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 the 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 set up 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 on the quic-go, we need to add a git remote pointing to the MPQUIC repository, 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
```

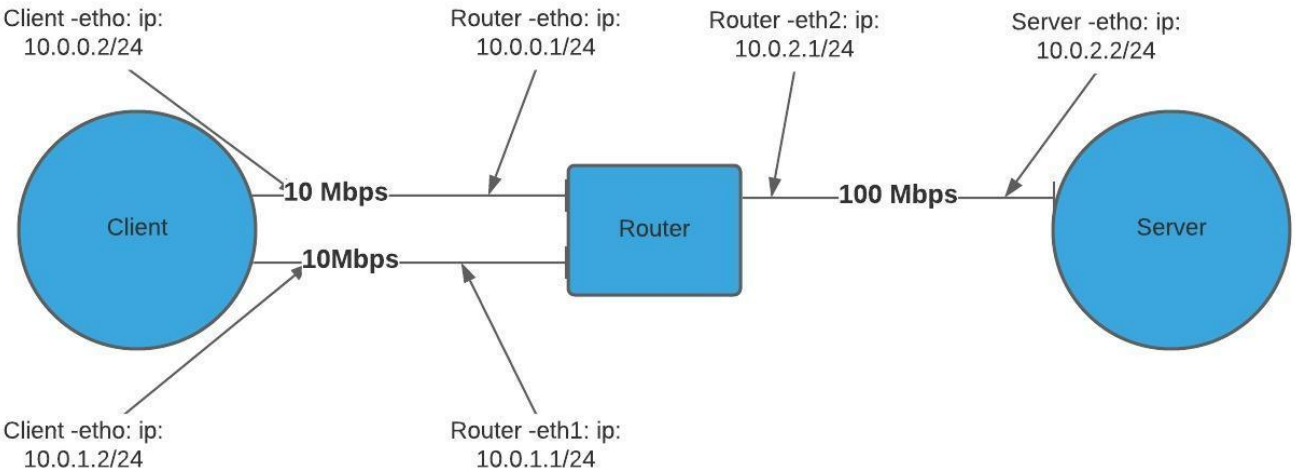
- We then download all dependencies

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

- We then fetch the mini topo 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.

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

- Two connections are made between the client and the router to simulate multiple connections, and a single connection is made between the server and the router. The bandwidth of client connections is 10 Mbps, and server connections are 100mbps for real-life simulation.

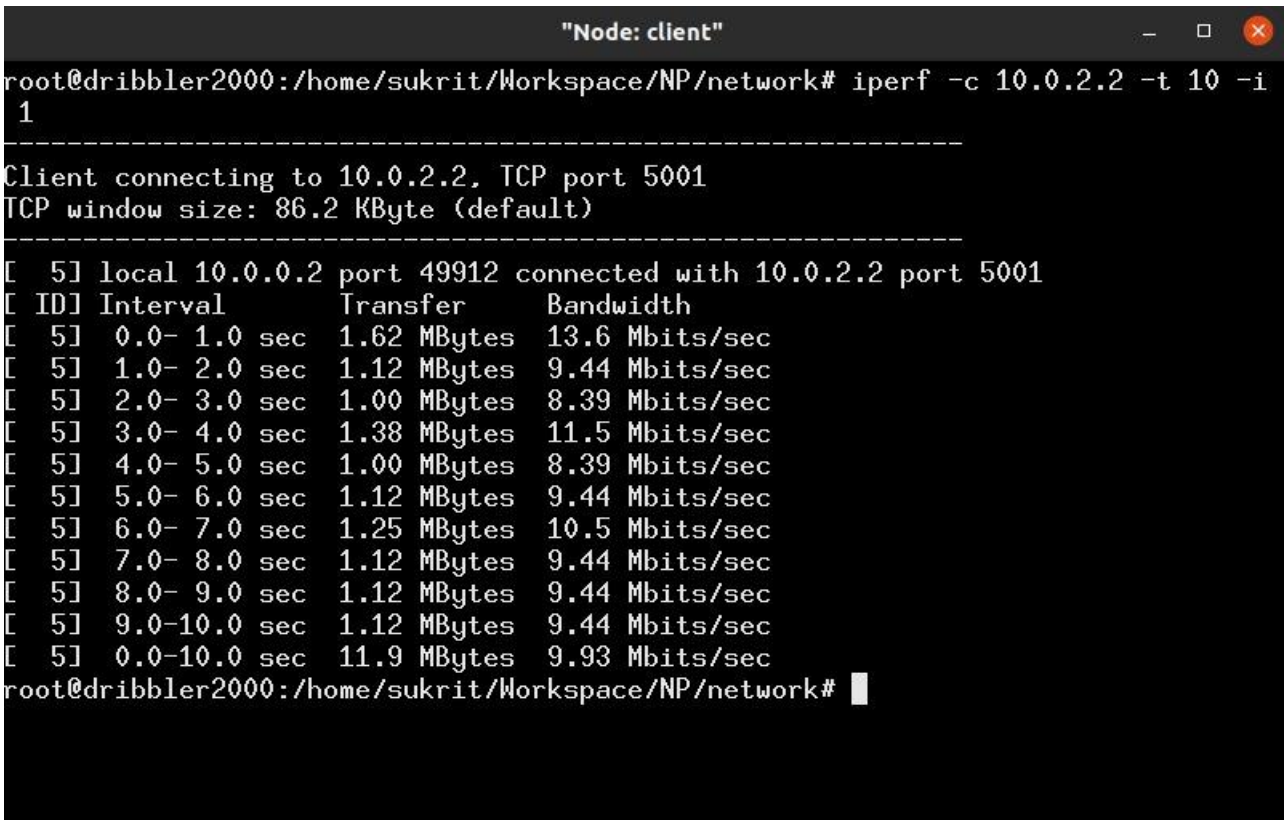


Fig. Bandwidth results with MPTCP disabled

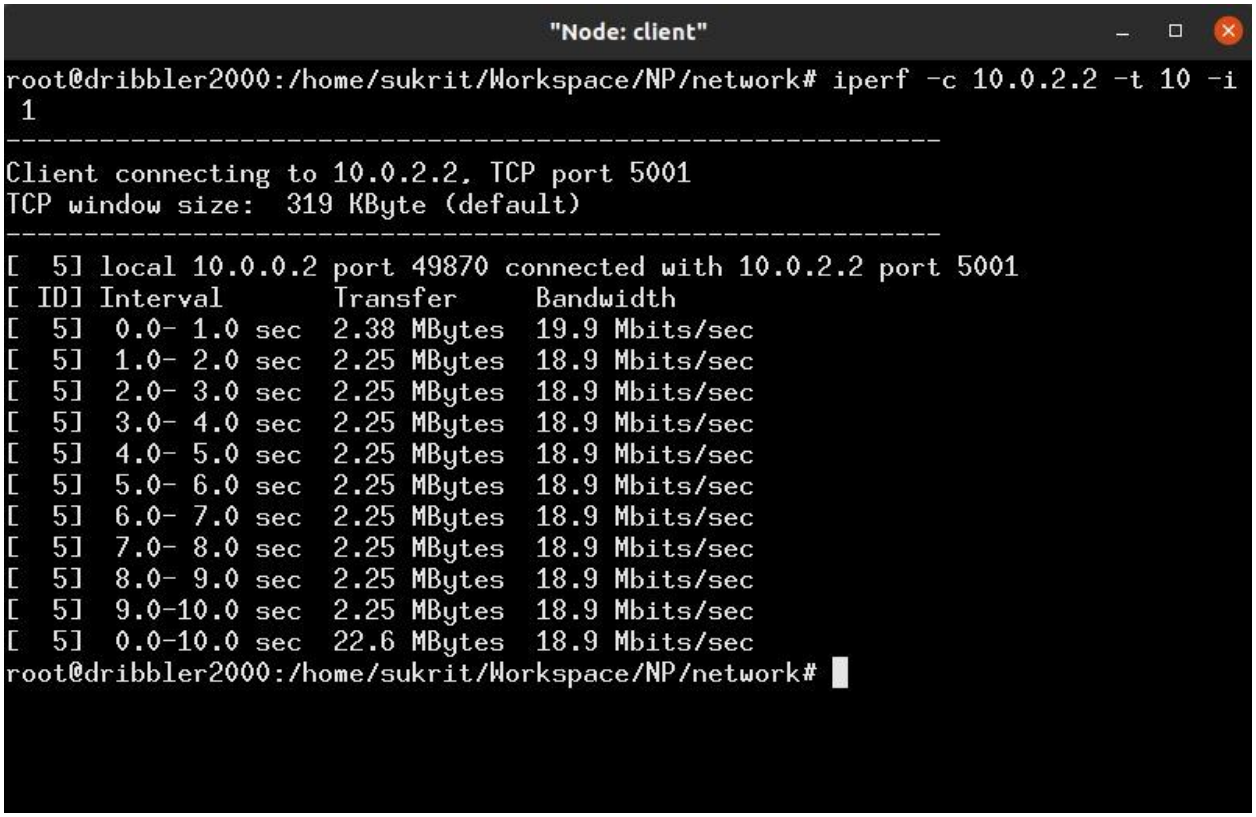


Fig. Bandwidth results with MPTCP enabled

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, which sets it as a server and sends the data at a gap of 1 second.

- The client is run using the **iperf -c <server_ip> -t 10 -i 1** command, which sends the requests to the specified server IP address for 10 seconds.

.

Implementation:

Experimentation with Virtual Network Topology

- After setting up a virtual network topology, we experimented with data 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 the 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. An 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.
- The client merges them into a single video playback file using the m3u8 file descriptor on receiving the TS files.

Results:

Result of Bandwidth Calculation Script

- We wrote a python script for the client and server to print the throughput, 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 various 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:

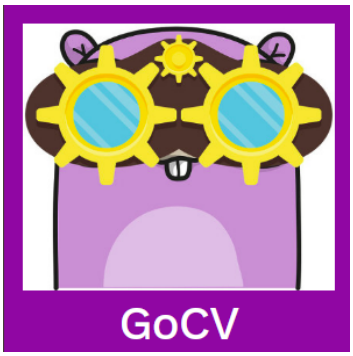
- Currently, our video streaming application works on the transfer of an m3u8 video cluster in the form of chunks of TS files. We can improve upon this idea to implement a **live video streaming application through a webcam.**
- The basic idea is that the server streams a video through a webcam in real-time 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 that 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

Innovation in Video Streaming Algorithm

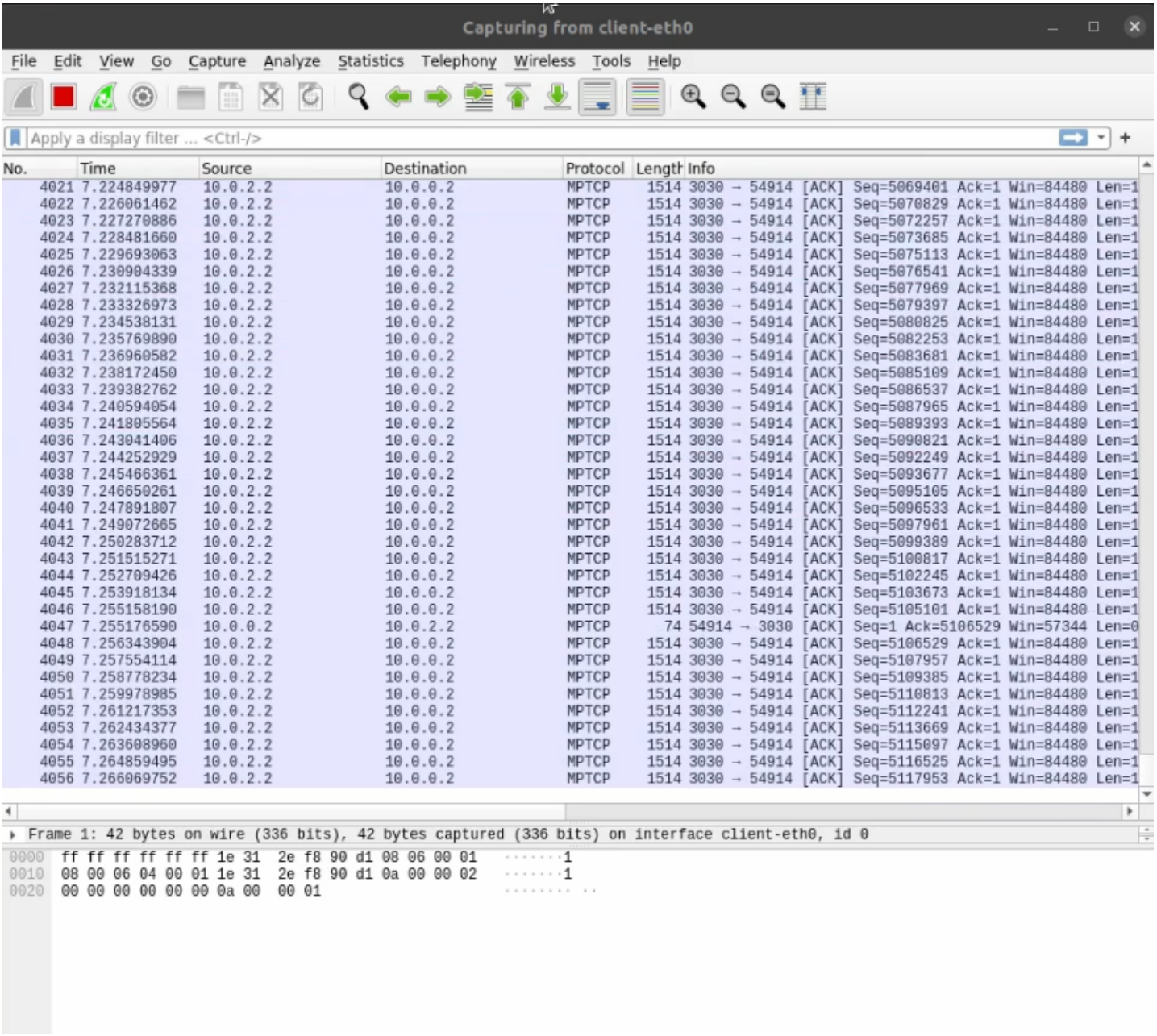
- Previously, as part of this project, we were transferring video files as m3u8 video clusters in the form of chunks of TS files. Later we came up with the idea of live-streaming video from the host webcam.
- We have implemented a **video streaming algorithm using OpenCV library for MPTCP and GoCV for MPQuic.**



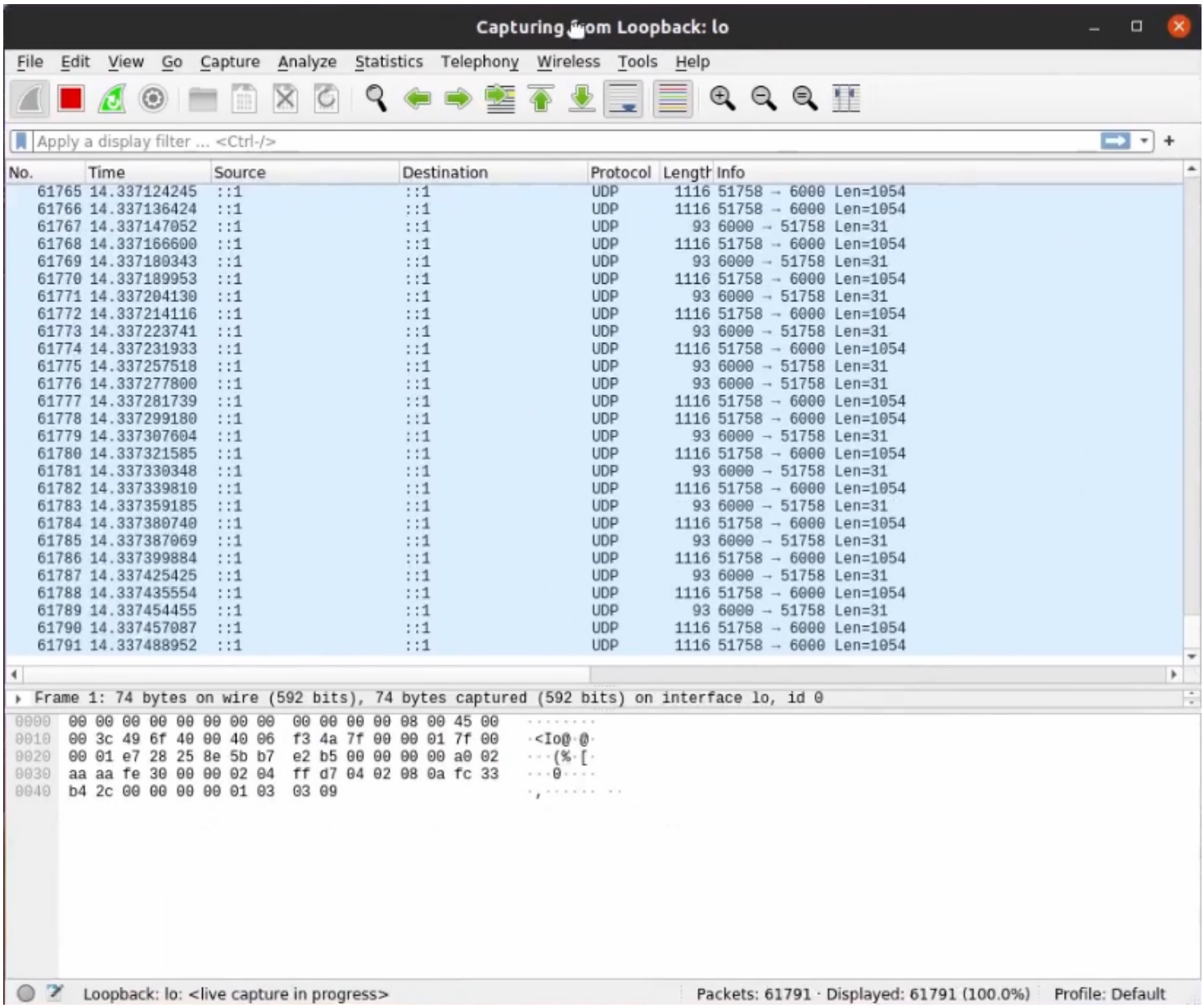
- The video is streamed to the client from the server in real-time by transferring a series of frames captured through the webcam.
- The virtual network created in the config file of mininet in Phase-1 was used as the multipath network topology for this application.
- We can see traces of Wireshark that depict that streaming is done efficiently through MPTCP and MPQUIC protocols.
- **Note**: *MPQuic traces show up as multiple UDP packet transfers in Wireshark as it is a work-in-progress and has not been yet implemented in Wireshark.*

  Reference: [Wireshark doesn't show QUIC protocol]

Wireshark traces for MPTCP



Wireshark traces for MPQuic



- Additional functionality of collecting all the frames, clustering them together and forming a video is implemented at the client side.
- We used additional python libraries in the video streaming algorithm, such as NumPy, to manipulate the frames and prepare them for transport.
- We have tested our innovation with mininet on the above network topology and in adverse network conditions. This was accomplished using the netem tool.
- We have tested our innovation against packet loss, packet corruption, packet and packet delay.

Commands for netem testing

```
sudo tc qdisc add dev lo root tbf rate 1mbit burst 32kbit latency 400ms
```

- **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
- **<this is bandwidth limiting test>**

➔ To check all active rules: `sudo tc qdisc show dev lo`

➔ To delete all active rules: `sudo tc qdisc del dev lo root`

➔ Add packet loss: `sudo tc qdisc add dev lo root netem loss 10%`

➔ Add packet delay: `sudo tc qdisc add dev lo root netem delay 200ms`

➔ Add packet corrupt: `sudo tc qdisc add dev lo root netem corrupt 20%`

Overview of the video streaming algorithm:

- Once the connection is established on the server-side, we use the cv2/ gocv library to open the webcam and immediately start streaming.
- We read frames and resize them to fit our needs and are sent to the client. We have kept 300 frames on an experimental basis, after which the webcam and the connection are closed.
- The frames are first converted into a numpy array and then sent in byte format to the client, which again decodes the data into an opencv frame.
- On the client-side, it receives all the frames until the data is not empty.
- We check for DataExcededError for each frame received from the server and store and show it back into a new frame after manipulation.
- We store each frame onto a separate folder in jpg format, and before the connection is closed, we convert these frames to a video file with a customised frame rate. This way, the client has access to frames, and video files streamed from the server at any given time.

Video Presentation

Video Link: Video Streaming using MPTCP and MPQUIC

Topic	TimeStamp
Config file (Explanation of network topology using diagrammatic representation)	0:00 - 1:43
Mininet (Ping demo to show multipath data transfer)	1:44 - 4:50
Code Overview (server-side and client-side video streaming algorithm code for MPTCP and MPQuic protocols, code for frame clustering to form video)	4:51 - 8:48
Wireshark traces for MPTCP (Video streaming demonstration)	8:49 - 12:30
Wireshark traces for MPQuic (Video streaming demonstration)	12:31 - 14:04