M. Tech. 2nd Semester Mini-Project Report

On

Experimental Evaluation of Congestion Control Algorithms of Multipath TCP

Poonam Dubey

(222IS019)

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

June, 2023

DECLARATION

I hereby declare that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Experimental Evaluation of Congestion Control Algorithms 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.

Poonam Dubey
222IS019
Department of Computer Science and Engineering
NITK, Surathkal

Place: NITK, Surathkal.
Date: 20-04-2023

CERTIFICATE

This is to certify that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Experimental Evaluation of Congestion Control Algorithms of Multipath TCP** submitted by **Poonam Dubey**, (Roll Number: 222IS019) 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

In Multipath TCP, the congestion control is realized by individual sub flows (like regular TCP). However, there is some problem that the total data sending rate becomes too high compared with other single path TCPs if the associated subflows increase their own congestion windows independently. To solve this problem, coupled congestion control algorithm was introduced. Linked Increase Adaptation (LIA), Opportunistic Linked-Increase Algorithm (OLIA), Balanced Linked Adaptation (BALIA) and wVegas are some coupled congestion control algorithms. We are going to compare the performance of coupled congestion control algorithms including LIA, OLIA, BALIA or wVegas and the uncoupled CUBIC algorithm with respect to the average throughput and variation in congestion window.

Keywords: Multipath-TCP, Congestion Control Algorithms, Network namespaces.

Contents

List of Figures

1 Introduction

The transmission control protocol (TCP) is the most widely used protocol over the internet. As TCP is a reliable protocol, most applications use TCP as a transfer protocol to transmit data over the internet. Every TCP connection is a single flow and limited to a unique socket (IP address and Port number) between the source and destination. Once the connection is established if for some reason the IP address of the source or destination is changed then the whole connection will fail. MPTCP is an extension to TCP, which allows multiple paths to be used simultaneously by a single TCP transport connection. MPTCP has reliability in case of link failures and for load balancing in case of multi-homed servers. There are two implementations of MPTCP, namely out-of-tree and upstream.

In terms of congestion control, using multiple subflows for one connection can be complex. With TCP, congestion occurs on a single path between two hosts. MPTCP uses different paths for transferring data. In general, the two paths will experience a different level of congestion and delay. A simple solution for MPTCP Congestion Control is to use a separate regular TCP congestion control algorithm on each subflow called 'Uncoupled Congestion Control'. However, the uncoupled CC algorithms might be unfair to the existing TCP flows in case multiple MPTCP flows share a common bottleneck with some TCP flows. Alternatively, a new class of congestion control algorithms have been implemented in which the total congestion window is shared between all sub flows. Such algorithms are called 'Coupled Congestion Control Algorithms'. In this work we aim to perform a comparitive analysis of the existing congestion control algorithms of Multipath TCP. We have used Linux network namespaces to create the testbed for experimental evaluation. Linux network namespaces are a light weight and scalable alternative to physical testbeds or virtual machines. The performance of the congestion control algorithms would be compared based on the average throughput and variation in congestion window size.

2 Literature survey

2.1 Linux Network Namespaces

In the Linux operating system, the concept of namespaces appears in a variety of forms. The general goal is to provide areas where processes can run in isolation from others in terms of system resources. Network namespaces are one of these types. It is a kernel feature that allows us to virtualize network environments and isolate them. Each network namespace has its network stack and networking-related system resources such network devices, IPv4 and IPv6 protocol stacks, IP routing tables, firewall rules, port numbers (sockets) and so on.

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 MPTCP Congestion Control

MPTCP is present on the transport layer (Fig. 2.1). An MPTCP connection is linked with one or more TCP connections called sub flows. During the initial connection establishment, a TCP option named MP_CAPABLE is used within *SYN*, *SYN/ACK*, and *ACK* segments, which leads to the creation of initial or master flow. When the master connection has established then further MP_JOIN option is used for creating additional sub-flows. In MPTCP, the data stream taken as input is divided in one or more subflows, with sufficient control information for reassembly so that data stream can reach to destination reliably and in order.

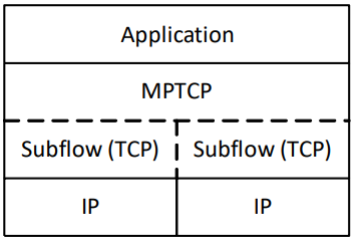

Figure 2.1: Layer Structure of MPTCP [1]

In MPTCP, the connection manages a data sequence number which is independent from the subflow sequence number. The *ACK* segment and data contain Data Sequence Signal (DSS) option shown in Fig. 2.2.

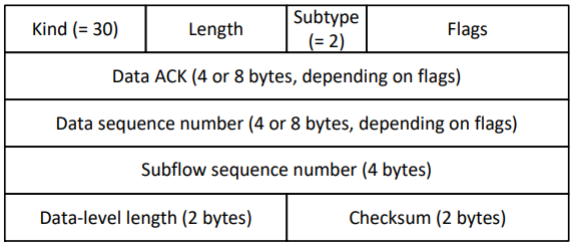

Figure 2.2: DSS Option [1]

Congestion Control is another important component of MPTCP which highly determines the performance of MPTCP. It only has to detect congestion and allocate resources, but also has to provide fairness among several subflows [3]. Few of the important goals which are expected to be achieved by any MPTCP Congestion Control Algorithm are listed below.

- It should provide at least the same or better throughput compared to a single TCP connection.
- One subflow should not harm the other subflows. It basically means one subflow should not take up more resources than it is expected to take such that it hampers the performance of other flows.
- A multipath congestion control algorithm should send as much traffic to a subflow, as much as its capacity at any particular point of time. This feature is also termed as resource pooling.

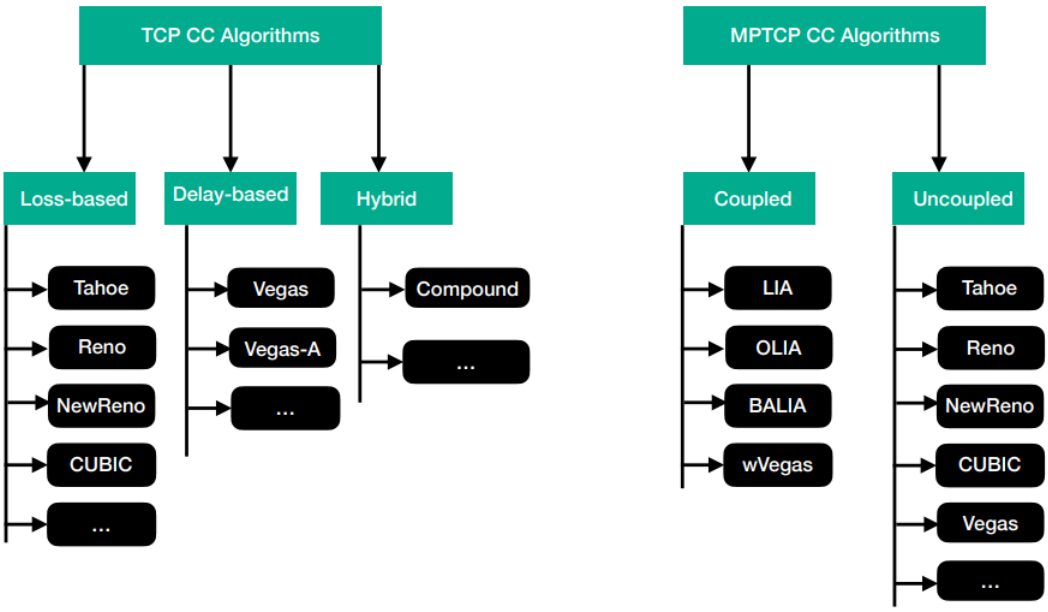

Figure 2.3: Congestion Control Algorithms

There are two most common categories of Congestion Control Algorithms based on whether they provide resource pooling or not. They include Uncoupled and Coupled Congestion Control Algorithms (Fig.2.3).

2.2.1 Uncoupled Congestion Control

Each MPTCP subflow functions as a separate TCP flow under uncoupled congestion control, executing a separate instance of the congestion control algorithm. To maximise the use of the path, cwnd and transmission rate are controlled independently for each subflow.

This approach is unfair because it will give the multi-path flow more bandwidth than is reasonable if multiple subflow pass through the same bottleneck link that is also used by the standard TCP flow.

2.2.2 Coupled Congestion Control

Coupled Congestion Control is a popular solution to the uncoupled congestion control's TCP-friendliness problem. To guarantee fairness to cross-traffic protocols in the event of shared bottlenecks, MPTCP modifies the cwnd at various rates. A number of coupled congestion control algorithms have been proposed in the literature like LIA [11], OLIA [12], BALIA [13] and wVegas[14].

2.2.3 Linked Increase Adaptation (LIA)

The first coupled congestion control algorithm was LIA. In LIA slow start (SS), fast retransmit and fast recovery algorithms are same as TCP-Reno. The increase part of the Additive Increase Multiplicative Decrease (AIMD) algorithm is linked among subflows during Congestion Avoidance (CA). Whenever subflow receives an acknowledgement, *cwnd* is increased by a proportion of the total congestion window. In LIA algorithm, when we receive *ACK* for *i'th* subflow then cwnd(i) will increase based on Eq. (1). In Eq. (2) α is also defined.

$$Min\left(\frac{\alpha}{cwnd_{total}}, \frac{1}{cwnd_i}\right) \qquad (1)$$

By the α Parameter calculation, it can be ensure that MPTCP bandwidth equal to TCP bandwidth on the best available path.

$$\alpha = cwnd_{total} * \frac{Max(\frac{cwnd_i}{RTT_i^2})}{(\Sigma_k \frac{cwnd_k}{RTT_k})^2} \qquad (2)$$

Whenever packet loss happens in subflow, cwnd(i) is decreased in the same way as in case of TCP-Reno.

$$cwnd_i/2 \qquad (3)$$

2.2.4 Opportunistic Linked-Increase Algorithm (OLIA)

TCP SS algorithm is used in OLIA with some modification. If there are multiple path created then SS threshold is set to one MSS and threshold will be same as TCP for one path. When multiple paths are available then this method will prevent the sending traffic over congested path. The OLIA increase algorithm updates the *cwnd* using the expression shown in Eq.(4), where L is the set of subflows with the largest congestion window, B is the set of subflows having bast path highest throughput and not having largest cwnd, and N is the number of subflows.

The increase algorithm uses two terms. The first term provides fairness, resource

pooling and congestion balancing and the second term, using parameter α, ensures non-flappiness and responsiveness.

$$\frac{\frac{cwnd_i}{RTT_i^2}}{(\Sigma_k \frac{cwnd_k}{RTT_k})^2} + \frac{\alpha_i}{cwnd_i} \tag{4}$$

Eq.(5) defined the parameter α. If a small *cwnd* exists in the best path then α will be positive and for maximum *cwnd* α will be negative. Parameter α will be zero if all the best path have largest *cwnd*.

$$\alpha_i = \begin{cases} \frac{1}{N \times |B \backslash L|} & , if\ i \epsilon\ B \backslash L\ \neq \phi \\ \frac{-1}{N \times |L|} & , if\ i \epsilon\ L\ and\ B \backslash L\ \neq \phi \\ 0 & , otherwise \end{cases} \tag{5}$$

2.2.5 Balanced Linked Adaptation (BALIA)

To maintain balanced performance (friendliness and responsiveness) and to generalise existing algorithms, the Congestion Avoidance algorithm is modified by BALIA in both the increase and decrease stages (AIMD).When several paths are open, the minimal SS threshold is set to one MSS. Only the subflow pathway's *cwnd* and RTT are used in the BALIA algorithm. For each subflow, the amount of increase is calculated according to Eq. (6) when an ACK arrives.

$$\frac{\frac{cwnd_i}{RTT_i}}{(\Sigma_k \frac{cwnd_k}{RTT_k})^2} \times \frac{\alpha_i + 1}{2} \times \frac{\alpha_i + 4}{5} \tag{6}$$

Using equation(7), BALIA lowers TCP-Reno cwnd between the range [1, 1.5].

$$\frac{cwnd_i}{2} \times \min(\alpha_i, 1.5) \tag{7}$$

In Formula (8), parameter α is specified. When BALIA employs a single path, the increment and decrement methods become identical to those used by standard TCP when α is set to 1.

$$\alpha_i = \frac{\max_k \frac{cwnd_k}{RTT_k}}{\frac{cwnd_i}{RTT_i}} \tag{8}$$

2.2.6 wVegas

Packet loss is an indication used by LIA, OLIA, and BALIA to show congestion. This implies that after a loss occurrence, traffic may divert from clogged paths. Based on TCP-Vegas, wVegas is a delay-based linked congestion control algorithm for MPTCP. The slow start and congestion avoidance stages of the congestion control process are modified by this algorithm. Congestion window is decreased and the queue starts to drain when a route queueing delay exceeds a user-defined threshold. Other paths offer *cwnd* the chance to grow in accordance with their predicted potential. In comparison to other coupled congestion control techniques, this makes wVegas more sensitive to network changes.

3 Problem Definition

3.1 Problem Statement

To evaluate and compare the performance of different congestion control algorithms 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 congestion control algorithms of Multipath TCP.
- To emulate Multipath TCP using Linux network namespaces and utilities.
- To perform experiment and collect statistics by configuring different path congestion control algorithms 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
```

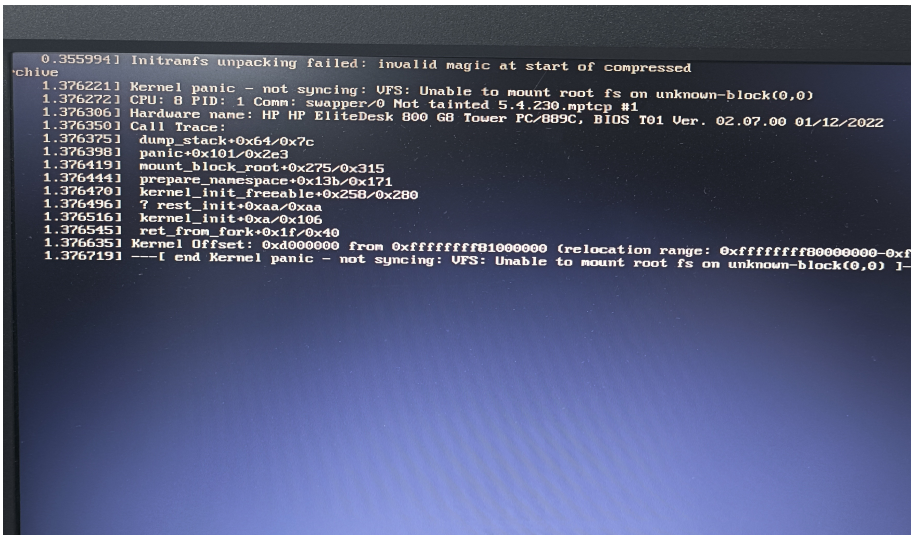

Figure 4.1: MPTCP Kernel not loaded into the memory in normal mode

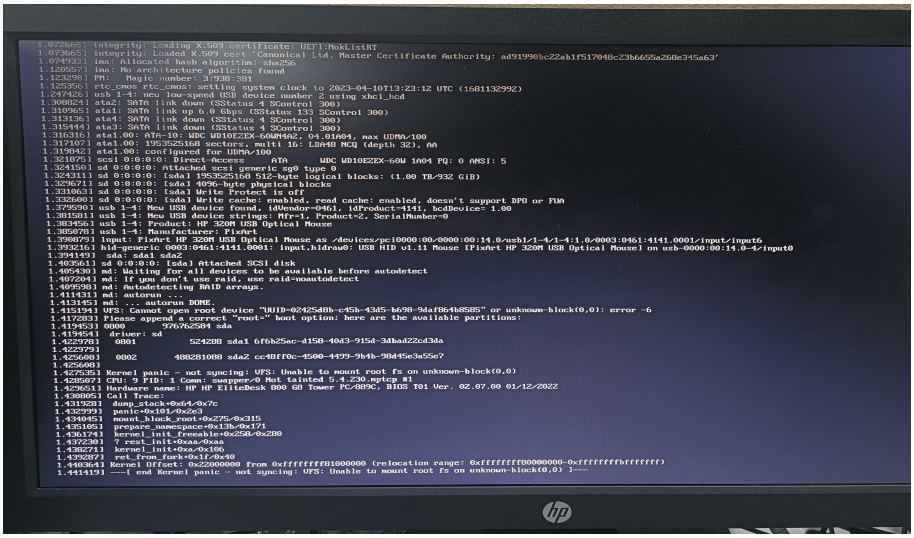

Figure 4.2: MPTCP Kernel not loaded into the memory in recovery mode

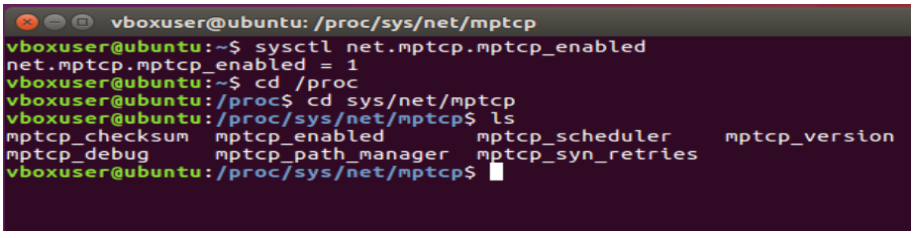

Figure 4.3: MPTCP linux kernel compilation

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

4.1.2 Creating the topology using Linux Network Namespaces

We used the topology shown in Figure 4.4 for our experiments. The topology consist of three hosts, namely h1, h2, and h3. Host h1 and h2 are connected via two virtual ethernet pairs and one link exist between hosts h2 and h3. In this topology, h1 will function as the client, and h3 will function as the server. The bsh script for creating this topology in available in Appendix A.

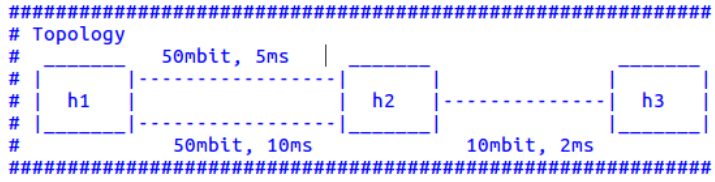

Figure 4.4: Three Nodes Topology

4.2 Results

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

4.2.1 Linked Increase Adaptation (LIA)

It is the first Coupled Congestion Control Algorithm.the LIA algorithm aims to achieve high network utilization while ensuring fairness and stability. We configure the LIA and path manager as full mesh using the following commands shown in Figure 4.5 :

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

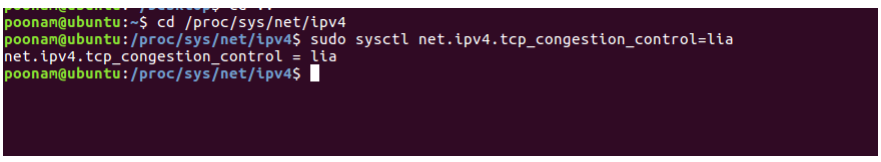

Figure 4.5: Set Congestion Control As LIA

On running the iperf for 20 seconds we obtained the output shown in Figure 4.6. The graph showing the variation in throughput is shown in Figure 4.7. The average Bandwidth using LIA is 9.25 Mbits/sec.

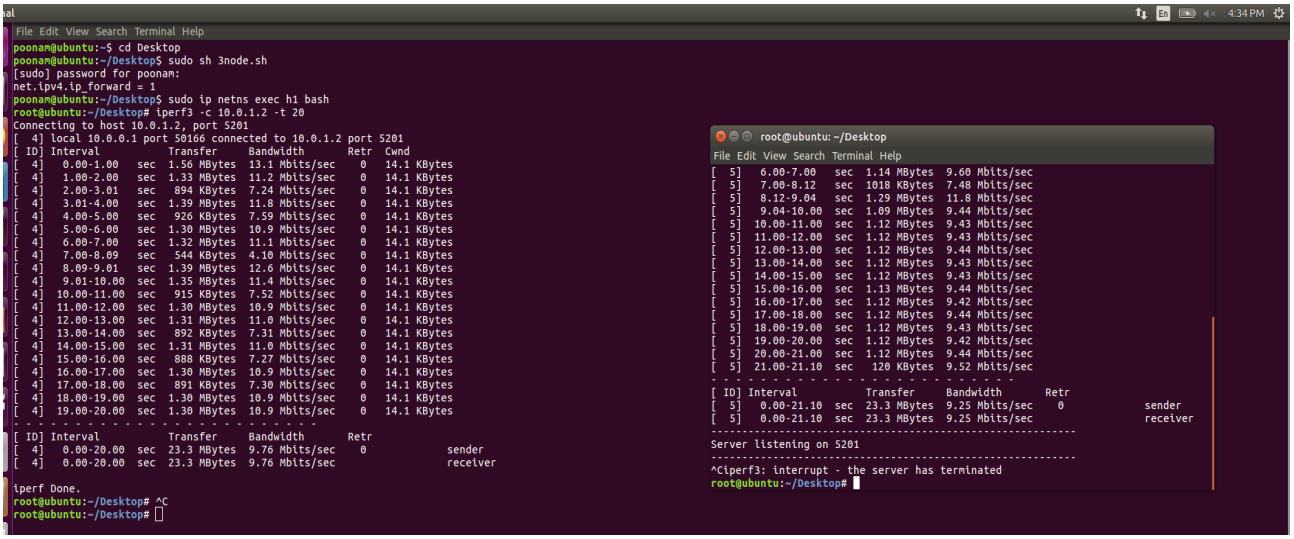

Figure 4.6: Output for LIA

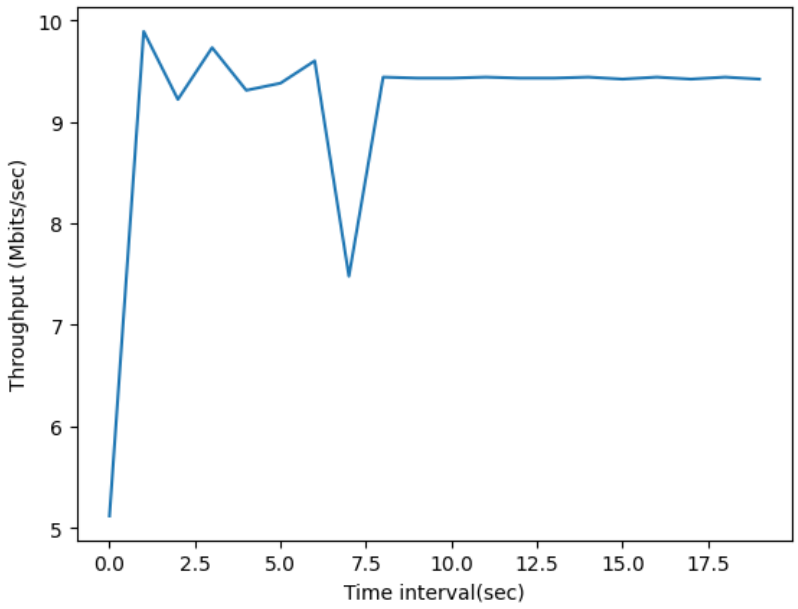

Figure 4.7: Throughput Analysis for LIA

4.2.2 Opportunistic Linked-Increase Algorithm (OLIA)

OLIA coupled congestion control algorithm is an extension of the LIA algorithm that aims to improve the scalability and responsiveness of the congestion control mechanism. We configure the OLIA and path manager as full mesh using the following commands shown in Figure 4.8:

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

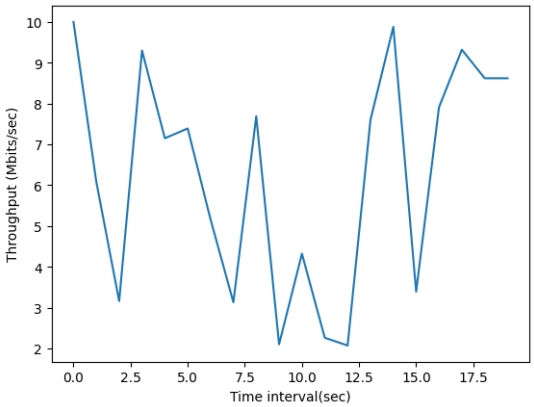

Figure 4.8: Set Congestion Control As OLIA

On running the iperf for 20 seconds we obtained the output shown in Figure 4.9. The graph showing the variation in throughput is shown in Figure 4.10. The average throughput using OLIA is 9.41 Mbits/sec.

4.2.3 Balanced Linked Adaptation (BALIA)

BALIA operates similarly to LIA and OLIA in that it divides flows into idle and active flows and adjusts the rate of active flows based on the availability of idle bandwidth. We configure the BALIA and path manager as full mesh using the following commands shown in Figure 4.11:

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

On running the iperf for 20 seconds we obtained the output shown in Figure 4.12. The graph showing the variation in throughput is shown in Figure 4.13. The average throughput obtained using BALIA is 9.41 Mbits/sec.

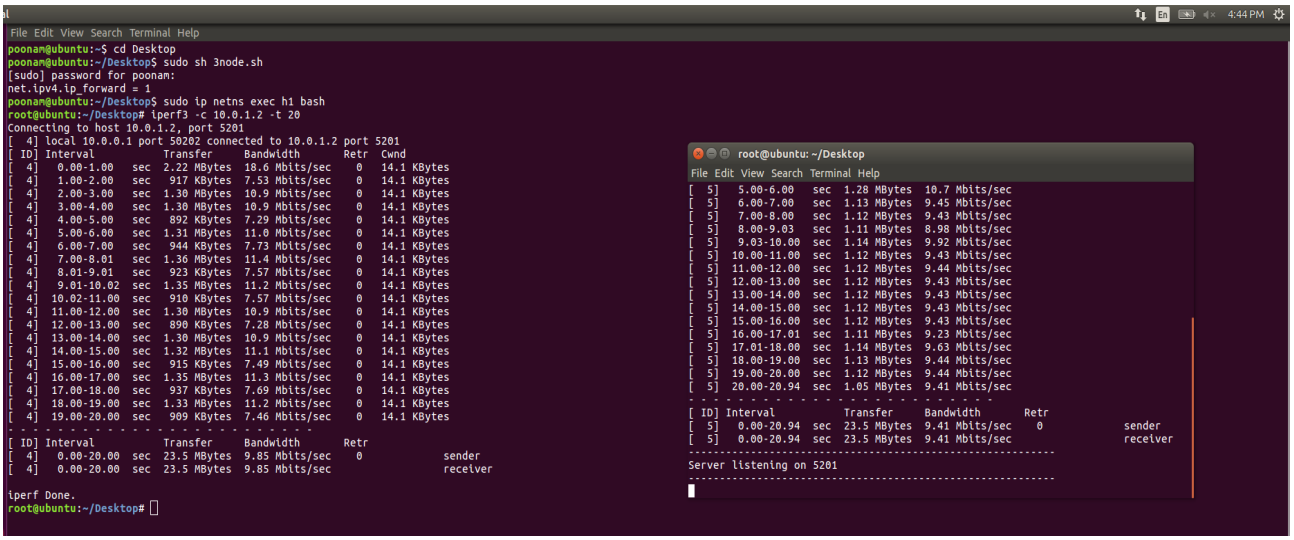

Figure 4.9: Output for OLIA

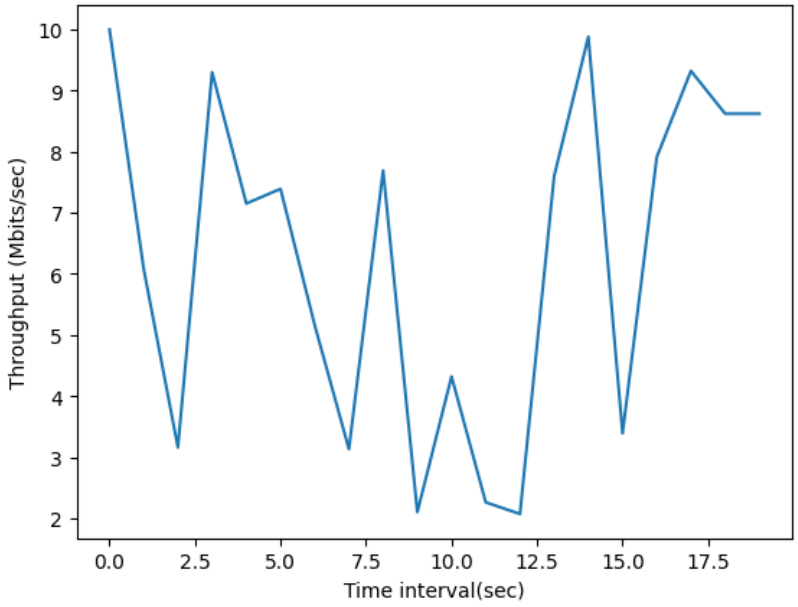

Figure 4.10: Throughput Analysis for OLIA

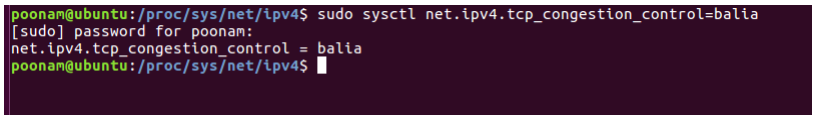

Figure 4.11: Set Congestion Control As BALIA

4.2.4 wVegas

In wVegas, each flow is assigned a weight that determines the proportion of the available bandwidth that the flow can use. This allows wVegas to allocate bandwidth

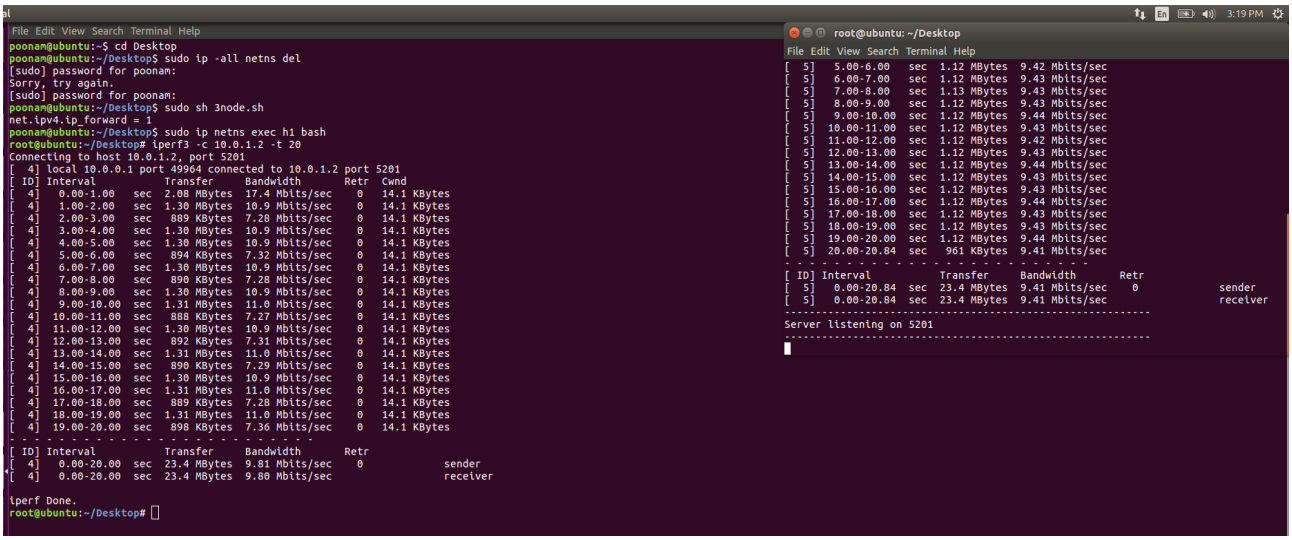

Figure 4.12: Output for BALIA

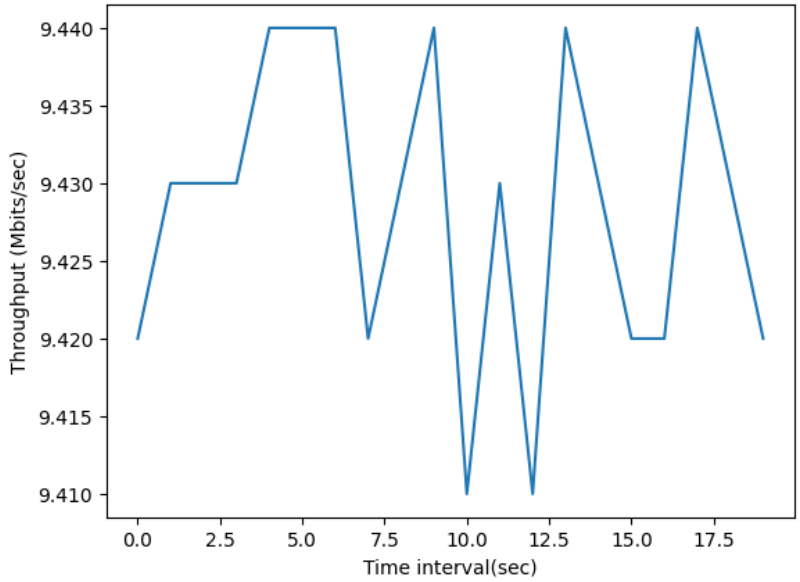

Figure 4.13: Throughput Analysis for BALIA

more fairly among competing flows.We configure the wVegas and path manager as full mesh using the following commands shown in Figure 4.14: commands:

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

On running the iperf for 20 seconds we obtained the output shown in Figure 4.15. The graph showing the variation in throughput is shown in Figure 4.16.The average throughput using wVegas is 6.05 Mbits/sec.

```
poonam@ubuntu:/proc/sys/net/ipv4$ sudo sysctl net.ipv4.tcp_congestion_control=wvegas
net.ipv4.tcp_congestion_control = wvegas
poonam@ubuntu:/proc/sys/net/ipv4$
```

Figure 4.14: Set Congestion Control As wVegas

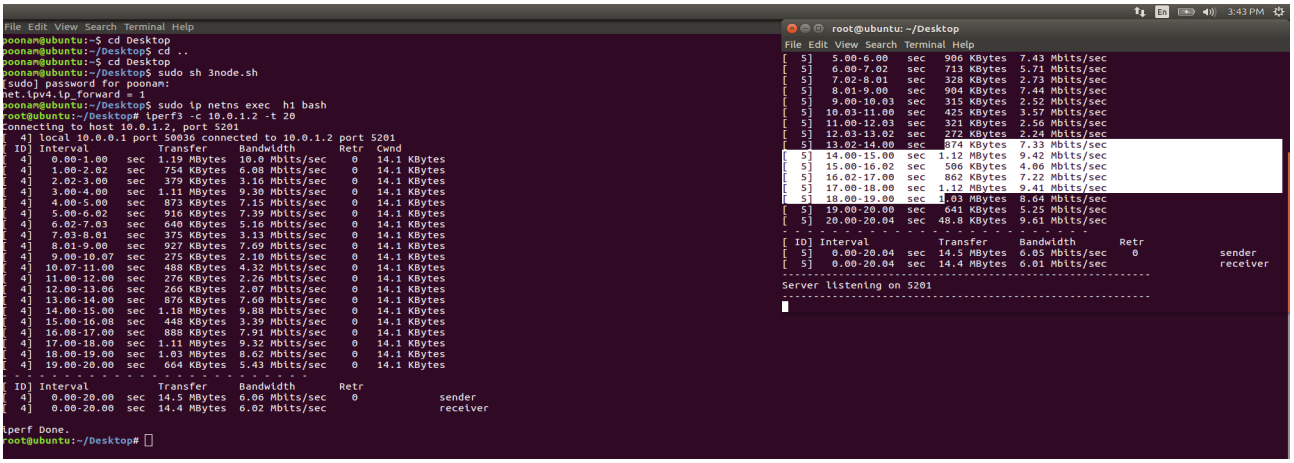

Figure 4.15: Output for wVegas

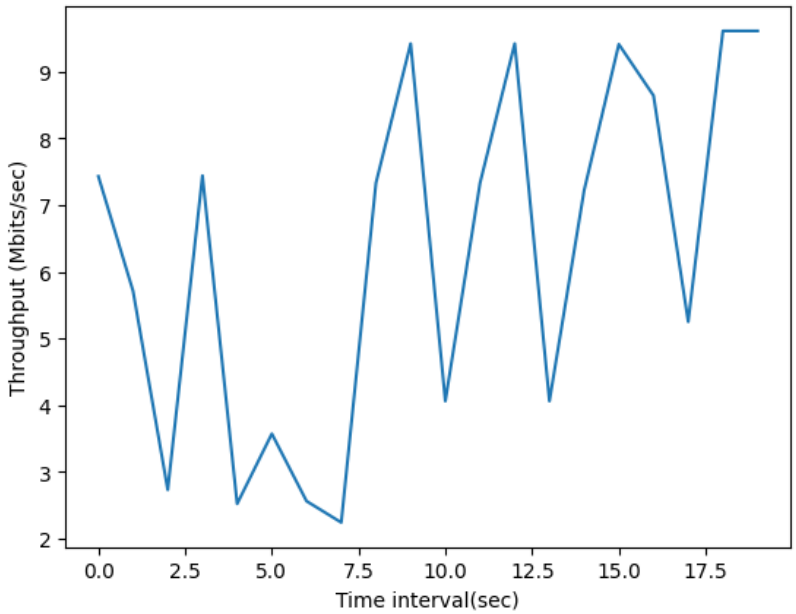

Figure 4.16: Throughput Analysis for wVegas

4.2.5 TCP CUBIC

TCP CUBIC (CUBIC TCP) is a congestion control algorithm for TCP that was developed to improve the performance of TCP in high-speed networks.We configure the TCP CUBIC and path manager as full mesh using the following commands shown

in Figure 4.17 :

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

```
poonam@ubuntu:~$ cd Desktop
poonam@ubuntu:~/Desktop$ cd ..
poonam@ubuntu:~$ cd /proc/sys/net/ipv4
poonam@ubuntu:/proc/sys/net/ipv4$ sudo sysctl net.ipv4.tcp_congestion_control=cubic
[sudo] password for poonam:
net.ipv4.tcp_congestion_control = cubic
poonam@ubuntu:/proc/sys/net/ipv4$
```

Figure 4.17: Set Congestion Control As CUBIC

On running the iperf for 20 seconds we obtained the output shown in Figure 4.18. The graph showing the variation in throughput is shown in Figure 4.19.Average throughput of CUBIC is 9.40 Mbits/sec.

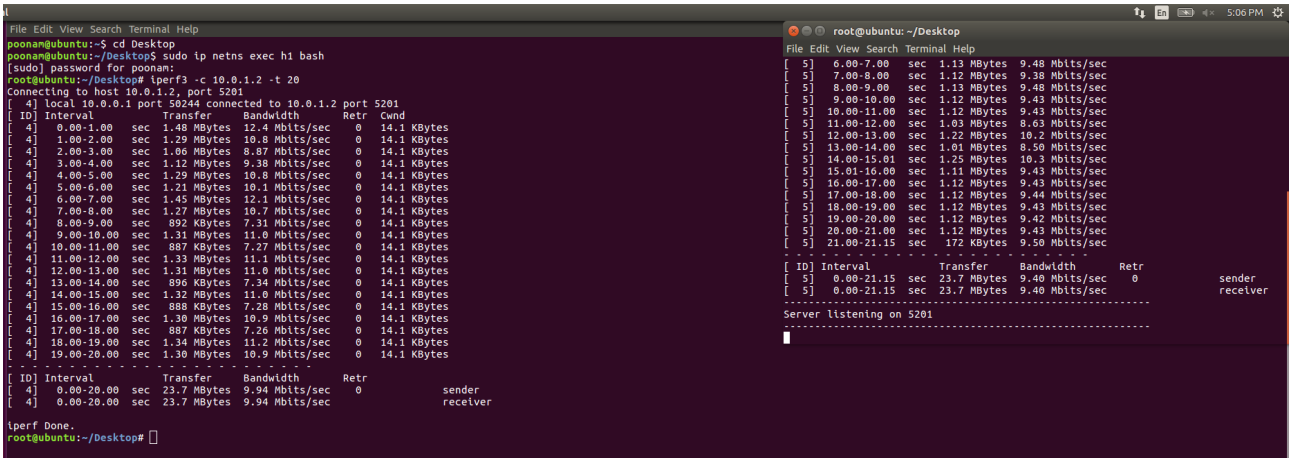

Figure 4.18: Output for CUBIC

Table 1 shows the result of different types of Congestion Control Algorithms and its average throughput. LIA, which is the standard congestion control algorithm of MPTCP, provides poor throughput when it compares with CUBIC algorithm. OLIA is giving better bandwidth compare to LIA.

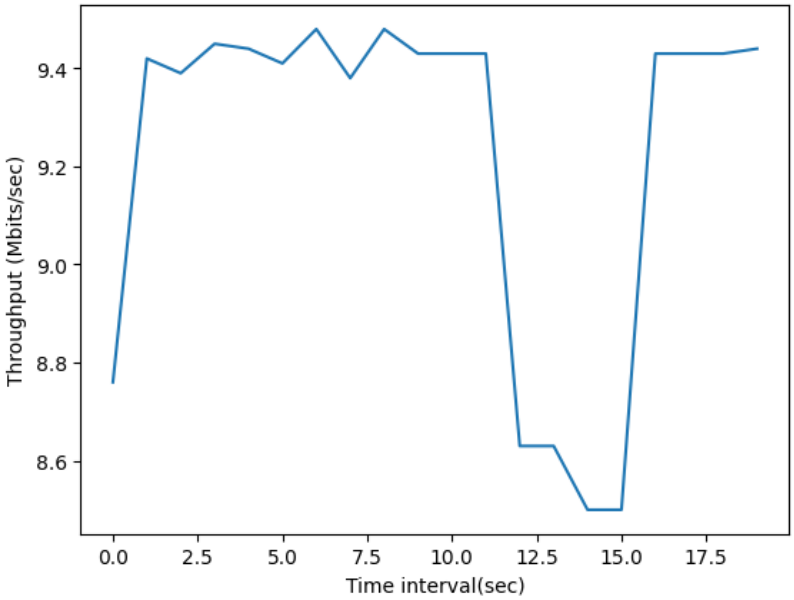

Figure 4.19: Throughput Analysis for TCP CUBIC

Congestion Control algorithm name	Average Throughput(Mbits/sec)
LIA	9.76
OLIA	9.85
BALIA	9.81
Wvgas	6.06
Cubic	9.94

Table 1: Average throughput

5 Conclusions and Future Work

In this work we performed a comparitive analysis of the congestion control algorithms of Multipath TCP. The performance evaluation findings for the LIA, OLIA, BALIA, wVegas and Cubic MPTCP congestion control algorithms are reported in this work. This report includes the analysis based on throughput. LIA, which is the standard congestion control algorithm of MPTCP, provides poor throughput when it compares with CUBIC algorithm. OLIA is giving better bandwidth compare to LIA. The evaluation based on variation in congestion window size is in progress. We have studied the performance of these algorithms in a ethernet. As part of the future work, we can perform a study of the congestion control algorithms in wireless network.

Bibliography

[1] MultiPath TCP - Linux Kernel implementation:https://www.multipath-tcp.org/

[2] RFC 8684:TCP Extensions for Multipath Operation with Multiple Addresses

[3] RFC6356:Coupled Congestion Control for Multipath Transport Protocols

[4] Telecom:An Analysis of MPTCP Congestion Control Farinaz Jowkarishasaltaneh * and Jason Bu

[5] Towards Evaluating Multipath TCP using Linux Tools and Utilities Suvam Mukherjee, Abhinaba Rakshit, Dayma Khan, Mohit P. Tahiliani Wireless Information Networking Group (WiNG) National Institute of Technology Karnataka, Surathkal, Mangalore, Karnataka - 575025, India

[6] EXPERIMENTAL ANALYSIS OF MPTCP CONGESTION CONTROL ALGORITHMS; LIA, OLIA AND BALIA Toshihiko Kato1, 2, Adhikari Diwakar1, Ryo Yamamoto1, Satoshi Ohzahata1 and Nobuo Suzuki2, 3

[7] NeST GitLab Repository: https://gitlab.com/nitk-nest/nest

[8] Hsu CH, Kremer U. IPERF: A framework for automatic construction of performance prediction models. InWorkshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France 1998 Oct 12.

[9] Hubert B. Linux advanced routing traffic control HOWTO. Netherlabs BV. 2002 Dec;1:99-107.

[10] Jones R. Netperf: A network performance monitoring tool. http://www.netperf.org/netperf/NetperfPage.html.2001.

[11] RFC 6356 - Coupled Congestion Control for Multipath Transport Protocols https://www.rfc-editor.org/rfc/rfc6356.html

[12] Khalili, R., Gast, N., Popovic, M. and Le Boudec, J.Y., 2013. MPTCP is not Pareto-optimal: Performance issues and a possible solution. IEEE/ACM Transactions On Networking, 21(5), pp.1651-1665.

[13] Peng, Q., Walid, A., Hwang, J. and Low, S.H., 2014. Multipath TCP: Analysis, design, and implementation. IEEE/ACM Transactions on networking, 24(1), pp.596-609.

[14] Cao, Y., Xu, M. and Fu, X., 2012, October. Delay-based congestion control for multipath TCP. In 2012 20th IEEE international conference on network protocols (ICNP) (pp. 1-10). IEEE.

A Appendix

A.1 Code for Creating the Topology

```
# Create three network namespaces: h1 h2 h3
ip netns add h1
ip netns add h2
ip netns add h3
# Create two virtual ethernet (veth) pairs between h1 and h2
ip link add eth1a netns h1 type veth peer name eth2a netns h2
ip link add eth1b netns h1 type veth peer name eth2b netns h2
ip link add eth3a netns h2 type veth peer name eth3b netns h3
# 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 eth1b
# Assign IP address to each interface on h2
ip netns exec h2 ip address add 10.0.0.2/24 dev eth2a
ip netns exec h2 ip address add 192.168.0.2/24 dev eth2b
ip netns exec h2 ip address add 10.0.1.1/24 dev eth3a
# Assign IP address to each interface on h3
ip netns exec h3 ip address add 10.0.1.2/24 dev eth3b
# 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 50mbit
ip netns exec h1 tc qdisc add dev eth1b 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 eth2a root netem delay 5ms rate 50mbit
ip netns exec h2 tc qdisc add dev eth2b root netem delay 10ms rate 50mbit
ip netns exec h2 tc qdisc add dev eth3a root netem delay 2ms rate 10mbit
# Set the data rate and delay on the veth devices at h3
ip netns exec h3 tc qdisc add dev eth3b root netem delay 2ms rate 10mbit
# Turn ON all ethernet devices
ip -n h1 link set lo up
ip -n h2 link set lo up
```

```
ip -n h3 link set lo up
ip -n h1 link set eth1a up
ip -n h1 link set eth1b up
ip -n h2 link set eth2a up
ip -n h2 link set eth2b up
ip -n h2 link set eth3a up
ip -n h3 link set eth3b up
# Enable IP forwarding
ip netns exec h2 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
ip netns exec h1 ip route add default via 10.0.0.2 dev eth1a table 1
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 192.168.0.2 dev eth1b table 2
ip netns exec h1 ip route add 192.168.0.0/24 dev eth1b scope link table 2
# Global Default route for h1
ip netns exec h1 ip route add default scope global nexthop via 10.0.0.2 dev eth1a
# Default route for h3
ip netns exec h3 ip route add default via 10.0.1.1 dev eth3b
```