assign IP address to each interface on r1
ip netns exec r1 ip address add 10.0.0.2/24 dev eth1b
ip netns exec r1 ip address add 10.0.1.1/24 dev eth3a
Assign IP address to each interface on r2
ip netns exec r2 ip address add 192.168.0.2/24 dev eth2b
ip netns exec r2 ip address add 192.168.1.1/24 dev eth4a
Assign IP address to each interface on h2
ip netns exec h2 ip address add 10.0.1.2/24 dev eth3b
ip netns exec h2 ip address add 192.168.1.2/24 dev eth4b
Set the data rate and delay on the veth devices at h1
ip netns exec h1 tc qdisc add dev eth1a root netem delay 5ms rate 5mbit
ip netns exec h1 tc qdisc add dev eth2a root netem delay 10ms rate 10mbit
Set the data rate and delay on the veth devices at r1
ip netns exec r1 tc qdisc add dev eth3a root netem delay 5ms rate 10mbit
```

```
ip netns exec r1 tc qdisc add dev eth1b root netem delay 10ms rate 5mbit
Set the data rate and delay on the veth devices at r2
ip netns exec r2 tc qdisc add dev eth2b root netem delay 10ms rate 10mbit
ip netns exec r2 tc qdisc add dev eth4a root netem delay 10ms rate 50mbit
Set the data rate and delay on the veth devices at h2
ip netns exec h2 tc qdisc add dev eth3b root netem delay 5ms rate 10mbit
ip netns exec h2 tc qdisc add dev eth4b root netem delay 10ms rate 50mbit
Turn ON all ethernet devices
ip netns exec h1 ip link set eth1a up
ip netns exec h1 ip link set eth2a up
ip netns exec r1 ip link set eth1b up
ip netns exec r1 ip link set eth3a up
ip netns exec r2 ip link set eth2b up
ip netns exec r2 ip link set eth4a up
ip netns exec h2 ip link set eth3b up
ip netns exec h2 ip link set eth4b up
Enable IP forwarding
ip netns exec r1 sysctl -w net.ipv4.ip_forward=1
ip netns exec r2 sysctl -w net.ipv4.ip_forward=1
Create two routing tables for two interace in h1
ip netns exec h1 ip rule add from 10.0.0.1 table 1
ip netns exec h1 ip rule add from 192.168.0.1 table 2
Configure the two routing tables of h1
ip netns exec h1 ip route add 10.0.0.0/24 dev eth1a scope link table 1
ip netns exec h1 ip route add default via 10.0.0.2 dev eth1a table 1
ip netns exec h1 ip route add 192.168.0.0/24 dev eth2a scope link table 2
ip netns exec h1 ip route add default via 192.168.0.2 dev eth2a table 2
Create two routing tables for two interface in h2
ip netns exec h2 ip rule add from 10.0.1.2 table 3
ip netns exec h2 ip rule add from 192.168.1.2 table 4
Configure the two routing tables
ip netns exec h2 ip route add 10.0.1.0/24 dev eth3b scope link table 3
```

```
ip netns exec h2 ip route add default via  10.0.1.1 dev eth3b table 3
ip netns exec h2 ip route add 192.168.1.0/24 dev eth4b scope link table 4
ip netns exec h2 ip route add default via 192.168.1.1 dev eth4b table 4
Global Default route for h1 and h2
ip netns exec h1 ip route add default global nexthop via 10.0.0.2 dev eth1a
ip netns exec h2 ip route add default global nexthop via 192.168.1.1 dev eth4b
```