**A Quick Study on Google QUIC**

A Survey of Internet Data Transfer Protocol: Quick UDP Internet Connection

Hongjie Liu ([hul419@psu.edu](mailto:hul419@psu.edu)) :

Team lead, PPT (100%), presentation(100%), paper selection (100%), report (70%), review(50%)

Yuhui Lyu ([ykl5501@psu.edu](mailto:ykl5501@psu.edu)):

Implementation (100%), report writing(30%), report review(50%)

**ABSTRACT**

Today, the Internet and many applications are running on top of TCP/TLS, while still facing performance challenges such as RTTs of a handshake, head-of-line blocking, and slow evolution or updates. Facing these issues, in particular, Google has proposed the Quick UDP Internet Connections (QUIC) as the alternative of TCP. QUIC is currently serving a wide range of Google servers and is supported by several mainstream browsers like Chrome, Firefox, etc. However, due to its recency, QUIC is not standardized or implemented globally yet. In this report, we will elaborate the QUIC based on its features, strengths, and weaknesses with a comparison study between traditional TCP protocol through a broad study of current research papers. Additionally, we take the survey and study QUIC's application in the real world in the past years as well as our own toy implementation and evaluation. Also, the hot topic that whether we should migrate from TCP to QUIC will be discussed in this report.

**1. INTRODUCTION**

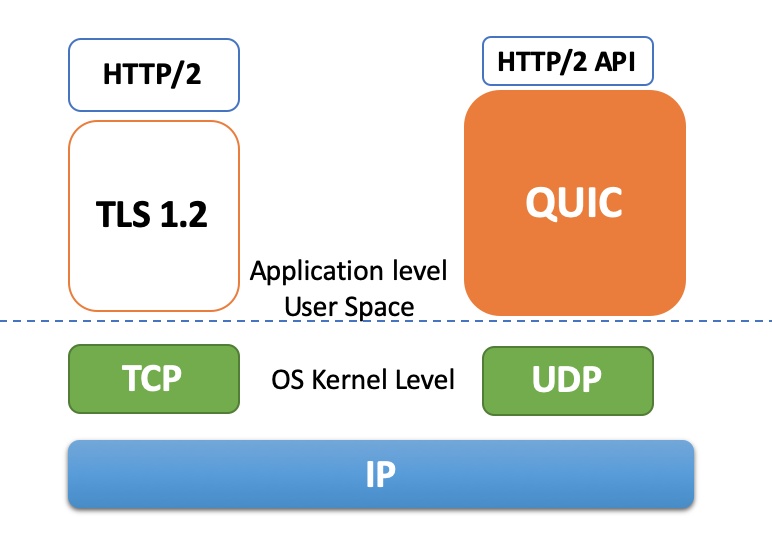
**1.1 TCP challenges**

The Internet has witnessed the development of HTTP/1.1 and HTTP/2 which represent a huge amount of portion of Internet traffic. HTTP is relying on TCP, the current de-facto protocol in the transport layer, and is used in billions of devices and the most widely adopted protocol so far. However, the increasing size and complexity of web pages are challenging the efficiency of HTTP and it also meets the bottleneck for TCP to update. One of the biggest fixes that HTTP/2 has over version 1.1 is that it solves application head-of-line blocking issues however it does not solve the head-of-line blocking problem at the TCP level. TCP delivery must follow the order and wait for retransmission of previously lost TCP segment, all segments are equally affected by packet loss. TCP is already a complex protocol and it meets the challenge of difficulty in evolution or so-called the 'protocol ossification' problem. While optimizing latency there is an additional demand to also provide an encrypted transport, typically realized by TLS on top of TCP1. While adding encrypted transport by TLS on TCP, more packets will be sent back and forth other than three-way handshake of TCP connection, this deployment is currently challenged by additional roundtrip delay, middleboxes, and legacy systems1.

**1.2 QUIC introduction**

QUIC is not exactly a recent protocol. It was developed by Google in 2012, and initial public releases in 2013. QUIC stands for Quick UDP Internet Connections, obviously, is running on top of UDP while HTTP is running on top of TCP. Although UDP is a connectionless and unreliable protocol, QUIC provides a reliable, connection-oriented, low latency, and fully encrypted transport1. The reason why QUIC relies on UDP is that using UDP enables fast deployability at user space, while modifying TCP could take years to be adopted. Google team is very smart to combine the speed and possibilities of the UDP protocol with the reliable nature of TCP. QUIC addresses several challenges in security and reliability while reduces transport and connection latency. Google has widely deployed QUIC in their servers and it accounts for almost 10% of the Internet traffic but the protocol is not standardized at IETF yet2.

QUIC shares several same features with TCP, it provides a connection-oriented, reliable, and in-order byte stream. QUIC embraces the TCP/TLS with a cross-layer approach to transport and security. As shown in Figure 1, it replaces TCP with UDP and on top of QUIC is HTTP/2 API used to communicate with remote servers. Since QUIC already handles the multiplexing and connection management, the HTTP/2 API is smaller than what is used in TCP/TLS stack.

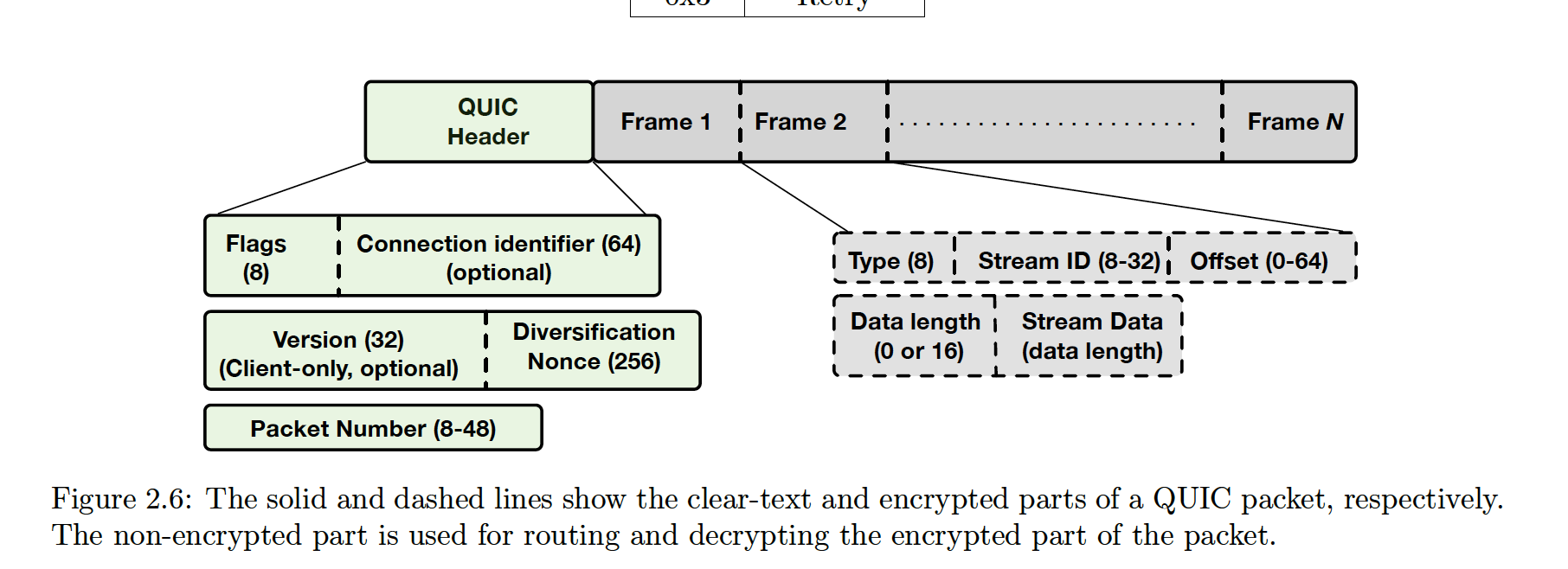


**Figure 1. QUIC architecture**

On the other hand, aside from the fact that what underneath QUIC is UDP because of its fast deployment at user space5, Google adds certain new features in QUIC: it was developed to speed up Web page loading, features 0-RTT connection, improved congestion control, and multiplexing without head-of-line (HOL) blocking. The advantages of QUIC over TCP will be discussed in detail in part 3.

**1.3 QUIC packet format**

QUIC adds its own header to the encrypted QUIC payload and then encapsulated it inside a UDP datagram before sending it 7. The head specifies a 64 bits Connection ID rather than a 4-tuple of source and destination IP addresses and port numbers used in TCP. The connection ID is used to distinguish each connection and each connection ID is independent and randomly selected by an endpoint (client). A QUIC packet is composed of a common header followed by one or more stream frames 6, and the streams are multiplex through one UDP connection. The header is unencrypted and the frames are encrypted.



**Figure 2. The solid lines represent the clear-text, the dashed lines show the encrypted parts of a QUIC packet3**

**2 RELATED WORK**

**2.1 QUIC Evolution**

QUIC was handed over by IETF in 2016 officially and the QUIC working group has been created by IETF to push QUIC from an amateur protocol to a professional one. The QUIC working group is engaged in QUIC's standardization, implementation, and deployment experiences. What sets it aside from Google's original Google-QUIC (GQUIC) which was born in 2012 is that IETF-QUIC is based on encryption and security of TLS 1.3 by default. And IETF is working on HTTP/3 which is the next generation ' HTTP over QUIC' protocol. IETF is has reached a significant milestone in 2020 and is near its completion. Google is also migrating its original GQUIC to HTTP/3 and QUIC. E.g., ss Chrome's HTTP/3 and QUIC deployment increases, its GQUIC deployment should decrease 16.

However, it is not easy to do the massive scale implementation, test, pre-release, make all the servers and devices ready for QUIC, and make QUIC/HTTP3 the de-facto mainstream protocol in a short time.

**Noted:** *In this report, we focus on the original Google QUIC rather than the IETF version, since the IETF-QUIC is not widely implemented or standardized yet.*

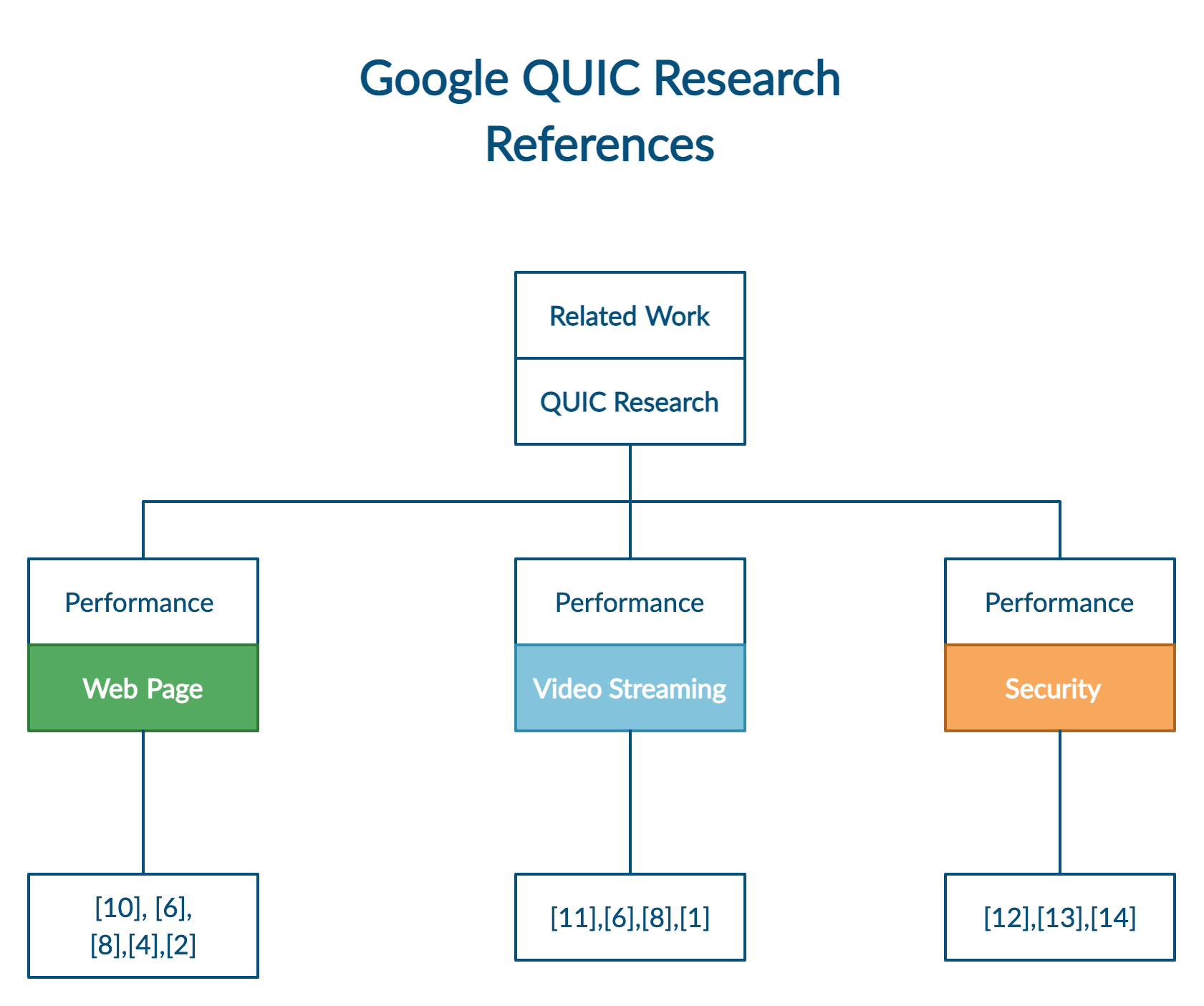
**2.2 QUIC Implementation**

Google has widely deployed QUIC in their servers as well as a few browsers (Google Chrome, Chromium, and Opera) support both HTTP/2 and QUIC 4. While QUIC can in theory be used to support any higher-layer protocol and be encapsulated in any lower-layer protocol, the only known deployments of QUIC use it for web traffic 8. Since 2017, QUIC is turned on for all users of Chrome and the Android YouTube app 6. In 2017, Google announced that around 7% of all Internet traffic were ready to use QUIC and this number was reported to be estimated around 10% in 20192.

There are also some challenges and fairness issues when comparing the released public code with what Google has deployed on clients and servers such as use code running in the wild or environment and testbeds diversity. However, QUIC is getting more and more attention and eventually has been deployed outside of Google and under the discussion of whether it will be the alternative of TCP in the future. For example, the Apple QUIC, Akamai CDN, Haskell QUIC, ngtcp2, quiche, etc. are stacks outside Google that currently implement the IETF version of QUIC 17.

**2.3 QUIC Research**

Although QUIC is relatively new, there have been several performances and optimization studies of QUIC. In this report, we will classify those researches into three different categories: data quality in web page load time, performance analysis in video streaming, and security considerations. Regarding the data quality in either web page or video streaming, most researchers care about whether QUIC demonstrates significant advantages when compared with TCP+TLS in a diverse environment and with different data content. For safety issues, some studies build certain safety models in the presence of attackers to investigate the vulnerability to attacks and robustness to safety.



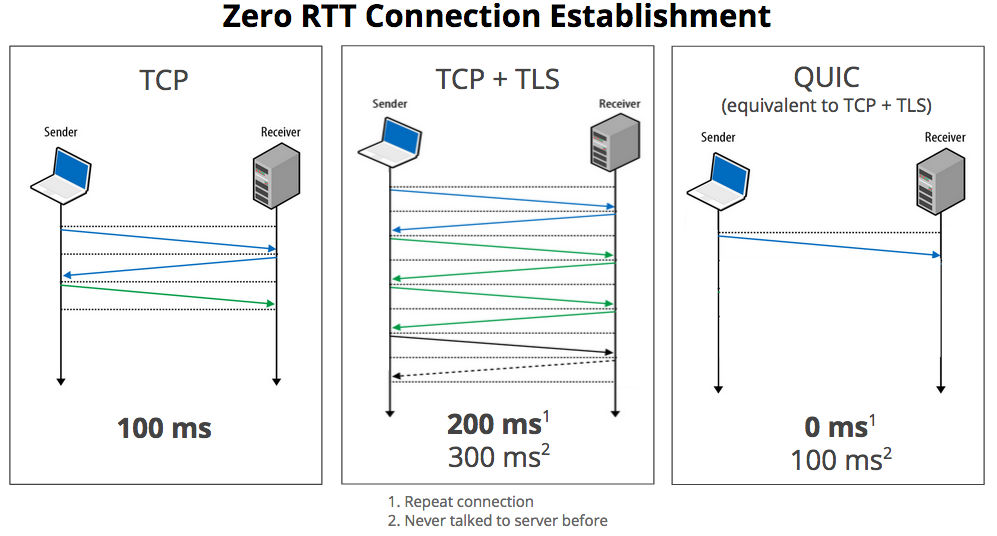
**Figure 3. QUIC research references categories.**

**3  ANALYSIS**

**3.1 TCP vs QUIC**

QUIC is built on decades of experience of TCP and it incorporates the best practices of TCP. The overall design aspirations of QUIC compared with TCP are as follows:

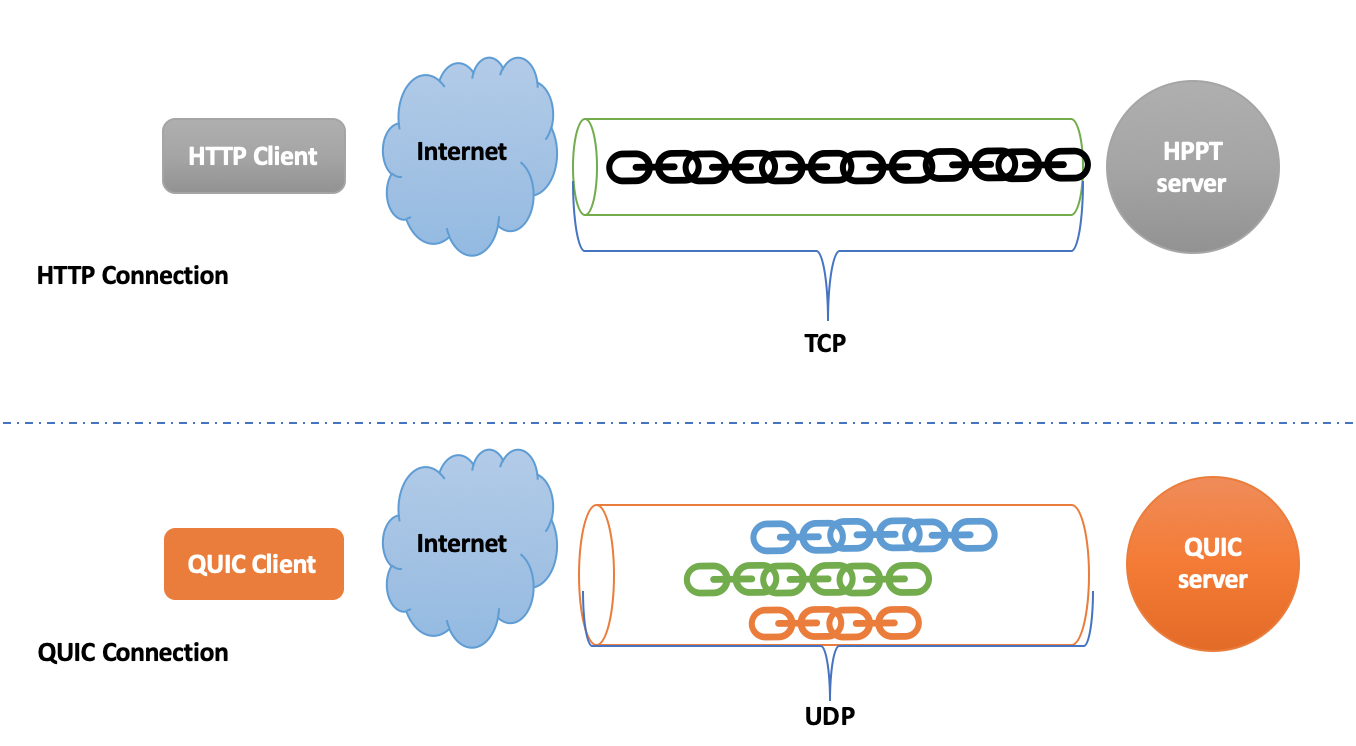
**Reduce connection establishment latency**: QUIC combines the cryptographic and transport handshake to minimize setup RTTs. QUIC reduces the connection setup time by introducing a 1 RTT handshake, unlike TCP+TLS which relies on a 3 RTT handshake (1 for connection setup and 2 for TLS delay). The server and client cache session keys by the time of the new request, there is no handshake necessary later. Thus, QUIC achieves 0-RTT for known servers. However, in a lossy network, TCP's connection delay can be a problem, while QUIC would cut this time into almost half with a quick and simple connection.



**Figure 4. Illustration on QUIC vs TCP/TLS connection establishment latency 18**

**Improved congestion control:** QUIC improves loss recovery by using unique packet numbers to avoid retransmission ambiguity and by using explicit signaling in ACKs for accurate RTT measurements. Also, QUIC supports up to 256 NACK ranges, so QUIC is more resilient to TCP 9.

**Multiplexing without head-of-line blocking**: TCP delivery must follow the order and wait for retransmission of previously lost TCP segment. While QUIC offers the same multiplexing capabilities as HTTP/2 without the issue of head of line blocking. Multiplexing streams that share the same QUIC connection do not need an additional handshake and are delivered independently. Therefore, a lost packet carrying data for an individual stream only impact the specific stream, streams without a loss will continue to be made forward in the processing9.



**Figure 5. QUIC multiplexing without head-of-line blocking**

**Forward Error Correction (FEC):** QUIC uses redundancy in the sent data stream to allow a receiver to recover lost packets without an explicit retransmission6.

**Connection Migration**: QUIC allows a seamless handover to migrate across IP addresses by using a connection ID to identify the connections, such as moving from Wi-Fi to cellular (LTE) without renegotiating the session. While TCP connection is bounded to an IP address, when the network changes, it has to have a new connection. While QUIC's connection ID is independent of IP addresses, It enables a QUIC client to keep the same QUIC connection while changing the IP/port-4-tuple.

**3.2** **QUIC in Web Page Performance**

Web page browsing has been the most commonly used application in web services. A good Internet protocol should contribute the least amount of time a user takes to load a page in a browser. Page load time （PLT） represents the time between the browser initiates a page-load to the end of the page load. Das 10 presented the first study of QUIC on web page performance in 2014. They investigated 500 web pages across at least 100 network configurations of bandwidth and minimum RTT, and compared HTTP/1.1, SPDY, and QUIC protocols. They demonstrate that QUIC has lower PLT than HTTP/1.1 and SPDY over low bandwidth and high RTT links. They also revealed several limitations of QUIC. First, QUIC code is not fully optimized due to recency. Second, there is additional overhead for always-on-fashion encryption in QUIC. Moreover, they observed that QUIC is even 80% slower than TCP over high bandwidth.

In Google's paper 6 of Internet deployment of QUIC in 2017, they estimated that 7% of the Internet is QUIC. They found that QUIC's handshake latency is insensitive to RTT due to the fixed latency cost of a 0-RTT handshake, which constitutes 88% of all QUIC handshakes6 in desktop connections. While, for mobile connections, QUIC only achieves a 68% 0-RTT handshake rate on average. This is due to the mobile client's changing networks and diverse server configurations across different data centers. Similar to Das10, Google also found that QUIC performs worse than TCP over high bandwidth.

Kakhki, et al. 8conduct an ad-hoc evaluation on QUIC and run QUIC in a large number of environments including traditional desktop, mobile, and proxy environments. They found that QUIC generally outperforms TCP and has a smaller PLT mainly due to QUIC 0-RTT connections. However, they also found QUIC's poor performance for a large number of small-sized objects and diminishing benefits of 0-RTT at 3G scenarios. They also found prior works of QUIC's poor performance at high bandwidth are misleading due to lack of calibration.

Similar findings are shown in Biswal, et al.4 They found that QUIC loads pages quicker than HTTP/2 under poor network conditions and its performance improves with the increase of object size. However, QUIC offers no significant improvement for web pages with many small-sized objects.

Thomas, et al.2 analyzed the behavior of QUIC over a satellite communication system and concluded that TCP has better PLT with transparent proxying when compared with Google QUIC (GQUIC). With public SATCOM access, for a large web page, QUIC requires twice the longer PLT than TCP due to the non-delegated Congestion Controller.

**3.3 QUIC in Video Streaming**

Video streaming is another important and popular area in web applications. In Cisco's white paper of global mobile data traffic forecast, they predicted that IP video will be 82% of all IP traffic in 2020, up from 70% in 2015 111.

Google 6 conduct the evaluation on mobile video streaming app YouTube in 2017 and found that QUIC reduces re-buffer rates of YouTube playbacks by 18% for desktops and 15.3% for mobile users. 85% of QUIC desktop connections for video playback benefits from a 0-RTT handshake and the rest benefits from 1-RTT handshake 6, while for the mobile app only 65% of QUIC connections benefit from 0-RTT due to YouTube's own optimization for server connection while a user's searching.

Kakhki, et al. 8 found that QUIC outperforms TCP in quality of experience (QoE) metrics for video streaming, but only for high-resolution video. At medium and low resolution there is no significant difference.

Cook, et al.1 evaluated demonstrated that QUIC does not show obvious benefits for end-users in the case of stable and reliable networks. QUIC outperforms HTTP/2.0 over TCP/TLS in unstable networks.

**3.4 Security**

The first analysis of QUIC security is presented in Fischlin, et al. 12 where they proved QUIC's cryptographic strength as an adequately secure multi-stage key exchange protocol.

A later analysis in Lychev, et al 13 in 2015 shed light on QUIC's security weaknesses. They introduced a security model to evaluate QUIC's security performance and revealed that QUIC is vulnerable to certain bit-flipping and replay attacks and can be switched back to TCP or a connection failure with those adversaries. QUIC does satisfies a security definition suitable for performance-driven protocols however does not satisfy the traditional notion of forward secrecy, provided by some TLS models. The authors show that using some simple attacks can prevent QUIC from achieving minimum latency and they have implemented those attacks and show the practicality.

In the same year, Jager, et al 14 proved that QUIC is vulnerable to Bleichenbacher attack. Since the singed values are independent of the client's connection request. Therefore, the attackers are able to pre-compute the signature long before the client initiating a connection.

**3.5 QUIC Challenges**

QUIC is getting more and more attention while still remains several weaknesses that needed to be discussed. First, it has security weaknesses due to its vulnerability to certain attacks. QUIC's technique to minimize latency may not be useful in the presence of malicious parties 13. Second, its implementation is far less mature than HTTP/2 over TCP/TLS in the real world. QUIC is not only deployed within Google servers and certain browsers, most prior evaluations are specific and limited in their scope and do not take the ecosystem into consideration 15. The IETF working group is still working on its standardization. There are only a few studies like 1 that did a large-scale analysis of QUIC and its traffic share. Thus, there are still limitations for academic understanding due to both performance and security issues. Finally, QUIC's low-latency performance is affected by diverse environments and configurations such as web pages with too many small objects, low-resolution video streaming, etc. as we have discussed early in this section. The question that whether QUIC always outperforms TCP is still ambiguous over different scenarios and requires a lot of future work.

**4 IMPLEMENTATIONS (Bonus)**

Since QUIC is an experimental protocol and there is no finalized draft of it, there are only a few websites that support QUIC. Fortunately, Google servers support HTTP-over-QUIC (renamed to HTTP/3), and most Google services such as Google Search, YouTube, and Google Play can be connected by QUIC. Chromium, an open-sourced browser project sponsored by Google, also provides a sample QUIC server and client in its repository. Chrome browser also enables QUIC and HTTP/3 by default.

**4.1 Implementation 1**

In the first implementation, we built a simple, toy QUIC server using the sample server provided in Chromium. Then we will connect a client to the toy QUIC server and request a web page (an HTML file copied from www.example.org).

First, we download the source code of Chromium and build the binaries for the QUIC server and client. To run the server, we also need a valid certificate, which can be generated by a provided script file. The generated CA certificate should be added to the system certificate store. Next, for simplicity's sake, we download a copy of www.example.org and serve it on our toy server. The final step is to use a client to connect to the server and request the web page. Both the server and client are connected by Wi-Fi.

Our first option for the connection test is a sample QUIC client provided in the Chromium repository. As shown in Figure 6, we used the built-in client to connect to the server and request a web page. Second, in order to better evaluate the performance of QUIC and visualize results, we used the Chrome browser to request the web page served locally in the toy server (shown in Figure 7). In Figure 7, we could realize that QUIC protocol could also help encrypt and authenticate the web page to keep the connection secure.

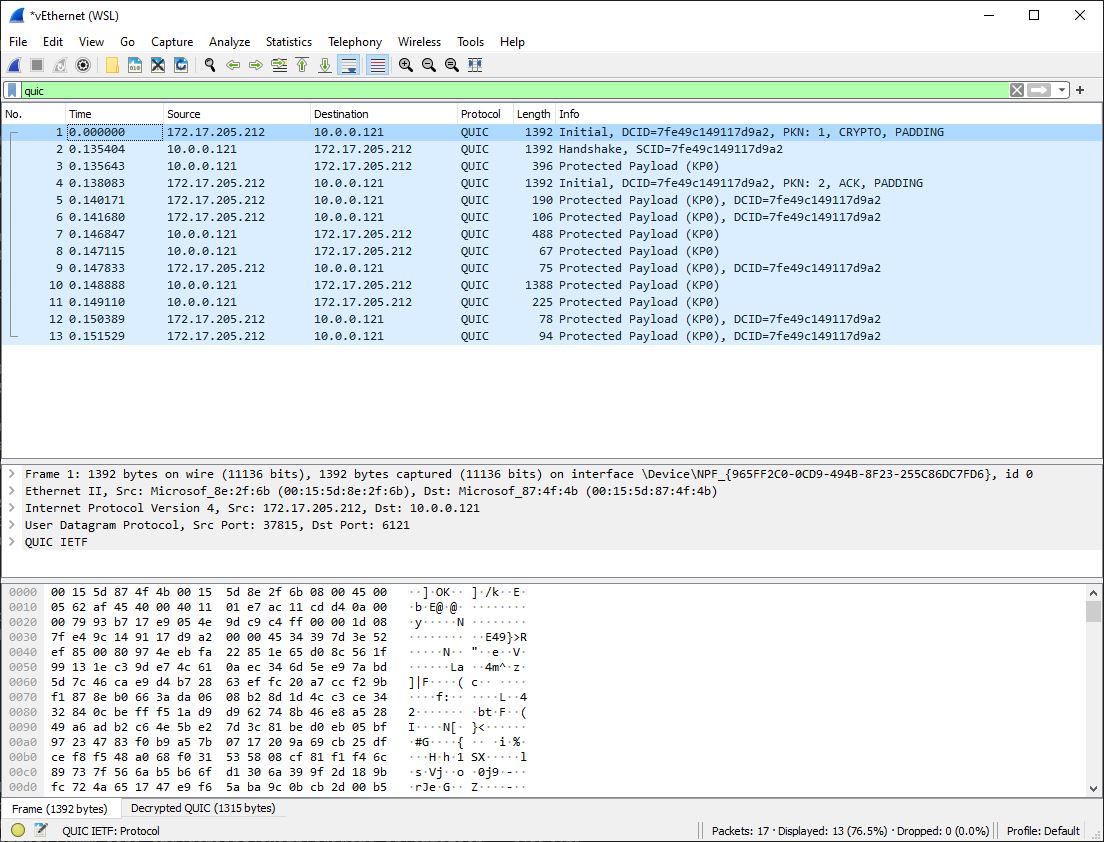


**Figure 6**

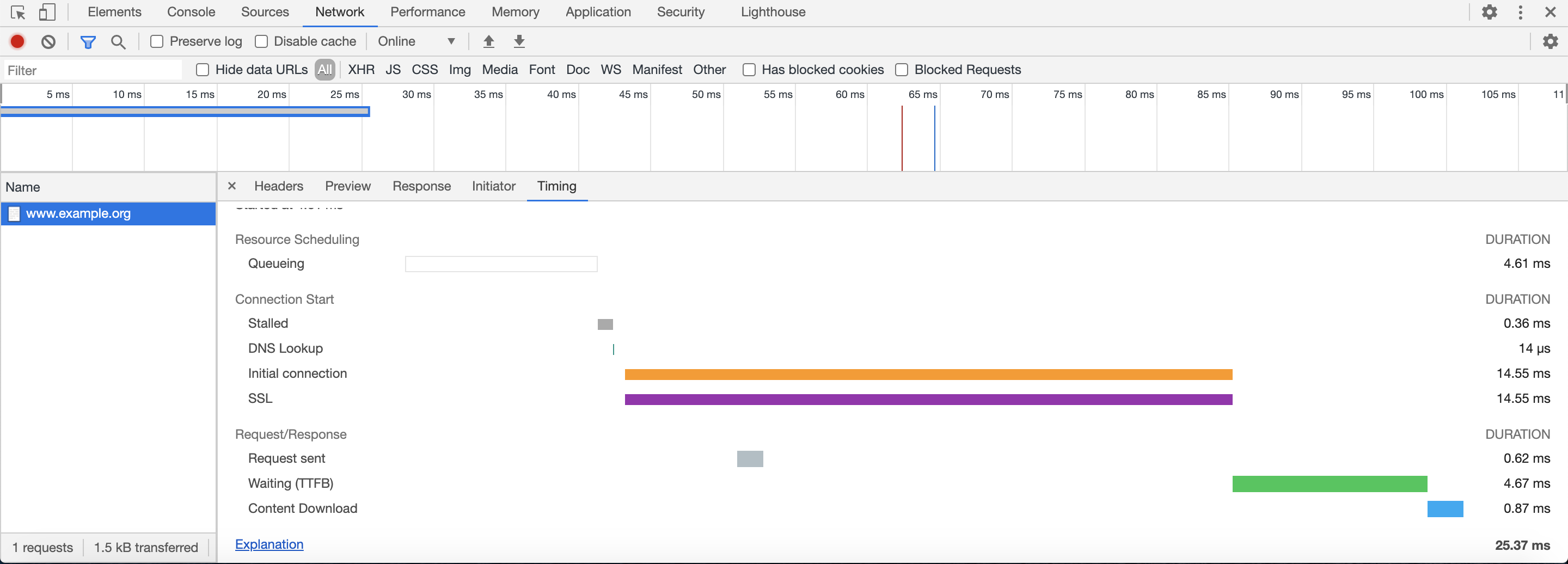
Graphical user interface, application

Description automatically generated

**Figure 7**



**Figure 8**



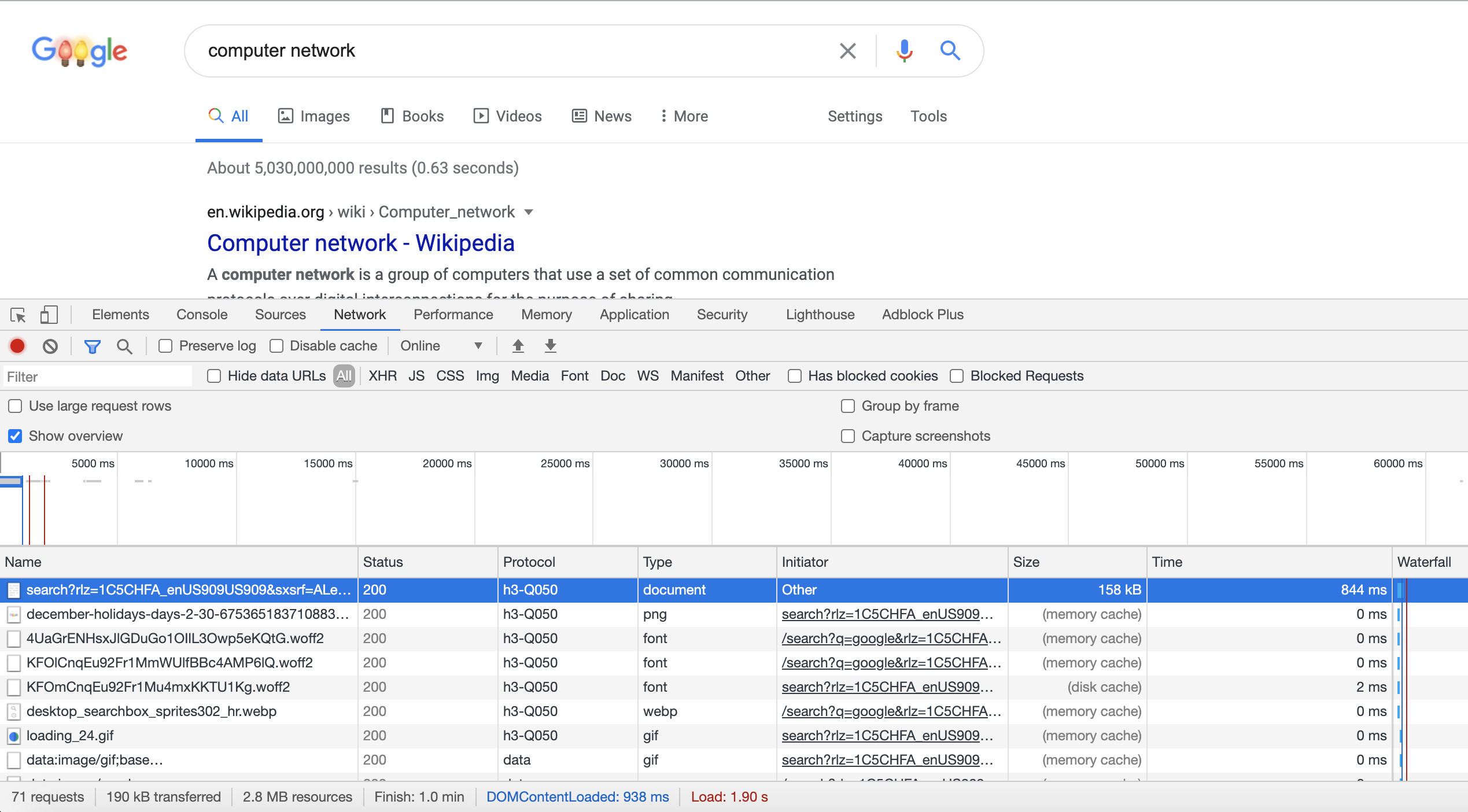
**Figure 9**

We also used Wireshark to monitor all network traffic (Figure 8). When the client requests a web page, we notice there are packets transmission between the server and the client by QUIC protocol. We could observe that QUIC only takes 1-RTT time to establish a connection.

We used developer tools in Chrome to evaluate the performance of QUIC protocol (Figure 9) and there are two important metrics: Time to first byte (TTFB) and page load time (PLT). TTFB stands for the time from making the HTTP request to the first bytes be received by the browser, which indicates the response time of a website. Page load time is the time a browser takes to load all elements of a page and show them on the screen.

**4.2 Implementation 2**

By default, Chrome and Google services support QUIC. If you use Chrome to visit a website like Google or YouTube, the connection will be established using QUIC. In Figure 10, we visit Google, and all files on the page are transferred using HTTP/3, which uses QUIC protocol.



**Figure 10**

A picture containing chart

Description automatically generated

**Figure 11**

We want to compare the performance of QUIC with that of TCP. In Chrome, we could manually enable/disable QUIC. We want to evaluate their performances based on two metrics we introduced before: TTFB and PLT. We want to check which protocol is faster to set up a connection with a website (shorter TTFB) and load the webpage on the screen (shorter PLT).

We did our test on a laptop connected by Wi-Fi. We test on 4 different websites using Google services and record their load time and TTFB. The results are shown in the following charts.

**Chart 1 Chart 2**

We realized that there is no significant advantage in terms of TTFB and load time. Most time, QUIC is a little bit faster than TCP, but the gap is very small. Sometimes, the performance of TCP is even better than that of QUIC. This experiment is not very rigorous and there are lots of factors that may affect the evaluation, such as the network environment, the size of a web page, and the number of elements on a page. We also ignore the effects of loss rate and bandwidth.

**5 CONCLUSION**

For TCP and QUIC, with the strengths and weaknesses of both protocols, there is no reason to abort either. Thus, choosing what type of future mainstream protocol should be applied depends on what type of data we have and what tradeoffs are there. In light of those findings we have read, QUIC paves the way for future application and transport layer landscape. QUIC traffic share is likely to increase in the future when being largely enabled at a wide range of infrastructures 1. With the emerging of 5G technology in new applications and standardization of HTTP/3, QUIC has a promising future and there will be more and more web services based on QUIC, and more and more players get involved with QUIC in the future.

Due to the diversity of network environments and the potential security challenges, there is still a wide space for both industry and academy to explore the possibility of a broad application of QUIC in the future. We believe that Google will take its own promises in future Internet mainstream protocol and make QUIC either take place of TCP or live harmoniously with TCP.

**Limitation:** *Due to the limited time and scope of this survey, we have neither dig into the optimization techniques of QUIC nor the evolution of the IETF version of QUIC in detail.*

**6 REFERENCES**

1. Rüth, Jan & Poese, Ingmar & Dietzel, Christoph & Hohlfeld, Oliver. (2018). A First Look at QUIC in the Wild. 10.1007/978-3-319-76481-8\_19.
2. Thomas, Ludovic & Dubois, Emmanuel & Kuhn, Nicolas & Lochin, Emmanuel. (2019). Google QUIC performance over a public SATCOM access. International Journal of Satellite Communications and Networking. 37. 10.1002/sat.1301.
3. Puneet Kumar, QUIC - A Quick Study, arXiv:2010.03059v2 [cs.NI] 10 Oct 2020
4. Biswal, Prasenjeet & Gnawali, Omprakash. (2016). Does QUIC Make the Web Faster?. 1-6. 10.1109/GLOCOM.2016.7841749.
5. QUIC vs TCP+TLS — and why QUIC is not the next big thing, https://medium.com/codavel-blog/quic-vs-tcp-tls-and-why-quic-is-not-the-next-big-thing-d4ef59143efd
6. Adam Langley, Alistar Riddoch, et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment, August 2017 Pages 183–196.
7. Amit Srivastava, Master Thesis, Performance Analysis of QUIC Protocol under Network Congestion, Worcester Polytechnic Institute, May 2017.
8. Kakhki, Arash & Jero, Samuel & Choffnes, David & Nita-Rotaru, Cristina & Mislove, Alan. (2017). Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. 290-303. 10.1145/3131365.3131368.
9. QUIC at 10,000 feet. https://docs.google.com/document/d/1gY9-YNDNAB1eip-RTPbqphgySwSNSDHLq9D5Bty4FSU/edit
10. Das, Somak. (2014). Evaluation of QUIC on web page performance.
11. Global-2020 Forecast Highlights, https://www.cisco.com/c/dam/m/en\_us/solutions/service-provider/vni-forecast-highlights/pdf/Global\_2020\_Forecast\_Highlights.pdf
12. Fischlin, Marc & Günther, Felix. (2014). Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. Proceedings of the ACM Conference on Computer and Communications Security. 1193-1204. 10.1145/2660267.2660308.
13. Lychev, Robert & Jero, Samuel & Boldyreva, Alexandra & Nita-Rotaru, Cristina. (2015). How Secure and Quick is QUIC? Provable Security and Performance Analyses. 2015. 214-231. 10.1109/SP.2015.21.
14. Jager, Tibor & Schwenk, Jörg & Somorovsky, Juraj. (2015). On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption. 1185-1196. 10.1145/2810103.2813657.
15. Cook, Sarah & Mathieu, Bertrand & Truong, Patrick & Hamchaoui, Isabelle. (2017). QUIC: Better for what and for whom?. 1-6. 10.1109/ICC.2017.7997281.
16. The state of QUIC and HTTP/3 2020, https://www.fastly.com/blog/state-of-quic-and-http3-2020
17. https://github.com/quicwg/base-drafts/wiki/Implementations
18. A QUIC update on Google's experimental transport, https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html