Schahin Rajab
sr2@kth.se
18 May 2016

Performance testing TCP and QUIC

Abstract

This paper examines how packet loss affect throughput when using TCP and QUIC. Experiments were performed on the QUIC toy client and server compiled from the Chromium projects. Traffic shaping was used to simulate different rates of packet loss. Results were similar for the two transport protocols. An explanation might be that QUIC is still undergoing design decisions for its error recovery features.

Table of contents

1. Introduction

Google has in their quest of speeding up the web developed a multiplexing protocol called *Quic UDP Internet Connections* (QUIC). QUIC (pronounced “quick”) is situated at the transport layer and runs over *User Datagram Protocol* (UDP) [1]. It is still at the experimental stage and is open source, a toy server and client has been released at the Chromium projects for anyone to try out [2]. With QUIC, Google aims to address some of the design problems with the widely used transport protocol *Transmission Control Protocol* (TCP). For example, some of the goals with QUIC is to reduce head of line blocking (HOL blocking), offer zero round trip times (0-RTT), use Forward Error Correction (FEC) to reduce retransmission times after packet loss, and to always use an encrypted connection [3].

Since middleboxes such as firewalls and *Network Address Translators* (NATs) will block packets using transport protocols which they do not understand, QUIC is limited to either run on top of TCP or UDP. But since TCP is designed to guarantee in-order delivery it would be difficult to avoid HOL blocking, and major changes to the transport protocol would cause middleboxes filtering out the segments [3][4]. Meanwhile, UDP does not guarantee its datagrams to be delivered in-order and can thereby avoid HOL blocking. Thus, QUIC uses UDP as its underlying protocol for sending data.

QUIC is enabled by default on Google’s Chrome browser and the protocol is supported by Google’s own servers [5]. Various Google owned websites (i.e. YouTube, Google Search, etc.) are using QUIC traffic and can be recognized by the network protocol analyzer *Wireshark* as such. But even though QUIC has been deployed publicly, not all design choices has been figured out yet, such as FEC which has been temporarily disabled for QUIC after it was proven to be not so effective as it was first anticipated to be [6].

Even though QUIC sounds very promising in improving on TCP’s weaknesses, the QUIC’s applicable uses does not stretch beyond web applications. Meanwhile, TCP is used widely in many more applications such as File Transfer Protocol (FTP), Secure Shell (SSH), and Routing Information Protocol [7]. QUIC might not actually replace TCP, but some of its successful features could instead one day be implemented in TCP.

2. Background

Since the tests later on in this study compares TCP with QUIC on the issue of packet loss and its consequence on throughput, this section will describe these protocols and their respective methods for error recovery.

2.1 TCP

One of the widely used transport protocols in the transport layer is TCP. It is connection-oriented, meaning two entities have to first exchange segments with each other that will set the parameters for the TCP connection, this process is referred to as "handshaking".

A TCP session provides a reliable transfer of data, guaranteeing every byte to be delivered. Each segment contains a 32-bit sequence number used to keep track of which data is being delivered. The initial sequence number is randomly generated and incremented with the size of each subsequent segment. For every segment transmitted, the sender waits within a timeout interval before sending the following segments for an acknowledgment (ACK) segment from the receiving end containing how many bytes it has received. If the sender failed to receive the ACK before the timeout, or the ACK number was less than expected, it retransmits the oldest unacknowledged segment and waits for the ACK again. This ensures data to be delivered in order. However, the drawback to this is the extra delay caused by retransmission of segments.

2.2 UDP

UDP is said to be a connectionless transport protocol since it does not perform any handshaking between two end-points. In difference to TCP, UDP datagram is not guaranteed to be delivered in order. A UDP header is only 8 bytes in size containing a destination port, source port, checksum, and length. The light overhead makes UDP useful for applications that require low latency.

The source and destination port number is used for multiplexing and demultiplexing. The length field contains the size in bytes for the UDP datagram. And the checksum is used for maintaining the integrity of the datagrams, this is the only error detection function UDP provides. However, UDP does not provide any function for error recovery, that is left for the application to implement.

2.3 QUIC

Introduced in 2013, QUIC is a multiplexing protocol developed by Google in their goal of making the web faster [8]. A QUIC client connecting to a server for the first time will typically only perform one handshake. The client begins by sending a packet containing its version of QUIC and also revealing that this is the first time this client talks with this server [9]. The server then compares the version of the receiving client's packet with the version it itself supports. If it does not support the chosen version, the server responds with a Version Negotiation Packet containing a list of versions the server can use. This delays the connection with an extra RTT [9]. The client selects one of the versions available to proceed to the next step in the handshake [9].

When both the client and the server has agreed on what version of the protocol to use, the server sends a single packet containing hashes of the certificate chain [3]. The client will attempt to validate the received certificate by decoding the certificate hashes [3]. In case of the client failing to decode the certificate, it sends a packet to the server requesting a full list of the certificate chain [3]. Once the client has validated the server certificate, it can perform a successful handshake with the server and start exchanging application layer data with the server securely [10]. When the client wants to reconnect with the server, it can do so with a 0-RTT handshake. This 0-RTT handshake works by the client sending a "Hello" message to the server and then assumes that its server certificate is still valid so it starts the stream immediately before the server's "SYN+ACK" message has been received. The packets are always encrypted with exception for the client hello [3].

QUIC has a goal of implementing FEC to be used for error recovery [6]. FEC packets uses XOR to recover lost packets without having to wait for retransmission of data, thus reducing the overall latency. Packets are labeled in groups with an additional XOR packet with redundant bytes which can be used together with the rest of the group packets to reconstruct one lost packet [6]. This reconstruction of a packet is only possible if no more than one packet in a group was lost, otherwise, QUIC will have to fall back to retransmitting every lost packet [6]. In the event of two, or more, packets from the same group gets lost, the XOR packet will have been proven as ineffective and a waste of load. Same can be said when no lost packets occur. Hence, it is of interest by the QUIC team to know the distribution of lost packets. After they did some experiments on this matter, the results showed either no improvement, or an increase, in latency, and therefore is QUIC FEC undergoing design changes [6].

3. Methodology

In this study, testing was performed to compare how different rates of packet loss affects the goodput for TCP and QUIC. The tests were performed using a HP desktop machine model p6-2430eo running a 64-bit Ubuntu 14.04 LTS. The computer's hardware includes a Quad-core AMD CPU of model A10-5700 @ 3.4 GHz and 8 GB of RAM.

Before setting up the QUIC test server and client, the Chromium repository had to be checked out from the Chromium project homepage [10]. Once the dependencies were installed and Chromium was built, the QUIC test server and client source code was built from Chromium [2]. The server's certificate and private key was generated. In addition, a CA certificate was also generated and trusted by following the instructions in the 'Linux Cert Management' page located in the Chromium website [11].

After adding all the necessary certificates, the QUIC server and client were executed from separate terminals. The tests were performed by having the client asking for files of different sizes. The file sizes were as followed: 10 kB, 100 kB, 1 MB, and 10 MB. Traffic shaping was made using *Traffic Control* (tc) to simulate specific rates of packet loss each time the client requested a file. The packet loss rates were as followed: 0%, 0.1%, 0.5%, 1%, and 5%.

To do the same tests but with TCP, Iperf was used for generating TCP segments between a client and a server [12]. The bytes transmitted was of the same size as the files used in the QUIC experiment. Traffic shaping was also made using tc to simulate the same rates of packet loss that was used in the QUIC test.

4. Results

This section presents the results obtained from the experiments. All results have been presented as graphs. Each file size that was used for the measurements has its own separate graph containing the throughput for TCP and QUIC presented in log-scale on the Y-axis, and the rate of packet loss on the X-axis presented in linear.

The results for the two protocols were fairly similar. Both protocols utilized over 100 Mbit/s of throughput for all file sizes when there was no packet loss. The throughput for both the protocols did not vary so much for packet loss rates between 0.1%-1%.

But there was a huge drop for both protocols after the packet loss rate increased to 5%, where QUIC ended on throughputs a little higher than TCP.

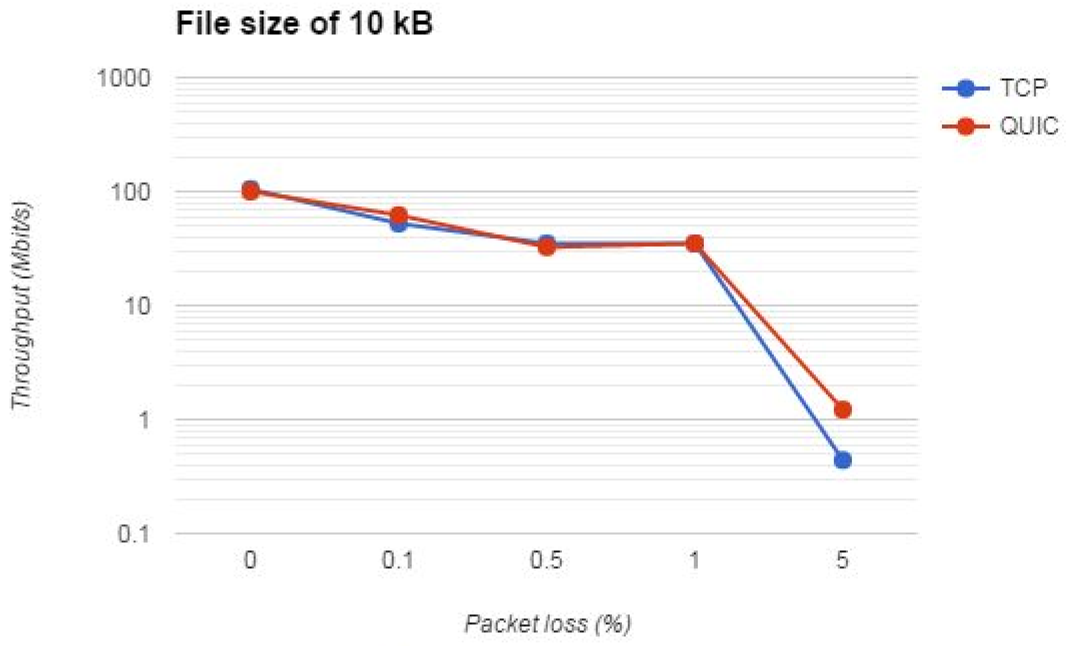

Figure 4.1: Packet loss test for file size 10 kB

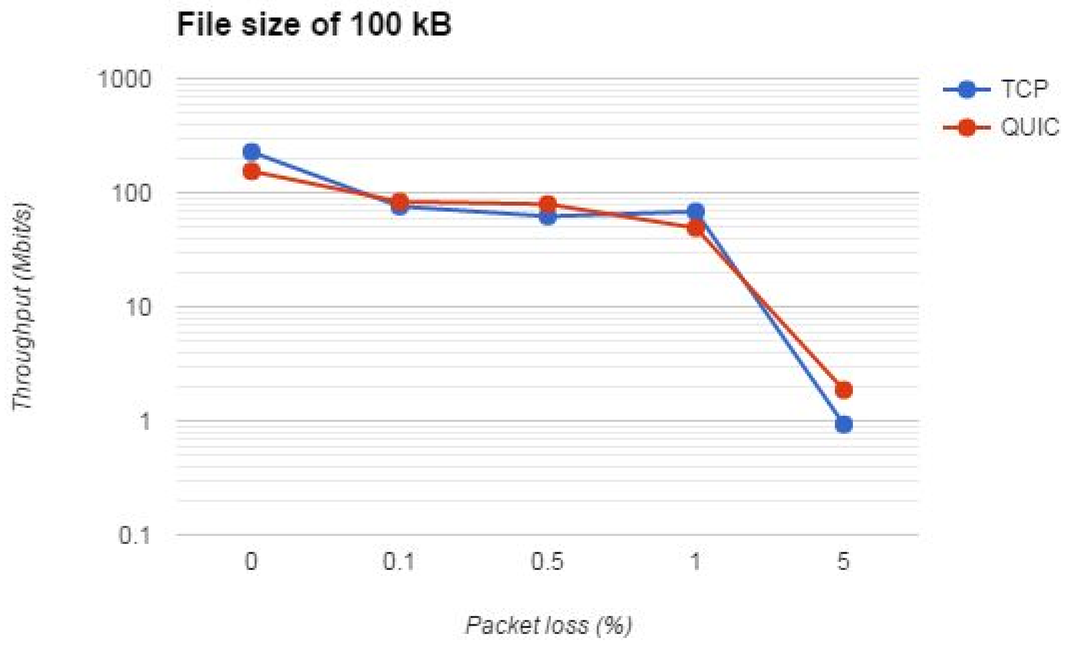

Figure 4.2: Packet loss test for file size 100 kB

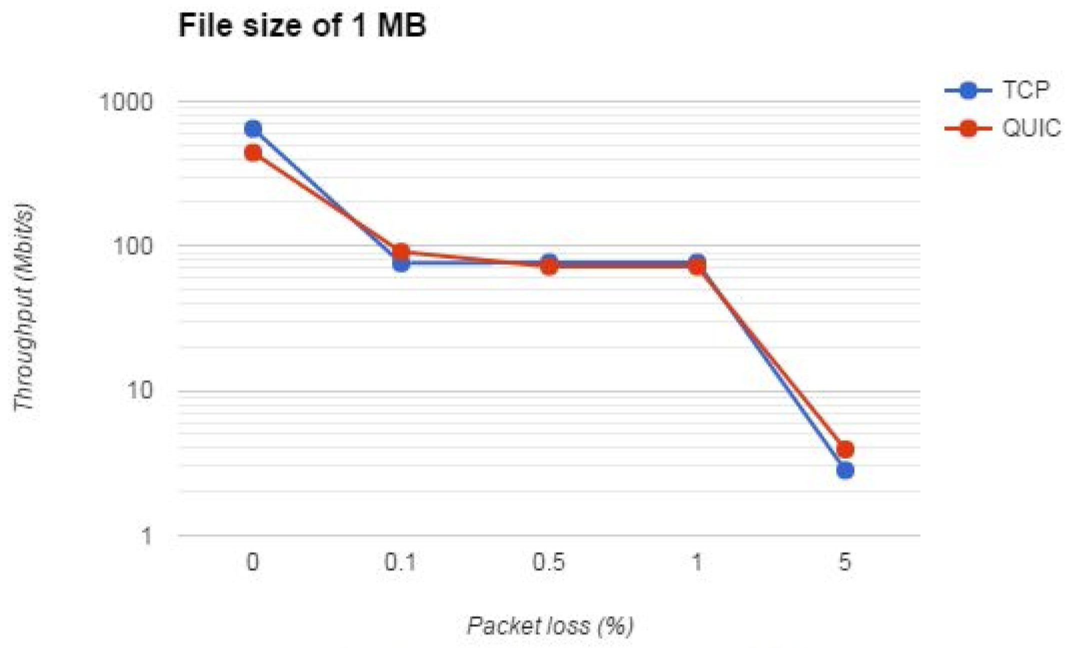

Figure 4.3: Packet loss test for file size 1 MB

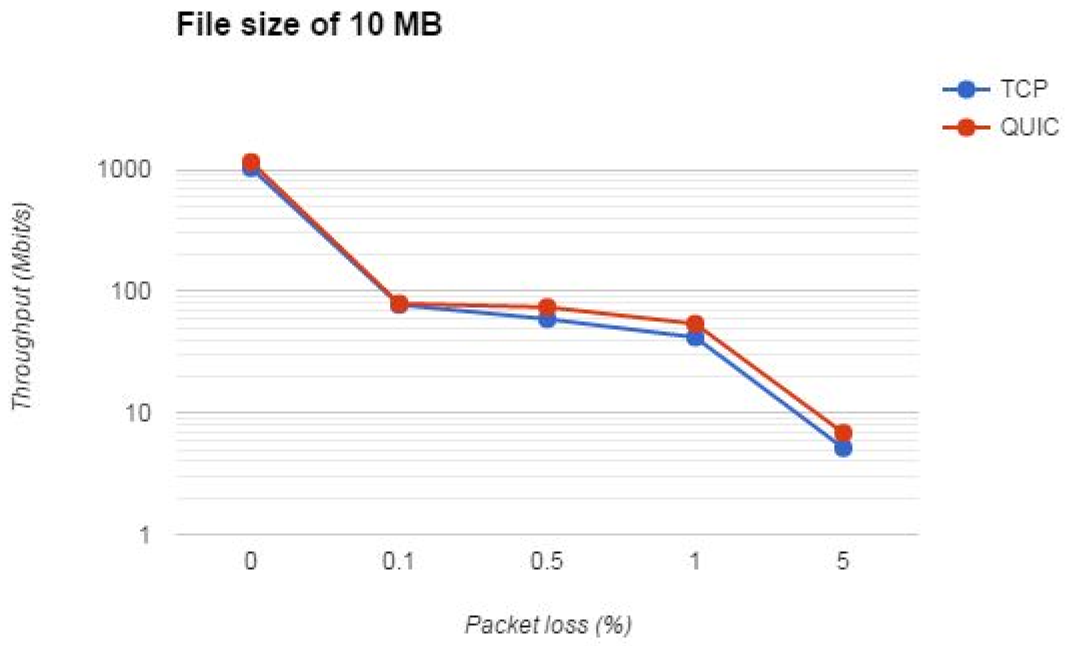

Figure 4.4: Packet loss test for file size 10 MB

5. Analysis

The QUIC protocol did not seem to handle packet loss significantly better than TCP. This is because FEC has been disabled for QUIC as described in Section 2.3. Therefore, the only enabled method QUIC has for recovering lost packets is by retransmitting them, which is the same method for error recovery that TCP uses.

TCP transmits segments at an increasing rate until a packet loss occur. Dropped segments are interpreted by TCP as congestion in the network, and responds to it by slowing down the transmission rate.

There are plans for QUIC to avoid congestion by implementing some bandwidth estimator that will estimate the available bandwidth in a connection and set proper packet pacing to transmit data packets according to this estimation [3]. Meanwhile, QUIC currently takes the same approach as TCP and interprets dropped packets as network congestion [3]. In such case, the source can choose to reduce its congestion window, or to increase the wait time between each UDP datagram in the queue (packet pacing) [3].

Since the used QUIC source code for the tests handles packet losses the same way as TCP does, it is only natural that the test results will be similar. The few differences are not of significant value and does not convincingly present one protocol being better at error recovery than the other. Although, these test results should be taken with a grain of salt as they were performed at home and not at Google where tests can be performed at large scale on millions of Chrome users worldwide and gather statistical results from them.

6. Conclusions and future work

The tests has shown that although QUIC has some interesting ideas for error recovery such as bandwidth estimation combined with packet pacing, and XOR based FEC packets, none of them seems yet to be enabled for the open source QUIC test server and client. Hence, the provided source code for QUIC have only reimplemented the same error recovery methods which TCP already uses. This meant that the test results were fairly similar for both protocols. Nonetheless, it was interesting to see how UDP datagrams could be modified to behave in the same fashion as TCP segments.

It would be interesting to test the QUIC protocol in different network conditions. For example, how delay, or different limits in bandwidth, affects the throughput for QUIC and TCP. Another interesting experiment would be to compare QUIC with some other protocol than TCP. such as SCTP, or Aspera’s FASP.

QUIC is still only an experimental protocol but has already been deployed at wide scale since Google Chrome and Google’s servers supports the protocol. It will be interesting to see if the QUIC team manages to address the design issues they are currently facing and to then see how QUIC can make an impact on our everyday web experience.

References

[1] Nathan Willis, ‘Connecting on the QUIC’, 17-07-13 [Online]. Available: http://lwn.net/Articles/558826/ [Accessed: 13 May 2016].
[2] The Chromium Projects, ‘Playing with QUIC - The Chromium Projects’, [Online]. Available: https://www.chromium.org/quic/playing-with-quic [Accessed: 13 May 2016].
[3] ‘QUIC: Design Document and Specification Rationale’, *Google Docs*, [Online]. Available:https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/mobilebasic?pli=1 [Accessed: 13 May 2016].
[4] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, ‘Is it Still Possible to Extend TCP?, Keio University, Universitatea Politehnica Bucuresti, University College London, 2011 [Online]. Available: http://conferences.sigcomm.org/imc/2011/docs/p181.pdf [Accessed: 13 May 2016].
[5] John Freml, ‘Enable QUIC in Google Chrome to speed up your secure web browsing’, 09 April 2014 [Online]. Available: http://www.pocketables.com/2014/04/enable-quic-google-chrome-speed-secure-web-browsing.html [Accessed 14 May 2016].
[6] ‘QUIC FEC v1’, *Google Docs*, [Online]. Available: https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit?pref=2&pli=1 [Accessed: 14 May 2016].
[7] ‘Common Application Ports - Bandwidth Controller’, [Online]. Available: http://bandwidthcontroller.com/applicationPorts.html [Accessed 14 May 2016].
[8] Chromium Blog, ‘Experimenting with QUIC’, 27 June 2013 [Online]. Available: http://blog.chromium.org/2013/06/experimenting-with-quic.html [Accessed: 15 May 2016].
[9] ‘QUIC Wire Layout Specification’, *Google Docs*, [Online]. Available: https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/edit [Accessed 15 May 2016].
[10] The Chromium Projects, ‘Get the Code: Checkout, Build, Run & Submit’, [Online]. Available: http://www.chromium.org/developers/how-tos/get-the-code [Accessed 16 May 2016].
[11] The Chromium Projects, ‘Linux Cert Management’, [Online]. Available: https://chromium.googlesource.com/chromium/src/+/master/docs/linux_cert_management.md#Add-a-certificate [Accessed 16 May 2016].
[12] Iperf, ‘iPerf - the network bandwidth measurement tool’, [Online]. Available: https://iperf.fr/iperf-download.php [Accessed 16 May 2016].