Comparison of Present-day Transport Layer network Protocols and Google's QUIC

Omkar Nalawade
Student
Department of Electronics Engineering
Vidyalankar Institute of Technology
Mumbai, India
Omkarn049@gmail.com

Amit Dhanwani
Assistant Professor
Department of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India
Amit.dhanwani@vit.edu.in

Tejashree Prabhu
Student
Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India
prabhutejashree@gmail.com

Abstract - **In 1983, Charles Bachman defined the concept of a model having seven layers through the work at Honeywell Information Services, after which an astonishing number of single and multi-layered protocols came into the picture. Lately designed on Connectionless UDP protocol and upgraded for HTTP/2 semantics. Google is beginning to find better solutions for downloading web load time and then they came up with two new protocols: SPDY in 2009 and QUIC in 2013. QUIC (Quick UDP Internet Connection) is a new multiplexed and secure transport protocol developed by Google and implemented in Chrome since 2013. QUIC claims to provide multiplexing and flow control equivalent to HTTP/2. QUIC implements the spirit of known TCP loss recovery mechanisms. As stated before being built with HTTP/2, also implements mechanisms that make its security equivalent to TLS and improves reliability. In this paper, we review QUIC based on the current protocols taking connection establishment, congestion control, error control, flow control and security as key parameters for comparison. We present the performance comparison of HTTP/2, TLS, SPDY and QUIC particularly based on page load time and security. Although in most cases QUIC is more efficient compared to the current protocols in certain scenarios, QUIC fails to provide the basic mechanisms it has been built for. As nascent as a protocol QUIC is we certainly hope a great deal of development in near future.**

Keywords—QUIC, HTTP/2, TLS, SPDY, HTTP/1.1, Connection Establishment, Congestion Control, Loss Recovery, Error Control, Flow Control, Security.

I. INTRODUCTION

Our websites have modified very much since the coming of HTTP 1.1 which is responsible for delivering news, video, and other web applications accessible from desktop computers to smartphones. HTTP represents a large portion of the Internet traffic even today. HTTP/1.1 is still used by many applications, even after HTTP/2 standardized in February 2015 at the IETF.

HTTP/2 has natural advantages being developed to replace HTTP/1.1. Like the multiplexing of streams which optimizes the delivery. However, HTTP/2 still relying on TCP as the successor of HTTP/1.1, HTTP/2 failed to overcome the problem of Head-of-Line (HoL) blocking, which can affect the end-users experience in case of packet losses. Moreover, setting up the first connection from a client to the server requires 3 RTT messages. As TCP is encrypting the data via the TLS protocol, which will need a handshake phase of 2 Round Trip Time (RTT) duration plus the necessary 1-RTT TCP handshake.

As web pages and web applications go on increasing and also the Internet traffic, it becomes unavoidable to explore for new and better technologies. Page Load Time (PLT) is a crucial aspect of web performance since it is directly correlated with page abandonment. The immense increase of mobile and web applications have exposed the limitations of current transport protocols. As a result, web service providers are constantly engaged in finding innovative solutions for this. Mainly, protocols like TLS being security focused protocol have relatively high overhead of connection establishment latency, often causing user dissatisfaction and resulting in less number of customers and financial losses.

As a result, several efforts are taken to design new transport protocols which in addition to confidentiality, authentication, and integrity has low latency as one of the main design focus. Firstly, Google developed a web transfer protocol called SPDY in 2009. Today, SPDY is famous and implemented not only on Google's servers and Chrome browser, but on the newest versions of Firefox and Internet Explorer are also supported by the protocol. The new HTTP/2 which is being developed by the Internet Engineering Task Force (IETF) is largely based on SPDY.

QUIC (Quick UDP Internet Connections) is a new network transport protocol by Google running on UDP, instead of TCP, thus there is no need for the initial TCP handshake mechanism. It implements its own encryption system, as good as TLS, not losing on transfer speed as it combines connection establishment and key agreement in only 1 RTT. It is developed such that, QUIC can start a connection in 0 RTT, immediately sending encrypted application data to the server most of the time. Furthermore, running on top of UDP, QUIC avoids the HoL issue reducing packet loss in the course. Until now, only Google uses QUIC for its own services and browsers, but more platforms will be needed to have QUIC deployed everywhere.

The performance evaluation of QUIC has been studied by a few research papers in the literature, mainly focusing on the use case of QUIC as a transport for HTTP

This paper is organized as follows. Section II describes how connection establishment, loss recovery and congestion control of QUIC differs from the connection establishment of TCP. Section III describes how error control and flow control in QUIC differ from HTTP/2.Section IV describes how security in QUIC differs from security in TCP-TLS. Section V focuses on a comparison table comparing QUIC with TCP-TLS-HTTP, TCP-HTTP/2 and TCP+TLS+SPDY.

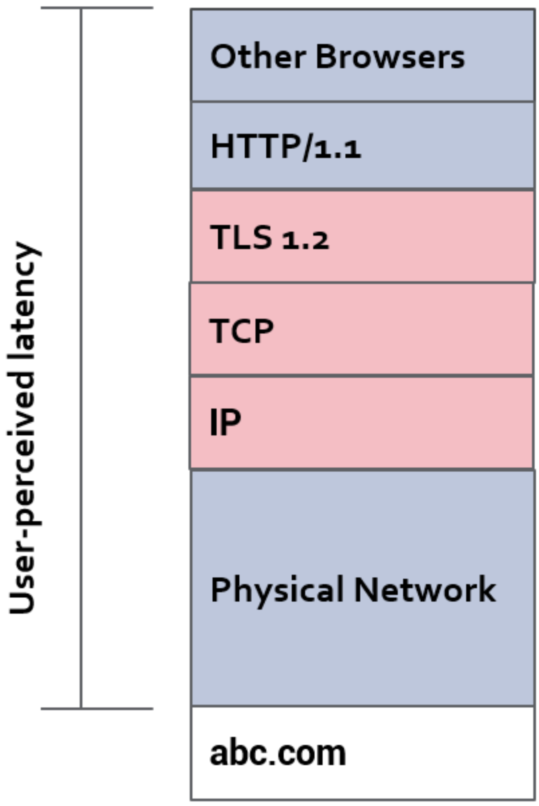

Fig. 1. Latency of Common Browser and their protocol stack.

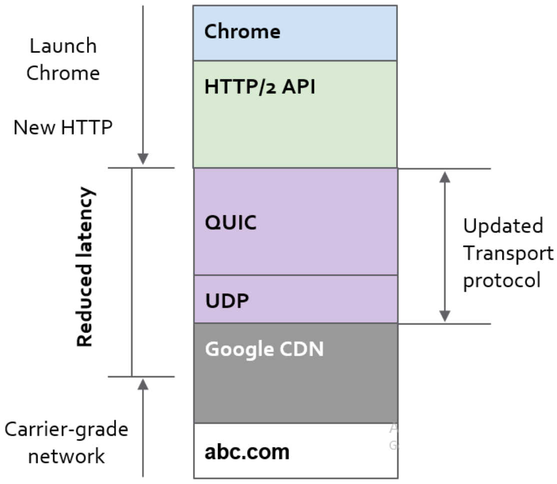

Fig. 2. Reduced Latency in Goggle Chrome.

II. Connection Establishment, Loss Recovery and Congestion Control

A. Connection Establishment in TCP vs Connection Establishment in QUIC

TCP is a connection oriented protocol. It uses three way handshake mechanism for connection establishment [1]. Initially the server must bind and listen to open the port for connection. This step is known as passive open. Once server opens up the ports the client will initiate an active open.

Now, to establish connection three way handshake is used.

Step1: Active open is performed by the client by sending a SYN signal to the server. This segment is randomly numbered X.

Step2: The server sends an SYN ACK to acknowledge the SYN request and to request active connection to the client. The ACK for X is numbered as X+1.SYN for client is numbered Y.

Step3: The client now acknowledges the connection request by the server by sending an ACK signal. This ACK signal is numbered as Y+1. Thus, enabling duplex client server connection.

A client initiates the connection. QUIC uses version negotiations with crypto and transport handshakes to reduce connection establishment latency as compared to TCP [1].

A QUIC client initiates a connection. QUIC's Connection establishment is initialized by sending packets with version flag on (SYN) and mentioning the version of the protocol being used. Every packet sent by the client should have the version flag on, until it receives a packet from the server having the version flag off (ACK). If the server encounters any more packets with version flag on it will have to ignore the packets.

When the server receives a packet with connection ID, It will compare the client's versions to the versions it supports. If the client's version is supported by the server, the server will use this version till the connection terminates.

If the client's version is not supported by the server then the server will send a version negotiation packet to the client.

This packet will have the version flag set and will include the server's set of supported versions.

When the server receives a packet with a Connection ID for a new connection, it will compare the client's version to the versions it supports. If the client's version is acceptable to the server, the server will use this protocol version for the lifetime of the connection. In this case, all packets sent by the server will have the version flag off. This will delay the connection by 1-RTT.When the client receives a Version Negotiation Packet from the server, it will select an acceptable protocol version and resend all packets using this version. These packets must continue to have the version flag set and must include the new negotiated protocol version. Eventually, the client receives the first Regular Packet (i.e. not a Version Negotiation Packet) from the server indicating the end of version negotiation, and the client now sends all subsequent packets with the version flag off.

In order to avoid downgrade attacks, the version of the protocol that the client specified in the first packet and the set of versions supported by the server must be included in the crypto handshake data. The client needs to verify that the server's version list from the handshake matches the list of versions in the Version Negotiation Packet. The server needs to verify that the client's version from the handshake represents a version of the protocol that it does not actually support.

Thus, we can conclude that QUIC combines crypto and transport handshakes for connection establishment between client and server to reduce the number of RRT's to 0-RTT as compared to TCP whose RTT's range from1-3-RTT.

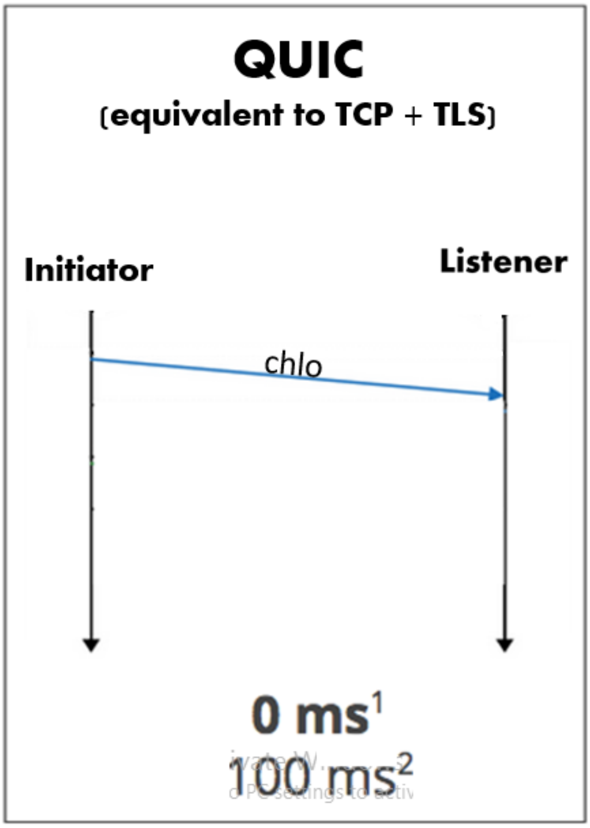

Fig. 3. Connection Establishment in QUIC.

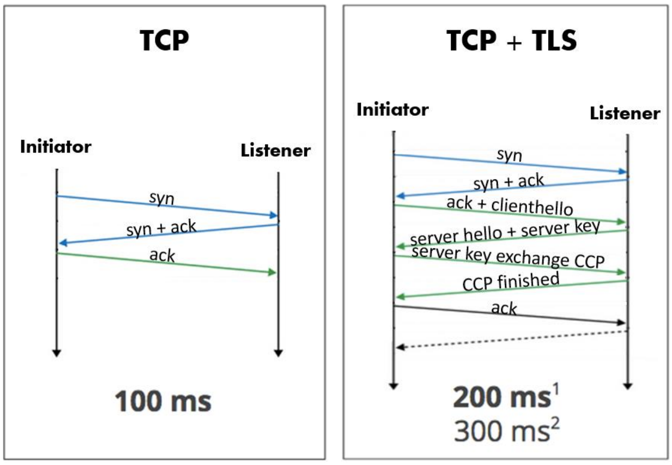

Fig. 4. Connection Establishment in TCP and TCP+TLS.

B. Congestion Control in TCP vs Congestion Control in QUIC

Like TCP QUIC uses pluggable congestion control algorithms [1]. Along with pluggable congestion control QUIC has richer signaling than TCP. This helps QUIC to provide richer information than TCP. To avoid retransmission ambiguity QUIC uses different packet sequence numbers so that the sender can distinguish between the original and retransmitted packet.

QUIC ACK's considers the delay between packet reception and acknowledgement being sent. Thus, efficiently calculating the round trip time. QUIC supports up to 256 NACK in ACK frames as compared to TCP 32 bit SACK. Thus, QUIC is more flexible towards packet reordering than TCP. Therefore, we can conclude that QUIC has better congestion control than TCP. Peculiarities.

C. Loss Recovery

We quickly portray QUIC's activities on packet transmission, ACK reception, and timer expiration events as stated in [2]. On Sending a Packet, retransmission timer may be set based on the situation. If the handshake has not finished, QUIC starts a handshake timer 1.5x the SRTT, with exponential back off. If there are outstanding packets which have not been acknowledged, perhaps QUIC will set the loss timer. Depending on the loss detection implementation, default is 0.25RTT in the case of Early Retransmit. If less than 2 TLPs have been sent, QUIC figures out and restarts TLP timer. First case, if there are multiple packets in flight. Timer is set for max (10ms, 2*SRTT). Second case, if there is only one packet in flight. Timer is set to max (1.5*SRTT + delayed ACK timer, 2*SRTT). If 2 TLPs have been sent, set the RTO timer, after the primary RTO. Timer is set to max (200ms, SRTT+4*RTTVAR) with exponential back off.

On Receiving an ACK, Receiver validates the acknowledgement, including discarding any out of order ACKs. It updates RTT measurements. Sender marks unacknowledged packets lower than the largest observed and not NACKed in this ACK frame as ACKED. Packets with packet number lesser than the largest observed that are NACKed have missing reports incremented in light of FACK. (Largest observed - missing packet number). After that, threshold is set to 3 by default. Packets with missing reports > threshold are marked for retransmission. If unacknowledged packets are outstanding and the largest observed is the largest sent packet, the retransmission timer will be set to 0.25SRTT, implementing Early Retransmit with timer. Finally, timers are stopped, if no packets are outstanding.

QUIC uses one loss recovery timer, which when set, can be in one of several states. At the point when the timer expires, the state decides the activity to be performed.

- Handshake state:
  - Retransmit any outstanding handshake packets.
  - Loss timer state:
  - Lose the outstanding packets which have been NACKed so far.
  - Report the loss to the congestion controller.
  - Retransmit as many as the congestion controller allows.
- TLP state:
  - Retransmit the smallest outstanding packet which is retransmittable.
  - Do not mark any packets as lost until an ACK arrives.
  - Restart timer for a TLP or RTO.
- RTO state:
  - Retransmit the two smallest outstanding packets which are retransmittable.
  - Do not collapse the congestion window until an ACK arrives and confirms that the RTO was not spurious. Note that this step obviates the need to implement FRTO.
  - Restart the timer for next RTO.

III. ERROR CONTROL AND FLOW CONTROL IN QUIC

A. Error Control

In order to recover lost packets without waiting for a retransmission, QUIC currently employs a XOR-based FEC scheme An FEC packet contains parity of the packets within the FEC group. If one in every one of the packets within the group is lost, the contents of that packet are recovered from the FEC packet and therefore the remaining packets within the group.

While HTTP/2 framing permits 2 categories of error:

- An error condition that makes the complete connection unusable could be a connection error.
- An error in an individual stream is a stream error.

In HTTP/2, if endpoint confronts a connection error should first deliver a GOAWAY frame with the stream identifier of the last stream that it definitely got from its peer [3]. The GOAWAY frame comprise an error code that shows connection is terminating. The endpoint should block the transmission control protocol connection, after sending the GOAWAY frame for an error condition.

If the GOAWAY won't be accurately received by the receiving endpoint. During connection error, GOAWAY solely provides a best-effort plan to communicate with the peer regarding why the connection is being terminated. An endpoint will finish a connection at any time.

Each implement completely different error management methods according to their needs.

QUIC executes connection reliability, congestion management, and flow control. QUIC flow management intently takes after HTTP/2's flow control. QUIC is connected and utilizes a single packet sequence number space for shared congestion management and loss recovery over the connection. All information exchanged during a QUIC connection, together with the crypto handshaking, is distributed as information inside streams, however the ACKs recognize QUIC Packets.

Streams are freelance sequences of bi-directional data divided into stream frames. Streams are created either by the client or the server, will at the same time send data interleaved with completely different streams and may be canceled. QUIC's stream lifespan is shown strongly after HTTP/2's [3].

B. Flow Control

QUIC executes connection reliability, congestion management, and flow control. QUIC flow management intently takes after HTTP/2's flow control. QUIC is connected and utilizes a single packet sequence number space for shared congestion management and loss recovery over the connection. All information exchanged during a QUIC connection, together with the crypto handshaking, is distributed as information inside streams, however the ACKs recognize QUIC Packets.
Stream creation is done implicitly, by sending a STREAM frame for a given stream. To stay away stream ID collision, the Stream-ID must be regardless of the possibility that the server starts the stream, and if the client starts the stream, it is odd. 0 isn't a legitimate Stream-ID. Stream 1 is constrained for the crypto handshaking that needs to be client- initiated stream. While utilizing HTTP/2 over QUIC, Stream 3 is held for sending compressed headers for all different streams.

Flow management is employed for each individual streams and for the connection as an entire. So as to recover lost packets while not expecting a retransmission, QUIC presently employs an easy XOR-based FEC scheme. An FEC packet contains parity of the packets within the FEC group. If one among the packets within the group is lost, the contents of that packet may be recovered from the FEC packet and also the remaining packets within the group. The sender could decide whether or not to send FEC packets to optimize specific situations (e.g., starting and finish of a request).

IV. SECURITY IN QUIC

QUIC is a performance driven protocol which combines the security mechanisms from TCP, TLS and DTLS to provide security similar to TLS. QUIC doesn't rely on TCP which eliminates redundant communication and uses initial keys to achieve faster connection establishment. QUIC doesn't rely on TCP like TLS and provides most of the functionalities provided by TCP itself [4]. TLS uses one session key while QUIC uses two session keys. QUIC cryptographic protection such as protection against IP spoofing and packet reordering. QUIC does not guarantee a strong forward secrecy as compared to TLS. In presence of adversaries QUIC doesn't provides the reduced communication feature it promises. Thus, QUIC is not worse than TLS during presence of adversaries but is much better than TLS in favorable conditions.

V. COMPARISON TABLE

TABLE I. Popular networking protocols comparison.

Parameters	***UDP + QUIC***	***TCP+TLS+ HTTP***	***TCP+HTTP /2***	***TCP+TLS+ SPDY***
Acronym	Quick UDP Integrated Connections	Transmission Control Protocol+ Transport Layer Security+ Hypertext Transfer Protocol	Transmission Control Protocol+ Hypertext Transfer Protocol 2.0	Transmission Control Protocol+ Transport Layer Security+ SPDY
Connection	Connection less protocol	Connection -oriented protocol	Connection -oriented protocol	Connection -oriented protocol
Ordering of data packets	QUIC has no packet ordering. It uses a single packet sequence number if packet ordering is required.	TCP arranges data packets in the specified order.	TCP arranges data packets in the specified order.	TCP arranges data packets in the specified order.
Reliability	More reliable in favorable conditions	Less reliable in favorable conditions as compared to QUIC	Less reliable in favorable conditions as compared to QUIC	Less reliable in favorable conditions as compared to QUIC
Streaming of data	A receiver announces the absolute byte offset in each stream up to which it is ready to receive data. As the data is transmitted, received and delivered on a particular stream it increments the offset if required, which allows the sender to send more data on that stream.	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.
Process Load	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. Therefore, the load increases.	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. Therefore, the load increases.	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. Therefore, the load increases.
Connection Establishment Latency	QUIC combines the crypto and transport handshakes, reducing the number of roundtrips required for setting up a secure connection. Therefore, the connection establishment latency is comparatively less.	3 roundtrips required for TCP+TLS before application data can be sent. Therefore, the connection establishment latency is comparatively more.	1 roundtrips required for TCP+TLS before application data can be sent. Therefore, the connection establishment latency is comparatively more than QUIC.	1 roundtrips required for TCP+TLS before application data can be sent. Therefore, the connection establishment latency is comparatively more than QUIC.
Congestion Control	Original and retransmitted data has unique packet sequence number which helps the sender to distinguish between the retransmission packets from ACKs of original packets. This allows QUIC to	TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.

Parameters	***UDP + QUIC***	***TCP+TLS+ HTTP***	***TCP+HTTP /2***	***TCP+TLS+ SPDY***
	carry out congestion control flexibly.			
Multiplexing without head-of-line blocking	QUIC currently compresses HTTP headers via HTTP/2 HPACK header compression, which imposes head-of-line blocking for header frames only.	HTTP multiplexes many streams atop TCP's single-byte stream abstraction	HTTP/2 multiplexes many streams atop TCP's single-byte stream abstraction	-
Authenticated and encrypted header and payload	QUIC packets are always authenticated and typically the payload is fully encrypted. The parts of the packet header which are not encrypted are still authenticated by the receiver, so as to thwart any packet injection or manipulation by third parties [1].	TCP headers appear in plaintext on the wire and not authenticated, causing a plethora of injection and header manipulation issues for TCP, such as receiver-window manipulation and sequence-number overwriting.	TCP headers appear in plaintext on the wire and not authenticated, causing a plethora of injection and header manipulation issues for TCP, such as receive-window manipulation and sequence-number overwriting.	TCP headers appear in plaintext on the wire and not authenticated, causing a plethora of injection and header manipulation issues for TCP, such as receive-window manipulation and sequence-number overwriting.
Forward error correction	QUIC uses a simple XOR-based FEC scheme.	TCP does error checking and retransmission.	TCP does error checking and retransmission	TCP does error checking and retransmission
Connection migration	QUIC connections are identified by a 64-bit Connection ID, randomly generated by the client. QUIC can survive IP address changes and NAT re-bindings since the	TCP connections are identified by a 4-tuple of source address, source port, destination address and destination port	TCP connections are identified by a 4-tuple of source address, source port, destination address and destination port	TCP connections are identified by a 4-tuple of source address, source port, destination address and destination port

Parameters	***UDP + QUIC***	***TCP+TLS+ HTTP***	***TCP+HTTP /2***	***TCP+TLS+ SPDY***
	Connection ID remains the same across these migrations. QUIC also provides automatic cryptographic verification of a migrating client, since a migrating client continues to use the same session key for encrypting and decrypting packet			
Multiplexing	QUIC is designed from the ground up for multiplexed operation, lost packets carrying data for an individual stream generally only impact that specific stream. Each stream frame can be immediately dispatched to that stream on arrival, so streams without loss can continue to be reassembled and make forward progress in the application	TCP's "slow start" forces applications to open multiple TCP connections to achieve parallelism and higher performance	TCP's "slow start" forces applications to open multiple TCP connections to achieve parallelism and higher performance	TCP's "slow start" forces applications to open multiple TCP connections to achieve parallelism and higher performance
Header compression	QUIC implements HPACK header compression for	_	HTTP/2 uses HPACK algorithm for header compressio	Headers can be compressed and new headers can be sent

Parameters	UDP + QUIC	TCP+TLS+ HTTP	TCP+HTTP /2	TCP+TLS+ SPDY
	HTTP/2, which Unfortunately introduces some Head-of-Line blocking since HTTP/2 header blocks must be decompressed in the order they were compressed.		n, which significantly reduces the GET request message sizes compared to SPDY	after the connection has been established
Handshake	0-RTT handshake	3-RTT handshake	3-RTT handshake	3-RTT handshake
Forward Secrecy Mechanism	QUIC does not provide forward Secrecy guarantees against attackers that may corrupt the server after, but in the same time period as, the data that was sent.	TLS Session Tickets are often used to minimize round trips. Their use in some sense cancels the forward secrecy guarantees provided by TLS because the Session Ticket key, which must be retained for sufficiently long periods of Time for resumption to be effective, can be used to decrypt previous communication.	TLS Session Tickets are often used to minimize round trips. Their use in some sense cancels the forward secrecy guarantees provided by TLS because the Session Ticket key, which must be retained for sufficiently long periods of Time for resumption to be effective, can be used to decrypt previous communication.	TLS Session Tickets are often used to minimize round trips. Their use in some sense cancels the forward secrecy guarantees provided by TLS because the Session Ticket key, which must be retained for sufficiently long periods of Time for resumption to be effective, can be used to decrypt previous communication.
Re-ordering attacks	As QUIC works on UDP so the receiver cannot reject any packets until all the packets are received and then it has to reorder them	If an adversary reorders the packets then the receiver will know about the reordering based on the order specified by the TLS layer sequence numbers and will reject the packets	-	If an adversary reorders the packets then the receiver will know about the reordering based on the order specified by the TLS layer sequence numbers and will reject the packets
IP Spoofing attacks	Attacks related to IP spoofing need to be addresses due to lack of sender authentication	Attacks related to IP spoofing do not necessarily need to be addressed due to three way handshake mechanism	-	Attacks related to IP spoofing do not necessarily need to be addressed due to three way handshake mechanism
Protection	QUIC provides Cryptographic protection	TLS does not provide cryptographic protection	-	TLS does not provide cryptographic protection

.

IV. CONCLUSION

In this paper, we have presented our analysis about QUIC, especially aspects to understand working of different protocol with respect to each other. We know that QUIC outperforms HTTP/2 over TCP/TLS in unstable networks such as wireless mobile networks but in case of stable and reliable networks, the benefits of QUIC are not so obvious. We have compared the performance of the latest QUIC implementation with the standard HTTP/1.1, HTTP/2 and its predecessor SPDY.

We show, QUIC combines crypto and transport handshakes for connection establishment between client and server to reduce the number of RRT's to 0-RTT as compared to TCP whose RTT's range from1-3-RTT. We can say that QUIC has better congestion control than TCP, as it is more flexible towards packet reordering than TCP. QUIC currently employs a XOR-based FEC scheme to recover lost packets without waiting for a retransmission. The error control strategies are different and adapted to work with their transport layer protocol. In case of security, QUIC is not worse than TLS during presence of adversaries but is much better than TLS in favorable conditions. To sum up all we compared all the aspect in easily understandable tabular form for reader convenience.

In our future work, we plan to evaluate the QUIC performance with regards with the standard HTTP/1.1 and other protocols under different network conditions. Up to now, only Google uses QUIC for its own services and browsers. Since the performance gain is so obvious, some improvement in security aspect is needed before a wide adoption.

Acknowledgment

We thank the anonymous reviewers for their useful Comments. This paper is based upon work supported in part by Vidyalankar Institute of technology.

References

[1] Janardhan Iyengar, Ian Swett, Alyssa Wilk," QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 " in Network Working Group Internet-Draft, December 2016

[2] J. Iyengar, I. Swett, "QUIC Loss Recovery And Congestion Control" in QUIC Internet-draft in Sept, 2017.

[3] Belshe, M., Peon, R. and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015.

[4] R. Lychev, S. Jero, A. Boldyreva, C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses" in 2015 IEEE Symposium on Security and Privacy

[5] Evolution of Web Protocols: HTTP/2 and QUIC BY: LASSE LUMIAHO, BINOY CHEMMAGATE, STEN HÄGGLUND, JÖRG OTT, 03 FEBRUARY 2017

[6] How Secure and Quick is QUIC? Provable Security and Performance Analyses Robert Lychev, Samuel Jero, Alexandra Boldyreva, Cristina Nita-Rotaru

[7] https://www.diffen.com/difference/TCP_vs_UDP

[8] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", November 2016.

[9] Thomson, M., Ed. and S. Turner, Ed, Ed., "Using Transport Layer Security (TLS) to Secure QUIC", November 2016.

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[11] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <http://www.rfc-editor.org/info/rfc4821>.

[12] Roskind, J., "QUIC: Multiplexed Transport Over UDP",December 2013, <https://goo.gl/dMVtFi>.

[13] Bishop, M., Ed., "Hypertext Transfer Protocol (HTTP) overQUIC", November 2016.

[14] Langley, A. and W. Chang, "QUIC Crypto", May 2016, http://goo.gl/OuVSxa>.

[15] Ford, B., "Structured Streams: A New Transport Abstraction", ACM SIGCOMM 2007 , August 2007.