An Outline of QUIC, the Future HTTP Standard

Aaron Xia, EE Dept. of Tsinghua University

December, 2018

Contents

1 Introduction

QUIC (Quick UDP Internet Connection) is a new multiplexed and secure transport atop UDP operating entirely in userspace, designed from the ground up and optimized for HTTP/2 semantics. While built with HTTP/2 as the primary application protocol, QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. QUIC provides *multiplexing* and *flow control* equivalent to HTTP/2, security equivalent to TLS, and connection semantics, reliability, and congestion control equivalent to TCP.

2 Conventions and Definitions

All integer values used in QUIC are in little-endian byte order (the least significant byte has the lowest address), and not in network byte order (big-endian). QUIC does not enforce alignment of types in dynamically sized frames.[1]

Before the explanation of QUIC protocol, I'd like to specify some terms that are to be used throughout the article.

- **Stream**: A *bi-directional* flow of bytes across a logical channel within a QUIC connection.
- **Connection**: A conversation between two endpoints with a single encryption context that multiplexes streams within it.
- **Connection ID**: The identifier for a QUIC connection.[2]
- **QUIC Packet**: A well-formed UDP payload that can be parsed by a QUIC receiver. QUIC packet size in this document refers to the UDP payload size.
- **Frame**: Frame is the most direct carrier of data which endpoints exchange in a successfully established HTTP/2 connection.
- **RTT**: Round-trip time (RTT) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgement of that signal to be received with the propagation time between two endpoints included. [3]

[1] Ports and addresses are always specified in calls to the socket functions using the network byte order convention. This convention is a method of sorting bytes that is independent of specific machine architectures. Host byte order, on the other hand, sorts bytes in the manner which is most natural to the host software and hardware. There are two common host byte order methods:

- Little-endian byte ordering places the least significant byte first. This method is used in Intel microprocessors, for example.
- Big-endian byte ordering places the most significant byte first. This method is used in IBM® z/Architecture® and S/390® mainframes and Motorola microprocessors, for example.

For instance, a piece of two-bytes data is expressed with b45d in hex. Then in the big-endian fashion or network byte order, the data piece should be stored in the form of b45d (It's natural). However, the Intel microprocessor store the data piece with 5db4 (the reversal byte order), which follows a little-endian manner.

[2] Streams are identified by an integer. Stream identifiers are assigned to streams by the endpoint initiating the stream.

[3] RTT updates in an iterative manner:

$$\text{RTT} = \alpha \times \text{RTT} + (1 - \alpha) \times \text{new\_RTT\_sample} \tag{1}$$

where α is a constant weight factor ($0 \leq \alpha \leq 1$)

3 Basic Mechanism of QUIC

QUIC is functionally equivalent to TCP+TLS+HTTP/2, but *implemented on top of UDP*.

3.1 Characteristics

- **Low Connection Latency**: QUIC connections are commonly 0-RTT, meaning that on most QUIC connections, data can be sent immediately without waiting for a reply from the server, as compared to the 1-3 roundtrips required for TCP+TLS before application data can be sent.

- **Congestion Control**: QUIC has pluggable congestion control and rich signaling. One example of richer information is that each packet, both original and retransmitted, carries a new packet sequence number. This allows a QUIC sender to distinguish ACKs for retransmissions from ACKs for original transmissions, thus avoiding TCP's retransmission ambiguity problem. QUIC ACKs also explicitly carry the delay between the receipt of a packet and its acknowledgment being sent, and together with the monotonically-increasing packet numbers.

- **Stream and Connection Flow Control**: QUIC implements *stream- and connection-level* flow control, closely following HTTP/2's flow control. The WINDOW_UPDATE [4] frame is used to implement the flow control. Different from the originally defined HTTP/2 flow-control, [5] QUIC's stream-level control works as follows: A QUIC receiver advertises the absolute byte offset within each stream upto which the receiver is willing to receive data. As data is sent, received, and delivered on a particular stream, the receiver sends WINDOW_UPDATE frames that increase the advertised offset limit for that stream, allowing the peer to send more data on that stream. The connection-level control works almost the same as that of the stream-level control except that the bytes delivered and highest received offset are all aggregates across all streams.

- **Multiplexing**: Similar to HTTP/2, QUIC is free of head-of-line blocking (a line of packets is held up by the first packet) issues. Multiplexing of requests is achieved by having each HTTP request/response exchange associated with its own stream. In HTTP 1.x, there is actually one "stream" (if we adopt the conception of HTTP/2) per connection, thus when one packet is dropped due to network faults or forwarding black hole, the entire connection has to be blocked to wait for retransmission of that packet.

- **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. QUIC protects connections from witting or unwitting middlebox manipulation of end-to-end communication. It sounds quite familiar, eh? So easy to

[4] The payload of a WINDOW_UPDATE frame is one reserved bit plus an unsigned 31-bit integer indicating the number of octets or bytes that the sender can transmit in addition to the existing flow-control window. The legal range for the increment to the flow-control window is 1 to $2^{31} - 1$ (2,147,483,647) octets.

[5] HTTP/2 defines only the format and semantics of the WINDOW_UPDATE frame. The official HTTP/2 document does not stipulate how a receiver decides when to send this frame or the value that it sends, nor does it specify how a sender chooses to send packets. Implementations are able to select any algorithm that suits their needs.

let people think of TLS. [6]

- **Connection Migration**: TCP connections are specified by source address, source port, destination address and destination port. A typical problem for TCP connection is NAT binding expire. QUIC connections are identified by a 64-bit connection ID which is not affected by IP address changes and NAT binding.

3.2 QUIC Crypto

The handshake and encryption detail of the crypto is omitted in this section. To find out more, please refer to the official document of QUIC crypto.

With the current QUIC crypto protocol [7] , when the client has cached information about the server, it can establish an encrypted connection with no round trips. TLS, in contrast, requires at least two round trips (counting the TCP 3-way handshake). QUIC handshakes should be ∼5 times more efficient than common TLS handshakes (2048-bit RSA) and at a greater security level. The mechanism is similar to the long-term TCP connection where a connection credential is cached both at the server end and the client end.

Very few protocols on the Internet work without at least one initialisation round trip. Most protocols have around trip due to TCP and TLS-based protocols additionally have at least one more before application data can flow. Both of these rounds trips exchange nonces: sequence numbers (or SYN cookies) in the case of TCP and cryptographic random values (client_random and server_random) in TLS. The TCP nonce prevents IP address spoofing [8] and the TLS nonce prevents replay attacks [9] . Any protocol which seeks to eliminate round trips has to somehow address these two problems. QUIC deals with the two problems separately (only one, actually).

3.2.1 IP Address Spoofing

The IP address spoofing problem is handled by issuing the client, on demand, a "source-address token". This is an opaque byte string from the client's point of view. From the server's point of view it's an authenticated-encryption block (e.g. AES-GCM) that contains, at least, the client's IP

[6]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. While some of these are active attacks, others are mechanisms used by middleboxes in the network sometimes in an attempt to transparently improve TCP performance. However, even "performance-enhancing" middleboxes still effectively limit the evolvability of the transport protocol, as has been observed in the design of MPTCP and in its subsequent deployability issues.

[7]The QUIC crypto protocol is the part of QUIC that provides transport security to a connection.The QUIC crypto protocol is *destined to die* It will be replaced by TLS 1.3 in the future, but QUIC needed a crypto protocol before TLS 1.3 was even started.

[8]IP packets are used in applications that use the Internet as their communications medium. Usually they are generated automatically for the user, behind the scenes; the user just sees the information exchange in the application. These IP packets have the proper source and destination addresses for reliable exchange of data between two applications. The IP stack in the operating system takes care of the header for the IP datagram. However, you can override this function by inserting a custom header and informing the operating system that the packet does not need any headers. What is the advantage of sending a spoofed packet? It is that the sender has some kind of malicious intention and does not want to be identied. You can use the source address in the header of an IP datagram to trace the sender's location.

[9]A replay attack occurs when an attacker copies a stream of messages between two parties and replays the stream to one or more of the parties. Unless mitigated, the computers subject to the attack process the stream as legitimate messages, resulting in a range of bad consequences, such as redundant orders of an item.

address and a timestamp by the server. The server will only send a source address token for a given IP to that IP. Receipt of the token by the client is taken as proof of ownership of the IP address in the same way that receipt of a TCP sequence number is.

Clients can include the source address token in future requests in order to demonstrate ownership of their source IP address. If the client has moved IP addresses, the token is too old, or the client doesn't have a token, then the server may reject the connection and return a fresh token to the client. But if the client has remained on the same IP address then it can reuse a source-address token to avoid the round trip needed to obtain a fresh one.

The lifetime of a token is a matter for the server but, since source address tokens are bearer tokens, they can be stolen and reused in order to bypass IP address based restrictions. (Although the attacker would not receive the response.) Source address tokens can also be collected and possibly used after ownership of the IP address has changed (i.e. in a DHCP pool). Reducing the lifetime of tokens ameliorates both of these concerns at the cost of reducing the number of requests that can be handled without additional round trips.

3.2.2 Replay Protection

QUIC just gives up the solution to the issue, instead, it's up the application layer to ensure that any such information is safe is replayed by attackers. Precisely speaking, the official document claims:

> In TLS, each side generates an nonce which is used to ensure that the other party is fresh by forcing them to include the (assumed to be unique for all time) value in the key derivation. Without around trip the client can still include an nonce and so ensure that the server is fresh, but the server doesn't have a chance to do so for the client.
>
> Providing replay protection without input from the server is fundamentally very expensive. It requires consistent state at the server. Although this is reasonable if the server is a single machine, modern websites are spread around the world.

4 Life of a QUIC Connection

4.1 Connection Establishment

QUIC's connection establishment intertwines version negotiation with the crypto and transport handshakes to reduce connection establishment latency.

4.1.1 Version Negotiation

The version negotiation could be described briefly as follows.

Each of the initial packets sent from the client to the server must set the version flag, and must specify the version of the protocol being used. When the server receives a packet, it will compare the version applied with the support version list of itself. If the client's version is supported, they will use this version for the rest of the lifetime of the connection. In this case, all packets sent by the server will have the version flag off. Indeed, the version flag is an indicator only for negotiation and once their version negotiation is finished, the version flag is always set off.

If the version in use is not acceptable to the server, a 1-RTT delay will be incurred. The server will send a Version Negotiation Packet to the client which has the version flag set and includes the server's support version list. When the client receives the Version Negotiation Packet, it will select an acceptable version protocol and resend all packets using this version with their version flags set. On receiving the packets resent from the client, the server will send Regular Packets to end the negotiation process. Eventually, when the client receives the first Regular Packet, the negotiation comes to an end and the client sends all subsequent packets using this protocol with the version flag set off.

Since the mechanism described above has an implicit assumption that the flag-on packet arrives sooner than the flag-off one, careful readers may notice there could be an issue in the entire procedure that what if a packet with the version flag on arrives later than the packet with the version flag off. Well, don't worry. After the server receives the first packet from the client with the version flag off, it must ignore any (possibly delayed) packets with the version flag on.

In order to avoid downgrade attacks [10], 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.

4.1.2 Transport Handshakes

QUIC is a datagram protocol, and the full payload of each datagram (above the UDP layer) are authenticated and encrypted once keys have been established. The underlying datagram protocol provides the crypto layer with the means to send reliable, arbitrary sized messages. These messages have a uniform, key-value format.

The keys are 32-bit tags. This seeks to provide a balance between the tyranny of magic number registries and the verbosity of strings. As far as the wire protocol is concerned, these are opaque, 32-bit values and, in this document, tags will often be written like **EXMP**. Although it's written as a string, it's just a mnemonic for the value 0x504d5845. That value, in little-endian, is the ASCII string **E X M P**. If a tag is written in ASCII but is less than four characters then it's as if the remaining characters were NUL. So **EXP** corresponds to 0x505845. If the tag value contains bytes outside of the ASCII range, they'll be written in hex, e.g. 504d5845. All values are little-endian unless otherwise noted.

A handshake message consists of:

1. The tag of the message.
2. A uint16 containing the number of tag-value pairs.
3. Two bytes of padding which should be zero when sent but ignored when received.
4. A series of uint32 tags and uint32 end offsets, one for each tag-value pair.The tags must be strictly monotonically increasing, and the end-offsets must be monotonic non-decreasing. The end offset gives the offset, from the start of the value data, to a byte one beyond the end of the data for that tag. (Thus the end offset of the last tag contains the length of the value data)

[10]A downgrade attack is a form of cryptographic attack on a computer system or communications protocol that makes it abandon a high-quality mode of operation (e.g. an encrypted connection) in favor of an older, lower-quality mode of operation (e.g. cleartext) that is typically provided for backward compatibility with older systems.

5. The value data, concatenated without padding.

The tag-value format allows for an efficient binary search for a tag after only a small fraction of the data has been validated. The requirement that the tags be strictly monotonic also removes any ambiguity around duplicated tags.

Although the 32-bit lengths are currently more than needed, 16-bit lengths ran the risk of being insufficient to handle larger, post-quantum values.

Any message may contain a padding (PAD) tag. These can be used to to defeat traffic analysis. Additionally, we may define a global minimum size for client hellos to limit amplification attacks. Client hellos that are smaller than the minimum would need a PAD tag to make up the difference.

4.2 Data Transfer

Closely following HTTP/2, QUIC sends data inside streams.

4.2.1 Stream Creation

Stream creation is done implicitly, by sending a STREAM frame for a given stream.

Streams are identified with an unsigned 31-bit integer. Streams initiated by a client MUST use odd-numbered stream identifiers; those initiated by the server MUST use even-numbered stream identifiers. A stream identifier of zero (0x0) is used for connection control messages; the stream identifier of zero cannot be used to establish a new stream. Stream 1 is reserved for the crypto handshake, which should be the first client-initiated stream. When using HTTP/2 over QUIC, Stream 3 is reserved for transmitting compressed headers for all other streams, ensuring reliable in-order delivery and processing of headers.

Stream identifiers cannot be reused. Long-lived connections can result in an endpoint exhausting the available range of stream identifiers. A client that is unable to establish a new stream identifier can establish a new connection for new streams. A server that is unable to establish a new stream identifier can send a GOAWAY frame so that the client is forced to open a new connection for new streams.

If the endpoint receiving a STREAM frame does not want to accept the stream, it can immediately respond with a RST_STREAM frame. Note, however, that the initiating endpoint may have already sent data on the stream as well; this data must be ignored.

4.2.2 Stream Termination

Either QUIC endpoint can terminate a stream normally. There are three ways that streams can be terminated:

1. **Normal termination**: Since streams are bidirectional, streams can be "half-closed" [11] or "closed". When one side of the stream sends a frame with the FIN bit set to true, the stream is considered to be "half-closed" in that direction. A FIN indicates that no further data will be sent from the sender of the FIN on this stream. When a QUIC endpoint has both sent and received a FIN, the endpoint considers the stream to be "closed". While the FIN should be

[11] There are two half-closed states, local and remote, according to different perspectives. A stream that is in the "half-closed (local)" state cannot be used for sending frames other than WINDOW_UPDATE, PRIORITY, and RST_STREAM; A stream that is "half-closed (remote)" is no longer being used by the peer to send frames. In this state, an endpoint is no longer obligated to maintain a receiver flow-control window.

sent with the last user data for a stream, the FIN bit can be sent on an empty stream frame following the last data on the stream.

2. **Abrupt termination**: Either the client or server can send a RST_STREAM frame for a stream at any time. A RST_STREAM frame contains an error code to indicate the reason for failure (error codes are listed later in the document.) When a RST_STREAM frame is sent from the stream originator, it indicates a failure to complete the stream and that no further data will be sent on the stream. When a RST_STREAM frame is sent from the stream receiver, the sender, upon receipt, should stop sending any data on the stream. The stream receiver should be aware that there is a race between data already in transit from the sender and the time the RST_STREAM frame is received. In order to ensure that the connection-level flow control is correctly accounted, even if a RST_STREAM frame is received, a sender needs to ensure that either: the FIN and all bytes in the stream are received by the peer or a RST_STREAM frame is received by the peer. This also means that the sender of a RST_STREAM frame needs to continue responding to incoming STREAM_FRAMEs on this stream with the appropriate WINDOW_UPDATEs to ensure that the sender does not get flow control blocked attempting to deliver the FIN.

3. Streams are also terminated when the connection is terminated.

4.3 Connection Termination

Connections should remain open until they become idle for a pre-negotiated period of time. When a server decides to terminate an idle connection, it should not notify the client to avoid waking up the radio on mobile devices. A QUIC connection, once established, can be terminated in one of two ways:

1. **Explicit Shutdown**: An endpoint sends a CONNECTION_CLOSE frame to the peer initiating a connection termination. An endpoint may send a GOAWAY frame to the peer prior to a CONNECTION_CLOSE to indicate that the connection will soon be terminated. A GOAWAY frame when sent signals to the peer that any active streams will continue to be processed, but the sender of the GOAWAY will not initiate any additional streams and will not accept any new incoming streams. On termination of the active streams, a CONNECTION_CLOSE may be sent. If an endpoint sends a CONNECTION_CLOSE frame while unterminated streams are active (no FIN bit or RST_STREAM frames have been sent or received for one or more streams), then the peer must assume that the streams were incomplete and were abnormally terminated.

2. **Implicit Shutdown**: The default idle timeout for a QUIC connection is 30 seconds, and is a required parameter("ICSL") in connection negotiation. The maximum is 10 minutes. If there is no network activity for the duration of the idle timeout, the connection is closed. By default a CONNECTION_CLOSE frame will be sent. A silent close option can be enabled when it is expensive to send an explicit close, such as mobile networks that must wake up the radio.

An endpoint may also send a PUBLIC_RESET packet at any time during the connection to abruptly terminate an active connection. A PUBLIC_RESET is the QUIC equivalent of a TCP RST.

5 Postscript

I picked the most important (I think) parts of the QUIC spec, thus the outline ignores much detail. If you only intend to have a general comprehension of QUIC, I believe the outline would satisfy your requirements. Nevertheless, if you are not content with these and thirst for more detailed demonstration, I highly recommend you refer to the QUIC official document. Besides, since QUIC closely follows HTTP/2 as for architecture, being familiar with HTTP/2 can help you much to understand the QUIC spec.