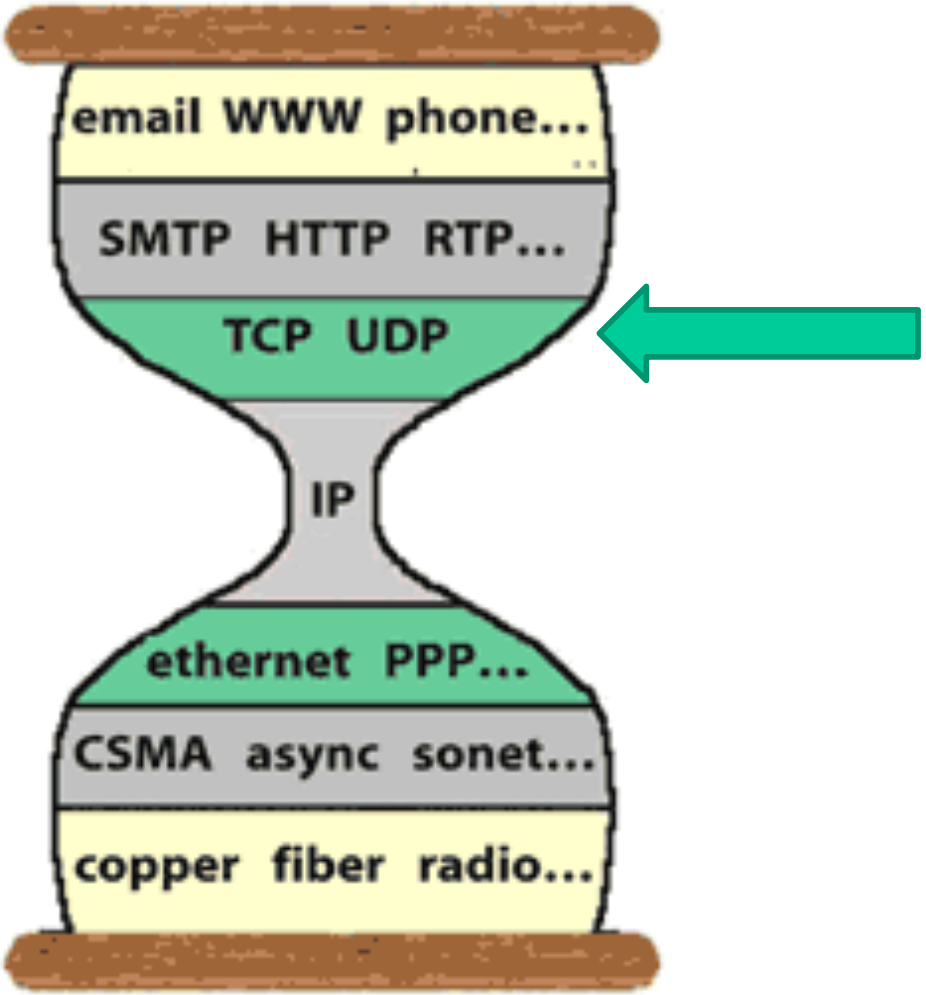
Lecture 8: QUIC



- Head of line blocking problem in HTTP/1.1
- Understand why HTTP/2 failed to fix it
- QUIC's basic concepts, how QUIC differs from TCP
  - QUIC's solution to head-of-line blocking
  - QUIC's improvement over TCP's congestion control

Why QUIC

- Make HTTP run faster
  - TCP: 40+ years old
    - Software does not wear out
    - Requirements do change over time
- Where QUIC came from
  - An assembly of lessons learned from the past in the transport protocol design
- TCP vs. QUIC: *What are the basic differences between the two?*

TCP

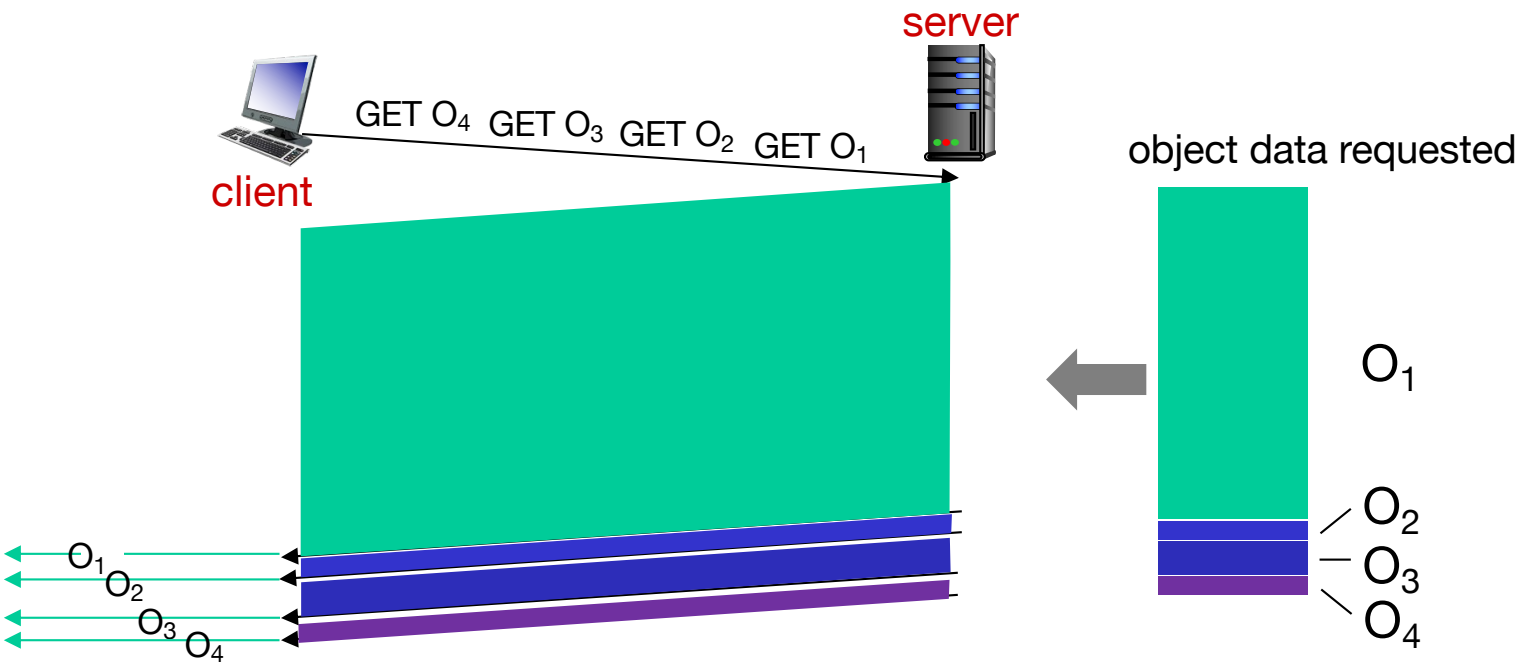
- Providing 4 basic functions
  1. Demultiplexing
  2. Reliable data delivery of a *single* byte stream
  3. Flow control
  4. Congestion control (added into the TCP implementation afterwards)
     - latched on the same window mechanism designed for flow control

needed by

- TCP connection management
  - Connection ID:
    [source addr. + port#, destination addr. + port#]
  - Connection setup: reach agreement on the initial seq# to be used by each end.
  - Connection tear down: reach agreement on the final data byte seq# of each end

HTTP/1.1: HOL blocking

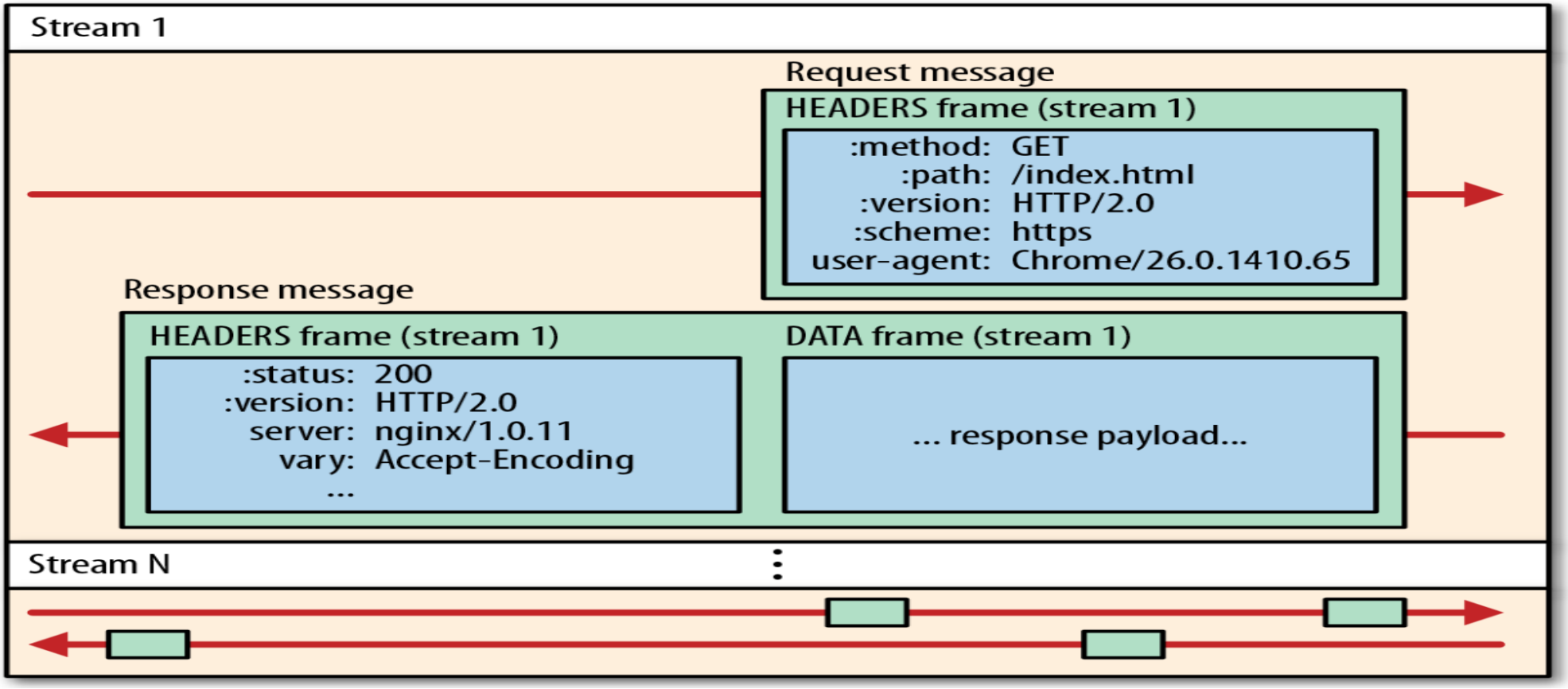
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

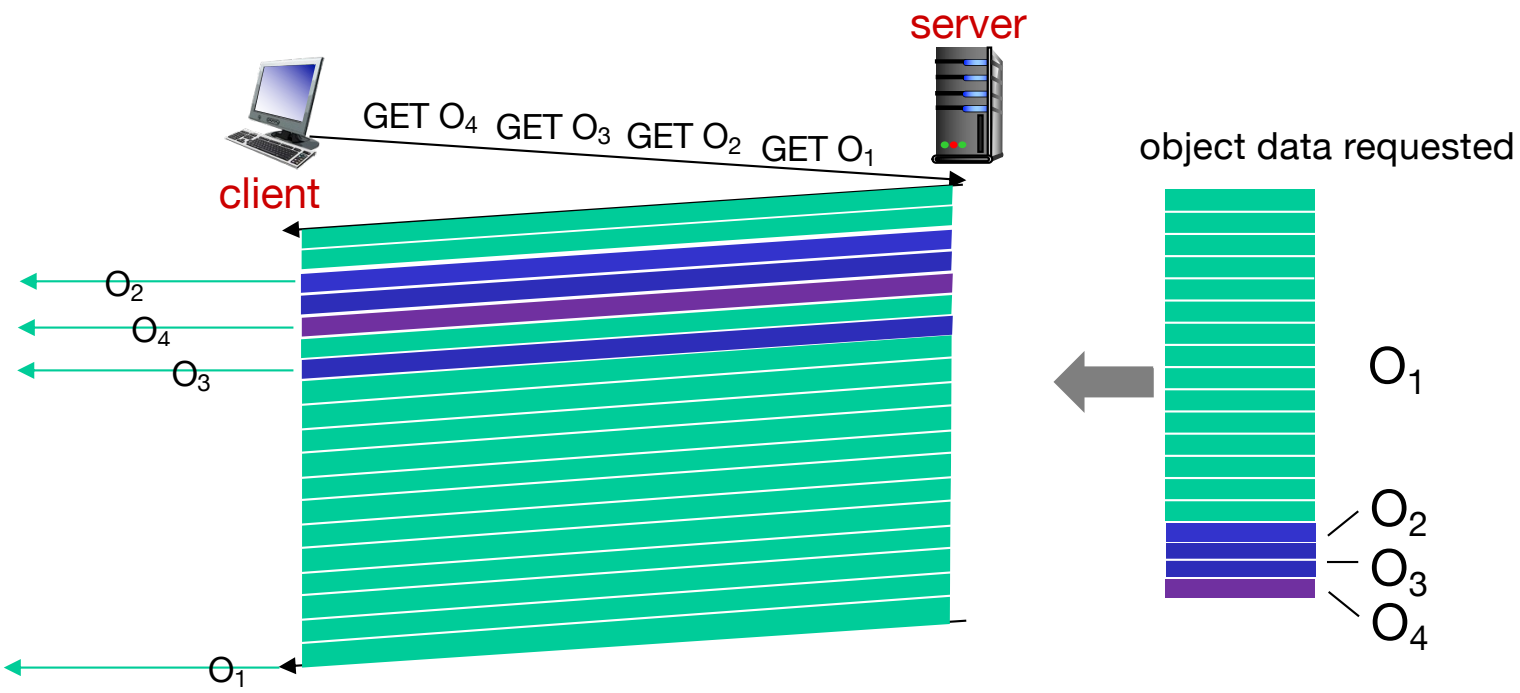
HTTP/2.0: Message, Frame, Stream

- Message: either an HTTP request or response
  - can be encoded in one or multiple frames
- **Frame**: basic communication unit, uniquely identified
- **Stream**: a virtual channel with priority, carrying frames in both directions using one stream for each [request, response] pair



HTTP/2: Mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1's finish-time slightly delayed

- What if 2nd frame of O_1 gets lost: can O_2, O_3, and O_4 be delivered to the browser app before the loss is recovered?

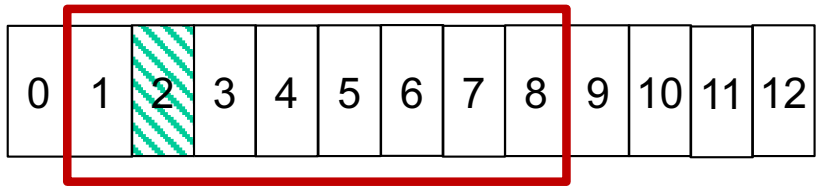
HTTP/2 Performance Improvements

- Reduced HTTP header overhead
  - Binary encoding
  - Header compression
- Attempted to remove head-of-line blocking
  - Multiple streams, one for each http request/reply
  - Big messages are broken down to multiple frames
  - Frames from all streams can be interleaved
- Above approaches avoids HOL *at HTTP level*
  - Single TCP connection between client-server → packet losses still lead to head-of-line blocking

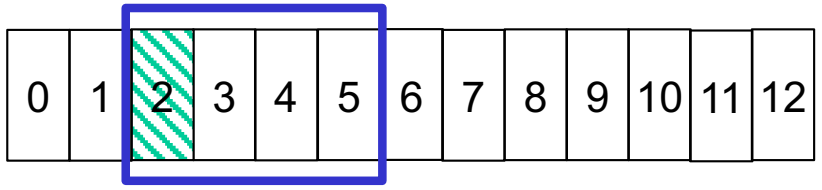
A few other TCP-induced performance issues

- Connection ID tied to IP addresses
  - Not good for mobile nodes
- **Congestion control** is tangled with the window flow control for reliable delivery
  - Congestion window: control the number of packets inside the network
  - But does it?
  - Fast Recovery
- Connection setup delay
  - TCP connection setup
    - Takes one RTT
  - Then TLS setup
    - Takes another RTT

cwnd = 8 packets, pkt-3 lost, detected by 3 dup-ACKs

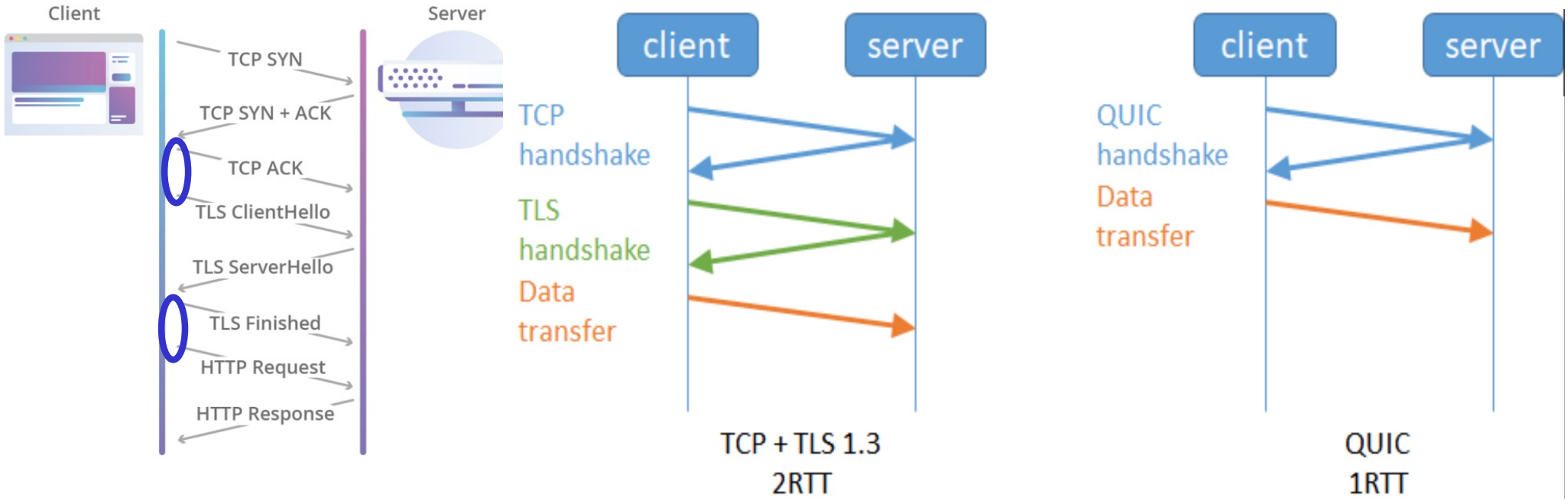


TCP connection (without FR): Cut cwnd to half, but can't send any new data



QUIC: 4 major components

1. Transport connection management
   - Connection ID: a pair of random number
     - independent from IP address & port numbers, enable a QUIC connection to continue over client address and port# changes
2. Secure connection setup is bundled together with connection setup



QUIC: 4 major components (II)

3. Structured data delivery

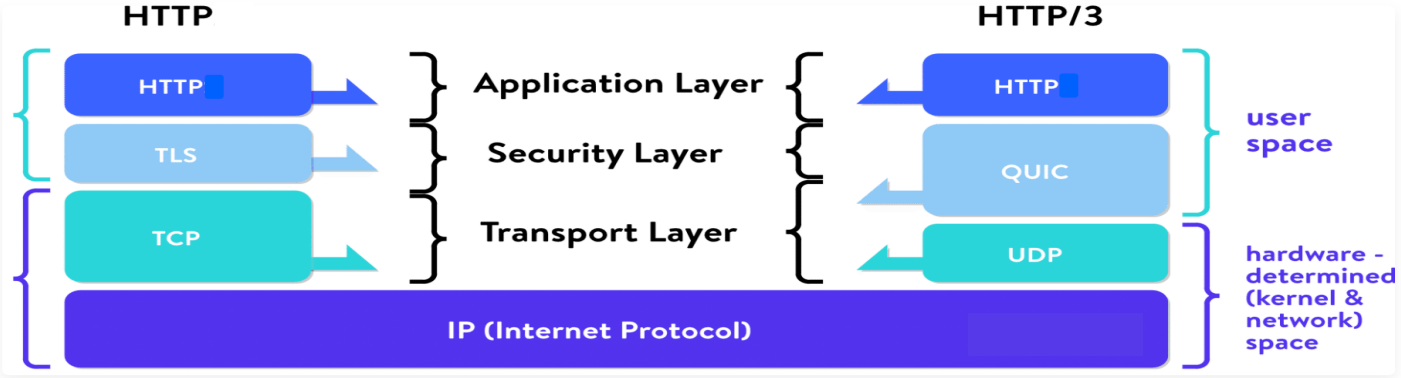
- Streams: multiplexing inside a single QUIC connection
- Frames: packaging into next outgoing QUIC packet based on priority

Inherited concepts from HTTP/2

4. Decouple congestion control from reliable data delivery

- Congestion control: count QUIC packets
- Flow control: on data byte in each stream

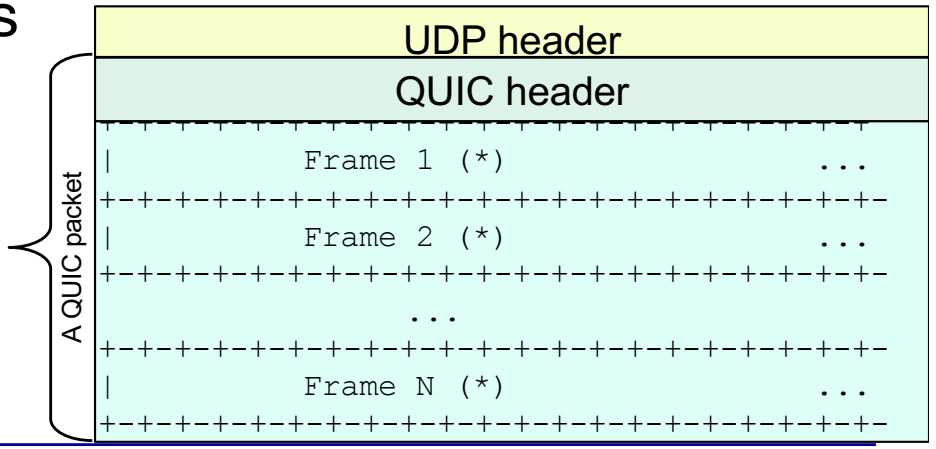
Each controls different things



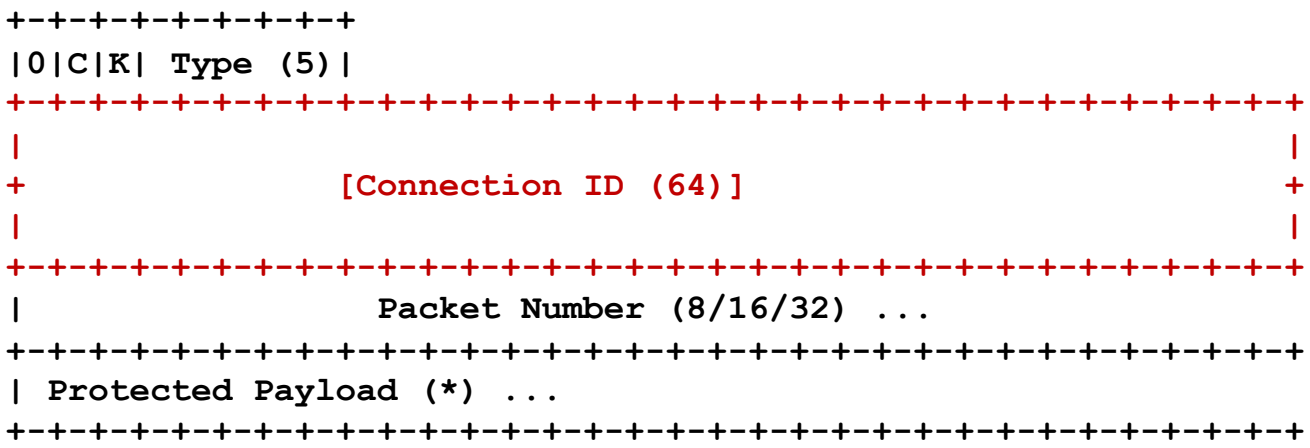
QUIC Terminology

- **Connection**: a conversation between 2 QUIC endpoints
  - Multiple **stream**s running within a single QUIC connection
- **Stream**: a single- or bi-directional byte stream, delivered within a QUIC connection
  - Each stream identified by a unique stream ID, can be established by either end
  - each stream can carry multiple **frames**
- **Frame**: basic unit in QUIC
  - QUIC defines multiple types of frames
    - we focus on **stream frame** and **ACK frame** only
  - size of a frame must not exceed a QUIC **packet** size
- **Packet:** a structured UDP payload
  - One UDP datagram encapsulates one or multiple QUIC packets
  - One QUIC packet contains one or multiple frames

An example: a UDP datagram carrying one QUIC packet



Resilience to IP address/port# change



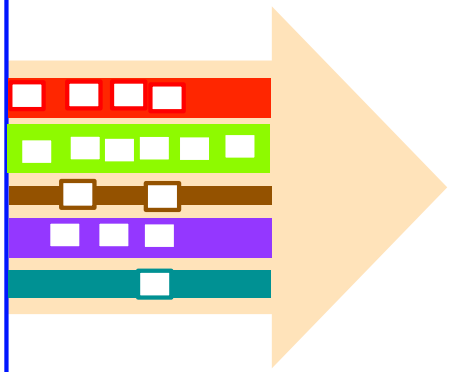
- An endpoint that receives packets containing a source IP address and port not seen before can start sending new packets with those as destination IP address and port.
- Packets from different source addresses might be re-ordered. The packet with the highest packet number MUST be used to determine which path to use.
- An endpoint MUST validate that its peer can receive packets at the new address before sending large quantity of data to that address

A TCP Connection Analogy for QUIC Terminology

- **QUIC connection**: a *super* connection, containing multiple "TCP connections"
- **QUIC stream**: "TCP connection"-equivalent
- **QUIC frame**: similar to TCP data segment
  - QUIC defines a number of control frames (not covered here)
  - **Stream frame**: contains a *data* segment
    - Difference from a TCP segment (chopped by TCP), a QUIC stream frame is segmented by the app
- QUIC packet: no correspondence in TCP

QUIC streams, frames, packet

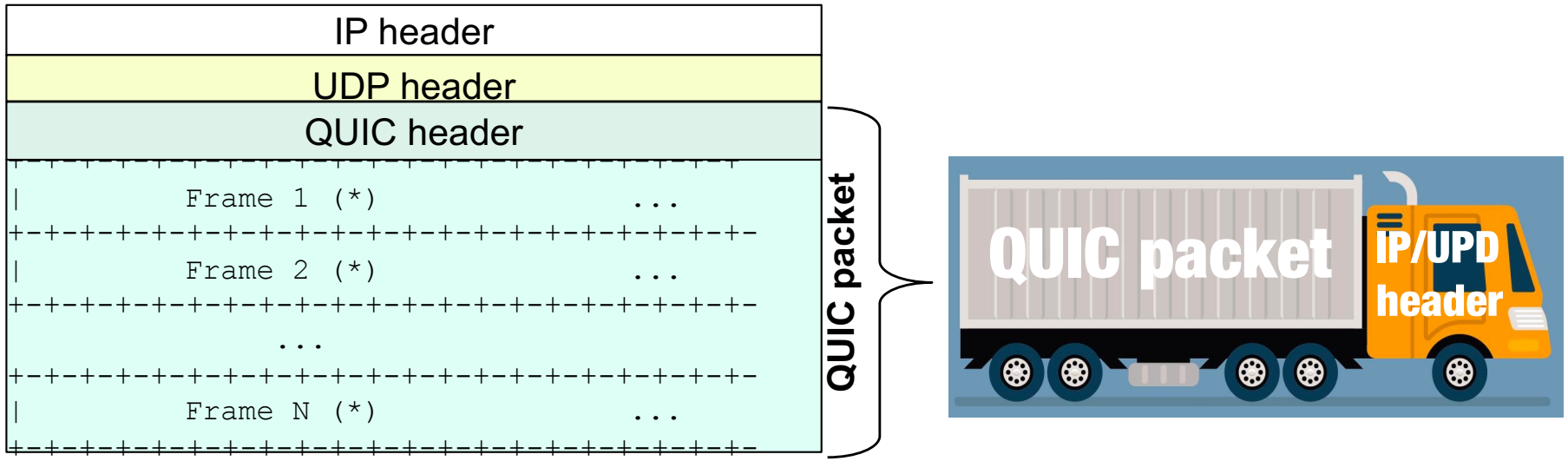
a stream: HTTP object request/reply



- Each stream may contain multiple frames
- Each stream may have different priority
- A QUIC packet is “container” of frames: pack into each outgoing QUIC packet the frames with highest priority
  - Each packet can carry multiple frames
    - can be different types of frames

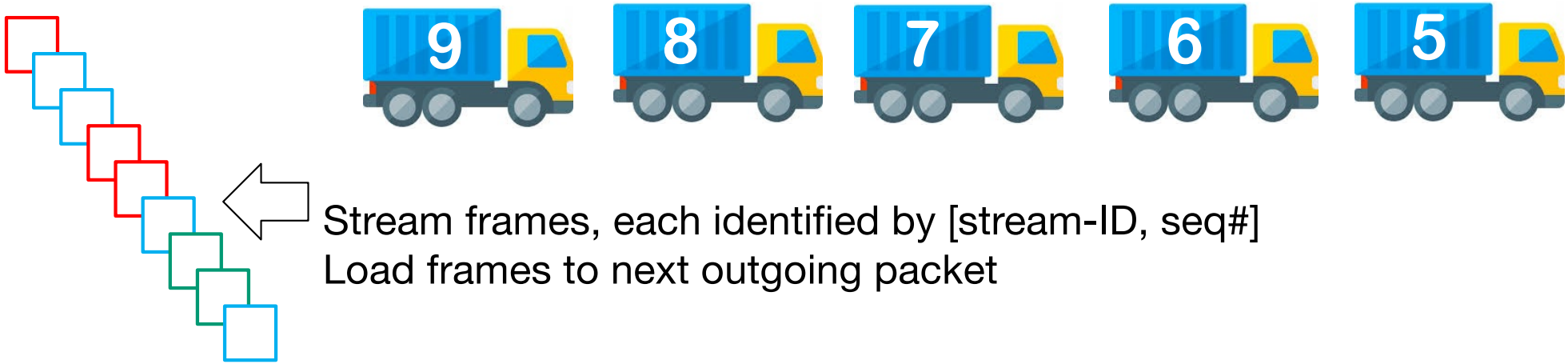
QUIC packet: containers of frames

- Data **stream frame** contains stream ID, seq#
  - Stream frames in the same packet may belong to different streams
- **ACK frame** acknowledges received QUIC packets
  - indirectly ACK the stream frames carried in those packets

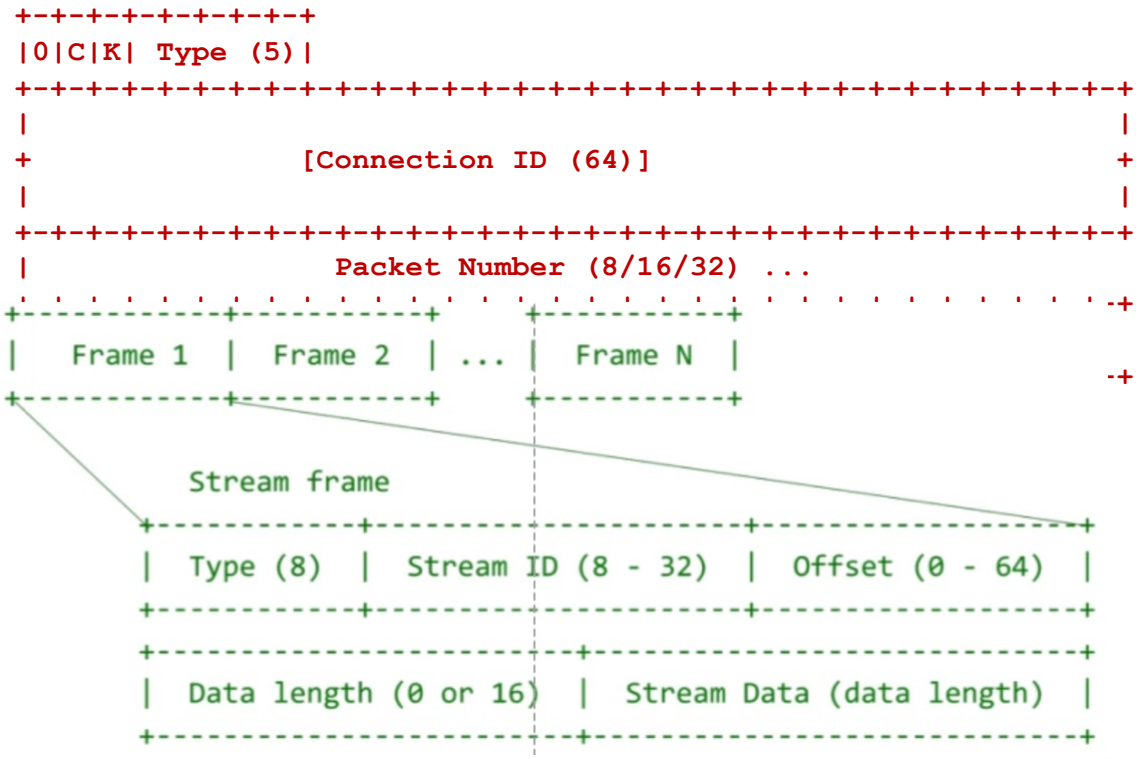


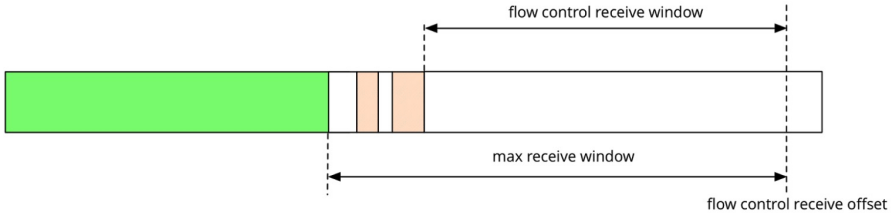
QUIC Packet Number

- Each QUIC packet is assigned a monotonically Increasing *Packet Number*
  - increase by 1 for *every QUIC packet* sent
- QUIC packet number used for loss detection
  - Receiver acknowledges all received packets by their numbers



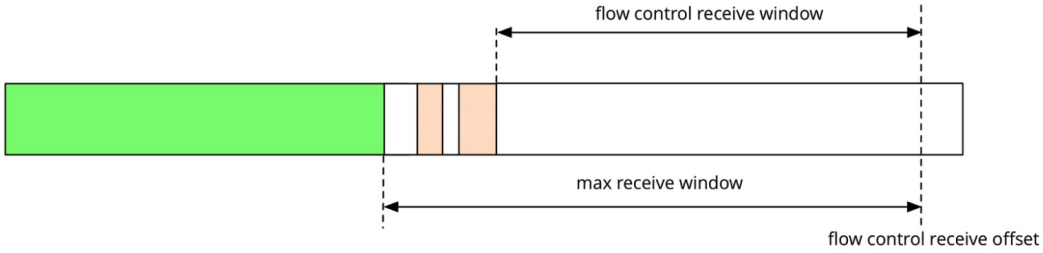
Multiplexing without head-of-line blocking



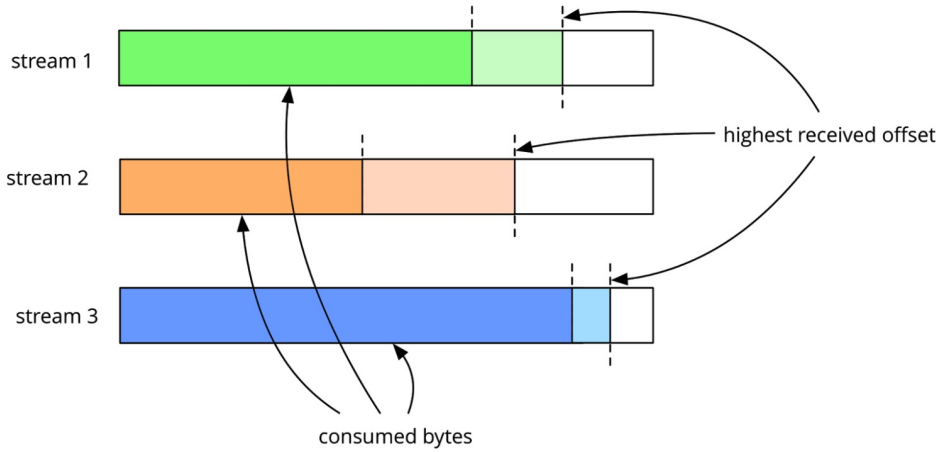


Stream and connection level flow control

- Stream flow control: same as TCP window flow control procedure (and assuring reliable byte stream delivery)

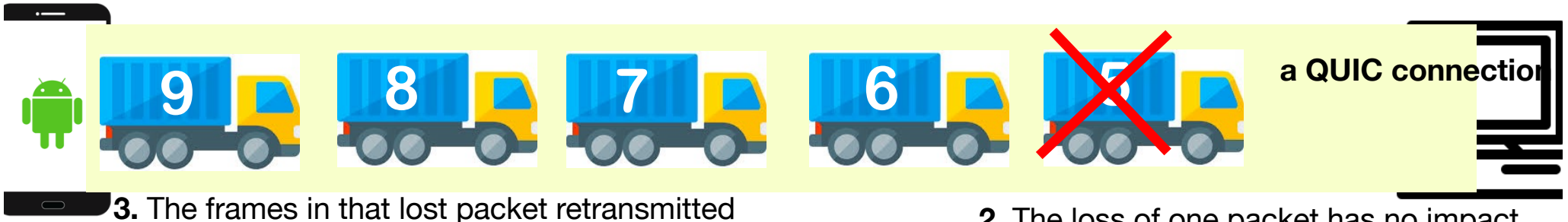


- Connection flow control: simply adds together the window sizes of all the streams



QUIC's reliable data delivery

- **Stream frame**: contains a data segment
- **ACK frame:** contains
  - ACK for the largest QUIC packet# received
  - **Selective ACKs** for all received packets
- *No packet retransmission* in QUIC
  - lost stream frames are put into future outgoing packets for retransmission



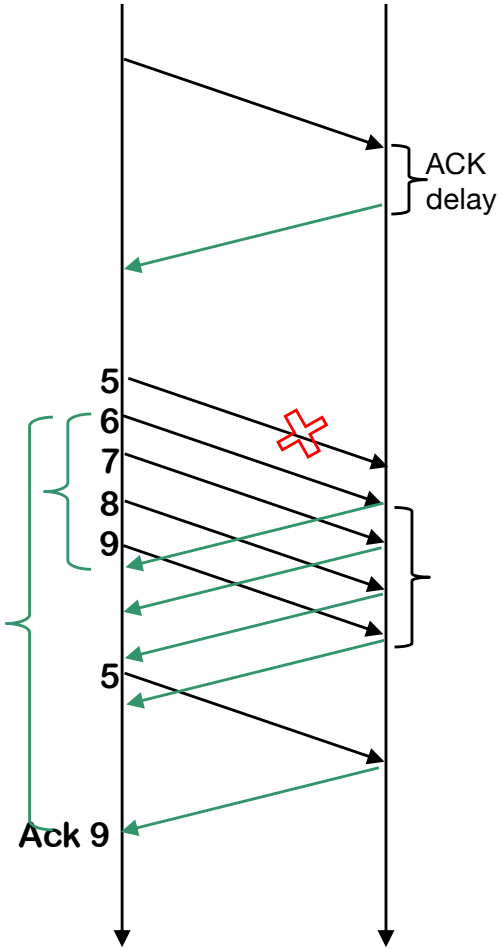
3. The frames in that lost packet retransmitted on later packets (if they need reliable delivery)

2. The loss of one packet has no impact on others (except signaling congestion)

1. Each packet can carry frames from multiple streams, which frame gets on which packet: based on priority

Improvement in RTT measurement

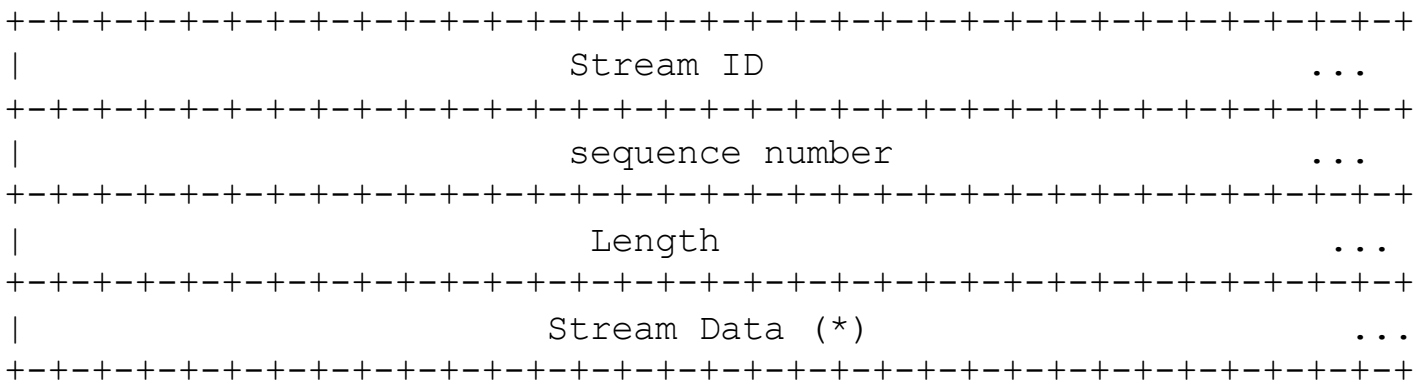
- Including in ACK frame the ACK-delay
  - Time period from receiving data to sending ACK frame
- Separation of packet acknowledgement from data reliability control
  - TCP can only use cumulative ACK to estimate round-trip-time



Stream Frame

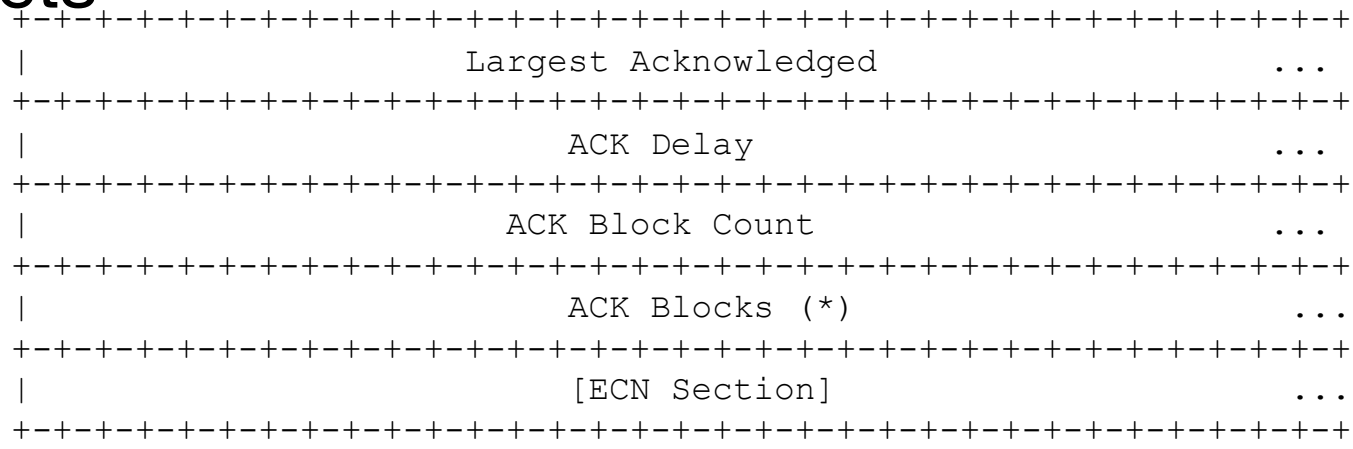
FYI

- User data is uniquely identified by its stream ID and sequence number (called “offset”)
- size of each frame must not exceed a QUIC packet size
- Each frame begins with a Frame Type byte followed by additional type-dependent fields



Information carried in an ACK frame F_{ack}

- The largest packet number P_n received
  - Together with the packet numbers of *all recently received* QUIC packets, encoded in ACK Blocks
    - Each "**ACK Block**" acknowledges a contiguous range of received packets
    - Question: How recent is recently?
- The time delay between the P_n reception time and F_{ack} sending time
- The number of ECN received, if any
  - ECN: Explicit Congestion Notification, carried in IP packets



Fitting ACK ranges in ACK frames

- An ACK frame must fit within a single QUIC packet
  - If it does not, then older ranges (those with the smallest packet numbers) are omitted.
- The receiver SHOULD repeatedly acknowledge newly received packets
- (if data flows in both directions) When a packet containing an ACK frame F_a is acknowledged, the receiver can stop acknowledging packets less than or equal to the largest acknowledged in F_a.
- *No guarantee* that every acknowledgement is seen by the sender before it is no longer included in the ACK frame
  - consequence? spurious data frame retransmissions

QUIC Loss Detection

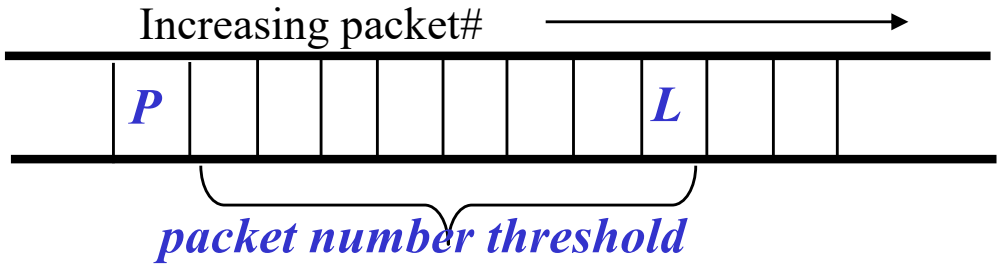
- QUIC receiver sends ACK frames to inform to the sender of all the packets it has received
- **ACK frame** acknowledges QUIC packets
  - Can be carried in any packet going the other direction
  - By acknowledging received packets: *indirectly* acknowledges the stream frames received, and the stream frames not received
- All frames carried in a packet ***P*** are considered lost, when ***P*** is assumed lost
- QUIC does not retransmit any lost packets
  - If a lost *frame* needs retransmission, it will be put onto future outgoing QUIC packet

two types of threshold

FYI

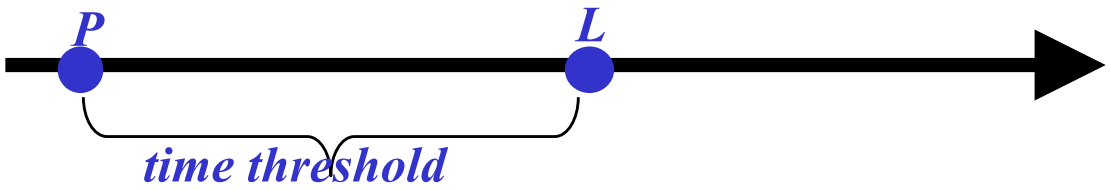
- All frames carried in a packet ***P*** are considered lost when ***P*** is assumed lost
- 2 types of threshold for packet loss detection
  1. packet number based:

     P's seq# **<** the largest ACKed packet# - a given threshold



  2. time-based:

     P's transmission time **<** the largest ACKed packet# **-** a given time threshold



When to declare a QUIC packet is lost

Recall in TCP, the sender retransmit when

1. Receive 3 duplicate ACKs, or
2. Retransmission timeout

1. 3 dup-ACK equivalence in QUIC: packet P is considered lost if
   - P is unack'ed, inflight, and was sent prior to an ACK'ed packet, and
     - P's packet number is smaller than an acknowledged packet by at least 3

     e.g.
       - sending packets ....4, 5, 6, 7, 8, 9, 10 ....
       - packets 4—6, and 8—10 are ACK'ed
       - Packets 6 & 7: considered lost
     - P was sent at least *t-threshold* seconds ago

2. ~~Retransmission timeout~~
   - ~~Retransmission timer~~ computation: *similar* to that of TCP (more elaborated)

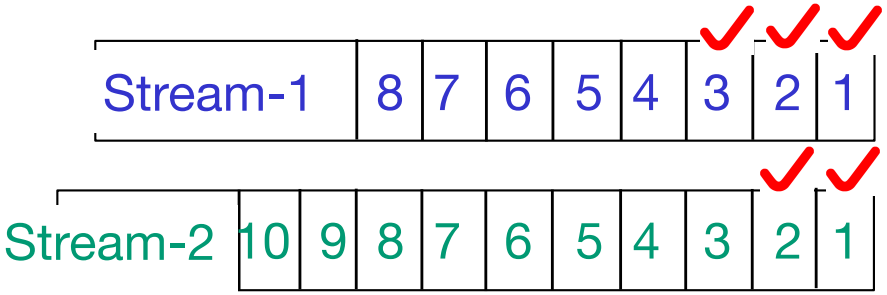
Probe Timeout (PTO)

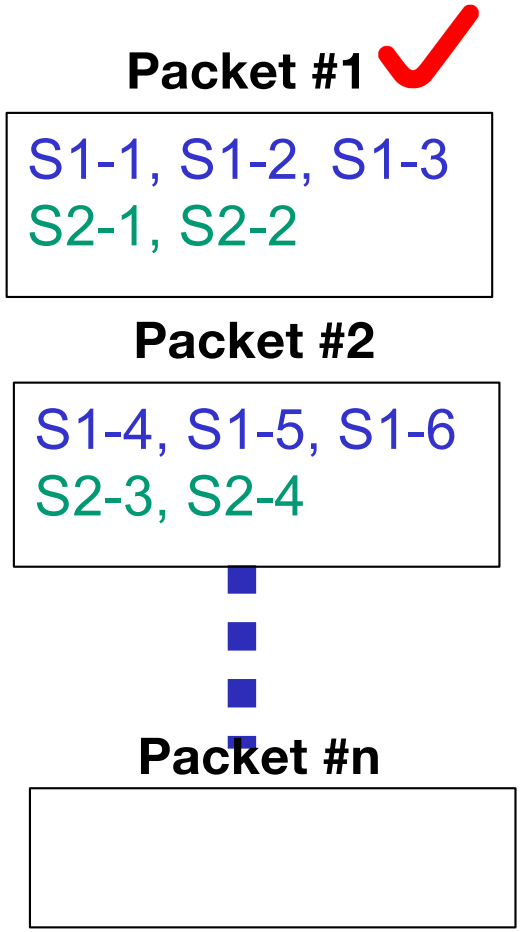
QUIC Probe Timeout

- Probe Timeout (PTO), similar to RTO
- When a PTO timer expires, a sender MUST send at least one ack-eliciting packet
  - SHOULD carry new data when possible
    - MAY retransmit unack'ed data when new data is unavailable, or when flow control does not permit new data to be sent
  - MAY send up to two datagrams containing ack-eliciting packets
    - to avoid an expensive consecutive PTO expiration due to a single lost datagram
- A PTO timer expiration event does not indicate packet loss
  - Do not mark unacknowledged packets as lost
  - But multiple PTOs signal *persistent congestion*

Acknowledge Packets, Retransmit Frames

- QUIC packet identified by packet number
- stream frame identifier: streamID + seq#
- Sender keeps mapping between packet# & stream frame identifier



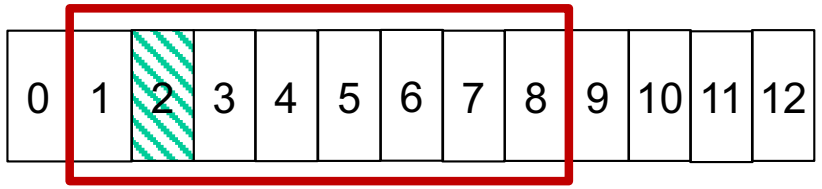


- When receive ACK for a packet number, sender marks all the frames carried in that packet *ACKed*
- If a frame is deemed lost, it is put into the transmission queue by priority order
- The next outgoing packet: pack in as many frames as possible from the top of the transmission queue

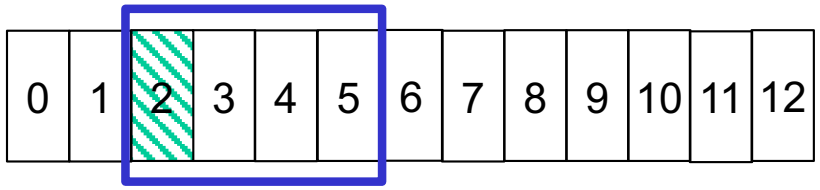
QUIC Congestion Control

- Congestion detection: (still) based on detecting packet losses
  - Can make use of ECN early congestion notification if available
- Congestion Control: (still) uses window-based congestion control scheme
  - QUIC adopted TCP congestion control
- QUIC's improvement for congestion control: using on packet-number that is decoupled from stream's flow control window
  - QUIC packet number: used by congestion control
  - Stream frame sequence number: used by flow control window and reliable delivery

Simplified Congestion Control



In case of a **TCP** connection without FR:
Cut cwnd to half, can't send any new data



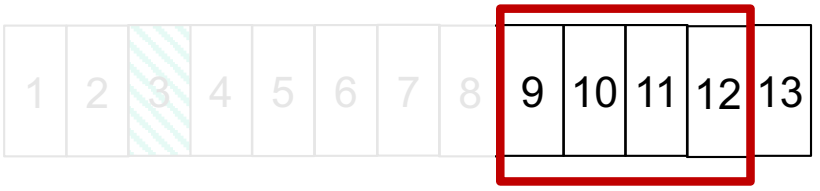
In case of a **TCP** connection with FR?

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Largest Acknowledged: 8       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ACK Delay: 20msec            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ACK Block Count: 2           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ACK Block 1: 3-8             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   gap: 1                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ACK Block 2: 1-1             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A simplified ACK frame

In case of a QUIC connection: Cut cwnd to half; packet-8 received, just send next 4 packets

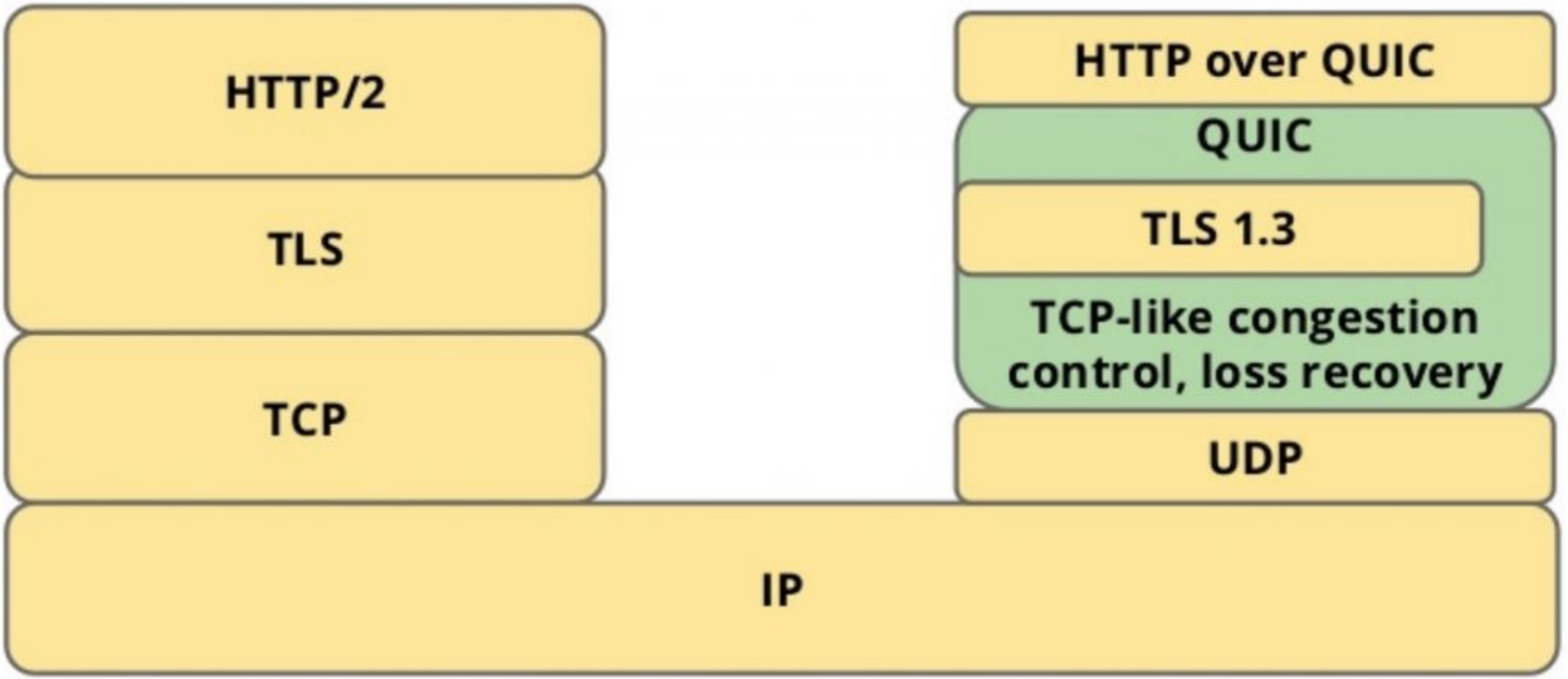
- QUIC congestion window has no concern with loss recovery



Transport Layer Security (TLS)

- Runs over a reliable connection between two end points to provide confidentiality and data integrity
- TLS is visible to application, making it aware of the cipher suites and authentication certificates negotiated during the set-up phases of a TLS session
- TLS itself has 2 layers
  - TLS Record Protocol
  - TLS Handshake Protocol

Comparison of protocol stack changes delivered with each new version after HTTP/1.0		
HTTP/1.1	**HTTP/2**	**HTTP/3**
• Some methods and response codes are added. • "Keep-Alive" becomes officially supported. "Host" header becomes supported for Virtual Domain. • Syntax and semantics are separated.	• Support of parallel request transmission by "stream" (elimination of ***HTTP*** HoL Blocking). • Addition of flow-control and prioritization function in units of "stream". • Addition of server-push function (send related file without request.)	• Lower protocol changes from TCP+TLS to UDP+QUIC • Streams and flow-control function are moved to QUIC. • Parallel request transmission is supported by QUIC stream (eliminating ***TCP*** HoL Blocking).


HTTP/2
TLS
TCP
IP
HTTP over QUIC
QUIC
TLS 1.3
TCP-like congestion control, loss recovery
UDP