UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Congestion Control Tuning of the QUIC Transport Layer Protocol

Spring 2018

Wendi Qu

Director: Llorenç Cerdà-Alabern

Departament d'Arquitectura de Computadors

Degree: Bachelor

Specialization: Information Technologies

Facultat d'Informatica de Barcelona (FIB)

Universitat Politecnica de Catalunya (UPC) - BarcelonaTech

April 2018

Abstract

The QUIC protocol is a new type of reliable transmission protocol based on UDP. Its establishment is mainly to solve the problem of network delay. It is efficient, fast, and takes up less resources. The QUIC gathers the advantages of both TCP and UDP.

The first part of this thesis studies the development background of the QUIC protocol in terms of characteristics and perspectives of what they can do and how they work. Because it adds the congestion control algorithm used by TCP based on the UDP protocol, we have conducted further research and analysis of the Cubic algorithm to investigate the impact of its parameters on the behavior.

The second part includes performance and fairness tests for QUIC and TCP implementations. The simulation framework Mininet is used to perform these tests using controlled network properties. In this process we verified the reliability of the mininet. This work shows how Mininet builds a test system to analyze the implementation of the transport protocol. QUIC's tests show that the performance of QUIC has improved, and the test of fairness have identified specific areas that may require further analysis.

In the third part, we test the influence of the parameter on the behavior of the algorithm in the congestion control algorithm. We present an initial experimental evaluation of the newly proposed Cubic-TCP algorithm.

Key words: QUIC, TCP, congestion control, fairness.

Contents

Chapter 1

Introduction

The use of computer applications is becoming a very common practice in the modern society. Just as we speak in a language, there is also a language between computers on the network, that is the network protocol. Different computers must use the same network protocol to communicate.

The Transmission Control Protocol (TCP) [1]is the foundation of the Internet of yesterday and today. In most cases it simply just works and is both robust and versatile. However, in recent years there has been a renewed interest in building new reliable transport protocols based on the unreliable User Datagram Protocol (UDP)[2].

In this thesis, we will first study one reliable UDP-based protocol-----QUIC[1] to learn what problems and situations it is trying to handle better, why Google want to launch it and how it is different from TCP.

Google's Quick UDP Internet Connections (QUIC), which implements TCP-like properties at the application layer atop a UDP transport, is used for transporting web requests and responses [3].

The second part of this thesis examines QUIC and TCP in more detail to test actual protocol implementations using network emulation. These tests are performed to evaluate the implementations' performance characteristics in different network situations and investigate the fairness of QUIC when competes with TCP.

Within a certain period of time, the demand for resources (link capacity, buffers in the switching node, etc.) in the network is greater than that available, causing congestion [4]. In order to solve the problem of network congestion, in addition to appropriately increasing the buffer capacity, increasing the link bandwidth as much as possible, and improving the capabilities of the processor, a congestion control mechanism is also needed.

Transport-layer congestion control [5] is one of the most important elements for enabling both fair and high utilization of Internet links shared by multiple flows. QUIC use a congestion control algorithm named Cubic which is also used TCP.

So, the next part of this thesis is to learn the aspects of this algorithm the QUIC improves and to investigate the congestion control window when tuning the parameter

of this algorithm.
The tests were performed using Mininet[6], a software based network emulator, which makes it possible to test implementations using different network properties in a controlled environment.

This thesis aims to give the reader a detailed description of how to test the performance.

1.1 Background

QUIC (Quick UDP Internet Connections, pronounced quick) was introduced in 2013, included as a separate module in the Chromium source which is an experimental transport layer network protocol designed by Google is aim at improve the speed of network transmission. [7]

TCP is connection-oriented, and more emphasis is placed on the reliability of the transmission. UDP is connection-free, that is, it does not need to establish a connection before data exchange between the two parties of the communication. It is only necessary to know the address of the other party to send data, because UDP protocol is none. The protocol of the connection mode, so it is efficient, fast, and takes up less resources. The QUIC gather the advantages of both which the other protocols do not have, that is also why QUIC is attractive.

1.2 QUIC Development

It has undergone rapid development by Google developers and has been deployed by companies such as Google and Akamai, with more than 20 implementations in progress, including for Microsoft, Mozilla, Verizon, and Facebook. Beyond Google, applications such as Snapchat have started to adopt QUIC, and more could follow in 2018. While some of us thought QUIC would “only” grow linearly with Android traffic, iOS devices have also started to adopt QUIC for YouTube. Google is moving to QUIC on the latest iOS and YouTube app versions.

1.3 Objectives

The main objective of this project is investigating experimentally the performance of the QUIC protocol and compare it with TCP to evaluate their performance and

congestion handling.

In particular, the thesis tries to answer the following questions:

Why study QUIC?

Which features do QUIC offer compared to TCP and how do they work?

Which parameters of Congestion control will have the influence on the protocol?
How can mininet be used to test implementation of QUIC?

How does the QUIC implementation perform compared to TCP?

What affect the fairness of QUIC when competing with TCP?

1.4 related works

Transport-layer performance. There is a large body of previous work on improving transport-layer and web performance, most of them focusing on TCP [8] and HTTP/2 . QUIC builds upon this rich history of transport-layer innovation. Vernersson[10]uses network emulation to evaluate UDP-based reliable transport, but does not focus specifically on QUIC.

QUIC emulation results Closely related to this work, several papers explore QUIC performance. Megyesi [9] use emulated network tests with desktop clients running QUIC version 20 and Google Sites servers. They find that QUIC runs well in a variety of environment.

1.5 Our contributions

This work makes the following new and extended contributions compared to prior work.

A lot of related work is to test the quic in practical applications, but I run the implementation of QUIC in the virtual network performance, so you can more intuitively compare QUIC performance optimization over TCP. Secondly, this experiment also focuses on analyzing the influence of parameters in the congestion control algorithm. Most of the experiments still analyze the effect of the algorithm on performance.

1.6 Thesis outline

Chapter 2 begins with a short introduction of the born of QUIC and why QUIC is attractive. There after follows the study of QUIC mechanisms to see what problems it is trying to solve and how they work, including related work. Further, the analysis about congestion control can help to understand why need TCP CUBIC congestion control algorithm and what its equations are like.

Chapter 3 describes the methodology of the work.

Chapter 4 describe the initial plan of the work.

Chapter 5 explains in detail the budget required for this project and the sustainability of the project..

Chapter 6 details how Mininet was used, including verification that the network emulation works as expected.

Chapter 7 and Chapter 8 contains the actual tests of QUIC and TCP about their comparation of performance and fairness. First describing how the tests were performed, followed by results and analysis.

Chapter 9 test the congestion windows tuning the parameters of Cubic, a congestion control algorithm.

Chapter 10 Problems and solutions that were encountered during the process are also described.

Chapter 11 contains the concluding words of this thesis; about the protocols, implementations and Mininet.

Appendix A lists details about the test system, hardware, software and tested versions.

Chapter 2

Theory

2.1 QUIC motivation

With the rapid development of the mobile Internet and the gradual emergence of the Internet of Things, the scenes of network interaction are becoming more and more abundant, and the content of network transmission is also becoming increasingly large. The users' demands for network transmission efficiency and WEB response speed are also increasing.

Initially, the developers want to find a protocol can increase the stability of the connection in order to dealing with highly variable network. Constant transition from this wifi to that wifi, intermittent cellular data usage, occasional cellular signal blackouts--this all makes mobile Internet connections very unstable and unreliable. loading web pages may seemingly take ages to finish. This kind of environment poses serious user experience problems. So--from the user's perspective--it is wiser to simply press the refresh button in this situation instead of waiting for the loading to complete. About this point, the TCP protocol has been difficult to improve. However, UDP protocol is a connectionless protocol which is efficient, fast, and takes up less resources.

Fortunately, Google is trying to solve this problem by developing and researching a new protocol named QUIC beyond UDP and take bidirectional control of bandwidth to avoid network congestion.

The goals of QUIC are many but essentially Google wants a protocol that can be deployed on today's Internet that reduces latency and also solves problems with multiple streams over a single TCP connections, in other words, to integrate the reliability of the TCP protocol and the rapidity and efficiency of the UDP protocol.

2.2 QUIC Mechanisms

QUIC replaces most of the traditional HTTPS stack: HTTP/2, TLS, and TCP , Figure 1 is how an application stack differs between TCP and QUIC.[11]

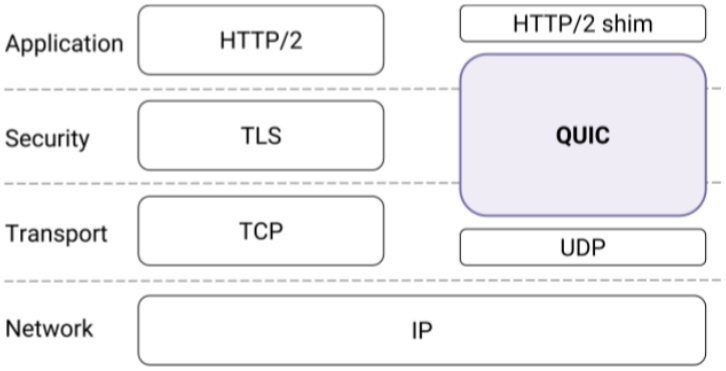


Figure 1: QUIC in the traditional HTTPS stack

The main features of QUIC include: All the advantages of SPDY (SPDY is a protocol developed by Google to improve HTTP speed, which is the basis of HTTP/2.0); 0-RTT connection; reduce packet loss; forward error correction, reduce retransmission delay; Adaptive congestion control, reducing reconnection.

The main performance improvement of QUIC over TCP come from two key differentiators:

Connection handshake[12]: TCP required a 3-way handshake to establish a connection, and, on top of that, you also need to negotiate the TLS connection. So, they wanted to reduce the e ff ects of round-trip time(RTT) when establishing new connections. By integrating TCP and TLS in a single protocol QUIC can avoid two sequential handshakes. QUIC can more importantly completely avoid round-trips, called 0-RTT connection latency. Clients that have previously communicated with a server can start a new session without a three-way handshake, using limited state stored at clients and servers. This shaves multiple RTTs from connection establishment. See figure 2 for a brief connection establishment comparison between TCP/TLS and QUIC.

Actually, if the client and the server have spoken in the past, then we are talking about a zero-handshake connection – that happens 75% the time.

Multiplexing: the communication between the client and the server is multiplexed and this overcomes the head-of-line blocking issues that are common with TCP connections. Individual QUIC streams can for example be decrypted independently. Multiplexing streams in TCP also leads to a bandwidth disadvantage compared to parallel

connections, partly because a lost packet reduces all streams bandwidth and partly because multiple connection can increase the total bandwidth faster during slow-start.

To compensate for this, QUIC's streams also have individual congestion control.

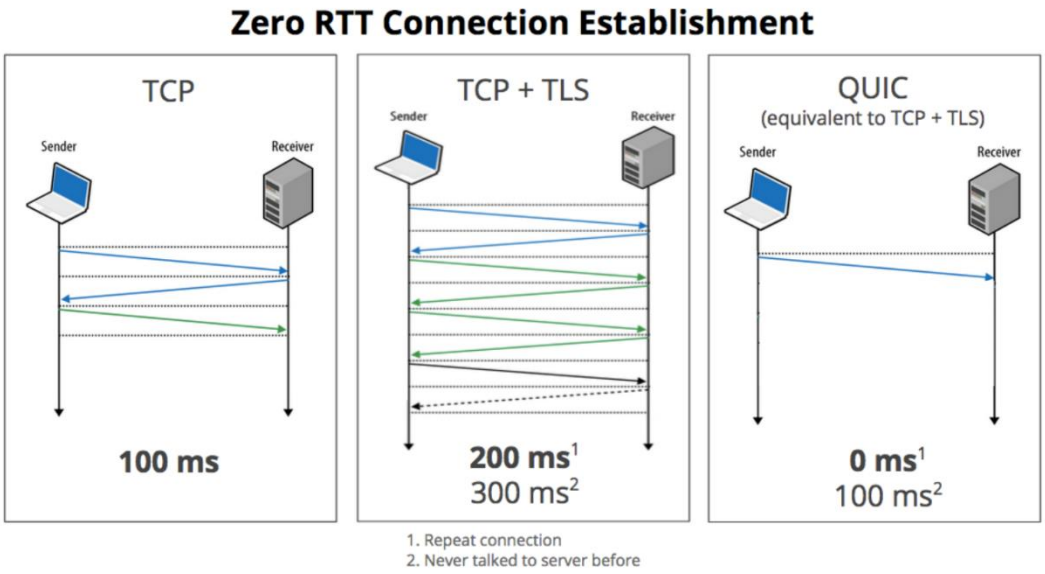


Figure 2 Comparation of connection establishment

In summary, the most attractive feature of the QUIC protocol has two point. First, Solve the problem of team leader blocking more thorough. Another feature is to keep the connection while switching networks.

2.3 Congestion control

Transport-layer congestion control is one of the most important elements for enabling both fair and high utilization of Internet links shared by multiple flows.[13]

At a certain time, if the demand for a resource in the network exceeds the available part of the resource, the performance of the network will deteriorate. This situation is called congestion.

The purpose of congestion control is to perform corresponding processing in the case of system overload, so that the system recovers to the normal load level and guarantees stable operation of the system. Congestion control is a global process. However, UDP itself is not controlled by congestion. Once unconstrained use, it will invade the bandwidth of other "rule-worth" network protocols.

Therefore, the UDP-based QUIC protocol draws on some of TCP's excellent congestion control algorithms [14]. For example, Cubic is used by default. At the same time, packet pacing is used to detect network bandwidth in order to avoid the low bandwidth utilization caused by the AIMD mechanism.

From the perspective of the congestion algorithm itself, it looks like the QUIC protocol is just a re-implementation of TCP's congestion algorithm, which is not the case. The QUIC protocol makes some improvements based on the TCP congestion algorithm:

Pluggable

- Different levels of congestion control algorithms can be implemented at the application level without the need for operating system or kernel support.
- Different connections for a single application can also support configuring different congestion controls.
- Changes to congestion control can be implemented without downtime and upgrades.

Monotonically increasing Packet Number

QUIC does not use TCP's byte order number and ACK to confirm the orderly arrival of the message. QUIC uses the Packet Number. Each Packet Number is strictly incremented, so if Packet N is lost, the Packet Number that retransmits Packet N is not. N, but a value greater than N. This makes it easy to solve the problem of TCP retransmission ambiguity.

More ACK blocks

The QUIC ACK frame supports 256 ACK blocks. Compared with the TCP SACK implemented in the TCP option, there is a length limitation, and only up to 3 ACK blocks are supported.

Accurately calculate RTT time

The QUIC ACK packet also carries the delay from the receipt of the packet to the reply ACK. In this way, the incremental packet number can be used to accurately calculate the RTT.

2.3.1 CUBIC Window Growth Function

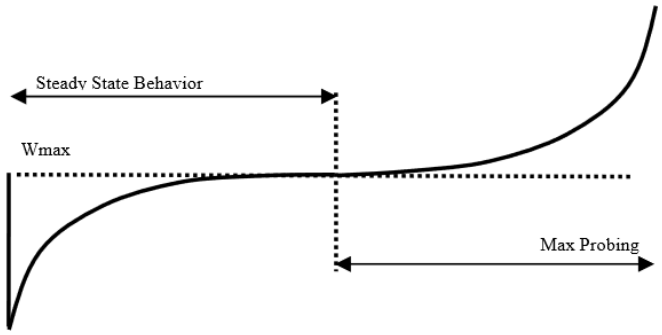


Figure 3: The Window Growth Function of CUBIC

the congestion window of CUBIC is determined by the following function:

$$W_{cubic} = C(t - K)^3 + W_{max}$$

where C is a scaling factor, t is the elapsed time from the last window reduction, W_{max} is the window size just before the last window reduction, K is the time period that the above function takes to increase W to W_{max} when there is no further loss event. And $K = \sqrt[3]{W_{max}\beta/C}$, where β is a constant multiplication decrease factor applied for window reduction at the time of loss event (i.e., the window reduces to βW_{max} at the time of the last reduction).

Fig. 3 shows the growth function of CUBIC with the origin at W_{max}. The window grows very fast upon a window reduction, but as it gets closer to Wmax, it slows down its growth. Around W_{max}, the window increment becomes almost zero. Above that, CUBIC starts probing for more bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from W_{max}. This slow growth around W_{max} enhances the stability of the protocol and increases the utilization of the network while the fast growth away from W_{max} ensures the scalability of the protocol.

Chapter 3

Methodology

We now describe our methodology for evaluating QUIC, learning its congestion control algorithms and comparing it to the TCP The tools we developed for this work and the data we collected are publicly available.

3.1 Information collection

First of all, we need a rigorous project analysis and careful project plan. This phase is the safeguard against risks and the success of the project. Only do we learn about whether the project is even possible can we decide on this project.
the we need to assure that the project is something we are able to do. For example, compared to rewriting a protocol, it is obviously a better choice to study the latest protocol.

The second part of this work naturally consisted of looking into available protocols. Further, get the knowledge of the computer network or other related paper concerning about the protocols. QUIC were chosen due to it widespread usage on the Internet compared to other candidates such as Structured Stream Transport (SST)[16] or UDP-based Data Transfer (UDT)[15]. And also, the function of QUIC is similar to TCP but with a more reliable and more efficient performance. So, comparing QUIC and TCP is necessary.

The next step is to read the documentation of QUIC and learn the congestion control implemented by QUIC. Read the protocols specifications and published articles to get an understanding of the protocols and current knowledge. This is also used to decide what kinds of tests that would be interesting to perform, such as verifying a specific feature or goal in a protocol or further analyzing a known problem. Learn the congestion control algorithms is helpful for us to research the source code of QUIC in later tests.

3.2 Source code

The implementation of QUIC was tested because it's a new and relatively complex protocol. QUIC is also interesting because of its wide deployment and availability in the Chrome browser [6][48]. What is more, an open source code implementation of QUIC can be search from Github. [17] I choose the QUIC-GO, a quick way to use QUIC, as an implementation of the QUIC protocol in Go [] . The implementation includes a test server and a test client which can be used for experimentation but are not tuned for production-level performance.

Quic-go is compatible with the current version(s) of Google Chrome and QUIC as deployed on Google's servers. It is one version the author of QUIC designed only to test QUIC. The author of quic-go is actively tracking the development of the Chrome code to ensure compatibility as the protocol evolves. In that process, we're dropping support for old QUIC versions. As Google's QUIC versions are expected to converge towards the IETF QUIC draft [18], quic-go will eventually implement that draft. So, it is valuable to choose QUIC-GO as the version of implementation of QUIC in my project.

After launched the QUIC-GO following the instruction in the Github, I need to Investigate the source code of QUIC and learn how the congestion control mechanism has been implemented. The QUIC use the Cubic algorisms. Regarding the quic-go implementation

I think the main implementation of Cubic is in this file
:~/go/src/github.com/lucas-clemente/quic-go/internal/congestion/cubic_sender.go

So, next I need to identify the parameters that can affect the performance of the congestion control implementation in QUIC.

3.3 Test method

Since the goal is to test the actual protocol implementation in different scenarios, the network simulator is chosen as the test method. This makes it possible to carry out controlled experiments through actual implementation. When a parameter is changed, the test can be repeated. You can use constant parameters to repeat the test to see the results change. This method is used to compare the implementation of various network conditions and display the results in several charts without testing all combinations.

3.4 Testbed

In this Section we describe the testbed employed to carry out the experimental evaluation of QUIC. We employ the testbed configurations shown in Figure 4.

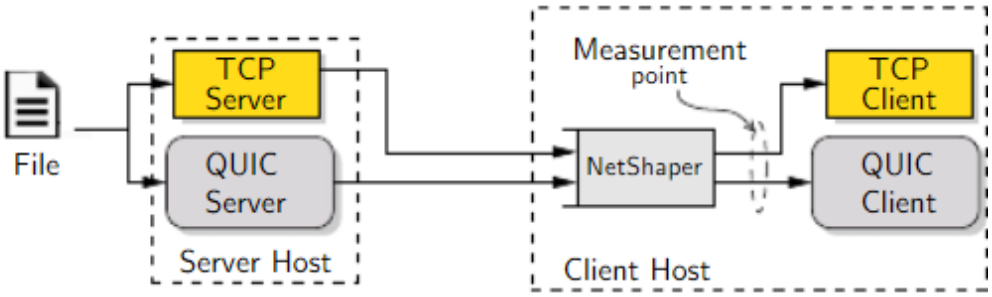


Figure 4 testbed

3.4.1 Testing download

First of all, we need an object (file) as one of the application scenarios is showed in the figure, so, we can download a test file (index.html) provided by https://www.example.org. , The next step is to set this HTML file of a desired size for testing download (filled with random data).

HTML (Hypertext Mark-up Language): a hypertext markup language or hypertext mark-up language. It is currently the most widely used language on the Internet and is the main language constituting webpage documents. HTML files are descriptive texts composed of HTML commands. HTML commands can describe text, graphics, animations, sounds, forms, links, and so on.

Python example are showed below to create a large index.html

1download index.html from www.example.org
mininet-vm # mkdir ~/quic-data
mininet-vm # sudo chown mininet:mininet quic-data/
mininet-vm # cd ~/quic-data
mininet-vm # wget https://www.example.org

2create a random le of 4Mbytes
mininet-vm # head -c 4M < /dev/urandom > random.data
3. convert to base64

mininet-vm # base64 random.data > random.data.base64
4. append random.data.base64 after <body> element of index.html
mininet-vm # cp index.html index.html.dst
mininet-vm # sed '/<body>/ r random.data.base64' index.html.dst > index.html
5. check the size of the bundled index.html
mininet-vm # du -h index.html
5.5M
index.html

3.4.2 Mininet

In order to test the protocol, we need to set up an integrated network. So, I choose the mininet as the testbed. Mininet[5] is network emulator that uses existing Linux features to enable both virtual networks and light weight virtual machines. In the mininet we can

We need to test the performance of TCP and QUIC across a wide range of network conditions (i.e., various bandwidth limitations, delays, packet losses),
Mininet[19] can provide emulation of the network parameters:

Parameter
Bandwidth The amount of data that can be transmitted per unit time
Delay One-way delay for all packets.
Jitter How much should delay change between different packets.
Expressed as the standard deviation from a normal distribution.
Loss How frequently are packets dropped, in percent.
Queue size How many packets the send queue can hold.

The RTT (round-trip time): it is the time during which a bit is sent to know that the bit has been received from the perspective of the sender, that is, the duration of the transmission. RTT is about 2*delay.
The maximum capacity of a network connection is only one factor that affects network performance. Packet loss, latency and jitter can all degrade network throughput and make a high-capacity link perform like one with less available bandwidth. An end-to-end network path usually consists of multiple network links, each with different bandwidth capacity. As a result, the link with the lowest bandwidth is often described as the bottleneck, because the lowest bandwidth connection can limit the overall data capacity of all the connections in the path.

3.5 Experiments and Performance Metrics

3.5.1 Experiments

According to the thesis goals, the experiment consists of several categories:

Comparing performance of QUIC and TCP modifying the network conditions.
Run experiments with QUIC varying the congestion control parameters to test the congestion windows.
Varying their fairness when QUIC competing TCP.
Produce figures with the data gathered in the experiments to evaluate the performance of QUIC.

3.5.2 Performances Metrics

We evaluate the QUIC and TCP performance across a range of network conditions (i.e., various bandwidth limitations, delays, packet losses). In this section, we define two key application metrics that drove QUIC's development and deployment, and we describe QUIC's impact on these metrics. We use Throughput and CWND as the metrics of the performances.

Throughput

Throughput is one of the most important performance metrics of a system's test performance. Throughput refers to the amount of data that passes through a network (or a channel, an interface) per unit time, that is, the amount of data processed per second. It is a measure of how many units of information a system can process in a given amount of time. For example, in an experiment, an interface achieves 2Mbps throughput. This means that applications on one host can send data to another host at 2 Mbps. Therefore, the greater the throughput of the system, the more users or system requests the system has completed in a unit of time, and the system resources are fully utilized. Throughput can be limited by the bandwidth of the network or the nominal rate of the network.

CWND（Congestion Window）：

The key parameter of congestion control, which describes the maximum number of data packets that can be sent at one time by the server in case of congestion control. The size of the window is the size of the data stream. When testing, you can use the number of packets to represent the size of windows.The size of the congestion window depends on the degree of network speed congestion and the amount of processing data. So it can be observed as a test metrics.

3.5.3 Experiment method

In order to test the throughput, tcpdump need to be used. Using the tcpdump which is a common packet analyzer that runs under the command line. It allows the user to display TCP and other packets being transmitted or received over a network to which the computer is attached. So， in my project, the tcpdump need to listening the interface of client, it can catch the packets the server sent and dump the trace. Then, save captured packets as local files, pcap file. Pcap is a commonly used datagram storage format, and mainstream packet capture software including wireshark can generate data packets in this format.

The python example of tcpdump:

tcpdump -ni h1-eth0 -w tcp-trace.pcap

In this case, there is need a test file to read information from local files for various data packets and check the trace of network analysis and operation. So, I create a perl ascript to parse all the UDP packets from an offline Pcap file.

What is more, when it is need to test the fairness, the QUIC and the TCP is in different flows. it is must to start the connections in the same time. So we need the bash script file to achieve this acquirement.

Bash script:

```
#!/usr/bin/env bash

ofile="tcp-and-quic.pcap"


echo "starting quic"
# start quic client
cd ~/go/src/github.com/lucas-clemente/quic-go
```

```
/usr/local/go/bin/go run example/client/main.go \
        https://10.0.0.2:6121/ > /dev/null 2>1 &

echo "starting wget"
wget http://10.0.0.3:8000 > /dev/null 2>1 &

echo "done, output file: $ofile"
echo "execute killall tcpdump to stop tcpdump"
```

3.6 Alternative test methods

3.6.1 Alternative testbed

Network simulators are often used to analyze network protocols. The main difference is that the simulator runs in an isolated environment, which makes the simulation unaffected by external factors and results can be repeated. The disadvantage of using network simulation is that the simplified model does not represent the complete dynamics of the entire Internet network path and real applications cannot be used in the simulator.

For this purpose, it can be simply tested and implemented on the Internet. A testbed may consist of a device machine running Google's Chrome browser connected to the Internet through a router under our control. For Chrome, we will evaluate QUIC performance using necessary webpages consisting of some files. It will be a huge program. This can be done at different scales on several hosts, or using test benches for network research that may involve hundreds of hosts or large-scale tests involving millions of test hosts. The larger the scale, the more extensive network conditions can be tested. However, replication testing is also a challenge for practical networks because parameters are uncontrollable and external factors change over time.

The small scale available for this work would only have provided a limited set of diff erent network properties, such as round-trip times, packet loss rates and bandwidths. The large scale tests can be performed by Google since they control both the server side and the Chrome browser[28]. These along with the problem of reproducing results made me dismiss this method.

3.6.2 Alternative test tool

I find another tool to analysis the trace of network is wiresharke. Wiresharke is a powerful sniffer which can decode lots of protocols, lots of filters, and it will feel good to analyze packets on a pretty window. However, tcpdump is a CLI tool, I can use it in most system and also can use through ssh. And also, tcpdump is enough to capture traffic and write it to a file, and then later use the test file to analyze it since the project is not complicated. Although I will see captured packets on a black & white command prompt but not like using Wireshark to see them on a window, it is easier to use and simple.

3.7 Development Tools

The tools that will be used to develop the project are the following:

First of all, GO programming language is most important, a programming language developed by Google, which has been used to implement QUIC-GO. This language is similar to other language we are familiar, so it is not difficult for us to learn it.

Mininet: the main tool we used is that allows to emulate a realistic computer network in a computer as a test bed. This platform will be used to perform the QUIC experiments. More details will be introduced in the chapter: Mininet.

Tcpdump - To record packets headers and/or data to packet capture files (PCAP files), including time stamp when each packet was received.

Ping- Utilities to send Internet Control Message Protocol (ICMP) echo packets and display round-trip-time, latency and packet loss.

Gnuplot aims to capture the traffic generated in the experiments. With the graphic, we can make a more intuitive analysis of the test.

GitHub, a platform to provide the source code of the QUIC.

Interpreted language, like perl, is to parse tcpdump traces and produce the numerical results.

Chapter 4

Project planning

This section is about temporal planning, and it aims to describe the tasks that are going to be executed in order to do the project, giving an action plan that summarizes the actions that have to be taken in order to finish the project in the desired time frame. However, we have to take into account that the planning described in this project is subject to modifications depending on the development of the project.

4.1 Schedule

4.1.1 Estimated project duration

The estimated project duration is approximately 5 months. The project starts on February 14th, 2018 and the deadline is on June 30th, 2018.

4.1.2 Consideration

It is important to consider that the initial planning could be revised and updated because of the evolution of the project. So, keep in touch with the teacher and Modify the plan is required during the project.

What is more, I’m an exchange student. So, I have an tutor in UPC and an tutor in Beihang. It is better to get their feedback and synthesize their opinions.

4.2 Project planning

4.2.1 Project planning and feasibility

This phase is currently running. It appertains to Project Management Course and it includes the next four stages:
i. Project scope.
ii. Project planning.
iii. Project budget.
iv. Initial state of art.

4.2.2 Task description

4.2.2.1 Project analysis and design

The main objective of this phase is to make an accurate analysis of the project and develop the consequent design. It is developed comprehensively of individual plans for – cost, scope, duration, quality, communication, risk and resources. This phase is the safeguard against risks and the success of the project. Only do we learn about whether the project is even possible can we decide on this project.

For example, economic feasibility analysis is that if there has a higher cost, the economic use this system to strengthen the registration efficiency of information management, to provide us with a high efficiency, can save the expenditure of human resources.

4.2.2.2 Initial system set up

Before starting the development of the project, we need to prepare the environment, set up the tools required to work on it and install the necessary frameworks to develop correctly the application. First, I need to install an Ubuntu system which used in all tasks of the project. And we know that when preparing the environment, it is not always fit with us own computer, so we need to debug all the time. It may seem a simple operation, but it takes enough time to be listed as an important task in my schedule.

4.2.3 Main development

This is the most important task of the project. It covers all the tasks related with implementation and testing of the program. It can be divided in the following stages:

4.2.3.1 Acquire background in QUIC protocol

The main task of this phase is to understand the network architecture and the function of the protocol in network transmission. The QUIC protocol is based on the TCP and UDP protocols. Therefore, we need to understand the common points and differences between QUIC and the other two protocols. In this case, it will help the project proceed smoothly. Moreover, understanding the algorithm of congestion control used by the QUIC protocol can help us more accurately find the impact of parameters on performance when we study the QUIC protocol. So, In the last months I have been learning about the TCP protocol and UDP protocol.

4.2.3.2 Get familiar with software

The implementation of the QUIC protocol is best performed on the ubuntu system, so I need to be familiar with the rules of the use of the ubuntu system in advance. Moreover, I need to use the mininet to build a network topology. In order to test the performance of the quic, a complete network structure is necessary. So, it is important to be familiar with this tool. What is more, I still need to be familiar with matlab and other drawing tools, so, we can better analyze the performance changes after modifying the parameters, which can help us intuitively analyze the protocol with graphics. So, this task also requires human resources to understand it.

4.2.3.3 QUIC-GO implementation

In this phase, I need to learn about the go language and search for some paper in order that I can get knowledge of the code of QUIC-GO. And the congestion control algorithms is the most important things so that I would pay more attention on it and research for other algorithms to know the difference about them. What is more, in this phase I need to install the QUIC-GO and use quic like using the loopback (client and server in the same host). Then go for mininet to create topologies. I should try different topologies and then find the difference of the performance when use these topologies. I think this phase will take several weeks. This task also needs human resources to collect the profiling data and then analyze that data to detect the performance of the QUIC-GO.

4.2.3.4 Performance test:

In this stage, we are going to modify the parameter and research this protocol implementation. Also, we have to take into account that the program and any of the components that are part of it can be improved before the delivery of the project.

4.2.4 Final task

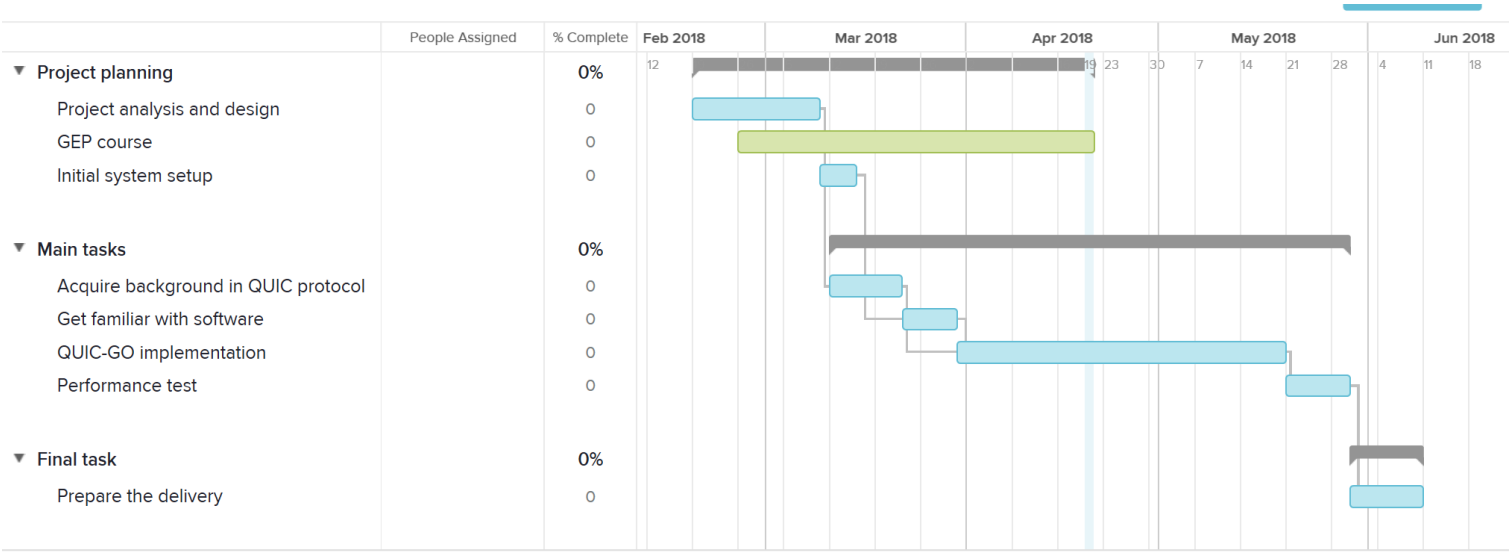
In this task we are going to check that everything works as expected and we are going to prepare the delivery of the project, assuring that the documentation is correct and preparing the final presentation.

4.3 Estimated time

Task	Estimated duration(h)
Project analysis and design	80
Initial system set up	20
Acquire background in QUIC protocol	45
Get familiar with software	35
QUIC-GO implementation	230
Performance test	50
Final task	40
Total	500

4.4 Gantt Chart

Our schedule is represented by the Gantt chart shown in the figure 1. We have taken into account the break for job and holidays mentioned before.



4.5 Action plan

Best case, I will work as what I have planned, but there would be some obstacles that may make it difficult to follow the plan. Fortunately, the agile methodology will allow us to revise and adapt dynamically the initial planning. I will try my best to do all the tasks. If is the time is not enough for me to do all the tasks stated. I think I will try to get a basic version only.

The most important thing in my project is that I need to learn about the QUIC and implement it on the mininet platform, then investigate the congestion control performed by QUIC, which I need to adjust the parameter and observe changes in individual performances. But we know that only investigate QUIC maybe is not enough. So, I am going to try to find some existing methods which can improve the performance of QUIC.

I am going to try to arrange meetings with the project tutor every time that an important stage of the project is finished. So, the tutor of the project will help me to analyses the project and confirm that the project is following a good process.

Chapter 5

Budget and Sustainability

5.1 Consideration

This section is about the budget and the sustainability of the project. For this reason, it contains a detailed description of the costs of the project, describing both material and human costs, an analysis of how the different obstacles could affect our budget and an evaluation of the sustainability of the project. Just like in the previous section, the budget described is subject to modifications depending on the development of the project.

5.2 Project budget

Project budgeting is critical to the success of any real estate development project. In this document, an estimation of the cost of the project is presented, taking into account the aforementioned hardware and software resources, and the corresponding amortizations.

To calculate the amortization we are going to take into account two factors, the first one being the useful life and the second one being the fact that our project is going to last for approximately five months.

5.2.1 Human resources budget

Budgeting involves the systematic the finances needed to support an organization’s objectives can be projected. Most organizations have some sort of process for developing a budget. This project is going to be developed only by one person. Hence, this person will need to be both a project manager and a software developer engineer, as well as a software developer engineer in test. Thus, we will need to difference between each role in the total of 500 hours. In

Table 1, an estimation of the cost is provided.

Role	Estimated hours	Estimated price per hour	Total estimated cost
Project manager	70 h	50 €/hour	3500.00 €
Software Engineer	240 h	35 €/hour	8400.00 €
Software Tester	190 h	2o €/hour	3800.00 €
Total	5oo h		15700.00 €

Table 1: Human resources budget

5.2.2 Hardware budget

In order to be able to design, implement and test all applications functionalities, a set of hardware will be needed for different purposes. In Table 2, an estimation of the cost of that hardware is provided taking into account their useful life, as well as their amortizations.

Product	Price	Units	Useful life	Total estimated amoritization
laptop	1,000 €	1	5 years	200 €
keyboard	20 €	1	5 years	8 €
Total	1,020 €		208 €	

Table 2: Hardware budget

5.2.3 Software budget

Additionally, some software products will be needed to carry out the project. Although some of them are available for free as this is an academic project, the real cost is considered. As in the hardware budget, their amortizations have been taken into account. In Table 3 the software budget is shown.

Product	**Price**	**Units**	**Useful life**	**Total estimated amortization**
Windows 10 professional	100€	1	4 years	20 €

Ubuntu 14.01	0.00€	1	N/A	0.00€
Mininet	0.00€	1	N/A	0.00€
gnuplot	0.00€	1	N/A	0.00€
Office 2016	9,99€/month	1	5 months	49.95€
GitHub	0.00€	1	N/A	0.00€
Total	149.95 €			69.95 €

Table3 software budget

5.2.4 Total budget

By adding all the budgets provided above, the total estimated budget for this project is computed, as shown in table 4

Concept	Cost
Hardware	208.00 €
Software	69.95 €
Human resources	15700.00 €
Total estimated cost	15977.95 €

Table 4 Total budget

5.3 Budget control

Budgetary control is the process of developing a spending plan and periodically comparing actual expenditures against that plan to determine if it or the spending patterns need adjustment to stay on track. This process is necessary to control spending and meet various financial goals.

When the project has the difficult or something unpredictable, we could adjust the plan and reorganize the work for the manager and other staffs. And this project is mainly use the software which are all for free. So the main probable cause of increasing the budget is that the time to test the performance of QUIC . So, creating an appropriate topology before investigating the program can avoid this deviation of causing.

Instead, a revised budget is necessary. This can happen when inflation drives prices up so high that it is not possible to stay within the original budget, requiring a revision to more accurately forecast financial performance.

5.4 Sustainability

5.4.1 Social dimension

Nowadays, everyone can get access to a computer and most people will use the Chrome desktop browsers. Google has been introduced QUIC in Chrome desktop browsers. Beyond Google, applications such as Snapchat have started to adopt QUIC, and more could follow in 2018. While some of us thought QUIC would “only” grow linearly with Android traffic, iOS devices have also started to adopt QUIC for YouTube. Google is moving to QUIC on the latest iOS and YouTube app versions. So, I think that QUIC has been actually coming to our life.

What is more, QUIC moves congestion control to the application and the user space, enabling a rapid evolution for the protocol, as opposed to kernel space TCP. So it improve the performance of the protocol used in network. It can help to attract company to use it.

At Openwave Mobility, we have witnessed how these solutions are used to deliver the same amount of QUIC video with 20% less data. As a result, mobile operators can achieve reductions in the number of congested cells by 15%, facilitating fairness in the distribution of video bitrates (and therefore video quality) across subscribers sharing physical network resources.

After Google introduced and implemented the QUIC agreement in 2013, the IETF QUIC Working Group is now responsible for standardizing the QUIC agreement. The IETF community has shown great interest in the standardization of QUIC. A preliminary QUIC protocol version has been used in Google services and Chrome browser, which was deployed by some other developers. Some individual audio and video sites are also beginning to use QUIC protocol. HTTP2 which based on QUIC will also serve as a new Internet standard in the future.

5.4.2 Economical dimension

A detailed quantification of all the costs involved in the project has been done, both of material and human resources, as shown in previous sections of this document.

The other solution like to write a protocol from scratch will be less expensive than current solutions from an economical point of view, simply because the ability to rewrite a protocol is not necessary for there actually exit a protocol which I want to research. Hence, I think modifying an existing implementation will be cheaper, since it will not require a data plan to work.

5.4.3 Environmental dimension

First, this project is a research about a protocol. The main purpose of the QUIC protocol is to integrate the reliability of the TCP protocol and the speed and efficiency of the UDP protocol. So, it can save network resources. If this study is adopted by others, it may save much resources, which is good for the environment.

What is more, the execution of this project uses the minimum amount of resources possible, limited only to the electricity required for the equipment to work. This fact limits the search of alternatives to reduce the consumption and the environmental impact. This also makes the reuse of resources difficult.

Chapter 6

Mininet

We conduct our evaluation on a testbed, Mininet. There are several reasons why Mininet was used as testbed.

The functionality provided seems to be consider with the needs of the test case. Ability is to test real implementations and modify different network properties in a simulated network environment. The envisioned network speed, number of links, and number of processes are within the scope of Mininet's ability to handle.

It is easy to installation, each "host" does not need a separate "guest operating system" installation is also very helpful. Since Mininet's "host" can access the same directory structure, there is no need to distribute binary files or test files to virtual hosts. All of this makes it an easy-to-use test and development environment.

Mininet is implemented in Python and also offers a Python API to construct different environments and run tests. Using Python also makes it easy to implement automatic tests of many different test cases. Mininet also offers a way to run test manually after the network is constructed.

Tests can be run on a laptop without external hardware and to be able to easily move back and forth to a desktop computer is also helpful.

However, for testing non-application levels (such as different operating systems or kernel-mode drivers), Mininet may not be practical because all Mininet hosts run the same kernel.

Test system setup

The tests have been performed on a regular computer, running Ubuntu and utilizing Mininet. Appendix A contains the hardware specifications and exact software versions that have been used.

The tests of QUIC and TCP both requires similar test environments where different network properties and hosts where commands executed.

Mininet can create a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single

command, for example:

sudo mn -x --topo=minimal --link tc,bw=5,delay=50ms,loss=0

In this command, the bw is to set the bandwidth, delay is to set delay, loss is to set loss.

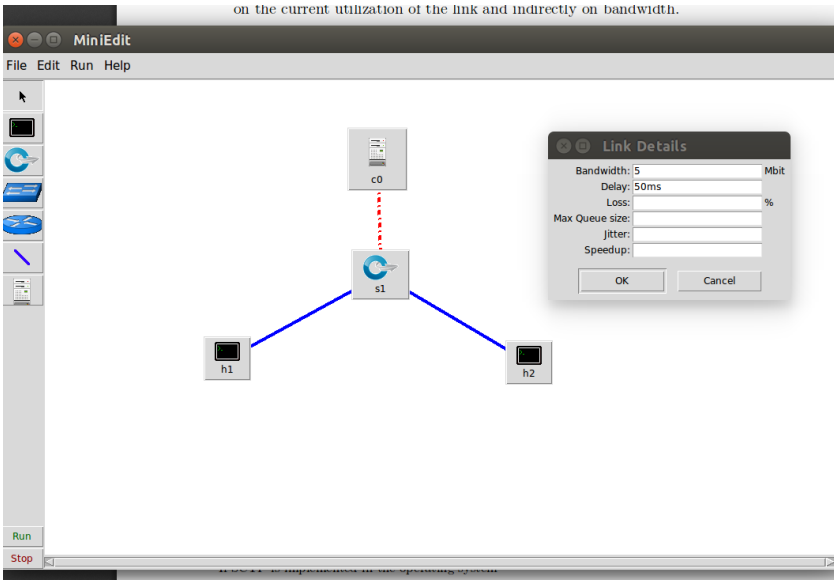


Figure 5: Utilized network layout in Mininet

Figure5 shows the actual network layout that has been used in the tests.
C0 is a controller, which designed to manage the physical network and virtual network architecture.

H2 is the server, h1 is the client, these hosts will be used for the main protocol under test. We will use tcpdump in the terminal of client to catch the packets of the flows.

The topology is only an example, we can create other topology to do the test. For example, we can add a server host,…

All network links can have different properties, we can set the parameters of link through the link details showed in the Figure.

Two different test systems (Test details can refer to Methodology) are been developed to run test cases. They are both very similar and use simple data structures to define a number of test-cases, which describes network properties, the commands to run and how to capture and extract results.

To capture the result of packets after passing the limited network link, the Tcpdump

will be used as we sayed in the Methodlogy. It will listen on the specific interface h1-eth0, the packets from server would be captured. The reason Tcpdump do not use in the server is that it cannot avoid the loss of the link.

6.1 Test system verification

Verification that a test system works as intended is of great importance and this have previously been done extensively on Mininet, for example in [23].

Since the combination of used hardware, operating system, software versions and test scripts also will influence the result; a few initial test verifications have been performed. These tests were also helpful in trying to better understand how the test system behaves and the influence of different parameters.

To verify delay and packet loss, ping packets were sent from client to server and the RTT time and loss will be captured showed in the mininet. So I use the ping command in this experiment. The value of table1 and table2 are all get through the ping command.

One of the test example is in Figure 6, Then modify the parameter and repeat the experiment.

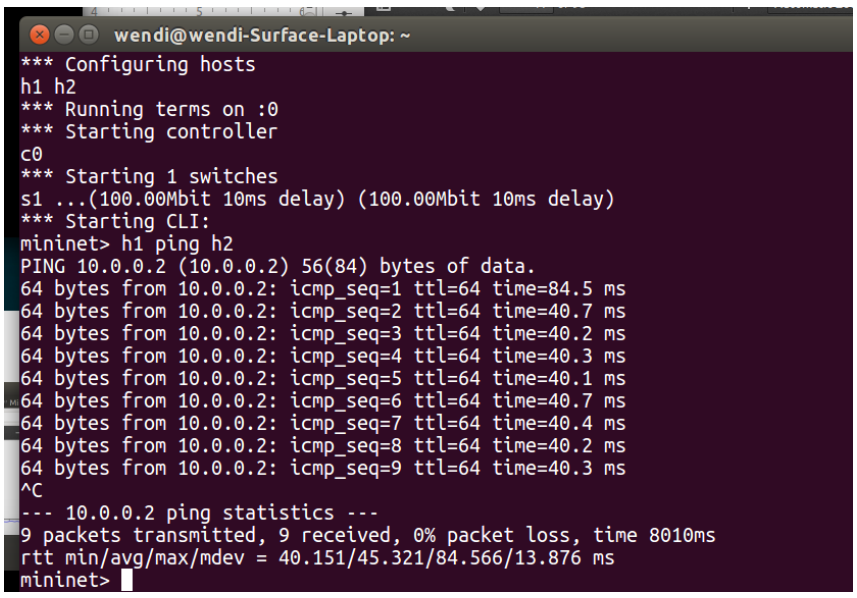


Figure 6 Ping example

6.1.1 Packet delay

Packet delay is specified as the minimum one-way delay. The total delay can be larger depending on current queuing delays.

Since the delay is applied in both directions in one link. In my test systems, there are two links between the client and server, one is between the client and switch and the other one is between the server and switch. In my topology, every links have been set the delay, so, totally, this round trip time should be close to 4*delay.

Target delay/ms	Average RTT/ms	Min RTT/ms	Max RTT/ms	Sent packets
0	0.034	0.026	0.048	20
5	20.591	20.089	23.896	20
10	40.667	40.071	40.879	20
25	100.764	100.081	101.072	20

Table5: Validation of delay

6.1.2 Packet loss

Packet loss is specified as the probability p that a packet is loss.

Target loss/ %	Actual loss/ %	Receive count	Sent packets
0.0	0.00	1500	1500
0.1	0.14	1498	1500
0.5	0.52	1493	1500
1.0	1.04	1484	1500
2.5	2.58	1461	1500

Table6: Validation of packet loss

The actual loss rate is close to the target loss, but for higher loss rates accuracy it is not very exact.

Chapter 7

Performance Comparation of QUIC and TCP

In this experiment, testing was performed to compare how different parameters of the link in the network affects the throughput for TCP and QUIC. The purpose of this experiment is to test the behavior of QUIC is same or better than TC through comparing how the protocols handle different network environments.

After setting a suitable test environment using mininet, to create a 5.5M file. Then, use the example network in the figure 7, Run the implementation of QUIC and record data. It is worth noting that the tcpdump need used before running the test of client.

One example of the information of testing trace:

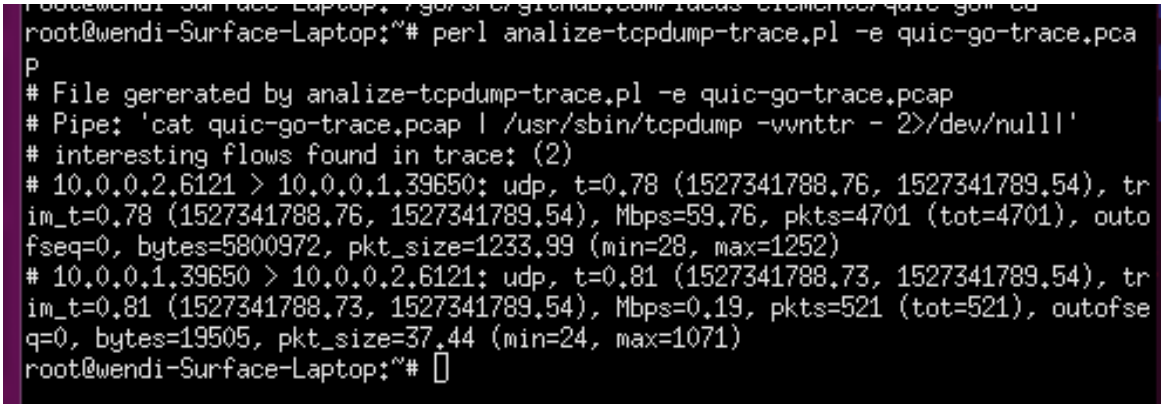


Figure 7 Evaluate of pcap

Each test case consists of a set of network parameters and the varying parameter's test interval. The protocols are then tested repeatedly while different results are captured, such as bandwidth and throughput.These results are later plotted to show how the varying parameter (on X-axis) yields different results (Y-axises).

To do the same tests but with TCP, the bytes transmitted was of the same size as the file used in the QUIC experiment. Traffic shaping was also made using the same with QUIC test.

The network I used is showed as Figure 5 in chapter 6.

Three different cases were tested and the results are shown below.

1) Varying bandwidth, 2-100Mbit/s bandwidths with 40ms RTT and 0% loss.

In this step I vary the bandwidths of all the links, we can see in the Figure 5,the network I use, the link between h1 and s1 and the link between the h2 and s1).

Bandwidth Mbit/s		100	70	50	30	20	10	5	2
Throughput Mbit/s	quic	53.76	45.77	37.82	25.35	17.61	9.26	4.76	1.92
	tcp	66.67	52.06	40.51	26.31	18.19	9.31	4.89	1.96

Figue 8 Bandwidth test 2-100 Mbit/s, 40 ms RTT and 0% loss

The bandwidth test in figure 8 shows that the curve of TCP Throughput is basically linear growth, QUIC can follow TCP until available bandwidth is 30 Mbit/s, after which throughput do not increase linearly with available bandwidth. QUIC ends at 53 Mbit/s and TCP at 67 Mbit/s.

The reason for QUIC not performing as well in the higher bandwidths may is that the 5.5 MB file completes very quickly and QUIC does not increase it's transfer speed as fast as TCP. A larger test file would have shown more equal QUIC throughput.

2) Varying network delay,80-800 ms RTT with 50Mbit/s bandwidth and 0% loss.

In this step, I vary the delay of all the link in the figure 5, the relation between network delay and RTT is explained in the chapter 6.

Delay ms		10	20	40	80	120	160	200
RTT ms		40	80	160	320	480	640	800
Throughput Mbit/s	quic	38.01	31.05	22.77	15.04	9.64	7.26	5.49
	tcp	40.98	33.29	23.54	14.03	8.99	6.78	5.02

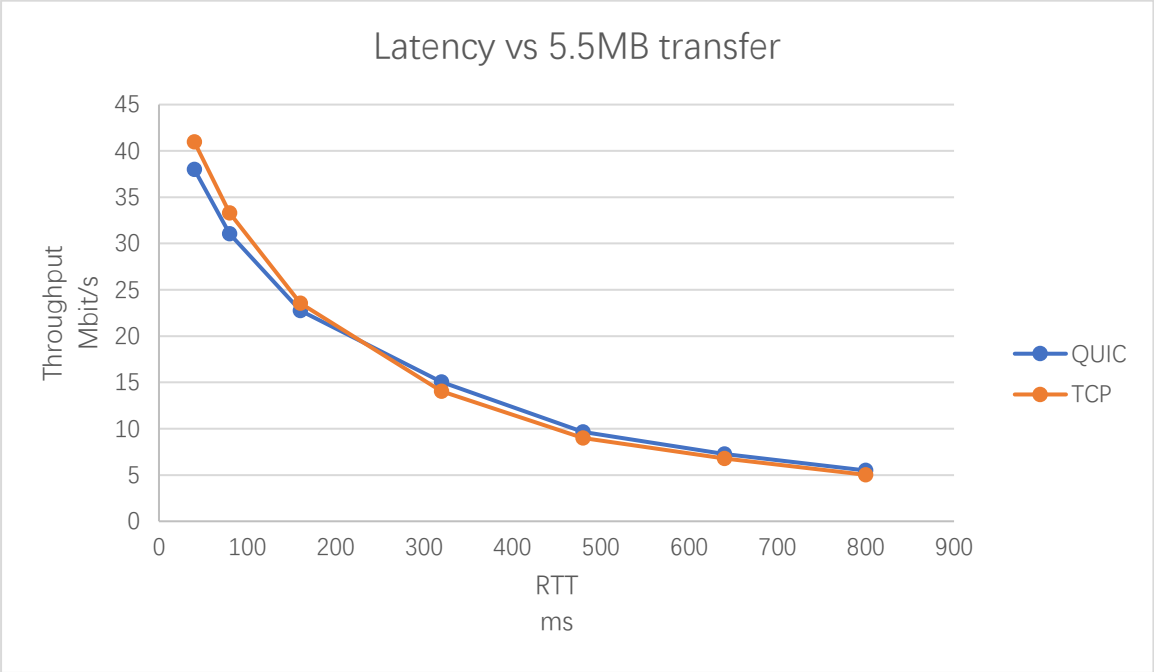


Figure 9 Latency test 80-800 ms RTT, 50 Mbit/s bandwidth and 0% loss

The latency test shows that QUIC and TCP are both affected well by the increase of RTT, but QUIC is less than TCP.

3) Varying packet loss probability, 0-5% loss with 50Mbit/s bandwidth and 40 ms RTT.

In this step, I vary the packet loss of each link.

Loss %		0	0.05	0.075	0.1	0.5
Throughput Mbit/s	quic	38.13	34.46	26.78	16.07	7.41
	TCP	41.26	34.5	23.74	9.18	2.37

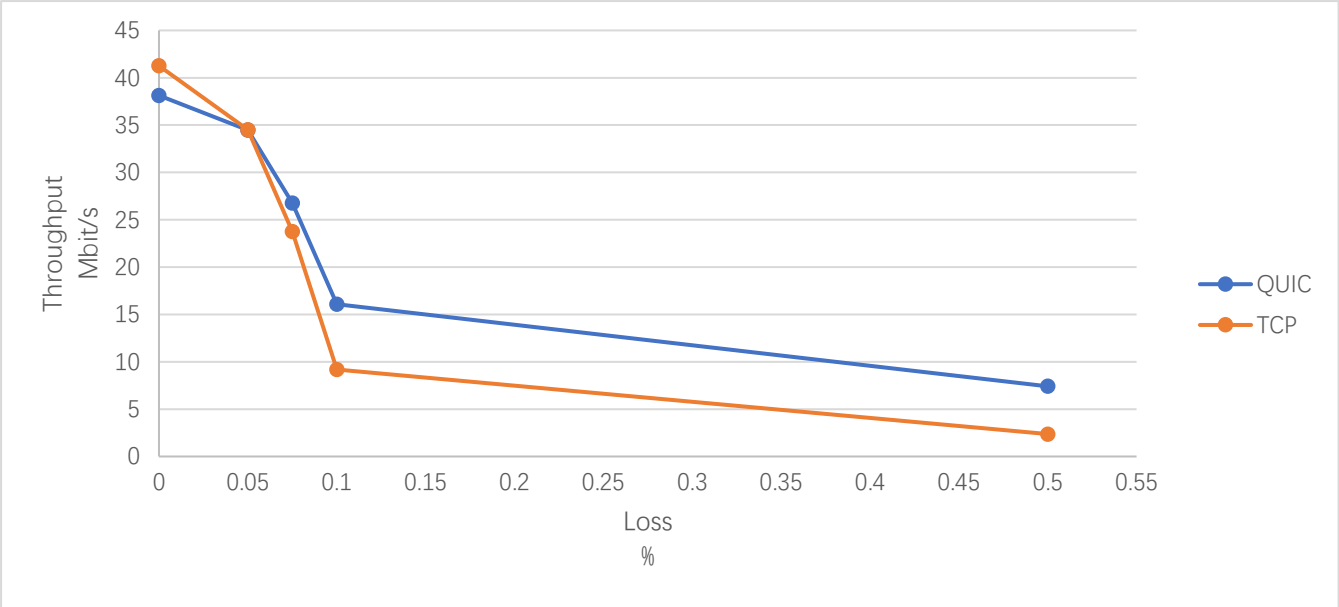


Figure 10 Packet loss test 0-0.5%, 50Mbit/s bandwidth and 40 ms RTT

And also, when the loss is 0.5%, the time to download the file is 34.56s, however, when the loss is 0.1%, the time is 2.89s, it changes well.

In the figure 10, the throughput for both the protocols did not vary so much for packet loss rates between 0%-0.05%. But there was a huge drop for both protocols after the packet loss rate increased to 0.05%, where QUIC ended on throughputs a little higher than TCP.

The QUIC ability to handle packet loss did not seem significantly better than TCP. Therefore, the only enabled method QUIC has for recovering lost packets is by retransmitting them, which is the same method for error recovery that TCP uses.

Chapter 8

Fairness tests

8.1 Fairness

QUIC's basic fairness handling against TCP was tested by running large transfers to run

the client and server. The test procedure is same as the to the steps as above. But the most important thing which is different from to test the performance comparation between QUIC and TCP is that the QUIC and TCP need to start to running simultaneously. The bash script I have write in the Methodology to achieve it. After running the server, I just need to run this bash script in the host of client, then test it.

An essential property of transport-layer protocols is that they do not consume more than their fair share of bottleneck bandwidth resources. Absent this property, an unfair protocol may cause performance degradation for competing flows. We evaluated whether this is the case for the following scenarios, and present aggregate results. We expect that QUIC and TCP should be relatively fair to each other because they both use the Cubic congestion control protocol. However, we find this is not the case at all.[24]

8.2 QUIC vs QUIC

Figure shows two QUIC flows are fair to each other. We also found similar behavior for two TCP flows. Although their throughput will

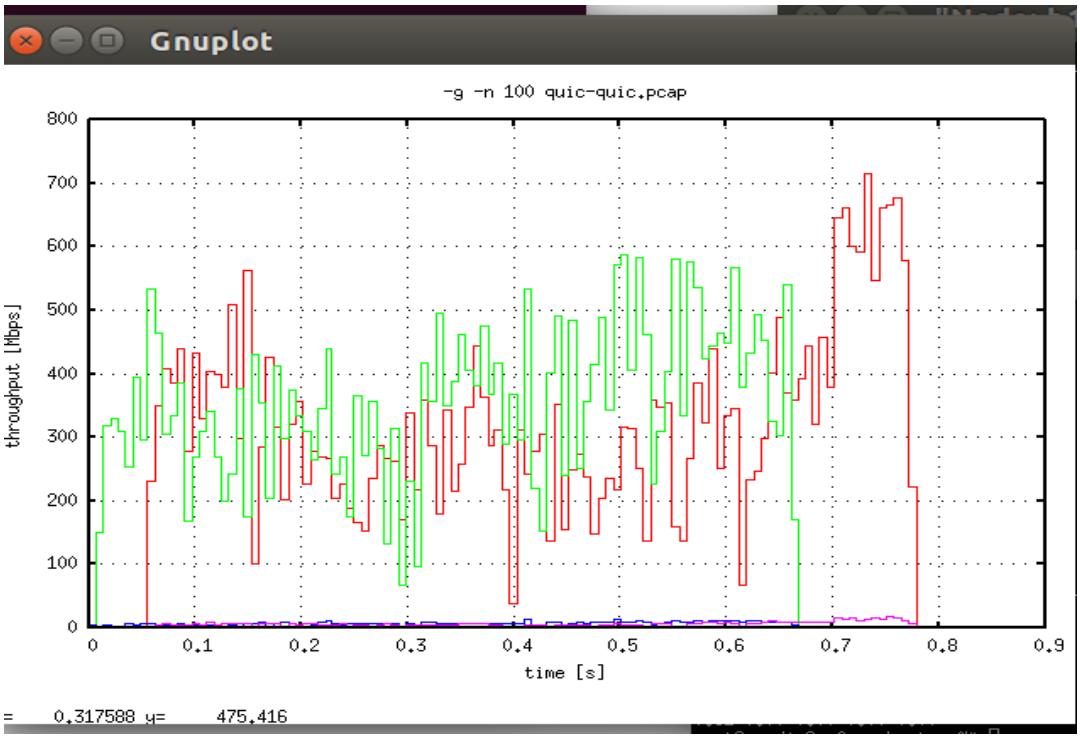


Figure 11 QUIC vs QUIC

8.3 QUIC vs TCP

When we first test the fairness between QUIC and TCP using the simple network, we find they are not fair. So, in order to know whether different scenes will have an impact on the fairness we have designed some other test to research it.

Offload effect on fairness

Offload designed to take processing of the network such as packet segmentation and reassembly processing tasks is a technology that increases the throughput of high-bandwidth network connections by reducing CPU overhead. This technique is applied to TCP. This has the effect of reducing the workload on the host CPU and moving it to the NIC(network interface card), allowing both the host to perform quicker and also speed up the processing of network traffic.
If offloading was turned on, we offloaded all TCP connections on a supported network interface, regardless of whether it would benefit or not. In this case, we test if the offload will have effect on the fairness.

The command to disable TCP offloading in both h1 and h2
disable TCP offloading in both h1 and h2
h1 # ethtool -K h1-eth0 tso off

h1 # ethtool -K h1-eth0 gso off
h2 # ethtool -K h2-eth0 tso off
h2 # ethtool -K h2-eth0 gso off

We totally have four different scenes but with the same parameters of network link, we set the delay to 10ms and the bandwidth to 10Mbit/s.

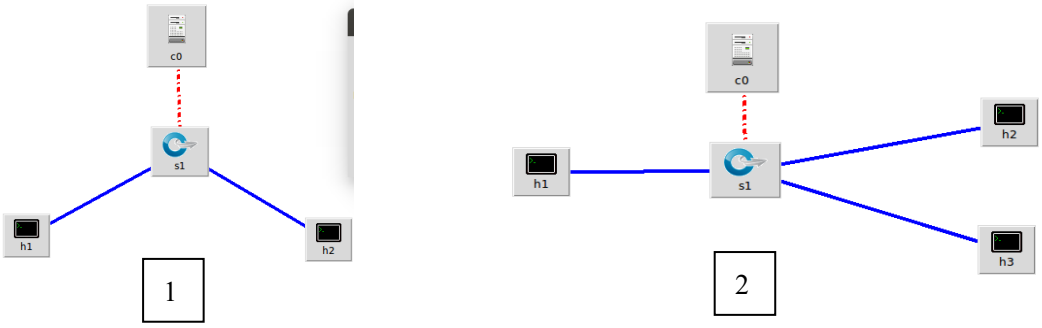


Figure 12 Examples of topology

Test 1, we use the first topology of network in the figure, which means we use the same server and client. Using the 20MB test file for downloading. Then we test with TCP offloading.

Test 2, we did not change the topology but test the fairness without TCP offload. Using 20MB file.

Test 3, we change the topology which has two connections and three host, h1 is the client, h2,h3 is the protocols server respectively. One connection is to run the QUIC, and the other one is to run TCP. Test the fairness with TCP offload.

Test 4, we use the same network environment with first test. But we change the size of file to 5 MB to see if the fairness will improve when downloading a small file.

8.3.1 Results

Test 1

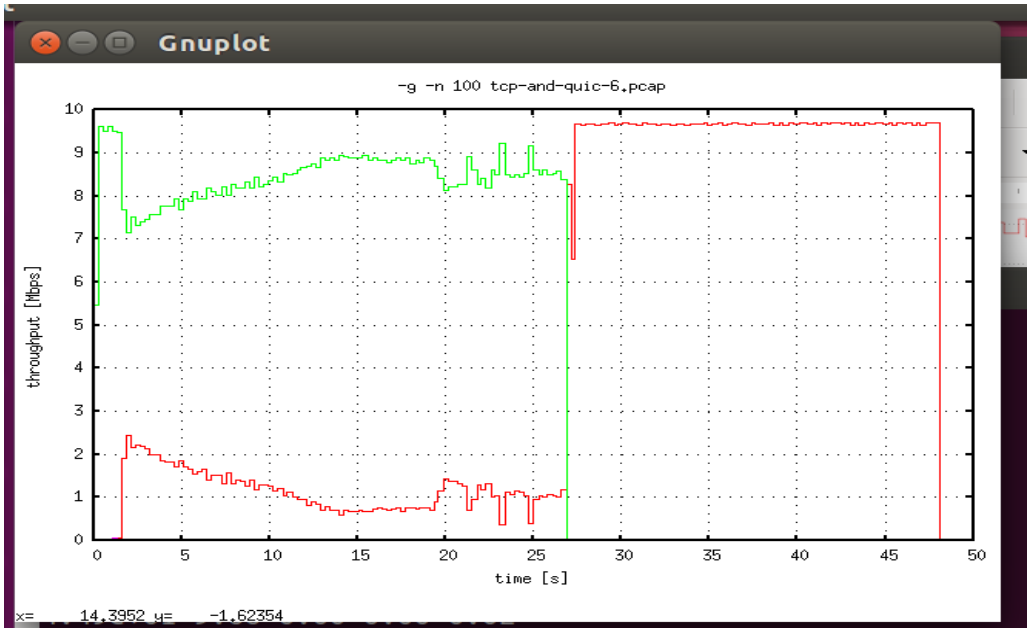


Figure13 One connection between client and server; with offloading; 20MB file

Test 2 Offloading test

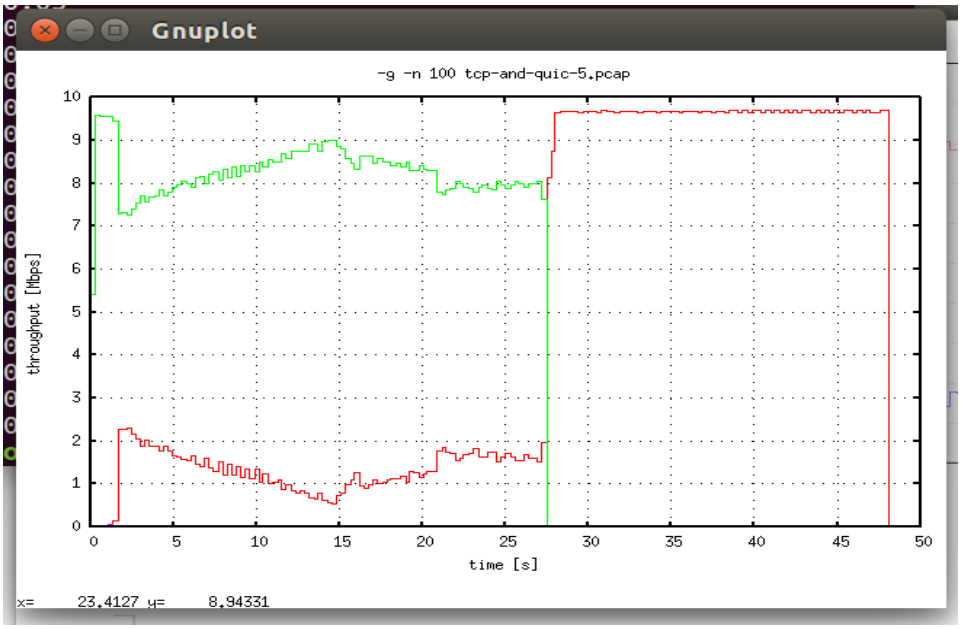


Figure14 One connection between client and server; without offloading; 20MB file

Test 3 network test

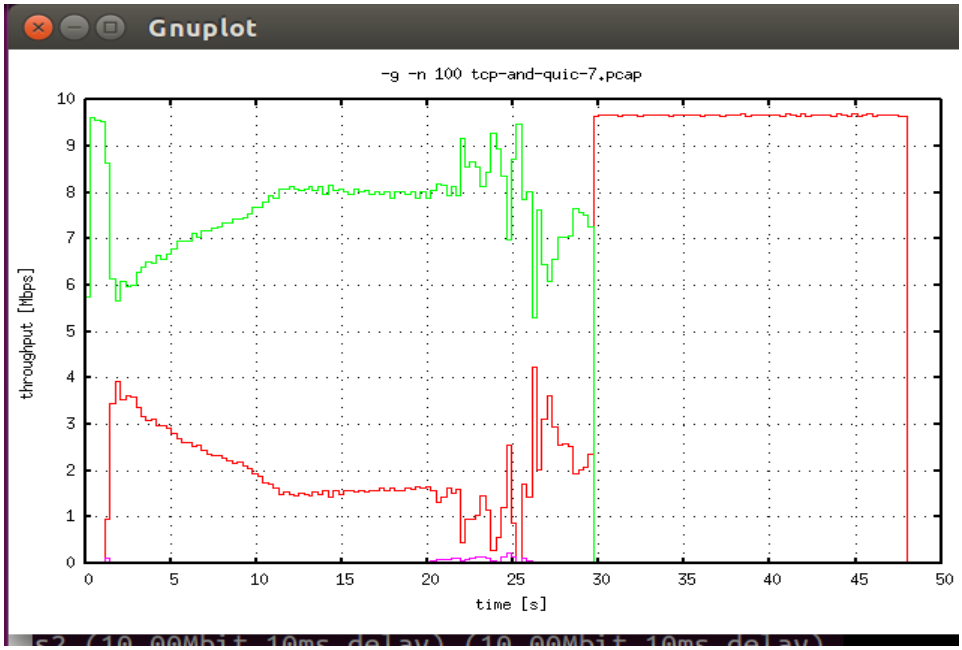


Figure 15 Two connection between client and server; with offloading; 20MB file

Test 4 Dowloading test

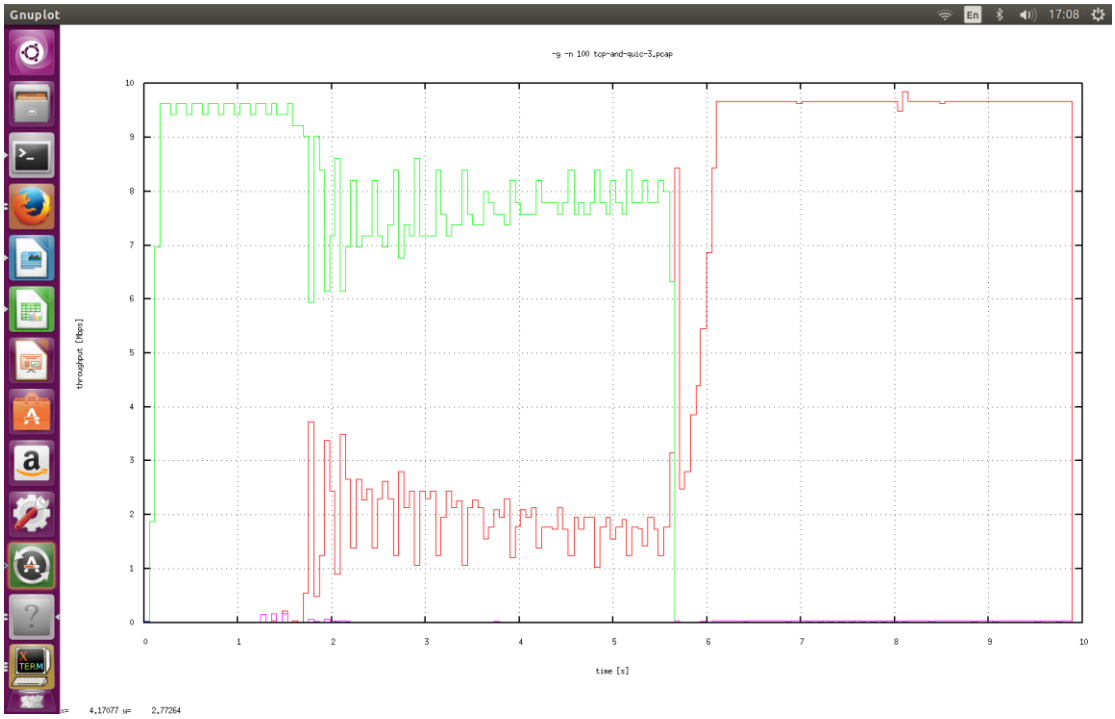


Figure 16 One connection between client and server; without offloading; 5MB file

The results are showed in the figures above. The green one is TCP, the red one is QUIC.

The result is as showed that QUIC is unfair to TCP as predicted. And the TCP is more aggressive.

First, we can see in the Figure13 Figure 14 that, TCP offloading has a little influence on the fairness. TCP offloading has been known to cause some issues, and disabling it can help avoid these issues. For example, TCP offloading will improve the performance of TCP, so, making the offloading will decrease the throughput of TCP. In this case, the TCP will use a small amount of bandwidth and the TCP and QUIC will be more fair.

Then, I find when I run the QUIC and TCP in different connections, there is a significant increase of the throughput of TCP. They are more fair with each other. It can be showed that when using the QUIC and TCP in one same connection, they will occupied with each other's bandwidth, so they will not fair as when they implement into different connection.

Finally, I find that when the host of client download a small file, they are more fair, becaue they have smaller objective size to allow TCP and QUIC to fairly share available bandwidth.

We further investigate why QUIC is unfair to TCP by instrumenting the QUIC source code, we can use the tool tcpdump to investigate the packets, which can be shown below. When looking at pcap file, in the sample of fraction of tcpdump, we can see that the pks can represent the congestion window sizes. To extract the average congestion window sizes, the average congestion window size of TCP and QUIC is 23406, 10283,represently.

When competing with QUIC, TCP is able to achieve a larger congestion window. Taking a closer look at the congestion window changes, we find that while both protocols use Cubic congestion control scheme, TCP increases its window more aggressively (both in terms of slope, and in terms of more frequent window size increases). As a result, TCP is able to grab available bandwidth faster than QUIC does, leaving QUIC unable to acquire its fair share of the bandwidth.

Sample of fraction of tcpdump:

wendi@wendi-Surface-Laptop:~$ perl analize-tcpdump-trace.pl -e tcp-and-quic-6.pcap
File gererated by analize-tcpdump-trace.pl -e tcp-and-quic-6.pcap
Pipe: 'cat tcp-and-quic-6.pcap | /usr/sbin/tcpdump -vvnttr - 2>/dev/null|'
interesting flows found in trace: (4)
10.0.0.2.6121 > 10.0.0.1.33462: udp, t=46.64 (1528019396.93, 1528019443.57), trim_t=46.64 (1528019396.93, 1528019443.57), Mbps=4.97, pkts=23406 (tot=23406), outofseq=956, bytes=28999972, pkt_size=1239.00 (min=28, max=1252)
10.0.0.2.8000 > 10.0.0.1.57852: tcp, t=27.04 (1528019395.68, 1528019422.72), trim_t=27.04 (1528019395.68, 1528019422.72), Mbps=8.38, pkts=10283 (tot=10283), outofseq=1, bytes=28331410, pkt_size=2755.17 (min=0, max=2896)
10.0.0.1.57852 > 10.0.0.2.8000: tcp, t=26.88 (1528019395.64, 1528019422.53), trim_t=26.88 (1528019395.64, 1528019422.53), Mbps=0.00, pkts=9640 (tot=9640), outofseq=0, bytes=140, pkt_size=0.01 (min=0, max=140)
10.0.0.1.33462 > 10.0.0.2.6121: udp, t=46.81 (1528019396.84, 1528019443.66), trim_t=46.81 (1528019396.84, 1528019443.66), Mbps=0.02, pkts=2950 (tot=2950), outofseq=629, bytes=94812, pkt_size=32.14 (min=24, max=1071)

Chapter 9

Congestion control tuning parameters

This part of the experiment is to study the influence of the parameters in the congestion control algorithm on the performance of the protocol, because the congestion control mainly affects the congestion windows, so I modified some command lines in the source code of the algorithm and can automatically generate the relationship between cwnd and time. Take the research method of control variables, keep other parameters unchanged, change one parameter, and conduct experiments.

Based on our study of the Cubic protocol in Chapter 2, the two parameters that control the behavior of the algorithm are C and β. Let's take a look at the beta and C in the curve that affect the characteristics of QUIC.

Pseudo code for the main functionality of the Cubic algorithm is shown in Figure17. The features of this algorithm can be summarized as follows,

Backoff factor 0.7. On packet loss, cwnd is decreased by a factor of 0.7 (compared with a factor of 0.5 in the standard TCP algorithm).

Cubefactor is a factor that can influence the performance.

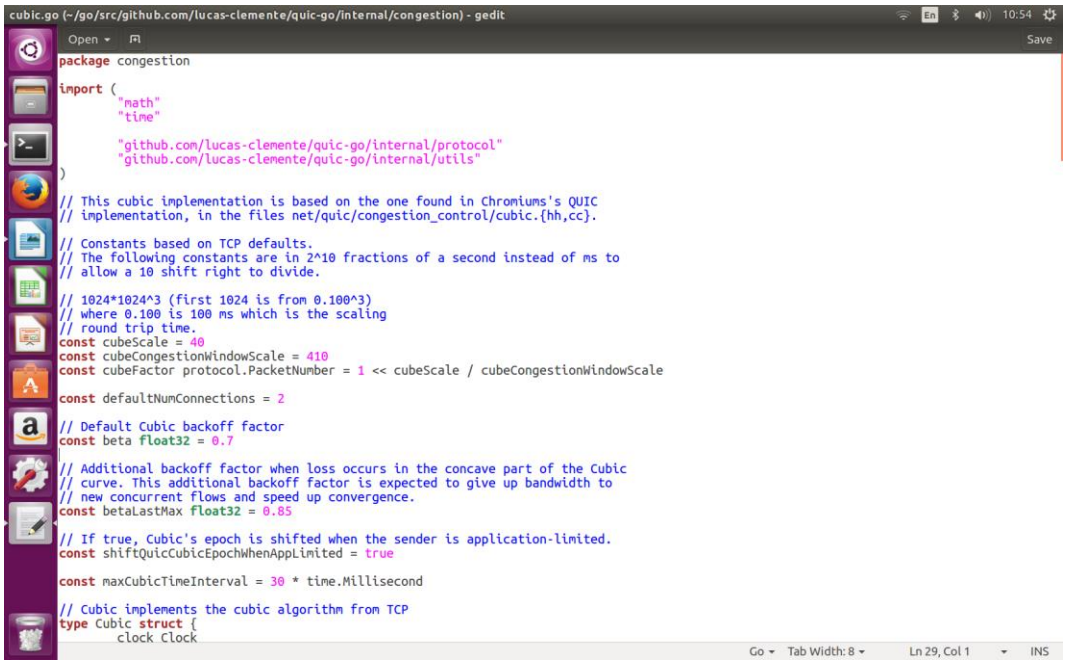


Figure 17 Cubic.go file

I adjust the CubeFactor and backoff factor separately, and observe the congestion window over time. First I set the parameters of network link: The bandwidth is 10 Mbit/s; RTT is 40 ms; Packet loss is 0.05%

The initial data is : CubeFactor: 1 backoff factor: 0.7

Parameters	Values tested
CubeFactor	0,1,2
Backoff factor	0.5 0.7 1

Results

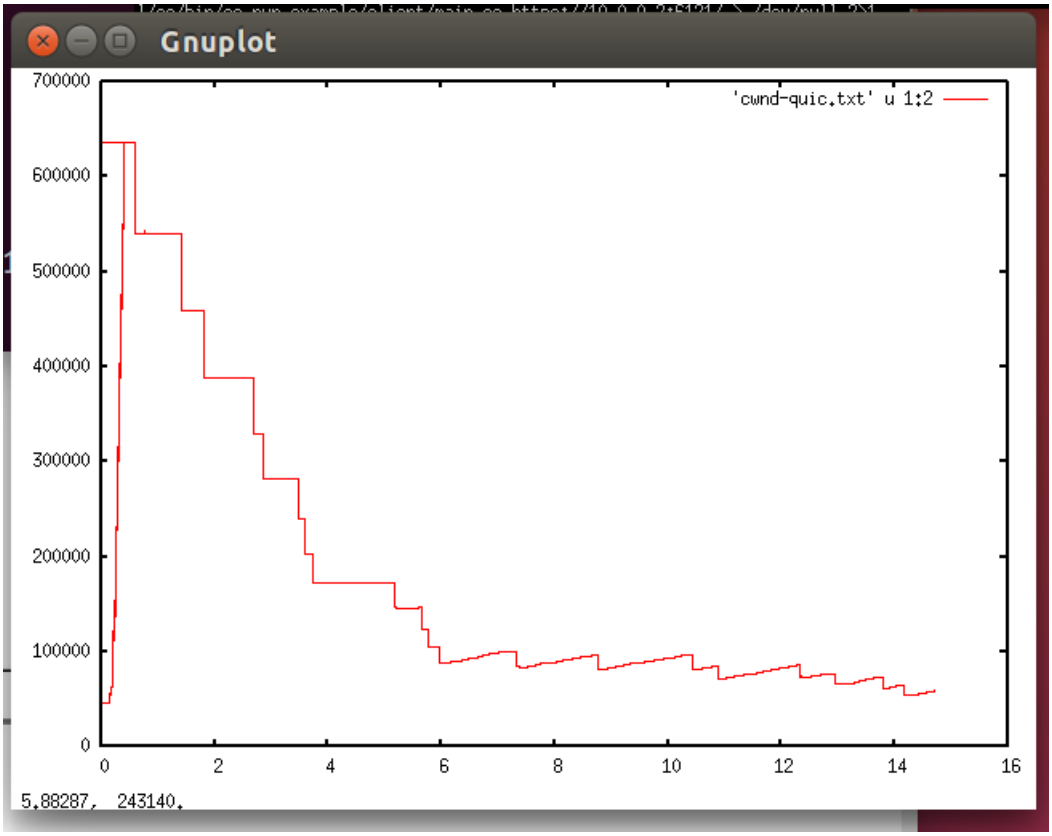


Figure 18 CubeFactor 1; Backoff factor 0.7

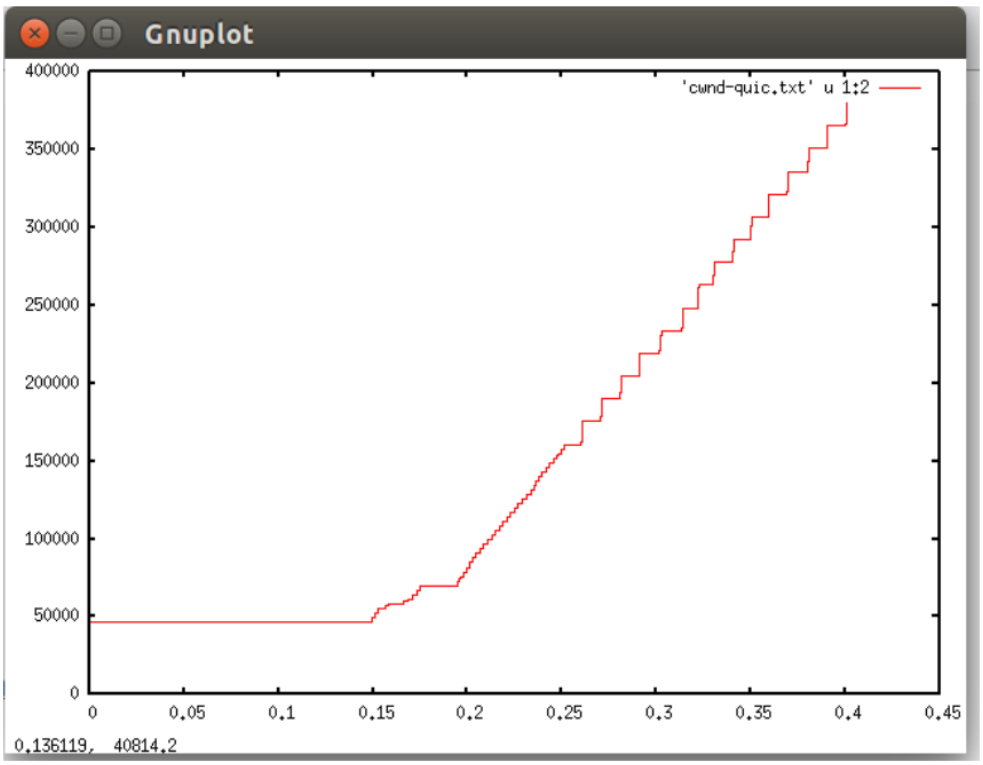


Figure 19 CubeFactor 1 Backoff Factor 1

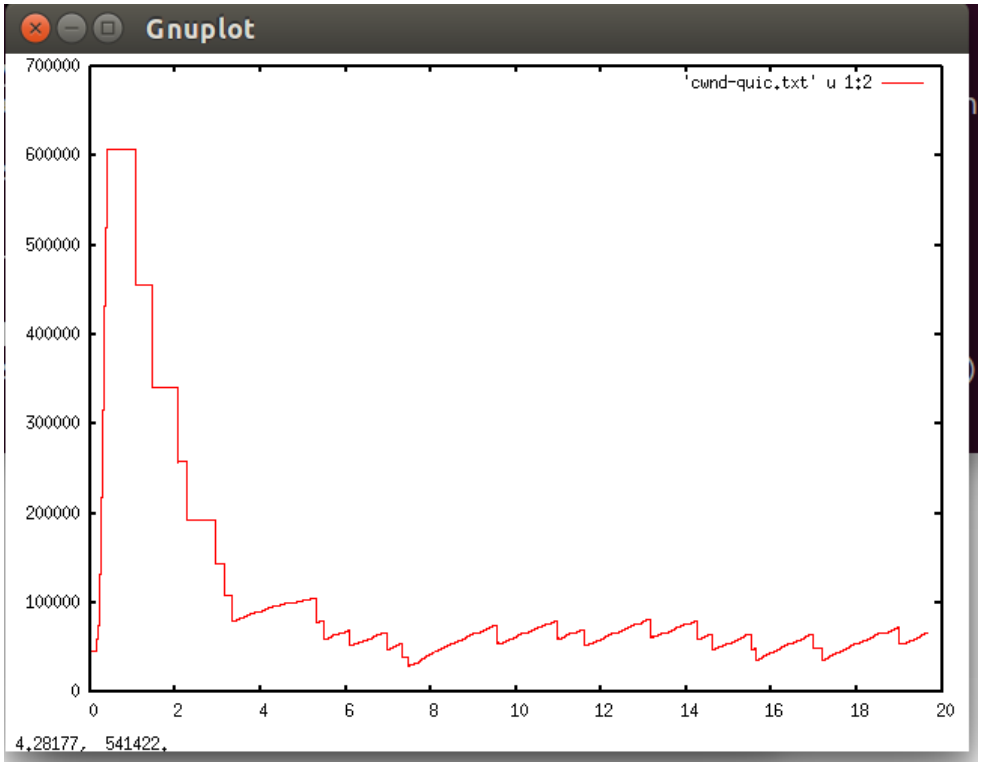


Figure 20 CubeFactor 1 Backoff Factor 0.5

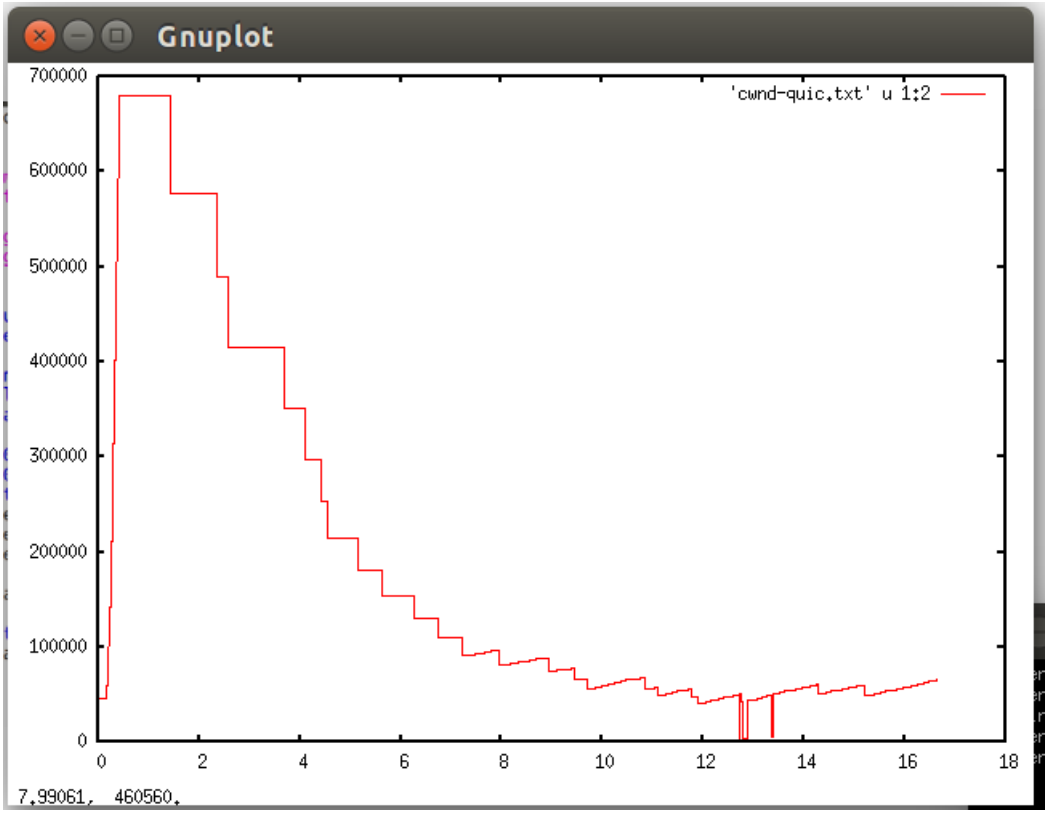


Figure 21 CubeFactor 2 Backoff Factor 0.7

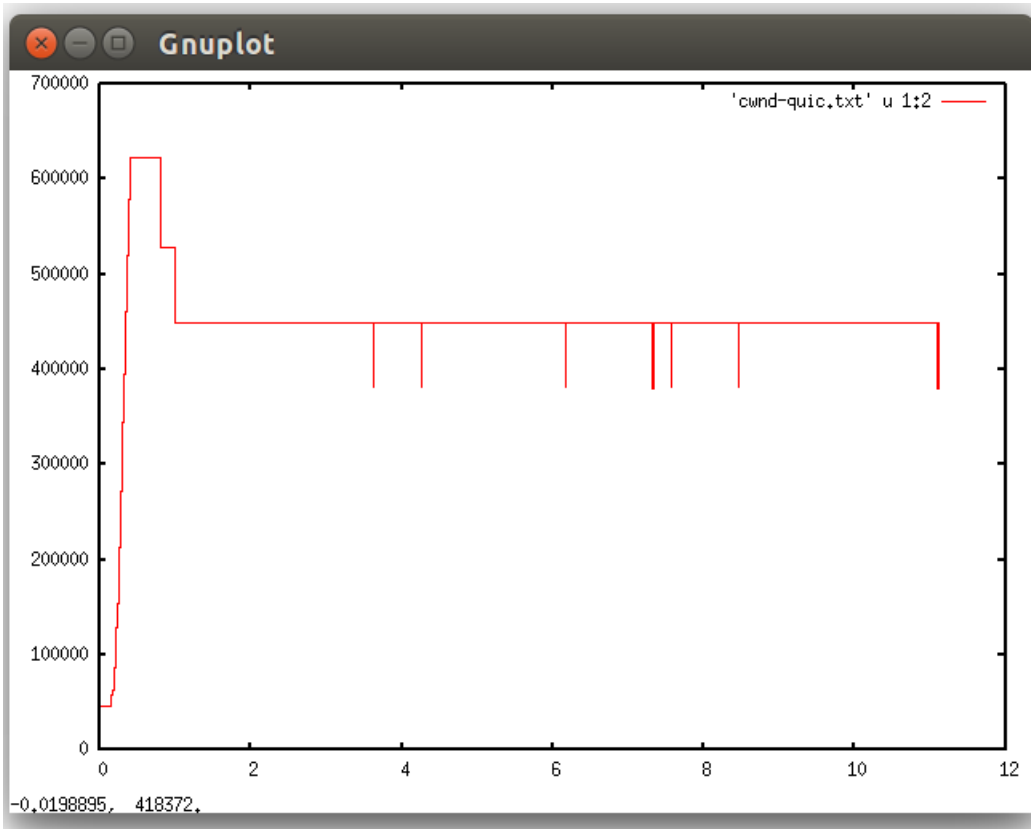


Figure 22 CubeFactor 0 Backoff Factor 0.5

Analysis

When a packet loss occurs, CUBIC reduces its window size by a factor beta, when it control the windows size, the new window size = beta* W max,in that way, seeing the Figure 18, 19, 20,we will find when the backoff factor is more small, the time is more long. And when the backoff factor is 1, we can see that the window size will not decrease.

Seeing the Figure 18, 21, 22,we found the Cube Factor is biger, the curve will be more fluent, so the network will be more stable.

In conclusion, Beta Controls its convergence rate. C Controls TCP friendiness

Chapter 10

Obstacle and solutions

10.1 Source code

When I download quic and implement the test files on the terminal of server and client in Ubuntu. The server and the client can establish a connection. And also, the server can successfully monitor the client, receive the user's request, and perform corresponding operations and return corresponding data.

However, when I implement the quic following the same steps as before on the emulated network created by the mininet and start the server and client, they can not connect.

Because the client tries to connect to the localhost address (127.0.0.1). Actually, I should use the IP address of the server. It is needed to execute ifconfig in the server to figure out what address it has. It is showed the address of server is 10.0.0.2. It is need to modify the code of the run files of server. Then this problem will be solved. The steps to modify it can be shown as follow.

After running the server it can been see the UDP socket:

```
netstat -nau
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 127.0.0.1:6121          0.0.0.0:*
```

So I changed the following in the server: example/main.go

```
// bs = binds{"localhost:6121"}
bs = binds{"0.0.0.0:6121"}
```

and execute the server again.

```
netstat -nau
Active Internet connections (servers and established)
```

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 0.0.0.0:6121            0.0.0.0:*
```

10.2 Fairness test

Although I have start the QUIC and TCP at the same time but as the figure23 shows, they don't have the interval of time to run together, that is because, although the client starts to connect the server through different protocol simultaneously, but the server need a while to send the package to the client through the QUIC protocol. However the flow using TCP does not have the buffer time. So, it can be seen that after the TCP finish the process have not the QUIC start.

In order to solve the problem, we can use a bigger test file or decrease the bandwidth of the link in order to increase the time to download the file.

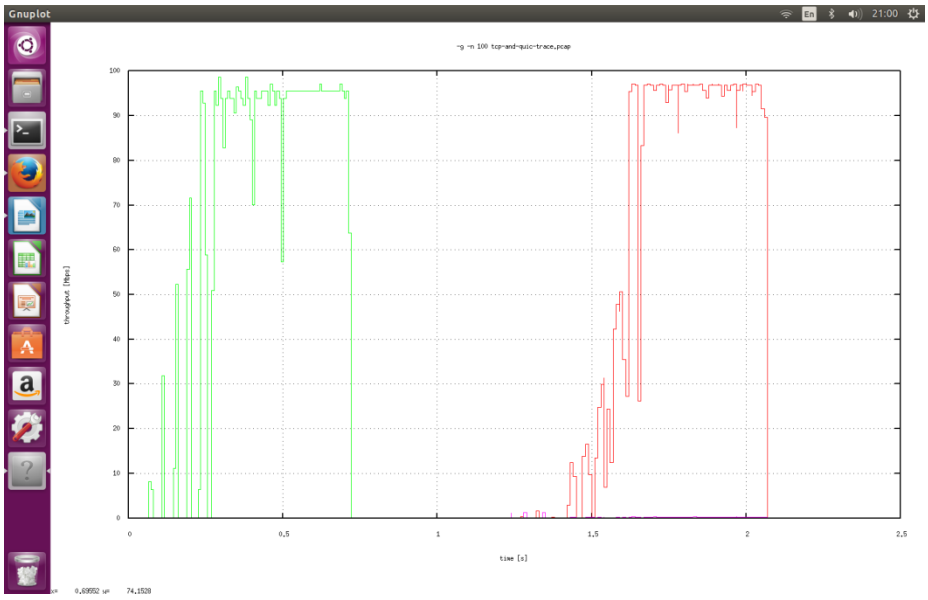


Figure 23

Chapter 11

Conclusion

The main contributions of this thesis are the tests of QUIC and TCP implementations of their network performance and congestions handling. We used various metrics such as fairness and stability of throughput to evaluate the QUIC. Further, the thesis give a parameter analysis of congestion control. The work has shown how Mininet can be used to build systems to test implementations of QUIC for multiple scenarios.

QUIC can avoid a number of limitations with TCP, such as: head-of-line blocking and the dependency of the TCP version in the operating system. Other new features are briefly mentioned and not described in much detail in this thesis.

The performance in low delay without packet loss is very similar to TCP, except for higher bandwidth which is mostly due to the slower CWND growth rate. The tests confirm that QUIC does work well when there's a small probability of packet loss, which is what QUIC is designed for. To cover larger packet loss would result in even more overhead, but packet loss must also affect the congestion control even if the packets can be recovered. QUIC outperforms TCP in most cases.

The congestion control is very important in QUIC which the UDP not have. I present an initial experimental evaluation of the Cubic-TCP algorithm. Chapter 7 give an intuitive test showing that how Beta and C influence the performance of QUIC.

The two QUIC flows are sharing a good fairness, but in the case of QUIC and TCP using same connection have the low fairness. It suggests that TCP obtains performance at the expense of QUIC. We made further studies to study what the network environments will affect fairness.

When testing in isolated and emulated environments, the results should always be taken with a bit of skepticisms since real networks are influenced by many other factors and simultaneous traffic. To further verify the results of these tests, comparison with tests on real network are necessary.

More analysis is needed to understand exactly when and why, to properly adjust the algorithm or parameters to be more TCP-friendly. A thorough evaluation needs to look

at a variety of testing scenarios to make a valid observation about the behavior of a protocol. But we believe that the steps we took are at least steps toward the right evaluation of these protocols and hope that our work improves the methodology in evaluting various congestion control protocols.

References

[1] B.M.Leiner,V.G.Cerf,D.D.Clark,et al.,"A brief history of the Internet", ACM SIGCOMM Computer Communication Review, vol. 39, no. 5, pp. 22–31, 2009. [Online]. Available: http://www.cs.ucsb.edu/~almeroth/classes/ F10.176A/papers/internet-history-09.pdf.

[2] M. Tuexen and R. Stewart, UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication, RFC 6951 (Proposed Standard), Internet Engineering Task Force, May 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6951.txt.

[3] QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02.

[4]Van Jacobson, Michael J. Karels. Congestion Avoidance and Control (1988). *Proceedings of the Sigcomm '88 Symposium*, vol.18(4): pp.314–329. Stanford, CA. August 1988. This paper originated many of the congestion avoidance algorithms used in TCP/IP.

[5] "Performance Analysis of TCP Congestion Control Algorithms" . Retrieved 26 March 2012.

[6] B. Lantz and B. Heller. Mininet - an instant virtual network on your laptop (or other pc), [Online]. Available: http://www.mininet.org.

[7] I.Swett. QUIC Deployment Experience @ Google. https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf, 2016.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting congestion control for consistent high performance. InProc. of USENIX NSDI,2015.

[9] P.Megyesi, Z.Krämer, and S.Molnár. How quick is QUIC? InProc. of ICC, May2016

[10] A.Vernersson. Analysis of UDP-based reliable transport using network emulation. Master's thesis, Luleå University of Technology, 2015.

[11] J. Roskind. Google QUIC design specification, [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFVev2jRFUoVD34.

[12] Google Chromium. QUIC wire layout specification, Available: https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHet AXV8I0fQe-B_U.

[13] C. Cimpanu. Google Creates New Algorithm for Handling TCP Traffic Congestion Control. http://news.softpedia.com/news/ google-creates-new-algorithm-for-handling-tcp-traffic-congestion-control-508398.shtml, September2016.

[14] S. Floyd, "High Speed TCP for Large Congestion Windows," RFC 3649, December 2003

[15]B. Ford, "Structured streams: a new transport abstraction", ACM SIGCOMM Computer Communication Review, vol. 37, pp. 361–372, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1282421

[16] Y. Gu. UDP-based data transfer protocol, [Online]. Available: http://udt.sourceforge.net.

[17] Github https://github.com/

[18] https://github.com/quicwg/base-drafts

[19] B. Heller, "Reproducible network research with high-fidelity emulation",

PhD thesis, Stanford University, 2013. [Online]. Available: https://stacks.stanford.edu/file/druid:zk853sv3422/heller_thesis-augmented.pdf.

[20] A.Langley,A.Riddoch,A.Wilk,A.Vicente,C.Krasic,D.Zhang,F.Yang,F.Kouranov,I.Swett, J.Iyengar,J.Bailey,J.Dorfman,J.Kulik,J.Roskind,P.Westin,R.Tenneti,R.Shade,R.Hamilton, V.Vasiliev,W.-T.Chang, and Z.Shi. The QUIC transport protocol : Design and Internet-scale deployment. InProc. of ACM SIGCOMM,2017.

[21] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: A New TCP-friendly High-Speed TCP Variant. ACM SIGOPS Operating Systems Review (2008).

[23] A. Jurgelionis, J. Laulajainen, M. Hirvonen, et al., "An empirical study of netem network emulation functionalities", in Computer Communications and Networks (ICCCN),2011 Proceedings of 20th International Conferenceon,IEEE,2011,pp.1– 6.

[24] P. Karn and W. Simpson, Photuris: Session-Key Management Protocol, RFC 2522 (Experimental), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2522.txt.

[25] M.Honda,F.Huici,C.Raiciu,J.Araújo,andL.Rizzo.2014. Rekindlingnetwork protocol innovation with user-level stacks. ACM Computer Communication Review (2014).

[26] C. Paasch. 2016. Network Support for TCP Fast Open. (2016). Presentation at NANOG 67, https://www.nanog.org/sites/default/files/Paasch_Network_Support. pdf.

[27]Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance," Proceedings of IEEE INFOCOM 2004, March 2004

[28] Chrome https://www.google.cn/chrome/

[29] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. 2008. RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State. Internet Engineering Task Force (IETF) (2008).

[30]S. R. Das. Evaluation of QUIC on web page performance. Master's thesis, Massachusetts InstituteofTechnology,2014.

[31] M.Allman,V.Paxson,andE.Blanton,TCP Congestion Control,RFC5681(Draft Standard), Internet Engineering Task Force, Sep. 2009. [Online]. Available: http: //www.ietf.org/rfc/rfc5681.txt.

Appendix A

Hardware:

Microsoft Surface

Host system

UPC Intel(R) Core(TM) i5-7300U 2.71GHz
RAM 8.00GB

Software versions:

• Ubuntu 14.04.1 LTS, 64-bit
• Linux kernel 3.13.0-43-generic Ubuntu SMP, CFS scheduler default, IPv6 disabled.
• Mininet 2.1.0 (Ubuntu packaged)
• Perl-6
Gnuplot-5.2.3

Tested UDP-protocol versions:

• QUIC - Chromium git checkout 2014-10-28, release compiled.