A Quick QUIC Introduction

Jinwei (Clark) Zhao

https://pan.uvic.ca/~clarkzjw/

2023/03/31

In CSC361

- TCP, reliable
  - Congestion control (Cubic, Reno, NewReno…)
  - Error control
- UDP, unreliable
- HTTP/1.1 -> HTTP/2
- Transport Layer Security (TLS), HTTPS

In CSC361

- TCP, reliable
  - Congestion control (Cubic, Reno, NewReno…)
  - Error control
- UDP, unreliable
- HTTP/1.1 -> HTTP/2
- Transport Layer Security (TLS), HTTPS
- What's next?

What’s wrong with TCP?

While TCP has been the dominant transport layer protocol for many years (since 1970s!), it has some limitations that can hinder performance, particularly in modern use cases.

What's wrong with TCP?

While TCP has been the dominant transport layer protocol for many years (since 1970s!), it has some limitations that can hinder performance, particularly in modern use cases.

- Connection establishment latency
  - TCP requires a 3-way handshake to establish a connection. If secure communication is required, a separate TLS handshake adds additional latency.
  - There is TCP Fast Open, but it's never been widely adopted.

What's wrong with TCP?

While TCP has been the dominant transport layer protocol for many years (since 1970s!), it has some limitations that can hinder performance, particularly in modern use cases.

- Connection establishment latency
  - TCP requires a 3-way handshake to establish a connection. If secure communication is required, a separate TLS handshake adds additional latency.
  - There is TCP Fast Open, but it's never been widely adopted.
- Lack of built-in encryption
  - TCP does not have built-in encryption, which means that an additional layer, such as SSL/TLS, is required to secure communications.
  - Encryption is optional in TCP.

What's wrong with TCP?

- Head-of-Line blocking
  - TCP delivers data in order, which means that a lost or delayed packet can block the delivery of all subsequent packets, waiting for lost packets to be retransmitted.

What's wrong with TCP?

- Head-of-Line blocking
  - TCP delivers data in order, which means that a lost or delayed packet can block the delivery of all subsequent packets, waiting for lost packets to be retransmitted.
- Inefficient loss recovery
  - TCP's retransmission mechanism is tied to in-order delivery, which can result in unnecessary delays when recovering from packet loss.

What's wrong with TCP?

- Head-of-Line blocking
  - TCP delivers data in order, which means that a lost or delayed packet can block the delivery of all subsequent packets, waiting for lost packets to be retransmitted.
- Inefficient loss recovery
  - TCP's retransmission mechanism is tied to in-order delivery, which can result in unnecessary delays when recovering from packet loss.
- Poor support for mobility and connection migration
  - TCP connections are bound to specific IP addresses, which means that if a user changes networks (e.g., switching from Wi-Fi to cellular), the connection must be re-established.

What's wrong with TCP?

- Head-of-Line blocking
  - TCP delivers data in order, which means that a lost or delayed packet can block the delivery of all subsequent packets, waiting for lost packets to be retransmitted.
- Inefficient loss recovery
  - TCP's retransmission mechanism is tied to in-order delivery, which can result in unnecessary delays when recovering from packet loss.
- Poor support for mobility and connection migration
  - TCP connections are bound to specific IP addresses, which means that if a user changes networks (e.g., switching from Wi-Fi to cellular), the connection must be re-established.
- These limitations create a need for a more efficient and optimized protocol.

What's wrong with TCP?

Why not continue with the TCP evolvements?

- TCP is deeply integrated into operating system kernels and network infrastructures. Middleboxes (Firewalls, routers, NATs, etc.) on the Internet expect existing TCP behaviours.
- New TCP options or new transport layer protocols designed from scratch would be hard to roll out.
- Previous failed attempts: T/TCP, SCTP, RTP, TCP Fast Open…

Now introducing QUIC

- QUIC (pronounced as “quick”) was originally designed and deployed by Google in 2012, later standardized by IETF as RFC 9000 in May 2021, and adopted by HTTP/3 in RFC 9114 in June 2022.

- *QUIC* was initially proposed as the acronym for “**Q**uick **U**DP **I**nternet **C**onnections”, IETF’s use of the word QUIC (RFC 9000) is not an acronym anymore. It is simply the name of the protocol.

- QUIC is based on UDP and it’s a new general-purpose transport layer network protocol.

Now introducing QUIC

- QUIC (pronounced as “quick”) was originally designed and deployed by Google in 2012, later standardized by IETF as RFC 9000 in May 2021, and adopted by HTTP/3 in RFC 9114 in June 2022.
- *QUIC* was initially proposed as the acronym for “**Q**uick **U**DP **I**nternet **C**onnections”, IETF’s use of the word QUIC (RFC 9000) is not an acronym anymore. It is simply the name of the protocol.
- QUIC is based on UDP and it’s a new general-purpose transport layer network protocol.
- Why QUIC succeeded?

Now introducing QUIC

- QUIC (pronounced as “quick”) was originally designed and deployed by Google in 2012, later standardized by IETF as RFC 9000 in May 2021, and adopted by HTTP/3 in RFC 9114 in June 2022.
- *QUIC* was initially proposed as the acronym for “**Q**uick **U**DP **I**nternet **C**onnections”, IETF’s use of the word QUIC (RFC 9000) is not an acronym anymore. It is simply the name of the protocol.
- QUIC is based on UDP and it’s a new general-purpose transport layer network protocol.
- Why QUIC succeeded?
- Core features:
  - Built-in and mandatory encryption with TLS v1.3
  - Fast connection establishment with 1-RTT or 0-RTT
  - Stream multiplexing fixes Head-of-Line blocking
  - Connection ID fixes mobility and connection migration issue

Why UDP?

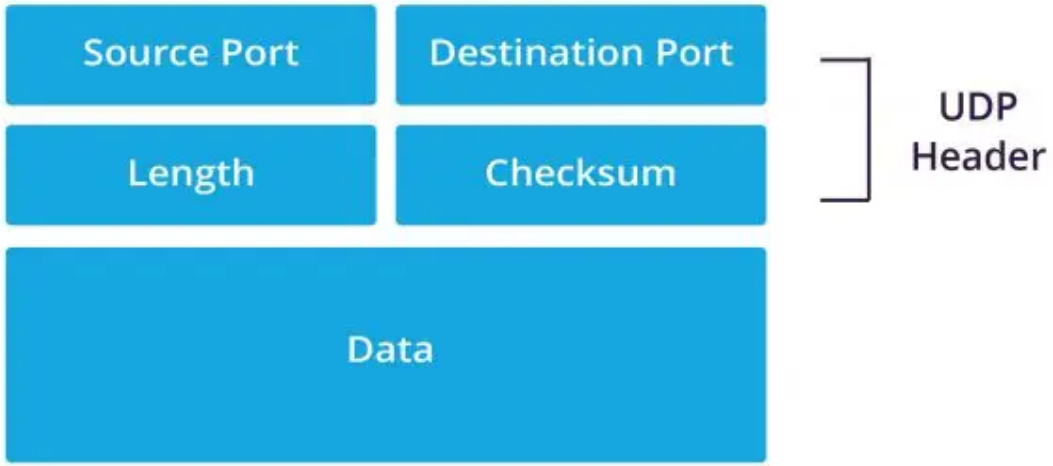

Figure 1: UDP packet structure

QUIC packets are encapsulated in UDP datagrams. A single UDP datagram can carry multiple QUIC packets.

Why UDP?

- QUIC is implemented in the user space, instead of in the kernel as with TCP and UDP.
- Flexibility and extensibility: Building QUIC on top of UDP allows it to be more easily extended and improved without the need for modifying the underlying protocol stack.
- Reduced latency: UDP is a connectionless protocol, meaning it does not require the establishment of a connection between sender and receiver before data transmission can occur.
- NAT traversal: QUIC can more easily traverse Network Address Translation (NAT) devices and firewalls, as it operates over UDP, which generally encounters fewer issues with middleboxes compared to TCP.

QUIC Packet

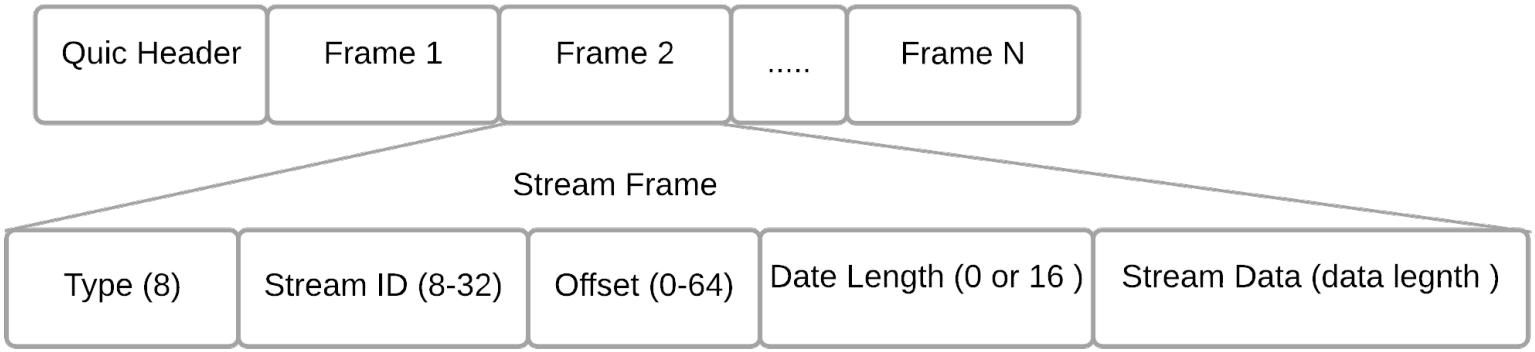

Figure 2: QUIC packet structure

There are multiple header and packet types in QUIC, and a QUIC packet consists of a QUIC header and different frames.

QUIC Packet

QUIC has two different packet formats. Common **long header** packets include: **Version Negotiation Packet**, **Handshake Packet**, **Retry Packet**, etc. **Short header** format is used for transmitting application data after the connection is established.

```
Long Header Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2),
  Type-Specific Bits (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Type-Specific Payload (..),
}
```

```
Short Header Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),
  Key Phase (1),
  Packet Number Length (2),
  Destination Connection ID (0..160),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Figure 3: QUIC Packet Format

Connection ID

- Each QUIC connection has a set of connection IDs.
- QUIC uses connection IDs to identify different connections.
- QUIC supports native connection migration. So QUIC doesn't have to reset the connection like TCP if the device switches to a new network or the IP addresses or port numbers change for any other reason.
- With the help of connection migration, QUIC doesn't have to redo the handshake under the new conditions and HTTP/3 doesn't have to re-request the files that were being downloaded when the network migration happened — which can be a problem in the case of larger files or video streaming.

QUIC Frames

Packet payloads are structured into QUIC frames. Frames are responsible for carrying different types of information, each with a specific purpose, enabling various protocol features and functionalities.

Common QUIC frame types:

- ACK - to acknowledge that packets have been received and processed
- STREAM - carries application data
- NEW_CONNECTION_ID
- CONNECTION_CLOSE
- HANDSHAKE_DONE
- ...
- `https://datatracker.ietf.org/doc/html/rfc9000#table-3`

1-RTT Connection Establishment

QUIC only takes one single round-trip time between client and server to complete the handshake.

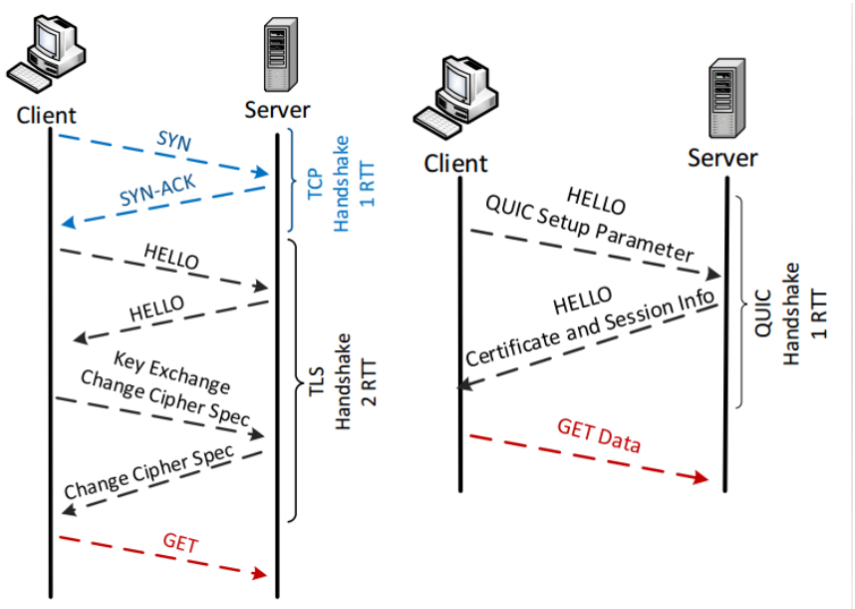

Figure 4: Handshake of QUIC compared to TCP with TLS v1.2

As QUIC combines TCP and TLS handshakes into one, HTTP/3 only needs one round-trip to establish a secure connection between the client and server which can create a faster connection setup and lower latency.

0-RTT Connection Resumption

- The 0-RTT feature in QUIC allows a client to send application data before the handshake is complete.
- This is made possible by reusing negotiated parameters from previous connections.
- 0-RTT depends on the client remembering critical parameters and providing the server with a TLS session ticket that allows the server to recover the same information.

QUIC Stream

Streams in QUIC provide a lightweight and ordered byte-stream abstraction to an application.

- Ordered within a single stream: QUIC ensures that data is reliably delivered in the order it was sent.
- Parallel and independent across different streams: Fixes Head-of-Line blocking.
- Bidirectional: QUIC streams support bidirectional communication, allowing both endpoints to send and receive data on the same stream.

Stream Multiplexing

QUIC can operate each stream independently according to their stream ID. Lost packets will only impact an individual resource, e.g., specific CSS or JS files in web browsing, which can avoid Head-of-Line blocking in TCP connection.

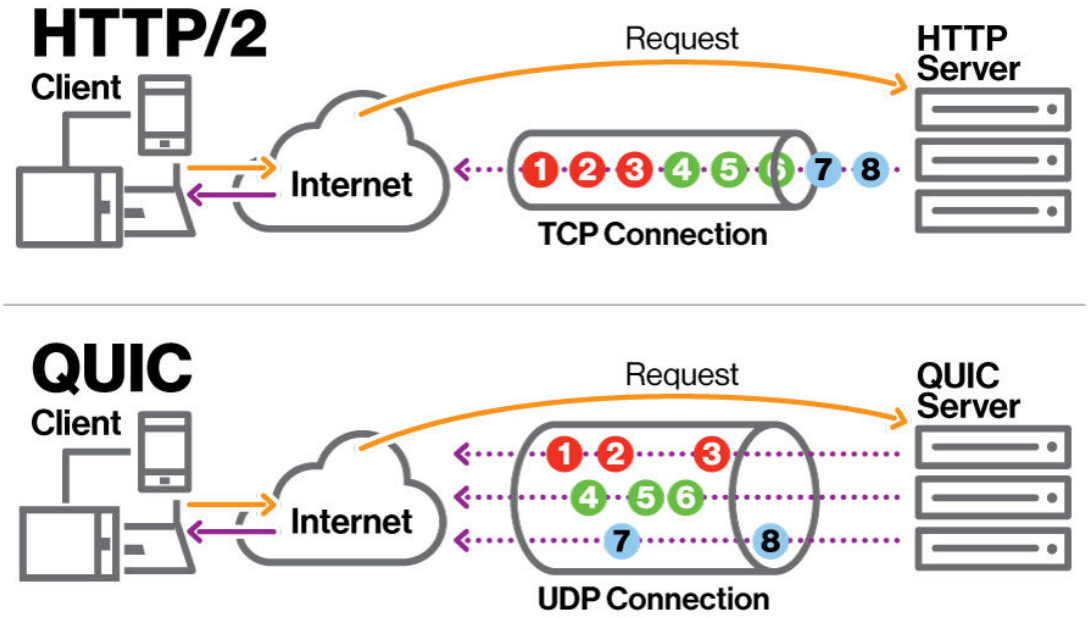

Figure 5: QUIC Streaming Multiplexing, source: Orozco 2018

Plugable Congestion Control

QUIC allows implementations to choose from different congestion control algorithms, as these are not specific to the transport protocol.

The most well-known algorithms are:

- NewReno - The congestion control algorithm used by TCP, defined in RFC 6582, and used as an explanation of QUIC's congestion control mechanism in RFC 9002
- Cubic - Defined in RFC 8312, similar to NewReno, but uses a cubic function instead of a linear one to calculate the congestion window increase rate.
- BBR (**B**ottleneck **B**andwidth and **R**TT congestion control) - Doesn't have an RFC yet; it's currently developed by Google, `https://github.com/google/bbr`.

Challenges of QUIC v1

- Since it's designed to be implemented in the user space, performance gains over TCP vary between different implementations. QUIC specifications only provide general implementation guidelines. QUIC specifications give a lot of freedom to implementation developers.
  - Existing QUIC implementations:
    `https://github.com/quicwg/base-drafts/wiki/Implementations`
  - Interoperability (Interop) tests between different QUIC implementations:
    `https://interop.seemann.io/`

Challenges of QUIC v1

- Since it's designed to be implemented in the user space, performance gains over TCP vary between different implementations. QUIC specifications only provide general implementation guidelines. QUIC specifications give a lot of freedom to implementation developers.
  - Existing QUIC implementations: `https://github.com/quicwg/base-drafts/wiki/Implementations`
  - Interoperability (Interop) tests between different QUIC implementations: `https://interop.seemann.io/`
- Some networks block UDP, except for essential UDP traffic such as DNS.
  - If UDP is blocked on a network, in HTTP/3, the traffic falls back to TCP-based HTTP/2 connections.
  - However, as RFC 9308, which discusses the applicability of QUIC, warns, "any fallback mechanism is likely to impose a degradation of performance and can degrade security".

Extentions, QUIC v2 and more

- qlog: logging format for QUIC and HTTP/3
- Multipath QUIC: enable the simultaneous usage of multiple network paths for a single connection
- QUIC v2
- New application scenarios, HTTP/3, DNS-over-QUIC…
- Know more at: https://quicwg.org/

QUIC@PanLab

- Multipath QUIC implementation in NS-3
- Adaptive video streaming with Multipath QUIC (My current MSc research topic)
  - Utilize multipath network paths for video streaming scenarios to improve user experience
  - The current focus is to develop efficient multipath scheduling and video bitrate selection algorithm