CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER’S THESIS

Title:	Performance of MultiPath TCP on OpenWRT
Student:	Balaji Subramani
Supervisor:	Ing. Viktor Černý
Study Programme:	Informatics
Study Branch:	Computer Systems and Networks
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2018/19

Instructions

This thesis investigates a performance of the Multipath TCP protocol implemented on GNU linux kernel.
1. Make a survey about the Multipath protocol and choose a suitable linux distribution or an operating system derived from GNU linux kernel for your experiments.
2. Compile a GNU linux kernel with support of MultiPath TCP.
3. Prepare an OS image with support of MultiPath TCP for the network simulator GNS3.
4. After supervisor's agreement suggest an experiments for investigation of performance on load balancing between two and three interfaces.
5. After supervisor's agreement suggest an experiments for investigation of performance on congestion control in unstable networks.
6. Prepare your experiments in GNS3 network simulator.
7. Get your results on real hardware routers.

A result of this work will be performance comparison between the classic TCP protocol and MultiPath TCP protocol.

References

Will be provided by the supervisor.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 6, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of systems computer

Master's thesis

Performance of Multipath TCP on OpenWRT

Bc.Balaji Subramani

Supervisor: Ing. Viktor Cerny

29th June 2018

Acknowledgements

I would like to express my gratitude to my supervisor Ing. Viktor Cerny for accepting this challenging topic and the useful comments, remarks and engagement through the learning process of this master thesis. I thank him for motivating me all the time when I faced problems in implementing the project.

Next, I am gratefull to all the Professors in Czech Technical University who constantly helped me gain knowledge throughout the Masters Degree.

A special thanks to my family. Words cannot express how grateful I am to my Parents who helped me in pursuing my Masters degree. My mother has been an inspiration throughout my life. She has always supported my dreams and aspirations. I would like to thank her for all she is, and all she has done for me. I would like to thank my Father who supported me financially all these years for my studies.

Declaration

Citation of this thesis

Subramani, Balaji. *Performance of Multipath TCP on OpenWRT*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Multipath TCP (MPTCP) je pokročilým rozšířením stávajícího TCP protokolu, které dokáže nabídnout ví Ãce než standardní varianta. Transmission Control Protocol (TCP) je dosud nejrozšířenější metodou pro spolehlivou komunikaci přes rozsáhlé sítě. V sou Äasné době je protokol TCP omezen na komunikaci pouze jedinou originální cestou mezi zdrojem a cílem, i když je v dané chvíli k dispozici více alternativních cest. TCP nepodporuje multi homing. Tato vlastnost omezuje maximální možný datový tok, protože nelze využívat více linek najednou. MPTCP pomáhá překonat tento nedostatek. Protokol umožňuje rozdělit komunikaci do několika nezávislých TCP spojení a každé z nich může využívat jednu alternativní cestu k cíli komunikace. Díky tomu dokáže MPTCP zvýšit rychlost připojení, rovnoměrně rozdělovat zátěž mezi několik různých připojení k internetu a zároveň pomáhá udržet spojení i v případÄ výpadku některé z linek. V této práci budou vysvětleny rozdíly mezi MPTCP a TCP protokoly a zároveň jak MPTCP funguje. Dále bude podrobněji vysvětlen způsob jak zkompilovat linuxové jádro s podporou MPTCP v kombinaci se Shadowsocks pro operační systém LEDE. V další části práce bude navržena sada experimentů, které otestují vlastnosti MPTCP z hlediska datové propustnosti, přenosu velkých bloků dat, reakce na zvýšené komunikační zpoždění a reakce na zvýšenou ztrátovost komunikační linky. Hlavním cílem práce je analyzovat a vyhodnotit výkonnost MPTCP oproti TCP v operačním systému OpenWRT.

Klíčová slova Multipath TCP, TCP, OpenWrt, LEDE, Shadowsocks, vytváření sítí, šířka pásma, řízení přetížení, vyvažování zatížení.

Abstract

Multipath TCP (MPTCP) is an advanced development of TCP/IP network which has better features when compared to TCP. Transmission Control Protocol (TCP) is the so far widely used method for data transfer and communication over network. Currently, TCP communication is limited to a single path which means no matter how many paths are available, data is transmitted only through single path at once from the source to the destination. TCP does not support multi homing. This feature restricts the use of bandwidth over the network. MPTCP is an evolution of TCP that supports multi homing which transmits data over multiple paths. Data transfer over multiple paths is achieved by distributing data over several TCP subflows. Therefore, MPTCP provides better throughput, load balancing among available paths and better handling of network failure. In this thesis, I explain about the difference between TCP and MPTCP, and how MPTCP works. I also explained in detail about MPTCP enabled Kernel patch along with Shadowsocks in LEDE (OpenWrt). Various experiments are carried out based on bandwidth, delay, loss and bulk data transfer to analyze the performance of MPTCP over TCP. The main goal of this thesis is to identify the performance analysis of MPTCP over normal TCP connection in OpenWRT.

Keywords Multipath TCP, TCP, OpenWrt, LEDE, Shadowsocks, networking, bandwidth, congestion control, load balancing.

Contents

List of Figures

List of Tables

Introduction

The evolution of portable devices, such as mobile phones, tablets and laptops, made it important to always be reachable and have a high throughput connection. Having a fast and reliable internet connection is something that individuals and companies would want. At the same time, many devices developed the capability of connecting to the Internet with at least two different interfaces in each type of device, such as WiFi and 3G, or Ethernet and WiFi, in order to optimize the available communication infrastructures. Nowadays, there are multiple solutions to this, for example companies can get a fiber optic connection but it is still very expensive.

For many years the Transmission Control Protocol (TCP) has been a fundamental component of the Internet protocol stack and the most reliable communication protocol for data transmission. Although the basic model of TCP understands the essential mechanisms required to control flow and congestion, by itself it does not assure real-time delivery in cases of critical congestion connections or breaks on a support link. This mismatch between todayâs multipath networks and TCPâs single-path design creates tangible problems.

In this paper, I describe and analyse another solution which is less expensive than optic fiber and will make better use of brandwidth and be more tolerant to failures than Multihoming. This solution implies using a Linux Kernel with MultiPath TCP enabled and OpenWRT router. Multipath TCP (MPTCP) is a major modification to TCP that allows multiple paths to be used simultaneously by a single transport connection. The Multipath TCP protocol has recently been standardized by the IETF, and an implementation in the Linux kernel is available today.

Structure of the Thesis

First chapter is about basic concepts and tools used for the experiments in this thesis. Basic concepts that includes how regular TCP works, protocol stack of TCP and connection establishment and termination of TCP. This chapter also consists of different protocols and tools used in this thesis.

Second chapter describes in brief about Multipath TCP and how it works theoritically. It includes different MPTCP operations like connection establishment, data transfer and connection termination of MPTCP. Multipath Kernel configuration in Lede is also explained in this chapter.

Third chapter consists of the experimental setup of TCP and Multipath TCP in Openwrt. Implementation of Multipath TCP with detailed description of network, firewall and routing configuration is illustrated.

Fourth chapter is about the results obtained from various experiments of TCP and MPTCP. Experiments includes performance of time comparison, bandwidth, data transfer delay, packet loss, bulk data transfer and network failure.

Final chapter is about the conclusion of the result obtained from the different experiments.

Motivation and objectives

The mismatch between today's multipath networks and TCP's single-path design creates tangible problems. For instance, if a smartphone's WiFi loses signal, the TCP connections associated with it stalls, there is no way to migrate them to other working interfaces such as 3G. This makes mobility a frustrating experience for users. Various techniques have been proposed in the past to make use of multiple paths to establish a connection between two hosts or to optimize network resource usage by means of traffic balancing capabilities. I believe Multipath TCP is optimal solution which solves the drawbacks of TCP. The objective of this thesis is to perform a series of experiments to compare the performance of TCP and MPTCP in OpenWRT.

Problem statement

The motivation for using MPTCP in order to fulfill the problem description can be broken into three different parts namely

1. MPTCP can increase efficiency. The protocol can take advantage of additional interfaces i.e multiple paths.
2. MPTCP reduces congestion.
3. MPTCP adds redundancy e.g. if one link fails, the connection should stay active.

Chapter 1

Basic concepts and tools

1.1 Protocols

A network protocol defines rules and conventions for communication between network devices. Network protocols govern the end-to-end processes of timely, secure and managed data or network communication. The protocols used in this thesis is explained below.

1.1.1 TCP (Transmission Control Protocol)

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. TCP is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks and in interconnected systems of such networks. TCP is a connection-oriented end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. TCP is encapsulated within the data field of IP datagrams and TCP encapsulates higher level protocol data such as HTTP (web), SMTP (email) and many other protocols.[1]

1.1.1.1 TCP in protocol stack

Application Layer: The application layer consists of user invoked application programs that access services available across a TCP/IP Internet. The application program passes data in the requires form to the transport layer for delivery.

Transport Layer: The primary purpose of the transport layer is to provide communication from one application program to another. The transport software divides the stream of data being transmitted into smaller pieces called packets in the ISO terminology and passes each packet along with the destination information to the next layer for transmission. This layer consists

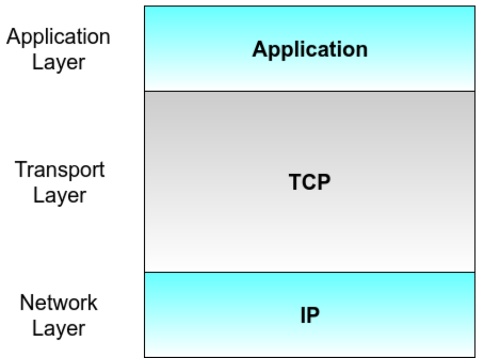


Figure 1.1: TCP in protocol stack

of Transport Control Protocol (TCP) which is a connection-oriented transport service and the User Datagram Protocol (UDP) which is a connectionless transport service.

Internet Protocol (IP) Layer: The Internet Protocol layer handles communication from one machine to another. It accepts requests to send data from the transport layer along with an identification of the machine to which the data is to be sent. It encapsulates the data into an IP datagram, fills in the datagram header, uses the routing algorithm to determine how to deliver the datagram, and passes the datagram to the appropriate device driver for transmission. The IP layer corresponds to the network layer in the OSI reference model. IP provides an unreliable connectionless packet-forwarding service which routes packets from one system to another.[2]

1.1.1.2 TCP connection establishment

A three-way handshake is a method used in a TCP/IP network to create a connection between a local host/client and server. The TCP connection establishment works as follows.

1. The client sends a SYN (synchronize) packet to the server, which has a random sequence number.
2. The server sends back a SYN-ACK packet, containing a random sequence number and an ACK (acknowledge) number acknowledging the client's sequence number. See figure 1.2
3. The client sends an ACK number to the server, acknowledging the server's sequence number.
4. The sequence numbers on both ends are synchronized. Both ends can now send and receive data independently.[3]

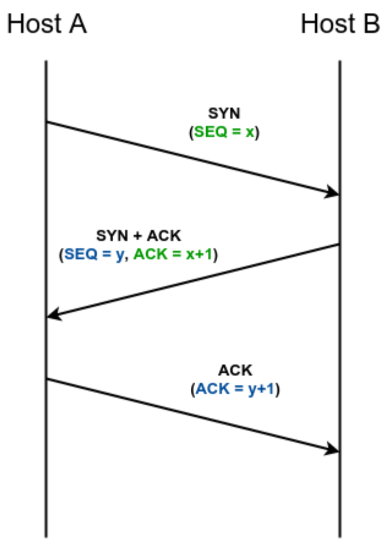


Figure 1.2: TCP connection establishment

1.1.1.3 TCP connection termination

TCP connection termination is implemented as follows:

1. One computer sends a FIN packet to the other computer including an ACK for the last data received (X).
2. The other computer sends an ACK number of X+1
3. It also sends a FIN with the sequence number of Y.
4. The originating computer sends a packet with an ACK number of Y+1. The connection is closed. See figure 1.3

Another way to close the connection is for one computer to send a packet with the RST (reset) bit set which will tell the other computer to immediately terminate the connection.[4]

1.1.2 MPTCP (Multipath TCP)

Multipath TCP (MPTCP) is a set of extensions to regular TCP to provide a Multipath TCP service which enables a transport connection to operate across multiple paths simultaneously. The design of Multipath TCP has been influenced by two main requirements namely application compatibility and network compatibility. Application compatibility implies that applications that today run over TCP should work without any change over Multipath TCP. Network compatibility means Multipath TCP must operate over any Internet path where TCP operates. The detailed explaination of MPTCP is explained in Chapter 2.

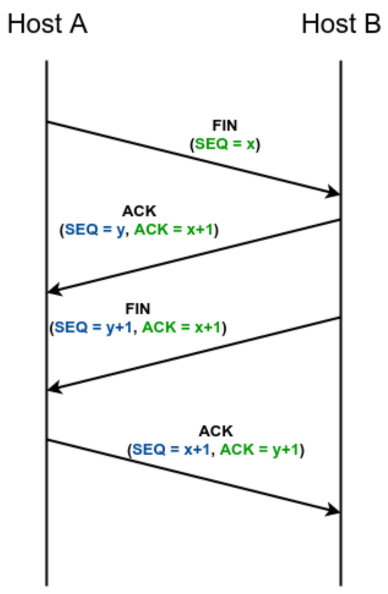


Figure 1.3: TCP connection termination

1.1.3 DHCP (Dynamic Host Configuration Protocol)

DHCP provides an automated way to distribute and update IP addresses and other configuration information on a network. A DHCP server provides this information to a DHCP client through the exchange of a series of messages known as the DHCP conversation or the DHCP transaction. If the DHCP server and DHCP clients are located on different subnets, a DHCP relay agent is used to facilitate the conversation.[5]

1.1.4 DNS (Domain Name System)

DNS syncs up domain names with IP addresses enabling humans to use memorable domain names while computers on the Internet can use IP addresses. For example, www.cvut.cz is resolved into machine-readable IP addresses like 204.13.248.115. DNS also provides other information about domain names, such as mail services.

1.1.5 ICMP (Internet Control Message Protocol)

The Internet Control Message Protocol (ICMP) is a supporting protocol in the Internet protocol suite. It is used by network devices including routers to send error messages and operational information indicating a requested service is not available or that a host or router could not be reached. ICMP is layer 4 protocol (it is on top layer 3 IP protocol), therefore it does not use TCP or

UDP for data delivery. ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems.

1.1.6 SSH (Secure Shell)

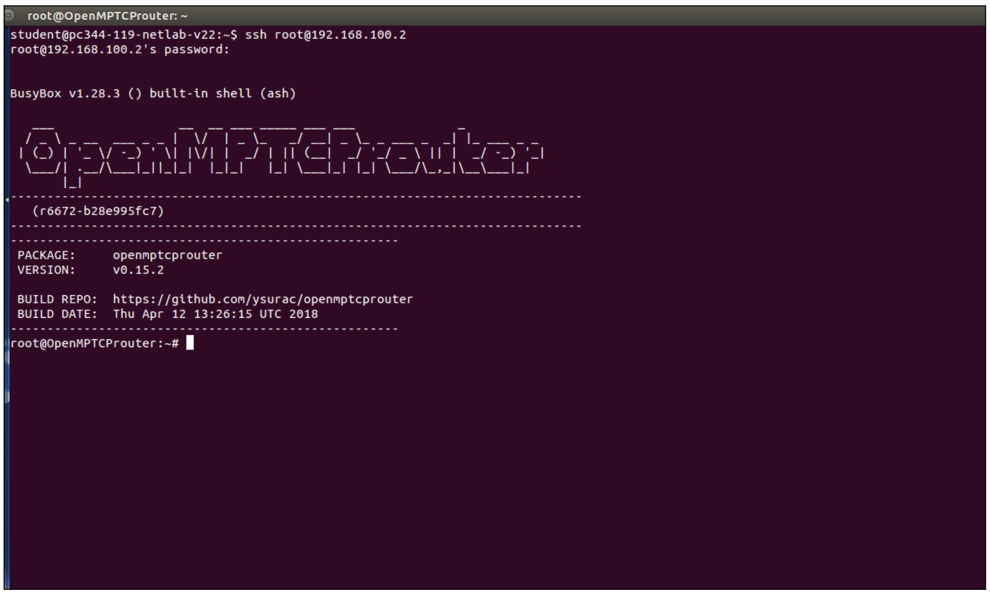
Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. The best known example application is for remote login to computer systems by users. SSH provides a secure channel over an unsecured network in a client-server architecture connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution also any network service can be secured with SSH.

1.1.7 HTTP (Hypertext Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTPS is the secured version of HTTP.

1.2 OpenWRT

The OpenWrt Project is a Linux operating system targeting embedded devices. Instead of trying to create a single static firmware, OpenWrt provides a fully writable filesystem with package management. This frees you from the application selection and configuration provided by the vendor and allows you to customize the device through the use of packages to suit any application. For developers, OpenWrt is the framework to build an application without having to build a complete firmware around it. For users this means the ability for full customization to use the device in ways never envisioned. OpenWrt is configured using a command-line interface (ash shell) or a web interface (LuCI).1.5 There are about 3500 optional software packages available for installation via the opkg package management system. As we enter 2018, both OpenWrt and the former LEDE project are happy to announce their unification under the OpenWrt name. The new, unified OpenWrt project will be governed under the rules established by the LEDE project. LEDE's fork and subsequent re-merge into OpenWrt will not alter the overall technical direction taken by the unified project. OpenWrt can run on various types of devices, including CPE routers, residential gateways, smartphones, pocket computers (e.g. Ben NanoNote) and laptops. It is also possible to run OpenWrt on personal computers, which are most commonly based on the x86 architecture. Moreover, the important ascpect why OpenWRT is chosen for this thesis is that it supports Multipath TCP.[6]



```
root@OpenMPTCProuter: ~
student@pc344-119-netlab-v22:~$ ssh root@192.168.100.2
root@192.168.100.2's password:


BusyBox v1.28.3 () built-in shell (ash)

   ___                   __  __ ___ _____ ___ ___            _
  / _ \ _ __  ___ _ _   |  \/  | _ \_   _/ __| _ \_ _ ___ _  _| |_ ___ _ _
 | (_) | '_ \/ -_) ' \  | |\/| |  _/ | || (__|  _/ '_/ _ \ || |  _/ -_) '_|
  \___/| .__/\___|_||_| |_|  |_|_|   |_| \___|_| |_| \___/\_,_|\__\___|_|
       |_|
-----------------------------------------------------------------------------
   (r6672-b28e995fc7)
-----------------------------------------------------------------------------
-----------------------------------------------------
 PACKAGE:     openmptcprouter
 VERSION:     v0.15.2

 BUILD REPO:  https://github.com/ysurac/openmptcprouter
 BUILD DATE:  Thu Apr 12 13:26:15 UTC 2018
-----------------------------------------------------
root@OpenMPTCProuter:~#
```

Figure 1.4: OpenMPTCPRouter - An OpenWRT with MPTCP support

1.2.1 Reasons to Use OpenWrt

The main advantage of OpenWRT is adaptation, simplicity and customizable under specific requirements of a particular project. OpenWRT operation system is the right choice for network devices which are not limited by standard router functions. However, this kind of distribution kit will not match in case of strict RAM and flash memory requirements.[7] A wide range of reasons to use OpenWRT is discussed below.

1.2.1.1 Performance and Stability

OpenWrt is designed by network professionals and others who care about the performance of their network. OpenWrt incorporates many algorithms from recent research that perform far better than vendor-supplied firmware. OpenWrt is stable, and operates reliably for long periods of time. OpenWrt reduces latency/lag and increased network throughput via bufferbloat control algorithms.[8]

1.2.1.2 Extensibility

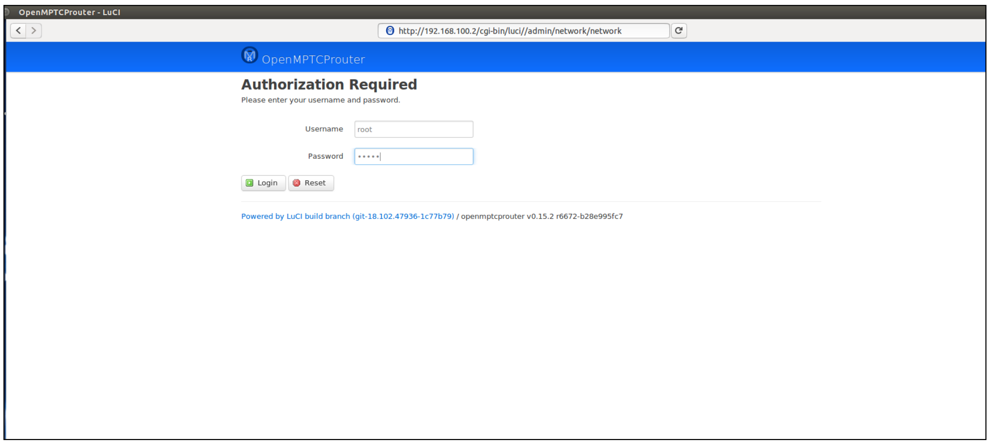
While vendor firmware for a router ships with a fixed set of capabilities, OpenWrt provides more than 3000 packages ready to be installed. Some of the more popular packages allows us to:

- Secure access to your home network when away via OpenVPN Server.

- Prevent your ISP from snooping on your DNS requests via DNSCrypt.
- Control access using the time limits and parental controls.
- Reduce latency/lag (bufferbloat) even during heavy traffic with Smart Queue Management

1.2.1.3 Strong Community Support

OpenWrt team members are regular participants on the LEDE Forum, OpenWrt Developer and OpenWrt Admin mailing lists. You can interact directly with developers, volunteers managing the software modules and with other long-time OpenWrt users, drastically increasing the chances you will solve the issue at hand. Frequent updates and enhancement of Luci makes OpenWRT easy to use. See figure 1.5



http://192.168.100.2/cgi-bin/luci//admin/network/network

Authorization Required

Please enter your username and password.

Username: root

Password: •••••

Powered by LuCI build branch (git-18.102.47936-1c77b79) / openmptcprouter v0.15.2 r6672-b28e995fc7

Figure 1.5: OpenMPTCPRouter Luci

1.2.1.4 Security

OpenWrt is an open source software. Many developers from all over the world review the code before it's released. The security advantages includes the following aspects.

- No hidden backdoors left by hardware vendors.
- OpenWrt is actively updated so any vulnerabilities are closed shortly after they are discovered.
- Many of the older devices are supported by OpenWrt and can enjoy security OpenWrt brings, long after vendors stop releasing firmware updates.

- OpenWrt is resistant to common vulnerabilities thanks to its Linux OS which is unaffected by many common attacks.

1.2.1.5 Open Source

OpenWrt is provided for free through its GPL license. There are no subscription or licensing fees.

1.3 Tools used

This section is about the tools used in this thesis to make Multipath TCP work. Also explains about the different tools used to measure the performance of TCP and MPTCP.

1.3.1 Shadowsocks proxy

Shadowsocks is an open-source encrypted proxy project, widely used in mainland China to circumvent Internet censorship. It was created in 2012 by a Chinese programmer named "clowwindy" and multiple implementations of the protocol have been made available since. Typically, the client software will open a socks5 proxy on the machine it is run, which internet traffic can then be directed towards which is similarly to an SSH tunnel. Unlike an SSH tunnel, shadowsocks can also proxy UDP traffic.[9]

Unlike a VPN, Shadowsocks is highly customizable and uses HTTPS to disguise online movements. This technology doesnât encrypt traffic like a VPN nor does it send all your traffic through a specific server. Instead, it works with a number of different TCP connections, making it much faster than its rivals and much more difficult for authorities to detect as a masked connection.
Version: Shadowsocks 3.0.0

1.3.1.1 Server implementations

The original Python implementation can still be installed with Pip (package manager), but the contents of its GitHub repository have been removed. Other server implementations include one in Go, C using the libev event loop library, C++ with a Qt GUI, and Perl. The Go and Perl implementations are not updated regularly and may be abandoned.

1.3.1.2 Client implementations

All of the server implementations listed above also support operating in client mode. There are also client-only implementations available for Windows (shadowsocks-win), macOS (ShadowsocksX-NG), Android (shadowsocks-android) and iOS (Wingy). Many clients including shadowsocks-win and shadowsocks-android support redirecting all system traffic over Shadowsocks, not just applications that have been explicitly configured to do so allowing Shadowsocks to be used similarly to a VPN.

1.3.2 iPerf

Iperf is a widely used tool for network performance measurement and tuning. It is significant as a cross-platform tool that can produce standardized performance measurements for any network. Iperf has client and server functionality, and can create data streams to measure the throughput between the two ends in one or both directions. Typical Iperf output contains a time-stamped report of the amount of data transferred and the throughput measured. Basically, iPerf operates over TCP optionaly over UDP.[10]
Version: iperf 3.0.11

1.3.2.1 iperf server

```
iperf3 -s
```

1.3.2.2 iperf client

```
iperf3 -c 192.168.200.1 -t 60
```

1.3.3 Traffic control tc

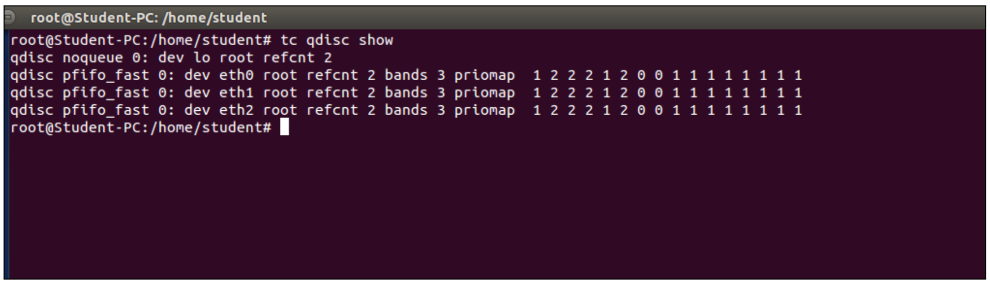
Linux offers a very rich set of tools for managing and manipulating the transmission of packets. The larger Linux community is very familiar with the tools available under Linux for packet mangling and firewalling (netfilter, and before that, ipchains) as well as hundreds of network services which can run on the operating system. Few inside the community and fewer outside the Linux community are aware of the tremendous power of the traffic control subsystem which has grown and matured under kernels 2.2 and 2.4.[11]
Version: tc 1.0.2

qdisc: Simply put, a qdisc is a scheduler. Every output interface needs a scheduler of some kind, and the default scheduler is a FIFO. Other qdiscs available under Linux will rearrange the packets entering the scheduler's queue in accordance with that scheduler's rules.

1.3.3.1 Bandwidth limit

In order to limit the egress bandwidth we can use the following options in traffic control. See figure 1.7

- **qdisc:** modify the scheduler (aka queuing discipline)

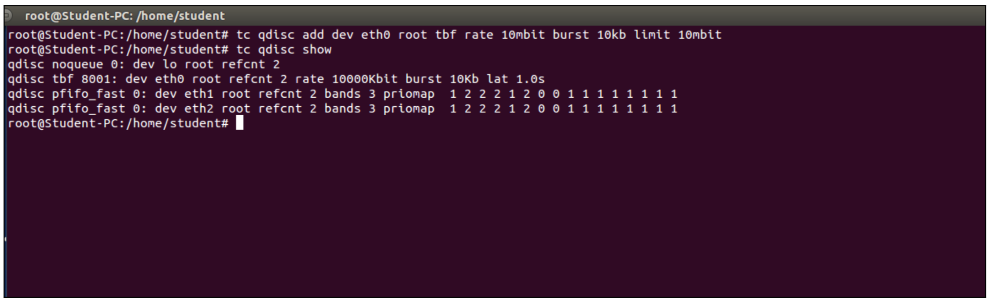


```
root@Student-PC: /home/student
root@Student-PC:/home/student# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth2 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@Student-PC:/home/student#
```

Figure 1.6: Traffic control without any bandwidht limit, delay or loss

- **add/del/change:** add a new rule, delete a rule or modify existing rule.
- **dev eth0:** rules will be applied on device eth0
- **root:** modify the outbound traffic scheduler (aka known as the egress qdisc)
- **tbf:** use the token buffer filter to manipulate traffic rates
- **rate:** sustained maximum rate
- **burst:** maximum allowed burst
- **latency:** packets with higher latency get dropped

```
tc qdisc add dev eth0 root tbf rate 10mbit burst 10kb
limit 10mbit
tc qdisc show
```



```
root@Student-PC: /home/student
root@Student-PC:/home/student# tc qdisc add dev eth0 root tbf rate 10mbit burst 10kb limit 10mbit
root@Student-PC:/home/student# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc tbf 8001: dev eth0 root refcnt 2 rate 10000Kbit burst 10Kb lat 1.0s
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth2 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@Student-PC:/home/student#
```

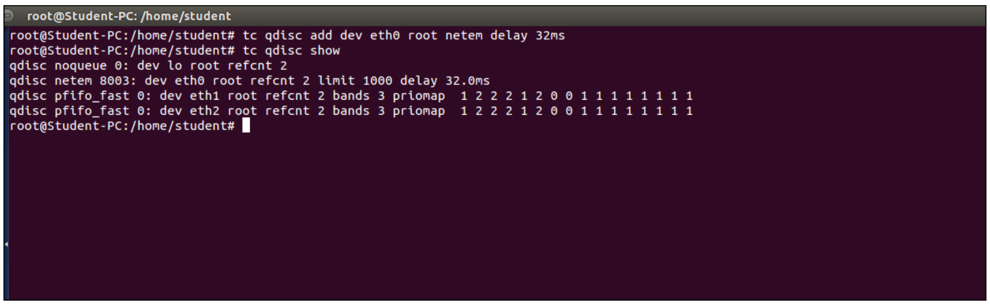
Figure 1.7: Traffic control limiting bandwidth

1.3.3.2 Delay

Delay in networking is the length of time between the intiation of a transaction by a sender and the first response received by the sender. Length of time required to move a packet from source to destination over a given path. Created by an application, handed over to the OS, passed to a network card (NIC), encoded, transmitted over a physical medium and received by an intermediate device (switch, router). The option for setting delay using traffic control is shown below [12]

- **qdisc:** modify the scheduler (aka queuing discipline)
- **add/del/change:** add a new rule, delete a rule or modify existing rule.
- **dev eth0:** rules will be applied on device eth0
- **root:** modify the outbound traffic scheduler (aka known as the egress qdisc)
- **netem:** use the network emulator to emulate a WAN property
- **delay:** the network property that is modified
- **64ms:** introduce delay of 64 ms 1.8

```
tc qdisc add dev eth0 root netem delay 64ms
tc qdisc show
```



```
root@Student-PC: /home/student
root@Student-PC:/home/student# tc qdisc add dev eth0 root netem delay 32ms
root@Student-PC:/home/student# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8003: dev eth0 root refcnt 2 limit 1000 delay 32.0ms
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth2 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@Student-PC:/home/student#
```

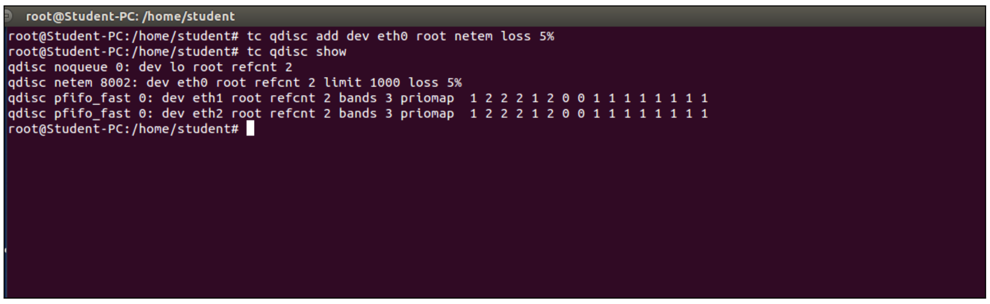
Figure 1.8: Traffic control setting Delay

1.3.3.3 Loss

Packet loss is the discarding of packets in a network when a router or other network device is overloaded and cannot accept additional packets at a given moment. The losses are usually due to congestion on the network and buffer overflows on the end-systems. The command options for setting packet loss using tc is shown below.

- **qdisc:** modify the scheduler (aka queuing discipline)
- **add/del/change:** add a new rule, delete a rule or modify existing rule.
- **dev eth0:** rules will be applied on device eth0
- **root:** modify the outbound traffic scheduler (aka known as the egress qdisc)
- **netem:** use the network emulator to emulate a WAN property
- **loss:** percentage of packet loss 1.9

```
tc qdisc add dev eth0 root netem loss 5%
tc qdisc show
```



```
root@Student-PC: /home/student
root@Student-PC:/home/student# tc qdisc add dev eth0 root netem loss 5%
root@Student-PC:/home/student# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8002: dev eth0 root refcnt 2 limit 1000 loss 5%
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth2 root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@Student-PC:/home/student#
```

Figure 1.9: Traffic control setting Loss

1.3.4 Netcat

Netcat (nc) is a computer networking utility for reading from and writing to network connections using TCP or UDP. Netcat is designed to be a dependable back-end that can be used directly or easily driven by other programs and scripts. At the same time, it is very useful network debugging and investigation tool, since it can produce almost any kind of connection its user could need and has a number of built-in capabilities.[13]

1.3.4.1 Netcat server

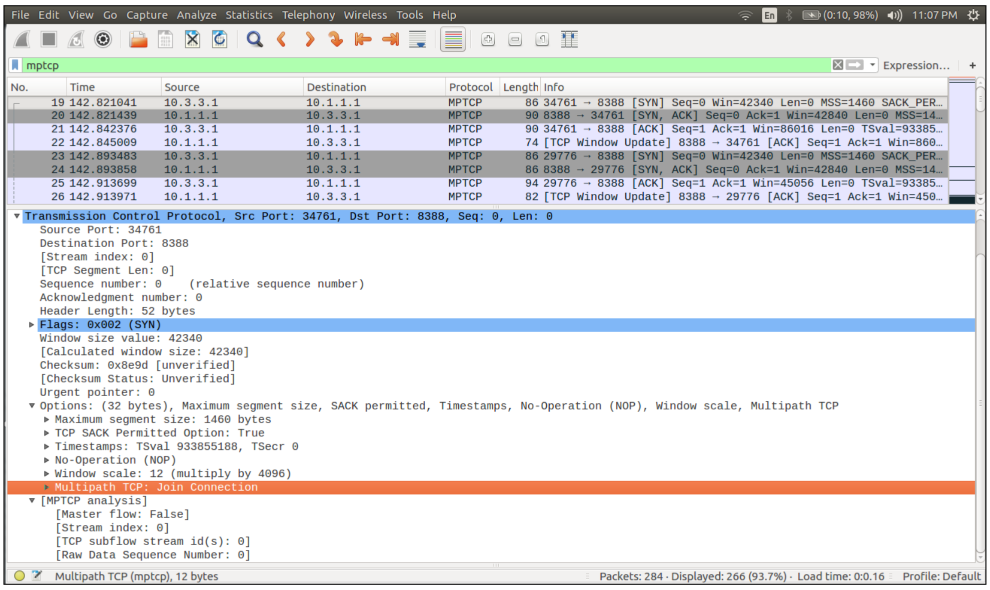
```
nc -l <port_number>
Eg: nc -l 2222
```

1.3.4.2 Netcat client

```
nc <source_ip_address> <source_port>
Eg: nc 192.168.200.1 1234
```

1.3.5 Wireshark

Wireshark is a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color coding and other features that let you dig deep into network traffic and inspect individual packets. Wireshark is a free application that allows you to capture and view the data traveling back and forth on your network, providing the ability to drill down and read the contents of each packet filtered to meet your specific needs. It is commonly utilized to troubleshoot network problems as well as to develop and test software. This open-source protocol analyzer is widely accepted as the industry standard winning its fair share of awards over the years.[14]


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
En (0:10, 98%) 11:07 PM
mptcp
Expression…

```
No.  Time        Source    Destination  Protocol Length Info
  19 142.821041  10.3.3.1  10.1.1.1     MPTCP    86 34761 → 8388 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PER…
  20 142.821439  10.1.1.1  10.3.3.1     MPTCP    90 8388 → 34761 [SYN, ACK] Seq=0 Ack=1 Win=42840 Len=0 MSS=14…
  21 142.842376  10.3.3.1  10.1.1.1     MPTCP    90 34761 → 8388 [ACK] Seq=1 Ack=1 Win=86016 Len=0 TSval=93385…
  22 142.845009  10.1.1.1  10.3.3.1     MPTCP    74 [TCP Window Update] 8388 → 34761 [ACK] Seq=1 Ack=1 Win=860…
  23 142.893483  10.3.3.1  10.1.1.1     MPTCP    86 29776 → 8388 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PER…
  24 142.893858  10.1.1.1  10.3.3.1     MPTCP    86 8388 → 29776 [SYN, ACK] Seq=0 Ack=1 Win=42840 Len=0 MSS=14…
  25 142.913699  10.3.3.1  10.1.1.1     MPTCP    94 29776 → 8388 [ACK] Seq=1 Ack=1 Win=45056 Len=0 TSval=93385…
  26 142.913971  10.1.1.1  10.3.3.1     MPTCP    82 [TCP Window Update] 8388 → 29776 [ACK] Seq=1 Ack=1 Win=450…

▼ Transmission Control Protocol, Src Port: 34761, Dst Port: 8388, Seq: 0, Len: 0
    Source Port: 34761
    Destination Port: 8388
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    Header Length: 52 bytes
  ▶ Flags: 0x002 (SYN)
    Window size value: 42340
    [Calculated window size: 42340]
    Checksum: 0x8e9d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▼ Options: (32 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale, Multipath TCP
    ▶ Maximum segment size: 1460 bytes
    ▶ TCP SACK Permitted Option: True
    ▶ Timestamps: TSval 933855188, TSecr 0
    ▶ No-Operation (NOP)
    ▶ Window scale: 12 (multiply by 4096)
    ▶ Multipath TCP: Join Connection
  ▼ [MPTCP analysis]
      [Master flow: False]
      [Stream index: 0]
      [TCP subflow stream id(s): 0]
      [Raw Data Sequence Number: 0]

Multipath TCP (mptcp), 12 bytes     Packets: 284 · Displayed: 266 (93.7%) · Load time: 0:0.16     Profile: Default
```

Figure 1.10: MPTCP support in Wireshark

1.3.6 GNS3

GNS3 is a free graphical network simulator capable of emulating a number of network devices. This makes it possible for anyone to quickly and easily spin up network hardware for testing and educational purposes without the heavy expense of physical hardware. Supported devices include Cisco routers and firewalls, Juniper routers and frame-relay switches. GNS3 works by using real Cisco IOS images which are emulated using a program called Dynamips. GNS3 is really like the GUI part of the overall product. With this GUI, users get an easy to use interface that allows them to build complex labs consisting of a variety of supported Cisco routers.[15]

GNS3 allows the same type of emulation using Cisco Internetwork Operating Systems. It allows you to run a Cisco IOS in a virtual environment on your computer. GNS3 is a graphical front end to a product called Dynagen. Dynamips is the core program that allows IOS emulation. Dynagen runs on top of Dynamips to create a more user friendly, text-based environment. A user may create network topologies using simple Windows ini-type files with Dynagen running on top of Dynamips. GNS3 takes this a step further by providing a graphical environment.[16]

CHAPTER 2

Multipath TCP (MPTCP)

2.1 How Multipath TCP works

Generally a computer device or any communication device is attached to one single network. However, these days we have our computing devices that is attached to multiple networks at one time. A good example of multi homing is a tablet or a multimedia phone that is attached to both mobile networks (either through GPRS or a 3G connection) and a local WiFi network. The technology that can be used to achieve this is called MPTCP or MultiPath TCP. Multipath TCP is a set of extensions to standard TCP that allows connections to use multiple paths simultaneously. Multiple regular TCP connections also known as subflows are aggregated into a single Multipath TCP connection.[17]

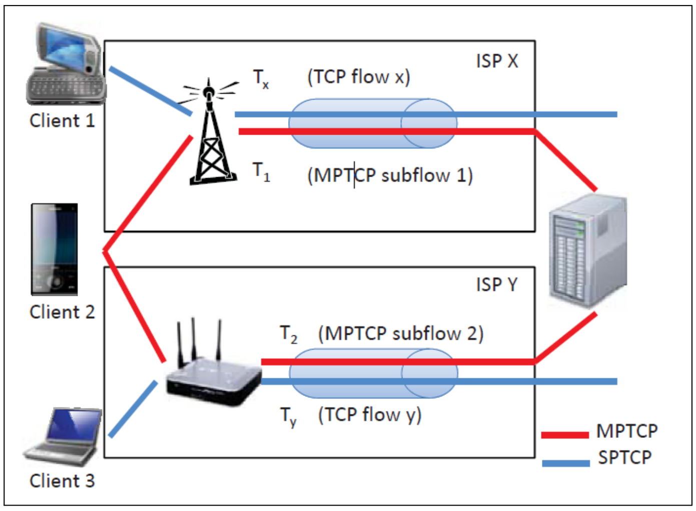


Figure 2.1: How MPTCP works

2.1.1 Multipath TCP in protocol stack

In regular TCP, an application initiates communication by opening a connection via an application programming interface (API) provided by the operating system. The TCP layer communicates in its turn with the IP layer. In Multipath TCP, the TCP layer has been extended. Upwards, the Multipath TCP layer exposes an interface that is perceived as regular TCP by the application. In the protocol stack, there are several TCP subflows in the transport layer rather than one TCP connection.[18]

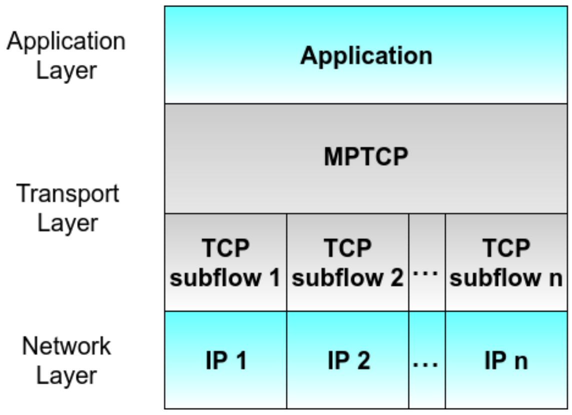


Figure 2.2: MPTCP in protocol stack

2.1.2 Multipath TCP modes

Multipath TCP can be used with different modes based the number of connection. The modes are listed below.

Table 2.1: MPTCP different working nodes

on	No special config
master	Like "on" but also set the default route for all other traffic (use it for one interface)
off	Disable the interface for MPTCP (default option)
backup	Use this interface but don't forward traffic until no other interface are available. Subflows are created on both interfaces, but data only flows on one of them. (faster switch since connection already established.
handover	Establish a connection only if no other interface available (slower switch since connection not established until all interfacs are down)[19]

2.2 MPTCP protocol

This section provides a high-level summary of normal operation of MPTCP.

- To a non MPTCP aware application, MPTCP will behave the same as normal TCP. Extended APIs could provide additional control to MPTCP aware applications. An application begins by opening a TCP socket in the normal way. MPTCP signaling and operation are handled by the MPTCP implementation.

- An MPTCP connection begins similarly to a regular TCP connection. An MPTCP connection is established between addresses A1 and B1 on Hosts A and B, respectively.

- If extra paths are available, additional TCP sessions (termed MPTCP "subflows") are created on these paths and are combined with the existing session which continues to appear as a single connection to the applications at both ends.

- MPTCP identifies multiple paths by the presence of multiple addresses at hosts. Combinations of these multiple addresses equate to the additional paths. In the example, other potential paths that could be set up are A1<->B2 and A2<->B2. Although this additional session is shown as being initiated from A2, it could equally have been initiated from B1.

- The discovery and setup of additional subflows will be achieved through a path management method.

- MPTCP adds connection-level sequence numbers to allow the reassembly of segments arriving on multiple subflows with differing network delays.

- Subflows are terminated as regular TCP connections with a four- way FIN handshake. The MPTCP connection is terminated by a connection level FIN.

2.2.1 Multipath TCP options

Multipath TCP uses options that are described in detail in RFC 6824. All Multipath TCP options are encoded as TCP options with Option Kind is 30, as reserved by IANA.

The Multipath TCP option has the Kind (30), length (variable) and the remainder of the content begins with a 4-bit subtype field, for which IANA has created and will maintain a sub-registry entitled "MPTCP Option Subtypes" under the "Transmission Control Protocol (TCP) Parameters" registry. Those subtype fields are defined as follows:

Table 2.2: MPTCP options

Value	Symbol	Name
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal (Data ACK and data sequence mapping)
0x3	ADD_ADDR	Add Address
0x4	REMOVE_ADDR	Remove Address
0x5	MP_PRIO	Change Subflow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close
0xf	(PRIVATE)	Private Use within controlled testbeds

2.2.2 Initiating MPTCP connection (MP_CAPABLE)

This is the same signaling as for initiating a normal TCP connection, but the SYN, SYN/ACK, and ACK packets also carry the MP_CAPABLE option. See figure 2.3 This is variable length and serves multiple purposes. Firstly, it verifies whether the remote host supports Multipath TCP and secondly, this option allows the hosts to exchange some information to authenticate the establishment of additional subflows.[20]

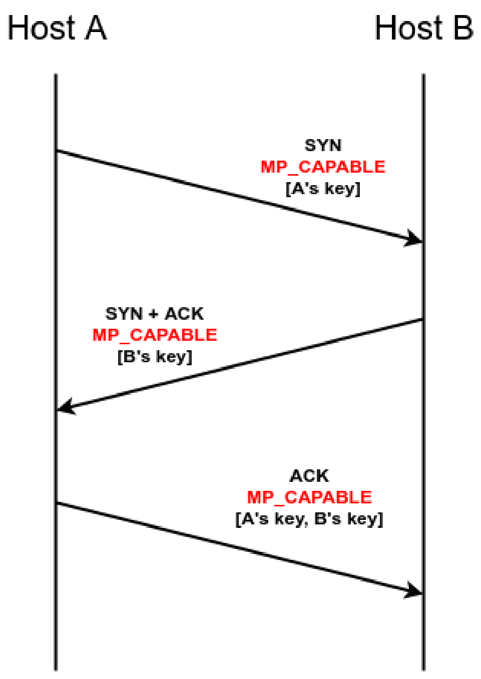


Figure 2.3: Initiating MPTCP connection MP_CAPABLE

2.2.3 Associating with an existing MPTCP Connection (MP_JOIN)

The exchange of keys in the MP_CAPABLE handshake provides material that can be used to authenticate the endpoints when new subflows will be set up. Additional subflows begin in the same way as initiating a normal TCP connection, but the SYN, SYN/ACK, and ACK packets also carry the MP_JOIN option.

Host A initiates a new subflow between one of its addresses and one of Host B's addresses. The token generated from the key is used to identify which MPTCP connection it is joining, and the HMAC is used for authentication. See figure 2.4 The Hash-based Message Authentication Code (HMAC) uses the keys exchanged in the MP_CAPABLE handshake, and the random numbers (nonces) exchanged in these MP_JOIN options. MP_JOIN also contains flags and an Address ID that can be used to refer to the source address without the sender needing to know if it has been changed by a NAT.

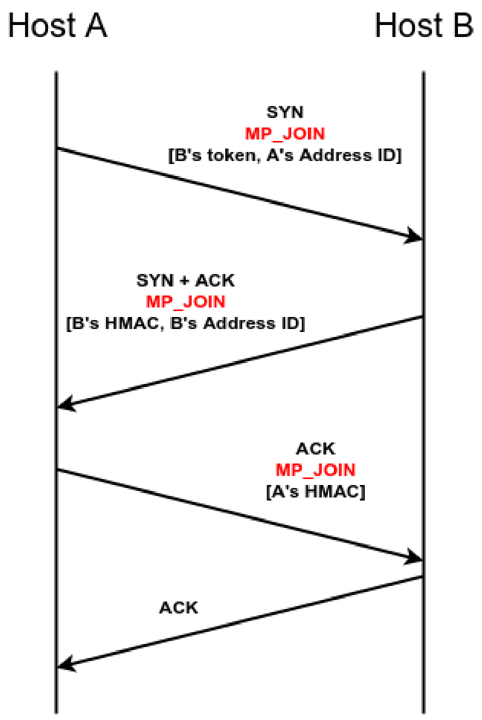


Figure 2.4: Associating with existing MPTCP connection MP_JOIN

2.2.4 Adding other available address (ADD_ADDR)

The set of IP addresses associated to a multihomed host may change during the lifetime of an MPTCP connection. MPTCP supports the addition and removal of addresses on a host both implicitly and explicitly. If Host A has established a subflow starting at address IP-A1 and wants to open a second subflow starting at address IP-A2, it simply initiates the establishment of the subflow as explained above. The remote host will then be implicitly informed about the new address.

In some circumstances, a host may want to advertise to the remote host the availability of an address without establishing a new subflow, for example, when a NAT prevents setup in one direction. In the example below, Host A informs Host B about its alternative IP address (IP-A2). See figure 2.5 Host B may later send an MP_JOIN to this new address. Due to the presence of middleboxes that may translate IP addresses, this option uses an address identifier to unambiguously identify an address on a host.

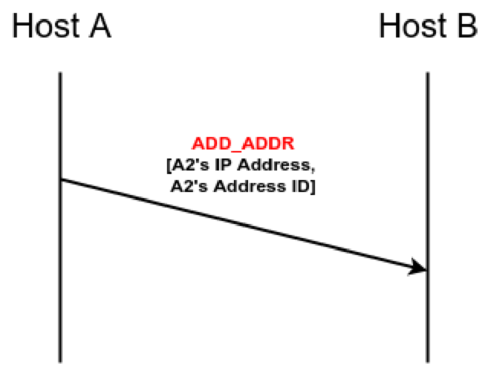


Figure 2.5: Adding other address ADD_ADDR

2.2.5 Removing invalid address (REMOVE_ADDR)

If, during the lifetime of an MPTCP connection, a previously announced address becomes invalid (e.g., if the interface disappears) the affected host should announce this so that the peer can remove subflows related to this address.
For security purposes, if a host receives a REMOVE_ADDR option, it must ensure the affected path are no longer in use before it instigates closure. The receipt of REMOVE_ADDR should first trigger the sending of a TCP keepalive on the path, and if a response is received the path should not be removed. Typical TCP validity tests on the subflow (e.g., ensuring sequence and ACK numbers are correct) MUST also be undertaken. An implementation can use indications of these test failures as part of intrusion detection or error logging.

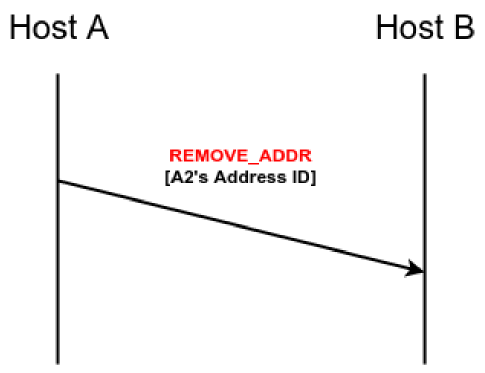


Figure 2.6: Removing invalid address REMOVE_ADDR

2.2.6 Data Transfer (DSS)

To ensure reliable delivery of data over subflows that may appear and disappear at any time, MPTCP uses a 64-bit data sequence number (DSN) to number all data sent over the MPTCP connection. Each subflow has its own 32bit sequence number space and an MPTCP option maps the subflow sequence space to the data sequence space. In this way, data can be retransmitted on different subflows (mapped to the same DSN) in the event of failure.

The "Data Sequence Signal" carries the "Data Sequence Mapping".[21] The data sequence mapping consists of the subflow sequence number, data sequence number, and length for which this mapping is valid. This option can also carry a connection-level acknowledgment (the "Data ACK") for the received DSN.

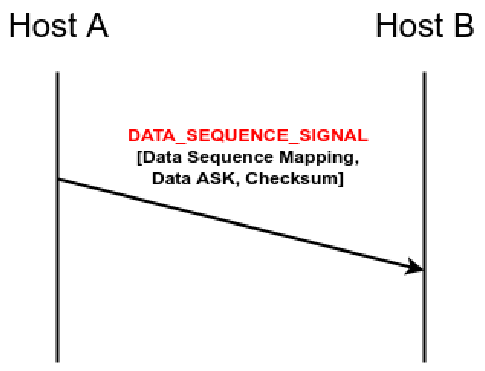


Figure 2.7: Data Sequence Signal DSS

2.2.7 Change in Path Priority (MP_PRIO)

Hosts can indicate at initial subflow setup whether they wish the subflow to be used as a regular or backup path a backup path only being used if there are no regular paths available. During a connection, Host A can request a change in the priority of a subflow through the MP_PRIO signal to Host B. Within a local MPTCP implementation, a host may use any local policy it wishes to decide how to share the traffic to be sent over the available paths.
The ability to make effective choices at the sender requires full knowledge of the path cost which is unlikely to be the case. It would be desirable for a receiver to be able to signal their own preferences for paths, since they will often be the multihomed party and may have to pay for metered incoming bandwidth.

2.2.8 Closing MPTCP connection

When Host A wants to inform Host B that it has no more data to send, it signals this "Data FIN" as part of the Data Sequence Signal. It has the same semantics and behavior as a regular TCP FIN, but at the connection level. Once all the data on the MPTCP connection has been successfully received, then this message is acknowledged at the connection level with a DATA_ACK. See figure 2.8 Regular TCP has the means of sending a reset (RST) signal to abruptly close a connection. With MPTCP, the RST only has the scope of the subflow and will only close the concerned subflow but not affect the remaining subflows. MPTCPś connection will stay alive at the data level, in order to permit break-before-make handover between subflows. It is therefore necessary to provide an MPTCP-level "reset" to allow the abrupt closure of the whole MPTCP connection, and this is the MP_FASTCLOSE option.

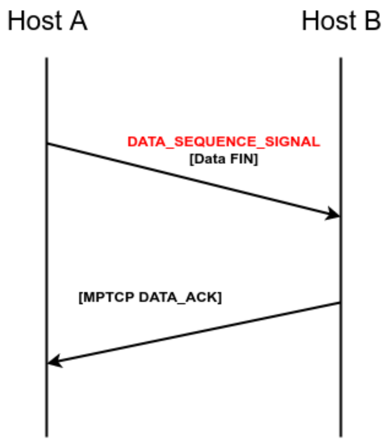


Figure 2.8: Closing MPTCP connection

2.3 Congestion control algorithms

2.3.1 Uncoupled Congestion Control

Uncoupled congestion control is the simplest form of congestion control for MPTCP. Each subflow is handled like an independent TCP connection, with its own instance of a TCP congestion control.[22] However, this solution is unsatisfactory, as it gives the multi-path flow an unfair share when the paths taken by its different subflows share a common bottleneck.

2.3.1.1 wVegas

wVegas (Weighted Vegas) is a delay-based congestion control based on TCP vegas. It is more sensitive to change because it does not have to wait for packet losses to react and shift between subflows to adapt to network congestion. For each subflow it calculate the difference between the expected sending rate and actual sending rate. During slow-start it double the congestion window every other RTT. If the difference is bigger than a threshold, it switches to congestion-avoidance. During congestion-avoidance it increase the window by one packet every RTT if the difference is small otherwise it reduces it by one packet. This means that overall the window grow slower than other algorithm and this avoids losses. However when the BDP is large, it switches too fast to the congestion avoiding phase and gives bad throughput because the window is too small.

Compared with loss-based algorithms, wVegas is more sensitive to changes of network congestion and thus achieves more timely traffic shifting and quicker convergence. The Congestion Equality Principle illustrates that a fair and efficient traffic shifting implies every flow strives to equalize the extent of congestion that it perceives on all its available paths.[23]

2.3.1.2 Cubic

Cubic is an implementation of TCP with an optimized congestion control algorithm for high bandwidth networks with high latency It is a less aggressive and more systematic derivative of BIC TCP, in which the window size is a cubic function of time since the last congestion event, with the inflection point set to the window size prior to the event. Because it is a cubic function, there are two components to window growth. The first is a concave portion where the window size quickly ramps up to the size before the last congestion event. Next is the convex growth where CUBIC probes for more bandwidth, slowly at first then very rapidly. CUBIC spends a lot of time at a plateau between the concave and convex growth region which allows the network to stabilize before CUBIC begins looking for more bandwidth. Another major difference between CUBIC and standard TCP flavors is that it does not rely on the receipt of ACKs to increase the window size. CUBIC's window size is dependent only on the last congestion event. The window growth function of CUBIC is governed by a cubic function in terms of the elapsed time since the last loss event. Furthermore, the real-time nature of the protocol keeps the window growth rate independent of RTT, which keeps the protocol TCP friendly under both short and long RTT paths.[24]

2.3.2 Coupled Congestion Control

The basic idea to solve the unfairness issue of uncoupled congestion control on shared bottlenecks is to couple the congestion windows of all subflows of an MPTCP connection with the resource pooling principle: detecting shared bottlenecks reliably is difficult, but it is just one part of a bigger issue. This bigger question is how much bandwidth a multi-path user should use in total, even if there is no shared bottleneck. The main idea is that by using a coupled congestion control method, the transport protocol can change the congestion window of each subflow and ensure bottleneck fairness and fairness in the broader, network sense. Several approaches to handle this issue are available, for example LIA (Linked Increases Algorithm), OLIA (Opportunistic LIA), Balia (Balanced LIA) and wVegas (Weighted Vegas). Coupled congestion control only applies to the increase phase of the congestion avoidance state, specifying how the congestion window inflates upon receiving an acknowledgement. Other phases are the same as in standard TCP.[25]

2.3.2.1 Linked Increase Algorithm (LIA)

The LIA calculation has three principle objectives :

1. Having an aggregate throughput greater than the one TCP can get using the best path
2. Not being more forceful than TCP, be reasonable for TCP.
3. Balancing the congestion over paths

As it accomplishes the two first goal but fails to accomplish the third one, a new algorithm has been developped, OLIA. In LIA, the slow start, fast retransmit, and fast recovery algorithms, as well as the multiplicative decrease of the congestion avoidance state are the same as in standard TCP. What changes is the increase phase of the congestion avoidance state, the window is increased by the minimum between the increase that would get normal TCP and the computed increase for the multipath subflow. The value of this parameter is chosen such that the aggregate throughput of the multipath flow is equal to the rate a TCP flow would get if it ran on the best path. alpha need to be calculated for each MPTCP flow. This guarantees the goal number two: not being more aggressive than TCP. Goal one is accomplished by computing an increase for the multipath subflow equal to the throughput a TCP flow would get if it ran on the best path.[26]

2.3.2.2 Opportunistic Linked-Increases Algorithm (OLIA)

OLIA is a window-based congestion-control algorithm that couples the increase of congestion windows and uses unmodifed TCP behavior in the case of a loss. Like LIA, the algorithm only modifies to the increase part of the congestion avoidance phase. It uses a set of best paths devided in two, a set with maximum windows and the rest of the paths. For the paths that have a small window, OLIA increase the window faster. For the path that have the maximum window, the increase is slower. The current congestion control algorithm of MPTCP, LIA, forces a tradeoff between optimal congestion balancing and responsiveness. OLIA's design departs from this tradeoff and provide these properties simultaneously. Hence, it solves the identified performance problems with LIA while retaining non-flappiness and responsiveness behavior of LIA. [27]

2.3.2.3 Balanced Linked Adaptation Algorithm (BALIA)

Balia is a generalized MPTCP algorithm that strikes a good balance between friendliness and responsiveness. The algorithm only applies to the AIMD part of the congestion avoidance phase. The other parts such as slow start, fast retransmit/recovery algorithms are the same as in TCP [RFC5681]. The minimum ssthresh is set to 1 MSS instead of 2 when more than 1 path is available. Responsiveness characterizes how fast the MPTCP algorithm reacts to changes in network conditions. the "Balanced linked adaptation", which is a window-based congestion control algorithm for MPTCP. The main design goal of Balia is to systematically tradeoff different properties such as TCP friendliness and responsiveness by developing structural understanding of MPTCP algorithms in a new design framework.

The window oscillation property characterizes how severely the window size fluctuates around the equilibrium point. It is an inherent property of AIMD-like algorithms.[28]

2.4 Unsuccessful MPTCP setup

Multipath is a developing technolgy that is standardized by Internet Engineering Task Force (IETF) under Request for Comments (RFC) 6824. Multipath TCP is used in many environment like Linux distributions, OpenWRT, Android, Planetlab, Raspberry Pi, etc. But choosing the right MPTCP kernel patch is not so easy, since there is not much documentation for designing and implementing Multipath TCP. I have spent most of my thesis time in trying different ways of MPTCP, some setup failed due to some reason and found the working configuration of MPTCP.

2.4.1 MPTCP in OpenWRT with VPN

I installed the MPTCP patched kernel on both devices that are involved in a Multipath TCP connection. Since my Ubuntu Host and Server have an ordinary TCP-connection, the router cannot use the MPTCP-protocol by default. To make it work, there is two possible solutions:

- Use a proxy on the router
- Use a VPN to an endpoint with faster network. In this way, we can use all uplinks for all traffic, even trafic to a non-MPTCP-server.

Issues arised while installing the MPTCP kernel patch in OpenWRT such as **recipe for target 'world' failed openwrt**. One of the main reason that this setup does not work is because of the uncompatible kernel of MPTCP and OpenWRT kernel. MPTCP kernel patch v3.14 is unsupported in latest OpenWRT Kernel Chaos Chalmer 15.05. Also it is unable to test the MPTCP in Ubuntu, since the MPTCP kernel created several issues while installing in latest Linux kernel 4.14.

2.4.2 OpenMPTCPRouter with VPS

Another avialable method to implement Multipath TCP in OpenWRT is to use OpenMPTCPRouter which is basically an OpenWRT router with MPTCP patched kernel version. This setup also includes usage of Virtual Private Sserver (VPS) and Shadowsocks proxy. The main reason for this setup failure, is using only one OpenMPTCPRouter with VPS did not initate the MPTCP. Also MPTCP cannot be enabled from this OpenMPTCPRouter from the command line and also the rules set in the ss-server(Shadowsocks server) and ss-redir(Shadowsocks client) does not start from rc.local.

2.5 MPTCP Kernel configuration

After several failures in choosing the right setup for MPTCP, finally I found a setup which supports MPTCP using Lede. The installation process of Lede with MPTCP kernel is explained below. The required software environment and build the LEDE images for the routers are available at [29]

First, select the device (router) where You want to install LEDE as the target. For testing, use x86 See figure 2.10 and VirtualBox VDI image as output. Then navigate to **Network –> Web Servers/Proxies** and select **shadowsocks-libev-nocrypto-ss-local and shadowsocks-libev-nocrypto-ss-redir**. See figure 2.11 Of course, you can select the regular versions from

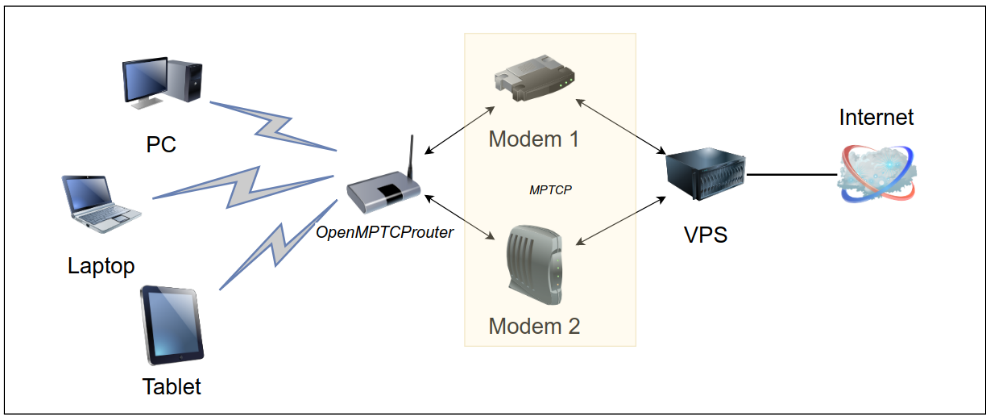
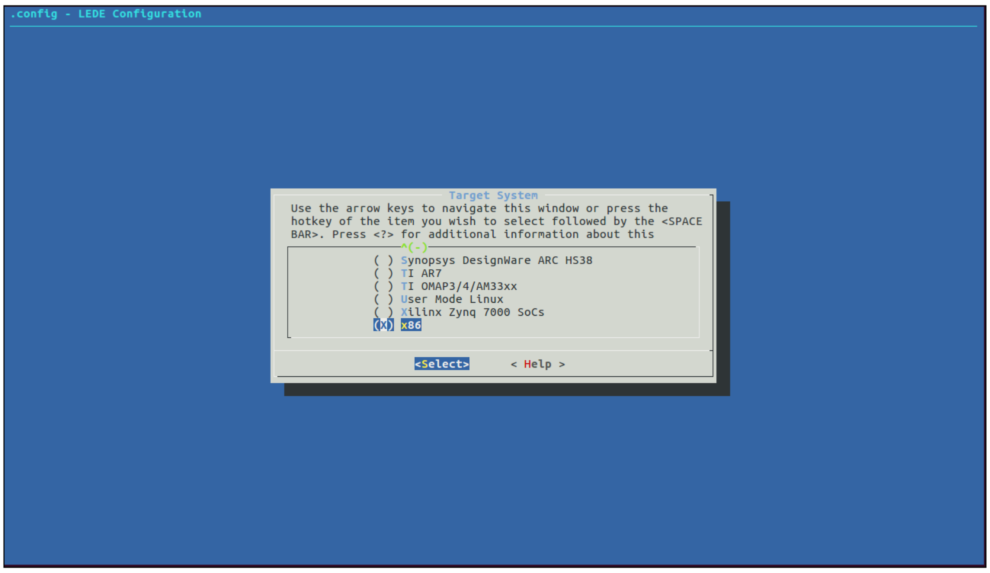


Figure 2.9: OpenMPTCPRouter with VPS setup

them without -nocrypto, but I recommend the nocrypto versions because of the better performance. Save the configuration!

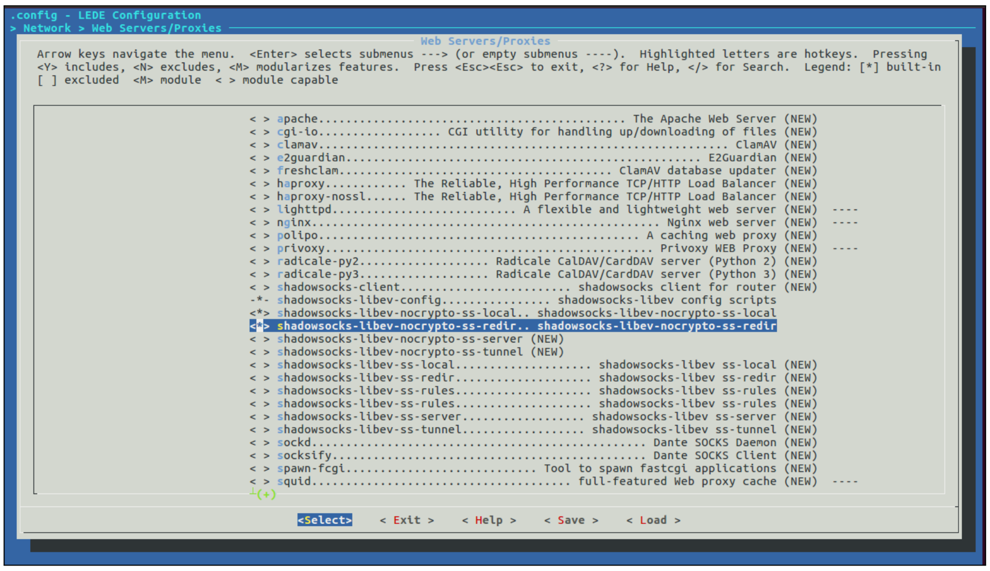


```
.config - LEDE Configuration
                        Target System
 Use the arrow keys to navigate this window or press the
 hotkey of the item you wish to select followed by the <SPACE
 BAR>. Press <?> for additional information about this
                  ^(-)
          ( ) Synopsys DesignWare ARC HS38
          ( ) TI AR7
          ( ) TI OMAP3/4/AM33xx
          ( ) User Mode Linux
          ( ) Xilinx Zynq 7000 SoCs
          (X) x86

              <Select>    < Help >
```

Figure 2.10: Lede configuration - Choosing target system

Then navigate into the **Networking support** > **Networking options** and enable MPTCP protocol. After that, go to the MPTCP: advanced path-manager submenu and select some path manager but at least one! 2.12 It is very important to select at least one, otherwise the multipath just not work! Full-mesh path manager should be enough for normal use. Save the configu-



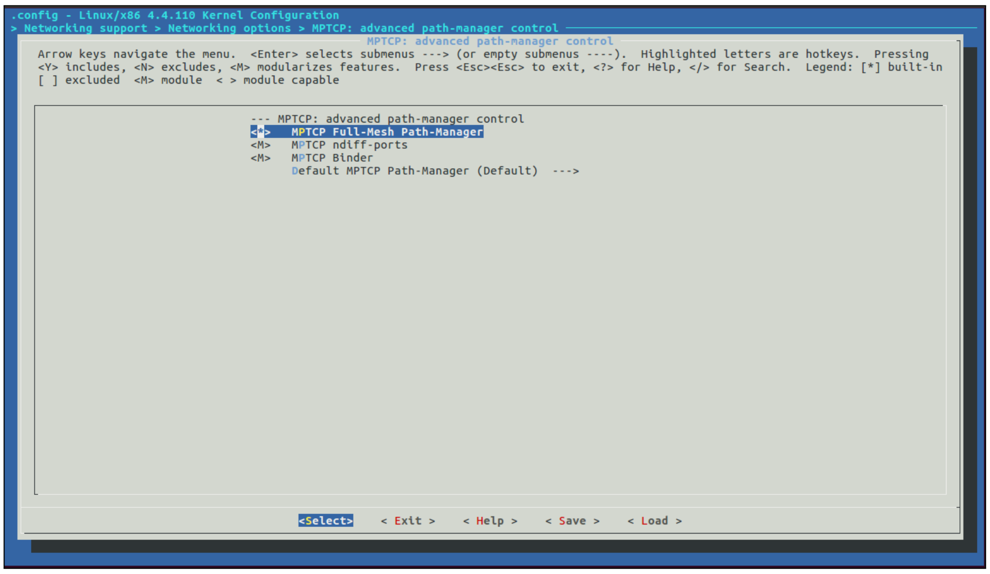
```
.config - LEDE Configuration
> Network > Web Servers/Proxies
                                  Web Servers/Proxies
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing
 <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
 [ ] excluded  <M> module  < > module capable

                        < > apache............................................ The Apache Web Server (NEW)
                        < > cgi-io................. CGI utility for handling up/downloading of files (NEW)
                        < > clamav.............................................................. ClamAV (NEW)
                        < > e2guardian.................................................... E2Guardian (NEW)
                        < > freshclam........................................ ClamAV database updater (NEW)
                        < > haproxy........... The Reliable, High Performance TCP/HTTP Load Balancer (NEW)
                        < > haproxy-nossl...... The Reliable, High Performance TCP/HTTP Load Balancer (NEW)
                        < > lighttpd........................... A flexible and lightweight web server (NEW)  ----
                        < > nginx.................................................. Nginx web server (NEW)  ----
                        < > polipo............................................. A caching web proxy (NEW)
                        < > privoxy.............................................. Privoxy WEB Proxy (NEW)  ----
                        < > radicale-py2.................. Radicale CalDAV/CardDAV server (Python 2) (NEW)
                        < > radicale-py3.................. Radicale CalDAV/CardDAV server (Python 3) (NEW)
                        < > shadowsocks-client......................... shadowsocks client for router (NEW)
                        -*- shadowsocks-libev-config............... shadowsocks-libev config scripts
                        <*> shadowsocks-libev-nocrypto-ss-local.. shadowsocks-libev-nocrypto-ss-local
                        <*> shadowsocks-libev-nocrypto-ss-redir.. shadowsocks-libev-nocrypto-ss-redir
                        < > shadowsocks-libev-nocrypto-ss-server (NEW)
                        < > shadowsocks-libev-nocrypto-ss-tunnel (NEW)
                        < > shadowsocks-libev-ss-local................... shadowsocks-libev ss-local (NEW)
                        < > shadowsocks-libev-ss-redir................... shadowsocks-libev ss-redir (NEW)
                        < > shadowsocks-libev-ss-rules................... shadowsocks-libev ss-rules (NEW)
                        < > shadowsocks-libev-ss-rules................... shadowsocks-libev ss-rules (NEW)
                        < > shadowsocks-libev-ss-server................. shadowsocks-libev ss-server (NEW)
                        < > shadowsocks-libev-ss-tunnel................. shadowsocks-libev ss-tunnel (NEW)
                        < > sockd............................................... Dante SOCKS Daemon (NEW)
                        < > socksify............................................ Dante SOCKS Client (NEW)
                        < > spawn-fcgi........................... Tool to spawn fastcgi applications (NEW)
                        < > squid.................................. full-featured Web proxy cache (NEW)  ----
                        └(+)

                   <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 2.11: Kernel configuration - Shadowsocks

ration! We are ready with the configuration so let's build the image!



```
.config - Linux/x86 4.4.110 Kernel Configuration
> Networking support > Networking options > MPTCP: advanced path-manager control
                              MPTCP: advanced path-manager control
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing
 <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
 [ ] excluded  <M> module  < > module capable

                        --- MPTCP: advanced path-manager control
                        <*>   MPTCP Full-Mesh Path-Manager
                        <M>   MPTCP ndiff-ports
                        <M>   MPTCP Binder
                              Default MPTCP Path-Manager (Default)  --->

                   <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 2.12: Kernel configuration - Choosing path manager

CHAPTER 3

Experimental setup

3.1 TCP setup

Measuring the performance of connection requires a client and a server. I use two Ubuntus to make this setup, where Ubuntu 1 acts as a client and Ubuntu 2 acts a server. Three routers are placed in between the Ubuntu 1 and Ubuntu 2, the same number of routers used in measuring the performance of Multipath TCP. This is becuase to make sure we use the consistent hop counts in both TCP and MPTCP setup. So, the overall resources are listed below. See figure 3.1

- 2 Ubuntu PCs as Host
- 3 Routers R1, R2 and R3 for traffic shapping

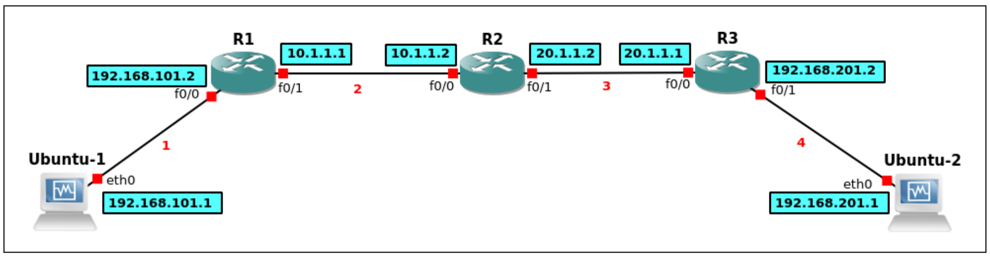


Figure 3.1: TCP experimental setup

3.1.1 Resources specification

Ubuntu PC

- Version - Ubuntu 14.04 LTS

- Memory - 3.6GB
- Processor - Intel Celeron(R) CPU G450 @ 2.50Ghz * 2
- Graphics - Intel Sandybridge Desktop
- OS type - 64-bit

3.2 Multipath TCP setup

Multipath TCP setup is constructed in such a way that 2 Ubuntus are connected with 2 OpenMPTCPRouter with MPTCP support. And 2 Routers R1 and R2 are kept in between 2 OpenMPTCPRouter to ensure the hop count consistency of TCP setup. SO, the overall Multipath TCP setup consists of the following resources. SEe figure 3.2

- 2 Ubuntu PCs as Host
- 2 OpenMPTCPRouter which is basically OpenWRT with MPTCP support
- 2 Routers R1 and R2 for traffic shapping

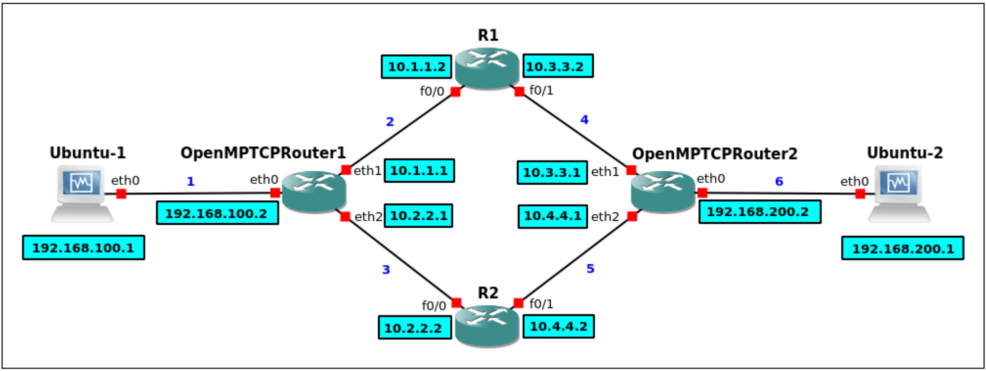


Figure 3.2: MPTCP experimental setup

3.3 OpenMPTCPRouter Setup

The OpenMPTCPRouter with MPTCP enabled patch in OpenWRT is available at [30]. I used this router because it has better features when compared to Lede MPTCP mentioned in Section 2.5. The notable features includes better Luci interface to analyze the network, security and using the multipath command in the terminal which did not work in other routers.

3.3.1 Network

The network configuration is placed under **/etc/config**. It consists of 1 lan interface which is connected to Ubuntu 1 and 2 wan interfaces which are connected to 2 Routers R1 and R2. The functionality of MPTCP such as congestion algorithm, mptcp path scheduler, mode of multipath and mptcp path manager is set under globals.

```
OpenMPTCPRouter 1

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'
        option multipath 'off'

config globals 'globals'
        option ula_prefix 'fd66:bd19:ca64::/48'
        option multipath 'enable'
        option mptcp_path_manager 'fullmesh'
        option mptcp_scheduler 'default'
        option congestion 'olia'
        option mptcp_checksum '1'
        option mptcp_syn_retries '4'

config interface 'lan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.100.2'
        option netmask '255.255.255.0'
        option multipath 'off'

config interface 'wan1'
        option ifname 'eth1'
        option proto 'static'
        option ipaddr '10.1.1.1'
        option netmask '255.255.255.0'

config interface 'wan2'
        option ifname 'eth2'
        option proto 'static'
        option ipaddr '10.2.2.1'
        option netmask '255.255.255.0'
```

The same network configuration is applied on OpenMPTCPRouter 2 as well which is basically a mirror like network of OpenMPTCPRouter 1.

```
OpenMPTCPRouter 2

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'
        option multipath 'off'

config globals 'globals'
        option ula_prefix 'fd66:bd19:ca64::/48'
        option multipath 'enable'
        option mptcp_path_manager 'fullmesh'
        option mptcp_scheduler 'default'
        option congestion 'olia'
        option mptcp_checksum '1'
        option mptcp_syn_retries '4'

config interface 'lan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.200.2'
        option netmask '255.255.255.0'
        option ip6assign '60'
        option multipath 'off'
        option ip4table 'lan'

config interface 'wan1'
        option ifname 'eth1'
        option proto 'static'
        option ipaddr '10.3.3.1'
        option netmask '255.255.255.0'

config interface 'wan2'
        option ifname 'eth2'
        option proto 'static'
        option ipaddr '10.4.4.1'
        option netmask '255.255.255.0'
```

3.3.2 Firewall

Some basic firewall rules needs to be set in both OpenMPTCPRouter1 and OpenMPTCPRouter 2. The firewall rules is shown below.

```
config zone
        option name                lan
        list   network             'lan'
        option input               ACCEPT
        option output              ACCEPT
        option forward             ACCEPT

config zone
        option name                wan
        option network             'wan1 wan2'
        option input               ACCEPT
        option output              ACCEPT
        option forward             REJECT
        option masq                0
        option mtu_fix             1

config forwarding
        option src                 lan
        option dest                wan

config forwarding
        option src                 wan
        option dest                lan
```

3.3.3 Routing

Next step is to add some basic IP rules and routes on both the OpenMPTCPRouters, so that it connected with the other routers in the middles namely R1 and R2.

```
OpenMPTCPRouter 1

ip rule add from 10.1.1.1 table 1
ip rule add from 10.2.2.1 table 2

ip route add 10.1.1.0/24 dev eth1 scope link table 1
ip route add default via 10.1.1.2 dev eth1 table 1

ip route add 10.2.2.0/24 dev eth2 scope link table 2
ip route add default via 10.2.2.2 dev eth2 table 2

route add -net 10.3.3.0 netmask 255.255.255.0 gw 10.1.1.2
route add -net 10.4.4.0 netmask 255.255.255.0 gw 10.2.2.2
```

```
OpenMPTCPRouter 2

ip rule add from 10.3.3.1 table 1
ip rule add from 10.4.4.1 table 2

ip route add 10.3.3.0/24 dev eth1 scope link table 1
ip route add default via 10.3.3.2 dev eth1 table 1

ip route add 10.4.4.0/24 dev eth2 scope link table 2
ip route add default via 10.4.4.2 dev eth2 table 2

route add -net 10.1.1.0 netmask 255.255.255.0 gw 10.3.3.2
route add -net 10.2.2.0 netmask 255.255.255.0 gw 10.4.4.2
```

3.3.4 Shadowsocks

3.3.4.1 Shadowsocks client

Shadowsocks client should be enabled while installing the kernel patch of MPTCP in OpenWRT. In my case, I use OpenMPTCPRouter 1 as the client for shadowsocks.

1. create /etc/ss_redir.json

```
{
    "server" : ["10.3.3.1", "10.4.4.1"],
    "server_port" : 8388,
    "local_address" : "0.0.0.0",
    "local_port" : 1080,
    "password" : "",
    "timeout" : 300,
    "method" : "none",
    "fast_open" : false,
}
```

2. Add the following in /etc/rc.local

```
ss-redir -c /etc/ss_redir.json
iptables -t nat -N SSREDIR
iptables -t nat -A PREROUTING -p tcp -j SSREDIR
iptables -t nat -A SSREDIR -d 127.0.0.0/8 -j RETURN
iptables -t nat -A SSREDIR -d 192.168.100.0/24 -j RETURN
iptables -t nat -A SSREDIR -p tcp -j REDIRECT --to-ports 1080
exit 0
```

3. Reboot the OpenMPTCPRouter 1

After rebooting the OpenMPTCPRouter 1, the shadowsocks client should start automatically from the rc.local after the boot. But in some cases it does not start automatically, so it needs to be started manually using the command **ash rc.local**.

3.3.4.2 Shadowsocks server

Shadowsocks server should also be enabled while installing the kernel patch of MPTCP in OpenWRT. In my case, I use OpenMPTCPRouter 2 as the server for shadowsocks.

1. Create /etc/ss_server.json

```
{
    "server" : "0.0.0.0",
    "server_port" : 8388,
    "local_address" : "0.0.0.0",
    "local_port" : 1080,
    "password" : "",
    "timeout" : 300,
    "method" : "none",
    "fast_open" : false,
}
```

2. Add the following in /etc/rc.local

```
ss-server -c /etc/ss_server.json
exit 0
```

3. Reboot the OpenMPTCPRouter 2

After rebooting the OpenMPTCPRouter 2, the shadowsocks server should start automatically from the rc.local after the boot. But in some cases it does not start automatically, so it needs to be started manually using the command **ash rc.local**.

3.3.5 Enable Multipath

MPTCP is running without any configuration. By default MPTCP is not enabled after the boot of the OpenMPTCPRouter.It needs to be started manually. If you want to use it with multiple interfaces on your device you have to configure these interfaces.

uci set network.<name>.multipath=<option>

Here you can choose one of the following options: Save your changes with:

uci commit /etc/init.d/network restart

```
uci set network.globals.multipath=enable
multipath eth1 on
multipath eth2 on
```

After the multipath TCP is enabled, the status can be seen in the Luci package. The green tick mark shows that multipath is enabled on both wan1 and wan2.

Chapter 4

Results

This chapter explains about the results obtained from the various experiments about the Multipath TCP. Experiments are conducted to analyze the performance of both TCP and Multipath TCP. Variety of experiments that includes data flow for different time, data flow with delays, different bandwidth limit, data losses, different congestion control algorithms, network failure and bulk data transfer. The results are showed in both tables and graphs for better understanding. Each experimental result is investigated and the reason which protocol is better is explained.
The units of the parameters mentioned in the results are seconds(sec) for time, milliseconds(ms) for delay, megabits/seconds(mbits/sec) for bandwidth and percentage(%) for data loss. All the experiments are evaluated for data flow of 60 seconds from Host 1 to Host 2 except for Time experiments.4.1. Default bandwidth of 100 mbits/sec on both the lines for TCP and MPTCP except bandwidth experiments.4.2 Time duration for bulk data transfer in both TCP and MPTCP is verified using Netcat tool and rest of the experiments are verified using iPerf.

4.1 TCP vs MPTCP - Basic speed experiments

Time experiments are conducted to investigate the attained bandwidth for continous data flow of different time range in both TCP and MPTCP. There is no restrictions such as delay, data loss and network failure in this experiments. Bandwidth is calculated for different time ranges from 5sec to 120sec. The bandwidth becomes stable for TCP after 40sec and the bandwidth is stable for MPTCP after 20sec. This proves that there is no need to verify for more than 120sec, and so I chose 60sec as default time of data flow for other experiments conducted in the later sections. As we can see from the Table 4.1, bandwidth of MPTCP is almost twice than the bandwidth of TCP. This is achieved because of the distribution of data in two TCP subflows which in turn uses bandwidth of both the lines in MPTCP. From the graph show in the Figure 4.1, it is clear that Multipath TCP has better bandwidth performance when compared to TCP.

Table 4.1: Performance of TCP vs MPTCP - Basic speed

Time	Bandwidth TCP	Bandwidth MPTCP	Ratio
(sec)	(mbits/sec)	(mbits/sec)	(TCP/MPTCP)
5	93.40	184.33	0.5067
10	93.67	185.33	0.5054
20	93.97	186.00	0.5052
40	94.07	186.00	0.5057
60	94.10	186.00	0.5059
100	94.10	186.00	0.5059
120	94.10	186.00	0.5059

Note: Performance comparison of uninterrupted dataflow for different time from 5 to 120(sec) and fixed bandwidth 100 (mbits/sec) on both lines for MPTCP.

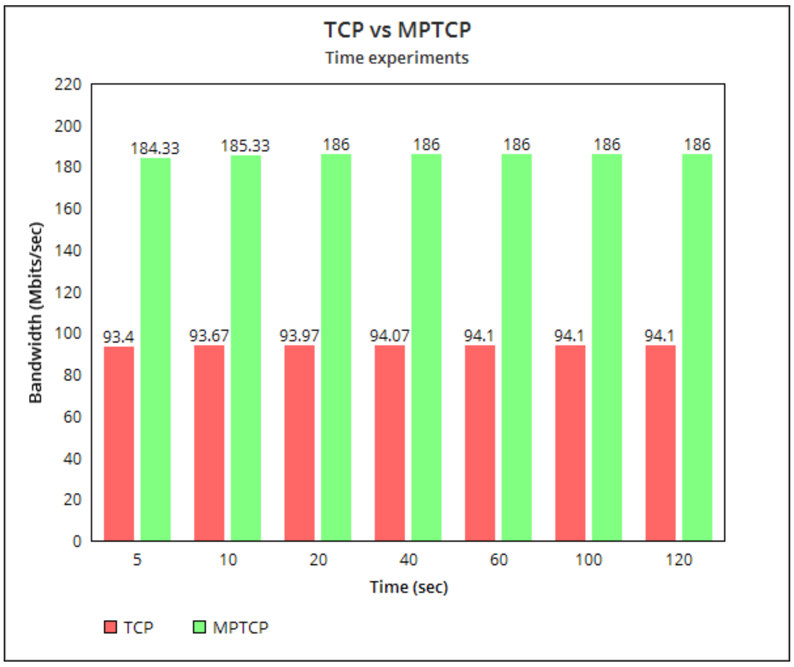


Figure 4.1: Performance of TCP vs MPTCP - Time experiments

4.2 TCP vs MPTCP - Bandwidth experiments

The purpose of these experiments is to identify the how the performance differs if different bandwidth limits are set in both TCP and MPTCP. Data flow is tested for bandwidth ranges from 8mbits/sec to 100mbits/sec. Limiting the bandwidth is attained using special tool named Traffic control (tc) in linux.1.6 The tc command to add the bandwidth limit on the outgoing traffic on the Router 1 is shown below.

```
tc qdisc add dev eth1 root tbf rate 32mbit burst 10kb limit 32mbit
```

Initially, same bandwidth limit is set on both lines for MPTCP and the results are shown in the Table 4.2. More than 95% of bandwidth is utilized in TCP for all ranges from 8 to 100mbits/sec. This is the same scenario in MPTCP as well but doubled than bandwidth of TCP. Therefore, we can conclude that different end devices with similar speed does not affect the performance of MPTCP. Multipath TCP uses almost 95% capacity on both end devices which is obviously better than TCP.

Table 4.2: Performance of TCP vs MPTCP - Bandwidth 1

Bandwidth Input	Bandwidth TCP	Bandwidth MPTCP	Ratio
(mbits/sec)	(mbits/sec)	(mbits/sec)	(TCP/MPTCP)
8	7.34	13.43	0.547
16	15.23	29.00	0.525
32	30.87	60.87	0.507
64	62.07	122.23	0.508
100	94.10	185.00	0.506

Note: Performance comparison of uninterrupted dataflow for different input bandwidth from 8 to 100(mbits/sec) and fixed time 60 (sec). However, same input bandwidth applied on 2 lines for MPTCP.

Another type of experiment is to observe how Multipath TCP behaves if both lines have different bandwidth. This main motivation for this experiments arises becuase of the different speed achieved from different interfaces that comes from the todayś end devices. For example , the lastest mobile phones supports 5G and ebility to use Wifi which in turn gives different speed. So, line 1 is limited with 32mbits/sec and the line 2 is tested with bandwidth ranges from 8 to 100mbits/sec. AS usual MPTCP distributes data in two lines and utilizes the maximum capacity of both lines. Figure 4.2 shows clearly that two lines with same bandwidth is better than different bandwidth. However, irrespective of different bandwidth used, MPTCP is always better than TCP.

Table 4.3: Performance of TCP vs MPTCP - Bandwidth 2

Bandwidth (mbits/sec)	Bandwidth TCP (mbits/sec)	Bandwidth MPTCP (mbits/sec)	Ratio (TCP/MPTCP)
32 and 8	7.34	37.37	0.197
32 and 16	15.20	45.13	0.337
32 and 32	30.87	60.37	0.511
32 and 64	62.17	91.30	0.681
32 and 100	94.10	123.53	0.752

Note: Performance comparison of uninterrupted dataflow for different input bandwidth (mbits/sec) and fixed time 60 (sec). However, different input bandwidth applied on 2 lines with fixed 32 (mbits/sec) on 1 line for MPTCP.

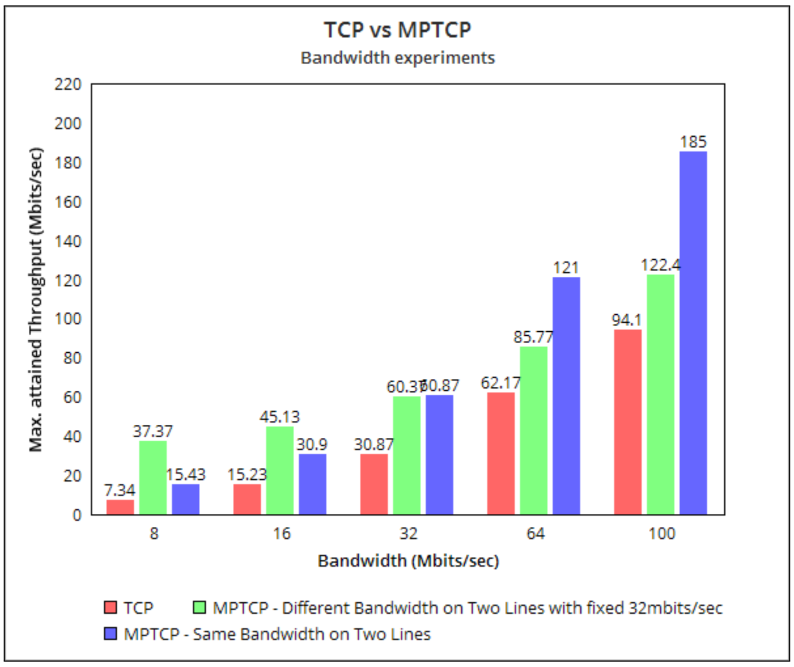


Figure 4.2: Performance of TCP vs MPTCP - Bandwidth experiments

4.3 TCP vs MPTCP - Delay experiments

This section describes about the experiments where data transfer is restricted to different delay in TCP and Multipath TCP. The delay of a network specifies how long it takes for a bit of data to travel across the network from one node to another. Again, traffic control is used to restrict the delay on one or both interfaces. The delay command is shown below.

tc qdisc add dev eth0 root netem delay 64ms

Initially, delay is set on only one line and next same delay is set on both the lines for MPTCP and various delay ranges from 2ms to 256ms. Bandwidth is almost identical for 2ms to 64ms in TCP and MPTCP. But when the delay goes higher, the performance of both TCP and MPTCP is getting poor. Delay on one line is optimal when compared to delay on two lines. Data flow with delay is compared as well in both protocols, and the results shows that Multipath TCP is optimal if delay is on one network rather than two networks. However, when compared to TCP, Multipath TCP achieves good performance irrespective of the delay.

Table 4.4: Performance of TCP vs MPTCP - Delay 1

Delay	Bandwidth TCP	Bandwidth MPTCP - Delay on Only 1 line	Bandwidth MPTCP - Same Delay on 2 lines	Ratio - 1 line	Ratio - 2 lines
(ms)	(mbits/sec)	(mbits/sec)	(mbits/sec)	(TCP/ MPTCP)	(TCP/ MPTCP)
2	94.10	185.67	184.33	0.51	0.51
8	94.03	183.97	179.13	0.51	0.52
32	93.83	179.87	161.87	0.52	0.58
64	93.50	166.23	124.53	0.55	0.75
128	56.63	109.43	81.50	0.52	0.72
256	28.23	82.07	36.17	0.34	0.78

Note: Performance comparison of delayed dataflow for different delay from 2 to 256 (ms), fixed bandwidth 100 (mbits/sec) and fixed time 60 (sec). However, same delay applied on 2 lines for MPTCP.

Next set of experiments is to analyze what happens if there is two different delays in two lines for MPTCP. Again, this experiment is necessary because of the distinct interfaces that comes in the end devices like Ethernet, Wifi,etc which consumes distinct delay. So, line1 is fixed with 64ms and line2 with ranges from 2ms to 256ms. The performance of MPTCP purely depends on the delay in both the lines such as 64ms and 256ms is obviously not better than 64ms and 2ms. MPTCP adds a scheduling layer over existing TCP connections and routes application packets to one of the subflows based on a decision parameter. Efficient scheduling decisions can improve the delay performance of MPTCP.

Table 4.5: Performance of TCP vs MPTCP - Delay 2

Delay	Bandwidth TCP	Bandwidth MPTCP	Ratio
(ms)	(mbits/sec)	(mbits/sec)	(TCP/MPTCP)
64 and 2	94.10	185.33	0.508
64 and 8	94.03	169.43	0.555
64 and 32	93.83	136.10	0.689
64 and 64	93.50	129.70	0.725
64 and 128	56.63	90.63	0.625
64 and 256	28.23	85.03	0.332

Note: Performance comparison of delayed dataflow for different delay from 2 to 256 (ms), fixed bandwidth 100 (mbits/sec) and fixed time 60 (sec). However, different delay applied on 2 lines with 64 (ms) fixed on 1 line for MPTCP.

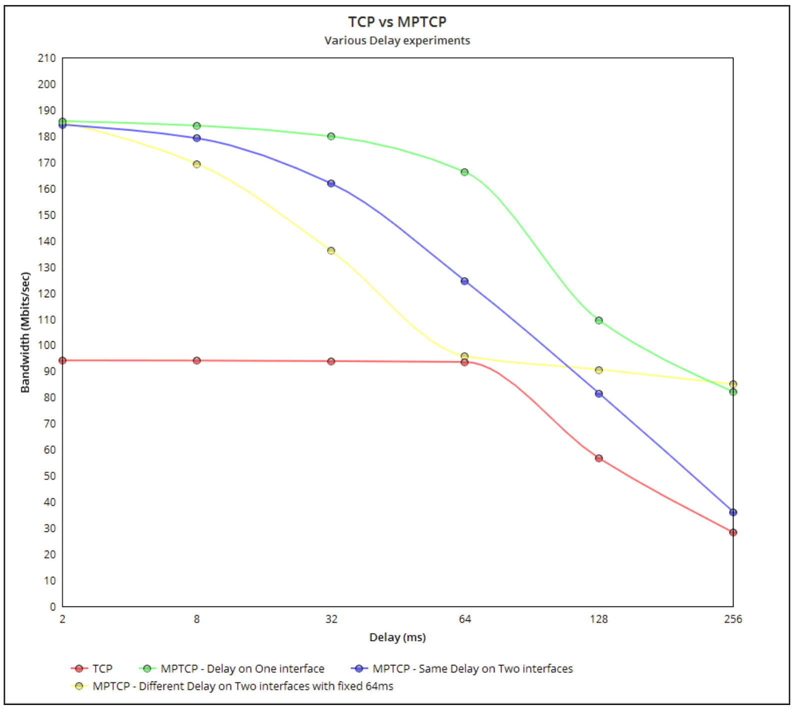


Figure 4.3: Performance of TCP vs MPTCP - Delay experiments

4.4 TCP vs MPTCP - Packet Loss experiments

Next set of experiments is to investigate about the performance of packet loss in both TCP and MPTCP. Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is either caused by errors in data transmissio or due t onetwork congestion and is closely related to quality of service. Traffic control is the again the best way to ensure the packet loss in the interfaces.

```
tc qdisc add dev eth0 root netem loss 5%
```

At first, on both TCP and MPTCP, packet loss is set on only one line ranges from 1.00% to 5.00%. Initial set of experiments is conducted with the basic congestion control algorithm LIA in MPTCP with TCP. Figure 4.4 clearly shows that even after packet loss, MPTCP is considerably optimal when compared to packet loss in TCP.

Table 4.6: Performance of TCP vs MPTCP - Packet Loss experiment

Loss	Bandwidth TCP	Bandwidth MPTCP	Ratio
(Percentage %)	(mbits/sec)	(mbits/sec)	(TCP/MPTCP)
1.00%	84.4	103.5	0.77
2.00%	66.5	101	0.73
3.00%	48.6	99.7	0.65
4.00%	36.8	97.95	0.55
5.00%	26.1	91.15	0.40

Note: Performance comparison of packet loss for different percentage from 1.00% to 5.00 %, fixed bandwidth 100 (mbits/sec) and fixed time 60 (sec). However, packet loss is applied on only 1 line for MPTCP.

4.5 MPTCP - Congestion control

Congestion control algorithm plays the fundamental role in MPTCP. Linking the subflows can reduce the traffic sent on the more congested path, and so balance load. Their main difference with classical TCP congestion control is that they need to react to congestion on the different paths without being unfair with single path TCP sources that could compete with them on one of the paths. Packet loss is tested in different congestion control algorithms like LIA, BALIA, OLIA and wVegas for MPTCP and Cubic for TCP. Initially, packet loss on only one line is checked range from 1.00% to 5.00%. From the Figure 4.5, it is without doubt that BALIA is the best congestion control algorithm of all. LIA and OLIA, suffer from either unfriendliness to Single Path TCP (SPTCP) or unresponsiveness to network changes under certain conditions. The tradeoff between friendliness and responsiveness is inevitable, but Balia judiciously balances this tradeoff.

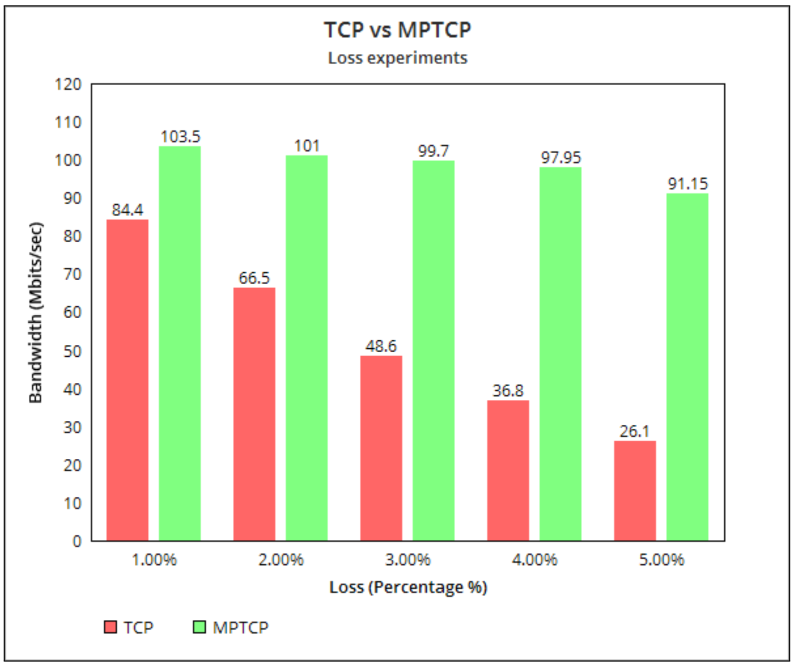


Figure 4.4: Performance of TCP vs MPTCP - Loss experiments

Table 4.7: Performance of TCP vs MPTCP - Packet Loss 1

Loss in 1 line	LIA	BALIA	OLIA	Cubic	wVegas
(Percent%)	(mbits/sec)	(mbits/sec)	(mbits/sec)	(mbits/sec)	(mbits/sec)
1.00%	105.00	**103.50**	104.50	105.00	104.50
2.00%	96.95	**101.00**	95.90	96.00	94.95
3.00%	89.60	**99.70**	90.95	88.25	88.80
4.00%	84.30	**97.95**	81.85	82.30	84.90
5.00%	79.15	**91.15**	76.85	75.50	80.40

Note: Performance comparison with data losses for different losses from 1.00 to 5.00 (%), fixed bandwidth 100 (mbits/sec) and fixed time 60 (sec). However, packet loss is applied on only 1 line.

Packet loss is applied only in one line in the previous experiment, whereas next experiment is to check the behaviour of packet losses in both the lines. However, the same amount of loss used in the above experiment **??** is splitted into two. So, theoritically the quantity of loss on 1 line is implemented in 2 lines now i.e from 1.00% on line1 to 0.5% and 0.5% on line1 and line2. The result of BALIA from Table 4.7 and BALIA from Table 4.8 shows that packet loss on both lines is not efficient when compared to packet loss on one line. Loss with 2.5% each on two lines of MPTCP is 38.65mbits/sec and loss with

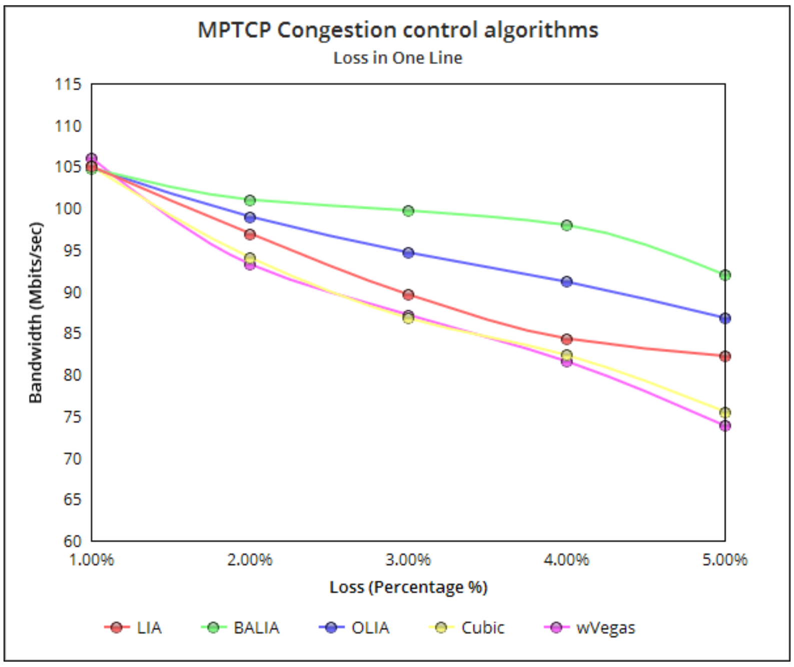


Figure 4.5: Performance of Congestion control algorithms in MPTCP

5% on TCP gives 26.1mbits/sec which is similar. Therefore, it is optimal to use TCP rather than MPTCP if loss is high on both the lines.

Table 4.8: Performance of TCP vs MPTCP - Packet Loss 2

Loss in 2 lines	LIA	BALIA	OLIA	Cubic	wVegas
(Percent%)	(mbits/sec)	(mbits/sec)	(mbits/sec)	(mbits/sec)	(mbits/sec)
0.5%, 0.5%	80.4	**89.5**	84.1	77.2	80.7
1.0%, 1.0%	64.15	**67.4**	64.35	60.45	59.05
1.5%, 1.5%	51.75	**54.1**	52.15	49.85	51.65
2.0%, 2.0%	44.8	**47.35**	44	45.2	43.25
2.5%, 2.5%	38.65	**40.25**	38.4	37.6	36.4

Note: Performance comparison with data losses for different losses from 0.5 to 2.5 (%), fixed bandwidth 100 (mbits/sec) and fixed time 60 (sec). However, same packet loss applied on 2 lines.

4.6 MPTCP - Bulk Data Transfer

The objective of this experiment is to compare how long does it takes to transfer huge data using TCP and MPTCP. Random data of size 1GB, 2GB

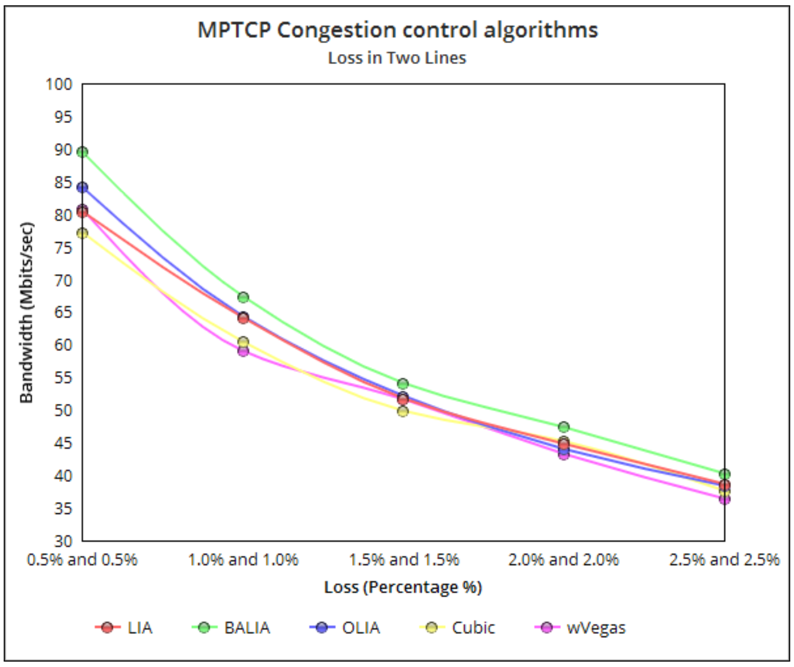


Figure 4.6: Performance of Congestion control algorithms in MPTCP 2

and 5GB is created using Data Duplicator (dd) and /dev/urandom which is a pseudo random number generator (PRNG). The huge data file is sent from Host 1 to Host 2 without any delay, packet loss and default bandwidth of 100 mbits/sec.

4.6.1 Generate Random file

One of the easy way to generate a random file is using Data Duplicator in Linux. The command to create random file is shown below where **if** is the input file name which is /dev/random and **of** is the output file name and **bs** is the block size. Using this feature, three huge random data files are created.

```
dd if=/dev/random of=data_1GB bs=1024 count=$((1024*1024))
```

4.6.2 Transfer Bulk Data using netcat

Once the huge data files are ready, it can be transfered using netcat tool which is efficient for calculating time of data transfer. On the server side (Ubuntu 2), netcat is started to listen at Port 2222. The file that is transferred can be neglected at the server using /dev/null, to avoid storing huge files which is unnecessary for this experiment, since we care about only the time. On the client side (Ubuntu 1), netcat started with IP address of server and same Port number 2222 along with the input file name.

```
Server:
nc -l 2222 > /dev/null

Client:
time nc 192.168.200.1 2222 < data_1GB
```

The above process is repeated 3 times for each file size and the average time is taken to ensure the consistency of the result. Table 4.9 illustrates the time taken to transfer bulk data using TCP and MPTCP. TCP takes takes 1 min 31 sec to transfer and MPTCP takes just 46 sec to transfer a 1GB file. The results proves that MPTCP always transfers huge data file almost twice the time faster than regular TCP.

Table 4.9: Performance of TCP vs MPTCP - Bulk data transfer

Data Size	TCP	MPTCP	Ratio MPTCP/TCP
(GB)	(min:sec.ms)	(min:sec.ms)	
1	01:31.290	00:46.289	0.5071
2	03:01.033	01:31.801	0.5071
5	07:25.665	03:45.996	0.5071

4.7 TCP vs MPTCP - Network Failure

The final experiment is related to comparing the performance of network failure using TCP and MPTCP. To achieve this, bandwidth test is started using iPerf and line1 is terminated purposely on 30th sec. The line is connected back again on 60th sec and tested till 180 sec without any interruption. By doing this, we infer that after termination of line1, TCP goes completely down to 0mbits/sec, whereas in MPTCP only line1 is down but line2 is still active. See figure 4.7

This proves that MPTCP is reliable over TCP. Traditional solutions cannot react quickly to the disappearance and reappearance of access links. Whenever a link disappears, sessions break and need to be reestablished, which can lead to data loss and the need for human intervention. Multipath TCP is able to react more quickly to access links disappearing and reappearing. And as long as at least one access link is up and running, a Multipath TCP enabled session will continue without interruption. Likewise, if an access link reappears, the bitrate goes up.

However, the recovery time of MPTCP is not optimal when compared to recovery time of TCP. After the line1 is connected back, the recovery time takes 9second for TCP, unlikely recoverly time takes 27second for MPTCP which is not optimal time.

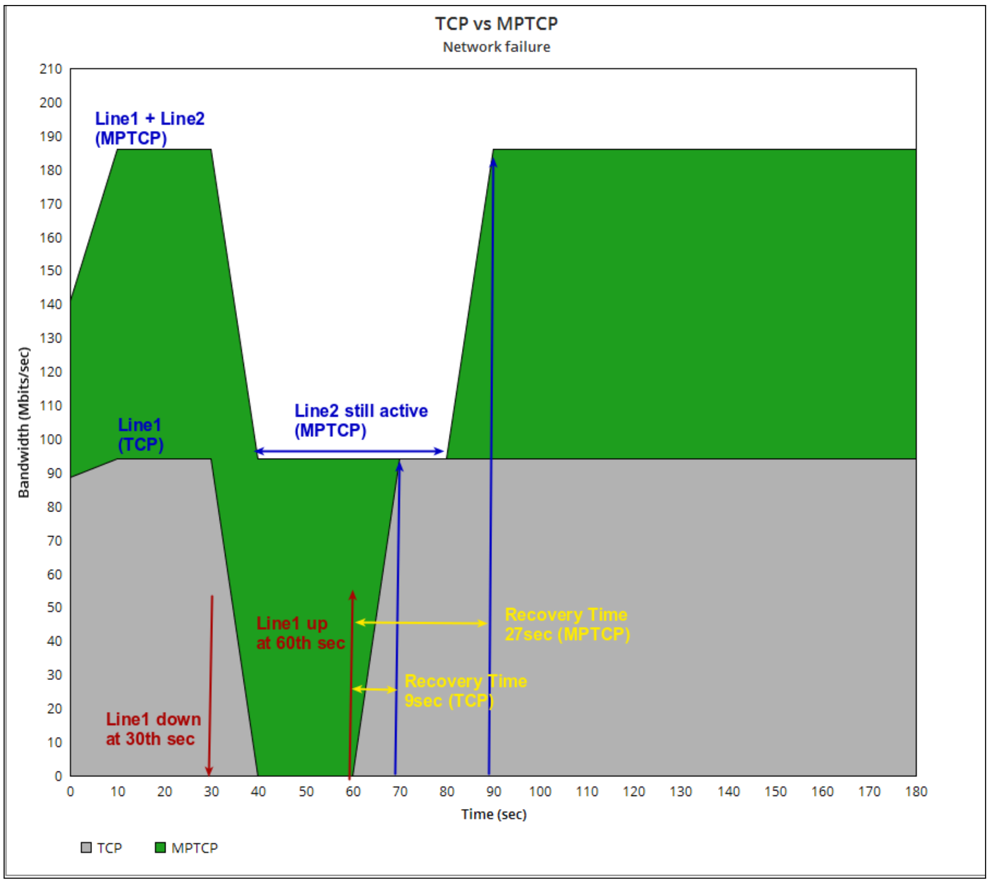


Figure 4.7: Performance of TCP vs MPTCP - Network failure

Chapter 5

Conclusion

In conclusion, results of the tests show that providing a robust and fast internet connection to companies and individuals using MPTCP coupled with Shodowsocks is possible. In this paper, I analyzed various methods to make the Multipath TCP work in OpenWRT router. I have spent most of the time of my thesis in choosing the appropriate setup to make Multipath TCP work. Different Multipath TCP setup includes OpenWRT with MPTCP kernel patch with Virtual Private Server (VPS), LEDE with MPTCP kernel patch and OpenWRT with MPTCP kernel patch with Shadowsocks proxy. Among the three setup, only the last one worked i.e OpenWRT with MPTCP kernel patch with Shadowsocks proxy. I explained in detail how to configure the setup of Multipath TCP in the experimental setup section.

After choosing the working Multipath setup, I carried out wide range of experiments to investigate the performance of regular TCP with Multipath TCP. Experiments consists of data flow for different time ranges, data with delay, packet losses, bulk data transfer and behaviour of network failure. All these experiments are conducted for both regular TCP and Multipath TCP.

Access aggregation is a viable option for service providers to boost the bandwidth. Typical access aggregation scenarios are the combination of DSL with LTE or the combination of LTE with Wi-Fi. Multipath TCP, as specified by the IETF, is ideal for access aggregation in the last mile, as it is able to boost bandwidth significantly, while simultaneously increasing reliability and ensuring seamless connectivity. Data flow with delay is compared as well in both protocols, and the results shows that Multipath TCP is optimal if delay is on one network rather than two networks.

Packet losses are common in the network due to inconsistent jitter, high latency, network congestion, etc. MPTCP provides improved congestion control algorithms like LIA, OLIA, BALIA and wVegas which gives optimal results when compared to regular TCP. Moreover, we made an extensive evaluation of the MPTCP protocol, and we verified that it can in fact balance network congestion, offering higher throughput and being more resilient to failures over the network. In several scenarios, MPTCP showed overall performance better than conventional TCP. However, the main drawback of MPTCP is that it is sensitive to packet loss on both the networks. MPTCP packet loss results shows that packet loss on both networks is almost similar to packet loss conventional TCP. It is better to use TCP rather than Multipath TCP if both network faces packet loss.

Finally all the tests in this paper were done in a closed lab environment in T344. An interesting thing to do would be to try that in the real world environment over the internet, to determine if additional problems than the ones represented in these experimental tests are encountered, potentially making the use of MPTCP a viable solution. Another interesting thing would be to try additional types of traffic, for example, performance of traffic generated by Voice over IP.

Bibliography

[1] *Transmission Control Protocol [online]*. Available from: https://tools.ietf.org/html/rfc793

[2] *TCP/IP Protocol Stack Description [online]*. Available from: https://docs.oracle.com/cd/E19504-01/802-5886/intro-14/index.html

[3] *A Three-Way Handshake for TCP Connection Establishment [online]*. Available from: www.mdpi.com/2076-3417/6/11/358/pdf

[4] *TCP connection termination [online]*. Available from: http://www.comptechdoc.org/independent/networking/terms/tcp-connection-termination.html

[5] *How DHCP Technology Works [online]*. Available from: https://technet.microsoft.com/pt-pt/library/cc780760(v=ws.10).aspx

[6] *OpenWRT Wireless Freedom [online]*. Available from: https://openwrt.org/

[7] *Porting and adaptation of OpenWRT for wireless routers [online]*. Available from: https://promwad.com/services/openwrt-for-wireless-routers

[8] *Reasons to use OpenWRT [online]*. Available from: https://openwrt.org/reasons_to_use_openwrt

[9] *Shadowsocks proxy [online]*. Available from: https://shadowsocks.org/en/index.html

[10] *iPerf 3 [online]*. Available from: https://iperf.fr

[11] *Traffic control tc [online]*. Available from: https://netbeez.net/blog/how-to-use-the-linux-traffic-control

[12] *Network delay [online]*. Available from: https://nsrc.org/workshops/2012/drukren-nsrc/raw-attachment/wiki/Agenda/types-of-delay.pdf

[13] *Netcat nc [online]*. Available from: http://netcat.sourceforge.net

[14] *Wireshark [online]*. Available from: https://www.wireshark.org

[15] *GNS3 [online]*. Available from: https://www.pluralsight.com/blog/it-ops/using-gns3-network-simulator

[16] *GNS3 [online]*. Available from: https://www.csd.uoc.gr/~hy435/material/GNS3-0.5-tutorial.pdf

[17] *What is TCP MULTIPATH and how does MULTIPATH in TCP work [online]*. Available from: https://www.slashroot.in/what-tcp-multipath-and-how-does-multipath-tcp-work

[18] *Bolstering the last mile with Multipath TCP [online]*. Available from: https://www.ericsson.com/en/ericsson-technology-review/archive/2016/bolstering-the-last-mile-with-multipath-tcp

[19] *Multipath TCP modes [online]*. Available from: https://wiki.openwrt.org/doc/uci/mptcp

[20] *Initiating MPTCP connection [online]*. Available from: https://tools.ietf.org/html/rfc6824

[21] *Multipath TCP Signalling [online]*. Available from: https://www.ietf.org/mail-archive/web/multipathtcp/current/pdf45oOxJBRQ9.pdf

[22] *Performance Comparison of Congestion Control Strategies for Multi-Path TCP in the NORNET Testbed [online]*. Available from: https://tools.ietf.org/html/draft-xu-mptcp-congestion-control-00

[23] *Delay-based Congestion Control for MPTCP [online]*. Available from: https://ieeexplore.ieee.org/document/7448667/

[24] *CUBIC: A New TCP-Friendly High-Speed TCP Variant [online]*. Available from: http://www4.ncsu.edu/~rhee/export/bitcp/cubic-paper.pdf

[25] *Coupled Congestion Control for MPTCP [online]*. Available from: https://www.ietf.org/proceedings/77/slides/mptcp-9.pdf

[26] *Linked Increase Algorithm [online]*. Available from: https://dial.uclouvain.be/memoire/ucl/en/object/thesis%3A2667/datastream/PDF_01/view

[27] *Opportunistic Linked Increase Algorithm [online]*. Available from: `https://tools.ietf.org/html/draft-khalili-mptcp-congestion-control-00`

[28] *Balanced Linked Adaptation Congestion Control Algorithm for MPTCP [online]*. Available from: `https://tools.ietf.org/html/draft-walid-mptcp-congestion-control-00`

[29] *Multipath Wi-Fi bridging with transparent MPTCP proxy on LEDE [online]*. Available from: `https://spyff.github.io/mptcp/2017/08/27/transparent-mptcp-proxy/`

[30] *OpenMPTCPRouter [online]*. Available from: `https://www.openmptcprouter.com/`

Acronyms

TCP	Transmission Control Protocol
MPTCP	Multipath Transmission Control Protocol
IP	Internet Protocol
SYN	Synchronize
ACK	Acknowledge
RST	Reset
DHCP	Dynamic Host Configuration Protocol
SSH	Secure Shell
ICMP	Internet Control Message Protocol
LIA	Linked Increase Algorithm
BALIA	Balanced Linked Adaptation Algorithm
OLIA	Opportunistic Linked-Increases Algorithm
PRNG	Pseudo Random Number Generator

Appendix A

All the configuration information about the resources used for setting up TCP and MPTCP is mentioned below.

TCP Setup

Ubuntu 1:

```
$ ifconfig eth1 192.168.101.1 netmask 255.255.255.0
$ route add default gw 192.168.101.2
```

Ubuntu 2:

```
$ ifconfig eth1 192.168.201.1 netmask 255.255.255.0
$ route add default gw 192.168.201.2

$ route add -net 10.1.1.0 netmask 255.255.255.0 gw 192.168.201.2
$ route add -net 20.1.1.0 netmask 255.255.255.0 gw 192.168.201.2
$ route add -net 192.168.101.0 netmask 255.255.255.0 gw 192.168.201.2
```

Router R1:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward

$ ifconfig eth0 192.168.101.2 netmask 255.255.255.0
$ route add default gw 192.168.101.1
$ ifconfig eth1 10.1.1.1 netmask 255.255.255.0
$ route add default gw 10.1.1.2
```

Router R2:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward

$ ifconfig eth0 10.1.1.2 netmask 255.255.255.0
$ route add default gw 10.1.1.1
$ ifconfig eth1 20.1.1.2 netmask 255.255.255.0
$ route add default gw 20.1.1.1

$ route add -net 192.168.101.0 netmask 255.255.255.0 gw 10.1.1.1
```

Router R3:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward

$ ifconfig eth0 20.1.1.1 netmask 255.255.255.0
$ route add default gw 20.1.1.2
$ ifconfig eth1 192.168.201.2 netmask 255.255.255.0
$ route add default gw 192.168.201.1

$ route add -net 10.1.1.0 netmask 255.255.255.0 gw 20.1.1.2
$ route add -net 192.168.101.0 netmask 255.255.255.0 gw 20.1.1.2
```

MPTCP Setup:

Ubuntu 1:

```
$ ifconfig eth1 192.168.100.1 netmask 255.255.255.0
$ route add default gw 192.168.100.2
```

Ubuntu 2:

```
$ ifconfig eth1 192.168.200.1 netmask 255.255.255.0
$ route add default gw 192.168.200.2
```

OpenMPTCPRouter 1:
Routing

```
$ ip rule add from 10.1.1.1 table 1
$ ip rule add from 10.2.2.1 table 2

$ ip route add 10.1.1.0/24 dev eth1 scope link table 1
$ ip route add default via 10.1.1.2 dev eth1 table 1
$ ip route add 10.2.2.0/24 dev eth2 scope link table 2
$ ip route add default via 10.2.2.2 dev eth2 table 2

$ route add -net 10.3.3.0 netmask 255.255.255.0 gw 10.1.1.2
$ route add -net 10.4.4.0 netmask 255.255.255.0 gw 10.2.2.2
```

ss-redir.json

```
{
"server" : ["10.3.3.1", "10.4.4.1"],
"server_port" : 8388,
"local_address" : "0.0.0.0",
"local_port" : 1080,
"password" : "",
"timeout" : 300,
"method" : "none",
"fast_open" : false,
}
```

rc.local

```
ss-redir -c /etc/ss_redir.json

iptables -t nat -N SSREDIR
iptables -t nat -A PREROUTING -p tcp -j SSREDIR
iptables -t nat -A SSREDIR -d 127.0.0.0/8 -j RETURN
iptables -t nat -A SSREDIR -d 192.168.100.0/24 -j RETURN
iptables -t nat -A SSREDIR -p tcp -j REDIRECT --to-ports 1080
exit 0
```

OpenMPTCPRouter 2:
Routing

```
$ ip rule add from 10.3.3.1 table 1
$ ip rule add from 10.4.4.1 table 2

$ ip route add 10.3.3.0/24 dev eth1 scope link table 1
$ ip route add default via 10.3.3.2 dev eth1 table 1

$ ip route add 10.4.4.0/24 dev eth2 scope link table 2
$ ip route add default via 10.4.4.2 dev eth2 table 2

$ route add -net 10.1.1.0 netmask 255.255.255.0 gw 10.3.3.2
$ route add -net 10.2.2.0 netmask 255.255.255.0 gw 10.4.4.2
```

ss-server.json

```
{
"server"        : "0.0.0.0",
"server_port"   : 8388,
"local_address" : "0.0.0.0",
"local_port"    : 1080,
"password"      : "",
"timeout"       : 300,
"method"        : "none",
"fast_open"     : false,
}
```

rc.local

```
ss-server -c /etc/ss_server.json
exit 0
```

Router R1:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward

$ ifconfig eth0 10.1.1.2 netmask 255.255.255.0
$ ifconfig eth1 10.3.3.2 netmask 255.255.255.0
```

Router R2:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward

$ ifconfig eth0 10.2.2.2 netmask 255.255.255.0
$ ifconfig eth1 10.4.4.2 netmask 255.255.255.0
```

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ....................................... the directory of source codes
  MPTCP setup ............ configuration information for MPTCP setup
  TCP setup .................. configuration information for TCP setup
  Results ........... the directory of results obtained from experiments
  thesis ............... the directory of LaTeX source codes of the thesis
    Pictures ...................... the directory of all thesis pictures
thesis.pdf ............................ the thesis text in PDF format
```