About the Design of QUIC Firefox Transport Protocol

Vraj Pandya
Lamar University, Beaumont, Texas, U.S.A.
vpandya@lamar.edu

Stefan Andrei
Lamar University, Beaumont, Texas, U.S.A.
Stefan.Andrei@lamar.edu

Abstract

QUIC (Quick UDP Internet Connections) Chrome is an experimental transport layer network protocol designed by Jim Roskind at Google, initially implemented in 2012 and announced publicly in 2013. One of the QUIC's goals is to improve performance of connection-oriented web applications that are currently using the Transmission Control Protocol (TCP). To do that, QUIC achieves a reduced latency and a better stream-multiplexing support to avoid network congestion.

In 2015, Firefox Mozilla started to work on an equivalent QUIC transport protocol for their browser. This idea was motivated by the differences between Chrome and Firefox. Despite the fact that Mozilla Firefox and Google Chrome are both web browser engines, there are some significant differences between them, such as file hierarchy, open source policies (Firefox is completely, while Chrome is only partial), tabs design, continuous integration, and more.

Likewise QUIC Chrome, QUIC Firefox is a new multiplexed and secure transport based on User Datagram Protocol (UDP), designed from the ground up and optimized for Hyper-Text Transfer Protocol 2 (HTTP/2) semantics. While built with HTTP/2 as the primary application protocol, QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. In addition to describing the main design of QUIC Firefox, this paper will compare Firefox with QUIC Firefox. Our preliminary experimental results support that QUIC Firefox has a faster execution time, less latency time, and a better throughput time than the traditional Firefox.

Keywords: User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Hyper-Text Transfer Protocol (HTTP)

1. Introduction

QUIC (Quick UDP Internet Connections) Chrome is an experimental transport layer network protocol designed by Jim Roskind at Google initially implemented in 2012 and announced publicly in 2013 (Cyr et al., 2013). QUIC supports a set of multiplexed connections between two endpoints over the User Datagram Protocol (UDP). QUIC was designed to provide security protection equivalent to TLS/SSL, along with reduced connection and transport latency, and bandwidth estimation in each direction to avoid congestion. QUIC's main goal is to improve perceived performance of connection-oriented web applications that are currently using Transmission Control Protocol (TCP). It also provides a venue for rapid iteration of congestion avoidance algorithms, placing control into application space at both endpoints, rather than (the relatively slow to evolve) kernel space (Willis; 2013). Since then, the research community investigated this new protocol. For example, paper (Carlucci et al., 2015) investigates QUIC, to check its reliability when reducing the Web Page retrieval time. They found that QUIC reduces the overall page retrieval time with respect to HTTP in case of a channel without induced random losses and outperforms SPDY (explained in Section 4) in the case of a lossy channel. However, when the FEC (explained in Section 3) module is enabled, it worsens the performance of QUIC.

Wireshark is the world's foremost and widely-used network protocol analyzer. It allows the view of the details of a network at a microscopic level and is the 'de facto' standard across many

commercial and non-profit enterprises, government agencies, and educational institutions (Combs et al., 2017).

In 2015, Firefox Mozilla started to work on an equivalent QUIC transport protocol for their browser. This idea was motivated by the differences between Chrome and Firefox. In fact, there is a significant competition between browsers, as illustrated in (Digital Trends Staff; 2016). While Mozilla Firefox and Google Chrome are both web browser engines, there are some significant differences between them, such as file hierarchy, open source (Firefox is completely, while Chrome is only partial), tabs design, continuous integration, and more. In addition, Mozilla Firefox and Google Chrome use entirely different code libraries (specifically Layout Engines) to render and handle web content. Another difference is that Firefox 'autocomplete feature' intelligently prioritizes substring matches against your history/bookmarks/page titles over search results. Apparently Chrome's 'autocomplete feature' is significantly less predictable than its Firefox counterpart, particularly with respect to interpreting the input as a substring, and then matching that against the strings history (Jasuja et al.; 2017).

QUIC (Quick UDP internet connection) is a transport layer protocol written on UDP (User Datagram Protocol). The primary attraction of this protocol is Zero RoundTrip Time (0-RTT) connection setup. This means that a QUIC client can establish a socket connection to a QUIC server in the first packet it sends, while maintaining the security of the entire traditional combination of the TCP and the Transport Layer Security (TLS) stack. Hence, QUIC is a multiplexed protocol which may transfer multiple files over the same connection (Official QUIC website; 2016). Although QUIC is a fairly recent protocol, there are some notable contributions in the research community about it. For example, (Wang et al.; 2013) put forth the QUIC-TCP congestion control algorithm, and a new queuing-delay based carrier sense multiple access (CSMA) scheduling scheme. Their technique can be implemented asynchronously without message passing among network nodes.

Figure 1 describes the TLS vs. QUIC protocol user level stack in the context of application layer and transport layer.

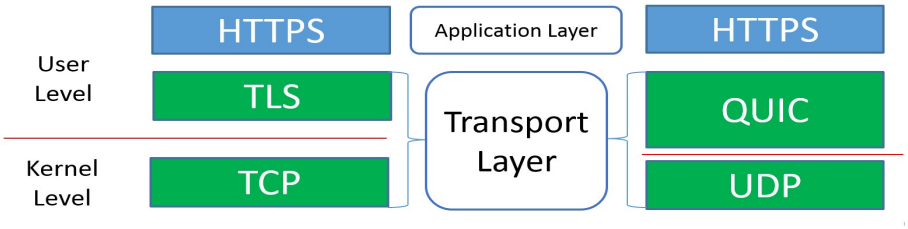

Figure 1. TLS vs QUIC protocol stack

QUIC addresses many network problems such as the Head Of Line (HOL) blocking as well as the TCP reconnection over a subnet/network change. In addition, QUIC has many features such as connection IDs, which can overcome the challenge of changing networks. In this way, if someone switches from a WIFI network to a cellular network, the connection to the server will not be broken or lost. Paper (Langley and Chang, 2016) described the QUIC crypto protocol, representing the part of QUIC that provides transport security to a connection. The QUIC crypto protocol is now replaced by TLS 1.3. Currently, QUIC provides security of TLS 1.3 (in an experimental stage), considered the highest security standards for the communication protocols (Valsorda; 2016).

The textbook (Grigorik, 2013) presents an overview of the various types of networks (WiFi, 3G/4G), transport protocols (UDP, TCP, and TLS), application protocols (HTTP/1.1, HTTP/2), and APIs available in the browser (XHR, WebSocket, WebRTC, and more) to deliver the best—fast, reliable, and resilient—user experience. The details of HTTP/2 protocol are described in (Belshe, Peon, Thomson, 2015).

Figure 2 describes a sequence diagram using QUIC VS TLS handshake protocol. The right side of Figure 2 shows the seven steps of calls and returns until a HTTP `Get()` method is successfully implemented using a TLS handshake. In contrast, on the left side of Figure 2, we see the implementation of HTTP `Get()` method using a single call of QUIC handshake.

Firefox is a completely open source browser with a tremendous community support. The latest version of Firefox supports TLS 1.3 protocol in an experimental stage. The primary purpose of this study was integrating the QUIC protocol in the Firefox web browser. The source code of this software product is as large as 650MB.

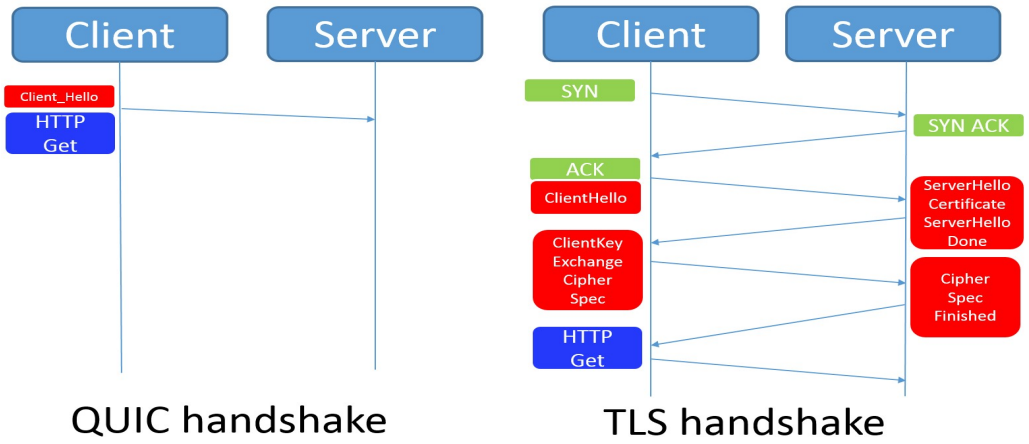

Figure 2. QUIC vs TLS handshake protocol

2. Motivation of QUIC Firefox

The research and educational communities are usually at the forefront of the methods that are being developed. The main reason for this experiment was to encourage these communities using QUIC for testing purpose and for offering their comments back to the QUIC developers.

In addition, QUIC is a user level protocol which means that the protocol does not need the support of the operating system to be upgraded. Instead, it can be upgraded with just some software upgrades. This is the most used approach by the researchers to try out new ideas.

Moreover, Firefox is a completely open source browser. That makes it a suitable candidate for implementing a new experimental network protocol based on the Creative Common License.

The main purpose of these experimental protocols is to try out new methodologies such as multiplexing, connection IDs and 0-RTT secure connection. All these methodologies can improve the throughput of the website accessed using QUIC Firefox.

The Internet Engineering Task Force (IETF) encourages these types of experiments because the protocols used by these technologies need to be standardized for a more general use.

The TLS 1.3 has many features inspired by QUIC. As a historical note, Firefox is actually the first browser to enable TLS 1.3. On a personal note of motivations to write this paper, the first author of this paper (Mr. Pandya) is one of the contributors of QUIC Firefox (Bugzilla@Mozilla; 2015) for solving a software bug.

3. The main QUIC Firefox features

The QUIC Firefox has many features worth to be mentioned in this paper, but we only present here a selection of them, such as:

1. Very Flexible crypto graphical suite: QUIC has the latest crypto suite available. QUIC functions are based on TLS 1.3 crypto suite, which is just a draft and/or experimentally deployed in

a few browsers over TCP. Since QUIC is a user level protocol, the crypto suite can be updated almost as soon as they are proposed and proved to be effective.

2. Pluggable congestion control/flow control: In many studies, research works, and observations, it was found that there is no optimal way to implement strategies for congestion/flow control in computer networks. An algorithm has to be selected based upon its strength and weakness. QUIC offers a pluggable congestion control mechanism. This means that we can switch congestion control algorithms for QUIC on the fly. For example, we can select one ACK policy for some traffic source and another ACK policy for a different traffic source.

3. Seamless network transition: While switching networks, QUIC can adapt to a network or a subnet switch (illustrated in Figure 3). This means that if the IP address of the device is changed, then the QUIC connection is not broken or lost. Unlike the TCP protocol where the connection is defined by the IP address and a port number, QUIC connections are defined by a connection ID. Whenever a network switch occurs, QUIC detects it and sends a piggybacked notification to the other party by indicating the new IP address and connection ID. In this way, the communication can resume normally. In contrast, TCP a new connection has to be established.

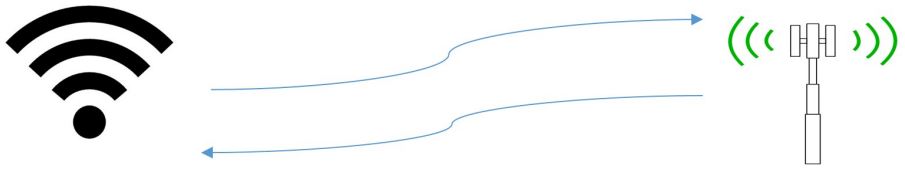

Figure 3. An example of a network switch

4. Flexible Forward Error Correction (FEC): The forward error correction is a mechanism in which it is assumed that the channel is a "lossy" entity, which means the chance of packets getting lost in this channel is very high and the unnecessary information is discarded. To mitigate that factor, a FEC packet is generated by XOR-ing a number of packets (as illustrated in Equation 1).

$$fec = p_1 \oplus p_2 \oplus p_3 \oplus p_4 \oplus p_5 \ldots \oplus p_n$$
$$p_n = fec \oplus p_1 \oplus p_2 \oplus p_3 \oplus p_4 \ldots \oplus p_{n-1}$$

Equation 1. Formation and use of FEC packet

The FEC mechanism is flexible, which means that the protocol adapts the number of packets that need to be incorporated in one FEC packet according to the packets that are lost. The number of packets in one FEC packet is inversely related to the number of packets lost (illustrated in Figure 4).

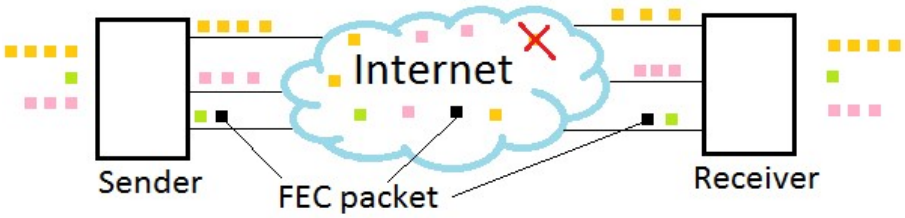

Figure 4. QUIC Streams and FEC packets

5. Multiplexed streams: After establishing a connection, each QUIC connection forms a stream for every needed resource. Streams can be represented by a two way communication channel abstraction. Each and every stream has a stream ID. All the data and ACK packets are sent through as QUIC streams (illustrated in Figure 5).

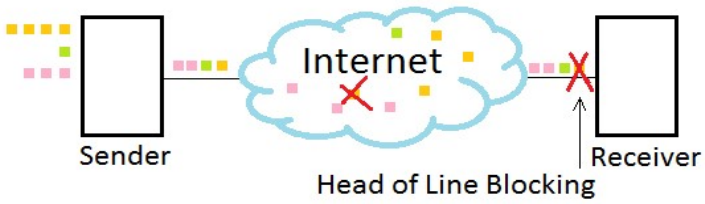

Figure 5. The head-of-line blocking in TCP/TLS

Considering Figure 5, the Head-of-line blocking (HOL blocking) in computer networking is a performance-reducing factor that occurs when a line of packets is held up by the first packet. Examples include input buffered network switches, out-of-order delivery and multiple requests in HTTP pipelining. One way to overcome this limitation is by using Virtual Output Queues. Only switches with input buffering can suffer HOL blocking. With sufficient internal bandwidth, input buffering is unnecessary; all buffering is handled at outputs and HOL blocking is avoided. This no-input-buffering architecture is common in small to medium-sized Ethernet switches.

4. Experimental Results

QUIC is tested under various laboratory condition as well as real time conditions by research organizations around the world (including Google itself is used to search information about Google). Researchers usually compare QUIC with another multiplexed protocol, that is SPDY. The SPDY specification is described in paper (Belshe and Peon, 2016).

Megyesi et al. (2016) represents a quite comprehensive study about the performance of QUIC, SPDY and HTTP particularly about how they affect page load time. The authors found that none of these protocols is clearly better than the other two and the actual network conditions determine which protocol performs the best. In addition, (Megyesi et al., 2016) contains a reference guide about the use of QUIC.

Here is an important note on FEC and its performance. Google conducted performance tests to check the effectiveness of FEC in working conditions (QUIC FEC v1; 2016). Some conclusions are eye-opener regarding the performance of FEC in a real life scenario.

Counter-intuitively, the performance of the transmission protocol did not increase in the presence of FEC. But its performance decreased in some cases and the bandwidth is wasted. However for YouTube videos, the median and mean join latency increased the re-buffer rate. In case of the Chrome web browser, FEC improved mean latency slightly and degraded median latency slightly, though neither change is statistically significant. However, FEC could be sometimes slow, but not statistically significantly slower than an aggressive TLP.

This is one of the reasons why FEC v1 QUIC design team considered to deprecate the FEC protocol (Google QUIC; 2016).

5. Comparison between traditional Firefox and QUIC Firefox

Among the contributions done and the tools used by QUIC Firefox, we enumerate the following:

1. We forked the current source trees of Firefox and QUIC protocol from official sources and using the version control systems to keep it up-to-date as we developed the code. To get that source code and to keep the source code up-to-date we used `mercurial` and `git` version control systems.

2. We used meta-build systems like `mozbuild`, `cmake`, `gyp` and `gn` to generate the build files.

3. The build files were based on `make` files and `ninja` build tools. The primary text editor was `vim`. We extensively used `gdb` to understand the structure and the flow of the source code and then debug the new code. We used standard `gcc` and `clang` compilers to compile the code.

4. We used chrome's "`netinternals`" feature to analyze the quic traffic that flows through chrome browser.

5. We can use wireshark to analyze each and every packet that goes through the network.

6. We incorporated QUIC "sockets" into Firefox core network stack.

As discussed in the previous sections, QUIC Firefox can establish connections with 0-RTT. This is equivalent to say that QUIC will inherently decrease the latency of content loading by 500mS. The QUIC stream also executes in parallel to load multiple objects concurrently. Because of QUIC streams, the objects can avoid head of line blocking. In addition, the explicit negative acknowledgement (NACK) packets can help resend only the lost packets. Hence, not all the packets that were sent after one packet were lost. All these differences make QUIC Firefox faster than the traditional Firefox.

There are many features of this kind of protocols that can be implemented. These protocols can be later standardized. SPDY is a good example of such occurrence because many of the features implemented by SPDY are now implemented in HTTP/2.0. Among these features, we enumerate multiplexing, server push and advanced crypto suite. The protocols HTTP/2.0 and TLS 1.3 were inspired from QUIC and 0-RTT secure connection. This implies that starting from TLS 1.3 protocol, a client can connect to a server in 0-RTT just like QUIC, after the TCP handshake. As such, the round-trip time is reduced and latency is greatly reduced, too. Currently, TLS 1.3 secure connections can be established in four packets.

6. Conclusion

QUIC is an attractive protocol with features that keep the setting of the modern day in mind. Implementing it in Firefox is one step forward in bringing the protocol to mainstream. The main purpose of this study was to encourage the use of user level transport layer protocols.

References

Belshe, M. & Peon, R. (2016). SPDY Protocol – Draft 3, *The Chromium Projects.* Retrieved October, 2016 from http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3.

Belshe, M., Peon, R., & Thomson, M. (2015). *Hypertext Transfer Protocol version 2* (*HTTP/2.0 IEFT draft*). Retrieved October, 2016 from https://tools.ietf.org/html/draft-ietf-httpbis-http2-17.

Bugzilla@Mozilla. (2015). Bug 1158011 - (QUIC) QUIC (Quick UDP Internet Connections) support. Reported on 2015-04-23, Modified: 2017-02-10. Retrieved from https://bugzilla.mozilla.org/show_activity.cgi?id=1158011.

Carlucci, G., De Cicco, L., & Mascolo, S. (2015). *HTTP over UDP: an Experimental Investigation of QUIC. ACM SAC'15*, April 13 – 17, 2015. Salamanca, Spain.

Combs, G. et. al. (2017). *Wireshark.* Retrieved February, 2017 from https://www.wireshark.org.

Cyr, B., Dorfman, J., Hamilton, R., Iyengar, J., Kouranov, F., Krasic, C., Kulik, J., Langley, A., Roskind, J., Shade, R., Shekhar, S., Shi, C., Swett, I., Tenneti, R., Vasiliev, V., Vicente, A., Westin, P., Wilk, A., Worley, D., Yang, F., Zhang, D., & Ziegler, D. (2013). *QUIC Wire Layout Specification*. Retrieved October, 2016 from https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/ edit?usp=sharing.

Digital Trends Staff. (2016). *Battle of the browsers: Edge vs. Chrome vs. Firefox vs. Safari vs. Opera vs. IE vs. Vivaldi*. Retrieved December 5, 2016 from http://www.digitaltrends.com/computing/best-browser-internet-explorer-vs-chrome-vs-firefox-vs-safari-vs-edge.

Google QUIC. (2016). *A group for QUIC development*. Retrieved October, 2016 from https://groups.google.com/a/chromium.org/forum/#!topic/proto-quic/Z5qKkk2XZe0.

Grigorik, I. (2013). *High performance Browser Networking*. O'Reilly Media, Inc.

Jasuja, N., Poonam S., Sehgal, P., Kate T., Stester, S.H., Cancelas, D., & Annaba, Y. K. (2017). *Firefox vs Google Chrome*. *Diffen.com*. Diffen LLC. Retrieved February, 2017 from http://www.diffen.com/difference/Firefox_vs_Google_Chrome.

Langley, A. & Chang, W.T. (2016). *QUIC crypto.* Revision 2016. Retrieved from https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit.

Megyesi, P., Kr´amer, Z., & Moln´ar´, S. (2016). High Speed Networks Laboratory, Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics. How quick is QUIC?. In *IEEE ICC 2016 - Communication QoS, Reliability and Modeling Symposium*. Budapest, Hungary.

Official QUIC. Retrieved October, 2016 https://www.chromium.org/quic.

QUIC FEC v1. (2016). Retrieved October, 2016 from https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit#heading=h.59osdxvm5jwp.

Valsorda, F. (2016). *An overview of TLS 1.3 and Q&A*. Cloudfare. Retrieved September 23, 2016 from https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a.

Willis, N. (2013). Connecting on the QUIC. *Linux Weekly News.* Retrieved from https://lwn.net/Articles/558826.

Xin, W., Zhaoquan, L., & Jie, W. (2013, December 12). Joint TCP Congestion Control and CSMA Scheduling without Message Passing. *IEEE Transactions on Wireless Communications*, Vol. 12.

Vraj Pandya received his Bachelor of Engineering in Computer Engineering (2015) from Gujarat Technological University, Gujarat, India. He is currently pursuing a Master's in Computer Science from Lamar University, TX, U.S.A., and a research assistant since Spring 2016. He is currently working on his master's thesis which is "optimizing algorithms for multiple GPUs". It is a program auto -tuner which tries to parallelize the costly serial kernel for CPU and GPUs. He works primary with CUDA, OpenCL, OpenGL, OpenMP and MPI. He also likes to work with network protocols.

Ştefan Andrei received his BSc in Computer Science (1994) and MSc in Computer Science (1995) from Alexandru Ioan Cuza University of Iasi, and PhD in Computer Science (2000) from Hamburg University. He was awarded with four competitive scholarships, as follows: the *Singapore-MIT Alliance Computer Science Fellowship*, 2002-2005, the *World Bank Joint Japan Graduate Scholarship Program*, 1998-2000, the *TEMPUS Fellowship*, 1998-1998, and *DAAD Scholarship*, German Government, 5/1997-7/1997. Dr. Andrei wrote over 100 publications published in prestigious journals and conference proceedings which have more than 200 non-self international citations in reputable publications. He has given invited talks at more than 20 reputable universities and industrial companies. He has been a Program Committee member or co-Chair of more than 40 international reputable conferences and a PI, co-PI, or Senior Personnel of more than 11 funded research grant proposals. He was promoted to ACM Senior Member in April 2013. Currently, he is an Associate Professor and Chair of the Department of Computer Science with Lamar University, Beaumont, TX, U.S.A. His research interests are in the areas of optimization techniques, verification, and scheduling analysis for multi-processor platforms for real-time embedded systems, software engineering and translation systems.