SCTP
Stream Control Transmission Protocol

Goals:
- Proposed in RFC 2960 (October 2000)
- Extended in RFC 4960 (September 2007)
- "Reliable transport protocol operating on top of a connectionless packet network such as IP"
- Combines advantages of TCP and UDP
- Multiple streams
- Supports of multi-homing

Sounds good, so why don’t we use it everywhere?
- TCP was already established as the default transport layer protocol (network ossification)
- Poor support in operating systems and applications
- Many middleboxes (e.g. firewalls, NAT) do not work with SCTP → packets are discarded
Chapter 7: Transport Layer Protocols

TCP

UDP

SCTP

QUIC

QUIC Features

In Practice

Standardization

Bibliography
What is QUIC?
- QUIC = Quick UDP Internet Connections
- Developed around 2012 by Google, deployed in Google Chrome and Chromium [12, 11]
- A substitute for the TCP/TLS protocol stack, based on UDP
- Since 2016 standardization ongoing by IETF

Motivation and Goals
- Decrease handshake delay
- Get rid of head-of-line blocking
- Faster development cycles
- Middlebox resistance
- IP mobility
QUIC Features

Connection ID
- Used instead of the 5-tuple as identifier
- This allows to change IP and port

Stream Multiplexing
- Multiple streams within a connection
- Each stream provides a reliable bidirectional bytestream
- QUIC packet contains several frames
- QUIC packet can carry stream frames from multiple streams

Different Frame Types
- Control frames
- Data and acknowledgement frames

Flow Control
- Stream flow control
- Connection flow control

Congestion Control
- Currently Cubic
- BBR implementation in progress

Different Packet Types
- Version Negotiation Packet
- Initial Packet
- Retry Packet
- Handshake Packet

Encryption and Authentication
- Packets are always protected using TLS 1.3

Loss Detection and Re-ordering
- Retransmissions have different packet numbers → use Stream Offset for in order delivery
- More elaborated acknowledgement mechanism including selective and negative ACKs (SACKs and NACKs)

Forward Error Correction
- Add redundancy to reconstruct lost packets
### QUIC Features

#### Header

**Long Header**

- **0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**
  - **1 1** Type Type-Specific
  - **Version**
  - **DCID Len** Destination Connection ID
  - **SCID Len** Source Connection ID
  - **Type Specific Fields**
  - **Payload**

**Short Header**

- **0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**
  - **0 1 S** Reserved K P
  - **Destination Connection ID (0..160 bit)**
  - **Packet Number (8/16/24/32 bit)**
  - **Protected Payload**

---

Disclaimer: Header formats are from the current QUIC draft, but Google uses a similar version at the moment.
QUIC Features
Decrease Handshake Delay

Problem:
- TCP does a 3-way handshake
- TLS does at least 3-way handshake (or more...)
- Results in a lot of RTTs before data transmission
- Can in part be decreased using TCP Fast Open (but not widely deployed)

Solution:
- Introduce a 0-RTT and a 1-RTT handshake
- Merge the TCP and TLS component into one protocol
- Reuse old connections
- Client saves information about the server
QUIC Features
Handshakes

(a) Initial 1-RTT Handshake

(b) Successful 0-RTT Handshake

(c) Rejected 0-RTT Handshake

Figure 6: (Source: [11])

Notes

- REJ contains (Diffie-Hellman value, certificate chain, signature, token, ...)
- Complete CHLO contains client's Diffie-Hellman public value
**QUIC Features**

**Handshakes in practice**

Figure 7: (Source: [11])

**Average Handshake Latency**

- $\text{TCP}_g$: TLS/TCP control group
- $\text{QUIC}_g$: QUIC experimental group including 0-RTT
- $\text{QUIC}_g(1\text{-RTT}+)$: QUIC experimental group not achieving 0-RTT handshake
QUIC Features
Get Rid of Head-of-Line Blocking

Problem:

- If one TCP segment is lost in transit, everything after that has to wait for delivery until successful retransmission (in-order property)
- Frequent goal: multiplexing multiple data streams over one TCP connection
- Example: Two videos get transmitted over one TCP connection
  - Server sends videos interleaved
  - One packet containing a part of video #1 gets lost
  - Following parts of video #2 cannot be processed, although they may already be present
  - Result: Video #2 has unnecessary quality impairments

Solution:

- Protocol is aware of multiple streams
- Retransmission is done at stream-level, not connection-level

Figure 8: Adopted from QUIC: Next generation multiplexed transport over UDP
Problem:

- TCP is implemented in the kernel
- \(\Rightarrow\) slow deployment of new mechanisms
  - Devices often don’t get updated to newer kernel
  - Getting modifications of kernel protocol mechanisms is a slow process
  - Involves a lot of testing with a lot of different applications
  - Running big-scale experiments with TCP is very difficult

Solution:

- QUIC is based on UDP, and implemented in user space
- The kernel is not involved in the protocol itself
- Experiments with new protocol mechanisms are straightforward, as long as user-space is controlled by the application vendor

Note: In Dec 2015 Google disabled QUIC due to a vulnerability in the client code
Why use UDP? Why not implement a new layer 4 protocol?

Problem:

- Middleboxes such as firewalls, “optimizers”, etc. exist
- In many cases, they make things worse
- May lead to obscure behaviour
- Get produced by a variety of different vendors/manufacturers
- Getting along with middleboxes is like herding cats

Solution by QUIC:

- Encrypt data stream transported by UDP
- ⇒ protocol headers above are not accessible to middleboxes
- TCP-like “optimizers” as not possible due to encryption
Problem:

- TCP connections are identified by the 5-tuple
- Client IP address may change during the connection
- DSL connection gets re-established after 24h
- Mobile clients move from one network to another

Solution:

- Do not use the 5-tuple as connection identifier
- QUIC identifies connections by a Connection ID
- Last client IP address to send a valid packet for a given Connection ID is the current IP address of the client
Figure 10: Comparison of QUIC and TCP for various metrics, versus minimum RTT. The graph shows that mean search latency (normalized) increases with decreasing minimum RTT. The orange line represents TCP/TLS, while the blue line represents QUIC. The graph indicates that QUIC experiences a lower mean search latency compared to TCP/TLS across different minimum RTT values.

(Source: [11])
Standardization

IETF

- QUIC standardization since July 2016 by the Internet Engineering Task Force (IETF)
- Current draft version November 4th, 2019
- https://datatracker.ietf.org/wg/quic/documents/
- 5 key goals:
  - Minimizing connection establishment and overall transport latency for applications, starting with HTTP/2
  - Providing multiplexing without head-of-line blocking
  - Requiring only changes to path endpoints to enable deployment
  - Enabling multipath and forward error correction extensions
  - Providing always-secure transport, using TLS 1.3 by default

Figure 11: Number of pages in the IETF QUIC draft.
Conclusion

QUIC

- Still under development and standardization
- Higher CPU costs as TCP/TLS, but optimization is ongoing
- Deploying networking protocols in user space
  - faster and easier development cycles
  - bypass problems like head-of-line blocking
- "Layering enables modularity but often at the cost of performance" [11]
- Achieve lower latency with 0-RTT handshake
Chapter 7: Transport Layer Protocols

TCP
UDP
SCTP
QUIC

Bibliography
Chapter 7: Transport Layer Protocols


