A Data Analysis on the Performance of QUIC on YouTube

Saige Liu

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

Sep 21st, 2020

Abstract

This research paper analyzes the YouTube video performances on QUIC comparing to TCP in a data science perspective; concurrently studies the behavior patterns of QUIC.

The research deployed repetitive tests of YouTube videos running on QUIC and TCP, used disparate tools to record network activities, implemented customized parser programs, and analyzed the log files. The critical evaluation criteria were the download speeds and packet transmission approaches. The research, thus based on the statistical and visual results, gave comprehensive comparisons and evaluation.

Research results show an exciting outcome that TCP is faster than QUIC for downloading video chucks. On the contrary, QUIC tends to be more stable, consistent, and error-resilient. QUIC supports multiplexing and transmits more steadily. It responds to network fluctuations faster in real-world network environments.

Thus, this study shows that QUIC's schema is more outstanding than TCP, but TCP is more rapid than QUIC. QUIC is still new and evolving; its efficiency and popularity could be further improved. Only when IETF officially releases a faster and more scalable version of QUIC can industrialize on a large scale.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Saige Liu

Signature: Saige Liu

Acknowledgements

I am deeply appreciative of supervisor Dr. Colin Perkins. He used his rich knowledge leading me to complete this project and provided patient guidance, critical advice, and thoughtful explanations. I immensely appreciated him for putting forward excellent research questions, explaining the research process's critical points, and following up on the project's progress patiently. Without his professional guidance and advice, there would not be this plentiful research.

I also want to thank the developers who created the python library. Their open-source libraries helped the software development. Thanks to the developers of dpkt. Also, thanks to Justin Crown that created haralyzer and helped me solve issues on GitHub. Thanks to Robin Marx for hosting the online Qvis website and providing a version update on my feedbacks.

I am grateful for my parents and friends for their help and support during the year. Without their support, I would not survive this difficult period and concentrate on project writing.

Contents

Chapter 1 Introduction

While using the internet, the communication protocol positively affects the network's functionality and performance. The protocol competition between UDP and TCP never ends. However, a new strong opponent appears; it is called Quick UDP Internet Connection (QUIC). TCP is the fundamental of the original worldwide web, QUIC is a promising next-generation newcomer. QUIC designs to be more advanced than TCP. This project is going to study QUIC, also comparing TCP and QUIC. It is an essential issue since the utilization of QUIC on the World wide web may bring considerable changes to the surfing experience.

Google had deployed QUIC on Chrome in experimental mode; Google searches and YouTube video streaming can set under QUIC protocol. Thus, YouTube video streaming is an excellent experimental example for studying network transmissions. More precisely, the problem in this project is whether QUIC performance is better on YouTube than TCP.

This research studies the performance of QUIC and TCP on YouTube video and makes a proper conclusion. QUIC is better than TCP in consistency and stability, but TCP is generally faster than QUIC. The research methodologies contain capturing the network activities, creating log file parser tools, and analyzing the results. The results and conclusion are notable. Since real-world datasets handling is complicated, the customized parser tool loads and filters the dataset adequately to demonstrate a reasonable result. The behavior of QUIC in the real-world is thus precisely studied. The experiments also contain repetitive tests, ensure that the results are not fortuitous. The results give a comprehensive assessment of the performance of QUIC.

This research is different from other publications since other papers built their own servers and used quic-go and aioquic as the QUIC protocol. In contrast, this project is testing QUIC (Version h3-q050) implemented by Google in real-world. It gives evaluations on QUIC from a different point of view. It is also new to use data science methodologies to analyze network activitiy log files for video performance evaluation.

The remainder of this paper is structured as follows. Chapter two is a pre-investigation on the topic, covering TCP, QUIC and video streaming's backgrounds. Chapter three is Requirements. This chapter discusses and analyzes the necessary requirements to fulfill the goal. Following chapter three is chapter four explaining the design decisions, project approaches and system architecture. It is a chapter explaining the reasons to use specific tools, test methodologies, and system design. Chapter five is implementations, explaining the tools' instructions and code implements. Then we have chapter six, the test results, and discussion. This chapter independently discusses the performance of TCP and QUIC in speed and packet transmission, and then makes a comparison result. Chapter seven is the conclusion, reflection, and future works. It answers research questions and proposes future research directions. The last part is References.

Chapter 2 Background

2.1 TCP and its Limitations

The World Wide Web is based on a client-server application, while the client uses the Hypertext Transfer Protocol to send requests to WWW servers. The most commonly used version is HTTP 1.0 and HTTP 2.0, established on the TCP/IP protocol. TCP is a 46 years old industry standardized protocol introduced in 1974. Nevertheless, it never stops evolving since then (Duke, Braden, Eddy, Blanton, & Zimmermann, 2015).The IETF always has a team of people continuously working on it (M. Duke, M. Scharf, M. Tüxen, & Nishida, 2020). It is prevalent, mature, well-established, and stable. Thus, it is not easy to replace TCP.

However, using the TCP protocol does have disadvantages. TCP's slow start policy will not transmit the data with maximum speed. It either supports multithreading. So that if one packet is lost, TCP will work on recovering it first. Thus, no further packets on the connection deliveries until the lost one is retransmitted because there is only one connection. If one frame is lost in video streaming, the video freezes until the lost frame is retrieved. TCP has a fundamental design issue. It changes the congestion window once it detects any losses. For wireless connections, an unstable network environment is typical. Nevertheless, people are using WIFI nowadays, while the TCP protocol Continuously slows down the network due to network fluctuation. The protocol transmits data with the maximum speed that will be ideal, but TCP slows down the network. Lastly, the complicated three-way handshake would be cumbrous if the round-trip time is relatively long in a long-distance connection.

Long story short, TCP is a mature and prevailing transport protocol. It is hard to alter, but it does have irreparable defects. It is doomed to implement a new and superior protocol in the transport layer.

2.2 The QUIC Transport Protocol

In 2012, Google announced a new protocol called QUIC; they refer to it as the "next generation multiplexed transport over UDP" (GoogleDevelopers, 2014). The TCP protocol is considered more secure and stable than UDP, while UDP is quicker and does not check for packet losses. QUIC is combing the merits of TCP and UDP. It aims to be as reliable as TCP with less latency. It is built on UDP but has similarities with TCP, TLS, and HTTP/2(Chromium, 2014). It establishes connections faster than TCP; it combines TCP and TLS handshakes. QUIC is multiplexing; each stream is controlled separately. Missing packets would not block the whole connection since it would be in a single multiplexed stream. Another major improvement is that QUIC supports 0-RTT for a previously established connection. It is valuable when users switch from WIFI to mobile networks; the re-establishing time is reduced.

Besides, HTTP/1.1 keeps HTTP requests and responses in text format, the new version HTTP/2 keeps version 1.1's semantics but store messages in binary format. HTTP/2 has another evolution called multiplexing; it supports multiple streams in one connection. Thus, the request and response run in parallel

simultaneously; one failure would not block the connection. The cutting-edge version HTTP/3 had further improvements; it can recover lost packet with the least performance loss. It keeps using the same semantics as HTTP/2; it is based on QUIC. HTTP/3 stands for Hypertext Transfer Protocol over QUIC.

QUIC amended the disadvantages of TCP, but it is still under development. There is an active IETF QUIC working group(M. Westerlund, L.Eggert, L. Pardue, & Nottingham, 2020), the QUIC transport protocol draft is still in progress(Thomson & Iyengar, 2020). Ideally, QUIC should be faster and more reliable than TCP. However, the performance, efficiency, and behavior of QUIC are unclear. It is designed to be a better protocol, but is it really doing a good job? While the version of QUIC is still evolving as IETF is still standardizing the protocol. QUIC is implemented on Chrome by Google for YouTube video streaming. This project is studying QUIC's behavior, comparing HTTP/3 over QUIC and HTTP/1.1 over TCP, evaluating QUIC's performance comparing TCP, thus propose a conclusion and further studies.

2.3 Video Streaming technologies in YouTube

The technologies behind YouTube are developing over time. When YouTube is first built, the user needs to install an Adobe Flash Player plug-in to use it. In Jan 2010, YouTube implemented an experimental site that the HTML5 player can play videos in the browser (Amit, 2009). Furthermore, since Jan 2015, YouTube sites are using HTML5 as default. With the switch from Flash to HTML5, playback methodology changed from Adobe Dynamic Streaming for Flash to MPGE-DASH to adopt with the player.

DASH is an adaptive bitrate HTTP-based streaming technology that adapts the video quality according to the network status. For video playbacks, DASH player selects the next video chuck quality by estimation download rates. The algorithm calculates the download rate based on the previous video chuck transmission and the playback buffer size(Ragimova, Loginov, & Khorov, 2019). If the user selects 'auto' mode in the YouTube player, the video resolution will drop if the network speed is slow; relatively, if the buffer is filled, the video will be delivered in better quality.

The video formats are evolving too; YouTube was supporting MP4, WebM(Stephen, 2010), then it is primarily using VP9 and MPEG-4 AVC (Barman & Martini, 2017). Currently, the official site is suggesting the MPEG-2 format and MPEG-4 format (YouTubeHelp). The recommended video codec and audio codec for MPEG-4 are H.264 and AAC, with 24~30fps framerate, 1280x 720 to 640x480 resolution. YouTube supports resolution from 2160p to 240p (YouTubeHelp). Starting from 2019, User can use the new AV1 format in YouTube place-back settings(YouTube, 2018).

By manual tests and network activity observation, YouTube Player is showing a pattern during video playbacks. It divides the videos by chucks and downloads one chuck in one HTTP response. The transactions are intermittent and partially depending on the buffer size. When the buffer size is about to reach 0, the player would quickly download many video chucks; when the buffer size is around 120s, the transaction would temporally rest. This project is investigating how QUIC and TCP would affect YouTube streaming playback performance.

Chapter 3 Requirements

As stated in the background section, the aim is to compare TCP and QUIC's performance using YouTube downloads and to study the behavior of QUIC.

This project needs to capture YouTube downloading's process, activities, and performance; it can use the HTTP requests/responses and packets transmissions for the study. Section 3.1 discusses the detailed requirements for stage one: Network Activity Capturing. Secondly, this project needs to read and analyze the network activities log files. If existing tools cannot fulfill the assessment requirement, it must build a system that can compare video streaming behavior over TCP and QUIC. The system should read in the log files and gave customized results and visual graphs—detailed requirements in Section 3.2 Establishing Log Files' Parser and Analyse Tool. Last but not least, this project needs to discuss and examine the results, and then make a proper conclusion. See 3.3 Comparative Analysis of Research Data for more detail.

Figure 1: The Three Essential Stages in Requirements

The specific requirements and descriptions are stated in the following section. This section uses the Moscow methodology marks each one of them. The Symbol [M] stands for Must-Have, [S] is Should-Have, [C] means Could-Have, and [W] represents Won't Have-this-time.

3.1 Network Activity Capturing

R1: [S] Complete comparison experiments on both TCP and QUIC; also ensure the experiments' objectivity.

Description In order the achieve the goal, experiments should be taken on both TCP and QUIC. Since it's s comparative experiment, the transport protocol should be the only variant. It is important to control variables in the experiments, including network environment, video, latency, network maximum speed, and the duration. However, the real-world network contains many uncertainties and disturbing factors. The network environment could not be guaranteed to be exactly the same. It is required to do repetitive test runs; the average performance is measured.

Analysis The experiment is taken in the same period of time under the same WIFI to control the variables. Besides, the YouTube webpage is the only application active during experiments. The object should be the same video in the same resolution (1080p). The test duration should be the same. The only difference would be the HTTP version and transport protocol: HTTP/1.1 over TCP or HTTP/3 over QUIC(h3-q050). The exact same experiments are repeated five times to ensure reliability, getting the average performance in

approximation. Thus, it is also required to build a tool to merge the 5 test runs on visual graphs.

R2: [M] Capture HTTP requests/response detail during the experiment.

Description The research object is YouTube video streaming; thus, browser fetching video chucks is the key for performance assessment. It is required to get the HTTP response's timestamp, file size, transmission speed. Note that this is the requirement for both TCP and QUIC experiments.

Analysis the Google Chrome Developer tool can be used to capture HTTP Archive format files. The HAR could be saved for further assessment, and it contains the required information. Specifically, HAR file needs to record the whole process, the start of the first video chuck and the last video chuck. Also, since 5 test runs are merged together, each test run need to have one HAR file, the starting time and ending time should be similar. Each HAR should mee the standard and can be parsed properly.

R3: [W] Control and note down the video status information

Description YouTube player is embedded with a video status function. It shows viewport, frames, current resolution, codecs, color, connection speed, and buffer health. This information might affect the performance, noting them down is a could have a requirement.

Analysis The information provided in the video status bar does not seem to help with the research topic. Since the test runs on the same video, its viewport, frames, codecs, and color would not affect the result. The 'connection speed' is showing the transmission speed for one video chuck. Furthermore, it changes quickly, thus challenging to write down. The buffer health is also a temporary status, showing how long the buffered video is. Only the current resolution has a significant impact on the experimental results. However, resolution can be controlled and set to a fixed value. Then this requirement is a Won't Have-this-time requirement.

R4: [M] Capture packets transmission during the experiment

Description To better analyze the protocols, capture the packets' transmission data is essential. It is required to use network activities dumping tools to monitor the network and save the transmission status into local files. These files should be qualified for analyzing the protocol behaviors.

Analysis Both Wireshark and TCPdump could fulfill the requirement. TCPdump is easier to use; it is a terminal-based network traffic capturing tool. The file being saved is called pcap files, an abbreviation of packet capture. The specific requirement will be, use TCPdump to capture packets transitions and save as pcap files. Run proper TCPdump command in the terminal; the program should monitor the background's network activities and then save it into a pcap file. TCPdump should begin before the video starts playing. It should record the timestamps, IP address, port numbers, ACK, SYN number, data length, and so on for TCP. For UDP, it should record timestamps, IP addresses, port numbers. QUIC is recognized as UDP in TCPdump.

R5: [M] Capture QUIC's transmission details during the experiment.

Description It is required to capture log file recording QUIC's transmission behaviors. Nevertheless, in pcap file, QUIC's behavior is invisible on the client side since it is encrypted. Besides, the HAR files are only recording the HTTP response. Thus, it is needed to use a separate specific tool to capture QUIC's behaviors in the experiment stage.

Analysis The Google chromium project implemented a chrome network logging system called NetLog. NetLog saves a JSON file containing QUIC's connection details. Therefore, capture a NetLog file is required for every QUIC test run. The NetLog logging system needs to turn on in the browser before the video starts playing. The NetLog needs to be a valid file that could be opened in the Chrome NetLog Viewer (THemming, 2016) and Qvis(rmarx, 2020).

3.2 Establishing Log Files' Parser and Analyze Tools

R6: [M] Build Video performance analyzing tool

Description The project needs to evaluate YouTube video performance on the player and compares the efficiency of TCP and QUIC to achieve the research goal. The browser log file is saved, but there is no existing tool, so it must build an HTTP request parser/analyze tool.

Analysis The HTTP request/response log file is called HAR. Thus, it is a customized HAR parser and visualization tool. It needs to have data process and data visualization functions. Since this project is doing repetitive tests, it needs to take multiple input datasets and generate a summarize outcome. It needs to filter out the unrelated HTTP request, such as the advertisement videos and any non-video response. The tool should calculate video status in general and generates graphs. The video status should cover the IP address, protocol version, total/average wait time, average speed, total/average size. The plot graph contains the five test runs' speed trending. Each test run would have a speed distribution bar chart.

R7: [S] Build a transport protocol behavior analyzing tool.

Description The other research topic is to study the behavior of both TCP and QUIC, to study the connection process, packets transfer process in packets transmission log file. If there is no existing tool, it must build one that read in the log file and generate summary and visual graphs.

Analysis The packets transmission log file is called Pcap. Neither TCPdump and Wireshark can provide required data and graphs. So, this project is building a customized Pcap parser and visualization tool. This customized Pcap parser and visualization tool should parser and convert the binary data into readable data. For each packet, timestamp, packet length, data length, IP source, destination/ source port, destination port, and time gap since the first transition is recorded. There should be a summary of the connection's status and a visual graph showing the speed or data length trending.

R8: [W] Build the QUIC behavior analyzing tool.

Description It will be beneficial if there is a customized QUIC analyzer since QUIC is accessed as UDP in pcap files, and details are encrypted. This tool could provide specific data and graphs for QUIC's connection and data transfer.

Analysis The QUIC analyzer could be a program in qlog or Netlog files and present statics and graphs. A customized one would be beneficial. However, there is an existing tool Qvis; it already covers the requirements for studying QUIC's behavior. So, this requirement is marked as Won't have this time. It can be implemented in future works.

3.3 Comparative Analysis of Research Data

R9: [S] Using the tools to analyze TCP test runs

Description Using the Video performance analyzing tool and transport protocol behavior analyzing tool to investigate TCP's performance and behavior.

Analysis Analyzing TCP protocol on YouTube video streaming should use the customized HAR, Pcap parser visualization tools. Based on the results, analyze the speed, time, data size, the connection establishment process, the congestion control, and packet transmission patterns

R10: [S] Using the tools to analyze QUIC test runs

Description Using the Video performance analyzing tool and transport protocol behavior analyzing tool to investigate QUIC performance and behavior. The results should also give supplementary QUIC log file analysis results.

Analysis The research should use Qvis, customized HAR, and Pcap parser visualization tools to evaluate the QUIC's performance. The focus is on the speed, time, data size, connect the establishment process, and packets transmission patterns. Plus, learning and studying QUIC's behavior patterns on multiplexing, consistency, and error remedy.

R11: [M] Compare TCP and QUIC on performance and behavior.

Description Summarizing the results and graphs link the results into the protocol's behavior. Check if TCP and QUIC have similar low latency, and QUIC establishes the connection faster than TCP. Compare the video downloading speed and packets transmission mode.

Analysis Based on data statistic and visual graphs, compare the performance of TCP and QUIC, present results, and conclude the research topic. State which one has a better performance on YouTube Video streaming and responds better to network fluctuations.

Chapter 4 Design

4.1 Design Decisions

The study of DASH over QUIC is not a brand-new topic, yet researchers had already published several papers on this research topic. For example, one paper discussed the adaptability of QUIC on adaptive bitrate streaming techniques(Mondal & Chakraborty, 2020). The other conducted a performance study of DASH over QUIC(Bhat, Rizk, & Zink, 2017).

These researches are using self-established servers and embedded DASH player to do the assessment. They implemented the latest QUIC protocol version back then onto their servers and tracked the DASH player's video quality experience (QoE). They concluded that TCP is better than QUIC for DASH videos.

However, I'm researching in a completely different way. This project is analyzing the network traffic of YouTube video streaming on the client-side. There are several reasons why I made this decision. Firstly, it takes time and effort to build and run my own server. The main goal is to evaluate the performance; thus, learning from zero to build a server and implement QUIC would be time-consuming. Secondly, I do not have the equipment to interfere with the network, simulating a real-world environment, adding latencies, errors, and retransmissions. Last but the most prominent reason is that this research is aiming to present a completely different view. The experiments run in real-world network environments, testing the h3-q050 version implemented by Google. It would tell how properly Google had implemented QUIC, whether the current h3-q50 would perform better than TCP. Thus, other video streaming sites could have a clue whether to implement QUIC onto their sites and how beneficial this development is in real world.

Also, the evaluation strategy is not video QoE. QoE's calculation is based on the total number of video chucks, playback bitrate, video quality, rebuffering time, and other factors. However, during testing, YouTube videos barely buffer or freeze, the resolution barely changes neither. Thus, this project ensures the video qualities are the same and calculates the bitrate and number of video chucks.

The research is initially taken on the browser's network logs because it shows how video chucks are transmitted to the client. This is for evaluating the video performance. Then the network packets and QUIC's connection information is captured to study the protocol's behavior. Although the data in QUIC is all encrypted, Google does have tools to record QUIC's connection establishment process. The detailed research approaches are discussed in the next section.

4.2 Approaches

In the direction of analyzing transport layer protocol performances, capture and process the network activities are fundamental methods. The network activities could be capture in different forms, and each one would contain specific information. After initial experiments and researches, I discover three types of

network activity log files. The following paragraphs explain why these log files are chosen and the way to use them.

The first type is HAR (HTTP Archive format) files recorded by the browser, logging all browsers' activities with a website. They store the file type, status, protocol, size, transition time, the request method, request URL, IP address for each file (ex. Image, CSS, JS, XHR). Videos are split into small videos chucks and sent to browsers. HAR catches the transition of each video chuck. Thus it is a reliable source for studying the behaviors of TCP and QUIC.

Since HAR could not provide information on how the protocol is running in the background, the second type Pcap file is essential. Pcap is the abbreviation of packet capture; it is an API for capturing network traffic; the captured file is saved as .pcap files. It can be used to analyze protocols, monitor network, and detect network intrusions. In this project, a saved pcap file shows each connection's establishment and methods sending video chucks from the server to the TCP client. QUIC is a highly secure network protocol; the data is encrypted. In a pcap file, QUIC is recognized as UDP, and the packet size, timestamp, and port number are recorded.

Since the pcap file could only provide limited information on QUIC, this project imported NetLog, Chrome's network logging system(Roman & Menke, 2018), which is logging Chrome's network stack network events, it serialized the event streams and saved them into a JSON file. It captured QUIC's activities in the chrome browser, including handshakes, stream number, ack number, and RTT counts. These data are useful in analyzing QUIC's behaviors.

4.3 System Architecture

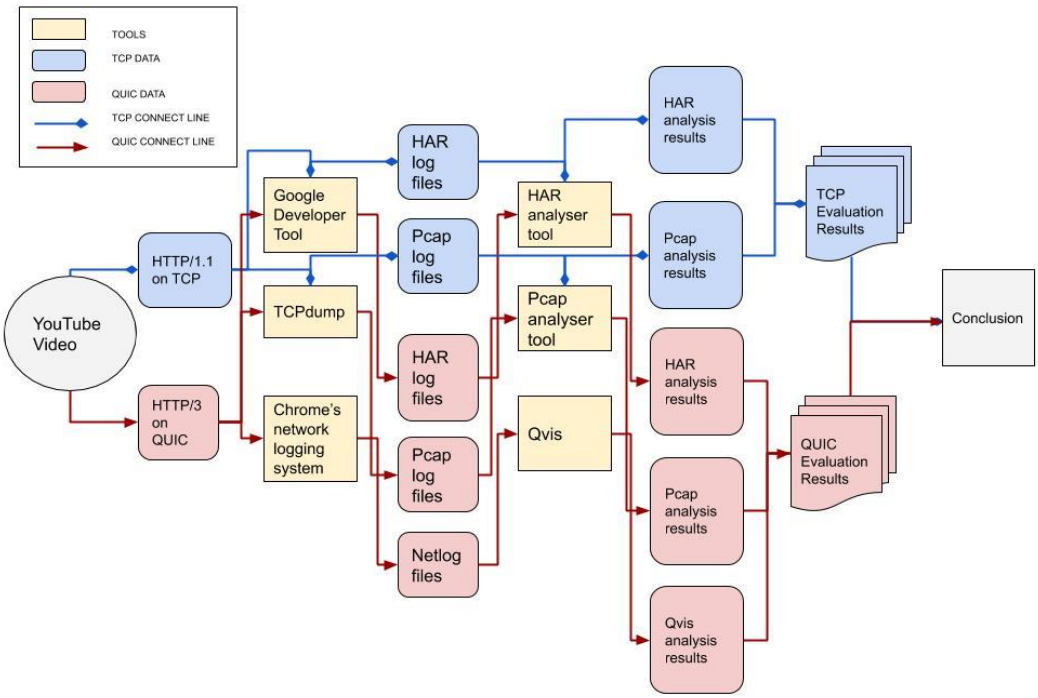

Figure 2: System Architecture for TCP/QUIC Analyzer on YouTube Videos

As shown in Figure 2, one YouTube video is converted into datasets by network capturing tools, and the dataset is summarized and visualized by data analyzer tools. The yellow rectangles are tools and technologies used for data processing. The round pink rectangles are QUIC's data, and the round blue rectangles are TCP's data. Both TCP and QUIC had an evaluation result on its behavior and performance; the conclusion is based on the comparison.

The TCP datasets, video streaming under HTTP/1.1 and TCP, are captured by Google Chrome DevOp Tool and TCPdump. Their outputs are HAR activity log files and Pcap log files. These two different types of log files are then passed to the HAR analysis tool and Pcap analysis tool separately. The HAR analysis results would include the video chucks' speed trend and speed distribution. The Pcap analysis results would include a visual presentation of speed/time, packets sizes/time, acknowledge and sequence number, TCP flag, destination port analysis. Combining HAR results, Pcap results, and TCP's designed behavior, the TCP evaluation on YouTube videos is made.

The same for QUIC datasets, the Chrome DevOp tool, and TCPdump, runs in the background while the video is streaming. QUIC used one more capturing tool, Chrome's network logging system, for NetLog files. Thus, the pipeline for QUIC has a HAR file, Pcap file, and Netlog file, which would be put into the HAR analyzer, pcap analyzer, and Qvis separately. The result of QUIC datasets includes video chucks' speed trending speed distribution in HAR, UDP's packet transmission speed/time and packets sizes/time in Pcap, and Qvis's QUIC connection process presentation. These results would then match with QUIC's designed protocol guidelines to see if it is behaving as expected.

The two evaluations are thus summarised to conclude which transport protocol is more suitable for YouTube video streaming.

Chapter 5 Implementation

This chapter would discuss the implementation. As explained in the system architecture section, there are six tools used in the implementation. Some of them are downloadable tools that are ready to use, while two are customized tools that need to be implemented. In this chapter, I will first introduce the available tools and techniques used in the project, and then show the implementation of two self-developed tools: HAR Parsing and Analyzing Tool and Pcap Parsing and Analyzing Tool.

5.1 Data Analyzing Tools and Techniques

As discussed in section 4.2 approaches, to study the research problem, this project needs to capture and analyze HAR files, Pcap files, and NetLog files. The tools and techniques used in this project are listed in the following paragraphs.

Google Chrome Canary

The browser used for YouTube video streaming is Google Chrome. More specifically, Google Chrome Canary, an advanced version for developers. Chrome supports both TCP and QUIC. The default setting is HTTP/1.1 TCP for YouTube streaming. To enable QUIC, go to the Experiments page[1], search for "Experimental QUIC protocol," and enable it, relaunch the browser. QUIC is known enabled in Chrome.

Google Chrome Developer Tool

The Chrome developer tool, as shown in Figure 3, is used to capture the HAR file. It is embedded in the Google Chrome browser and records the network activities in Chrome. Right-click on the webpage and select 'inspect,' the DevOp Tools would come up. Then click on the 'Network' tab. It is the network activity monitor page. Video chucks are recognized as 'XHR' types. The network protocols are not initially shown on the page, so right-click on the HTTP request header bar and select 'Protocol' and 'Scheme.' Make sure the 'preserve log' is not checked and export the HAR file and refresh the page for a new test run.

Tcpdump

It is a sniffer and packet analyzer tool and can catch TCP/IP and other protocol packets. Tcpdump is running under a command-line interface. Tcpdump can run in the background, but it needs to be enabled before the video starts streaming. The command varies for different operating systems. I'm using a MacOS, and I used commands formatted as below.

```
tcpdump -ni en0 -w 910-tcp5.pcap
```

This command calls Tcpdump and monitoring the en0 network interface. -n disables the reverse DNS so that packets capture will not be delayed by reverse

[1] chrome://flags/

lookup.(Netgate, 2020) -w enables the Tcpdump to write packets into a file; in this case, it is '910-tcp5.pcap'.

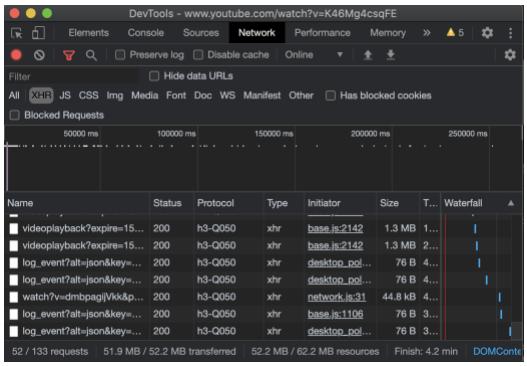

Figure 3: Google Chrome Developer Tool.

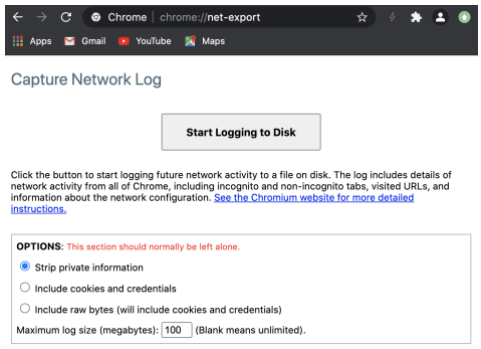

Figure 4: Chrome's Network Log Collecting tool

Chrome's network log collecting tool

This is an online tool built by Google to capture and analyze the network activities in NetLog format. Its screenshot is shown in Figure 4. The merit of it is that it supports QUIC protocol. This tool is also inside Google Chrome. Go to the 'Network Log Export' Page[2]. Before the video starts, click on 'Start Logging to Disk', select the directory NetLog file is saved. Click on 'Stop Logging' after the video ends. Then click on 'Start Over' for the next test run (GoogleChromeEnterpriseHelp).

Qvis

Qvis is a QUIC and HTTP/3 visualization tool suite. It supports NetLog and provides a visual presentation on QUIC's streams, congestion and packetization. Robin Marx develops qvis from Hasselt University. Qvis it is hosted online (qvis). Change the Netlog file extension from '.json' to '.netlog' and upload it onto the site. Then check the 'Sequence' and 'Congestion' Page for more information.

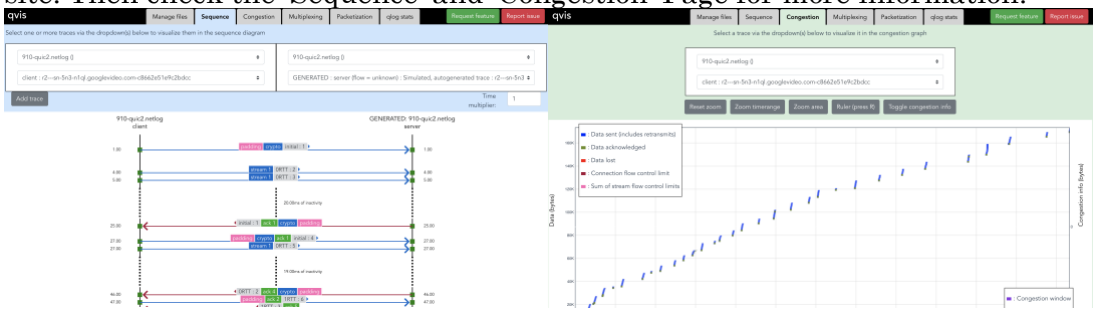

Figure 5: QUIC and HTTP/3 Visualization Tool Screenshots

5.2 HAR Parsing and Analyzing Tool

The customized HAR parsing analyzing tool is developed because existing tools cannot fulfill certain requirements. This analyzing tool is programmed with Python on Google Colab notebook. The imported Python libraries are the introduction is in technical specifications; the program structure and functions

[2] chrome://net-export/

demonstrations are in implantation detail. The code implementation and results are available to check online[3]

5.2.1 Technical Specification

The python libraries imported include Colab, Json, Pandas, Matplotlib, Datetime and haralyzer. The following table introduces the python libraries.

Library Name	Description
Colab	Import' drive' from google.colab can mount the Google driver onto the Colab notebook So that the program can read the network activity log files from Google Drive.
Haralyzer	Haralyzer is a python framework parsing HAR files. It separates the HAR file into single entries. (mrname, 2020) 'HarPage' from Haralyzer process one web page's HTTP request. The self- developed analyzer tool is based on Haralyzer, converting JSON format logs to a python list. Haralyzer also supports filtering the HTTP entries to video chucks only by checking the content type.
Json	The HAR files are in Json format. So, reading the file would need Json to load and process it first.
Pandas	Pandas is a commonly used data analysis and manipulation tool. It is used to better filtering, processing, and calculating the datasets. The HTTP entries are converted into pandas dataframe. As a result, the dataset is better organized and calculated. It calculated and presented the subtotal, average, speed, and other data.
matplotlib	Matplotlib is a python library for generating static data visualization graphs. It is used in this project for data's line plots, histograms and scatter plots.
Datetime	Datetime is a module in python library, it helps to manipulate dates and times (Python, 2020). The HAR's date time data is recorded this way, need to import datetime library to process them.

Table 1: Python Libraries used in HAR and Pcap Parser Tools

5.2.2 Implementation Details

As shown in figure 6 below, the program takes in HAR files from Google Drive and store in the Python list. Then it filters out the non-video entries, stores the data in pandas dataframe. Calculate each video chucks' transmission speed and

[3] https://colab.research.google.com/drive/1DzWjfMbtLfO13Sw_qN8mN_u4AaTR0zRM?usp=sharing

time gap since the first transmission. After that, the program filters out the advertisement video entries, leave only the main video. It then calculates the video statistics and generates graphs.

The video statistics cover the IP address, protocol version, total time, number of video chucks, wait time, average speed, content size, transfer size, and average size. The plot graph contains the five test runs' speed trending and content size trending by time. Each test run would have a speed distribution and a size distribution bar chart.

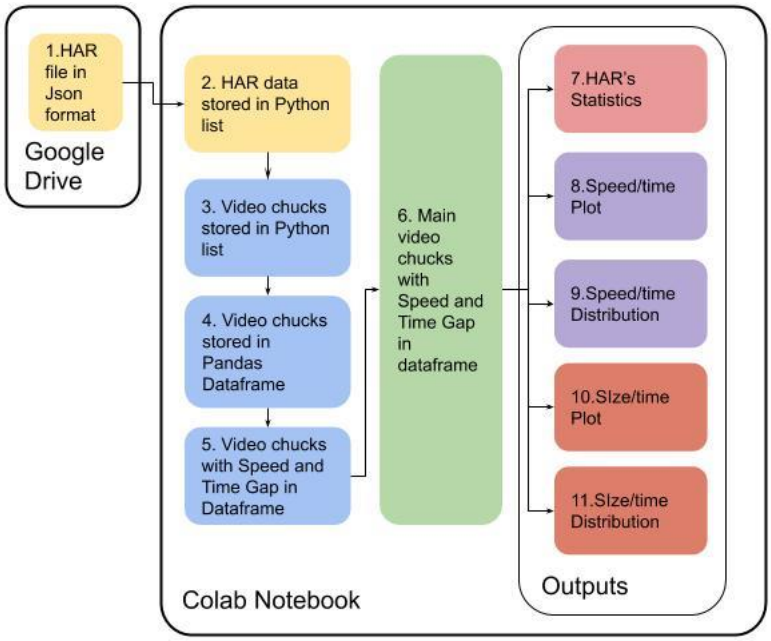

Figure 6: HAR Parsing and Analyzing Tool Flow Chart

There are three functions implemented, and I will introduce the function's description and functionalities.

HARtoDF: Converting a HAR file into pandas dataframe. Imported and used Haralyzer, Json, Pandas, datetime libraries. This function covers stages 1, 2, 3, 4, 5 in the Figure 6 flow chart. It is mainly converting and processing the HAR file, also adding computed data. This function reads the HAR file in Google Drive, creates a python dictionary, and reads through the HAR file to get status, transfer size, content size, HTTP version, server IP, start time, receive the time, wait time. Then it converts the dictionary into dataframes, adds HTTP response's time duration since the first transaction into the it. Then return the new dataframe.

Statistics: Filter out unrelated data and print out data statistics. Imported and used the Pandas library. This function covers stage 6,7 in the Figure 6 flow chart. This function reads the video chuck dataframe, determines the IP address lists. There are unrelated IP addresses sending video chucks, such as the advertisement videos. However, the primary video should be sending the most significant data size. So, this function finds the IP address sending the maximum size, and marks it as the main video, then filters out all the other IP addresses. Then it iterates through the dataframes to calculate and print out the subtotals and averages. In the end, return the main video chucks' dataframe.

GenereateGraph: Input five filtered dataframes, generate, and present visual presentations. Imported and used Pandas and matplotlib library. This function covers stages 8, 9, 10,11 in Figure 6. This function takes in five video chucks dataframes, use the speed, size, time in the dataframes generates four types of graphs. The first graph presents five test runs' data transmission speed in one graph. It marks the x-axis as 'Speed (Mbps)' and the y-axis as 'Time (Seconds)' and adds the legends. Then this function generates the speed histogram graphs; it buckets the value range to be 60 and add colors for each range based on quantities. The most frequent ones are yellow, followed with green and blue. Then the function uses the content size and time gap in the dataframe to generate five test runs' data size plot and each test's size's histograms. The manifestation is similar to speed's graphs; the only difference is showing content size (bytes) instead of speed.

5.3 Pcap Parsing and Analyzing Tool

This analyzing tool is also programmed with python on Google Colab notebook, the python models used in the program is introduced in technical specification, and the program structure and functions are demonstrated in implantation detail. The code implementation and results are available to check online[4].

5.3.1 Technical Specification

This customized Pcap parser and visualization tool are also using python models such as pandas and matplotlib, to process data. In addition, it used the dpkt library and socket library.

Library Name	Description
Dpkt	Dpkt is a python module for network protocol parsing(Bandla & Obormot, 2020); it is mainly for IP, UDP, and TCP parsing in this project. Import the dpkt.ip, dpkt.udp, dpkt.tcp would help to identity and parser the protocols. The original data is just binary data; use dpkt to convert them to be readable strings.
Socket	The python socket library can convert packed IP addresses into standard, readable string representations.

Table 2: Pcap Parser Tool Deliciated Python Libraries

5.3.2 Implementation Details

As shown in figure 7, the program takes in pcap files from Google Drive and uses different algorithms to process TCP and QUIC's pcap files and then save the packets data in Pandas Dataframe. QUIC (recognized as UDP in Pcap) datasets are red rectangles, TCP's datasets are blue rectangles, and the yellow rectangles are universal stages.

The tool picks the HTTPS packets. Calculate and find the IP address transmitting the maximum data, mark them as the video's packets. To calculate

[4] https://colab.research.google.com/drive/1APGo4t_ws4uNy8ge3r52IIrjnlmsiXKa?usp=sharing

the speed, divide the packets into groups on time interval 1 second. It is adding up all the data transmitted in 1 second to get an approximate speed. Then the tool determines and prints out the video's statistics and generates a speed plot, speed distribution, overall packet size plot. The plot graph contains the five test runs' speed trending. Each test run would have a speed distribution and a packet size trending plot.

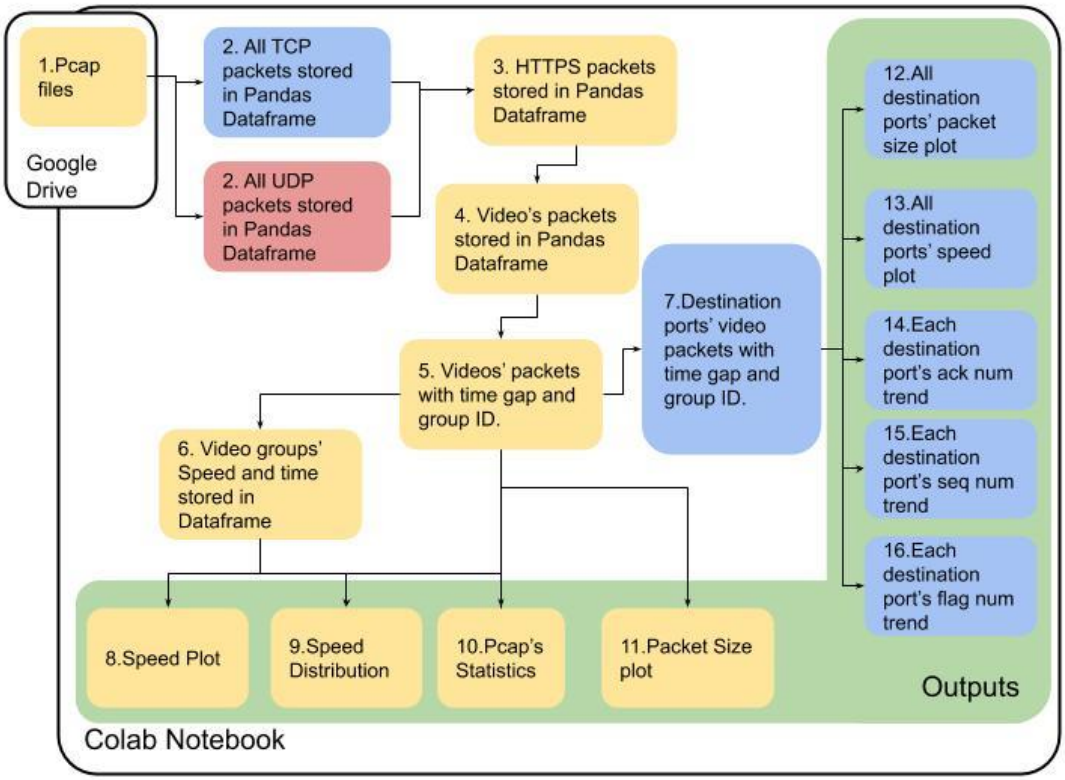

Figure 7: Pcap Parsing and Analyzing Tool Flow Chart

TCP and UDP are behaving differently; TCP might change the destination port in one connection. Thus, this program finds different destination ports in one connection and output their packet size plot. And each port's acknowledge, sequence number and flag number trending.

Based on the data processing structure, there are unique functions implemented in this program. Those that have not been introduced in HAR parser tool will be introduced here. I will introduce their description and functionalities.

ProcessPcap: This is a comprehensive function. It takes in one pcap file, loads it, and calls other functions to process it. As a result, it prints out pcap's statistics and returns dataframes for plot generation. This function covers steps 1, 2, 3, 4, 5, 6, and 10 in Figure 7. This function checks the protocol type and calls either function readUDP or readTCP to parser the input file. Then it filters for the HTTPS packets and calls function findvideo to find the video's packets. It thus calls function addgroup to add time gap and group ID onto the video's packets. Call getSpeed to calculate the speed for each group. Then print out some of Pcap's statistics.

readUDP: Using dpkt to read and parse QUIC's pcap files. Filter for UDP packets and save them into the dataframe. Imported and used dpkt.ip, dpkt.udp, pandas, and socket library. This function is for QUIC's pcap file only, covers stages 1 and 2 in Figure 7. It loads pcap files. Use dpkt to check for UDP packets, use dpkt and socket to convert the data into readable strings, find the timestamp, length, data, IP source, IP destination, source port, destination port, and save into python dictionary. Then convert Python dictionary to Pandas Dataframe.

readTCP: Using dpkt to read and parse TCP's pcap files. Filter for TCP packets and save them into the dataframe. Imported and used dpkt.ethernet, dpkt.ip, dpkt.tcp, pandas, and socket library. This function is for TCP pcap file only, covers stages 1 and 2 in Figure 7. This function's functionalities are the same as in readUDP, but it saves more data into the TCP dataframes: the acknowledge number, sequence number, and TCP flag in the decimal presentation.

findvideo: Find the video's packets from the HTTPS packets. It used the Pandas library. This function is stage 4 and 10 in Figure 7. This function goes through the HTTPS packets' IP list and finds the one with the maximum bytes transferred, mark it as the video's IP address. And then filter the dataframe to video packets only. It also calculated the total content size, transfer size, and print out information on size, speed, and packets.

addGroup: Read the video packets and separate them into groups based on timestamp. It used Pandas library. This function is stage 5 and 10 in Figure 7. This function calculates each packet's time gap, which is the time difference from the first packet. It also tracks whether the destination port had changed in the connection. It divides the packets into groups; one group contains all the packets transmitted in 1 second. Then it outputs the video packets with a time gap and group ID added.

getSpeed: Combine the video packets by group ID and calculate the data size transmitted in 1 second as the speed. Output a new dataframe of video groups. It used the Pandas library. This function is stage 6 in Figure 7. This function takes in a dataframe and finds all the group IDs in this dataframe. Each group adds up all packet's data length, thus getting a subtotal of bytes transferred in 1 second. It creates a new dictionary with the video group's information and the speed, then converts to the dataframe. Output the new video group dataframe.

Showpackets: Take in the video packets dataset and list of destination ports. Print video packets' size plot. Separate the video packets into different destination ports and study the speed, size, ack number, seq number, and flag number. Present the results. Imported and used Pandas and matplotlib library. This function is stage 11,12,13,14,15,16 in Figure 7. This function generates a video packets size/time gap plot for both UDP and TCP datasets. However, TCP might change destination port numbers, so when the port number is larger than 1, this function will divide the video packets by different port numbers and show each one's speed and the packet length. Furthermore, it also uses line plots and scatter graphs to show each port's acknowledge number/sequence number/flag number trending.

Chapter 6 Testing and Evaluation

6.1 Testing and Evaluation Strategies

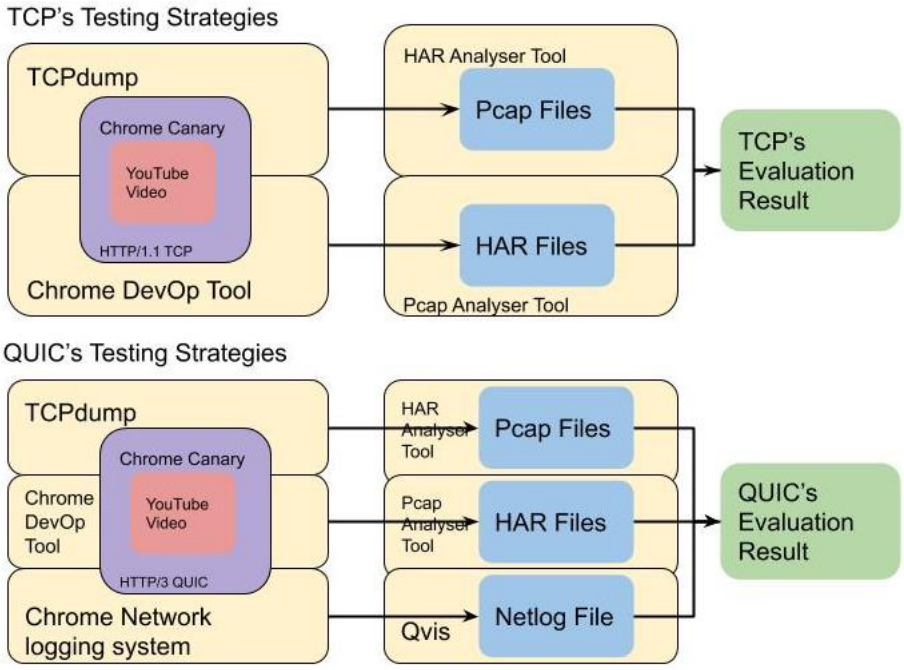

Figure 8: Testing Strategies' Flow Chart

As shown in Figure 8, the testing strategies for TCP and QUIC are similar; QUIC has one more step than TCP. These strategies are used repetitively for five times to ensure the testing results' reliabilities.

Testing YouTube video performance on TCP needs to set the Chrome Canary to HTTP/1.1 TCP and start TCPdump and Chrome DevOp Tool before video playing. Then export Pcap files from TCPdump and HAR files from the Chrome DevOp tool, load these files into their analyzer tool separately, and give overall feedback.

Testing QUIC's performance needs to set Chrome Canary to HTTP/3 QUIC first, and then do the same steps and TCP. However, there is one more step. Start the Chrome network logging system before video playing and load the Netlog into Qvis to get accurate analysis results. Combine the Pcap, HAR, Netlog analyze results and present the QUIC's evaluation Results.

The evaluation strategies of experimental results are answering behavior questions. QUIC's theoretical behavior pattern is in definition and research paper, but the actual application's real behaviors are unknown before running the experiment. It is why questions were raised. The test results are used to answer these questions.

1 How many connections does TCP and QUIC establish for one video? Did it change the destination port frequently?
2 For both TCP and QUIC, how does the connection performance?

3 Compare the HAR and Pcap files; how does each video chuck transmit in packets?
4 How many streams do QUIC have in one connection? Is it one stream for one video chuck? Is there any pattern?
5 Compare TCP and QUIC, which one is more stable?
6 Does TCP and QUIC had a significant overall variation in download speed?
7 How much different is TCP and QUIC in packet transitions?
8 Which one establishes the connection quicker?
9 Which one response better to network fluctuation?

In general, the evaluation criteria are the transport protocol's stability and efficiency. The speed, data size, and network transmission method are evaluated.

6.2 Evaluation Results and Discussion

6.2.1 TCP's Evaluation and Discussion

In this section, I will combine the HAR and Pcap analyzing results. Furthermore, explain and discuss them. The HAR and Pcap's statics are shown below.

	The IP Address: 213.143.10.237 Version: HTTP/1.1								
Test No.	**Total Time**	**Receive time**	**Receive time ratio**	**Content Size**	**Transfer Size**	**Video chucks**	**Size (Avg)**	**Receive time (Avg)**	**Speed (Avg)**
1	79 s	8.632s	0.851	49383906 bytes	49418219 bytes	31	1593029 bytes	0.278s	63.534 Mbps
2	123 s	7.735s	0.907	49383906 bytes	49417113 bytes	30	1646130 bytes	0.257s	74.421 Mbps
3	123 s	10.365s	0.931	49383906 bytes	49417114 bytes	30	1646130 bytes	0.345s	66.299 Mbps
4	122 s	13.158s	0.903	49383906 bytes	49417114 bytes	30	1646130 bytes	0.438s	57.977 Mbps
5	123 s	12.37s	0.938	49383906 bytes	49417114 bytes	30	1646130 bytes	0.412s	46.507 Mbps

Table 3: TCP Five Test Runs' HTTP Archive Statistic

From the statics in Table 3, we can see that test 1 ends earlier than other tests, but the average speed is not the fastest, and the receive time ratio is the smallest. That means the videos loaded faster in test 1, but the video chuck downloading speed is not accelerated, it actual spend more time on waiting. Thus, test 1's performance is not a typical example.

The Content size is all the same for five tests. That means that the HAR analyzer is doing a good job, filtering out all the unrelated HTTPS requests and summing up the main video chucks' size. The same video with the same resolution should be the same size. The Transfer size is slightly bigger than the content size; this is because the HTTP response also contains HTTP status code and headers. Test 1 is bigger than other test runs; I think it is because it has 31 video chucks so that it would have more header size.

The HTTP's transmission timing includes blocked, DNS, SSL, connect, send, wait, queue, and receive. The receive ratio is the receive time divide by the total time, showing how much percentage is the HTTP request spend on receiving video chucks.

Speed is an essential criterion for performance. The average speed is video chuck's size divide by the receive time; it is precisely for video chuck downloading. Receive time, and average speed are corresponding inverse function. TCP is showing a relatively fast downloading speed from **46 Mbps to 74 Mbps.**

	The IP Address: 213.143.10.237							
Test No.	**Total Time**	**Content Size**	**Transfer Size**	**Packets**	**Groups**	**Size (Avg)**	**Speed (Avg)**	**Dst Port**
1	207.2s	54170022 bytes	56153978 bytes	38153	30	1419.8 bytes	13.77 Mbps	53502, 53501
2	169.18s	51998419 bytes	53903695 bytes	36638	35	1419.2 bytes	11.334 Mbps	54741, 54742
3	168.99s	52001556 bytes	53905936 bytes	36622	38	1419.9 bytes	10.44 Mbps	54741, 54742
4	188.56s	52006862 bytes	53913894 bytes	36672	39	1418.1 bytes	10.173 Mbps	57788, 57789
5	169.34s	52000333 bytes	53905725 bytes	36642	38	1419.1 bytes	10.44 Mbps	57789, 58706, 57788

Table 4: TCP Five Test Runs' Packets Capture Statistic

Table 4 is showing the statistic for Pcap files. Network packets' contents are binary data, so it is hard to filter out all the non-video packets. Only the IP address transmitting video chucks could be defined. The connection might be transmitting other data with that IP after fully downloaded the video. Luckily, we can still estimate the end time in Figure 10, when the speed reaches 0 in the packet speed plot. Thus, the 'time' is only the time TCPdump stops running.

Expect for test 1, the packets' content size is 52000000 bytes, which is slightly bigger than HAR files' content size. It proves that the contents of packets are not all videos, but also other data. Test 1 might have some errors, so it retransmitted more packets (38153 packets) than usual (36672 packets) for the same video. The Transfer size is around 53900000 bytes, and it contains both the contents and packets headers. The packets' average size of all five is 1419 bytes. It shows that the analyzer is doing a good job. The maximum transmission unit for Ethernet is 1500 bytes, remove the header size, TCP is trying to use the largest size for each transmission.

The original video is about 3 minutes, one TCP connection usually changes destination port once, sometimes twice, as shown in 'Dst Port.'

The TCP packets are separated into groups by 1 second time interval; the group numbers depend on the time interval length. Ideally, the number should be close

to HAR's video chuck number. The number of groups and average speed is showing an inverse proportional pattern.

The essential criterion is the average speed. Test 1 has an extremely high speed of 13.77 Mbps comparing to others; the most common value is 10.44 Mbps.

Comparing the HAR and pcap results, real-time spent on video transmission is 123s. The video content size is 49383906 bytes, packets' content size is 52000000 bytes. TCP packets are not only transmitting video. The average single speeds vary greatly (10.44 Mbps, 46~74 Mbps). In a real-world connection, the packets spend much time waiting. Especially, TCP is non-multiplexing and recovering missing packets.

The HAR and Pcap's Speed Trending and Speed distribution are shown below.

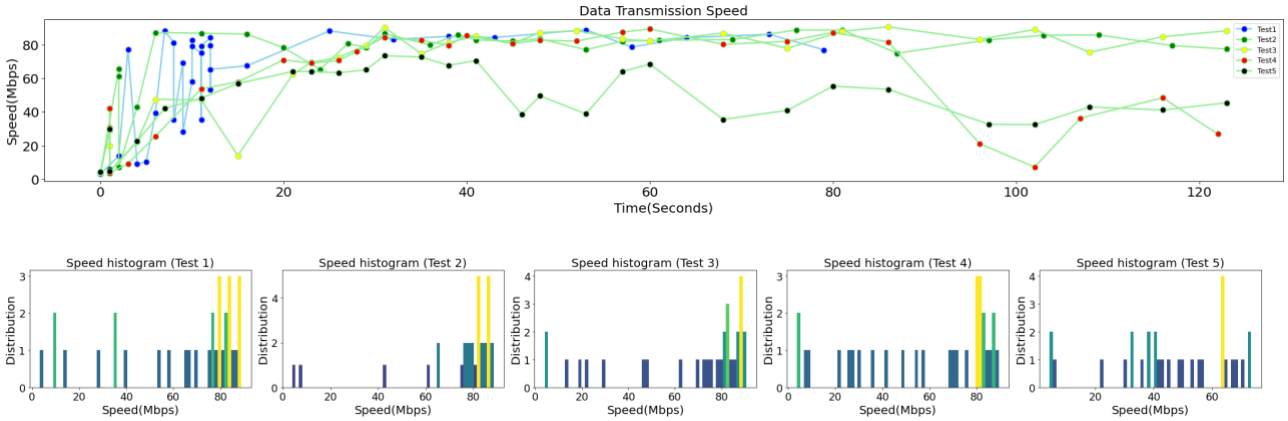

Figure 9: HAR's Speed/Time Plot and Speed Distribution.

As in Figure 9, test 1 is super fluctuating in the first 10 seconds in the speed plot. I think there is a transmission error that caused TCP to transfer quickly to remedy the error. Test 1's last video chuck is transmitted in 80 seconds. All the other test runs end around 120 seconds. The video is almost 3 minutes long, that means the video playback is loaded in 2 minutes, the player is using the buffer for the rest of the time. Test 2 and Test 3 are relatively stable; the speed gradually increases from 0 to 85Mbps. Test 4 and Test 5 are somehow descent slowly. From the speed distribution plot, most of the speed is more than 80 Mbps in test 1, test 2, and test 3. Test 4's majority is around 80Mbps, and test 5's majority is around 65 Mbps.

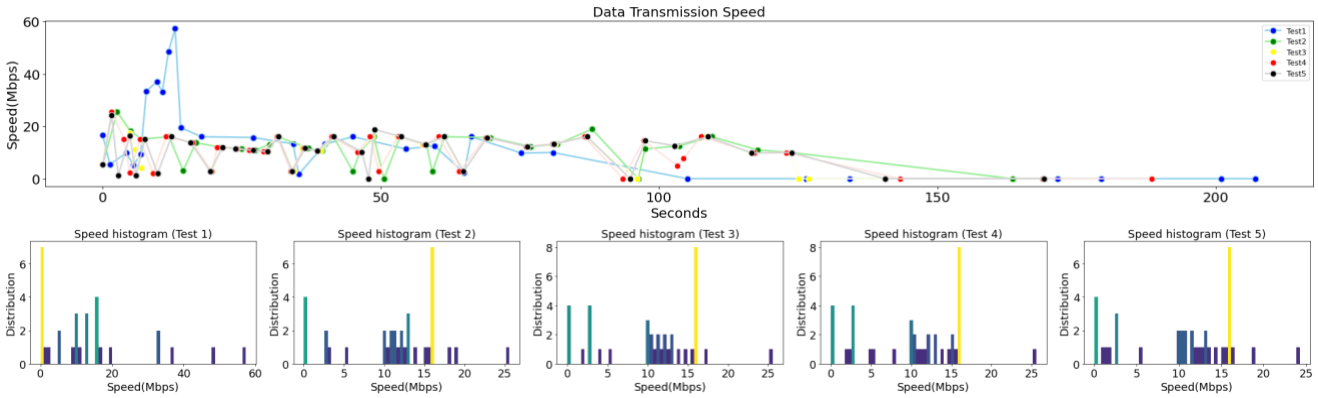

Figure 10: Pcap's Speed/Time Plot and Speed Distribution.

From Figure 10 Pcap's speed plot, we can see that most test's connection ends on the 140th second, test 1 end on the 110th second, and test 1 is transmitting a massive number of packets from 10th to 20th seconds. The average transmitting speed is 10Mbps to 20 Mbps, while test 1 reaches 60 Mbps in the beginning. The lines of other tests almost overlapped, and the speed distribution also shows that the majority is 16 Mbps.

The HAR and Pcap's Packets Length (Size) Trending and Speed distribution are shown below.

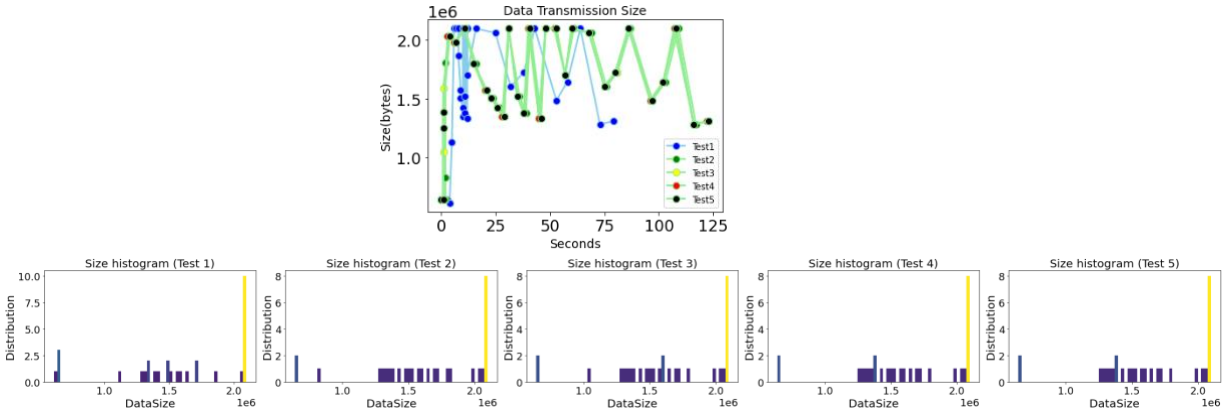

Figure 11: Video Chucks' Size Plot and Distribution

Figure 11 shows that most of the video chuck's size is 2 x 10^6 bytes in the distribution plot. Test 1 sends many video chucks from 0 seconds to 20 seconds. The other tests overlap with each other; this means that in usual condition, the YouTube video player is fetching video chucks for the same size at the same time.

Figure 12 below is showing the size of the packets by time in test 5. Each dot represents one packet, and its y-axis is packet length, the x-axis is the time. There are multiple packets sent at the same time. The port number changes back and forth. Test 5 has three ports, but only two are active. It also changes back forth from the second port and the third port. The interval is distributed evenly to expect the beginning of the connection. TCP might need to send more to establish the connection in the beginning. As in the packet length distribution plot, almost all packets are over 1400 bytes, and a few packets are 1000 bytes or around 600 bytes.

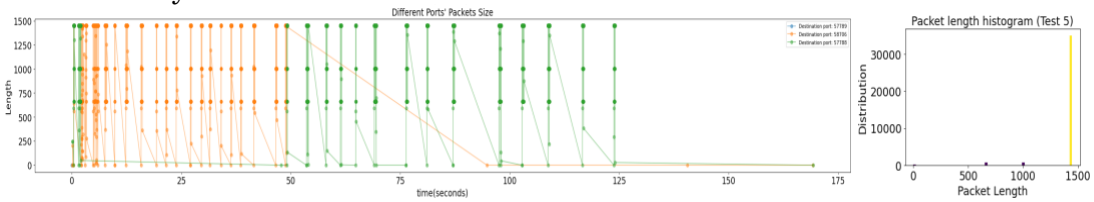

Figure 12: Destination Port's Packet Size /Time and Distribution Plot in Test 5

Destination port's connection usually starts with the TCP flag 18 (ACK+ SYN) and ends with flag 17(ACK+ FIN), all the other flags inside the domain are 24 (ACK+PSH) and 16(ACK). When the connection is established, SYN is used, sending data come with a PSH, and connection termination contain a FIN.

Figure 13 is showing test 5's acknowledge number grow slowly over time while the sequence number grow dramatically for each piece. The beginning of

acknowledge numbers shows congestion control. At the beginning, there is a slow start, and then it starts to grow linearly.

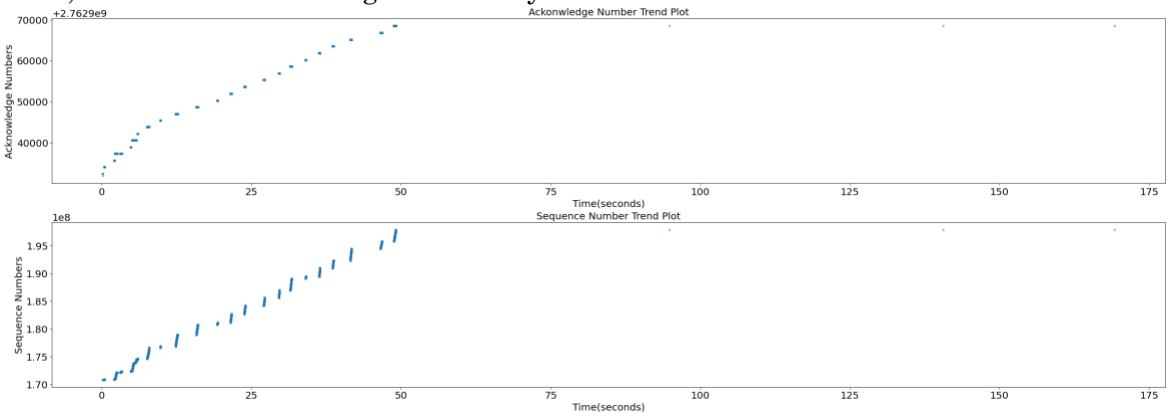

Figure 13: Ack, Seq Number Trend for Destination Port 58706 in Test 5

6.2.2 QUIC's Evaluation and Discussion

In this section, I will present and discuss QUIC's HAR and Pcap file's results. HAR and Pcap's statistics are shown below. Since the value definitions methods are explained in the TCP section, this section only summarizes the data.

	The IP Address: 213.143.10.237 Version: h3-q050								
Test No.	**Total Time**	**Receive time**	**Receive time ratio**	**Content Size**	**Transfer Size**	**Video chucks**	**Size (Avg)**	**Receive time (Avg)**	**Speed (Avg)**
1	123s	11.59s	0.896	49383906 bytes	49387196 bytes	30	1646130 bytes	0.386s	47.938 Mbps
2	122s	14.4s	0.934	49383906 bytes	49389131 bytes	30	1646130 bytes	0.482s	41.855 Mbps
3	123 s	12.18s	0.932	49383906 bytes	49389257 bytes	30	1646130 bytes	0.406s	47.15 Mbps
4	122 s	10.93s	0.927	49383906 bytes	49389257 bytes	30	1646130 bytes	0.364s	51.208 Mbps
5	122 s	13.76s	0.937	49383906 bytes	49388670 bytes	30	1646130 bytes	0.458s	48.628 Mbps

Table 5: QUIC Five Test Runs' HTTP Archive Statistic

As in Table 5, QUIC's statistic is consistent. The time for video chuck downloading is 122s, receive time is around 12s, content size is all the same 49383906 bytes, the average size is identical 1646130 bytes. The transfer size is slightly bigger than the content size; the most shared content size is around 49389257 bytes; test 1 is smaller than other tests.

The average receive time and average speed are inversely proportional. The average speed is around **41.85 ~ 51.2 Mbps.**

Table 6 shows the consistent test statistic for all five in QUIC's pcap files. The total time is around 123.2s; the content size is approximately 53080000 bytes, the Transfer size 53800000 bytes, the number of packets around 36600, average

size 1450 bytes. Only Test 2 is 1435 bytes; more non-video packets are transmitted in it. There is only one destination port for one QUIC connection.

	The IP Address: 213.143.10.237							
Test No.	**Time**	**Content Size**	**Transfer Size**	**Packets**	**Groups**	**Size (Avg)**	**Speed (Avg)**	**Dst Port**
1	124.25s	53070385 bytes	53801685 bytes	36565	33	1451.39 bytes	12.26 Mbps	61436
2	123.19s	53081365 bytes	53821065 bytes	36985	33	1435.21 bytes	12.27 Mbps	52327
3	123.87s	53075347 bytes	53806707 bytes	36568	34	1451.41 bytes	11.9 Mbps	55124
4	123.20s	53164810 bytes	53897610 bytes	36640	31	1451 bytes	13.08 Mbps	53250
5	123.15s	53069006 bytes	53800286 bytes	36564	34	1451.4 bytes	11.9 Mbps	59225

Table 6: QUIC Five Test Runs' Packets Capture Statistic

The number of groups is again, how many groups of QUIC packets when separating them into 1 second time interval. The more compact the packets are, the faster the speed. The average speed is from **11.9 Mbps~13.0 Mbps**.

Comparing the results of HAR and Pcap, we can see that the analyzer tool is doing a proper job. Video transition time is around 122s, the Pcap analyzer correctly filtered out the un-related packets. Thus its total time is also 123s. The total packets' total size is slightly larger than video chucks, but the average speeds are much slower than video download speeds. It is because not all packets are for videos and the receive time becomes a lot bigger.

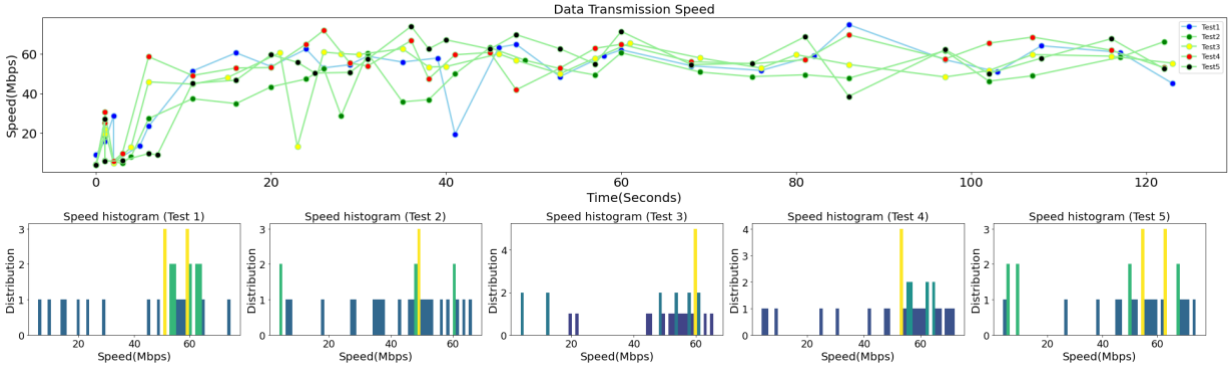

Figure 14: QUIC's HAR Speed/Time Plot and Speed Distribution.

Figure 14 shows that the act of video chuck speeds is impressive; it raises to 30 Mbps and then drops to 10 Mbps, then gradually increases to 60Mbps. The five test results follow the same route, there are some fluctuations, but the connection soon recovers. Test 2 is starting slower than other tests, the average speed is also the slowest in HAR files, but it catches up with the trend after 30 seconds and is

stable after that. If the sample video is longer than this, I assume the average speed will be more consistent because it is regular only expect the beginning. In Figure 15, the most prominent feature is in the distribution graphs. All tests' speed majority is 16 Mbps. The yellow bar presenting 16 Mbps is much higher than other bars in the distribution plots.

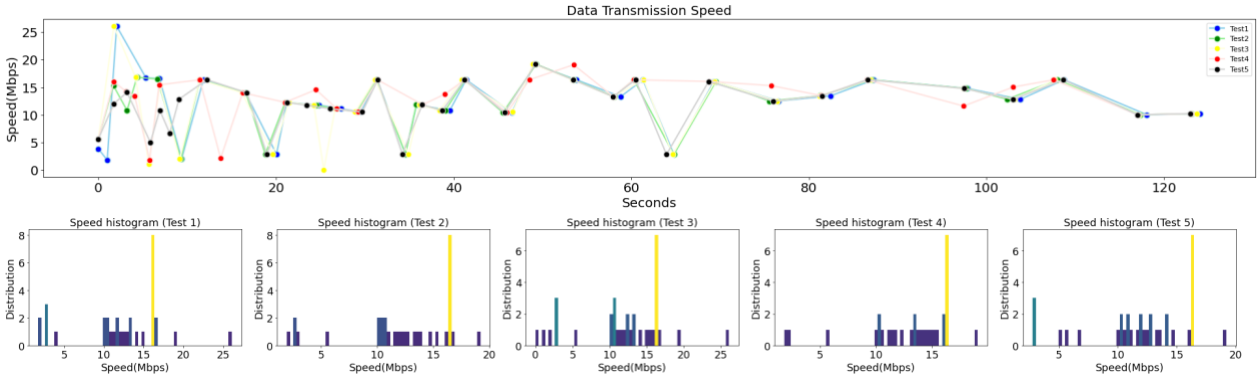

Figure 15: QUIC's Pcap Speed Plot and Distribution.

In Pcap's speed plot, all five tests are capricious before 20 seconds. After that, they follow a pattern until the end. Test 1, Test 2, Test 5 overlaps a lot. Test 3 drops to zero once; it has the lowest average speed among all in Table 6. Test 4 is the most unconventional. It is usually faster than others; its average speed is also the fastest, 13.08 Mbps.

The next step is studying the video chuck size and the packet length. Figure 16 is showing the video chucks size for all five QUIC test runs. It is clear that the size trends overlap with each other, and the distribution majorities are all 2.1x10^6 bytes. That means the YouTube video player uses the same algorithm to request video chucks size at the same time.

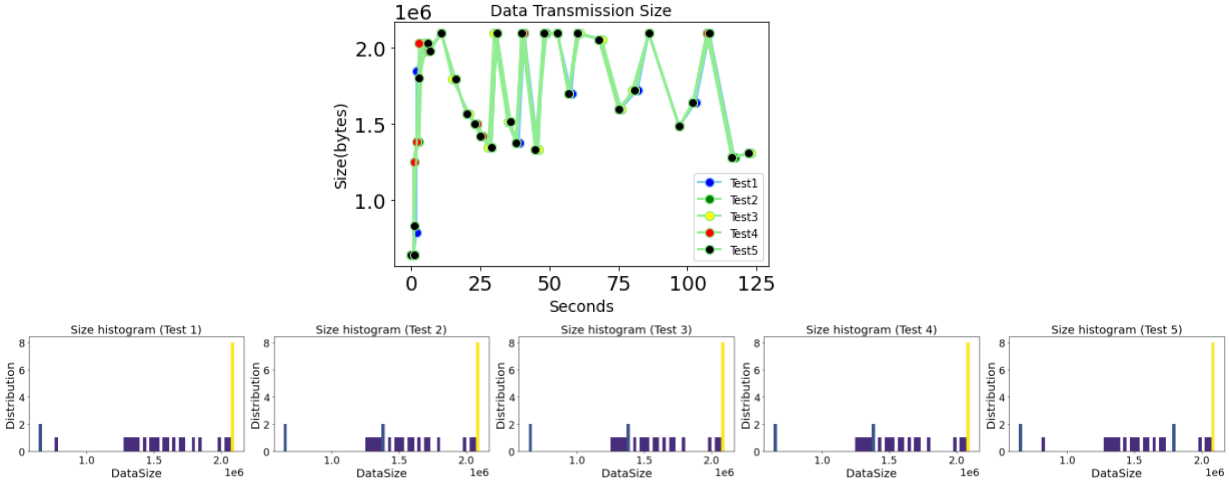

Figure 16: QUIC's Video Chuck Length Plot and Distribution

The Packet Length flow in QUIC is pretty consistent, the outputs for five test runs are similar, so I'm presenting Test 2's Packet Length plot only. As shown in Figure 17, more packets are transmitted in the first 10 seconds; then, each packets group keeps a fixed distance. The time intervals are getting bigger slowly until the connection ends. As shown in the distribution graph, most of the packet lengths are around 1400~1450 bytes. Most likely, these packets are video chucks.

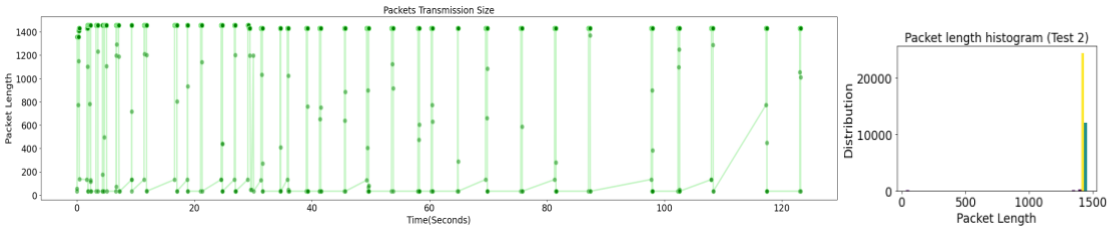

Figure 17: QUIC's Packet Length and Distribution Plot for Test 2

The last QUIC analysis step is to use Qvis to process the Netlog files. In this way, QUIC's behavior is captured. Figure 18 is showing the connection establishment stage of QUIC. The client sends an event with packet type 'initial,' packet number 1, packet size 1350, frame type 'crypto' to the server. Then the server sends back an event with packet type 'initial,' packet number 1, packet size 1350, frame type 'crypto,' and an 'ack' 1. The server shows that it acknowledged packet one from the client. The client and server send some follow up 'crypto' packets; the connection is then officially established. The packet type changed to '1RTT', and the server tells the client stream three and stream five is closed.

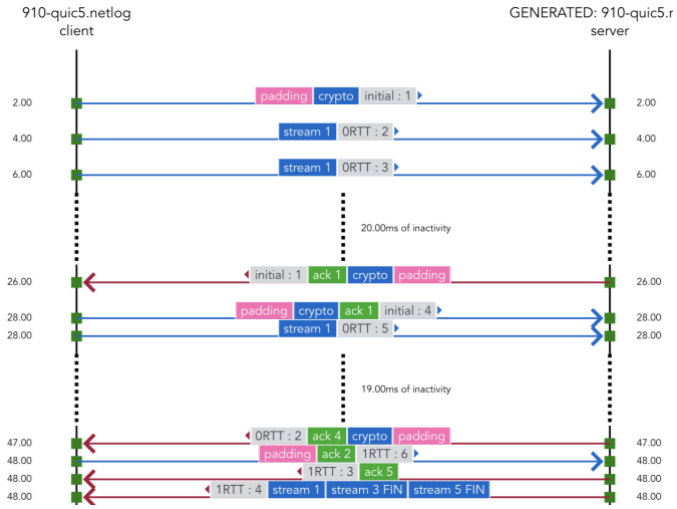

Figure 18: QUIC's connection establishment's visual presentation on Qvis.

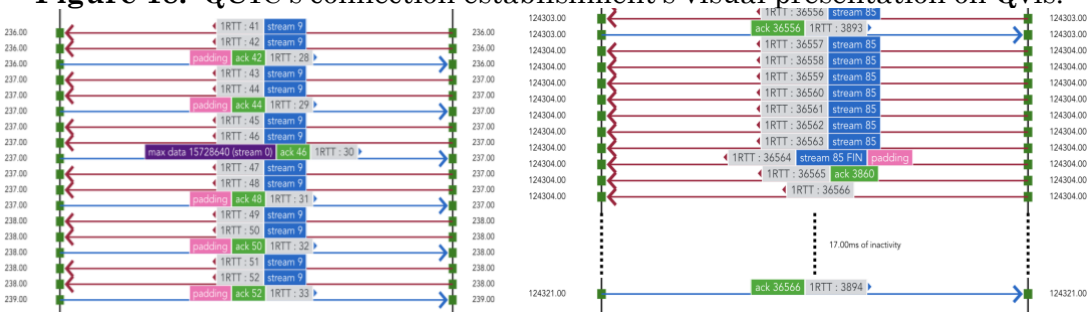

Figure 19: QUIC's transmitting and closing visual presentations on Qvis.

Figure 19 is the data transmitting stage and connection close stage. During data transmitting, the server keeps sending the '1350' size packets with packets numbers. The client acknowledged the packets by returning the same packets number as the acknowledge number. If a stream is about to close, the server will send a 'fin' to tell the client. The client also keeps telling the server in stream 0 that the max stream data size. When a QUIC connection is closing, it usually has an event with packet type '1RR', frame type 'stream,' with a flag 'fin.' The

connection is requested to be closed, but it would take several milliseconds to receive an ack, then the connection is closed.

6.2.3 Comparison between TCP and QUIC

Comparing the results of TCP and QUIC in the previous sections, the following answers according to the evaluation questions are made.

1 TCP and QUIC establish one connection for one video. In the 3 minutes video, TCP changes the destination port once or twice, QUIC used 85 streams for video downloading. QUIC supports multiplexing; the server controls the start and the closure of streams. TCP is a single thread but changes the destination port to recover the connection.
2 Both TCP and QUIC establish the connection and send 'Acknowledge' to ensure the packets are received. When one port or one stream is discarded, there will be a 'FIN' flag.
3 There is no direct connection between video downloading and packets transmission. The video chucks' size and time follow a prescribed pattern by YouTube player. Furthermore, the performance of packets is highly dependent on the transport protocol. There are approximately 1220 packets used for one video chuck.
4 QUIC's five test runs all have 85 streams in the connection. The streams can run in parallel. It is not one stream for one video chuck. Mass data transfer starts from stream 7, ends in stream 85. There are patterns in the transition. Stream with flag 'fin' usually follows with seconds of connection inactivity. The stream numbers are all odd since the results are only showing the client side's point of view, which is mainly the server sending data to clients. The even numbers streams are from client to server.
5 Comparing Figure 12 and Figure 17, QUIC is much more stable than TCP. TCP's speed fluctuates, and packets are centralized during transmission. QUIC's speed, packet size, transmission time are identical for all five test runs. The video chuck and packets clearly show a pattern. Even if one test run deviates from the track, it will be corrected immediately.
6 TCP is faster downloading video chucks. TCP is 65 Mbps, and QUIC is 55 Mbps for video chucks only. TCP is slower in packets transitions in this research, TCP is around 10.44 Mbps, QUIC is 11 Mbps. I think it is because TCP has more non-video chuck packets. Thus, TCP does have a faster speed in video downloading.
7 TCP and QUIC is different in packet transitions. The time interval for transmitting TCP packets is inconstant; they are occasionally too large or too small. The common lengths for TCP packets are 0, 600, 1000, and 1400. In contract, QUIC's packets transmission interval is distributed more uniformly. The typical lengths for QUIC packets are 0 and 1400.
8 There is no summarized data in this research, but QUIC should establish the connection faster than TCP. TCP needs to send TCP and TLS and HTTP requests to the server, while QUIC is already encrypted.
9 QUIC responses better to network fluctuations. Once an error occurs in TCP, the subsequent transmission will be positively affected by the error and retransmission packets. In contrast, QUIC is stable and can go back to normal after the error occurs.

Chapter 7 Conclusion and Reflections

7.1 Conclusion

In conclusion, I believe that TCP is faster concerning only the video downloading speed. However, in a general consideration. QUIC is more stable, simpler, and more error-resilient than TCP.

Both of QUIC and TCP ensure the data is delivered successfully. Although QUIC has no transmission speed assets, it is still worth deploying QUIC to improve the connection performance. The problem that needs to be solved is how to industrialize QUIC in a more comprehensive environment and achieve better efficiency.

Deploying QUIC for video streaming would not have a dramatic performance quality change. Only when errors occur, or the network environment changes QUIC will respond better and recover faster. Enterprises should thoroughly consider whether they want to spend to a vast budget to implement QUIC.

7.2 Achievements and Reflections

In this project, I learned and understood essential technologies, ex. QUIC, DASH. I researched lots on YouTube video performance evaluation, found tools to trace the network activities, and programmed customized tools to analyze the HTTP and Network Packets data. I verified that the customized tools are working correctly. I thoroughly evaluated and explained the test result, then presented the conclusion.

Most of the original design requirements are fulfilled, the most important achievement is that I concluded on the research topic, that is, QUIC is not faster than TCP, but it is more stable and more error-resilient than TCP.

I have researched, programmed, and enjoyed the project. I learned exciting knowledge and mastered new skills. QUIC is an undoubtedly promising transport protocol that worth more research.

7.3 Future Work

This research topic can extend in different ways. The first method, establishing the researchers' own server. The tests I have in this project are only evaluating on the client-side. If researchers establish a server, they can also examine QUIC on the server-side. Secondly, building a customized QUIC analyzer, a requirement not implemented this time, can be fulfilled in the future. The customized QUIC analyzer tool can link Netlog to HAR files, find the relation between the streams and video chucks, and how video scene changes would affect the QUIC streams. Furthermore, the research topic could extend to live video streaming. Video playback and live video streaming are different; live video streaming on YouTube can also be a meaningful research topic.

Chapter 8 References

Amit, A. (2009). Watch this YouTube Video without the Flash Player. Retrieved from https://web.archive.org/web/20171207093949/https://www.labnol.org/internet/youtube-video-without-flash-player/9016/

Bandla, K., & Obormot, O. (2020). dpkt. Retrieved from https://github.com/kbandla/dpkt

Barman, N., & Martini, M. G. (2017, 31 May-2 June 2017). H.264/MPEG-AVC, H.265/MPEG-HEVC and VP9 codec comparison for live gaming video streaming. Paper presented at the 2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX).

Bhat, D., Rizk, A., & Zink, M. (2017). Not so QUIC: A Performance Study of DASH over QUIC. Paper presented at the Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, Taipei, Taiwan. https://doi.org/10.1145/3083165.3083175

Chromium. (2014). QUIC, a multiplexed stream transport over UDP. Retrieved from https://www.chromium.org/quic

Duke, M. H., Braden, R., Eddy, W., Blanton, E., & Zimmermann, A. (2015). A Roadmap for Transmission Control Protocol (TCP) Specification Documents. RFC, 7414, 1-57.

GoogleChromeEnterpriseHelp. How to collect Chrome device logs. Retrieved from https://support.google.com/chrome/a/answer/3293821

GoogleDevelopers (Producer). (2014). QUIC: next generation multiplexed transport over UDP. Retrieved from https://www.youtube.com/watch?v=hQZ-0mXFmk8&ab_channel=GoogleDevelopers

M. Duke, M. Scharf, M. Tüxen, & Nishida, Y. (2020). TCP Maintenance and Minor Extensions (tcpm). Retrieved from https://datatracker.ietf.org/wg/tcpm/about/

M. Westerlund, L.Eggert, L. Pardue, & Nottingham, M. (2020). QUIC (quic). Retrieved from https://datatracker.ietf.org/wg/quic/about/

Mondal, A., & Chakraborty, S. (2020). Does QUIC Suit Well With Modern Adaptive Bitrate Streaming Techniques? IEEE Networking Letters, 2(2), 85-89. doi:10.1109/LNET.2020.2991867

mrname. (2020). Haralyzer. Retrieved from https://github.com/mrname/haralyzer

Netgate. (2020). Examples of using tcpdump on the command line. Retrieved from https://docs.netgate.com/pfsense/en/latest/book/packetcapture/using-tcpdump-from-the-command-line.html

Python. (2020). datetime — Basic date and time types. Retrieved from https://docs.python.org/3/library/datetime.html

Ragimova, K., Loginov, V., & Khorov, E. (2019, 3-6 June 2019). Analysis of YouTube DASH Traffic. Paper presented at the 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom).

rmarx. (2020). quiclog/qvis. Retrieved from https://github.com/quiclog/qvis

Roman, E., & Menke, M. (2018). NetLog: Chrome's network logging system. Retrieved from https://www.chromium.org/developers/design-documents/network-stack/netlog

Stephen, S. (2010). Google tries freeing Web video with WebM. Retrieved from https://www.cnet.com/news/google-tries-freeing-web-video-with-webm/

THemming. (2016). harviewer-chrome-app. Retrieved from https://github.com/THemming/harviewer-chrome-app

Thomson, M., & Iyengar, J. (2020). QUIC: A UDP-Based Multiplexed and Secure Transport.

YouTube, D. (Producer). (2018). AV1 Beta Launch Playlist. Retrieved from https://www.youtube.com/playlist?list=PLyqf6gJt7KuHBmeVzZteZUlNUQAVLwrZS

YouTubeHelp. Video and audio formatting specifications. Retrieved from https://support.google.com/youtube/answer/4603579?hl=en

YouTubeHelp. Video resolution & aspect ratios. Retrieved from https://support.google.com/youtube/answer/6375112?hl=en