[Paper] The QUIC Transport Protocol- Design and Internet-Scale Deployment

Nov. 12, 2019

As the author said the QUIC protocol is **"an encrypted, multiplexed, and low-latency transport protocol designed from the ground up to improve transport performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms."**

QUIC has been globally deployed at Google on thousands of servers and is used to serve traffic to a range of clients including a widely-used web browser (Chrome) and a popular mobile video streaming app (YouTube).

QUIC is an encrypted transport: packets are authenticated and encrypted, preventing modification and limiting ossification of the protocol by middleboxes.

Why QUIC (Quick Udp Internet Connection)

Protocol Entrenchment

As a result, even modifying TCP remains challenging due to its ossification by middleboxes [29, 49, 54]. Deploying changes to TCP has reached a point of diminishing returns, where simple protocol changes are now expected to take upwards of a decade to see significant deployment (see Section 8).

Implementation Entrenchment

TCP is commonly implemented in the Operating System (OS) kernel. As a result, even if TCP modifications were deployable, pushing changes to TCP stacks typically requires OS upgrades. This coupling of the transport implementation to the OS limits deployment velocity of TCP changes; OS upgrades have system-wide impact and the upgrade pipelines and mechanisms are appropriately cautious [28].

Handshake Delay

TCP connections commonly incur at least one round-trip delay of connection setup time before any application data can be sent, and TLS adds two round trips to this delay.

Head-of-line Blocking Delay

TCP’s bytestream abstraction, however, prevents applications from controlling the framing of their communications [12] and imposes a "latency tax" on application frames whose delivery must wait for retransmissions of previously lost TCP segments.

QUIC's Goal

QUIC is designed to meet several goals [59], including deployability, security, and reduction in handshake and head-of-line blocking delays.

The QUIC protocol combines its cryptographic and trans- port handshakes to minimize setup RTTs. It multiplexes multiple requests/responses over a single connection by providing each with its own stream, so that no response can be blocked by another.

Design and Implementation

Connection Establishment

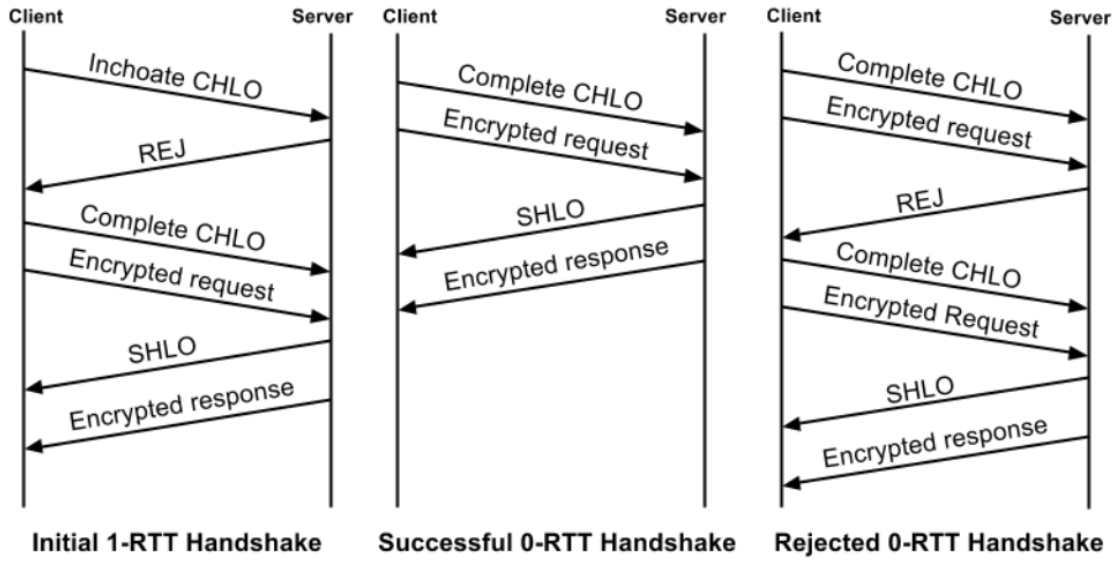


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

- inchoate client hello (CHLO)
- reject (REJ)
- server hello (SHLO)

An origin is identified by the set of URI scheme, hostname, and port number [5].

Cryptography

Diffie-Hellman

Suppose

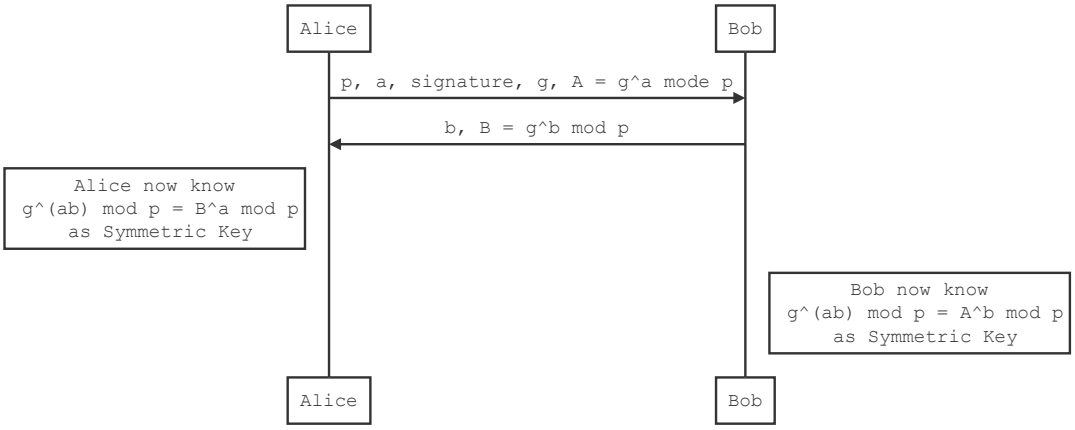
Client (Bob)

- $Private\ Key\ b$

Server (Alice)

- $A\ long\ term\ public\ value,\ large\ prime\ p$
- $Private\ Key\ a$
- $Signature\ by\ trusted\ third\ party\ if\ needed$
- $Public\ prime\ base\ g$

Goal: Symmetric Key g^(ab) mod p



$(g^a mod\ p)^b = (g^b mod\ p)^a = g^{ab} mod\ p$

Prove of Diffie-Hellman

Let:

$Prime\ p,\ g$

$Private\ Key\ a,\ b$

Proof:

$(g^a mod\ p)^b = (g^b mod\ p)^a = g^{ab} mod\ p$

Solution:

$\exists\ integer\ K\ let\ (g^a mod\ p)^b = (g^a + Kp)^b mod\ p$

$(g^a + Kp)^b mod\ p = [g^{ab} + C_b^1 g^{a-1} pK + C_b^2 g^{a-2} (pK)^2 + \ldots + (pK)^a]\ mod\ p$

$= g^{ab} mod\ p$

REJ Message from Server

- server's long-term Diffie-Hellman public value
- a certificate chain authenticating the server
- a signature of the server config using the private key from the leaf certificate of the chain
- a source-address token: an authenticated-encryption block that contains the client's publicly visible IP address (as seen at the server) and a timestamp by the server.

Version Negotiation

A QUIC client proposes a version to use for the connection in the first packet of the connection and encodes the rest of the handshake using the proposed version. If the server does not speak the client-chosen version, it forces version negotiation by sending back a Version Negotiation packet to the client carrying all of the server's supported versions, causing a round trip of delay before connection establishment.

Loss Recovery

Each QUIC packet carries a new packet number, including those carrying retransmitted data.

This design obviates the need for a separate mechanism to distinguish the ACK of a retransmission from that of an original transmission, thus avoiding TCP's retransmission ambiguity problem. Stream offsets in stream frames are used for delivery ordering, separating the two functions that TCP conflates. The packet number represents an explicit time-ordering, which enables simpler and more accurate loss detection than in TCP.

Relations between Stream ID, Offset and Packet Number.

For each stream contains a **Stream ID** and frames. If a packet including frames lost, the sender will retransmit this packet with a increased **Packet ID** but let frame's **Stream ID and Offset** unchanged. **(Increased packet id and still stream offset when retransmitting contributed to accurate RTT estimation)**

Flow Control

Similar to HTTP/2 [8], QUIC employs credit-based flow-control. A QUIC receiver advertises the absolute byte offset within each stream up to which the receiver is willing to receive data. As data is sent, received, and delivered on a particular stream, the receiver periodically sends window update frames that increase the advertised offset limit for that stream, allowing the peer to send more data on that stream. Connection-level flow control works in the same way as stream-level flow control, but the bytes delivered and the highest received offset are aggregated across all streams.

Congestion Control

The QUIC protocol does not rely on a specific congestion control algorithm and our implementation has a pluggable interface to allow experimentation.

NAT Rebinding and Connection Migration

While QUIC endpoints simply elide the problem of NAT rebinding by using the Connection ID to identify connections, client-initiated connection migration is a work in progress with limited deployment at this point.##

QUIC Discovery for HTTPS

On a subsequent HTTP request to the same origin, the client races a QUIC and a TLS/TCP connection, but prefers the QUIC connection by delaying connecting via TLS/TCP by up to 300 ms. Whichever protocol successfully establishes a connection first ends up getting used for that request. If QUIC is blocked on the path, or if the QUIC handshake packet is larger than the path's MTU, then the QUIC handshake fails, and the client uses the fallback TLS/TCP connection.

References

https://zhuanlan.zhihu.com/p/32553477

https://math.stackexchange.com/questions/61358/prove-equivalence-of-diffie-hellman-shared-secret

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange