QUIC - Quick UDP Internet Connections

Florian Gratzer
Betreuer: Sebastian Gallenmüller, Quirin Scheitle
Seminar Innovative Internet-Technologien und Mobilkommunikation SS2016
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: florian.gratzer@tum.de

ABSTRACT

QUIC is a transport protocol on top of UDP, developed by Google. It uses mechanisms of TCP and introduces new features not used by other transport protocols. QUIC is optimized to be used for HTTP/2 connections and aims to reduce the end-to-end latency. QUIC works best (compared to TCP) for slow connections with high latency. This paper addresses the problems of TCP and the mechanisms of QUIC are discussed. QUIC packet and frame types are examined. The performance of HTTP over QUIC is compared to HTTP over TCP and HTTP + SPDY over TCP. The results show that QUIC is not better than the other protocols in all scenarios, but it can outperform TCP under certain network conditions.

Keywords

QUIC, Quick UDP Internet Connections, TCP, UDP, Layer 4, Google, Transport Protocol, HTTP/2, congestion control

1. INTRODUCTION

These days, the Internet is used for read the latest news or watching videos on platforms like YouTube. When the page load time is high, the user experience can become very bad.

In the last years, many approaches were considered to make Internet surfing faster. Internet Service Providers (ISPs) try to reduce page load times by increasing the bandwidth and using faster networking devices. Developers of web browsers tempt to make their software more efficient. Google's recent approach is to reduce the latency in the middle of the OSI model, namely in Layer 4, called Transport Layer.

Today, TCP and UDP are the most used protocols in the Transport Layer. While TCP is connection oriented, UDP is connectionless. Both protocols have advantages and disadvantages. TCP provides a reliable connection, but the TCP three-way handshake increases the latency for establishing a connection. This can be problematic for short living connections. In contrast, UDP establishes no connection, which results in a fast, but unreliable data transfer. To unite the advantages of both protocols, Google is developing a new transport protocol called **QUIC - Quick UDP Internet Connections** [8].

According to Google, QUIC performs better than TCP in scenarios with high Round Trip Time (RTT) and at least as good as TCP in most other scenarios [4]. This claim will be evaluated in Section 10. The most important differences between TCP and QUIC connections are that QUIC connections are always encrypted and connection establishment takes 0 RTTs when a server is known by a client and 1 RTT for the first connection to an unknown server. Figure 1 shows the connection establishment of QUIC compared to the TCP three-way handshake [10].

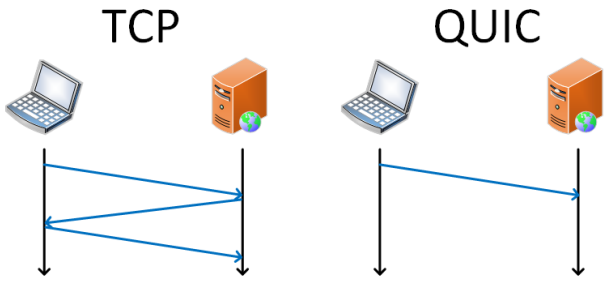


Figure 1: Connection Establishment of TCP and QUIC

TCP cannot be used to establish an encrypted connection within 0 RTT. Consequently, a different protocol is used as Transport Layer protocol. Therefore UDP is used, because it is already supported by most devices. Google's services like YouTube are already supporting QUIC , as well as Google's Browser Chrome [6]. In 2016, QUIC is still in development, so it is likely that the protocol changes in the near future. For this reason, Internet drafts of the IETF are used as sources instead of RFCs. These drafts can be updated or replaced by other documents at any time [9, 14].

This paper discusses the mechanisms of QUIC compared to TCP. Moreover, the QUIC packet and frame types are observed and the performance of QUIC is evaluated.

2. OVERVIEW OVER QUIC

QUIC is a transport protocol built upon UDP, designed to be used mainly for HTTP/2 [10], but it can be used by every application layer protocol. It is developed by Google and aims to reduce the connection establishment latency [10]. In contrast to TCP and UDP, QUIC connections are always encrypted and authenticated (see Section 5.7). The handshake is inspired by the handshake of TLS [5]. QUIC can be compared to TCP + TLS + HTTP/2. According to Google, QUIC has a better congestion control compared to TCP [10]. In Section 10, different scenarios are considered to evaluate this claim.

doi: 10.2313/NET-2016-09-1_06

3. PROBLEMS OF TCP

Today, TCP is widely used. It is more reliable than UDP but nevertheless it has several problems, especially when using it for HTTP/1.1. When designing a protocol based on TCP, these problems have to be addressed. To establish a TCP connection, the TCP three-way handshake is used. This handshake increases the connection establishment latency significantly. This is not problematic for long lasting connections, but can decrease the user experience for web surfing significantly. HTTP/1.1 uses a new TCP connection for each fetched URL [17].

Another problem is that a TCP segment can only carry a single HTTP/1.1 Request/Response. Consequently it is possible that a large number of small segments are sent within an HTTP/1.1 session. This can lead to a large overhead.

Also, HTTP/1.1 transfers are always initiated by the client [17]. This decreases the performance of HTTP/1.1 significantly when loading embedded files, because a server has to wait for a request from the client, even if the server knows that the client needs a specific resource [7].

The last problem discussed in this Section is Head-of-line (HOL) blocking. It occurs, when a packet is lost and consecutive packets arrive at the destination. The receiving host has then to wait for the retransmission of the lost packet until it can process consecutive packets, which arrived at the host without a loss. In scenarios like video streaming, a small number of lost packets do not have a notable influence on the user experience. Nevertheless, a TCP receiver has to wait for the lost packet before it is able to continue playing the video.

One way to overcome this problem with TCP is to open multiple connection between the same endpoints. This can work satisfyingly for a small number of connections, but when too many connections are opened, the connections tend to oscillate between very small and too large congestion windows when losses occur. This leads to a bad throughput and a bad user experience.

4. TCP MECHANISMS IN QUIC

Although QUIC uses UDP, many mechanisms are inspired by TCP. QUIC uses acknowledgments like TCP to inform the sender, that segments arrived at the receiver. The congestion control and loss recovery of QUIC is a reimplementation of TCP cubic with additional mechanisms [10]. TCP Cubic is optimized for high bandwidth networks with high latency [2]. In this section, the most important mechanisms of TCP used in QUIC are discussed. More can be found in the according Internet draft [14]. The mechanisms not used in TCP are focused in Section 5. QUIC uses a retransmission timer. Each segment, which is not acknowledged within this timer is considered to be lost (exceptions are discussed in Section 4.3).

QUIC also distinguishes between two phases: Slow Start and Congestion Avoidance. In the Slow Start phase, the congestion window grows exponentially, while it grows linearly in the Congestion Avoidance phase. A new connection always starts in the Slow Start phase until a loss occurs. After a loss, usually the congestion window the Fast Retransmit mechanism triggers and the connection changes to/stays in the Congestion Avoidance phase (more details in Section 4.1) [16].

4.1 Fast Retransmit

Fast Retransmit is a mechanism to avoid retransmission timeouts (RTOs). It triggers, when the sender receives three duplicate acknowledgments (ACKs) (this threshold is used in TCP as well as in QUIC) [4, 16]. A duplicate ACK is an acknowledgment for a segment, which has already been acknowledged and indicates a packet loss. When a RTO occurs, the congestion window is set to one maximum segment size (MSS), while the Fast Retransmit mechanism sets the congestion windows (and the Slow Start threshold) to a value dependent on the value it was before the loss. Also, the connection stays in the Congestion Avoidance phase and does not start with a new Slow Start as it is done after a RTO. According to evaluations by Google, over 99% of the packet losses are recognized by duplicate ACKs and so the Fast Retransmit mechanism kicks in in most cases [10].

4.2 Tail Loss Probe (TLP)

When a receiver does not receive the last segment of a transmission, Fast Retransmit cannot be triggered, because the receiver needs to receive (any) segments to identify the loss and therefore to send duplicate ACKs. To overcome this problem, QUIC uses Tail Loss Probes (TLPs). Before a RTO, the sender sends two TLPs containing the last unacknowledged segment [4]. The receiver then triggers the fast recovery mechanism [21].

4.3 Forward RTO-Recovery (F-RTO)

F-RTO aims to avoid unnecessary retransmissions to improve the performance of TCP/QUIC. TCP and QUIC use two mechanisms to trigger retransmission. As described in Section 4.1, Fast Retransmit kicks in, when the sender receives three duplicate ACKs. The second reason for retransmissions are RTOs. After RTOs, a sender sets the congestion window to one MSS and continues with a new Slow Start phase.

It is possible that a RTO occurs, even without a packet loss [18]. There are several reasons for spurious retransmissions, including delay spikes in mobile networks and different priorities of the sent data from the sender and the acknowledgments of the receiver [18]. After reducing the congestion window caused by delayed (but not lost) ACKs, the delayed ACKs reach the sender. As a result of the reduced congestion window, the ACKs are not inside the congestion window and trigger additional spurious retransmissions [18]. QUIC avoids this problem by not reducing the congestion window (and the Slow Start threshold) until the sender receives a subsequent ACK [4].

5. QUIC IMPROVEMENTS

As discussed in Section 4, QUIC's congestion control is inspired by TCP and therefore uses mechanisms of TCP. Additionally, QUIC uses mechanisms not used in TCP. Not all new mechanisms of QUIC are discussed in this paper and can be found in the Internet draft about QUIC's congestion control [4]. All mechanisms are pluggable, so QUIC can be configured to fit best in different scenarios.

5.1 Faster Connection establishment

When a QUIC client connects to a QUIC server for the first time (currently a server is identified by its IP address and UDP port [6]) it sends an empty, so-called inchoate, Client-HELO (CHLO) [10]. The server then responds with a rejection (REJ) including the server configuration and certificates [10]. The client uses this information to send another CHLO (can already contain application data) which is then accepted by the server (provided the versions of client and server are compatible) and all consecutively sent data are encrypted and authenticated [10]. When the server is known by the client, this inchoate CHLO is not needed, resulting in a 0 RTT handshake instead of the 1 RTT handshake for unknown server. More details about the handshake can be found in the QUIC Internet draft [9].

5.2 Multiplexing

TCP uses TCP ports and IP addresses (of both endpoints) to identify a connection, when Multipath TCP is not used. So it is not possible for a client to communicate with a server over multiple ports via a single connection. In contrast, QUIC uses a 64 bit connection identifier (which is randomly selected by the client) [5]. Within these connections, multiple streams are used to transport segments. This allows clients to establish connection mobility across IP addresses and UDP ports [14]. It is also possible to use multiple ports for an application, but the application has then to listen to all these UDP ports, because QUIC uses UDP as Layer 4 protocol. Connection IDs also allow connection migration. Thus, a connection can stay established, even when the IP address of one of the endpoints changes.

The support of multiple streams within a single connection also addresses Head-of -line blocking by sending independent data via different streams. All streams are identified by a stream identifier (stream ID) and can be established by the client or the server. To avoid collisions, the stream ID has to be even, when the client initiates the stream and odd when initiated by the server. Each participant has to increase the stream ID monotonically for new streams [14]. The stream IDs 0-3 are reserved. QUIC provides flow control on stream- and connection level.

5.3 Monotonically Increasing Sequence Numbers

TCP uses the same sequence numbers for retransmitted segments. This leads to the problem, that a host cannot distinguish between original and retransmitted segments. Contrary, QUIC uses a monotonically increasing sequence number for every segment, also for retransmitted ones, resulting in unique sequence numbers [14]. This helps to estimate the RTT more accurately, because the RTT can also be calculated for delayed ACKs. TCP can use an extension called Timestamps to distinguish between original and retransmitted segments too [22].

User data is transferred within stream frames (see Section 7). This frames also contain sequence numbers, which stay the same for retransmitted data [6].

5.4 Better Signaling

QUIC has a more verbose signaling than TCP. As focused in Section 5.3, all packets get a new sequence number, even retransmitted ones. Therefore every sequence number is unique, and a receiver can distinguish between an original packet and its retransmitted equivalent. As a result, a sender can calculate the RTT more accurately, because it can recognize the difference between a delayed ACK and the ACK of the retransmitted packet.

TCP uses cumulative ACKs. This means that all segments with a sequence number smaller than an acknowledged packet are also acknowledged. To reduce the amount of ACKs, only the packet with the highest sequence number in the receive window is acknowledged. As a result, the sender has to wait a whole RTT to notice the loss or to unnecessarily retransmit received packets [15].

To overcome this problem, TCP has the option to use Selective Acknowledgments (SACKs). SACKs are used by the receiver to inform the sender, which packets are received, so a sender can retransmit lost segments. More details can be found in RFC 2018 [15].

QUIC uses Negative Acknowledgments (NACKs) instead of SACKs with a bigger range (up to 255 instead of 3) [9]. Negative Acknowledgments report lost packets directly instead of implicating it by not getting acknowledged. Although NACKs and SACKs are different approaches to report lost packets, the result is comparable. The larger range of QUIC NACKs compared to TCP SACKs is advantageous for large receive windows.

5.5 Forward Error Correction (FEC)

QUIC can use Forward Error Correction (FEC) to reconstruct lost packets [14]. A scheme similar to RAID systems using XOR operations is used for this purpose. This approach for reconstructing packets is simple and therefore effective, but it cannot reconstruct lost packets when multiple packets are lost within a group. As it will be discussed in Section 10 this feature can have a negative effect when the loss rate is low or too high [13].

5.6 Packet Pacing

TCP tries to send data as fast as possible. When data is sent too fast, losses are very likely. When a loss occurs, and Fast Retransmit kicks in, the congestion window is decreased. Then the congestion windows grows again until the next loss occurs. This can result in a very bursty transmission [23].

Packet Pacing is an approach to make the transmission less bursty by not sending at full rate. This has usually a positive effect in scenarios with low bandwidth, but it can decrease the overall throughput for fast connections (see Section 10). Packet Pacing is also used for TCP by some Linux kernels.

5.7 Authentication and Encryption

TCP Headers (and the payload, as long as no other protocols are used) are neither encrypted nor authenticated. Conversely, QUIC packets are always encrypted (except for the public header, see Section 6) and authenticated (including the public header) after the connection is established.

This also includes IP spoofing protection. The handshake is inspired by TLS. More details can be found in the QUIC crypto documentation [11].

6. PACKET TYPES AND HEADER FORMAT

QUIC differentiates between two types of packets: Regular Packets and Special Packets. Special Packets can either be Version Negotiation Packets or Public Reset Packets. Regular Packets are divided into Frame Packets and FEC Packets (see Section 5.5). QUIC packets have two different headers: an unencrypted Public Header and an encrypted Private Header. More information about these headers can be found in the according Internet draft [9].

6.1 Public Header

Figure 2 shows the Public Header of Regular Packets. Special Packets also have a Public Header with a slightly different design. The flag field, which is the same for all packet types is shown more in detail in Figure 3:

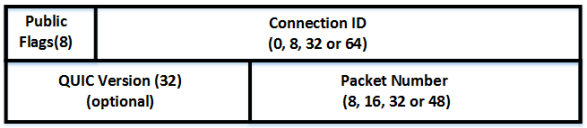


Figure 2: QUIC Public Header (Numbers in bits) [9]

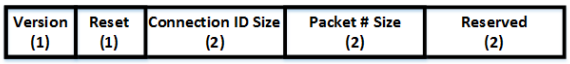


Figure 3: Flag Field in QUIC's Public Header (Numbers in bits) [9]

Public Flags: The *Version* flag is interpreted differently, depending on the sender of the packet (client or server). When set by the client, the packet includes the *QUIC Version* field. A client has to set this flag until it receives a packet from this server without setting this flag. When *Version* is set by the server, it indicates that the packet is a *Version Negotiation Packet*.

The *Reset* flag is set for Public Reset Packets. When both, *Version* and *Reset* are set, the packet is treated as *Public Reset* Packet.

The *Connection ID Size* and *Packet Number Size* determine the length of the according header field. The exact interpretation can be found in the according Internet draft.

The last two bits are unused and reserved for future use. More details can be found in the QUIC Internet draft.

Connection ID: A randomly generated identifier, used to identify the connection instead of the four-tuple (source IP address, source port, destination IP address, destination port) used by TCP.

QUIC Version: This field is only present, when the *Version* flag is set and is used by the client to propose a QUIC version to be used when establishing a connection.

Packet Number: Used to identify packets. As described in Section 5.3, each packets gets a unique identifier, even retransmitted ones.

6.2 Regular Packets

Regular Packets are always encrypted (except for the public header) and authenticated (including the public header). All regular packets have a common Private Header format. This header starts directly after the Public Header and is followed by the payload, which has a different format for Frame Packets and FEC Packets. The Private Header is shown in Figure 4:

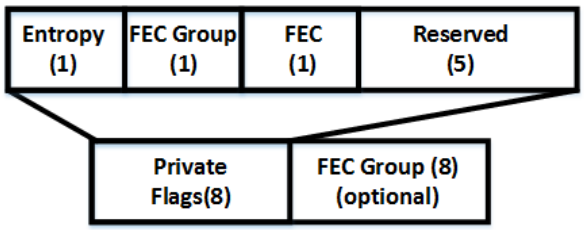


Figure 4: Private Header Format (Numbers in bits) [9]

Private Flags: The *Entropy* flag indicates that this packet contains the 1 bit of entropy in frame packets or the result of the XOR-Operation of the entropy flag of the other packets in the FEC group when it is an FEC Packet. The *FEC Group* flag determines whether FEC is used and therefore if the FEC field is present. The last flag (*FEC*) is set for FEC packets.

FEC: This field is used to determine, which packets belong to the same FEC group. The value in this field is the offset from the first packet in the FEC group to this packet.

6.2.1 Frame Packet

Frame Packets carry the actual application data within a connection. The payload is located directly after the Private Header and formatted as shown in Figure 5:

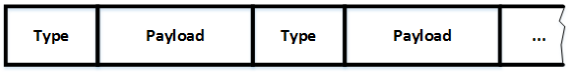


Figure 5: Frame Packet Format [9]

Type: Specifies the Type of the Frame. The Frame types are focused in Section 7.

Payload: Payload of the frame, is dependent on the type of the frame.

6.2.2 FEC Packet

Regular Packets with the *FEC* flag set are FEC Packets and contain the result of the XOR-Operation of the Frame Packets of the same FEC group.

7. FRAME TYPES

QUIC differentiates between Regular Frame Types and Special Frame Types. The frame types are not discussed in detail in this paper, a detailed list of all frame types and the values of the field type for the according frame type can be found in the QUIC Internet draft. Not all frame types are used in the current implementation (see description of according frames) [9].

7.1 Regular Frame Types

Regular Frames can either be Stream, Acknowledge or Congestion Feedback Frames.

Stream Frames: Used to send data over a stream and to (implicitly) create a stream.

Acknowledge Frames: Used to signal that a packet (not necessarily a Frame Packet) has been received or is missing. The exact format can be found in the QUIC Internet draft.

Congestion Feedback Frame: Currently not used. Designed to provide additional congestion feedback in future implementations.

7.2 Special Frame Types

In Addition to the Regular Frame Types, the following Special Frame Types are specified.

Padding Frame: Used to pad a packet with binary zeros.

Connection Close Frame: Used to notify the communication partner that a connection is closing. All unclosed streams within a connection are also closed, when a connection is closed.

Reset Stream Frame: Used to irregularly terminate a stream. When sent by the stream creator, the creator indicates that he wants to close a stream. When the receiver of a stream sends a Reset Stream Packet, he either does not accept the stream or an error occurred.

Go Away Frame: Used to inform the opposing endpoint that the connection will be closed in the near future and should not be used any more. Usually sent shortly before a Connection Close Frame.

Windows Update Frame: Used to tell the opposing peer to update the receive window.

Blocked Frame: Used for debugging purposes. Inform the receiver of a frame that the sender is ready to transmit data, but is blocked by flow control mechanisms.

Stop Waiting Frame: Used to tell the communication partner not to wait for packets any more.

Ping Frame: Used to verify, that the opposing endpoint is still available. A ping frame contains no payload and the receiver has to answer with an Acknowledgment Frame.

8. QUIC DISCOVERY IN CHROME

Chrome uses TCP to send requests to unknown server. When a server supports QUIC, it uses the Alternate Protocol Header in HTTP Responses to inform a client, that it supports QUIC [12]. For the next requests, Chrome tests which protocol is faster and uses the faster protocol. It also stores a list of all server, which support QUIC for subsequent connection [12]. When a QUIC connection fails (e.g. incompatible versions of client/server, Firewalls, ...) the host is marked as broken for a certain time span (currently five minutes), where QUIC will not be tested again. After this time span, QUIC is tested again and is marked as broken for twice as long as before, when QUIC is still unable to establish a connection [12]. When a user does not want to use QUIC (for any reason), it is possible to disable it manually.

9. SPDY

The performance of QUIC is usually evaluated by comparing the page load time of HTTP/1.1 over TCP, HTTP/1.1 + SPDY over TCP and HTTP/1.1 over QUIC. SPDY is discussed shortly in this section.

SPDY ("speedy") was developed by Google and enhances the HTTP/1.1 Protocol. In Contrast to pure HTTP/1.1, SPDY allows hosts to send multiple HTTP/1.1 Requests/Responses within a single TCP segment. With SPDY it is also possible to prioritize HTTP/1.1 Requests/Responses. So SPDY solves one of the problems, discussed in Section 3, but the other problems can't be solved by SPDY, because it is still using TCP.

HTTP/2 is based on SPDY and today it is used more often than SPDY. As a consequence, the use of SPDY is not longer supported by Google [3]. As SPDY is not in the focus of this paper and just used for comparison, it is not discussed here further. More informations can be found in the SPDY Internet draft [20] and on the SPDY Chromium project page [19].

10. EVALUATION

QUIC was evaluated by Google and independent testers. This Section summarizes their results. The experiments of Google set the focus on the effect of different mechanisms. On the one hand, they are made from the developing company, so they have to be treated with care. On the other hand, Google can interpret the results best, as they know the scenario, which QUIC is intended for better than independent testers. Moreover they show the impact of different mechanisms more in detail than other testers. When HTTP, QUIC and SPDY are compared, HTTP refers to HTTP/1.1 over TCP, QUIC refers to HTTP/1.1 over QUIC and SPDY is short for HTTP/1.1 + SPDY over TCP.

10.1 Experiments by Google

According to Google's experiments in 2015, 75% of the connections are connections to known hosts and therefore it takes *0 RTT* to establish this connections. Google assumes that this is the reason for 50 - 80% of the overall median latency improvements when comparing TCP and QUIC [4].

Packet Pacing reduces retransmission by 25%. To do this, the sender does not send at the maxiumum speed. As a

result the page load latency is reduced for slow connection, but fast connections suffer from this mechanism. All in all it does not change the median page load latency [4].

TLP has no effect on the median latency, but it improves the 95% - quantile of the latency and the YouTube rebuffer rate by almost 1% [4].

All in all, Google comes to the conclusion, that QUIC performs significantly better for slow connections with high latency and performs as good as TCP for fast connections with a low latency [6].

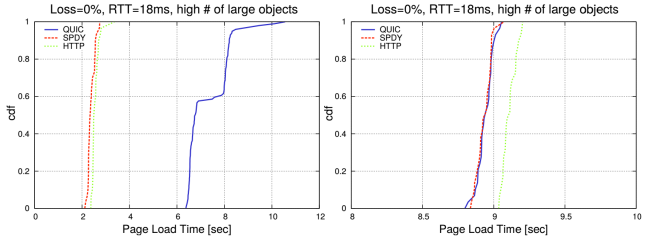
10.2 Measurements from the Budapest University of Technology and Economics

A group from the Department of Telecommunications and Media Informatics of the Budapest University of Technology and Economics tested QUIC compared to HTTP and SPDY [7]. Their measurement environment and results are shown here. The group did a higher number of test, only the most interesting ones are discussed in this paper.

Measurement Environment: A regular laptop with Google Chrome and additional tools for measurement purposes were used on the client side. On the server side, they used four sample pages, hosted on Google Sites. In their scenario, the demo pages contained either small (400 B - 8 kB) or large (128 kB) objects. For each scenario, there was a page with a small (5) and a page with a large (50) amount of the according objects. Between the client and the server, a shaper server was used in order to change the network conditions [7]. In contrast to the experiments of Google, they do not test the influence of enabling/disabling different mechanisms.

The following Figures show the Cumulative Distribution Function (CDF) (y-axis) of the Page Load Times (PLT) (x-axis) under different conditions [7].

In one of their tests, they showed that QUIC performs significantly worse in a scenario with, 50 Mb/s bandwidth, low RTT (18 ms), a packet loss rate of 0% and 50 objects with a size of 128 kB. When TCP is used (with and without SPDY) for HTTP connections, the average page load time is about 2 seconds, while it it higher than 7 seconds, if QUIC is used. After reducing the bandwidth to 10 Mb/s, the performance of QUIC is again comparable to the other ones and the average page load time is about 9 seconds. The result is shown in Figure 6.

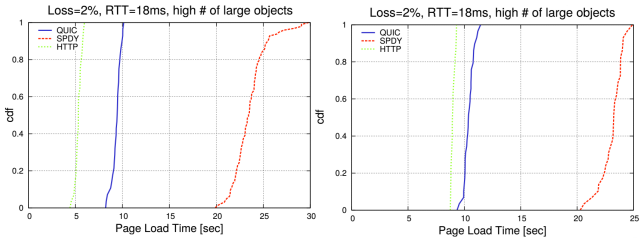


a) Bandwidth = 50Mbps b) Bandwidth = 10Mbps

Figure 6: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

The reason for the weak performance of QUIC is packet pacing [7]. This mechanisms tries to reduce retransmissions by sending below the maximum rate possible. In their scenario, the loss rate is 0%, so packet pacing tries to avoid packet losses, which wouldn't be there even when the host sends at full speed. The researchers argue, that QUIC underestimates the maximum bandwidth and sends at a far too slow rate [7]. When using a lower bandwidth of 10 Mb/s this behavior cannot be observed.

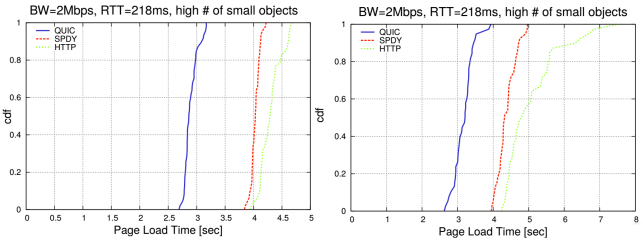
A completely different picture is shown, when the loss rate in the former scenario is changed. Figure 7 shows the effect of setting the loss rate to 2% in an apart from that unchanged scanario [7]. HTTP still performs better than QUIC, but the difference got smaller. In contrast, SPDY performs very bad. The reason for this is Head-of-line blocking, when sending multiple HTTP responses via a single TCP segment (see Section 3).



a) Bandwidth = 50Mbps b) Bandwidth = 10Mbps

Figure 7: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

Google claims that QUIC performs significantly better than other protocols when QUIC is used for slow connections with high RTTs, especially when requesting a large number of small objects. The researchers used a scenario with 2 Mb/s bandwidth, a high RTT of 218ms and 50 objects between 400B and 8kB to verify this claim [7]. The results shown in Figure 8 support Google's claim. QUIC performs better in both scenarios (0% and 2% loss rate). The benefits are mostly gained by the multiplexing mechanism of QUIC, which reduce the overhead significantly when sending a high number of small objects [7]. The group also argue that the 0 RTT connection establishment has a positive effect on page load time, which has to be mentioned [7].



a) Loss = 0% b) Loss = 2%

Figure 8: Comparison of QUIC, HTTP and SPDY under shown circumstances [7]

The group comes to the conclusion, that QUIC performs badly under high bandwidth, but outperforms the other protocols when the RTT is high, especially under low bandwidth

doi: 10.2313/NET-2016-09-1_06

[7].

10.3 Measurements of the Politecnico di Bari

A group from Gaetano Carlucci Politecnico di Bari & Qualvlive in Italy also performed measurements to evaluate QUIC [1]. Their test environment was similar to the environment of the Hungarian group, but they used different scenarios. Details can be found in their paper [1].

In contrast to the testers of the Budapest University of Technology and Economics, the Italian group tested the influence of FEC on the loss rate and the channel utilization. They used scenarios with 3, 6 and 10 Mb/s and a loss rate of 0 and 2% The results are shown in Figure 9.It can be seen that the loss rate is higher when FEC is enabled.

These results may come as a surprise at first glance. FEC cannot reconstruction lost packets, when more than one packet of a group is lost. When using FEC, the sender of the packet is not informed about the loss after the receiver was able to reconstruct the packet and so it continues to increase the congestion window until a packet loss occurs, where FEC cannot reconstruct the packet. This results in the observed behavior. Also, the channel utilization is higher, when using FEC because of the required redundancy, which also has to be sent via the same channel.

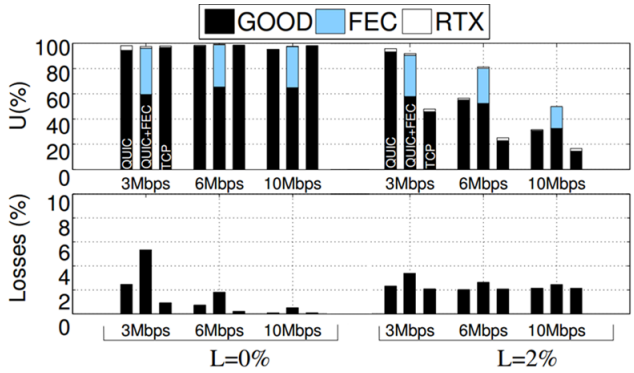


Figure 9: QUIC with enabled/disabled FEC compared to TCP [1]

The group also tested a similar scenario with low bandwidth and a high round trip time to verify Google's claims, that QUIC performs significantly better than TCP under this circumstances. This study confirms the results of Section 10.2. They used a bandwidth of 3 Mb/s, a (high) loss rate of 2% and an RTT of 50ms [1]. The result is shown in Figure 10. The y-axis show the throughput in kb/s and the x-axis shows the time in seconds.

The last interesting part of the evaluation, discussed in this paper, is the performance of QUIC when it is used in parallel with TCP with limited buffers. The research group used a bandwidth of 5 Mb/s and a RTT of 50ms for this scenarios. Then the used buffers of 13, 30 and 60 kB as bottleneck (with Tail Drop policy) [1]. Figure 11 show the throughput of the two protocols when using different buffer sizes. When QUIC and TCP are used in parallel, QUIC uses more of the buffer size unless the network is over-buffered (60kB under their conditions). This is because QUIC uses a smaller congestion window reduction factor than TCP [1].

Figure 12 shows that QUIC does not only use more of the buffer, but also has a higher throughput. When the buffer is large enough, both protocols have a comparable throughput.

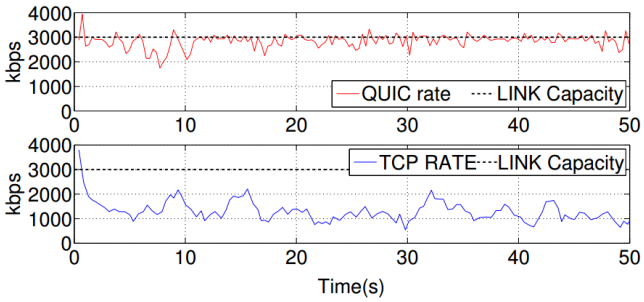


Figure 10: Comparison of TCP and QUIC (without FEC) [1]

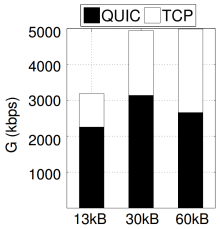


Figure 11: Impact of buffer size on QUIC/TCP [1]

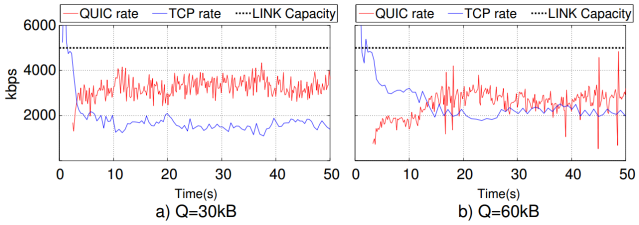


Figure 12: Parallel use of TCP and QUIC with limited buffers [1]

11. CONCLUSION

QUIC is a transport protocol, developed by Google, meant to be used by HTTP/2 [10]. It is still under development, so not all mechanisms discussed in this paper are implemented yet [9, 14]. QUIC uses UDP as Layer 4 protocol, but several mechanisms used are reimplementations of TCP best practices. QUIC also has features, which are not possible for TCP [4]. In contrast to TCP, encrypted QUIC connections can be established within 0 RTTs.

Google claims that QUIC works at least as good as TCP in all scenarios [6]. Measurements showed, that this is not entirely true. QUIC performs better than TCP in networks with high latency, especially when the bandwidth is low. The gains come from the 0 RTT handshake and QUIC's multiplexing mechanism. However, TCP performs better, when the bandwidth is high and the latency is low [1, 7].

doi: 10.2313/NET-2016-09-1_06

Also, QUIC introduces Forward Error Correction, which is used to reconstruct lost packets instead of requesting it again. Therefore, redundant data has to be sent. The current implementation does not work as well as Google intends, because it increases the amount of retransmissions (see Section 10.3). The problem with the FEC mechanism of QUIC is that the sender is not informed about a loss, when the packet was successfully reconstructed. Also it cannot reconstruct packets, when multiple packets of a FEC Group are lost. All in all the performance of QUIC is worse, when FEC is enabled [1].

In 2016, QUIC is used by Google services like YouTube and Google's Browser Chrome, but QUIC is still under development [1, 7]. When Google wants to establish QUIC on more server in the future, QUIC has to become better and the unimplemented mechanisms must be implemented. Some mechanisms of QUIC are innovative, but the actual implementation is not sufficient to replace TCP. QUIC has a lot of potential,but it has to be observed if QUIC can coexist with TCP in the future or if it can even replace the currently most used transport protocol.

12. REFERENCES

[1] G. Carlucci, L. De Cicco, and S. Mascolo. HTTP over UDP: An Experimental Investigation of QUIC. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 609–614. ACM, 2015. http://c3lab.poliba.it/images/3/3b/QUIC_SAC15.pdf.

[2] CUBIC for Fast Long-Distance Networks. https://tools.ietf.org/html/draft-rhee-tcpm-cubic-02.

[3] Hello HTTP/2, Goodbye SPDY. https://blog.chromium.org/2015/02/hello-http2-goodbye-spdy.html.

[4] IETF93 QUIC BarBoF: Congestion Control and Loss Recovery. https://docs.google.com/presentation/d/1T9GtMz1CvPpZtmF8g-W7j9XHZBOCp9cu1fW0sMsmpoo/edit#slide=id.gb7abf88bb_0_28.

[5] IETF93 QUIC BarBoF: Protocol Overview. https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs/edit#slide=id.g99041b54d_0_124.

[6] IETF93 QUIC video (BAR BOF). http://recordings.conf.meetecho.com/Playout/watch.jsp?recording=IETF93_QUIC&chapter=BAR_BOF.

[7] P. Megyesi, Z. Krämer, and S. Molnár. Comparison of web transfer protocols. http://proprogressio.hu/wp-content/uploads/2016/01/MolnarSandor_2015.pdf.

[8] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[9] QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 IETF Internet Draft. https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02.

[10] QUIC at 10,000 feet. https://docs.google.com/document/d/1gY9-YNDNAB1eip-RTPbqphgySwSNSDHLq9D5Bty4FSU/edit.

[11] QUIC crypto design doc. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit.

[12] QUIC Discovery. https://docs.google.com/document/d/1i4m7DbrWGgXafHxwl8SwIusY2ELUe8WX258xt2LFxPM/edit.

[13] QUIC FEC v1. https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit.

[14] QUIC Loss Recovery And Congestion Control IETF Internet Draft. https://tools.ietf.org/html/draft-tsvwg-quic-loss-recovery-01.

[15] RFC2018 - TCP Selective Acknowledgment Options. https://tools.ietf.org/html/rfc2018.

[16] RFC2581 - TCP Congestion Control. https://tools.ietf.org/html/rfc2581.

[17] RFC2616 - Hypertext Transfer Protocol – HTTP/1.1. https://www.ietf.org/rfc/rfc2616.txt.

[18] RFC5682 - Forward RTO-Recovery (F-RTO): An Algorithm for Detecting. https://tools.ietf.org/html/rfc5682.

[19] SPDY Chromium Project Page. http://dev.chromium.org/spdy/.

[20] SPDY IETF Internet Draft. https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00.

[21] Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses. https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01.

[22] TCP Extensions for High Performance. https://www.ietf.org/rfc/rfc1323.txt.

[23] TCP Performance Implications of Network Path Asymmetry. https://tools.ietf.org/html/rfc3449.