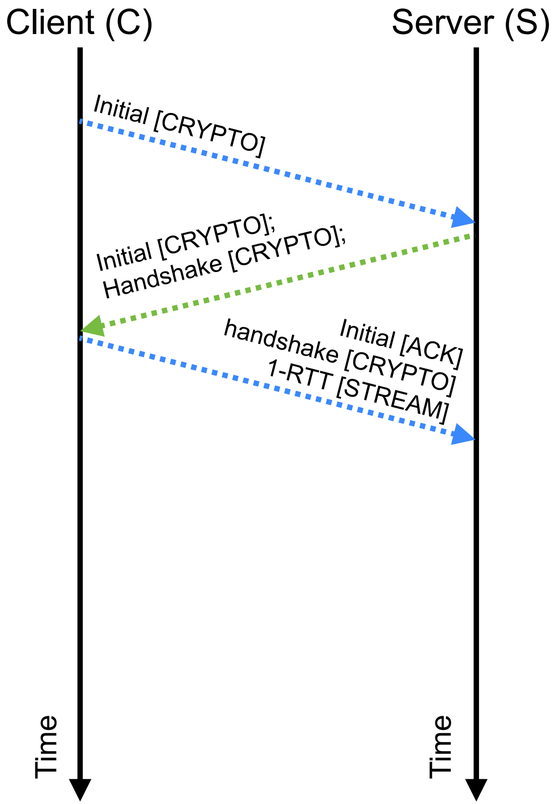
QUIC Transport Protocol

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

QUIC Connection Establishment and Data Transfer

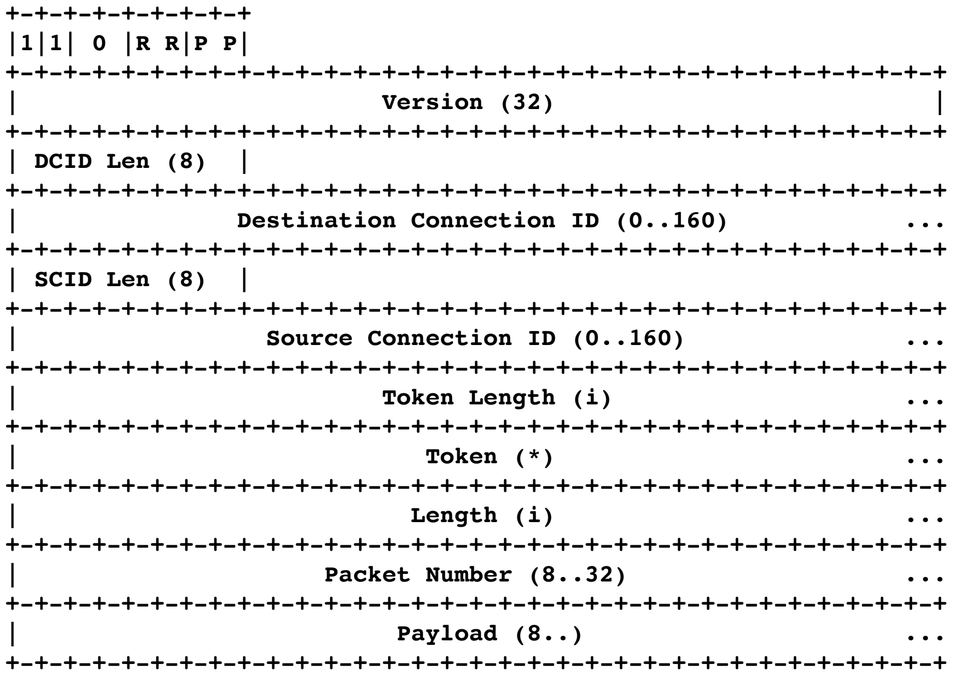
- A QUIC connection proceeds in two phases: handshake and data transfer
  - The **handshake** uses long header packets – establishes the connection, negotiates encryption keys, authenticates the server
  - The **data transfer** phase uses short header packets – sends data and acknowledgements after the connection is established

QUIC Connection Establishment (1/5)



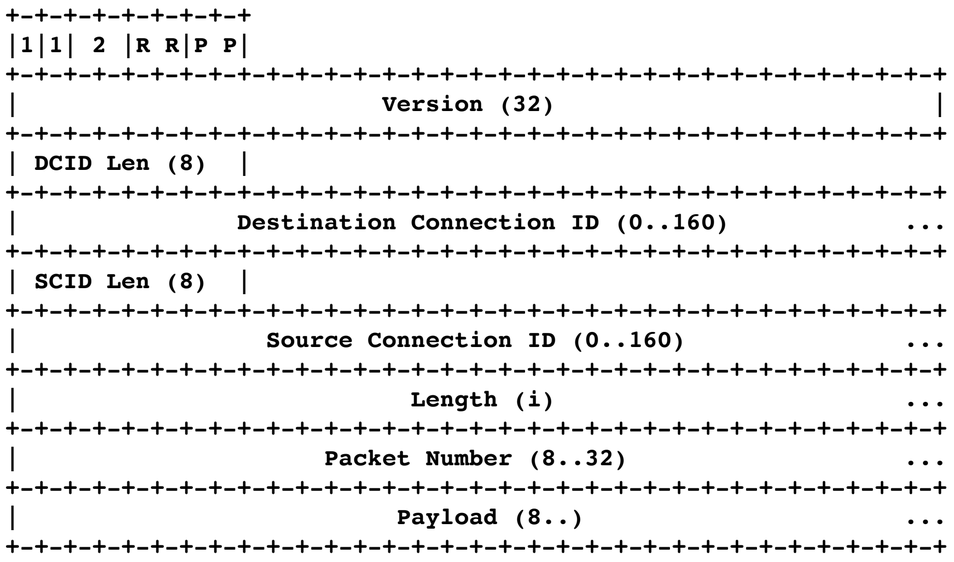
- QUIC combines connection establishment and TLS handshake into one round-trip
  - C → S: QUIC **Initial** packet
    - **Initial** packet contains a CRYPTO frame that contains TLS **`ClientHello`**
  - S → C: QUIC **Initial** and **Handshake** packets
    - Both QUIC packets can be sent in a single UDP datagram
    - **Initial** packet contains CRYPTO frame that contains TLS **`ServerHello`**
    - **Handshake** packet contains other connection setup information
  - C → S: QUIC **Initial**, **Handshake**, and **1-RTT** packets
    - All three QUIC packets can be sent in a single UDP datagram
    - QUIC **Initial** packet contains ACK frame, acknowledging the server's Initial packet
    - QUIC **Handshake** packet contains CRYPTO frame that contains TLS **`Finished`**
    - QUIC **1-RTT** (short header) packet contains STREAM frame with initial data sent from client to server

QUIC Connection Establishment (2/5)



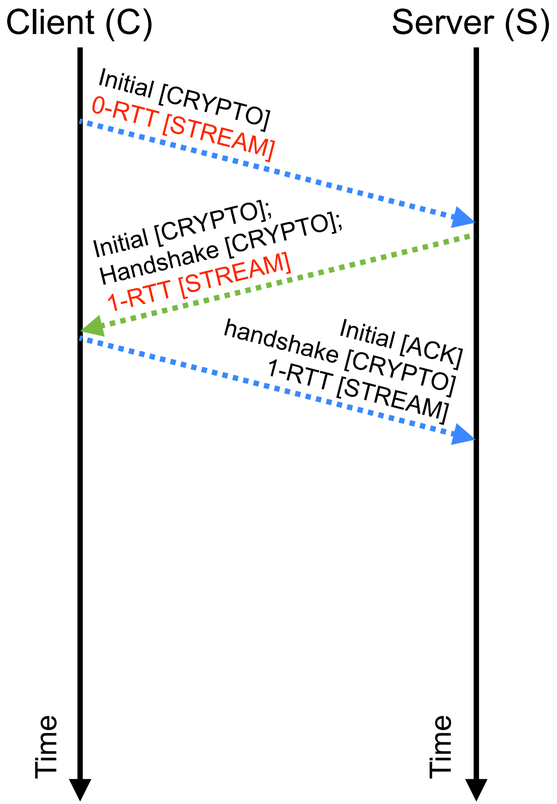
- QUIC **Initial** packets play two main roles:
  - Synchronise client and server state – like TCP's **`SYN`** and **`SYN+ACK`** packets
  - Contains CRYPTO frame, with TLS **`ClientHello`** or **`Finished`**, and may also contain ACK frame
  - Combines connection setup and security negotiation in one packet
- QUIC **Initial** packets also carry optional **`Token`**
  - Server can refuse the initial connection attempt, and send a **Retry** packet containing a **`Token`**.
  - Client must then retry the connection, providing the **`Token`** in its **Initial** packet
  - Can be used to prevent connection spoofing

QUIC Connection Establishment (3/5)



- QUIC **Handshake** packets complete the TLS 1.3 exchange
  - The TLS **`ServerHello`** and **`Finished`** messages
  - TLS can also renegotiate new keys part-way through a connection – this is done in **`Handshake`** packets

QUIC Connection Establishment (4/5)

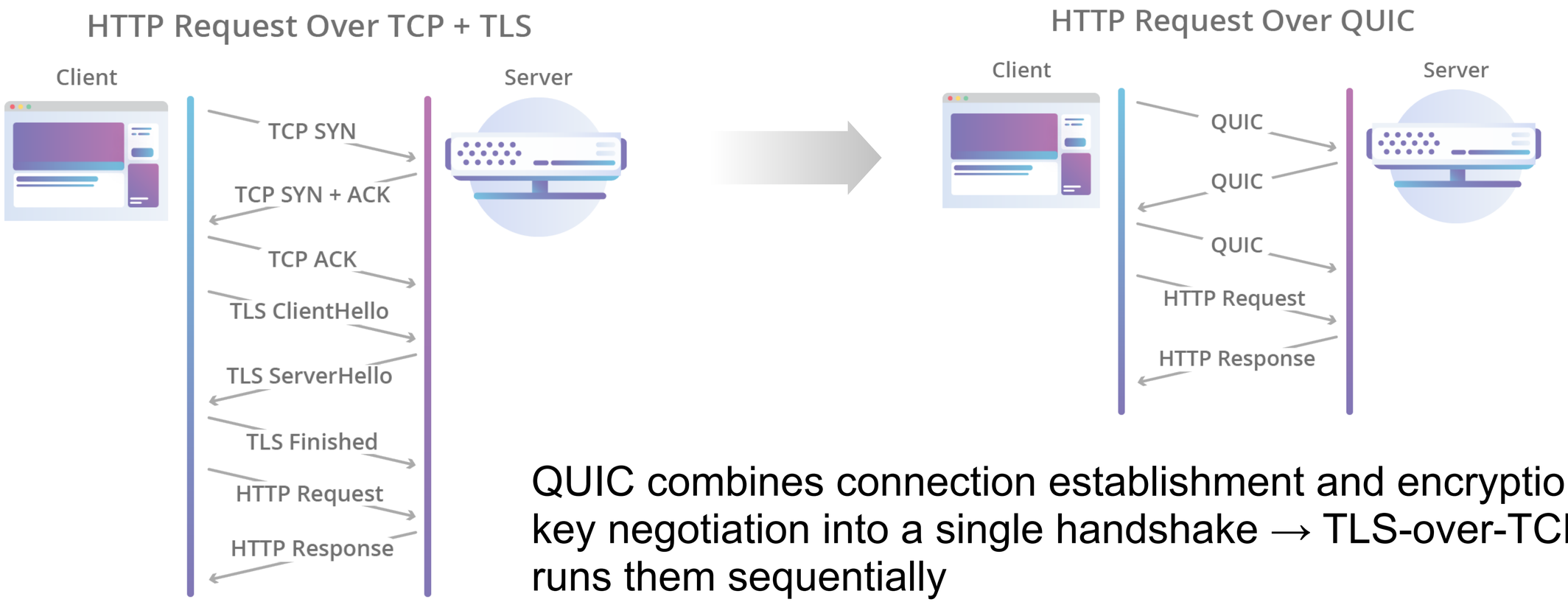


- QUIC supports TLS 0-RTT session re-establishment:
  - QUIC **Initial** packet contains CRYPTO frame with a TLS **`ClientHello`** and a **`SessionTicket`**
  - QUIC **0-RTT** packet included in the same UDP datagram contains a STREAM frame carrying idempotent 0-RTT data:

```
+-+-+-+-+-+-+-+-+
|1|1| 1 |R R|P P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0..160)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| SCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0..160)               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Length (i)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Packet Number (8..32)                  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Payload (8..)                      ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  - Server responds with data in 1-RTT (short header) packet, along with its **Initial** and **handshake** packets

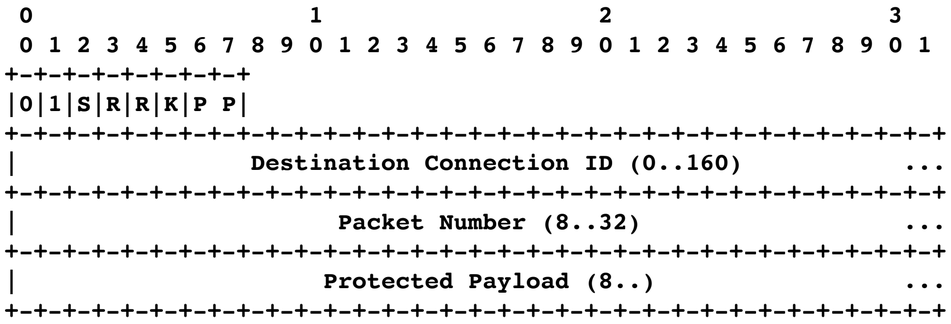
QUIC Connection Establishment (5/5)



QUIC combines connection establishment and encryption key negotiation into a single handshake → TLS-over-TCP runs them sequentially

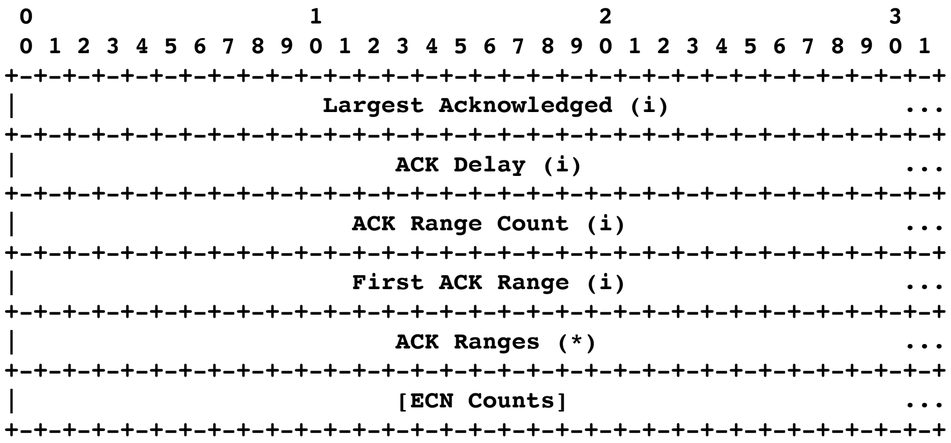
Data Transfer, Streams, and Reliability (1/3)

- After handshake has finished, QUIC switches to sending short header packets
  - The short header contains a **Packet Number** field
    - Packet numbers **increase by one** for each packet sent; ACK frames indicate received packet numbers
      - QUIC packet numbers count packets sent; TCP sequence numbers count bytes of data sent
    - QUIC **never** retransmits packets – retransmits frames sent in lost packets in new packets, with new packet numbers
      - TCP retransmits lost packet with original sequence number
  - Protected payload section of short header packets contains encrypted QUIC frames
    - STREAM frames contain data
    - ACK frames contain acknowledgements
  - Performs congestion control as in TCP → Lecture 6



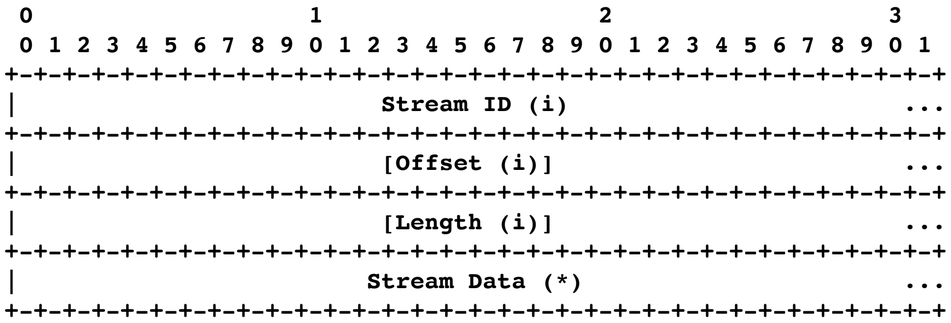
Data Transfer, Streams, and Reliability (2/3)

- QUIC sends acknowledgements of received packets in ACK frames
  - Sent **inside** a long- or short-header packets; unlike TCP, not part of headers
  - Indicate sequence numbers of QUIC **packets** that were received, not frames



Data Transfer, Streams, and Reliability (3/3)

- Data is sent within STREAM frames, sent within QUIC packets
  - Contain a stream identifier, offset of the data within the stream, data length, and data
  - QUIC provides **multiple** reliable byte streams within a single connection
    - Data for each stream is delivered reliably and in-order
    - Order is not preserved between streams
    - Avoids head-of-line blocking between streams → Lecture 5



- Can view QUIC streams as multiple unframed byte streams sent within a single connection; alternatively can view each stream as framing a message, and the connection as a series of messages

QUIC Transport Protocol

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