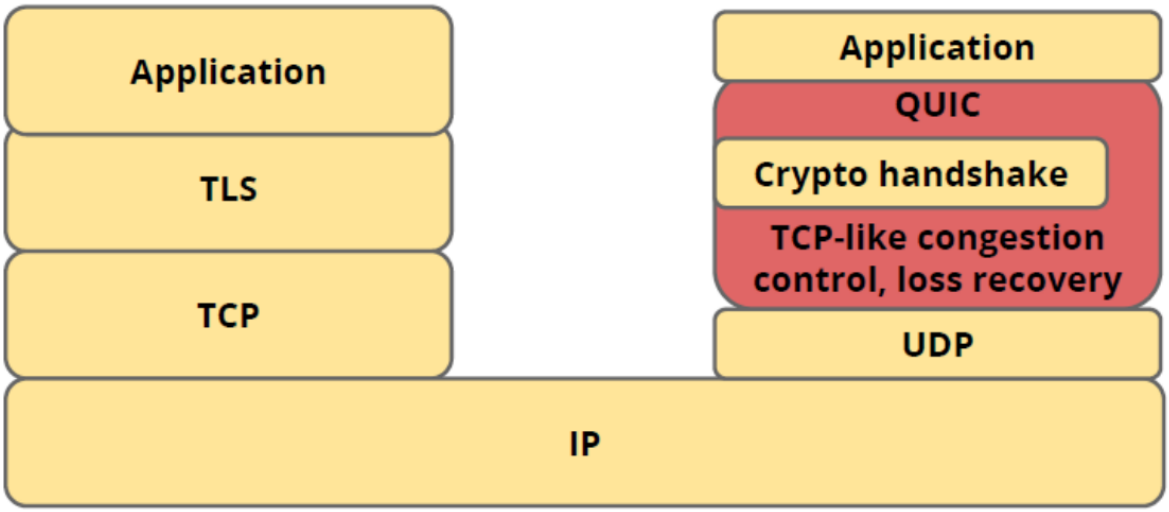
QUIC protocol Interview Prep

What is QUIC?

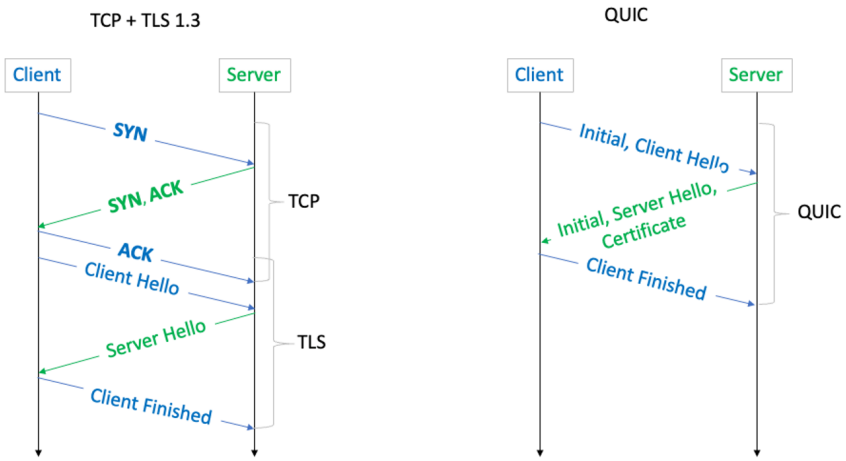
QUIC (Quick UDP Internet Connections) is a new transport protocol for the internet, developed by google. It solves a number of transport-layer and application-layer problem experienced by modern web applications, while requiring a little change or no change form application writers.



Key features:

Key features of QUIC over existing TCP+TLS+HTTP2 include

- Dramatically reduced connection establishment time



- Improved congestion control
- Multiplexing without head of line blocking
- Forward error correction
- Connection migration

Sure, here's the formula for the cubic function used in the Cubic algorithm for congestion control:

```
W_cubic(t) = C*(t-K)^3 + W_max
```

Where:

- `W_cubic(t)` is the congestion window size at time `t`
- `C` is a constant that controls the aggressiveness of the algorithm
- `K` is the time at which the congestion window was last reduced
- `W_max` is the maximum allowed congestion window size

Note that this formula is used to calculate the growth of the congestion window over time, and is used in conjunction with other parts of the Cubic algorithm to maintain fairness and avoid congestion collapse.

QUIC supports multiplexing without head of line blocking. This means that multiple streams of data can be sent over a single connection, and if one stream is blocked for any reason (e.g. waiting for data to arrive), other streams can continue to be processed.

This improves performance by reducing the number of connections needed to transfer data, and by allowing data to be transmitted more efficiently.

Goals/Metrics

Analyze performance of TCP and QUIC in terms of:

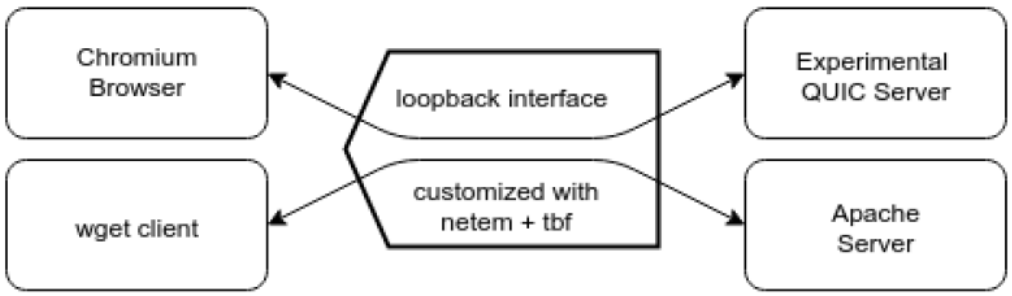
- Total transfer time
- Average bandwidth used
- Overhead in bytes

Methods

Experimental Setup:

A 33.6 MB testfile `index.html` will generate in `/var/www/html/` and we will get it from *quic server* and *apache2 server* with *quic client* and *wget*. The protocol two way used is **QUIC** and **TCP**. And we will run the experiments under difference network enviroments.

For practical, we will use simulate enviroment in **local**. We use *tc netem* and *tbf* to config local loopback interface.



Experimental Platform

- Hardware

Hardware	Parameters
Memory	16GB

Processor	Inter i5 12500H
Disks	SAMSUNG M.2 SSD

- Software

Software	Parameters
OS	Ubuntu22.04
OS-type	64 bit
Kernel	Linux 4.4.0-104-generic
GCC	GCC 5.4
Python	Python 3.10

Compile Chromium

Because of the quic protocal is embedded in Chromium, so we must build our *quic_server* and *quic_client* from the source of Chromium.

1. clone the source of chromuim
2. building for the first time, install dependencies

```
./src/build/install-build-deps.sh
```

1. Build the QUIC client, server, and tests:

```
cd src
gn gen out/Default && ninja -C out/Default quic_client quic_server net_unittests
```

1. Prepe test data from www.example.org

```
mkdir /tmp/quic-data
cd /tmp/quic-data
wget -p --save-headers https://www.example.org
```

The command `wget -p --save-headers <https://www.example.org` > downloads a web page, its resources, and saves the HTTP response headers using the `wget` utility.

1. Generate certificates In order to run the server, you will need a valid certificate, and a private key in pkcs8 format.

```
cd net/tools/quic/certs
./generate-certs.sh
cd -
```

1. In addition, a CA certificate was also generated and trusted by following the instructions in the ‘Linux Cert Management’ page located in the Chromium website

Apache2 Server

We will test TCP with Apache2 Server, to be closer to the reality world, we config the server with TLS.

1. Create the SSL Certificate (An SSL certificate is a digital credential that enables secure, encrypted communication between a web browser and a server, ensuring data confidentiality and authenticity.)
2. Configure Apache to Use SSL
3. Adjust the Firewall
4. Enable the Changes in Apache

Prepare for Experiments

Before we start the experiments, we need finished this four steps:

1. Set loopback interface mtu to 1500
2. IPv6 disabling on loopback
3. Start Apache2 Server
4. Start QUIC Server

See detail in env_setup.sh.

Run and Analyze

Usage

```
./scripts/env_setup.sh
./scripts/run.sh
./scripts/analyse.sh
```

The Emulating Environment's

1. Control Parameters **bandwidths** : Limiting the maximum link bitrate. **delay** : One-way delay to packets that are going from a server to client. **losses** : Drop packets that are going from a server to client. **spikes** : A period of time(default 200ms) when bandwidth drop to a certain percentage.
2. Parameters with values used in our experiments

```
protocal = ['quic', 'tcp']
bandwidths = ['100', '40', '5']
delay = ['10', '50'] or ['10', '20', '40', '60', '80', '100', '120']
losses = ['0.0', '5.0']
spikes = ['0', '1']
```

In the context of this document, spikes refer to a period of time during which the available bandwidth drops to a certain percentage.

Details

1. Generate raw data This function is finished in `run_benchmark.py`, the scripts include three steps:

- Generate the `Params Queue` from the arguments parsing
- Configuration of local loopback interface for every params
- Data captured with *tcpdump*, and stored into `./raw/` for every params.

`tcpdump` is a command-line tool used for capturing and analyzing network traffic in real-time. It allows you to monitor the data flowing across a network interface and display information about each packet, such as source and destination addresses, protocols, port numbers, and payload data. This tool is valuable for diagnosing network issues, monitoring network activity, and analyzing the behavior of network applications.

Raw (pcap) data contains captured network packets in their original binary format, including header and payload information, enabling detailed analysis of network

communication, protocol behavior, and troubleshooting.

1. Data Analysis This function is finished in `preprocess.py` and `average.py`, the scripts include three steps:

- Clean the raw data and stored the preprocessed data in `./processed/`, in order to extract only the data required (timestamp and bytes).
- `average.py` averages different instances of the same test. By default, each test is run five times.

1. Visualization This function is finished in `plot.py` and `plot2.py`, the scripts include three steps:

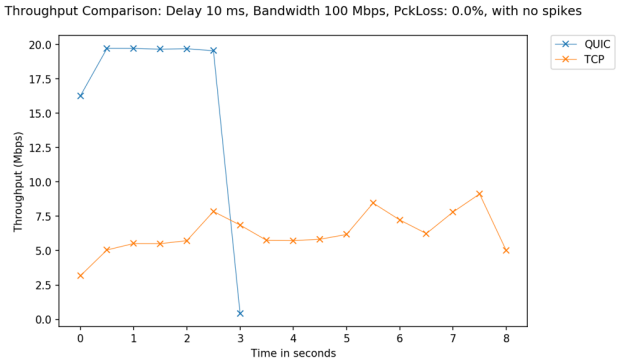
- Creates all plots that are not time series (bandwidth, overhead and time) vs (delay, bandwidth, packet loss)
- Generates time series for the processed data extracted from the tests in the `/processed/` folder
-

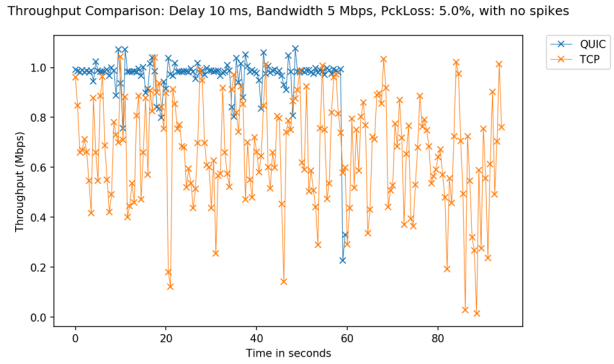
```
sudo ifconfig lo mtu 1500
sudo sysctl -w net.ipv6.conf.lo.disable_ipv6=1
```

The first command sets the Maximum Transmission Unit (MTU) size of the loopback interface to 1500 bytes, and the second command disables IPv6 on the loopback interface.

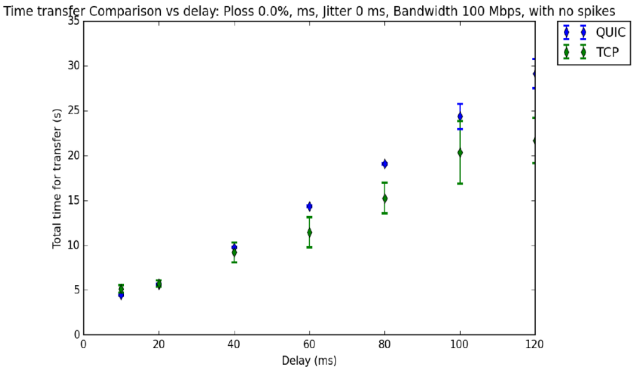
Results

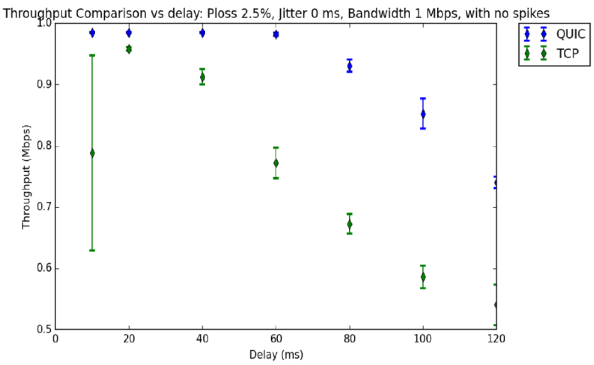
Time series



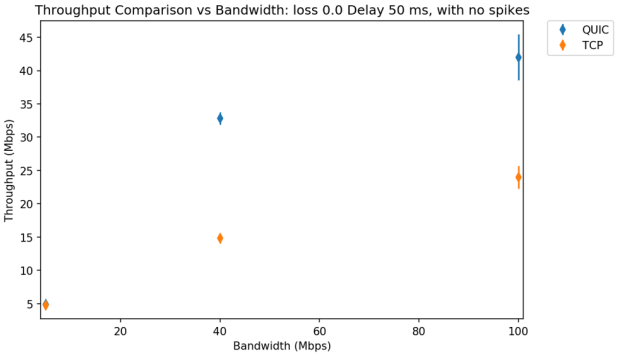


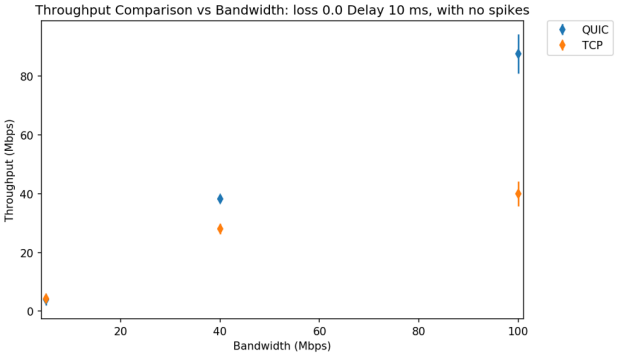
delay



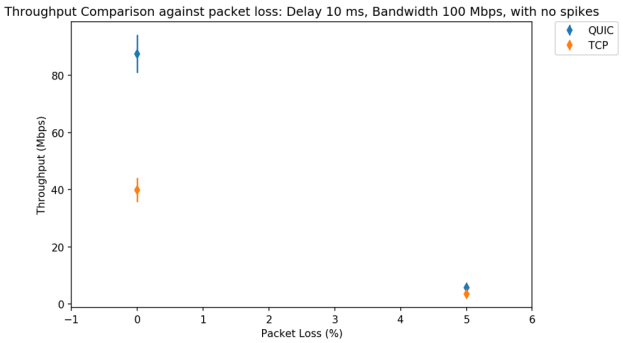


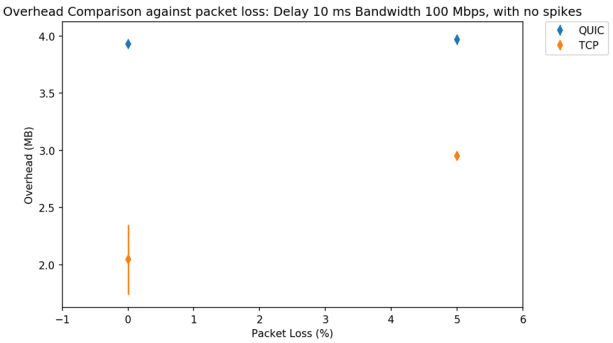
bandwidth



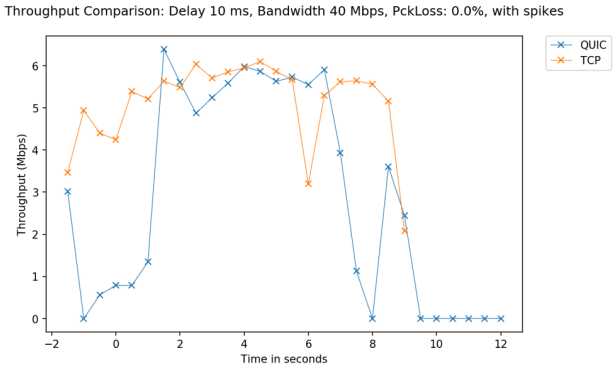


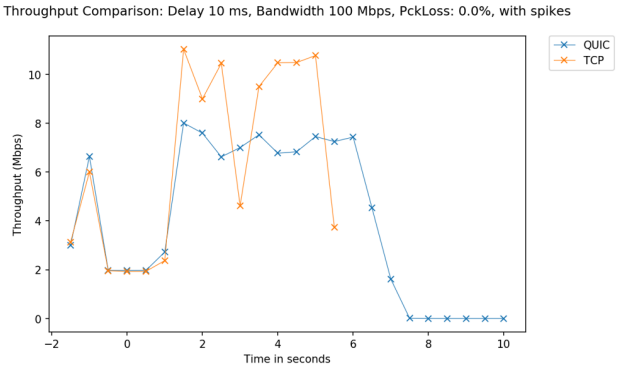
packet loss

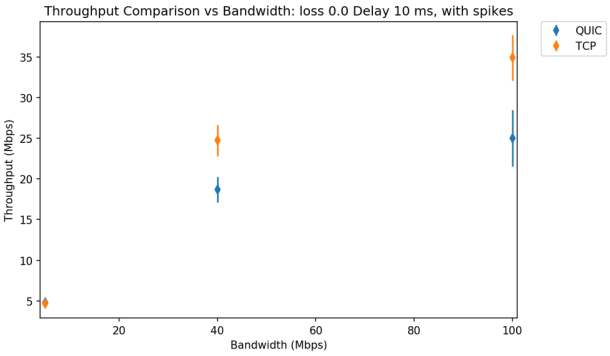


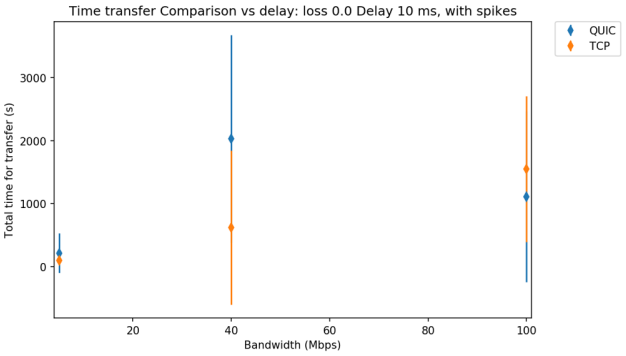


Jitter









Analysis

- At the cost of higher overhead, QUIC outperforms TCP in terms of time for transfer and average bandwidth used.
- When high delay, packet loss, and high bandwidth, QUIC will perform much better than TCP including time for transfer and throughput.
- Under favorable conditions, The QUIC will be more stable than TCP. You can see two picture in section Time series.

- Under packet loss, QUIC also surpasses TCP. When packet loss is 0%, throughput of QUIC is much higher than TCP. When packet loss is 5%, throughput of two protocol is very close, but QUIC is higher still.
- But when jitter happen, TCP can surpasses QUIC. Because the feature of the QUIC, QUIC can't handle the jitter better than TCP. It imply that QUIC is immature and not prefect.

Conclusions

QUIC is a new network protocol that resides in the application layer over UDP. Google developed QUIC as an alternative to TCP. Two browsers (Chrome and Opera) and Google servers are the only entities that support QUIC. When a user accesses Google’s services such as Gmail over the aforementioned browsers, the data transfer will use UDP-based QUIC.

Future Works

- Designing new tests to measure fairness when sharing bandwidth with other QUIC/TCP flows
- Stream Multiplexing in QUIC: Evaluate advantages over loading HTTP pages, for example.
- Connection Migration
- QUIC over a Wireless Network

Analysis of Terminal Commands in run_benchmark.py

1. Introduction

This document provides an analysis of the terminal commands used in the 'run_benchmark.py' script. These commands are used for various purposes including network emulation, file transfers, and packet capturing.

2. Netem Commands

- `sudo tc qdisc del dev lo root`: Deletes any existing queuing discipline (qdisc) rules on the loopback interface ('lo').
- - `sudo tc qdisc show dev lo`: Shows the current qdisc rules on the loopback interface ('lo').
- - `sudo tc qdisc add dev lo root handle 1: netem`: Adds a new NetEm (Network Emulator) qdisc rule to the loopback interface ('lo'). - `delay xxxms`: Adds a delay in milliseconds to packets. - `xxxms distribution normal`: Adds a normal distribution to the latency. - `loss xxx%`: Adds packet loss with a given percentage.

3. Bandwidth Commands

- `sudo tc qdisc add dev lo parent 1: handle 2: tbf rate xxxmbit burst 256kbit latency 1000ms mtu 1500`: Adds a Token Bucket Filter (TBF) to control bandwidth. - `sudo tc qdisc change dev lo parent 1: handle 2: tbf rate xxxmbit burst 256kbit latency 1000ms mtu 1500`: Changes the existing TBF rule.

4. Ping Command

- `/bin/ping 127.0.0.1 -c 1 -s 1`: Sends a single ICMP echo request packet of size 1 byte to the loopback address.

5. Client Commands

- `wget -O ./tmp/index.html https://127.0.0.1/{testFile} --no-check-certificate`: Downloads a file over HTTPS without checking the certificate. - `/home/csg/software/proto-quic/src/out/Default/quic_client --host=127.0.0.1 --disable-certificate-verification --port=6121 https://www.example.org/`: Uses a QUIC client to download a file from a server.

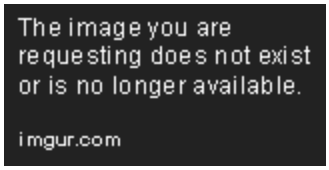
6. Tcpdump Commands

- `touch /tmp/test.pcap`: Creates an empty pcap file for packet capturing. - `sudo tcpdump -i lo -w /tmp/test.pcap`: Captures packets on the loopback interface and writes them to a pcap file. - `sudo kill`: Kills a process (presumably the tcpdump capture). - `tcpdump -r /tmp/test.pcap -tttttnnqv`: Reads the pcap file and prints detailed packet information.

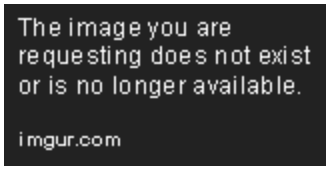
7. Chromium Command

- `google-chrome --user-data-dir=/tmp/chrome-profile --no-proxy-server --enable-quic --origin-to-force-quic-on=www.example.org:443 --host-resolver-rules='MAP www.example.org:443 127.0.0.1:6121' https://www.example.org/`: Opens Google Chrome with specific flags for QUIC testing.

As you can see, CUBIC tends to have a more aggressive growth rate when network conditions are favorable, while AIMD maintains a more stable growth rate. However, both algorithms are effective at maintaining network stability and avoiding congestion collapse.



CUBIC Congestion Control



AIMD Congestion Control

Here's a comparison of the two algorithms:

CUBIC, on the other hand, uses a cubic function to calculate the growth of the congestion window over time. This algorithm aims to better utilize available bandwidth while still avoiding congestion collapse through the use of a concave growth curve.

AIMD uses a linear increase in the congestion window size until packet loss is detected, at which point it reduces the window size multiplicatively. This method is effective at maintaining network stability and avoiding congestion collapse.

AIMD (Additive Increase Multiplicative Decrease) and CUBIC (Cubic TCP) are both congestion control algorithms used in TCP.