CS120: Computer Networks

Lecture 19. Other Topics in Transportation Layer

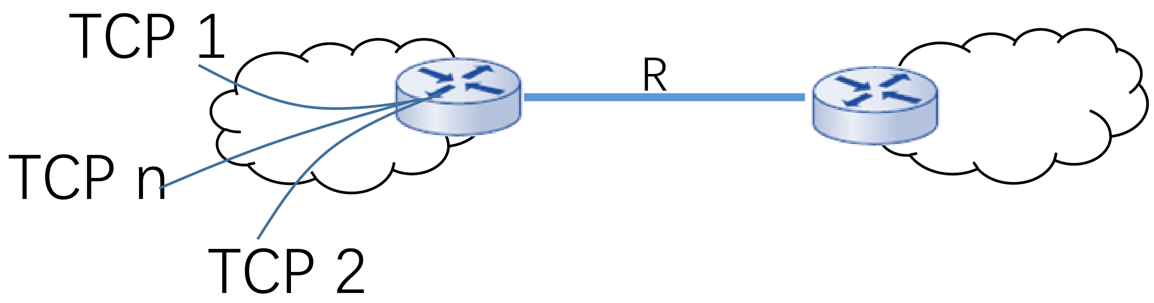
Zhice Yang

Outline

- TCP Fairness
- QoS
- QUIC

Evaluation Criteria

- Defining fairness is hard
  - In terms of a host, a TCP link, or an application ?
- TCP fairness goal: if **n** TCP sessions share same bottleneck link of bandwidth **R**, each should have average rate of ***R/n***



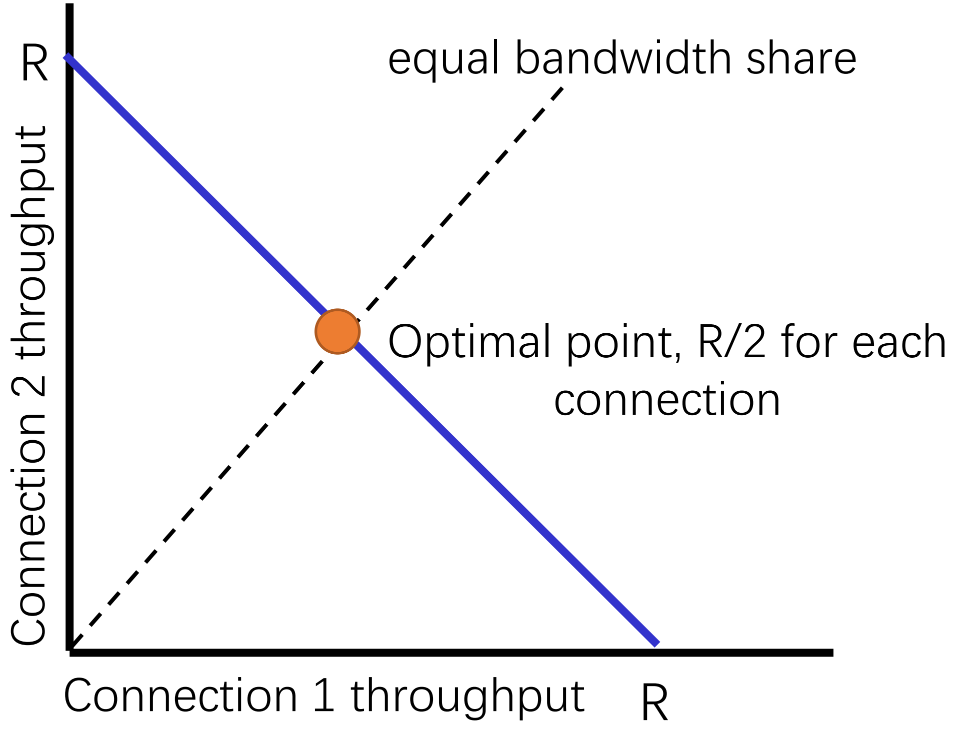
- Fairness Index
  - $f(x_1 \dots x_n) = \frac{(\sum x_i)^2}{n * \sum x_i^2}$

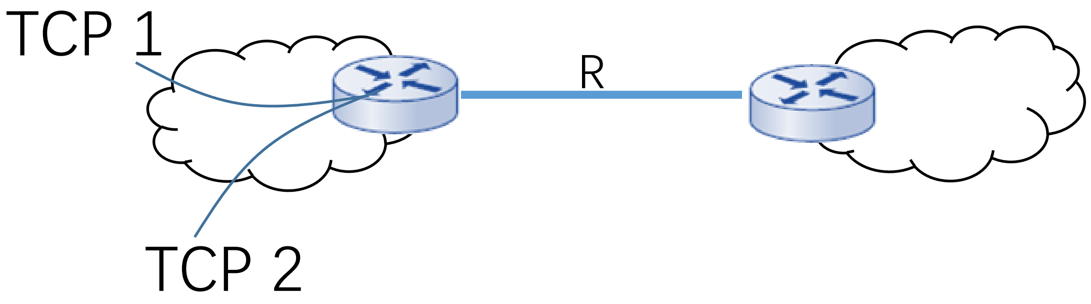
Fairness in TCP

- Consider the steady state, TCP uses a (linear) scheme to adjust its window cwnd
  - cwnd' = b*cwnd + a
- Possible Designs
  - Additive increase, additive decrease
  - Additive increase, multiplicative decrease (AIMD)
  - Multiplicative increase, additive decrease
  - Multiplicative increase, multiplicative decrease

Fairness in TCP

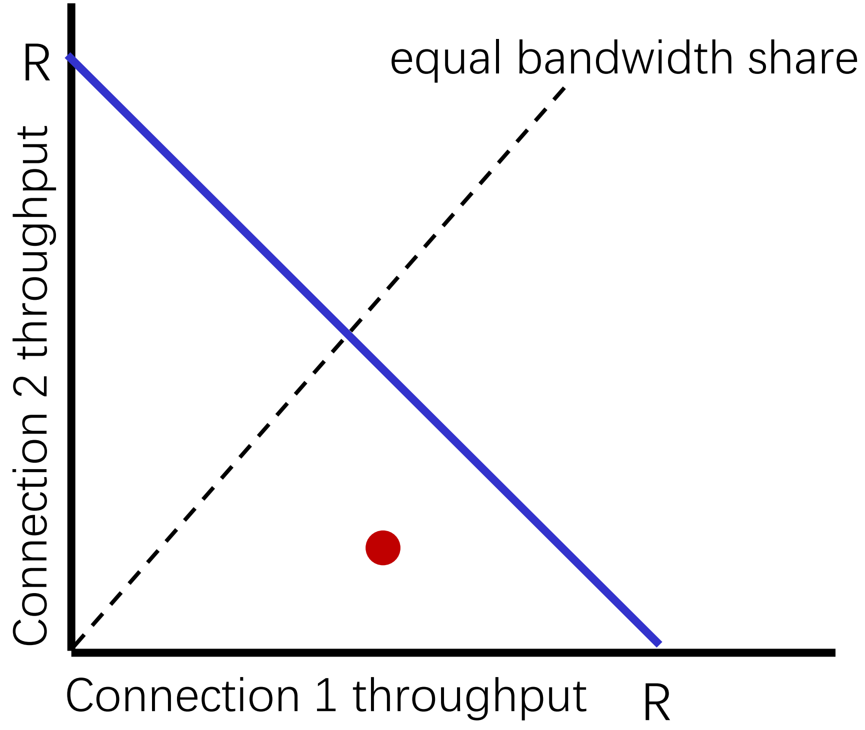
- Consider a case with two TCP connections





Fairness in TCP

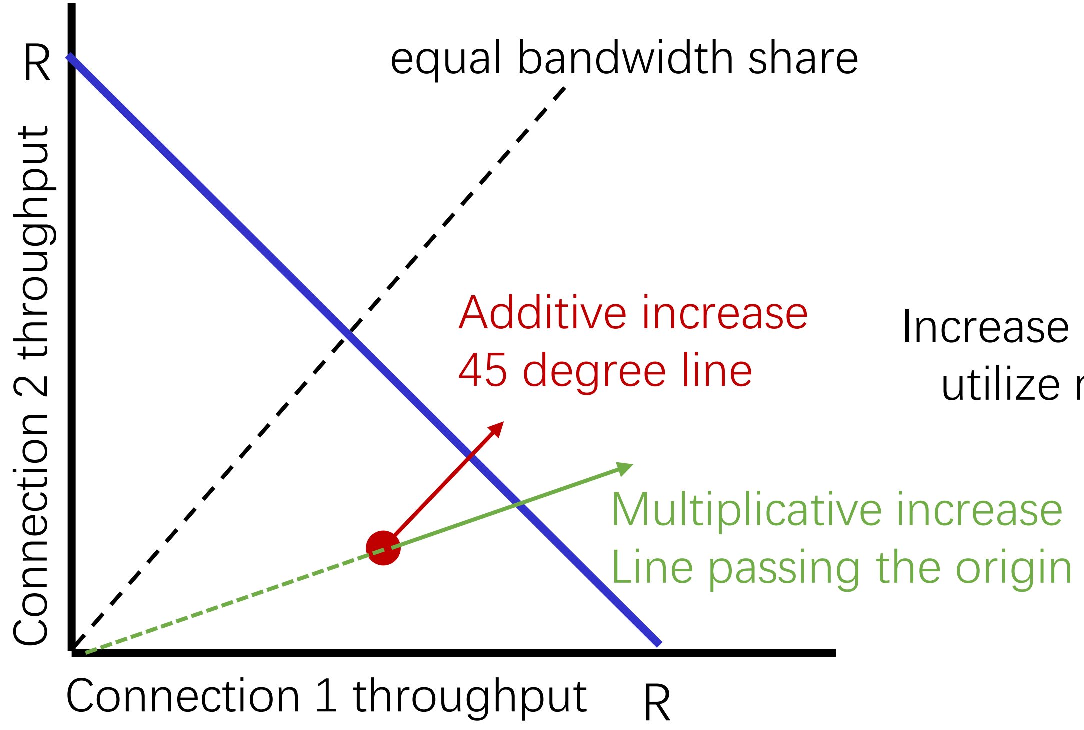
- Consider a case with two TCP connections



Increase the throughput to utilize more bandwidth

Fairness in TCP

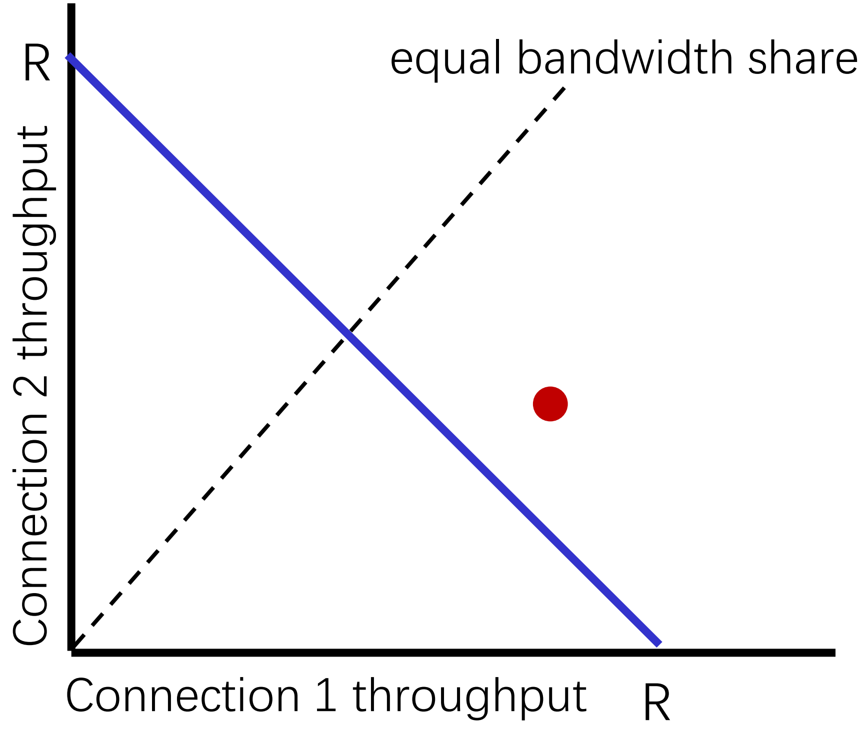
- Consider a case with two TCP connections



Increase the throughput to utilize more bandwidth

Fairness in TCP

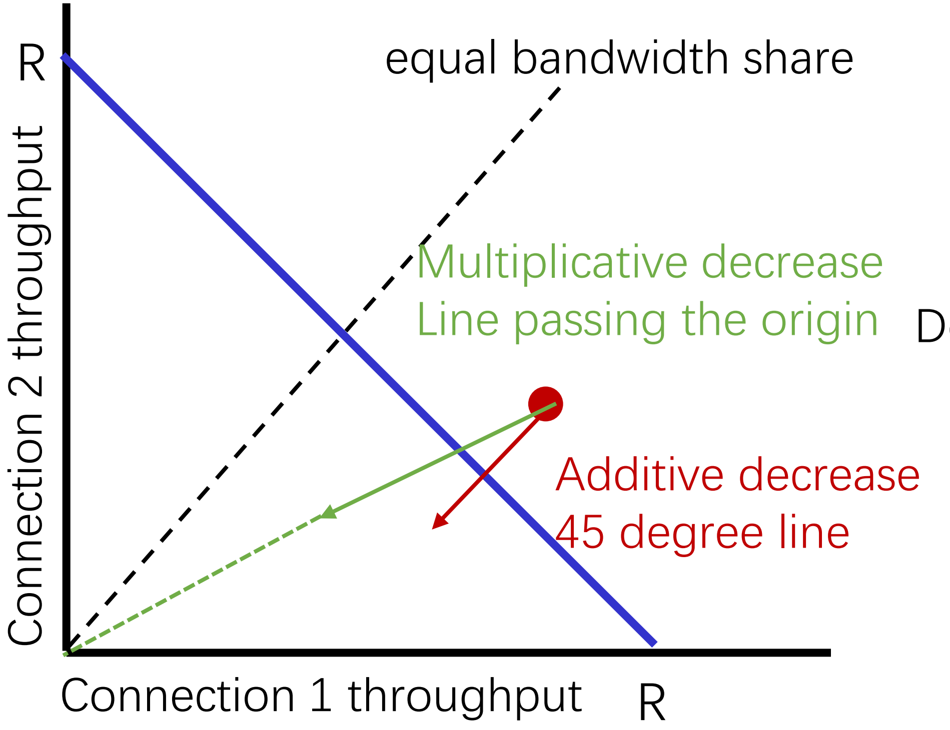
- Consider a case with two TCP connections



Decrease the throughput
to avoid congestion

Fairness in TCP

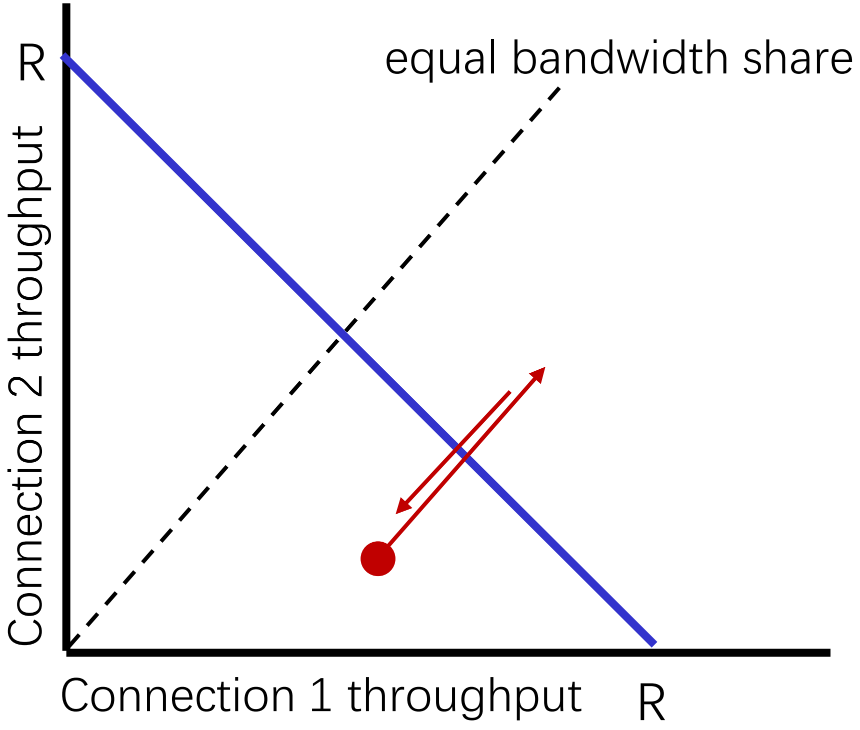
- Consider a case with two TCP connections



Decrease the throughput
to avoid congestion

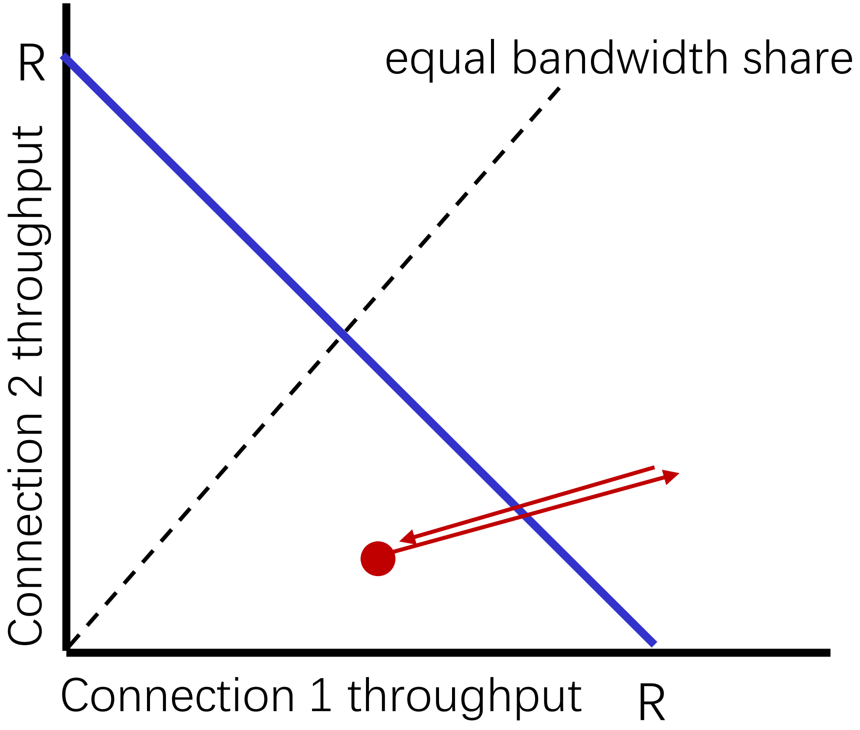
Fairness in TCP

- Consider a case with two TCP connections
  - Behavior of additive increase additive decrease
    - Stable but not fair



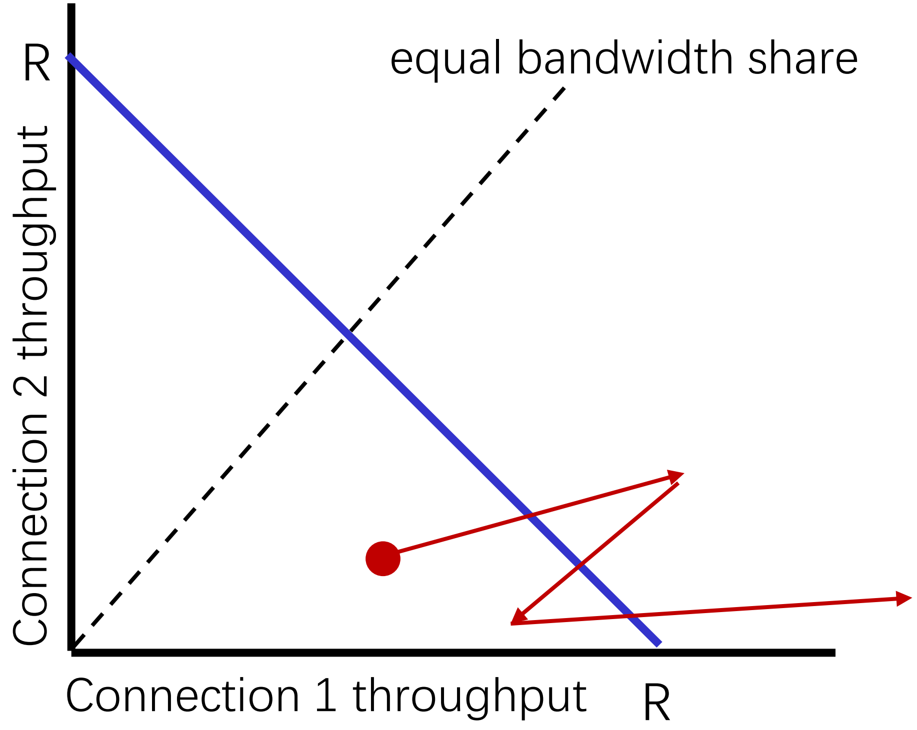
Fairness in TCP

- Consider a case with two TCP connections
  - Behavior of multiplicative increase multiplicative decrease
    - Stable but not fair



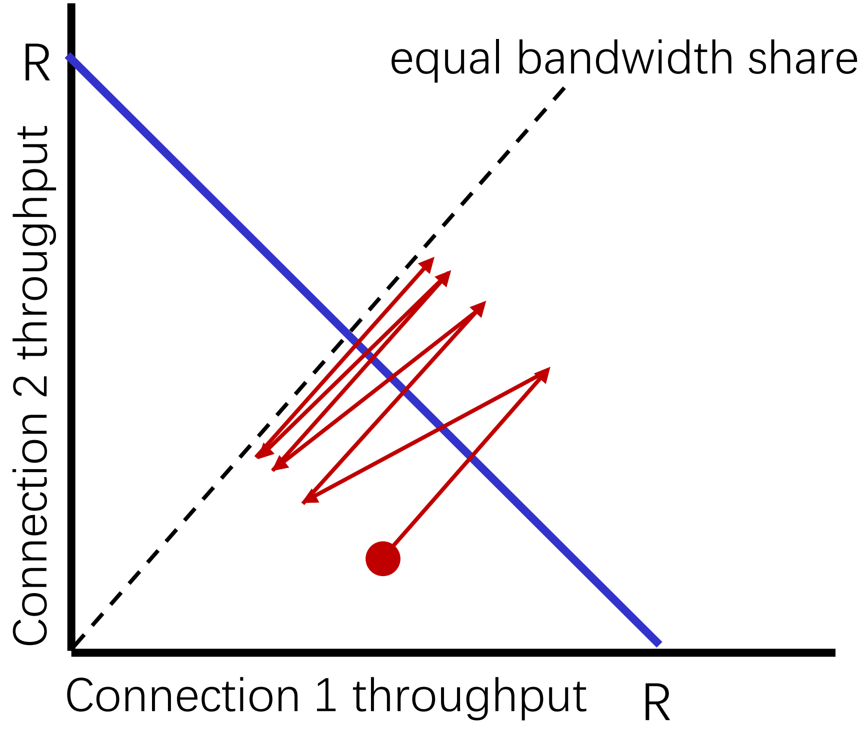
Fairness in TCP

- Consider a case with two TCP connections
  - Behavior of multiplicative increase additive decrease
    - Not stable



Fairness in TCP

- Consider a case with two TCP connections
  - Behavior of AIMD
    - Stable and fair



Fairness and RTT

- TCP connation with smaller RTT occupies more bandwidth
  - When congestion happens, they recover more quickly
    - TCP adjust cwnd in RTT basis

Fairness and Parallel TCP Connections

- Application can open multiple parallel connections between two hosts
  - web browsers do this , e.g., link of rate R with 9 existing connections:
    - new app asks for 1 TCP, gets rate R/10
    - new app asks for 11 TCPs, gets 11R/2o

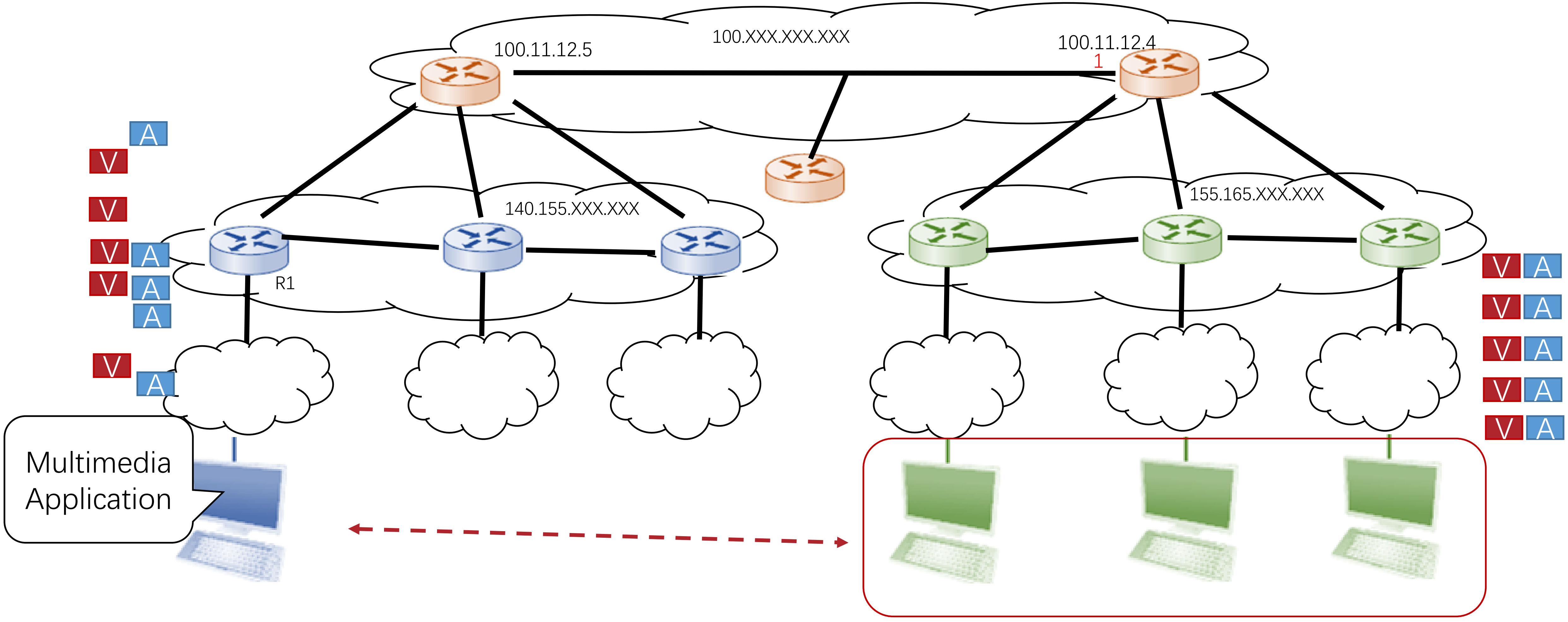
Fairness and UDP

- Some apps do not use TCP
  - do not want rate throttled by congestion control
- Instead, use UDP:
  - send audio/video at constant rate, tolerate packet loss
- There is no “Internet police” policing use of congestion control

Outline

- TCP Fairness
- ➢ QoS
- QUIC

Realtime Application

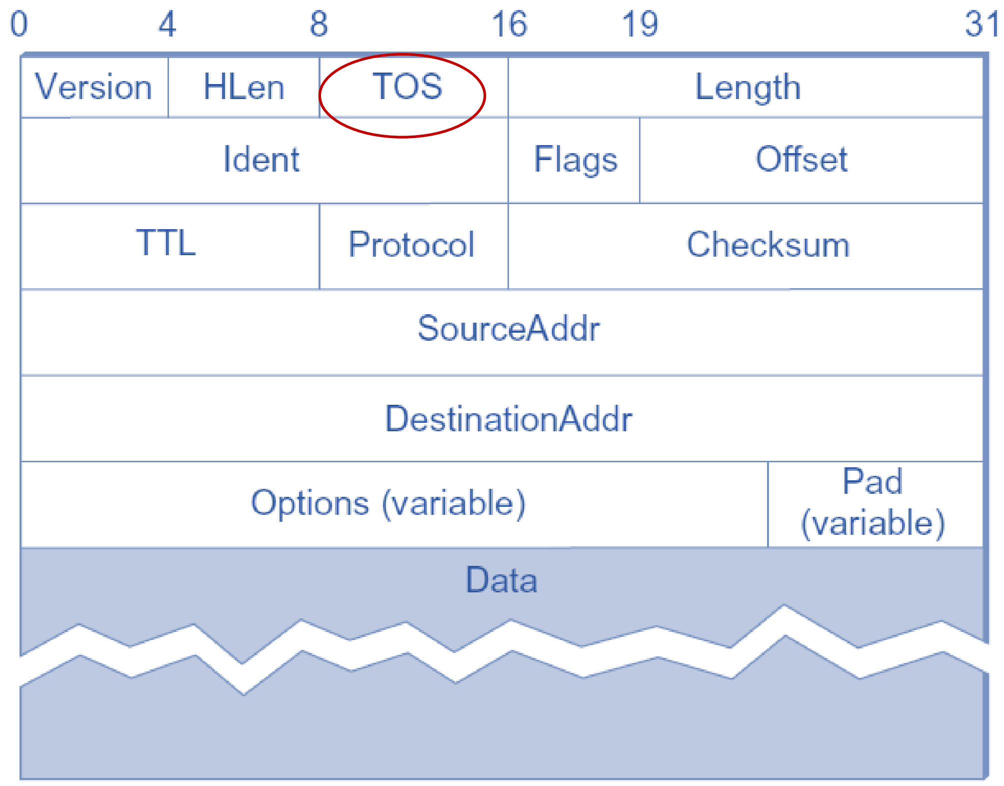


Packet Classification

- Classify Packets into Flows according to
  - Protocol
  - Source Address
  - Destination Address
  - Source Port
  - Destination Port
  - ToS

Differentiated Service Code Point

- Reuse ToS Field
  - 0-5bit: Differentiated Service Code Point (DSCP) Field
  - 6-7bit: Explicit Congestion Notification
- DSCP field encodes Per-Hop Behavior
  - Expedited Forwarding (all packets receive minimal delay & loss)
  - Assured Forwarding (packets marked with low/high drop probabilities)



Set Packet Class

- DSCP Field in Practice
  - Edge Routers
    - Set Differentiated Service (DS) Field in IP header
      - e.g., because of subscription
  - Core Routers
    - Implement Per Hop Behavior
      - According to DS Field of packets

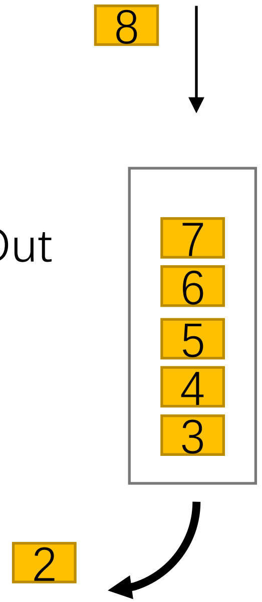
Commonly used DSCP values

DSCP value	Hex value	Decimal value	Meaning	Drop probability	Equivalent IP precedence value
101 110	0x2e	46	Expedited forwarding (EF)	N/A	101 Critical
000 000	0x00	0	Best effort	N/A	000 - Routine
001 010	0x0a	10	AF11	Low	001 - Priority
001 100	0x0c	12	AF12	Medium	001 - Priority
001 110	0x0e	14	AF13	High	001 - Priority
010 010	0x12	18	AF21	Low	010 - Immediate
010 100	0x14	20	AF22	Medium	010 - Immediate
010 110	0x16	22	AF23	High	010 - Immediate
011 010	0x1a	26	AF31	Low	011 - Flash
011 100	0x1c	28	AF32	Medium	011 - Flash
011 110	0x1e	30	AF33	High	011 - Flash
100 010	0x22	34	AF41	Low	100 - Flash override
100 100	0x24	36	AF42	Medium	100 - Flash override
100 110	0x26	38	AF43	High	100 - Flash override

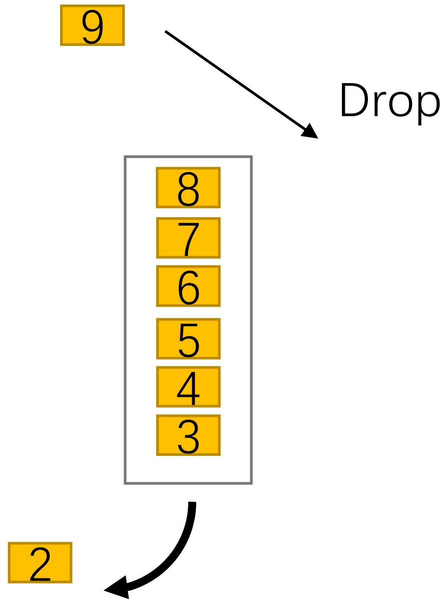
Queuing Discipline

- First-In-First-Out (FIFO)

First In First Out

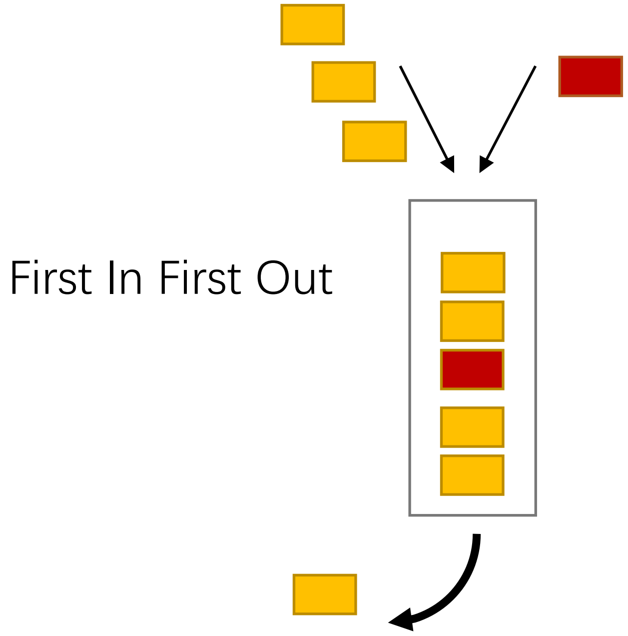


Drop Policy
Simplest one: Tail
Drop if FIFO is full



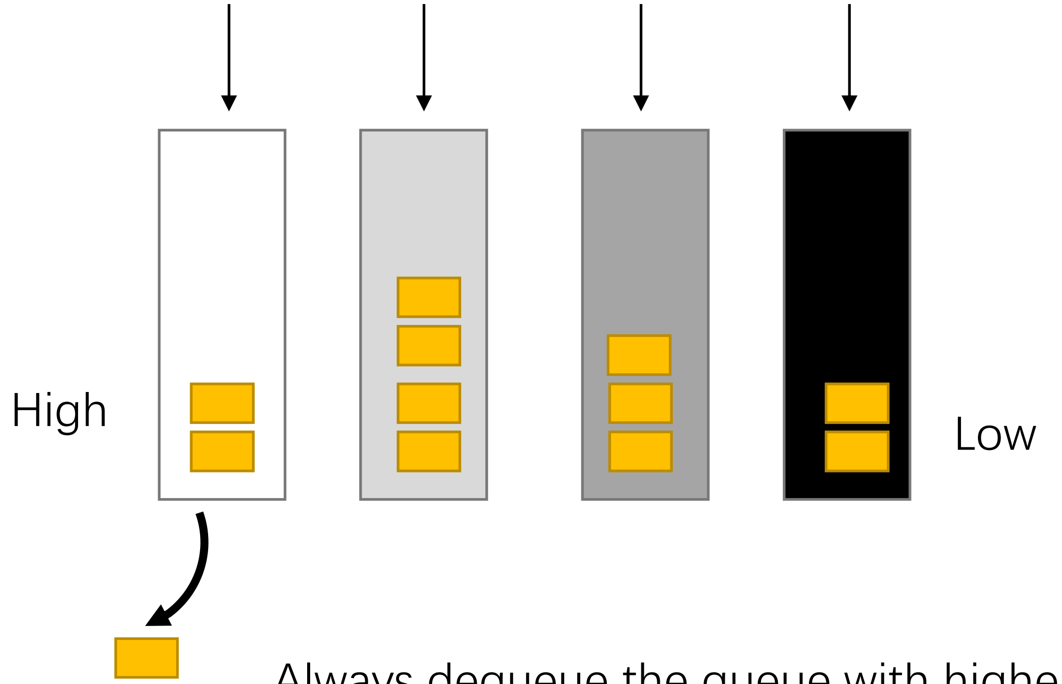
Queuing Discipline

- Problems in FIFO
  - Too simple to provide resource allocation polices
    - e.g., yellow src does not follow congestion control (UDP), can occupy more network source



Queuing Discipline

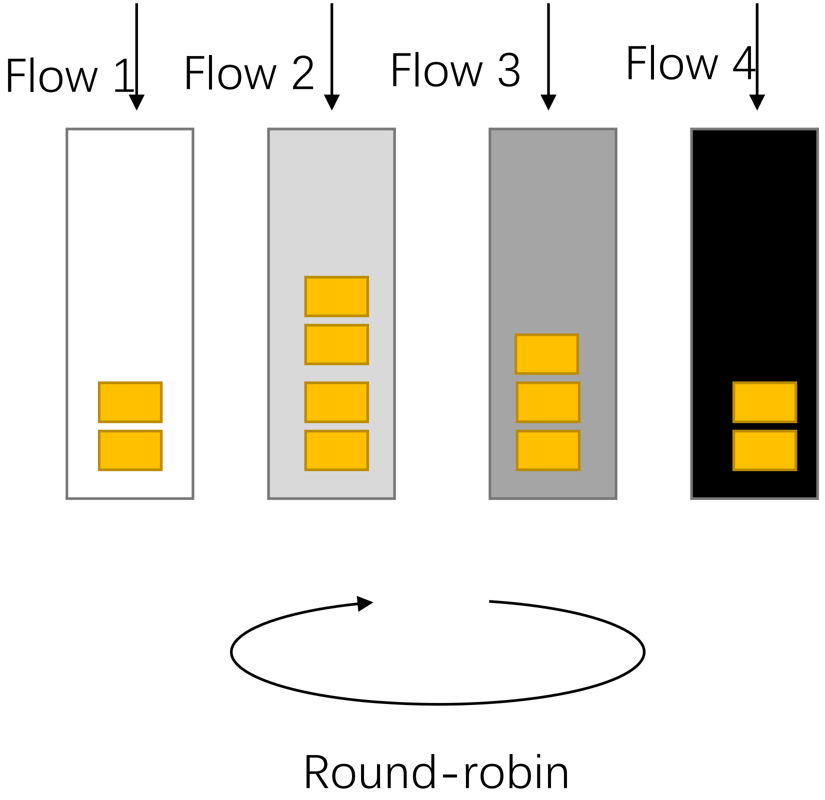
- First-In-First-Out (FIFO) with Priority



Always dequeue the queue with higher priority
unless it is empty
Problem: starvation

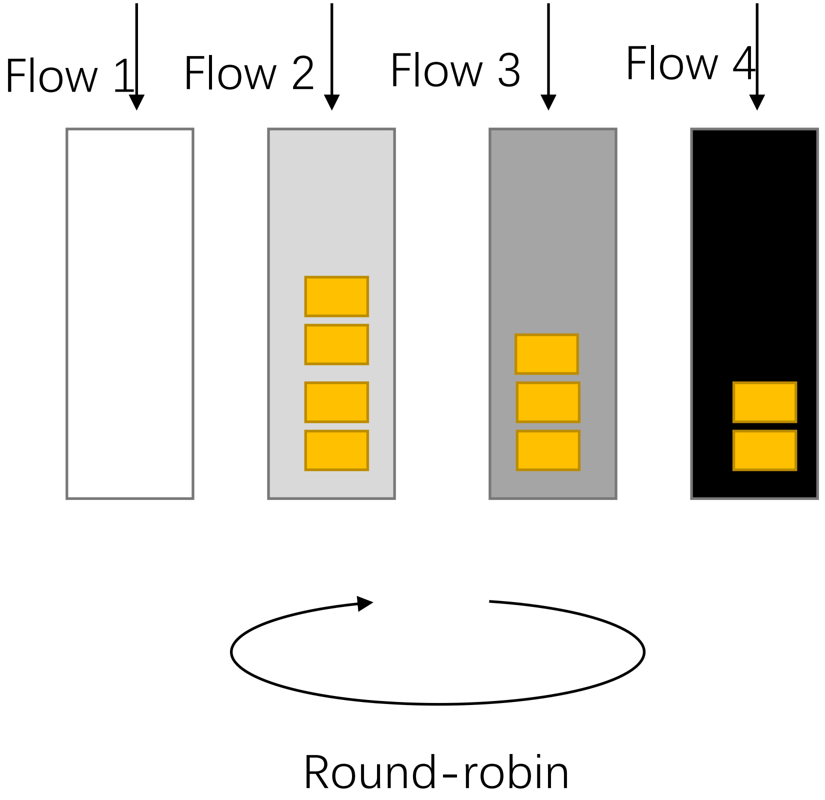
Queuing Discipline

- Fair Queuing (FQ)
  - Each flow gets 1/4 output bandwidth



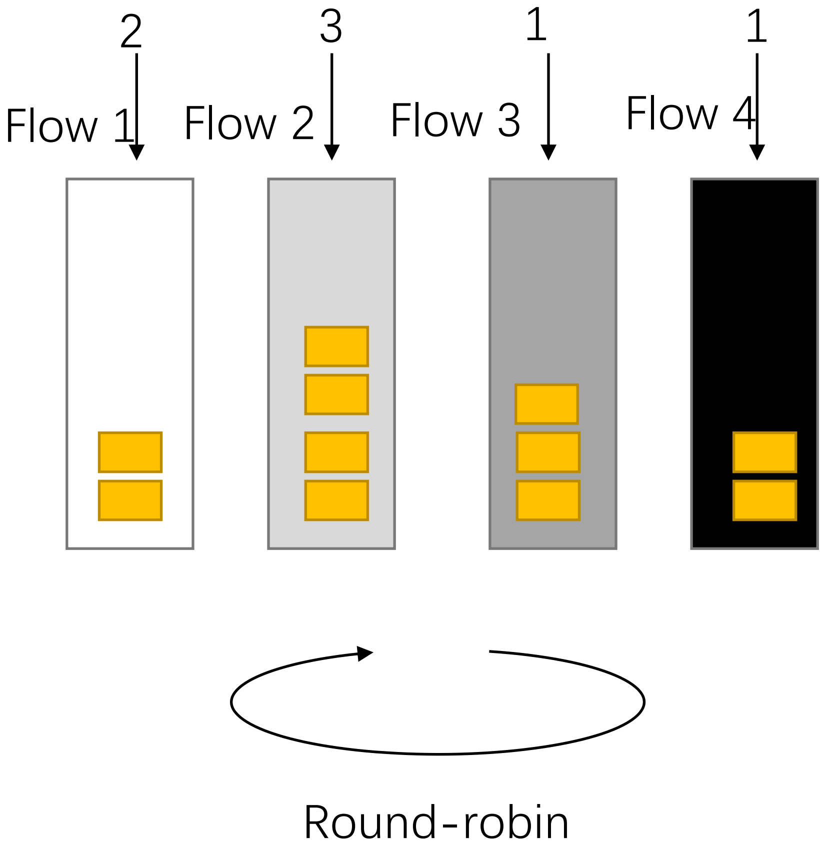
Queuing Discipline

- Fair Queuing (FQ)
  - Each flow gets 1/3 output bandwidth



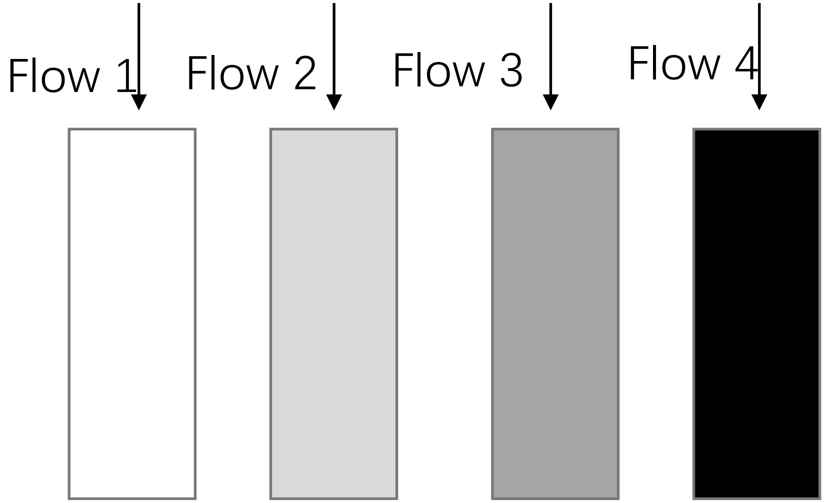
Queuing Discipline

- Weighted Fair Queuing
  - Flows with higher weight get more output bandwidth (2/7, 3/7, 1/7, 1/7)



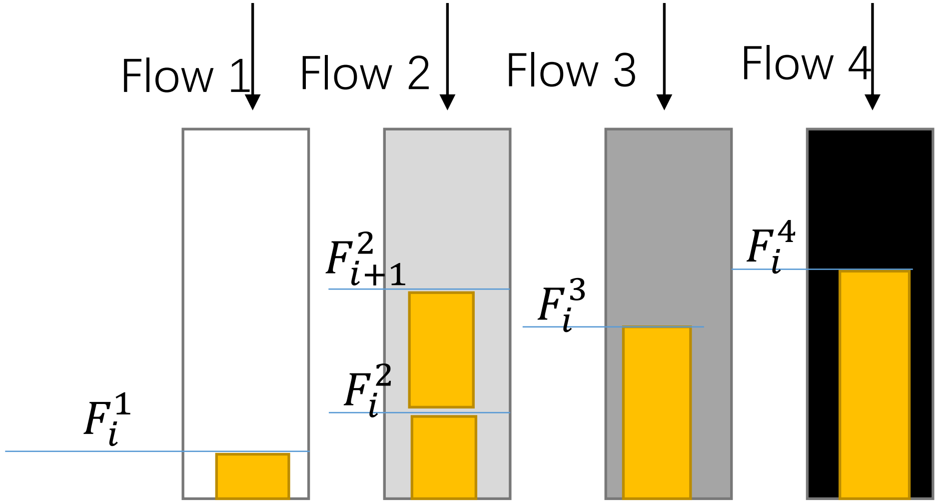
Queuing Discipline

- Bit-Level Fair Queuing
  - Schedule according to earliest finish time
  - Finish time of packet i: $F_i = \max(F_{i-1}, A_i) + P_i$
  - P_i is the transmitting duration of packet i, A_i is the arriving time of packet i, F_{i-1} is the finish time of packet i-1 of the same flow



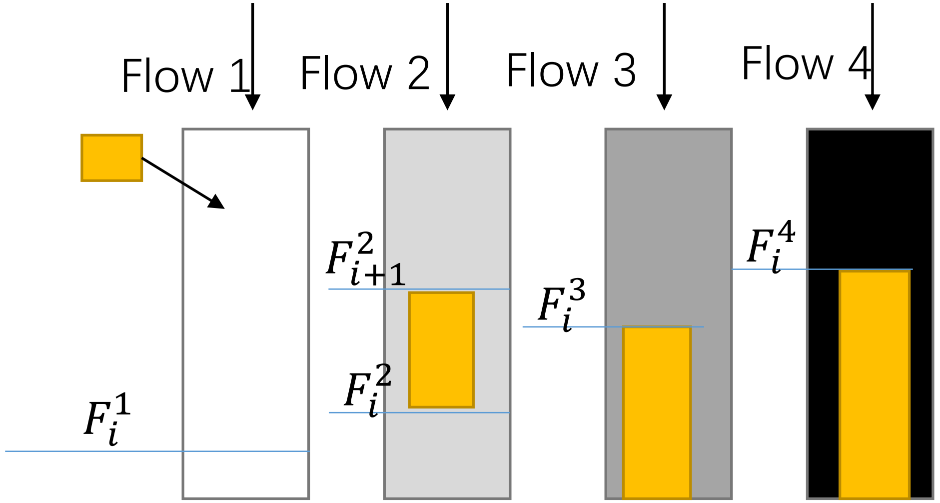
Queuing Discipline

- Bit-Level Fair Queuing
  - <u>Schedule according to earliest finish time</u>
  - Finish time of packet i: $F_i = \max(F_{i-1}, A_i) + P_i$
  - P_i is the transmitting duration of packet i, A_i is the arriving time of packet i, F_{i-1} is the finish time of packet i-1 of the same flow
  - Assume A_i=0, the finish time is virtually calculated in bit-level
  - Packets are dequeued according to the order: $F_i^1 < F_i^2 < F_i^3 < F_{i+1}^2 < F_i^4$



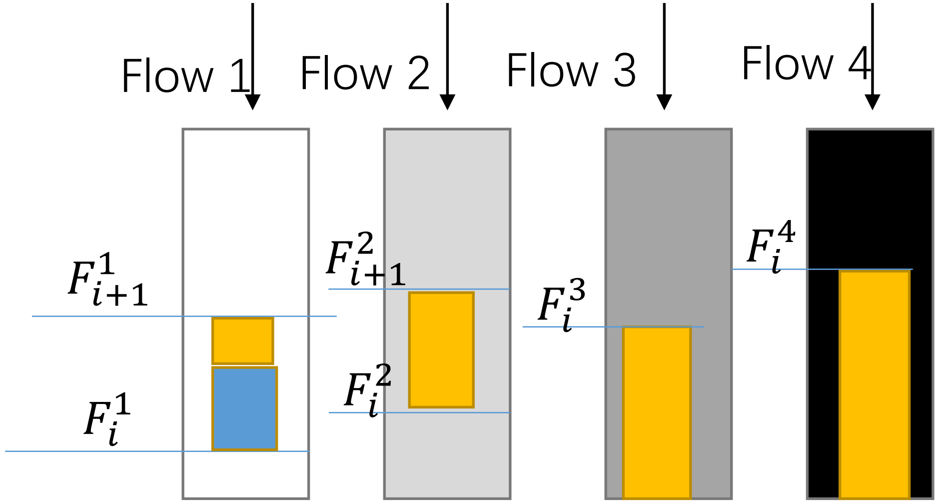
Queuing Discipline

- Bit-Level Fair Queuing (FQ)
  - Schedule according to earliest finish time
  - Finish time of packet i: $F_i = \max(F_{i-1}, A_i) + P_i$
  - P_i is the transmitting duration of packet i, A_i is the arriving time of packet i, F_{i-1} is the finish time of packet i-1 of the same flow
  - Case: when one queue is idle, idle time also count



Queuing Discipline

- Bit-Level Fair Queuing (FQ)
  - Schedule according to earliest finish time
  - Finish time of packet i: $F_i = \max(F_{i-1}, A_i) + P_i$
  - P_i is the transmitting duration of packet i, A_i is the arriving time of packet i, F_{i-1} is the finish time of packet i-1 of the same flow
  - Case: when one queue is idle, idle time also count



Outline

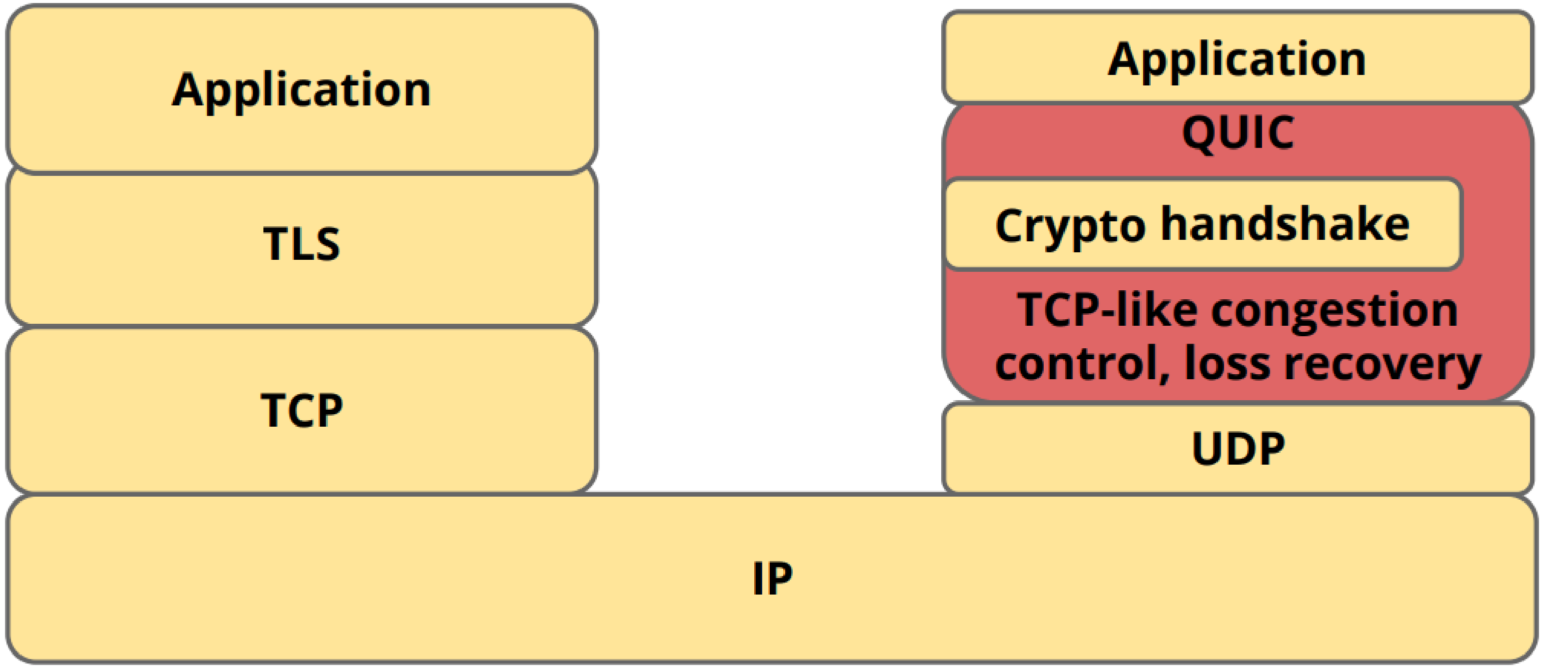
- TCP Fairness
- QoS
- ➢QUIC

QUIC

- QUIC: Quick UDP Internet Connections
- Application-layer protocol, on top of UDP
  - Deployed by Google staring at 2014
    - Deployed on many Google servers, apps (Chrome, mobile YouTube app)
  - QUIC working group formed in Oct 2016
- Initial goal: increase performance of HTTP

QUIC

- Protocol Stack

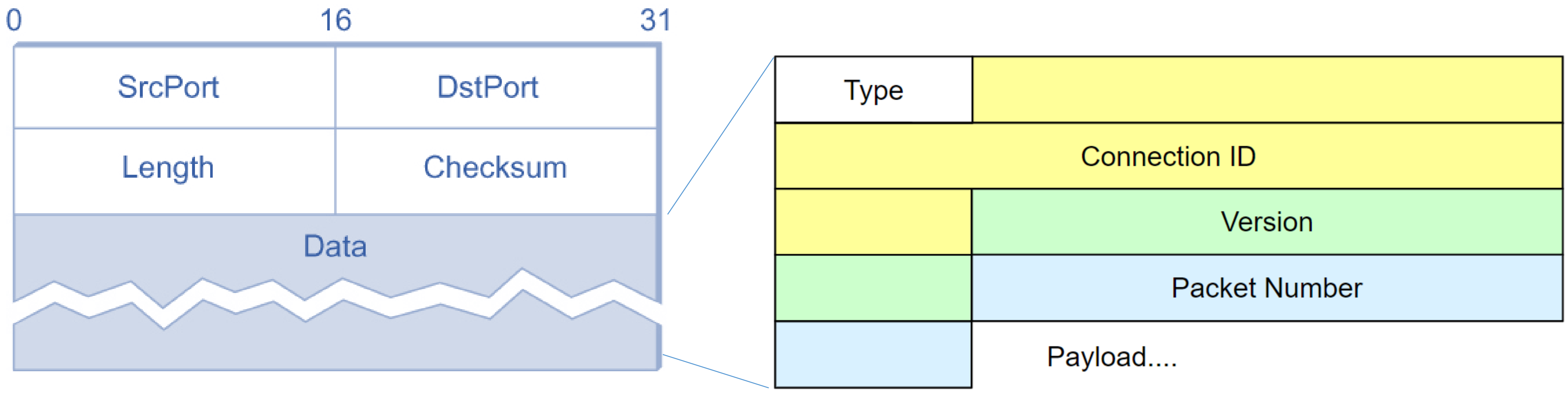


https://www.ietf.org/proceedings/98/slides/slides-98-edu-sessf-quic-tutorial-00.pdf

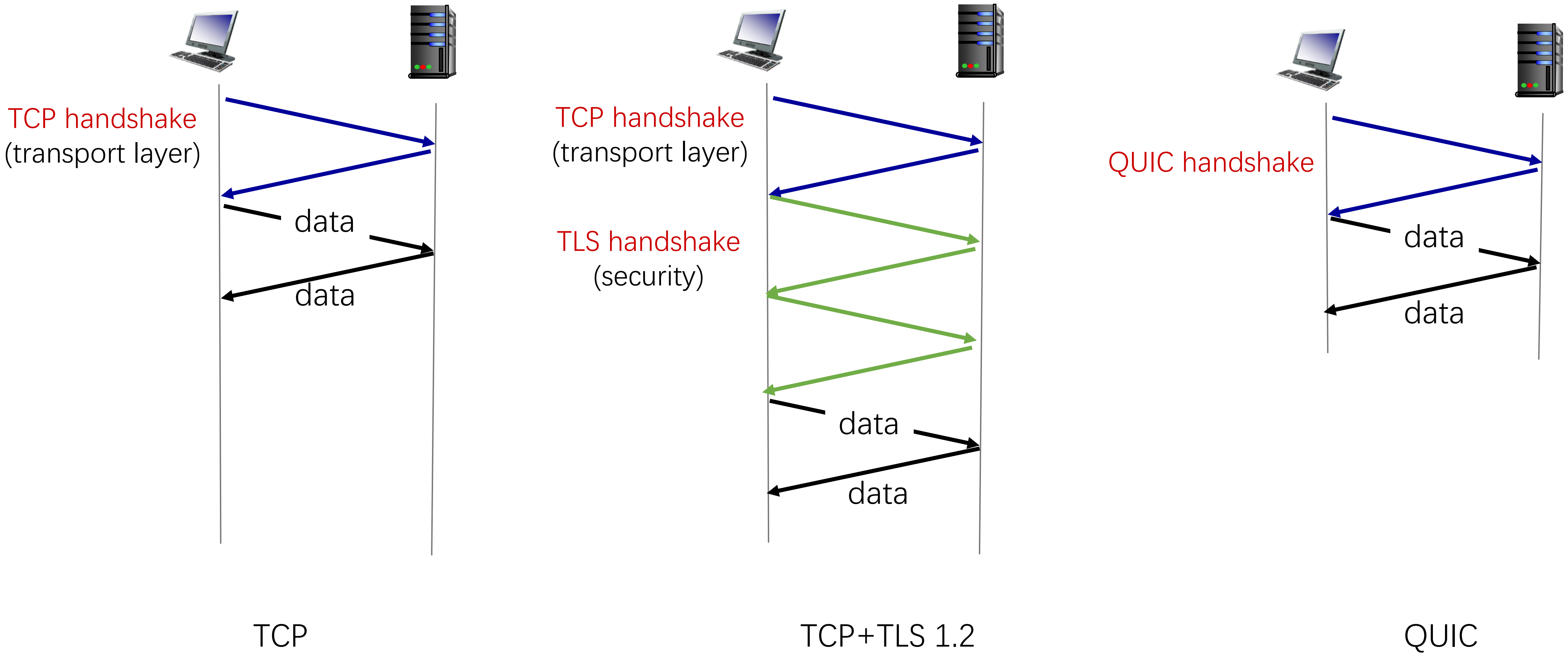
QUIC

- Key features
  - Always encrypted
  - 0-RTT connection establishment
  - Connection migration
  - Congestion control
  - Parallel Streams

QUIC - Header

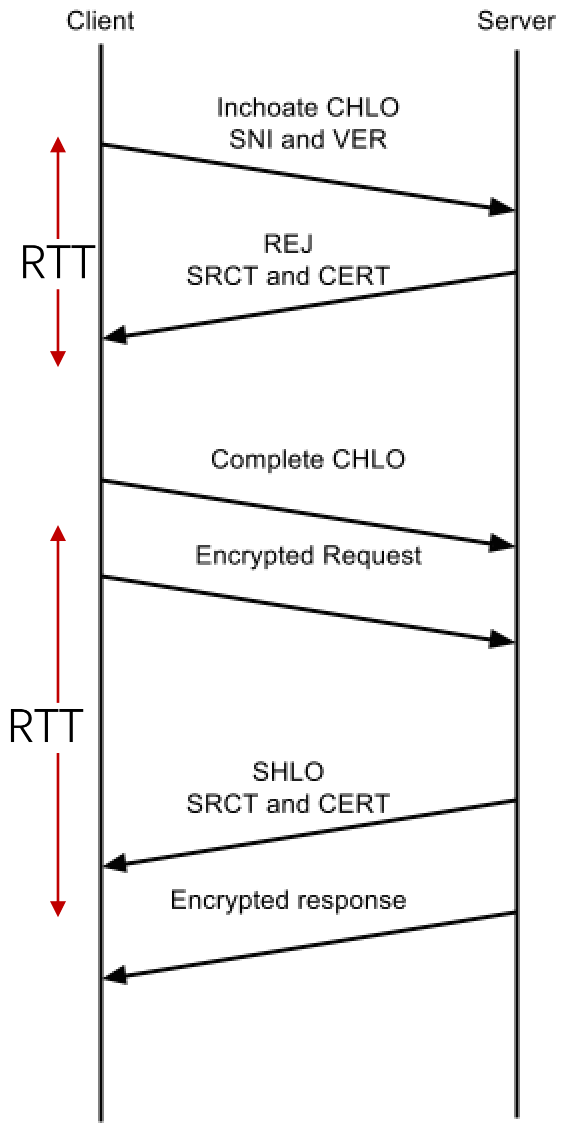

0
16
31
SrcPort
DstPort
Length
Checksum
Data
Type
Connection ID
Version
Packet Number
Payload....

QUIC Connection Establishment



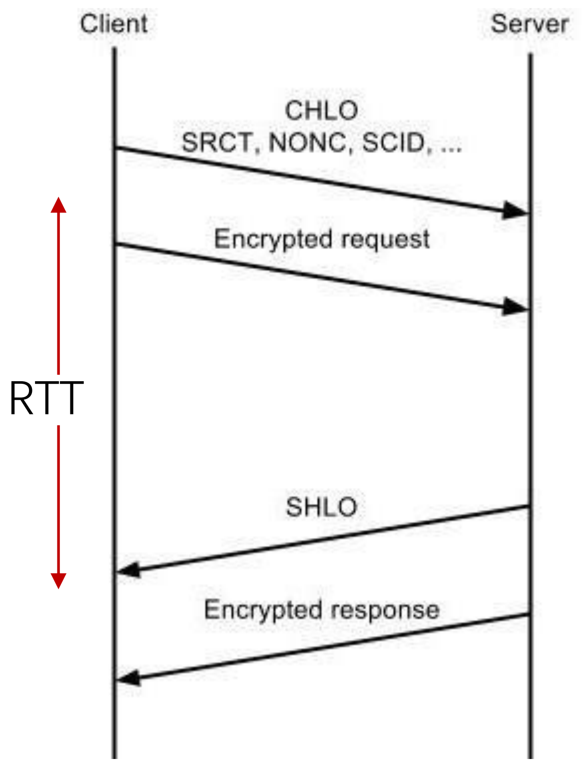
QUIC Connection Establishment

- 1-RTT (First-ever connection)
  - No cached information available
  - First CHLO is inchoate (empty)
    - Simply includes version and server name
  - Server responds with REJ
    - Includes server config, certs, etc.
    - Allows client to make forward progress
  - Second CHLO is complete
    - Followed by initially encrypted request data
  - Server responds with SHLO
    - Followed immediately by forward-secure encrypted response data



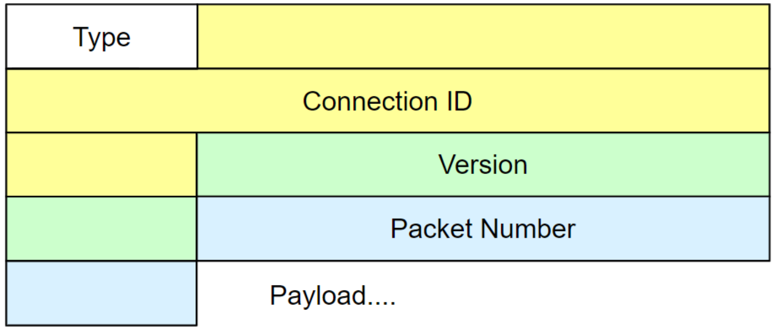
QUIC Connection Establishment

- 0-RTT (Subsequent connection)
  - Motivation: client can cache information about the *origin* it connected to
  - First CHLO is complete
    - Based on information from previous connection
    - Followed by initially encrypted data.
  - Server responds with SHLO
    - Followed immediately by forward-secure encrypted data



QUIC Connection Migration

- NAT Rebinding
  - NATs remaps port
    - Frequency (~ mins)
    - Why ? to release unused ports
      - According to TCP connection state (if they are closed)
    - UDP does not have connection state, QUIC state is encrypted
- Mobility
  - Switching between different IP
    - Wi-Fi and cellular network
- Connection Migration
  - Keep QUIC connections alive even if port and IP are changed
  - Detect connection path changes via Connection ID and IP/port
    - Connection is identified by connection ID rather than <IP, port>
    - 64-bit connection ID
    - randomly chosen by client



QUIC Congestion Control

- Incorporates TCP best practices
  - TCP Cubic, Fast Retransmission, Selective ACK, etc.
- Better signaling than TCP
  - Each packet carries a monotonically increasing packet number
    - Better RTT measurement
  - Retransmitted packets also consume new sequence numbers
    - no retransmission ambiguity
- More verbose ACK
  - support 256 Selective ACK ranges (vs. TCP's 3 Selective ACK ranges)

QUIC - Parallel Streams

- Handle HOL blocking

Reference

- Textbook 6.5
- Some slides are adapted from http://www-net.cs.umass.edu/kurose_ross/ppt.htm by Kurose Ross
- https://www.chromium.org/quic