Ethan Behar
10/7/2021
Professor Swany
P538 Computer Networks

QUIC Report

QUIC is a transport protocol designed from the ground up to improve performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms. It is a user-space transport. That is, it mainly functions in the Application Layer. And uses UDP as a substrate. We'll start with the design motivations behind QUIC and then dive deeper into the protocol and features.

There are four major motivations behind the creation of QUIC. They are protocol entrenchment, implementation entrenchment, handshake delay, and head-of-line blocking delay. Protocol entrenchment refers to the phenomenon that while new transports have been developed many do not see widespread deployment. This is an unintended effect of middleboxes and the Internet's structure overall. Without adding explicit support for these new transports, they are often blocked by firewalls or overwritten by NATs. QUIC address this issue by working in the Application Layer and using UDP as the Transport Layer protocol. Implementation entrenchment is a problem with easily TCP. TCP is commonly implemented in the OS kernel. As a result, pushing changes to TCP typically requires OS upgrades. This is true for clients and servers alike. This situation limits the deployment velocity of TCP changes. Simply put OS upgrades are not rapidly deployable due to system-wide impacts, cautious upgrade pipelines and mechanisms, and rigorous stability and performance testing. This all leads to the fact that OS upgrades can take many months or even a year to deploy confidently. Handshake delay refers to costs of layer TLS on top of TCP. TCP's 3-way handshake takes 2 RTTs to complete and TLS adds an addition RTT for a total of 3 RTTs. Head-of-line blocking is when a packet is stuck in the input queue because a packet in front is stuck waiting for its output port to open. Even if the packet, who is stuck behind another packet, has their output port open they cannot move there. With these four problems in mind QUIC looks to eliminate them or alleviate them as much as possible.

The QUIC protocol has three different handshake scenarios. The first scenario is when a client is establishing a new connection to the server. An initial Inchoate CHLO request is sent to the server and the server replies with a REJ response. This REJ response has the necessary information to establish connection and encryption. A client will cache this information for future use. Then the client can send a Complete CHLO request as well as encrypted requests without waiting for an SHLO response from the server. This enables QUIC to allow applications to send data without waiting for a full RTT for the SHLO response. The second scenario is when a client wants to re-establish a connection with a server. The client retrieves the cached information from the original REJ response and sends a Complete CHLO request as well as immediately sending encrypted requests. The server will respond with the SHLO response and encrypted responses for the encrypted requests. The third scenario is when the second scenario happens but now the cached information is expired or changed. The server will reply to the Complete CHLO request with the REJ response. The client can then update its cached information and proceed to send a new Complete CHLO and encrypted requests. The server will respond with the SHLO response and encrypted responses for the encrypted requests. Since a client can send requests without waiting for the SHLO response this allows clients to send data sooner and eliminates the RTT waiting that typical TCP/TLS protocols experience.

During connection establishment version negotiation is performed. The client sends a version to use in the first packet. The server will either support the proposed version or reply with a list of versions it does support in a Version Negotiation packet. This does cause a RTT delay before connection establishment if the client's proposed version is not supported.

Ethan Behar
10/7/2021
Professor Swany
P538 Computer Networks

QUIC streams are a lightweight abstraction that provide a reliable bidirectional bytestream. A QUIC packet is composed of a common header followed by one or more frames. Multiplexing is achieved by encapsulating stream data in one or more stream frames. A single QUIC packet can carry stream frames from multiple streams. Streams are identified by stream IDs which aid in the demultiplexing process. QUIC supports multiple streams within a connection. This ensures that a lost UDP packet only impact streams whose data was carried in the lost packet. On the receiver side any subsequent packets that arrive pertaining data for other streams can continue to be reassembled and sent to the application without issue.

QUIC packets are fully authenticated and mostly encrypted. With the exception of the few early handshake and reset packets. The parts that are not encrypted is the following header information: Flags, Connection ID, Version Number, Diversification Nonce, and Packet Number. All of these parts are required for routing or for decrypting the packet so as it stands encrypting would not be wise. It's worth mentioning that the list may change during QUIC's IETF standardization.

Unlike TCP packets and sequence numbers, QUIC packets always carry a new packet number, including those carrying retransmission data. Since each packet has a new packet number this design obviates the need for a mechanism to distinguish the ACK of a retransmission from the original transmission. Thus, the ambiguity problem of retransmission in TCP is eliminated. Stream offsets in stream frames are used for determining order of streams. While the packet number represents an explicit time-ordering. This means packet numbers are not used for delivery ordering and enable a better loss detection than TCP offers.

QUIC has two mechanisms for flow control: connection-level flow control and stream-level flow control. Connection-level flow control works by limiting the aggregate buffer that a sender can consume at the receiver across all streams. Stream-level flow control works by limiting the buffer that a sender can consume on any given stream. Similarly, to TCP, these mechanisms use a credit-based flow control. To put in short, the receiver adversities the availability of buffer space by periodically sending window update frames. This works the same way for stream-level and connection-level flow control with the difference being how much the advertised space a stream has versus the aggregation across all streams for a connection.

QUIC's congestion control is plug and play so-to-speak. It has a pluggable interface that can be configured. This allows different algorithms to be used and experimented with. In the deployment of QUIC they used Cubic congestion control.

To avoid connection lose or reestablishing connections due to NAT rebinding or IP changes QUIC connections are identified by a 64-bit Connection ID. This moves away from the reliance of IP and Port to identify an established connection. Thus, QUIC connections are more resilient to NAT timeout and rebinding or changes to the clients IP address. It is worth mention this is still a problem for the client-initiated connection. It is still susceptible to these problems because a connection ID is not created at this point.

Since clients do not know whether a server speaks QUIC or not a QUIC server will advertise this in an "Alt-Svc" header in the HTTP response. This header will inform a client that connections to the origin may use QUIC. The client will then send a QUIC and TCP/TLS connection but delays the TCP/TLS connection by 300ms to favor the use of QUIC. Whichever protocol successfully establishes a connection first with the origin is used moving forward. For whatever reason if the QUIC connection fails the client will fall back to TLS/TCP.

Ethan Behar
10/7/2021
Professor Swany
P538 Computer Networks

This was a short summary of QUIC with the intention of summarizing the QUIC protocol, its features, and its design motivations. For a full fledge introduction to QUIC one should seek and read the QUIC paper: The QUIC Transport Protocol: Design and Internet-Scale Deployment, SIGCOMM 2017.