IETF QUIC v1 Design

Saleh Alawaji (A paper written under the guidance of Prof. Raj Jain)

Abstract

QUIC is a transport layer protocol developed on top of UDP (User Datagram Protocol) to overcome the inherited issues with TCP (Transmission Control Protocol), such as Head of Line blocking and connection establishment. The primary purpose of QUIC is to reduce connection latency. This paper discusses QUIC packet format, QUIC headers, how QUIC establishes a connection, how QUIC overcomes TCP issues, and the congestion control methods used by QUIC.

Keywords:

QUIC, transport layer protocols, TCP, UDP, Web transport, QUIC connection, QUIC headers, QUIC packet, QUIC congestion control, 0-RTT, 1-RTT.

Table of Contents:

1. Introduction

QUIC is a transport protocol launched by Google back in 2012, and Internet Engineering Task Force (IETF) introduces its first version in 2021. QUIC was developed to overcome multiple issues in TCP protocol. First, in TCP, if a packet is lost, all subsequent packets arriving later will be stopped from reaching the application because TCP uses sequential delivery and window-based flow management. Second, TCP relies on IP address and port number to identify a connection, so if an IP address or port number changes, the connection needs to be re-established again with a costly 3-way handshake because TCP doesn't support any communication reuse context. Third, TCP natively doesn't support multiplexing. A question may be raised why a new protocol needs to be developed instead try to improve TCP. That is because TCP is implemented in the kernel, where QUIC is implemented over UDP, making it more flexible than TCP. Also, QUIC protocol includes multiplexing, cryptography, congestion control, and loss recovery, as shown in Figure 1. Contrastingly, these features are patched into TCP protocol. QUIC has multiple implementations. However, this paper focuses only on IETF QUIC V1 design features.

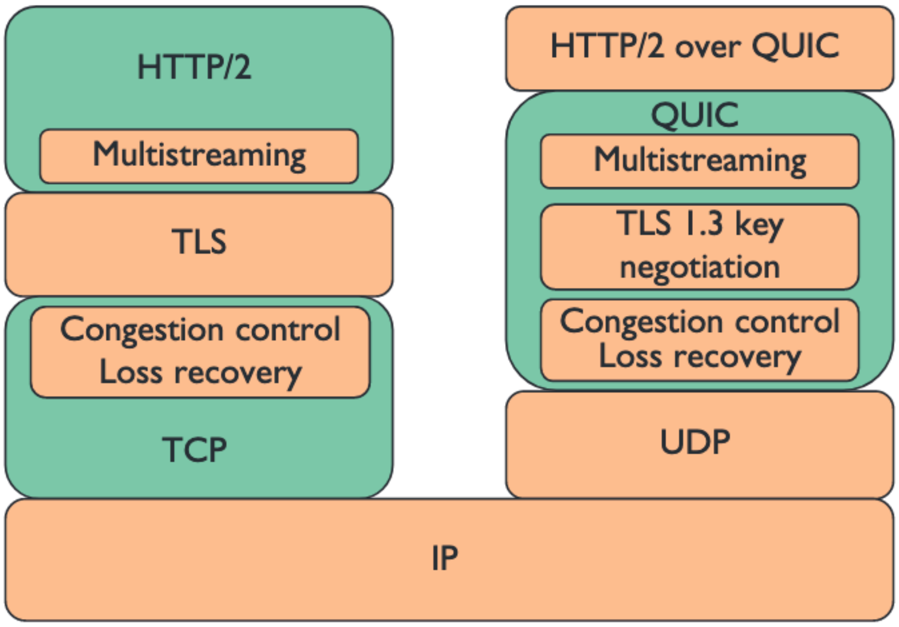

Figure 1:QUIC architecture [Cui17]

2.QUIC Packet

UDP datagrams containing one or more QUIC packets are exchanged between QUIC endpoints. The immutable properties of a QUIC packet are described in this section. Multiple QUIC packets might be carried in a single UDP datagram in one version of QUIC, but the invariant properties only apply to the first packet in a datagram. There are multiple header and packet types in QUIC transport protocol, and the QUIC packet consists of a QUIC header and frames, as shown in Figure 2.

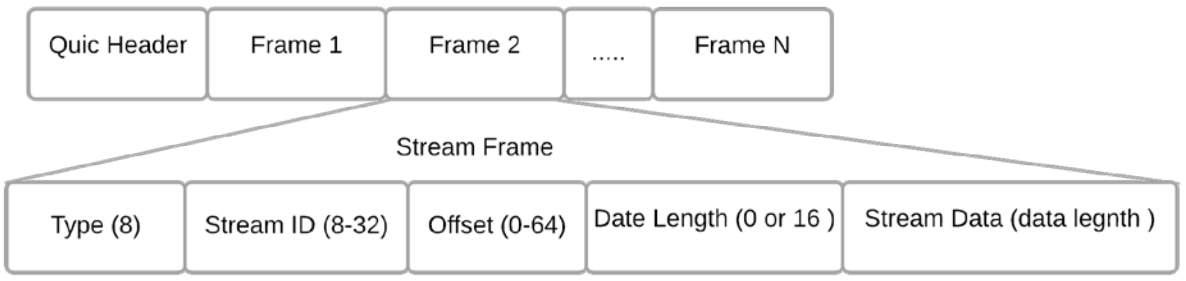

Figure 2:QUIC packet format

2.1 QUIC Header

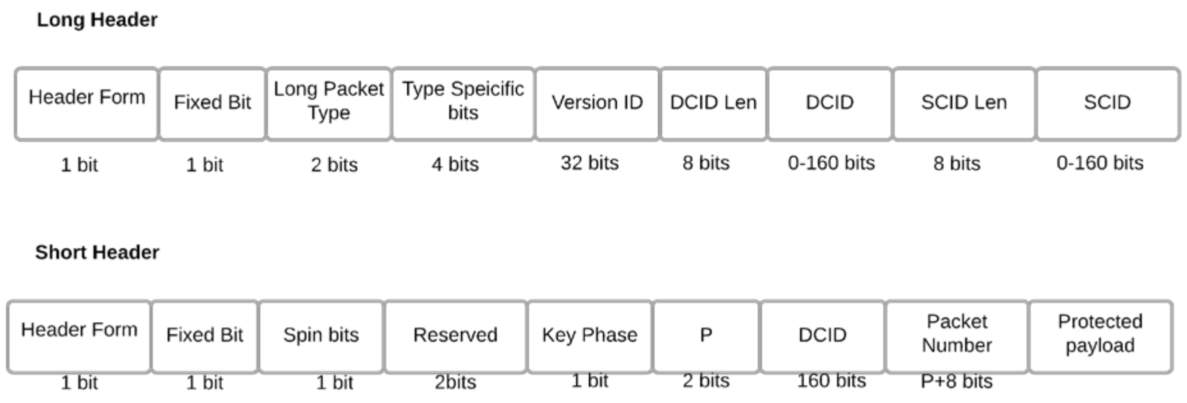

Figure 3:QUIC header based on IETF QUIC V1

QUIC has two different types of headers. The long header is used prior to the connection establishment. The short header is used after the first connection established. The long header contains the following fields in its header, as shown in Figure 3 [RFC 8999]:

- Header Form (HF) - identify the header type
- Fixed Bit (FB) - indicates if the packet is valid or not. If it is set to 0, the packet is invalid
- Long Packet Type (T) - indicates the type of long header packet, as show in Table 1
- Type-Specific Bits (S) - Bits specific for the long header packet types
- Version ID (VID) - 32 bits to identify the version of QUIC
- Destination Connection ID Length (DCID Len)
- Destination Connection ID (DCID)
- Source Connection ID Length (SCID Len)
- Source Connection ID (SCID) [McQuistin21]

The Long Packet Type flag consists of two bits indicating the type of long header, where one indicates a long header. A long QUIC header can be a Version Negotiation packet, Initial packet, 0-RTT packet, Handshake packet, or Retry packet, as shown in Table 1. The short header is only used after key negotiation, in 1-RTT packet, and consists of few fields than the longer header [RFC9000]:

- Header Form (HF)
- Fixed Bit (FB)
- Spin Bit - latency spin bit
- Reserved Bits
- Key Phase - identify packet protection key
- Packet Number Length (P)
- Destination Connection ID (DCID)
- Packet Number
- Packet Payload

Table 1: Long header packet types based on IETF RFC 9000

Type	Name	Description
0x00	Initial	Transports the first CRYPTO frames transmitted by the client and server during the key exchange and the Acknowledgment frames in both directions.
0x01	0-RTT	A 0-RTT packet sends "early" data from the client to the server before the handshake is completed.
0x02	Handshake	Packet is for sending and receiving encrypted handshake messages and acknowledgments between the server and the client.
0x03	Retry	The packet contains a server-generated address validation token. It's used by a server that wants to retry a connection.

2.2 Connection Identifier

A Connection ID or CID's main function is to ensure that packets for a QUIC connection are not delivered to the wrong endpoint due mainly to addressing changes at lower protocol levels (UDP, IP). Each endpoint selects connection IDs using an implementation-specific technique, which allows packets with that connection ID to be routed back to the endpoint and identified by it. Each connection contains a set of identifiers, also known as connection IDs, that may be used to identify it. Endpoints select their own connection IDs; each endpoint selects CID that its peer use. CID must be unique for every connection to avoid any middlebox forwarding traffic to a wrong endpoint [RFC9000].

There are two field for CID: one for the source and the other for the destination. For the long header, the packet contains the source and destination connection ID and the initial Connection ID assigned by the sender in the source CID filed. The initial value for CID is 0, and it's incremented by one for every new CID issued, in case of the short header has only the destination CID used. If a CID isn't required to route to the right endpoint, a zero-length CID can be utilized. In the case of multiplexing connections on the same local IP address and port, CID is required for routing [RFC9000].

2.3 Stream Frame

IETF QUIC v1 Design

QUIC is a multiplexed protocol that run over UDP. QUIC streams are a simple abstraction for creating a reliable bidirectional byte stream. Streams may be used to frame application messages of any size-up to 2^(64) bytes can be carried on a single stream-but they're light enough that a different stream can be utilized for each one when transmitting a succession of little messages. Stream IDs, which are statically assigned as odd IDs for client-initiated streams and even IDs for server-initiated streams to minimize clashes, are used to identify streams. The least significant bits in Stream ID are used to determine which endpoint originated the stream and whether it is bidirectional or unidirectional. Stream Offset is like TCP Offset, where Stream Offset indicates the byte offset in the steam. Multiple streams of data can flow across a QUIC connection. This guarantees that if a packet is lost, it will not affect other streams, which avoids the Head-of-Line Blocking issue that is a well-knowing issue with TCP [Dellaverson17].

2.4 Packet Number

Packet Number is a number used to identify cryptography nonce for packet protection. It ranges from 0 to 2^(62-1). This limitation in range due to the Packet Number should be in the target Acknowledgement filed in an ACK frame. If a long or short header is used, the Packet Number length can be reduced to 4 Byte. The Packet Number is different for incoming and outgoing traffic for every endpoint. QUIC divides Packet Number into three spaces: (i) Initial space: contains all initial packets. (ii) Handshake space: contains all handshake packets. (iii) Application data space: contain all 0-RTT & 1-RTT packets [RFC9000].

A packet number space is a context in which a packet can be processed and acknowledged. Initial packets, for example, can only be sent using initial packet protection keys. Handshake packets, likewise, are only sent at the handshake encryption level. This ensures that the data sent in the various packet sequence number spaces are cryptographically separated. Each space starts with a 0-packet number and succeeding packets must increase the packet number by one. 0-RTT and 1-RTT data can be found in the same packet number space to reduce the complexity for the recovery algorithm between 0-RTT and 1-RTT packets [Kumar20].

3.QUIC Connection

3.1 Connection Establishment

For establishing a transport connection, QUIC combines cryptographic and transport handshake. In the initial connection, the client doesn't have information about the server. After the first successful handshake, the client stores the server information so, in any succeeding connection, the client does not require an extra round-trip time (RTT). Also, data can be sent directly after the first handshake without waiting for the server's reply, as shown in Figure 4.

First-time Connection Establishment: For the first time a client tries to establish a connection with a server, the client will send an inchoate client hello (CHLO) message to the server to receive a reject (REJ) message. The REJ message contains the server configuration, authentication certificate, signature for the server certificate, and source-address token. The source-address token is used to verify the identity of the client in future communications. The client uses the information provided by the REJ message to construct the complete CHLO message. Then, the client sends a complete CHLO message that contains a temporary Diffie-Hellman public key for the client [Langley17].

0-RTT connection establishment: After the initial handshake, the client has the initial keys for the connection. Once the client sends a complete CHLO message to the server, it can start sending data before receiving the Server Hello (SHLO) message. This is how QUIC achieves 0-RTT. After the client receives SHLO, the client starts sending data using final keys calculated from the information provided in the SHLO message [Langley17].

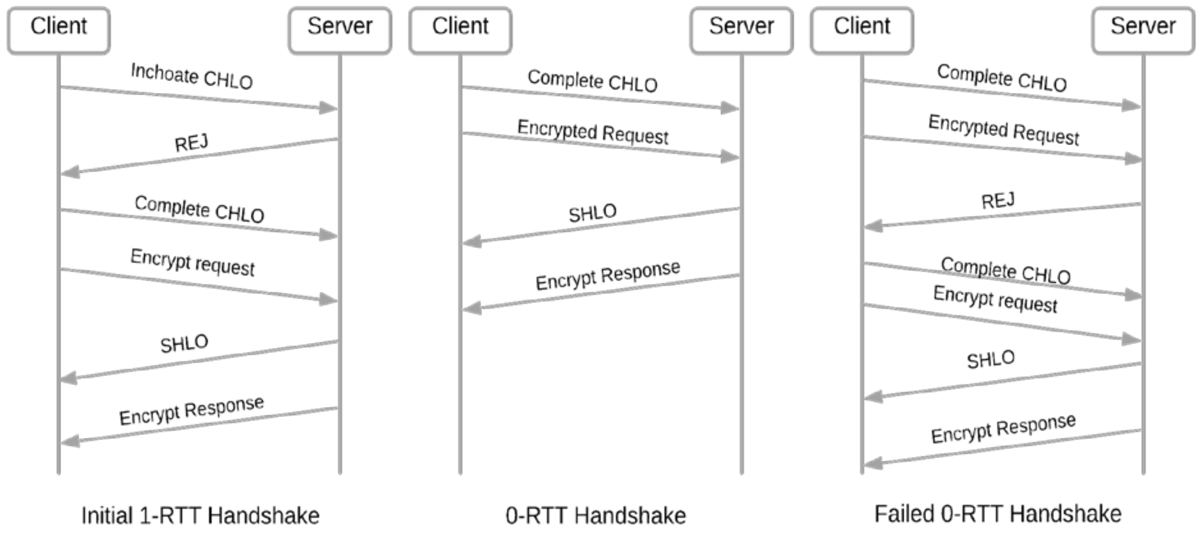

Figure 4:QUIC Connection establishment 0-RTT and 1-RTT handshake

3.2 Connection Migration

QUIC allows a connection between a client and a server to remain active even after the client IP or port changes because QUIC uses Connection ID to identify a connection. In the case of TCP, a connection is identified by IP addresses and port numbers. For instance, after a change happens to the client's network, the client can use the Connection ID previously initialized from the prior connection to initiate a Connection Migration (CM) request to the server using a probing packet (PATH_CHALLENGE). If the server accepts the Connection Migration request, it starts path validation by sending PATH_CHALLENGE and PATH_RESPONSE frames to confirm that the client's ownership of the new address. Then the client needs to response to PATH_CHALLENGE that has been sent by the server by sending a Path_RESPONSE frame, as shown in Figure 5. After the path validation, the peers can send and receive data based on the new address. Also, the server can ask the client to contact it at a different IP address. However, endpoints must maintain a consistent address throughout the length of the handshake. Before the handshake is verified, an endpoint can't initiate connection migration [Dellaverson17].

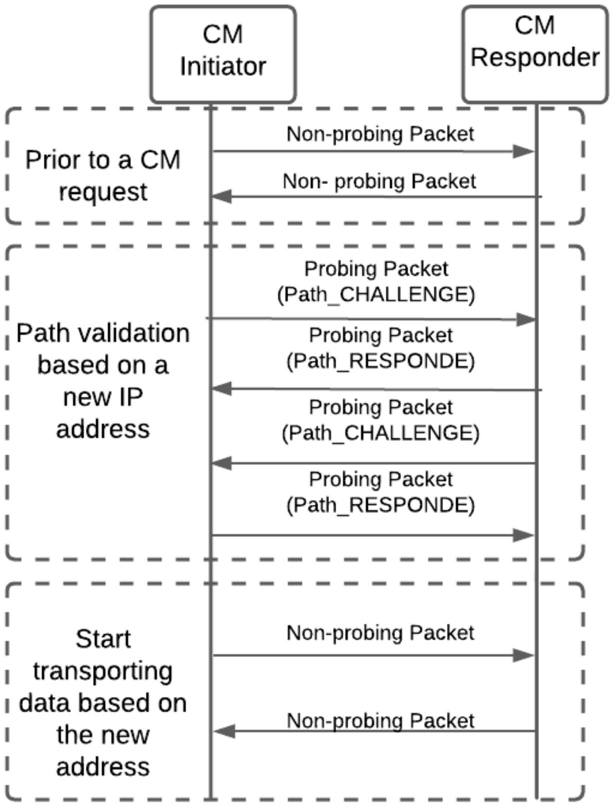
CM Initiator
CM Responder
Prior to a CM request
Non-probing Packet
Non- probing Packet
Path validation based on a new IP address
Probing Packet (Path_CHALLENGE)
Probing Packet (Path_RESPONDE)
Probing Packet (Path_CHALLENGE)
Probing Packet (Path_RESPONDE)
Start transporting data based on the new address
Non-probing Packet
Non-probing Packet

Figure 5: QUIC connection migration

3.3 Flow Control

Credit-based flow control is used by QUIC. The absolute byte offset inside each stream that a QUIC receiver is willing to receive data is advertised.The receiver regularly transmits window update frames that raise the stated offset limit for that stream, when data is transferred, received, and delivered on that stream. This allows the peer to send more data on that stream. The bytes transmitted and the highest received offset are aggregated across all streams in connection-level flow control, which functions similarly to stream-level flow control. Flow control restricts the buffer size that the receiver must maintain when an application reads data slowly from QUIC's receive buffers. A slowly draining stream might fill the whole receive buffer of a connection, preventing the sender from delivering data on additional streams. QUIC mitigates this risk by restricting the amount of buffer that a single stream can utilize. As a result, QUIC uses connection-level flow control, which limits the total buffer a sender may utilize at the receiver across all streams, as well as stream-level flow control, which limits the buffer a sender can spend on a single stream[Langley17].

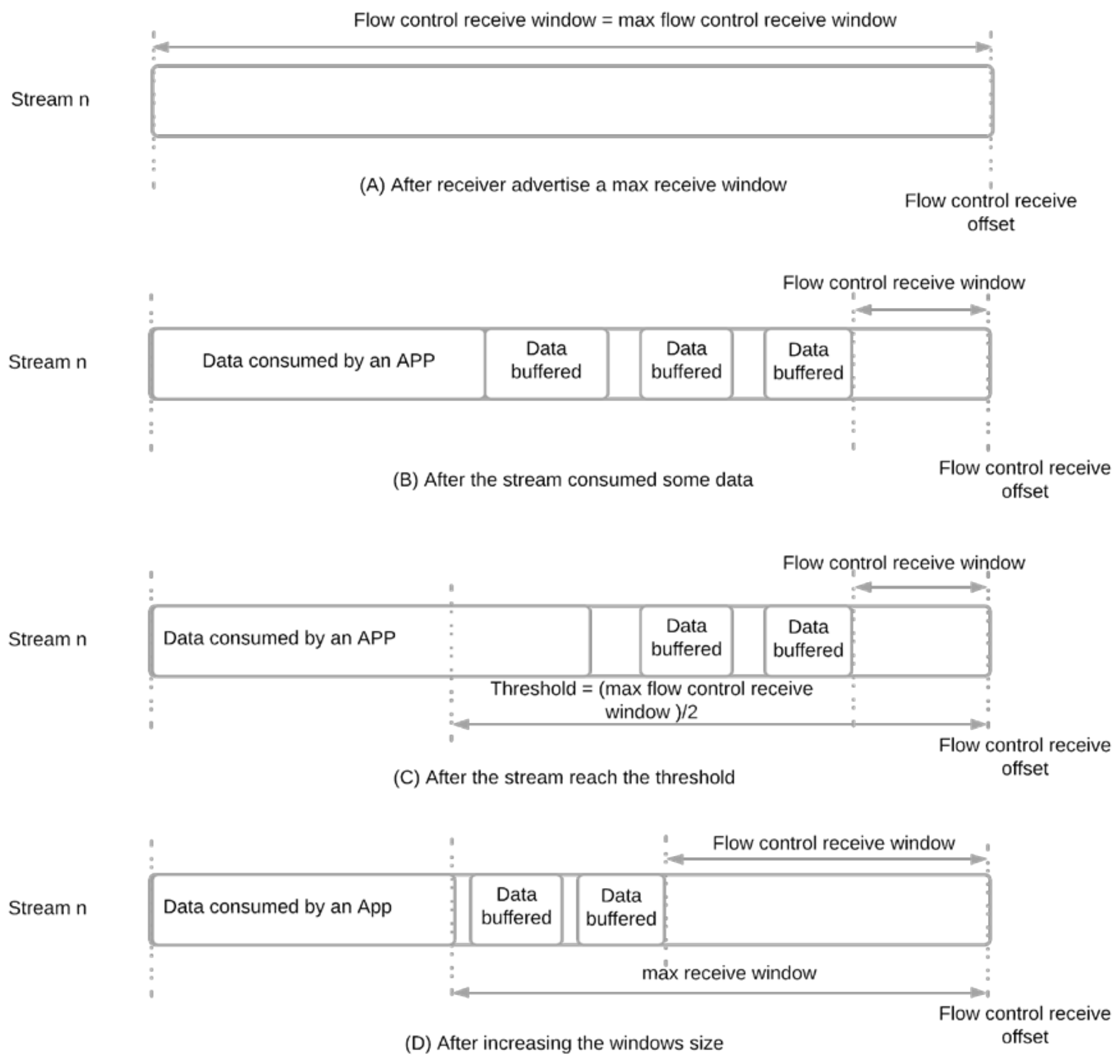
Flow control receive window = max flow control receive window
Stream n
(A) After receiver advertise a max receive window
Flow control receive offset
Flow control receive window
Stream n
Data consumed by an APP
Data buffered
Data buffered
Data buffered
(B) After the stream consumed some data
Flow control receive offset
Flow control receive window
Stream n
Data consumed by an APP
Data buffered
Data buffered
Threshold = (max flow control receive window)/2
Flow control receive offset
(C) After the stream reach the threshold
Flow control receive window
Stream n
Data consumed by an App
Data buffered
Data buffered
max receive window
Flow control receive offset
(D) After increasing the windows size

Figure 6: QUIC stream flow control

To elaborate more on how QUIC manages the stream-flow control. For example, initially, the receiver advertises the max flow receive window during the handshake, as shown in Figure 6, part (A). Then, the sender starts sending traffic based on flow control receive offset advertised by the receiver. Some of the traffic will be consumed by an application at the receiver end, while the remaining packet will be buffered, as shown in Figure 6, part (B). The gaps between data buffered shown in Figure 6, part (B) indicate out-of-order delivery or packet loss/drop. The sender will keep a record of every packet's offset that has been sent to guarantee that the receiver is willing to receive an additional packet. As shown in Figure 6, part (C), the receiver will trigger a window update once the threshold is reached. The threshold is implementation-specific and differs based on QUIC implementation. However, the majority of QUIC implementations specify it as half of the max receive window. The receiver will trigger a window update frame based on the following Boolean condition [Volodina20]:

(Flow control receive offset - consumed bytes) < (max receive window)/2

After the receiver advertises the new max flow receive window, the sender will keep sending packets based on the new parameters, as shown in Figure 6, part(D). The connection-level flow control follows the same principle, but it aggregates the flow control for all streams.

4.QUIC Congestion & Recovery

IETF QUIC has multiple techniques to handle congestion & recovery. QUIC implements loss detection, congestion control, and estimating RTT like TCP and only differs from TCP techniques in the algorithm implementation. Also, IETF standardization of QUIC allows the usage of congestion controller as an extension.

4.1 Estimating RTT

QUIC uses an RTT sample to detect any delay or lost packet in a connection. An RTT sample is generated by measuring the time between sending a packet and receiving the acknowledgment. Each endpoint shares its sample RTT with its peer. Based on the shared RTT samples per peer, each endpoint calculates three additional variables for delay detection: (i) min_rtt: The sender's estimate of the minimal RTT seen over time for a specific network route. min_rtt set to the lowest value of all RTT Sample. (ii) smoothed_rtt: a moving average of an endpoint's RTT samples that is exponentially weighted. (iii) rtt_var: the variation in the RTT samples using a mean variation [RFC9002].

4.2 Loss Detection

QUIC uses acknowledgments to discover missing packets. If a packet is lost, the QUIC transport must either retransmit the data, deliver an updated frame, or trash the frame to recover from the loss. Loss detection is done separately for each packet number space to avoid head-of-line blocking. Acknowledgment-Based detection uses packet threshold and time threshold, where time threshold is calculated based on RTT estimate variables defined in the previous section. A packet is considered missing if it's not acknowledged and the next package's acknowledgment was received or based on the packet threshold. A probe timeout (PTO) is used by QUIC senders to verify that acknowledgments are received. When ack-eliciting packets are not acknowledged within the required time frame, or the server has not confirmed the client's address, a Probe Timeout (PTO) is triggered, which sends one or two probe datagrams. A PTO allows a connection to recover from packet loss or acknowledgment loss [RFC9002].

4.3 Congestion Control

QUIC offers a somewhat different congestion management environment than TCP. For starters, it incorporates current loss-recovery algorithms like retransmission timeout (RTO), tail loss probe, Forward RTO-Recovery (F-RTO), and Early Retransmit from the start. In addition, it provides more thorough feedback for loss detection. It employs a monotonically growing packet number, so it does not retransmit at the packet level (only on a per-frame base). This enables

QUIC to identify retransmissions from the initially delivered packets, preventing retransmission uncertainty. QUIC also stores the time difference between when a packet is received and when the ACK is issued. With this information, the original sender may better estimate the RTT of the route.QUIC also uses TCP's selective acknowledgment mechanism and allows up to 255 ACK ranges, which makes it more resistant to reordering and loss [Cui17]. In addition to these congestion management techniques, QUIC has an interface that allows the usage of different congestion algorithms [Polese19]. However, Multiple frames of various sorts may be included in QUIC packets. The recovery methods guarantee that data and frames that need to be sent reliably are acknowledged or declared lost, and replacement packets are sent as needed. Recovery and congestion management logic are affected by the kinds of frames in a packet, and any congestion control can be used with QUIC under condition it follows specific criteria defined in IETF RFC8085, but it will not be discussed in this paper.

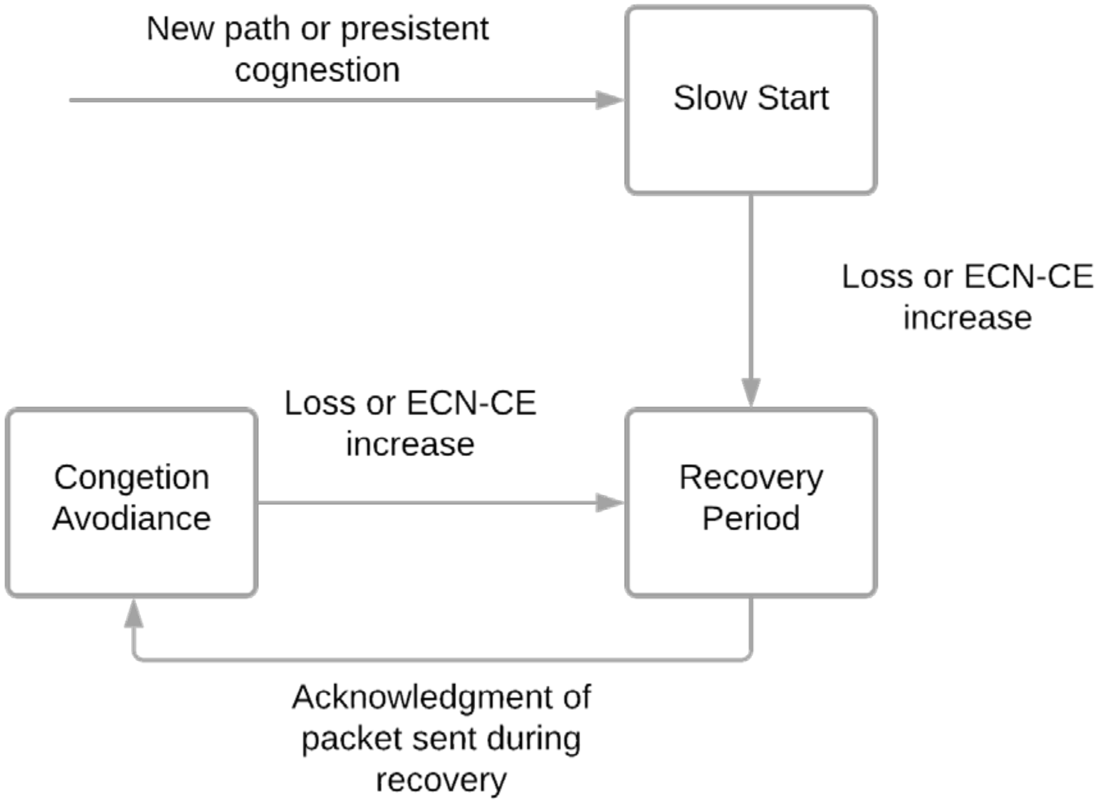

Figure 7: Congestion Control States and Transitions

IETF QUIC uses NewReno for sender control congestion, which is similar to TCP NewReno. QUIC starts every connection in Slow Start state. For instance, a sender starts in a Slow Start state, which is below the slow start threshold. Then congestion window grows by the number of bytes acknowledged. The sender will enter the Recover Period state due to lost packet or the Explicit Congestion Notification Congestion Experience (ECN-CE) increased. Once the sender enters the Recovery Period State, its slow start threshold decreases to half of the congestion windows once per RTT. Then, suppose the sender receives all acknowledgments while it is in the Recovery Period state. In that case, it will enter the Congestion Avoidance state and remain in this state until ECN-CE increases or the packet is lost, as shown in Figure 7 [RFC9002].

5.Summary & Future Consideration

We have discussed how most current issues with TCP, such as Head-of-Line blocking, IP address change, and costly handshake, have been remediated by QUIC. Furthermore, the design techniques used for congestion control and recovery and how QUIC handles multiplexing, connection establishments, and connection migration. Even though QUIC focuses on web transport, it still has potential in other areas, and IETF is currently working in multiple extensions for tunneling, unreliable datagram, and multipath QUIC. First, the IETF MASQUE group is working on generalization tunneling, leveraging QUIC unified congestion control to reduce latency in proxies and VPNs. Second, IETF is working on unreliable datagram for real-time applications. Last, QUIC Multipath extension keeps the single-path design features of QUIC while leveraging several pathways for a single connection [Kosek21].

References

[RFC 9000] J. Iyengar, M. Thomson, "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force (IETF), 2021, https://www.rfc-editor.org/rfc/rfc9000.html

[RFC 9002] J. Iyengar, I. Swett, "RFC9002:QUIC Loss Detection and Congestion Control," Internet Engineering Task Force (IETF), 2021, https://datatracker.ietf.org/doc/html/rfc9002

[RFC 8999] M. Thomson, "RFC 8999: Version-Independent Properties of QUIC," Internet Engineering Task Force (IETF), 2021, https://www.rfc-editor.org/rfc/rfc8999.html

[Langley17] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17), Association for Computing Machinery, New York, NY, USA,2017, 183-196. DOI:https://doi.org/10.1145/3098822.3098842

[Dellaverson17] J. Dellaverson, T. Li, Y. Wang, J. Iyengar, A. Afanasyev, L.Zhang "Understanding Quic," 2017, https://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf

[Kumar20] P. Kumar, "QUIC (Quick UDP Internet Connections)--A Quick Study," arXiv preprint arXiv:2010.03059, 2020, https://arxiv.org/abs/2010.03059

[Polese19]	M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella and M. Zorzi, "A Survey on Recent Advances in Transport Layer Protocols," in IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3584-3608, Fourthquarter 2019, doi: 10.1109/COMST.2019.2932905, https://ieeexplore.ieee.org/abstract/document/8786240
[Kosek21]	M. Kosek, T. Shreedhar, V. Bajpai. "Beyond QUIC V1: A First Look at Recent Transport Layer IETF Standardization Efforts," IEEE Communications Magazine 59.4, 2021,: 24-29. Crossref. Web, https://arxiv.org/abs/2102.07527
[McQuistin21]	S. McQuistin, V. Band, D. Jacob, C. Perkins, " Describing QUIC's Protocol Data Units with Augmented Packet Header Diagrams." Internet Engineering Task Force (IETF), 2021, from https://www.ietf.org/id/draft-mcquistin-quic-augmented-diagrams-05.html
[Volodina20]	E. Volodina and E. P. Rathgeb, "Flow Control in the Context of the Multiplexed Transport Protocol QUIC," 2020 IEEE 45th Conference on Local Computer Networks (LCN), 2020, pp. 473-478, doi: 10.1109/LCN48667.2020.9314796. from https://doi-org.libproxy.wustl.edu/10.1109/LCN48667.2020.9314796
[Cui17]	Y. Cui, T. Li, C. Liu, X. Wang and M. Kuhlewind, "Innovating Transport with QUIC: Design Approaches and Research Challenges," in IEEE Internet Computing, vol. 21, no. 2, pp. 72-76, Mar.-Apr. 2017, doi: 10.1109/MIC.2017.44, https://ieeexplore.ieee.org/document/7867726

Acronyms

ACK	Acknowledgment
APP	Application
CHLO	Client Hello message
CID	Connection Identifier
CM	Connection Migration
ECN-CE	Explicit Congestion Notification - Congestion Experience
F-RTO	Forward - Retransmission Timeout
IETF	Internet Engineering Task Force
PTO	Probe Timeout
REJ	Reject message
RTO	Retransmission Timeout
RTT	Round Trip Time
SHLO	Server Hello message
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

IETF QUIC v1 Design

Last modified on December 15, 2021
This and other papers on recent advances in networking are available online at
http://www.cse.wustl.edu/~jain/cse570-21/index.html
Back to Raj Jain's Home Page