www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 9 Issue 01 January 2020, Page No. 24921-24924
ISSN: 2319-7242 DOI: 10.18535/ijecs/v9i01.4425

Implementation of Quick UDP Internet Connections (QUIC) Protocol

Minakshi Roy, Shamsh Ahsan, Gaurav Kumar, Ajay Vimal
Dept. of CSE, Sikkim Manipal Institute of Technology, SMU India

Abstract

With the advent of the Internet growth worldwide, we need to have a protocol which is faster and provides a better support for the following problems:

- Faster Connection Establishment Time
- Good Congestion Control
- Connection Migration
- Good Error Correction

One of the key aspects taken under consideration was current scenario of connection establishment time whenever a website is requested and poor video buffering over existing Internet Connections.

The prime objective is to create a proxy server which routes the incoming connection requests to QUIC supported libraries if the client supports QUIC. If the client does not support QUIC then it routes the incoming request to existing web server which can then handle the request using TCP. After creation of the proxy server a website has to be created using which we can test various aspects of the QUIC protocol.

1. Introduction

Quick UDP Internet Connections(QUIC) is a new transport which aims to reduce latency when compared to that of Transmission Control Protocol(TCP). QUIC is very similar to TCP+TLS+HTTP/2 implemented on User Datagram Protocol(UDP). Since TCP is implemented in operating system kernels, and middle box firmware, making significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP, it suffers from no such limitations.
QUIC is an encrypted transport: packets are authenticated and encrypted, preventing modification of the protocol by middleboxes.
Features of QUIC Protocol:

- **Zero Round Trips for Connection Establishment**

QUIC handshakes frequently require zero roundtrips before sending payload, as compared to 1-3 roundtrips for TCP+TLS. The first time a QUIC client connects to a server, the client must perform a 1-roundtrip handshake in order to acquire the necessary information to complete the handshake. The client sends an inchoate (empty) client hello (CHLO), the server sends a rejection (REJ) with the information the client needs to make forward progress, including the source address token and the server's certificates. The next time the client sends a CHLO, it can use the cached credentials from the previous connection to immediately send encrypted requests to the server.

- **Pluggable Congestion Control**

QUIC has pluggable congestion control and provides richer information to the congestion control algorithm than TCP. One example of richer information is that each packet, both original and retransmitted, carries a new sequence number.

- **Solution to Parking Lot Problem**

QUIC connections are identified by a 64-bit connection ID, randomly generated by the client. When a QUIC client changes IP addresses, it can continue to use the old connection ID from the new IP address without interruption.

- **Packet Recovery without Retransmission**

In order to recover from lost packets without waiting for a retransmission, QUIC can complement a group of packets with a Forward Error Connection(FEC) packet. Much like RAID-4, the FEC packet contains parity of the packets in

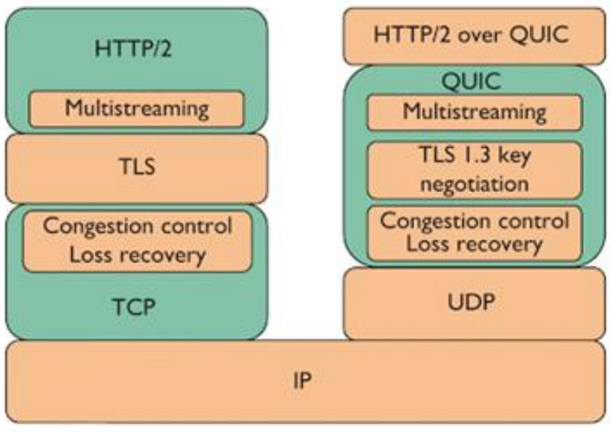

Figure 1 : QUIC in HTTPS Stack [5]

the FEC group. If one of the packets in the group is lost, the contents of that packet can be recovered from the FEC packet and the remaining packets in the group.

2. Quic Features

- **Built-in security (and performance)**

One of QUIC's more radical deviations from the now venerable TCP, is the stated design goal of providing a secure-by-default transport protocol. QUIC accomplishes this by providing security features, like authentication and encryption, that are typically handled by a higher layer protocol (like TLS), from the transport protocol itself.

The initial QUIC handshake combines the typical three-way handshake that you get with TCP, with the TLS 1.3 handshake, which provides authentication of the end-points as well as negotiation of cryptographic parameters. For those familiar with the TLS protocol, QUIC replaces the TLS record layer with its own framing format, while keeping the same TLS handshake messages.

Not only does this ensure that the connection is always authenticated and encrypted, but it also makes the initial connection establishment faster as a result: the typical QUIC handshake only takes a single round-trip between client and server to complete, compared to the two round-trips required for the TCP and TLS 1.3 handshakes combined.

But QUIC goes even further, and also encrypts additional connection metadata that could be abused by middle-boxes to interfere with connections. For example packet numbers could be used by passive on-path attackers to correlate users activity over multiple network paths when connection migration is employed (see below). By encrypting packet numbers QUIC ensures that they can't be used to correlate activity by any entity other than the end-points in the connection.

Encryption can also be an effective remedy to ossification, which makes flexibility built into a protocol (like for example being able to negotiate different versions of that protocol) impossible to use in practice due to wrong assumptions made by implementations (ossification is what delayed deployment of TLS 1.3 for so long, which was only possible after several changes, designed to prevent ossified middle-boxes from incorrectly blocking the new revision of the TLS protocol, were adopted).

- **Head-of-line blocking**

One of the main improvements delivered by HTTP/2 was the ability to multiplex different HTTP requests onto the same TCP connection. This allows HTTP/2 applications to process requests concurrently and better utilize the network bandwidth available to them.

This was a big improvement over the then status quo, which required applications to initiate multiple TCP+TLS connections if they wanted to process multiple HTTP/1.1 requests concurrently (e.g. when a browser needs to fetch both CSS and Javascript assets to render a web page). Creating new connections requires repeating the initial handshakes multiple times, as well as going through the initial congestion window ramp-up, which means that rendering of web pages is slowed down. Multiplexing HTTP exchanges avoids all that.

This however has a downside: since multiple requests/responses are transmitted over the same TCP connection, they are all equally affected by packet loss (e.g. due to network congestion), even if the data that was lost only concerned a single request. This is called “head-of-line blocking”.

QUIC goes a bit deeper and provides first class support for multiplexing such that different HTTP streams can in turn be mapped to different QUIC transport streams, but, while they still share the same QUIC connection so no additional handshakes are required and congestion state is shared, QUIC streams are delivered independently, such that in most cases packet loss affecting one stream doesn't affect others.

This can dramatically reduce the time required to, for example, render complete web pages (with CSS, Javascript, images, and other kinds of assets) particularly when crossing highly congested networks, with high packet loss rates.

3. Quic Implementation

A proxy server will route the incoming connection requests to the appropriate protocols.

By accessing the website via Transport Layer Sequrity(TLS)/Transmission Control Protol(TCP) the browser checks if the http header returned by the website contains the alt-svc field.

If the response contains a header: **alt-svc: 'quic=":443"; ma=2592000; '**, the UDP port 443 of the website supports the QUIC protocol; max-age is 2592000 seconds.

Then, the browser will initiate a QUIC connection. Before the connection is established, the http request is still sent via TLS/TCP.

Once the QUIC connection is established, subsequent requests are sent through QUIC.

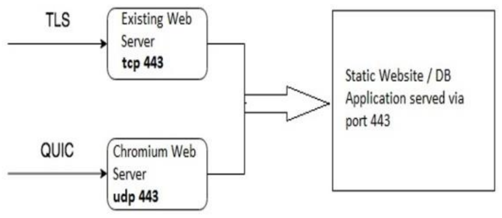
TLS
Existing Web Server tcp 443
QUIC
Chromium Web Server udp 443
Static Website / DB Application served via port 443

Figure 2: QUIC Proxy Server Requests Routing

When the QUIC connection is not available, the browser will take a 5min, 10min interval to check if the QUIC connection can be recovered. If it cannot be recovered, it will automatically fall back to TLS/TCP.

4. Result And Observations

In Figure 3 a plot is made in performance differences between QUIC and TCPwith each cell representing a different data size. Boxes with purple colours indicate that QUIC is faster than TCP and green indicates that TCP is faster than QUIC. Darker colours show more performance difference, and white cells indicate no significant difference between QUIC and TCP.

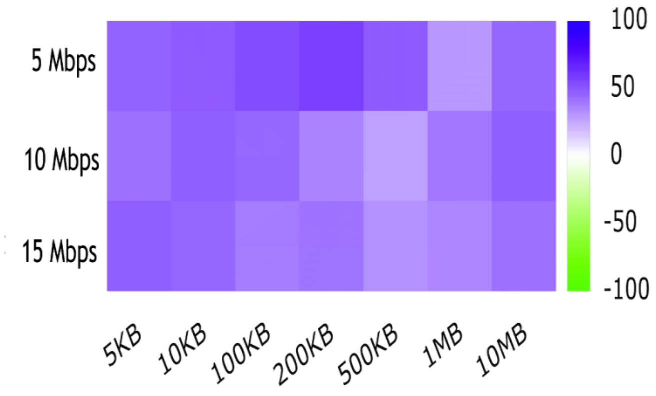

Figure 3: QUIC outperforms TCP under various load and speed.

5. References

[1] Byron Caughey, Christina D. Orru, Bradley R. Groveman, Andrew G. Hughson, Matteo Manca, Lynne D. Raymond, Gregory J. Raymond, Brent Race, Eri Saijo, Allison Kraus,Chapter Seventeen - Amplified Detection of Prions and Other Amyloids by RT-QuIC in Diagnostics and the Evaluation of Therapeutics and Disinfectants,Editor(s): Giuseppe Legname, Silvia Vanni,Progress in Molecular Biology and Translational Science,Academic Press,Volume 150,2017,Pages 375-388,
ISSN 1877-1173,ISBN 9780128112267

[2] R Hamilton, J Iyengar, “QUIC: A UDP-Based Secure and Reliable Transport For HTTP/2”., 2016

[3] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi * ,”The QUIC Transport Protocol: Design and Internet-Scale Deployment”, Google. 2017

[4] A. Barth. 2015. RFC 6454: The Web Origin Concept. Internet Engineering Task Force (IETF) (Dec. 2015).

[5] M Fischlin, F Günther, "Multi-stage key exchange and the case of google's QUIC protocol”, Proceedings of the 2014 ACM SIGSAC

[6] Hamilton, J Iyengar, I Swett,” QUIC: A UDP-based secure and reliable transport for HTTP/2R”, A Wilk - IETF, draft-tsvwg-quic-protocol-02, 2016

[7] G Carlucci, L De Cicco, S Mascolo,” HTTP over UDP: an Experimental Investigation of QUIC” 30th Annual ACM Symposium, 2015