QUIC Transport Protocol

- Development and basic features
- Connection establishment; data transfer
- Avoiding ossification; limitations

CC BY ND

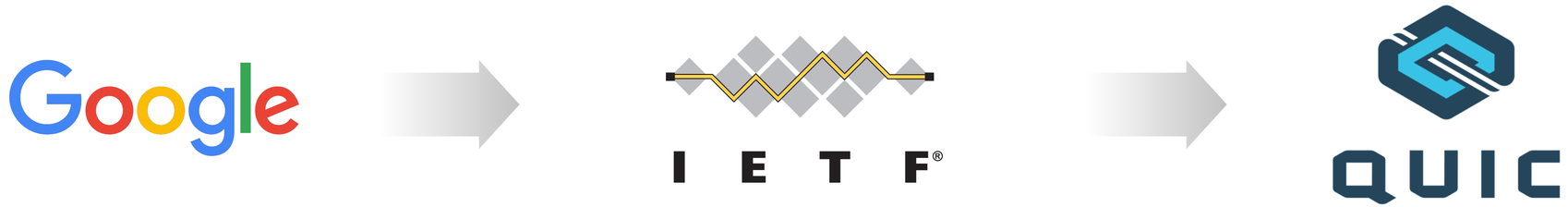
QUIC: Performance, Security, Avoiding Ossification

- What's wrong with TLS v1.3 over TCP?
  - Slow to connect – due to sequential TCP and TLS handshakes
  - Leaks some metadata
  - Ossified and hard to extend

- QUIC aims to replace TLS v1.3 and TCP with a single secure transport protocol
  - Reduce latency by overlapping TLS and transport handshake
  - Avoid metadata leakage via pervasive encryption
  - Avoid ossification via systematic application of GREASE and encryption

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

History of QUIC



- 2012-2015: Initial proposal and experimental deployment by Google
- 2016-2021: IETF QUIC working group
  - https://quicwg.org and https://datatracker.ietf.org/wg/quic/
  - As of January 2021, specifications have gone through 34 iterations in IETF – standardisation led to **significant differences and improvements compared to initial Google proposal**
  - GitHub repo for the specification has ~2200 issues and ~2500 pull requests
- Expect Internet standards track RFCs describing QUIC to be published in early 2021

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Overview (1/2)

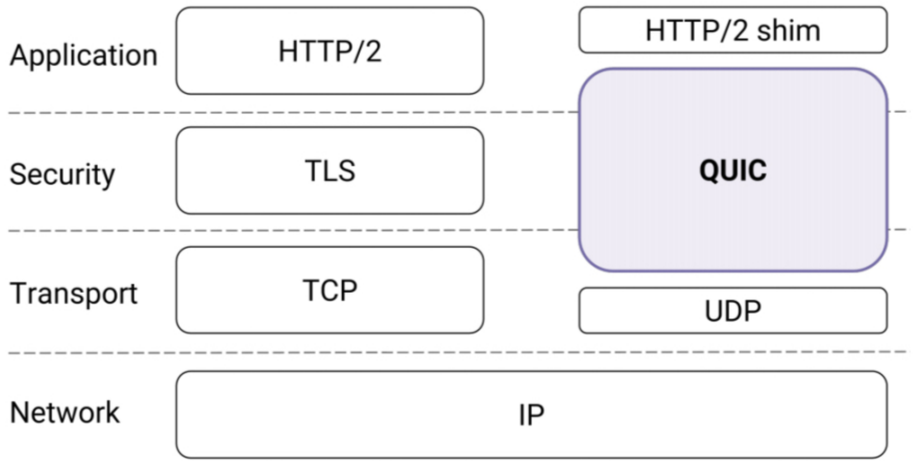


Figure 1: QUIC in the traditional HTTPS stack.

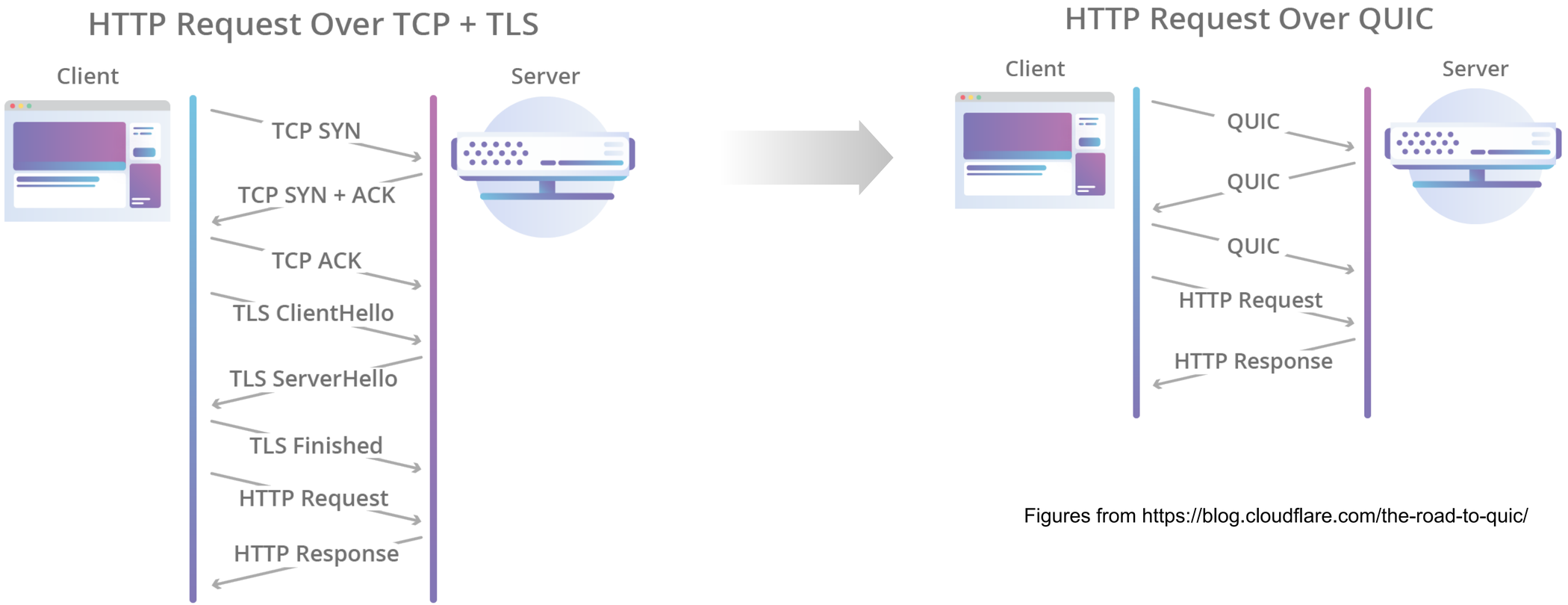
Figure from: A. Langley *et al.*, "The QUIC transport protocol: Design and Internet-scale deployment", Proceedings of the ACM SIGCOMM Conference, Los Angeles, August 2017. https://dl.acm.org/authorize.cfm?key=N33907

- QUIC replaces TCP, TLS, and parts of HTTP
  - HTTP → stream multiplexing
  - TLS → security
  - TCP → reliability, ordering, congestion control
  - Runs on UDP for ease of deployment
- QUIC is a general purpose client-server transport
  - Designed to run HTTP effectively

HTTP/3

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

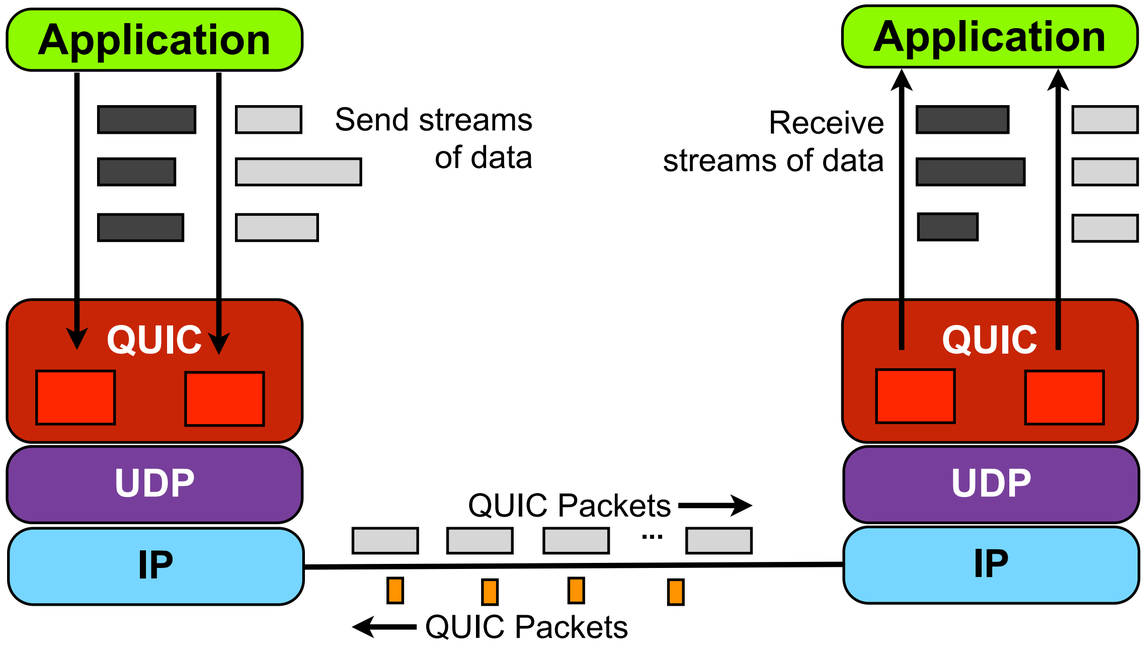
QUIC Overview (2/2)



Figures from https://blog.cloudflare.com/the-road-to-quic/

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Packets, Headers, and Frames

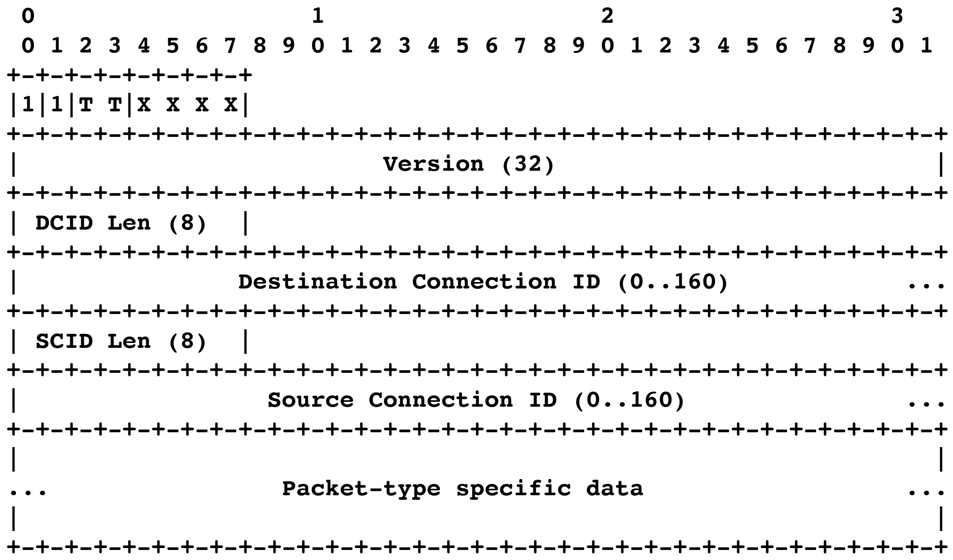


- QUIC sends and receives streams of data within a connection
  - Up to 2^{62} different streams in each direction in a single QUIC connection
- A connection comprises QUIC packets sent within UDP datagrams
- Each QUIC packet starts with a header, and contains one or more frames
  - Frames contain data for streams, or acknowledgements, or other control messages

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Headers (1/2)

- QUIC packets can be long header packets or short header packets
- **Long header packets** are used to establish QUIC connections
  - They all start with a common header, followed by packet-type specific data
  - Four different long-header packet types, denoted by the TT field in the header:
    - **Initial** – initiates connection, starts TLS handshake
    - **0-RTT** – idempotent data sent with initial handshake, when resuming a session
    - **Handshake** – completes connection establishment
    - **Retry** – used to force address validation

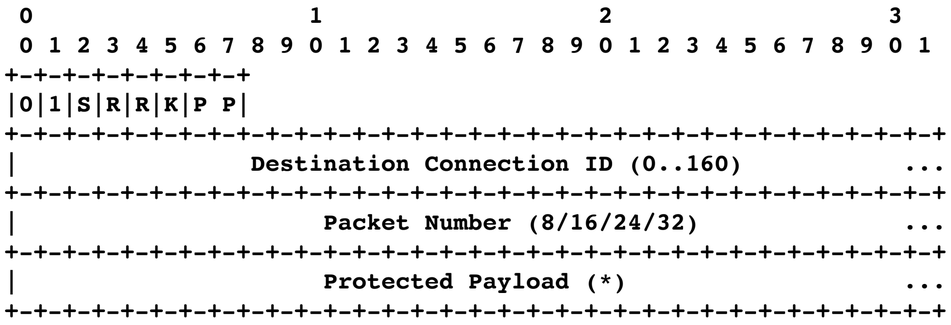


- Several **Initial**, **0-RTT**, and **handshake** packets can be included in one UDP datagram, one after the other, followed by a short header packet

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Headers (2/2)

- QUIC packets can be long header packets or short header packets
- The is one **short header packet** defined in QUIC:
  - **1-RTT** – Used for all packets sent after the TLS handshake is complete
  - The short header is followed by QUIC frames in the enclosing UDP packet
  - Compared to long header, omits information that can be inferred from context



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Frames

- QUIC packet contain an encrypted sequence of **frames**
  - CRYPTO frames are used to carry TLS messages such as the **`ClientHello`**, **`ServerHello`** etc.
  - STREAM and ACK frames send data and acknowledgements
  - Migration between two network interfaces is supported by PATH_CHALLENGE and PATH_RESPONSE frames
  - Other frames control progress of a QUIC connection

- Compared to TCP, QUIC headers are limited in scope
  - Functionality provided by frames instead → more extensible
  - e.g., TCP sends sequence numbers and acknowledgements in the header; QUIC sends this information in STREAM and ACK frames

Type Value	Frame Type Name
0x00	PADDING
0x01	PING
0x02 - 0x03	ACK
0x04	RESET_STREAM
0x05	STOP_SENDING
0x06	CRYPTO
0x07	NEW_TOKEN
0x08 - 0x0f	STREAM
0x10	MAX_DATA
0x11	MAX_STREAM_DATA
0x12 - 0x13	MAX_STREAMS
0x14	DATA_BLOCKED
0x15	STREAM_DATA_BLOCKED
0x16 - 0x17	STREAMS_BLOCKED
0x18	NEW_CONNECTION_ID
0x19	RETIRE_CONNECTION_ID
0x1a	PATH_CHALLENGE
0x1b	PATH_RESPONSE
0x1c - 0x1d	CONNECTION_CLOSE
0x1e	HANDSHAKE_DONE

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Transport Protocol

- Development and basic features
- Connection establishment; data transfer
- Avoiding ossification; limitations

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow