Transport layer: overview

- Transport services and protocols
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Connection-oriented transport: TCP
- TCP congestion control
- Evolution of transport-layer functionality

TCP congestion control

- reliable data transfer function: retransmission of lost segments
- network congestion can be the root cause of loss, reliability is not a mitigation
- mechanisms are needed to throttle senders, i.e. having sender limits
    * if sender perceives that there is little congestion, then the sender increases its send rate
    * if the sender perceives that there is congestion along the path, then it reduces its send rate

- how does a TCP sender limit its sending rate?
- how does a TCP sender perceive a congestion?
- what algorithm is used to change the sending rate?

TCP congestion control

- how does a TCP sender limit its sending rate?

- flow control: receive buffer, send buffer, and some variables
- an additional variable, the congestion window (cwnd)
- imposes a constraint on the rate at which a TCP sender can send traffic
- amount of unacknowledged data at a sender may not exceed the minimum of cwnd and rwnd.
- cwnd limits the amount of unacknowledged data at the sender and therefore limits the sender's send rate
- cnwd/RTT: sender rate

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

TCP congestion control

- how does a TCP sender perceive a congestion?

- when there is excessive congestion, then router buffers along the path overflows and this causes a TCP segment to be dropped

- dropped segment results in a loss event at the sender (timeout or receipt of 3 duplicate ACKs): indications of congestion

on the path

- when a loss event doesn't occur, sender will use ACKs to increase its cwnd size (and so its transmission rate)

- if ACKs arrive at a relatively slow rate then cwnd will be increased at a relatively slow rate, vice versa

- TCP uses ACKs to trigger or clock its increase in cwnd size, so TCP is said to be self-clocking

TCP congestion control

- what algorithm is used to change the sending rate?

or how do the TCP senders determine their sending rates such that they don't congest the network but at the same time make use of all the available bandwidth?

principles:

- A lost segment implies congestion: TCP sender's rate should be decreased when a segment is lost:

* a timeout event or the receipt of 4 ACKs (1 original + 3 duplicate) means "loss event"

* sender's rate can be increased when an ACK arrives (for a previously UNACKed segment)

- Bandwidth probing:

* TCP sender increases its transmission rate to probe for the rate at which congestion begins

* Backs off from that rate and begins probing again (to see if it has changed or not)

* no loss or signals of congestion: increase the sending rate; when loss occurs: stop increasing and start decreasing the rate; again start increasing the rate and determine the congestion rate

TCP congestion control: details

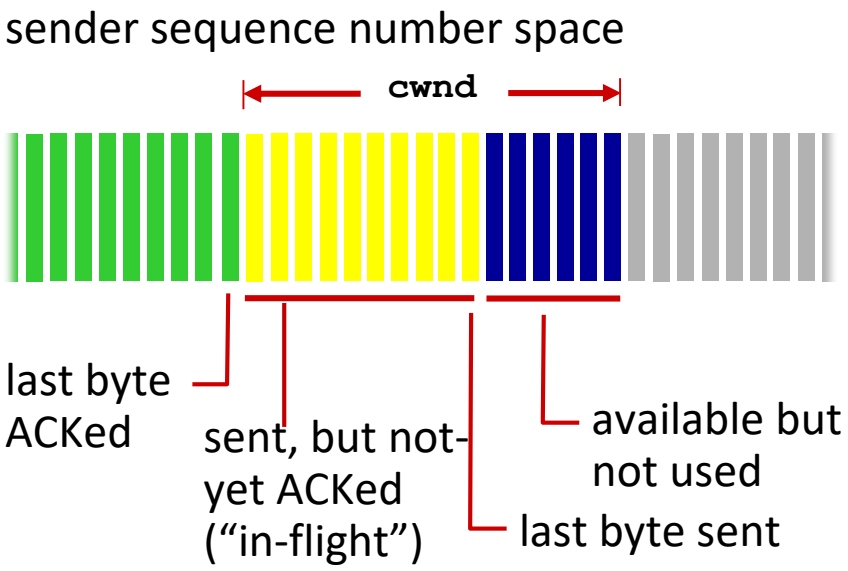
TCP congestion-control algorithm to change the sending rate, 3 components:

- slow start
- congestion avoidance
- fast recovery

1-2 mandatory, 3 is recommended

TCP congestion control: details

cwnd is 1 MSS initially, initial sending rate: MSS/RTT



TCP sending behavior:

- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

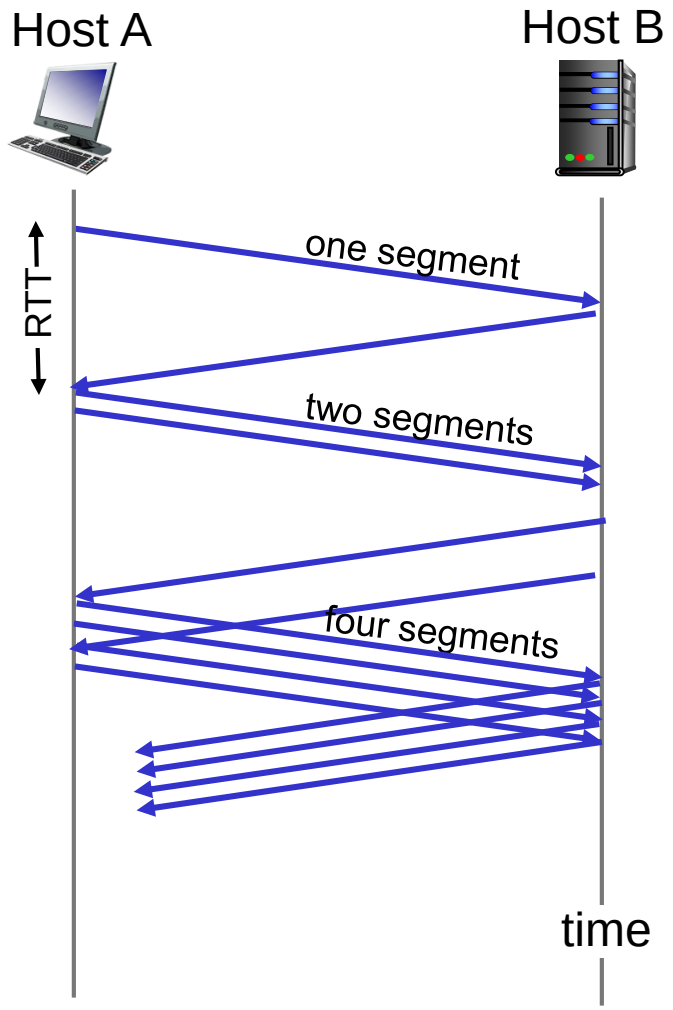
MSS=500 bytes & RTT=200 msec
I.S.R. = 500/200 = 20 kbps

- TCP sender limits transmission: `LastByteSent- LastByteAcked ≤ cwnd`
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT (double every time a transmitted segment is ACKed)
  - done by incrementing **`cwnd`** for every ACK received

- *summary:* initial rate is slow, but ramps up exponentially fast



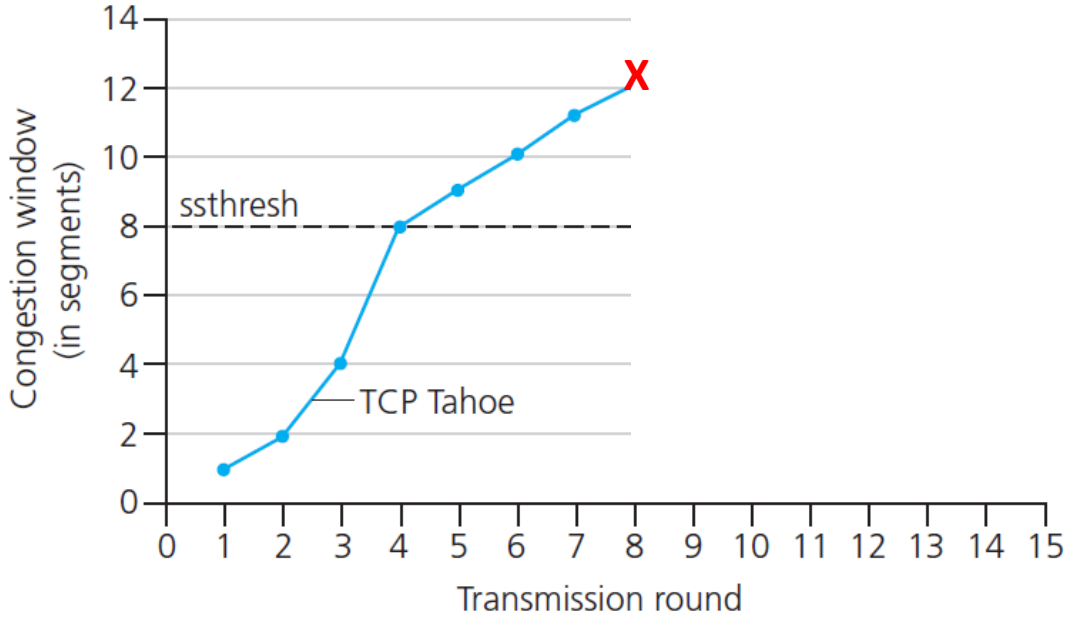
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to 1/2 of its value before timeout.

Implementation:

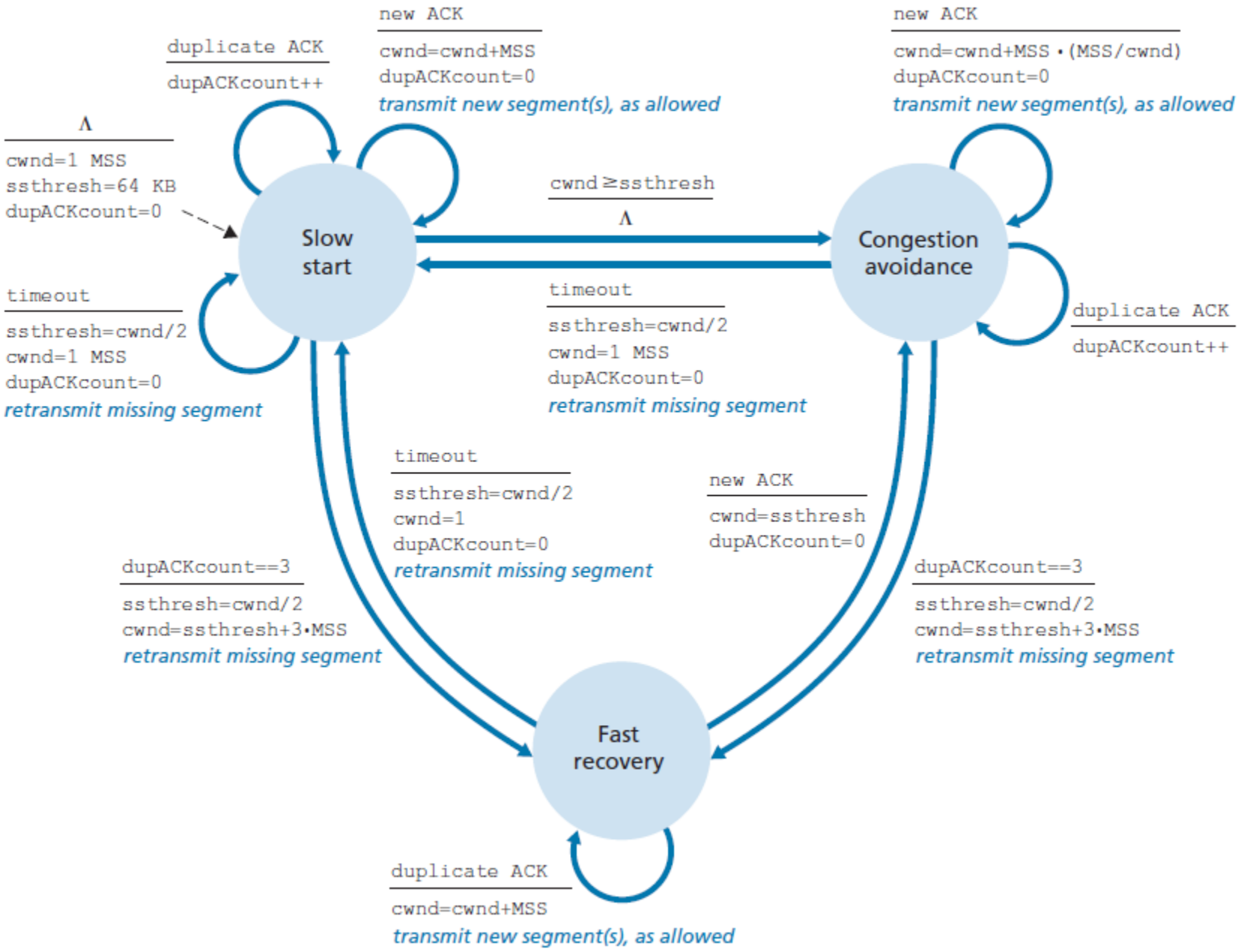
- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event



cwnd is increased exponentially till it arrives at the value of ssthresh: 8, a loss occured for the previous segments when cwnd was 16, ssthresh was set to 8 beforehand

set cwnd to 1 MSS and continue slow start after the congestion, and when we arrive at ssthresh then we start to increase cwnd a single MSS

for 3 duplicate ACKs it is time for fast recovery



The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

TCP congestion control: AIMD

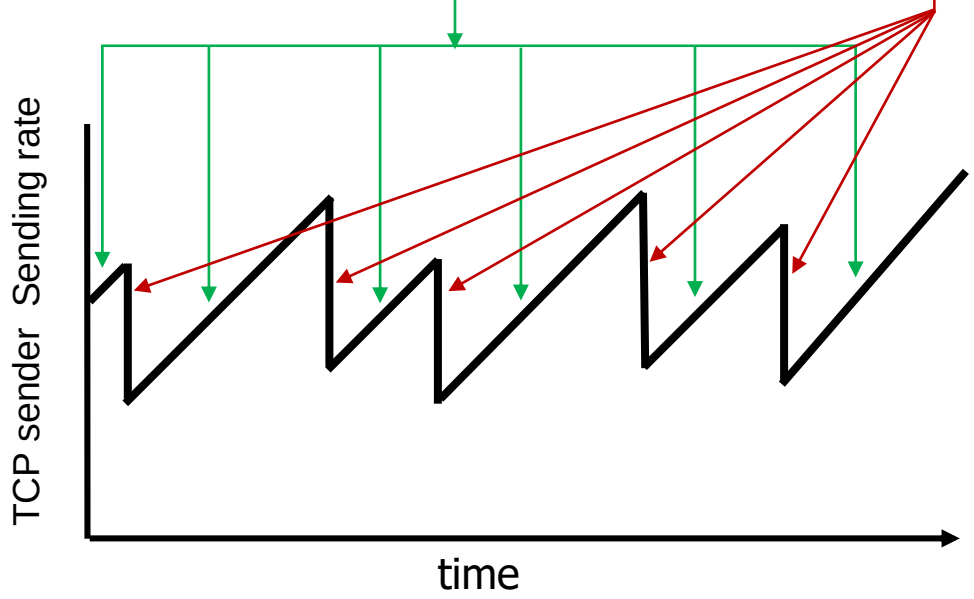
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

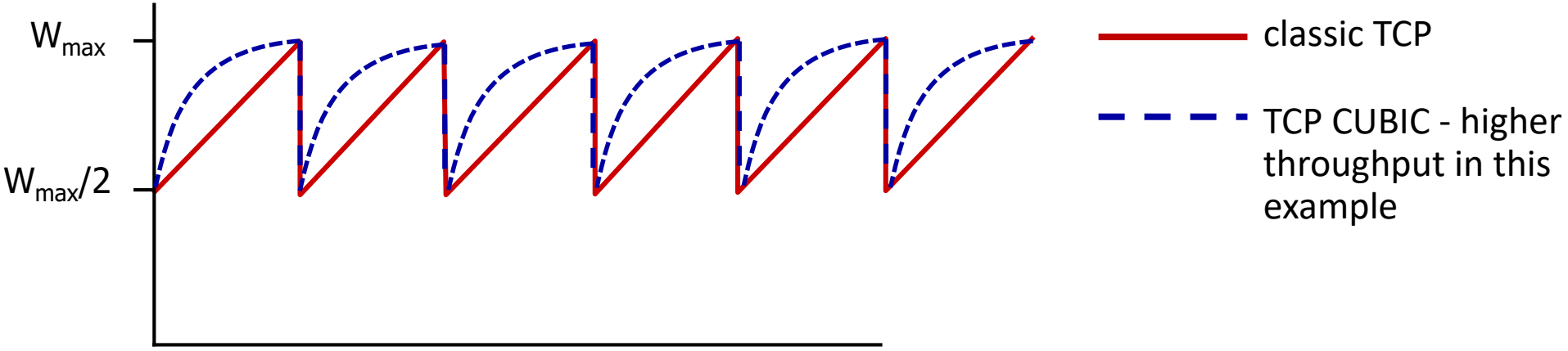
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP CUBIC: changes the avoidance way

- usual way: decrease the sending rate and then increase it slowly when congestion occurs
- it may be better to more quickly ramp up to get close to the pre-loss sending rate
- W_{max}: sending rate at which congestion loss was detected
- K: point in time (future) when TCP window size will reach W_{max}
- t: current time
- after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*
- *increases the congestion window as a function of cube of the distance between the current time, t, and K*



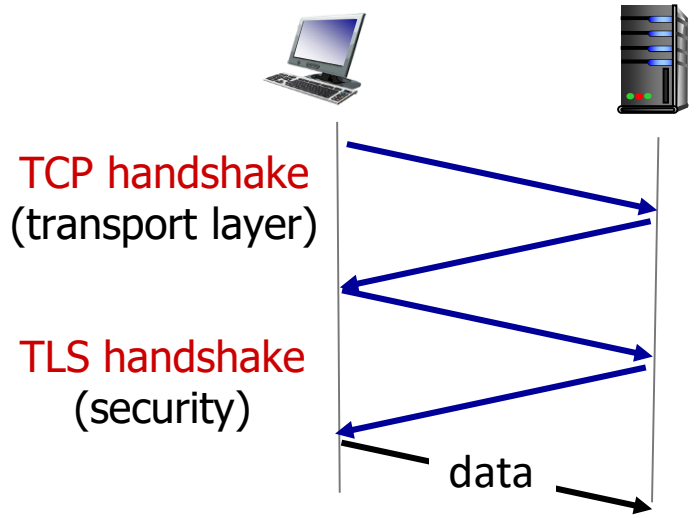
Transport layer: roadmap

- Transport services and protocols
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Connection-oriented transport: TCP
- TCP congestion control
- Evolution of transport-layer functionality

QUIC: Quick UDP Internet Connections

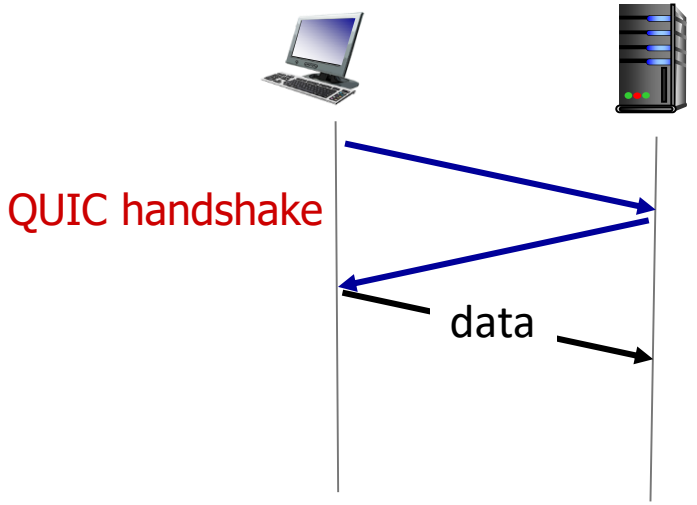
- application-layer protocol, on top of UDP
  - increase performance of secure HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)
  - uses UDP as its underlying transport-layer protocol, adopts approaches for connection establishment, error control, and congestion control

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake