QUIC - Reexamination

Final Report

Authors:
Yoav Henig (204097836), Levi Dworkin (204807044),
Ori Moldovan (204322747), Uriel Fluss (203572656)

Supervisor: PhD. Amit Dvir

September 3, 2019

Abstract
Googles QUIC transport layer protocol has become more widespread amongst the web, due to its high performance, since its experimental establishment in 2013. Nowadays, QUIC can only be used on a connection where both the client and server are Google, also known as GQUIC. This means that in order to use QUIC you must be using the chrome browser and only on a website that is Googles (i.e. YouTube, Gmail etc..). In the near future, Google plans to release an IETF compatible version of QUIC which will allow everyone across the web to enjoy its benefits.
Speaking of QUICs benefits, we would we like to reexamine QUIC and compare its performance to that of TCP over HTTP.
First, we research the preexisting TCP+HTTP and how it works in order to help us understand QUIC and how/if it actually helps reduce web latency. Next, we run QUIC and TCP+HTTP in multiple scenarios and compare their performance results.

Introduction
Our research started with the fine line between the application layer and transport layer of the internet. We divided our team into two research groups. One group to research the TCP over HTTP mechanism and the other to research QUIC.
TCP over HTTP is the default protocol and most widely used across the web. It is the stable protocol most commonly known for its three-way handshake which guarantees reliability unlike its colleague the UDP protocol, which sends messages without confirming that the message reached its destination. TCP over HTTP has the option to run with TLS (transport layer security), but this means that there is the three-way handshake and then the TLS handshake which increases web latency.
QUIC recently implemented by Google is a transport layer protocol which runs on UDP whilst providing reliability. QUIC implements several optimizations and features borrowed from existing and proposed TCP, TLS and HTTP/2 designs. These include:

- 0-RTT connection establishment: Which based on previous connections allows a client to start a new connection without a three way handshake, using a pre-shared key and already sends its request together with the client hello. This saves some RTTs which improves its performance significantly.

- Reduced head of line blocking: HTTP/2 allows multiple objects to be fetched over the same connection, using multiple streams within a single flow. If a loss occurs in one stream when using TCP, all streams stall while waiting for packet recovery. In contrast, QUIC allows other streams to continue to exchange packets even if one stream is blocked due to a missing packet.

- Improved congestion control: QUIC implements better estimation of connection RTTs and detects and recovers from loss more efficiently

We would like to focus on the performance differences between QUIC and TCP over HTTP.

ROAD START

As we mentioned in the previous paragraph, our first goal of the project was to understand the current protocol, i.e. TCP/HTTP2.0. To do so we had to understand the road to HTTP2.0, and TLS. In a meet-up with our supervisor (Dr. Amit Dvir) he suggested us to make a small investigation on those subjects and so we made presentation on each subject.
Both, the HTTP and TLS slides with notes are in the GitHub repository.

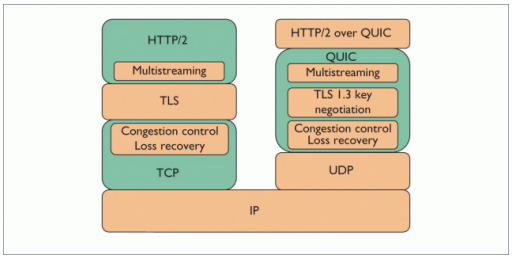

Figure 1: the differences architecture.

After we have understood the TCP/HTTP2.0 protocol, it's time to get to know QUIC protocol. For this propose we read a few articles that describe the implementation of QUIC and made a presentation in which we explained every component of QUIC. The presentation has many slides so we are not adding It to this final report as we did in the last two presentation, you can see it in our GitHub repository.

EXPERIMENTATION FRAMEWORK

Our experiment is about examine QUIC by .har files (brief explanation of .har files follows this paragraph). To do so we wrote and ran a python scripts on several computers. For observing QUIC with WI-FI we used our own computers and the cyber lab computers in four different places (the cyber lab computers were only in Ariel of course.): Giva'at Shmuel, Ariel, Halamish and Ma'alle Shomron. Each execution, which contained QUIC and TCP2.0 was in the same environment and settings to assure it will be as accurate as we can be.
For Ethernet connection we executed our tests in Ariel and Giva'at Shmuel. Again, we use the same environment and setting for the most accurate results.

HAR FILE

"HAR" file (HTTP Archive format) is a Jason formatted archive file format for logging of a web browser's interaction with a site. The specification for the HTTP Archive (HAR) format defines an archival format for HTTP transactions that can be used by a web browser to export detailed performance data about web pages it loads.
For getting the amount of the packets we asked for the size of the "entries" array, which hold all the packets that have been sent through the process.
For the size of each packet we take the "_transfersize" which presents the size of each packet.
And for the time section we did two things. First, we took the time for each packet. and second, we took the start time of the first Get and subs it from the end time of the last packet that arrived. This is the overall time of a session.
Eventually, we made an average of each parameter, so we have three parameters for every session we did:

- Average **amount** of packets.
- Average **size** of the packets.
- Average **time** of the packets to transport.

Methodology (Practical part)
After gathering enough information on the previous protocols and QUIC, this gave us the tools to analyze and compare QUICs performance to that of TCP over HTTPs. The next phase is the more practical part of our study. In this phase we are going to build the foundations for our research, allowing us to gather data and analyze it.
Firstly in order to check QUICs performance, we needed a QUIC compatible environment. In the process of building the workspace, we encountered quite a few challenges. Initially, we thought of download the full implementation of the Chromium project. After a brief examination, we realized that it was pointless to do so for two reasons:
1. Chromium project turned out to be very heavy on normal laptops.
2. The chromium project changes frequently, and the current version that actually works in the Internet is not the one in the Chromium project.
We later tried to implement the protocol using external libraries, but we quickly understood that we want to check chrome directly via the browser.
Now we need to come up with a practical way to run our tests. We use Python scripts which will allow us to easily run multiple tests for each scenario that we would like to examine, both on QUIC and on TCP over HTTP.
First, we used selenium library in order to build an automation test. We understand the only way to export the connection data into HAR file is to use proxy server, so we tried to use "browsermobproxy" library to create har files. Unfortunately, all connections needed to be with certificate but the webmobproxy created just http1.1 files. Therefore, we had to use another solution- using chrome with "haralyzer" library which allow us to open automatic chrome-browser and capture the har file.

Measurements
A scenario is comprised of four components:

- QUIC:
  - Chrome regular browser
  - Chrome incognito
- TCP over HTTP2:
  - Chrome regular browser
  - Chrome incognito

The scenarios we test are:

- VM with WIFI
- VM with Ethernet
- Linux with Ethernet

After receiving the data we extracted it into a graph configuration for easy comparison of the metrics. The results show that QUIC isnt necessary faster than TCP.
Here are the graphs about the scenarios:

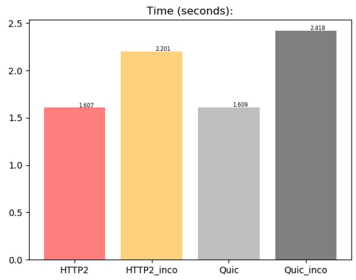

Figure 2: Linux ethernet

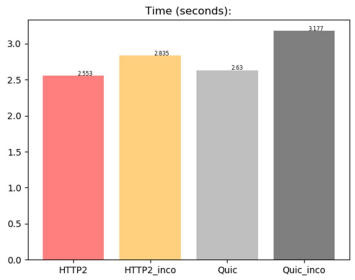

Figure 3: VM ethernet

And here the summary graphs:

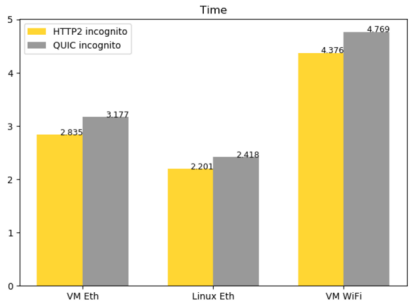

Figure 4: Summary Incognito

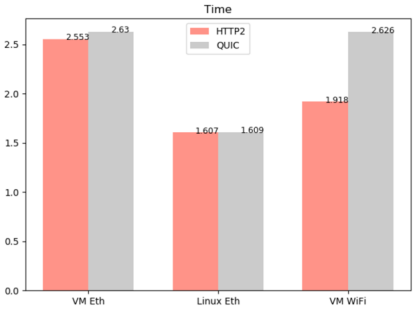

Figure 5: Summary normal

QUIC PERFORMANCE

Our tests show us a very interesting results, due to our expectations from the articles we read, the common opinion was 'QUIC will be faster than TCP/HTTP2.0'. But in fact, the results show us differently:

Field	Expected	Actual
Time	QUIC < TCP	QUIC ≥ TCP
Requests	QUIC ≈ TCP	QUIC ≥ TCP
Weight	QUIC ≈ TCP	QUIC ≈ TCP
Incognito Time	QUIC < TCP	**QUIC > TCP**
Incognito Requests	QUIC ≈ TCP	QUIC ≥ TCP
Incognito Weight	QUIC ≈ TCP	QUIC ≈ TCP

Figure 6: This table display the outcome of the data we have outputted.

As we can see from the table above, the performance of QUIC is worse than those of TCP/HTTP2.0 in the time and requests fields.
Why is it happened?
We consider a few answers for these results.
First, although we try to spread our research in four areas (Giva'at Shmuel, Ariel, etc..) it still not spread enough. In order to take this search to the next level and really explorer the new protocol, we should take the test in plenty of places so we could reach more data centers. In the article " the QUIC Transport protocol", written by Google's team, the test result came from multiple countries around the world. The way they did it is to implement the tests in Google Chrome browser and so each individual person "donates" his information to the research.
Second, as we can see from the results the best performance of QUIC is in the Linux over ethernet connection. That might happen due to less packet lost than there are in Linux WIFI connection and VM (ethernet and WIFI).
It is important to say that there are some articles that support our results and claims that QUIC is NOT faster than TCP/HTTP2.0. more interesting to see that most of the articles which support QUIC and claim it is indeed faster are made by Google's.