NETWORK PROGRAMMING PROJECT

Group Name: NP03

Group Members:

Name	ID No.
Sukrit	2018A7PS0205H
Karthik Shetty	2018A7PS0141H
Thakkar Preet Girish	2018A7PS0313H
Koustubh Sharma	2018A7PS0114H

Title of the Report: Efficient Video Streaming with multiple connection support using MPTCP/MPQUIC protocols in linux kernel

Selected Project: Project 2

Initial Setup Design:

❏ MPTCP

Installation

➢ For Mininet:

- Instructions provided at Download/Get Started With Mininet were used to set up Mininet locally.

➢ For MPTCP:

- We used the apt-repository mentioned at multipath-tcp.org website to download the precompiled linux kernel supporting MPTCP.

```
$ sudo apt-key adv --keyserver hkps://keyserver.ubuntu.com:443
--recv-keys 379CE192D401AB61

$ sudo sh -c "echo 'deb
https://dl.bintray.com/multipath-tcp/mptcp_deb stable main' >
/etc/apt/sources.list.d/mptcp.list"

$ sudo apt-get update

$ sudo apt-get install linux-mptcp
```

- Ubuntu is then booted from the same kernel.
- The installation is then verified using curl to http://multipath-tcp.org.

```
$ curl http://multipath-tcp.org
```

❏ MPQuic

Installation

- For Mininet:

    - Instructions provided at Download/Get Started With Mininet were used to set up Mininet locally.

- For MPQuic:

Since the QUIC and MPQUIC implementations use Go, setting it up is essential.

```
$ sudo apt-get install golang-go
```

The following steps were taken to setup QUIC followed by MPQUIC. These steps are:

- Use the go binary to clone the quic-go GitHub Repository. Change directory to the cloned folder.

```
$ go get github.com/lucas-clemente/quic-go
$ cd go/src/github.com/lucas-clemente/quic-go
```

- Since MPQUIC is based off of the quic-go, we need to add a git remote pointing to the MPQUIC repository and fetch those changes and switch to the branch containing MPQUIC.

```
$ git remote add mp-quic https://github.com/qdeconinck/mp-quic.git
$ git fetch mp-quic
$ git checkout conext17
```

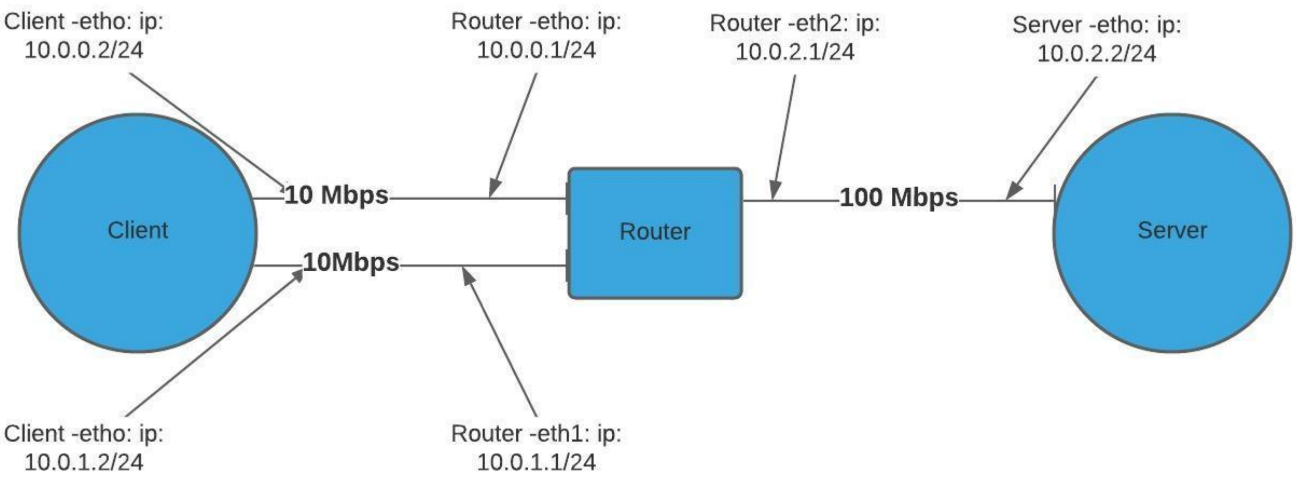
- We then download all dependencies

```
$ go get -t -u './...'
```

- We then fetch the minitopo repository

```
$ mkdir ~/git && /git
$ git clone https://github.com/qdeconinck/minitopo.git
```

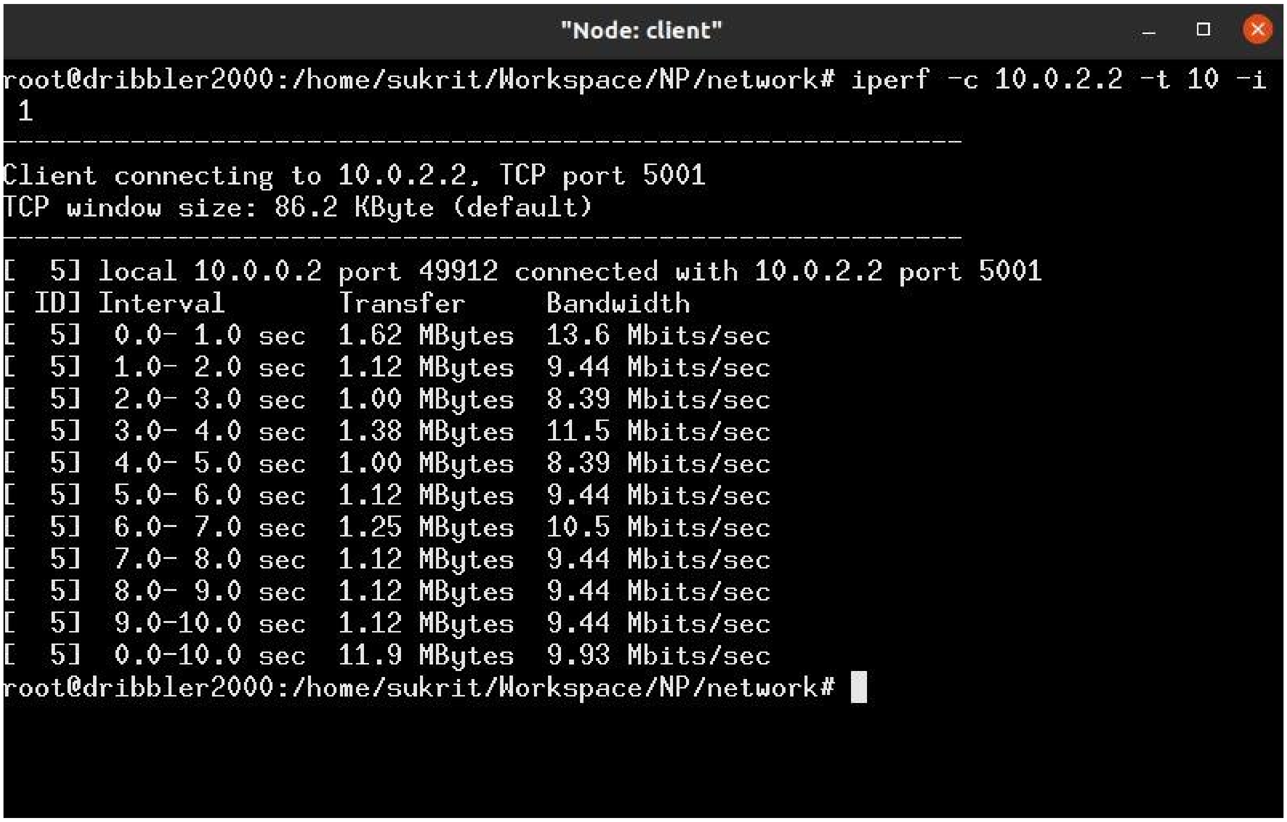
Virtual Network Setup



- A python script is written using the mininet library to set up a virtual network topology to experiment with MPTCP/MPQUIC protocol.

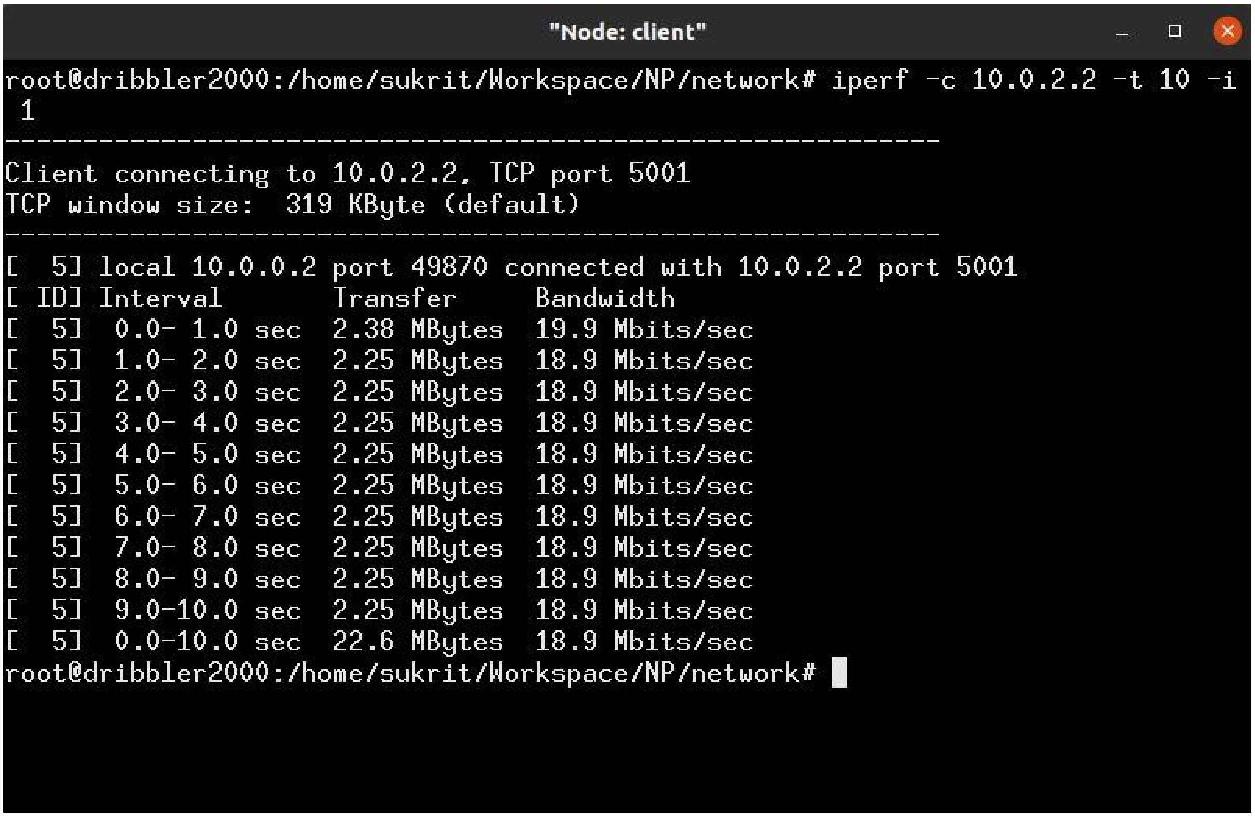
- Topology consists of two end hosts - client and server and a router connecting them.

- To simulate multiple connections, 2 connections are made between the client and the router and a single connection is made between the server and the router. Bandwidth of client connections is 10 mbps and server connections is 100mbps for real life simulation.



```
root@dribbler2000:/home/sukrit/Workspace/NP/network# iperf -c 10.0.2.2 -t 10 -i
 1
------------------------------------------------------------
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 86.2 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 49912 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 1.0 sec  1.62 MBytes  13.6 Mbits/sec
[  5]  1.0- 2.0 sec  1.12 MBytes  9.44 Mbits/sec
[  5]  2.0- 3.0 sec  1.00 MBytes  8.39 Mbits/sec
[  5]  3.0- 4.0 sec  1.38 MBytes  11.5 Mbits/sec
[  5]  4.0- 5.0 sec  1.00 MBytes  8.39 Mbits/sec
[  5]  5.0- 6.0 sec  1.12 MBytes  9.44 Mbits/sec
[  5]  6.0- 7.0 sec  1.25 MBytes  10.5 Mbits/sec
[  5]  7.0- 8.0 sec  1.12 MBytes  9.44 Mbits/sec
[  5]  8.0- 9.0 sec  1.12 MBytes  9.44 Mbits/sec
[  5]  9.0-10.0 sec  1.12 MBytes  9.44 Mbits/sec
[  5]  0.0-10.0 sec  11.9 MBytes  9.93 Mbits/sec
root@dribbler2000:/home/sukrit/Workspace/NP/network# 
```

Fig. Bandwidth results with MPTCP disabled



```
root@dribbler2000:/home/sukrit/Workspace/NP/network# iperf -c 10.0.2.2 -t 10 -i
 1
------------------------------------------------------------
Client connecting to 10.0.2.2, TCP port 5001
TCP window size:  319 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 49870 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 1.0 sec  2.38 MBytes  19.9 Mbits/sec
[  5]  1.0- 2.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  2.0- 3.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  3.0- 4.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  4.0- 5.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  5.0- 6.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  6.0- 7.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  7.0- 8.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  8.0- 9.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  9.0-10.0 sec  2.25 MBytes  18.9 Mbits/sec
[  5]  0.0-10.0 sec  22.6 MBytes  18.9 Mbits/sec
root@dribbler2000:/home/sukrit/Workspace/NP/network# 
```

Fig. Bandwidth results with MPTCP enabled

Routing Table Configuration

Client:

Destination	Gateway	Subnet Mask	Interface
0.0.0.0	10.0.0.1	0.0.0.0	client-eth0
10.0.0.0	0.0.0.0	255.255.255.0	client-eth0
10.0.1.0	0.0.0.0	255.255.255.0	client-eth1

Server:

Destination	Gateway	Subnet Mask	Interface
0.0.0.0	10.0.2.1	0.0.0.0	server-eth0
10.0.2.0	0.0.0.0	255.255.255.0	server-eth0

Router:

Destination	Gateway	Subnet Mask	Interface
10.0.0.0	0.0.0.0	255.255.255.0	router-eth0
10.0.1.0	0.0.0.0	255.255.255.0	router-eth1
10.0.2.0	0.0.0.0	255.255.255.0	router-eth2

Comparing MPTCP/MPQUIC with normal TCP/QUIC connection

- Using the mininet's xterm command, we compare the bandwidths of the connections with MPTCP/MPQUIC enabled and disabled both. Hence, **xterm client server** is run.

- The server is run using the **iperf -s -i 1** command, that sets it as a server and sends the data at a gap of 1 seconds.

- The client is run using the **iperf -c <server_ip> -t 10 -i 1** command, that sends the requests to the specified server ip address and for a time of 10 seconds.

.

Implementation:

Experimentation with Virtual Network Topology

- After setting up a virtual network topology, we began experimenting with data being shared within this network to experiment with MPTCP/MPQUIC protocol.
- Using mininet's xterm command, it simulates a command line interface for each node. In our case, we maintained two nodes - server and client.
- To enable MPTCP, we need the socket level option MPTCP_ENABLED/ specified to 1.
- To enable MPQUIC, we need to set the multipath parameter of the quic-go library.
- As we know, we use level argument to manipulate the socket-level option. In this case MPTCP will be enabled if the application has set the socket-option MPTCP_ENABLED (value 42) to 1. MPTCP_ENABLED is part of SOL_TCP level.
- We used setsockopt(socket.SOL_TCP,MPTCP_ENABLED, 1) to enable MPTCP.
- We tested a few python scripts to exchange data between the two nodes - server and client with MPTCP enabled.
- We also tested python scripts to calculate bandwidth with which data is transferred between the nodes with MPTCP enabled and disabled.

Integration of Video Streaming Algorithm

A MPEG-2 video can be seen as a playlist of TS (transport stream) files. A M3U8 file provides metadata about the same.

Hence, we propose the video streaming implementation in the following manner:

- The TS files are stored on the server side inside a directory.
- The client sends a request to the server for video streaming.
- The server starts sending the TS files to the client, through MPTCP/MPQUIC protocols.
- On receiving the TS files, the client merges them into a single videoplayback file using the m3u8 file descriptor.

Results:

Result of Bandwidth Calculation Script

- We wrote a python script for the client and server to print the throughput and ran it in the respective CLI of the nodes and compared results with MPTCP/MPQUIC enabled and disabled.
- The result was in line with the network topology we have in place for the setup.
- It produced a bandwidth close to 20 Mb/s with MPTCP/MPQUIC enabled and around 10Mb/s with MPTCP/MPQUIC disabled.
- We tested different scenarios using netem tools to test different network scenarios of adding delays, packet loss, packet duplication and packet corruption.
- We successfully transferred data between the client and server node with MPTCP enabled.

Innovation in Video Streaming Algorithm:

- Currently our video streaming application works on transfer of a m3u8 video cluster in the form of chunks of TS files. We can improve upon this idea to implement a **live video streaming application through webcam.**
- The basic idea is that the server streams a video through a webcam in real time and then sends the data over MPTCP/MPQUIC protocols to the client. This video stream is buffered and displayed on the client side.
- For video capturing, python libraries like **OpenCV** can be used on the server side and then the frames are sent over a **socket connection** or **quic** library to the client.
- Further innovations in the quality of videos can be made by providing sufficient bandwidth for a video stream with specified parameters can give better video qualities in different network conditions.
- We can implement this parameter based video streaming as an innovation with the help of **Dynamic Adaptive Streaming (DASH)** over HTTP which is one of the most popular implementations of **HTTP Adaptive Streaming (HAS)** standard to enhance the quality of our existing video streaming application.

References:

- draft-pantos-http-live-streaming-23 - HTTP Live Streaming
- https://github.com/qdeconinck/mp-quic
- https://www.multipath-tcp.org/
- https://github.com/multipath-tcp/mptcp
- https://tools.ietf.org/html/draft-samar-mptcp-socketapi-03
- lucas-clemente/quic-go: A QUIC implementation in pure go
- MultiPath TCP - Linux Kernel implementation : Main - Home Page browse
- Multipath QUIC
- The simulation study on the multipath adaptive video transmission