Introduction of Lightweight Quick UDP Internet Connection (QUIC) Protocol

- **Description:**
  - QUIC, a reliable transport layer protocol, was designed to solve current problems on TCP and replace it. This talk will introduce it in following topics. What is QUIC? Why QUIC becomes popular? How to implement it with switchyard and socket? And what my future work towards this project?

- **Speaker:** Shirley Shu (ssjtoshirley@gmail.com)
  - 4th year CS student
  - Directed study supervised by Dr. Jianping Pan

What is QUIC

- Quick UDP Internet Connection
- a transport layer network protocol designed by Google
- First announced in 2013, applied in Chrome
- In 2018, the IETF[1] officially revealed “HTTP/3” using QUIC instead of TCP in HTTP/2
- Now supported by Firefox, Safari, Microsoft Egde, etc

[1] https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force

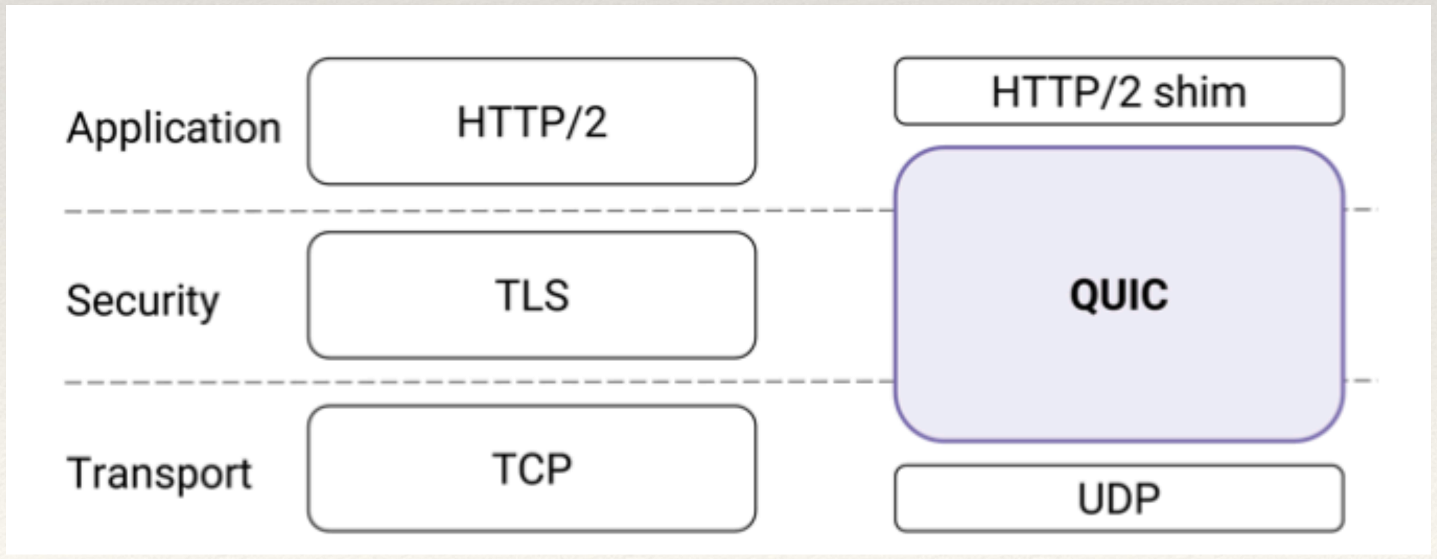
TCP vs QUIC

TCP

- Fixed in middlebox and operating system
- Head-of-line blocking
- Three-way handshake
- Loss recovery with timeout

QUIC

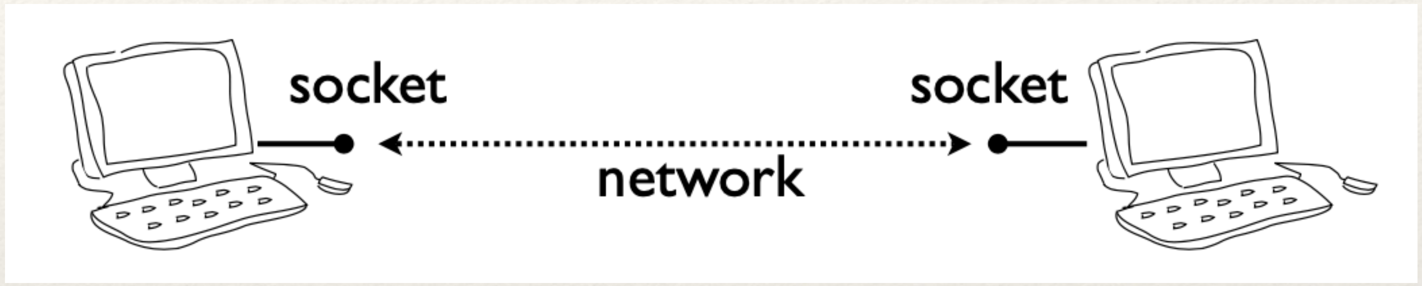
- Build on application layer and based on UDP
- Stream multiplexing
- 0-RTT handshake
- Low-latency loss recovery



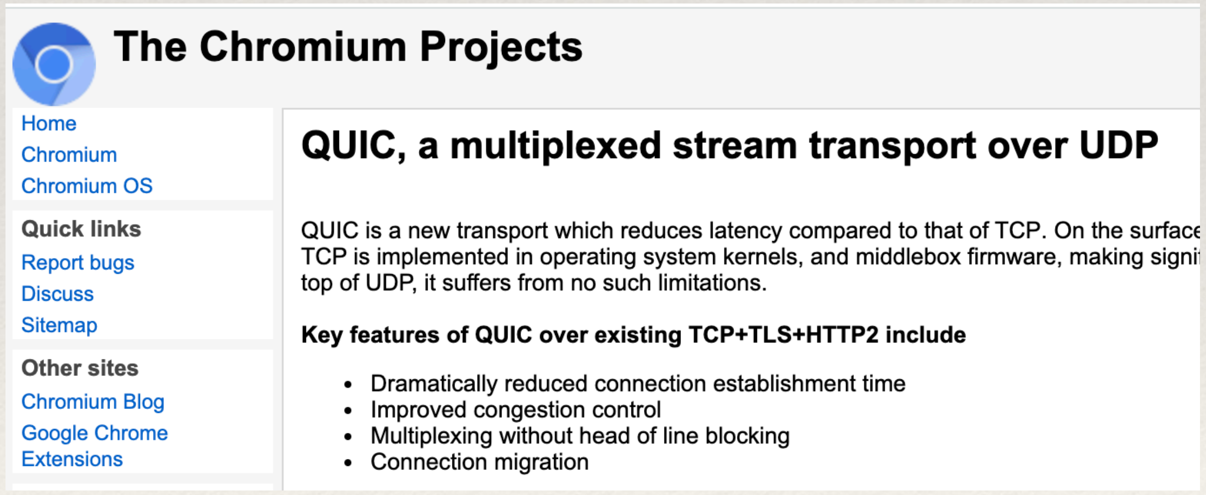
Source: https://dl.acm.org/doi/abs/10.1145/3098822.3098842

How to implement QUIC

- Network Simulator
  - Switchyard
  - UDP socket
- QUIC Functions
  - Connection Management
  - Stream Multiplexing
  - Reliable Transmission Control (EC, FC, CC)



https://webhome.csc.uvic.ca/~mcheng/361/fall.2019/handouts/python-network- programming.pdf


The Chromium Projects

Home
Chromium
Chromium OS

Quick links
Report bugs
Discuss
Sitemap

Other sites
Chromium Blog
Google Chrome Extensions

QUIC, a multiplexed stream transport over UDP

QUIC is a new transport which reduces latency compared to that of TCP. On the surface
TCP is implemented in operating system kernels, and middlebox firmware, making signi
top of UDP, it suffers from no such limitations.

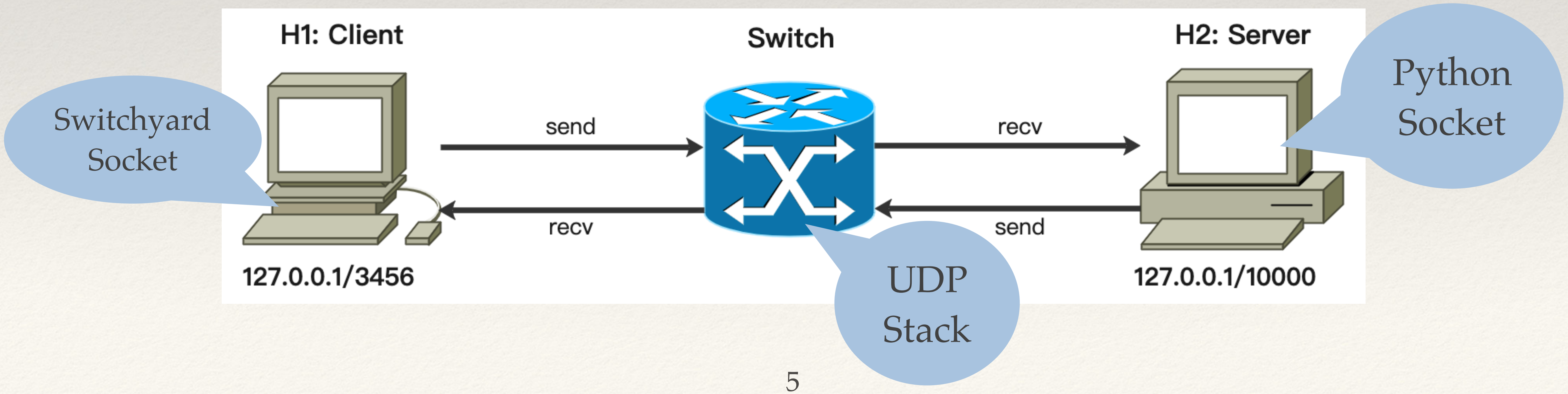
Key features of QUIC over existing TCP+TLS+HTTP2 include

- Dramatically reduced connection establishment time
- Improved congestion control
- Multiplexing without head of line blocking
- Connection migration

https://www.chromium.org/quic

Switchyard & Socket

- **Switchyard**: a framework for building real networked system software
  - Open source library in Python
  - Network **middlebox**: packet capture and transmission
  - Application layer socket method
- **Socket:** the communication **endpoint** in the network
  - Open source library in Python
  - Binding with IP address and transmission type

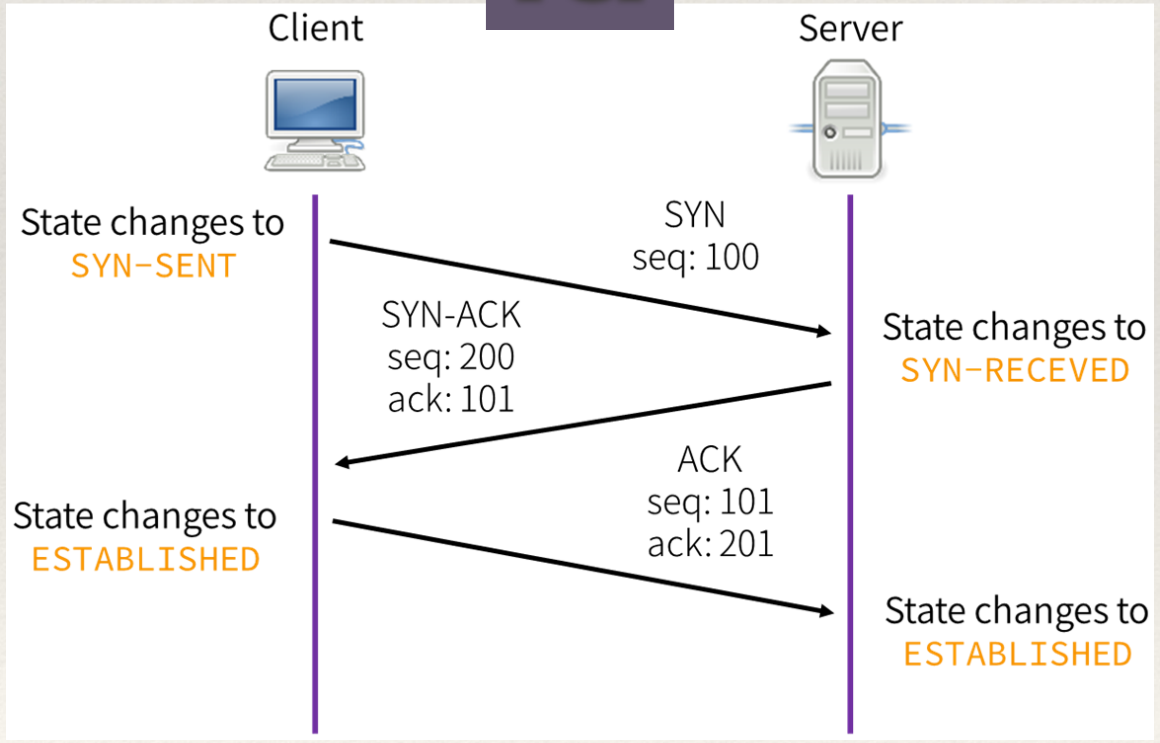


Connection Establish

- Three-way handshake

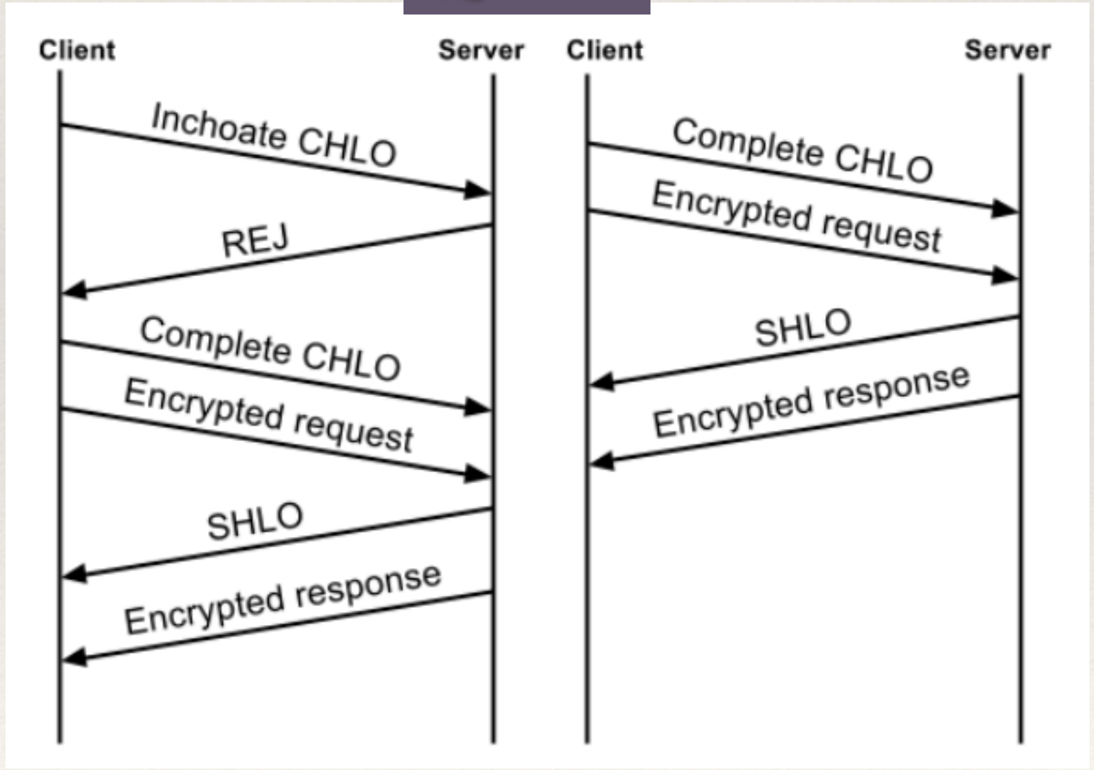
- 1-RTT: no key of server
- 0-RTT: have key of server

TCP



Client
Server
State changes to SYN-SENT
SYN seq: 100
SYN-ACK seq: 200 ack: 101
State changes to SYN-RECEVED
ACK seq: 101 ack: 201
State changes to ESTABLISHED
State changes to ESTABLISHED

QUIC

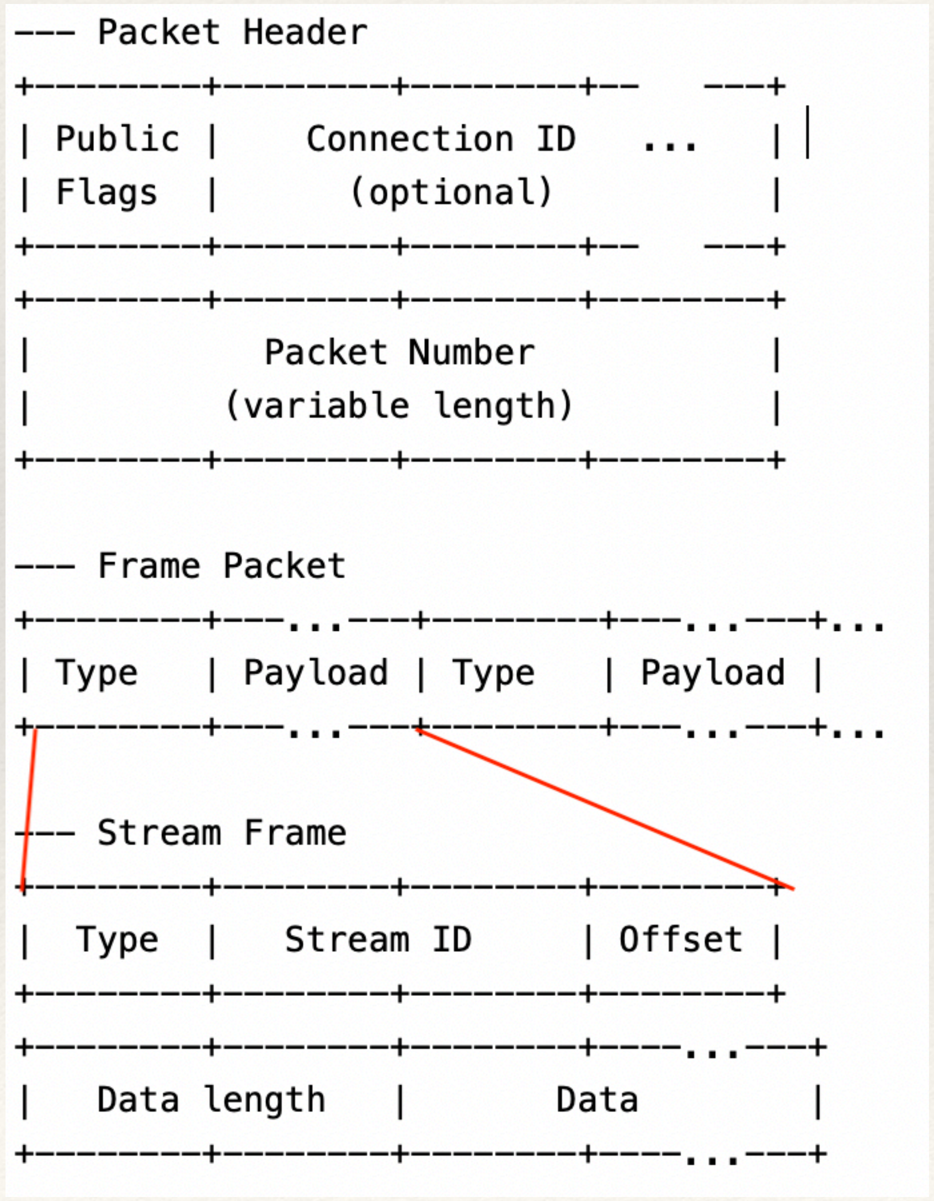


Source: https://www.mdpi.com/2076-3417/6/11/358/htm

Source: https://dl.acm.org/doi/abs/10.1145/3098822.3098842

Stream Multiplexing

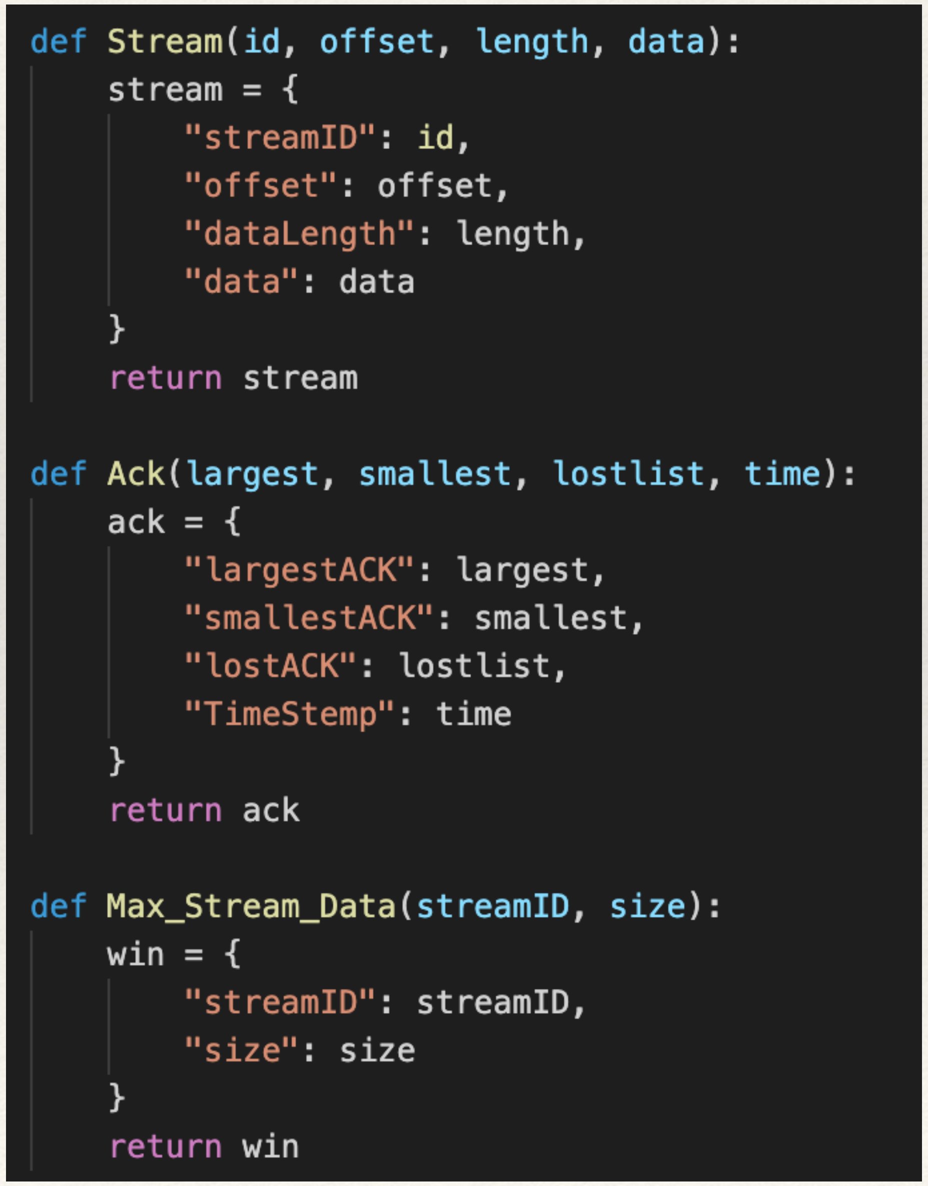
- Several frames in one packet



```
--- Packet Header
+--------+--------+--------+--    ---+
| Public |    Connection ID    ...   | |
| Flags  |       (optional)          |
+--------+--------+--------+--    ---+
+--------+--------+--------+--------+
|             Packet Number          |
|          (variable length)         |
+--------+--------+--------+--------+

--- Frame Packet
+--------+---...---+--------+---...---+...
| Type   | Payload | Type   | Payload |
+--------+---...---+--------+---...---+...

--- Stream Frame
+--------+--------+--------+--------+
|  Type  |   Stream ID     | Offset |
+--------+--------+--------+--------+
+--------+--------+--------+----...---+
|   Data length   |       Data        |
+--------+--------+--------+----...---+
```



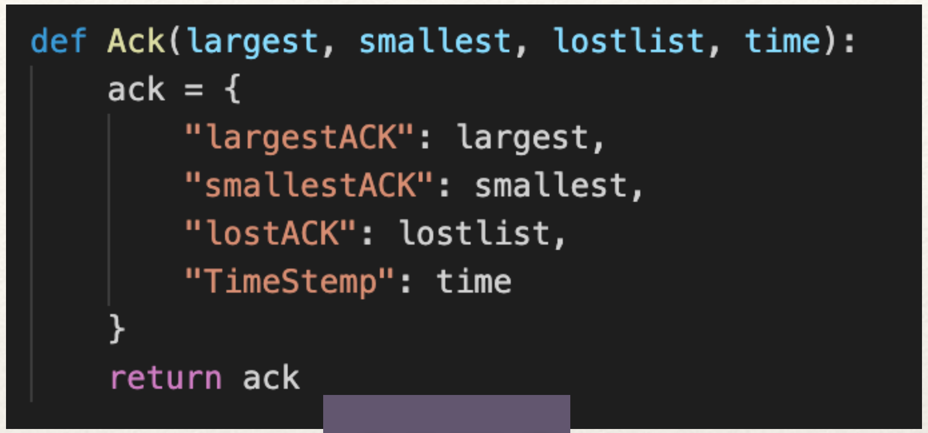
```python
def Stream(id, offset, length, data):
    stream = {
        "streamID": id,
        "offset": offset,
        "dataLength": length,
        "data": data
    }
    return stream

def Ack(largest, smallest, lostlist, time):
    ack = {
        "largestACK": largest,
        "smallestACK": smallest,
        "lostACK": lostlist,
        "TimeStemp": time
    }
    return ack

def Max_Stream_Data(streamID, size):
    win = {
        "streamID": streamID,
        "size": size
    }
    return win
```

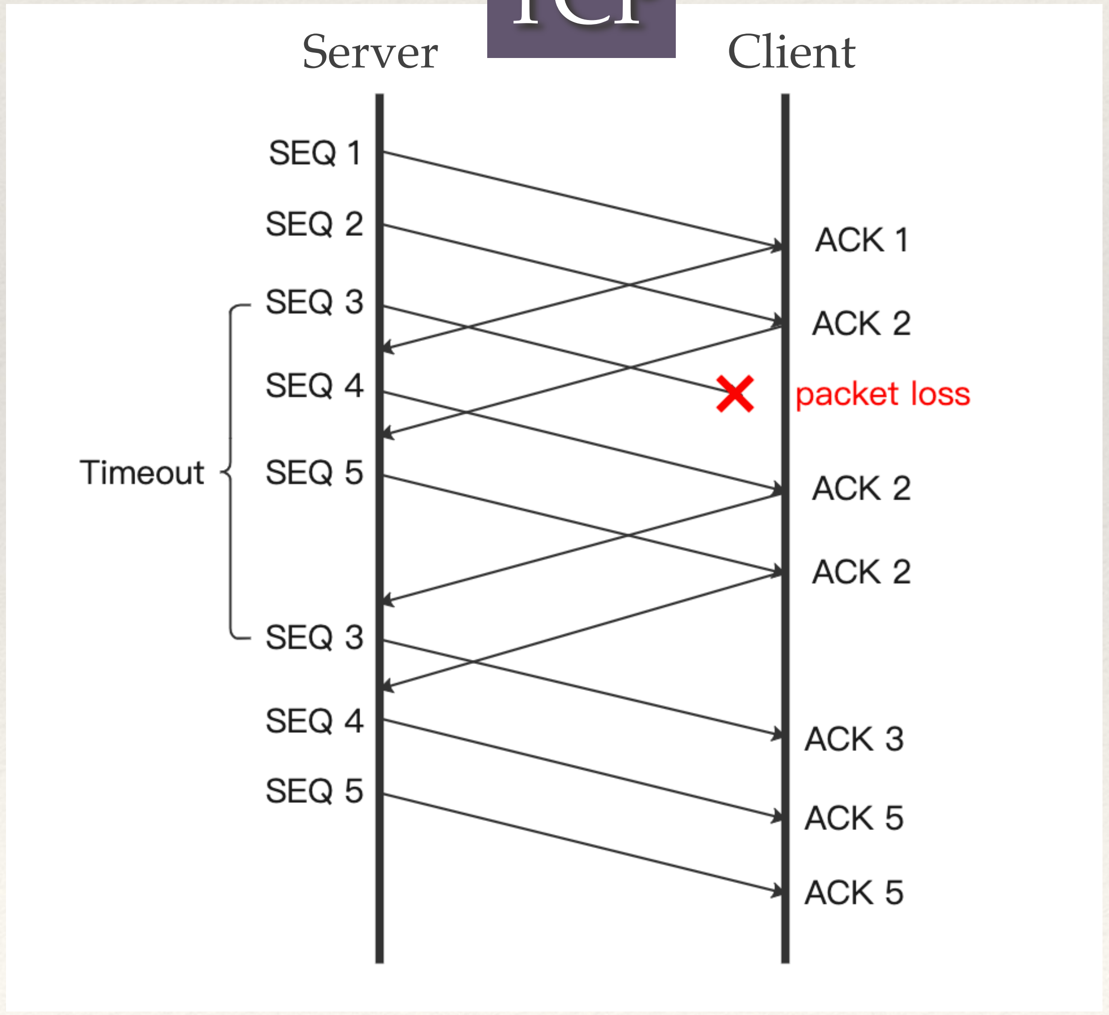
Error Control

- Based on ACK frame

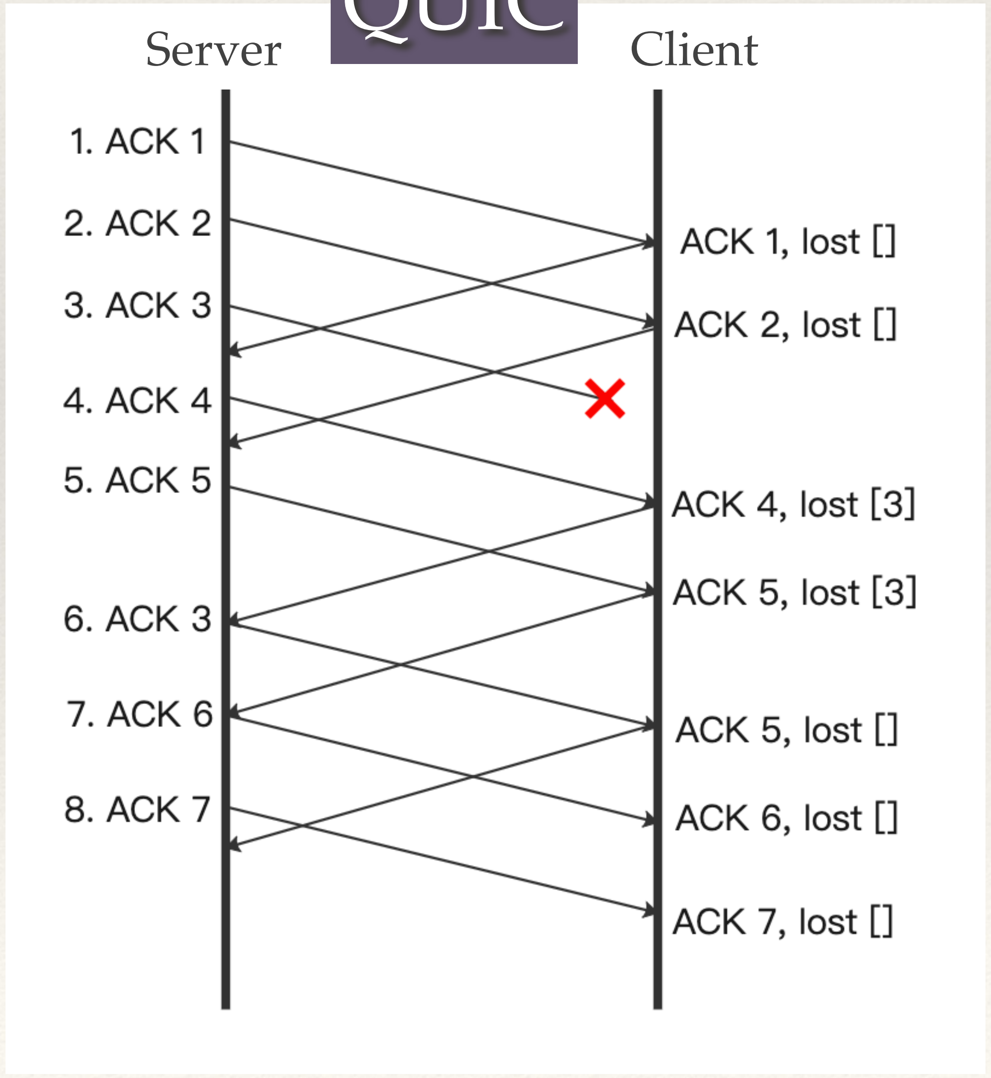


```
def Ack(largest, smallest, lostlist, time):
    ack = {
        "largestACK": largest,
        "smallestACK": smallest,
        "lostACK": lostlist,
        "TimeStemp": time
    }
    return ack
```

TCP

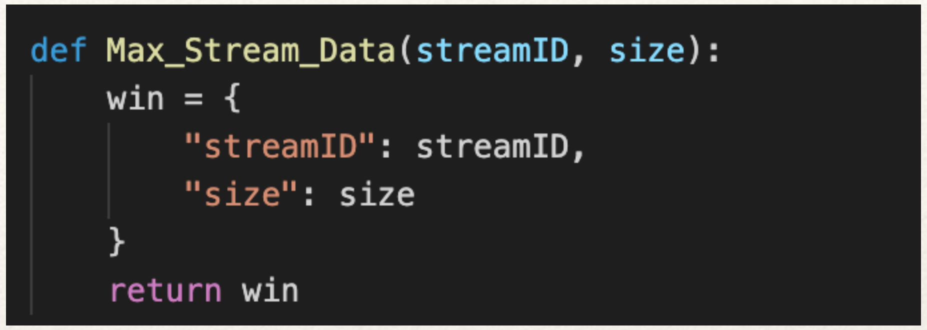


QUIC

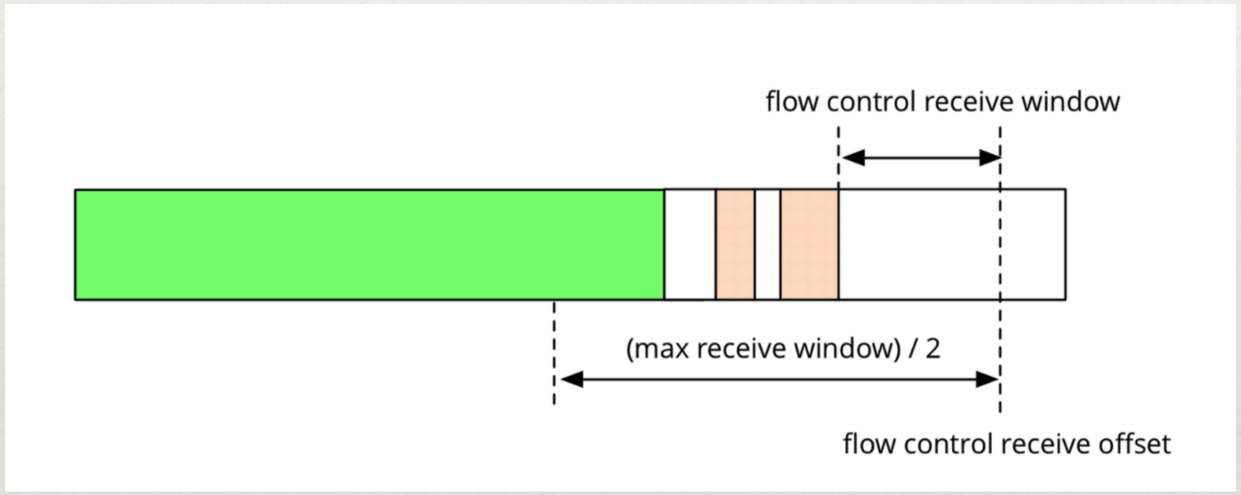


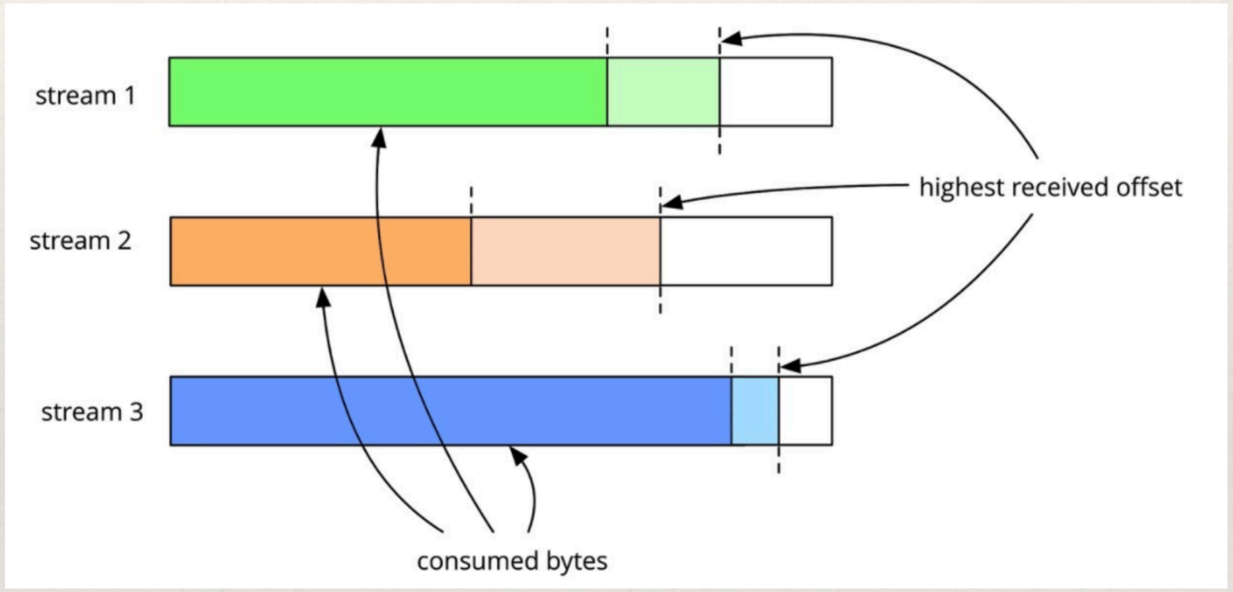
Flow Control

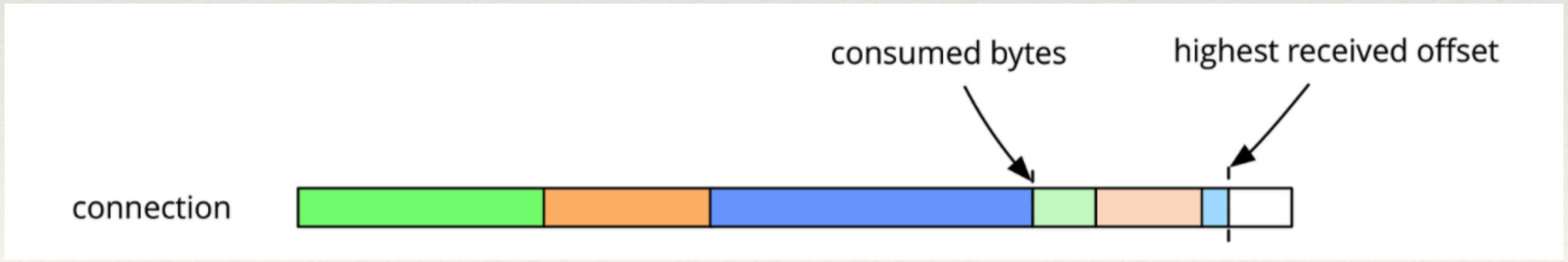
- Works on each stream



```python
def Max_Stream_Data(streamID, size):
    win = {
        "streamID": streamID,
        "size": size
    }
    return win
```

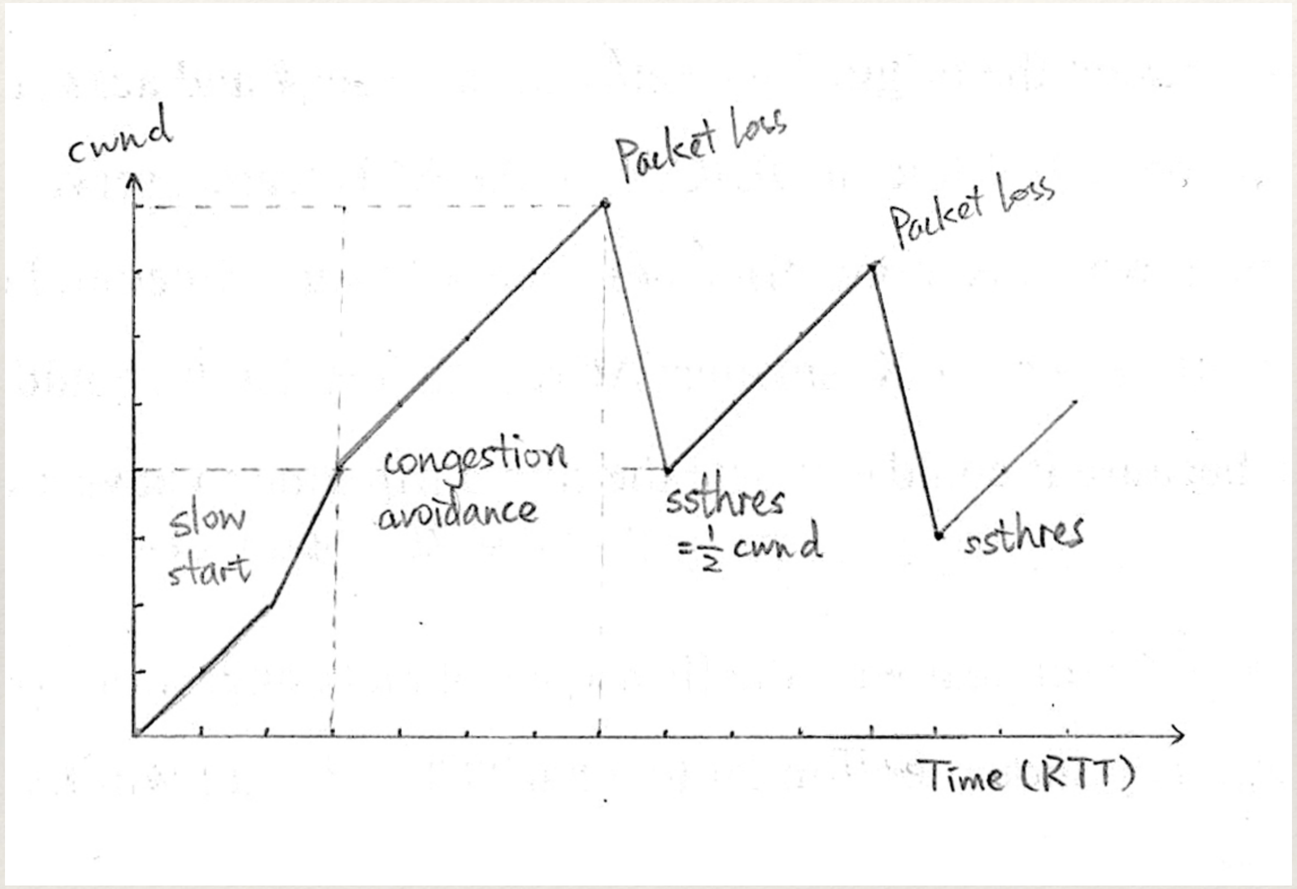






Source: https://docs.google.com/document/d/1F2YfdDXKpy20WVKJueEf4abn_LVZHhMUM S5gX6Pgjl4/edit

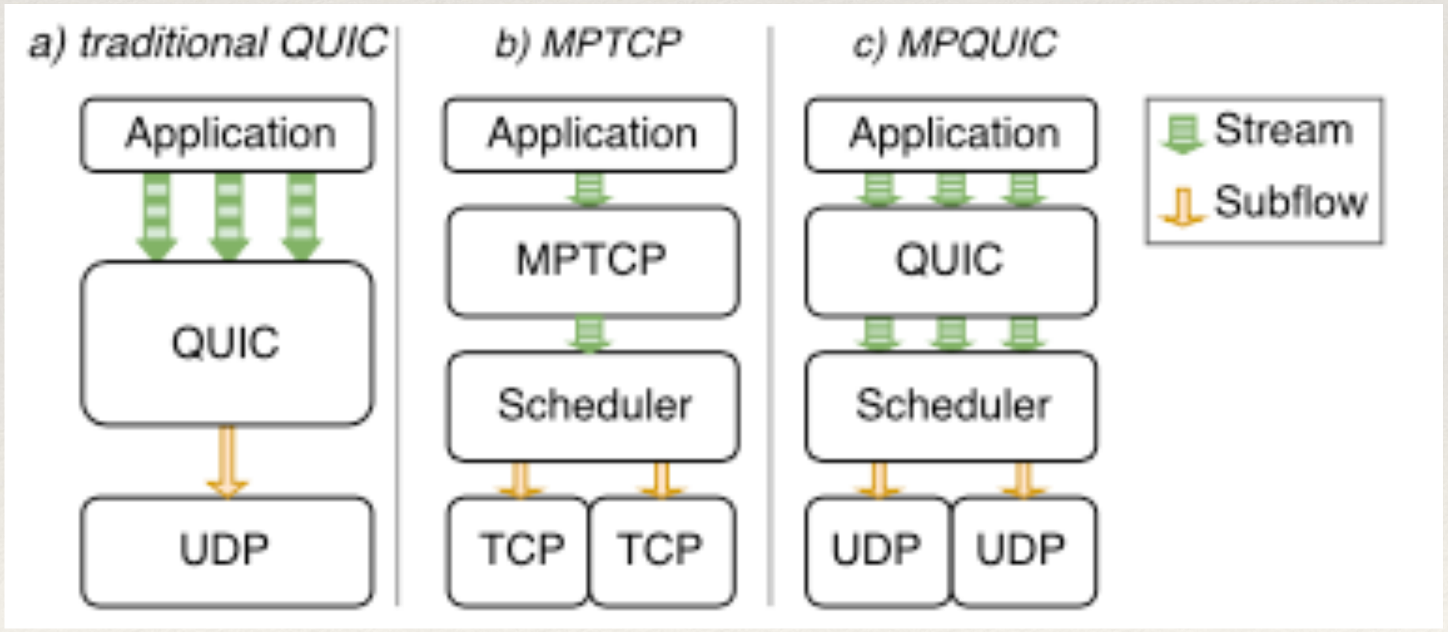
Congestion Control



- Slow start
  - cwnd = 2 * cwnd for each RTT
  - Stop at: cwnd = ssthres
- Congestion avoidance
  - cwnd = cwnd + 1 for each RTT
  - Stop at: packet loss happen
  - ssthres = half of cwnd

Further work

- Multi-path transmission over QUIC
- Authentication and encryption



Thanks!

Any questions?