TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\text{timeout} \quad = \quad \text{RTT}_{est} \quad + \quad 4 \times \text{var}_{est}$$

TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\text{timeout} \quad = \quad \text{RTT}_{est} \quad + \quad 4 \times \text{var}_{est}$$

- RTT_{est} is an exponentially weighted moving average:

$$\text{RTT}_{est} \quad = \quad (1\text{-}\alpha) \times \text{RTT}_{est} \quad + \quad \alpha \times \text{RTT}_{measured}$$

TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\text{timeout} \quad = \quad \text{RTT}_{est} \quad + \quad 4 \times \text{var}_{est}$$

- RTT_{est} is an exponentially weighted moving average:

$$\text{RTT}_{est} \quad = \quad (1\text{-}\alpha) \times \text{RTT}_{est} \quad + \quad \alpha \times \text{RTT}_{measured}$$

TCP uses $\alpha = \frac{1}{8}$

TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\text{timeout} \quad = \quad \text{RTT}_{est} \quad + \quad 4 \times \text{var}_{est}$$

- RTT_{est} is an exponentially weighted moving average:

$$\text{RTT}_{est} \quad = \quad (1-\alpha) \times \text{RTT}_{est} \quad + \quad \alpha \times \text{RTT}_{measured}$$

TCP uses $\alpha = \frac{1}{8}$

- var_{est} is similarly weighted:

$$\text{var}_{est} \quad = \quad (1-\beta) \times \text{var}_{est} \quad + \quad \beta \times |\text{RTT}_{measured} - \text{RTT}_{est}|$$

TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\text{timeout} \quad = \quad \text{RTT}_{est} \quad + \quad 4 \times \text{var}_{est}$$

- RTT_{est} is an exponentially weighted moving average:

$$\text{RTT}_{est} \quad = \quad (1\text{-}\alpha) \times \text{RTT}_{est} \quad + \quad \alpha \times \text{RTT}_{measured}$$

TCP uses $\alpha = \frac{1}{8}$

- var_{est} is similarly weighted:

$$\text{var}_{est} \quad = \quad (1\text{-}\beta) \times \text{var}_{est} \quad + \quad \beta \times |\text{RTT}_{measured} - \text{RTT}_{est}|$$

TCP uses $\beta = \frac{1}{4}$

TCP Congestion Control

Flow control relies on `rwnd` and `window` ≤ `rwnd`

TCP Congestion Control

from TCP header

Flow control relies on `rwnd` and `window` ≤ `rwnd`

TCP Congestion Control

Flow control relies on `rwnd` and `window` ≤ `rwnd`

Congestion control uses `cwnd` and `window` ≤ `cwnd`

TCP Congestion Control

Flow control relies on `rwnd` and `window` ≤ `rwnd`

Congestion control uses `cwnd` and `window` ≤ `cwnd`

- start `cwnd` at maximum segment size, `MSS`
- grow until `ssthresh` , which starts large, but can adapt

TCP Congestion Control

Flow control relies on `rwnd` and `window` ≤ `rwnd`

Congestion control uses `cwnd` and `window` ≤ `cwnd`

- start `cwnd` at maximum segment size, `MSS` about 1.5KB
- grow until `ssthresh` , which starts large, but can adapt

TCP Congestion Control

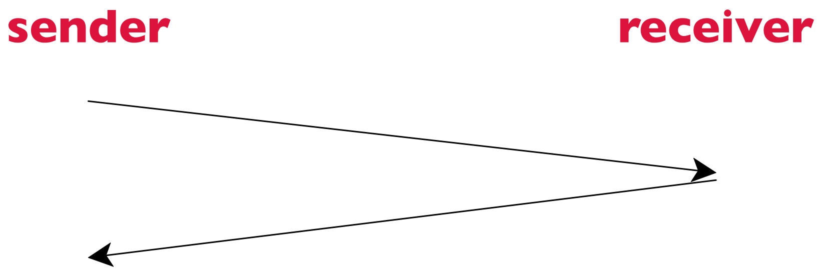
Flow control relies on `rwnd` and `window` ≤ `rwnd`

Congestion control uses `cwnd` and `window` ≤ `cwnd`

- start `cwnd` at maximum segment size, `MSS`
- grow until `ssthresh` , which starts large, but can adapt

start at 64KB

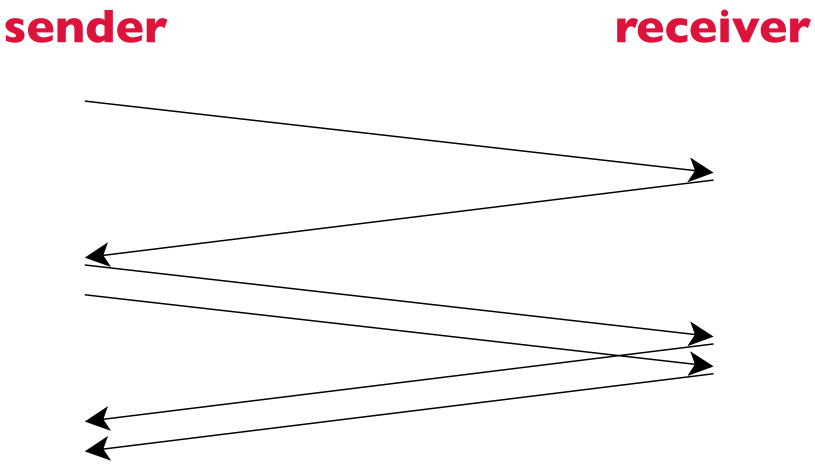
TCP Slow Start



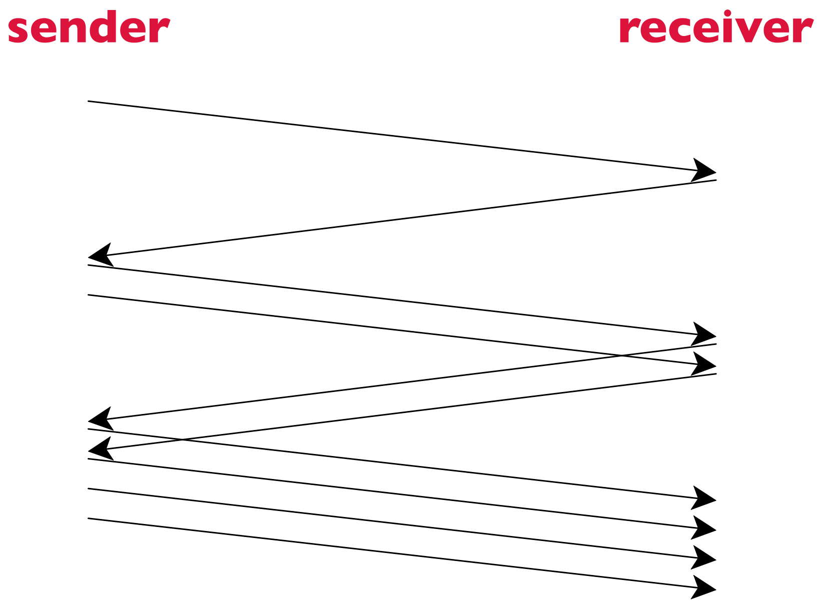
sender

receiver

TCP Slow Start



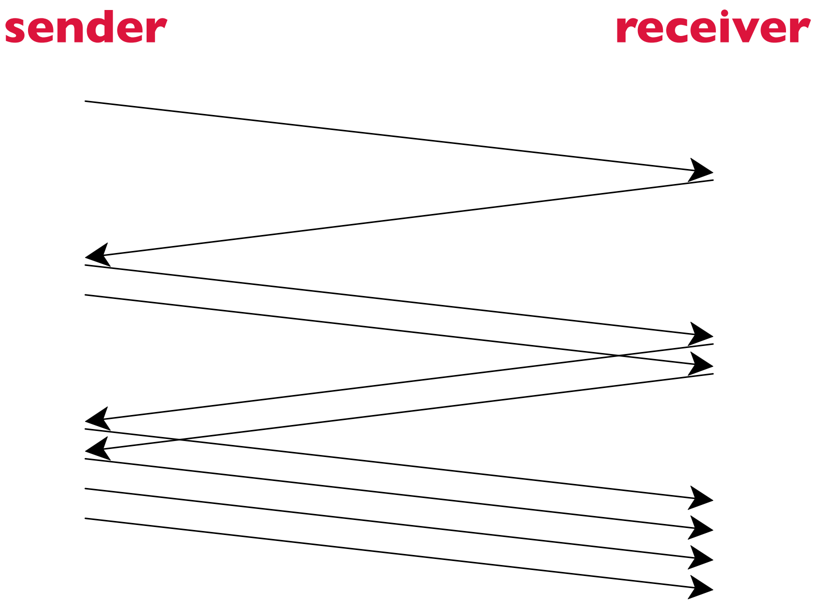
TCP Slow Start



sender

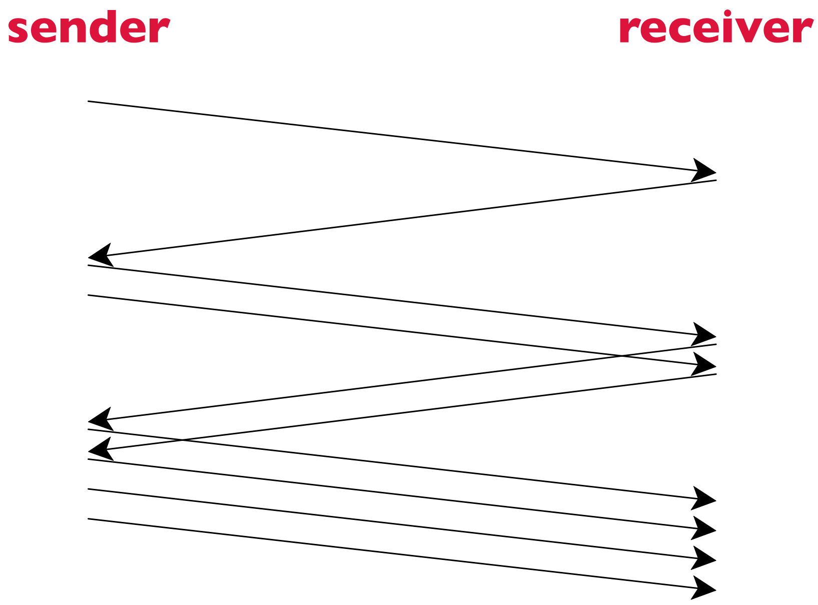
receiver

TCP Slow Start



Initially, `cwnd = MSS`

TCP Slow Start

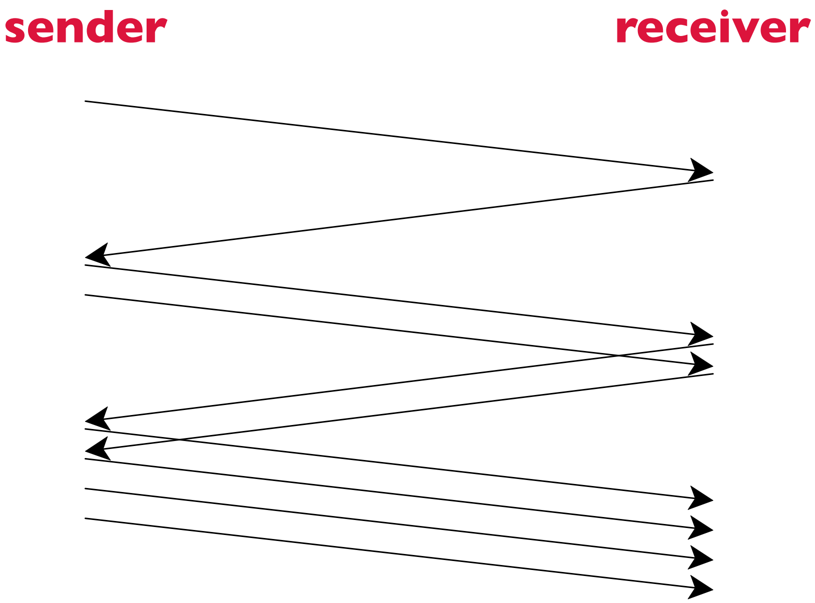


Initially, `cwnd = MSS`

After each ACK, `cwnd += MSS`

⇒ double `cwnd` each RTT

TCP Slow Start



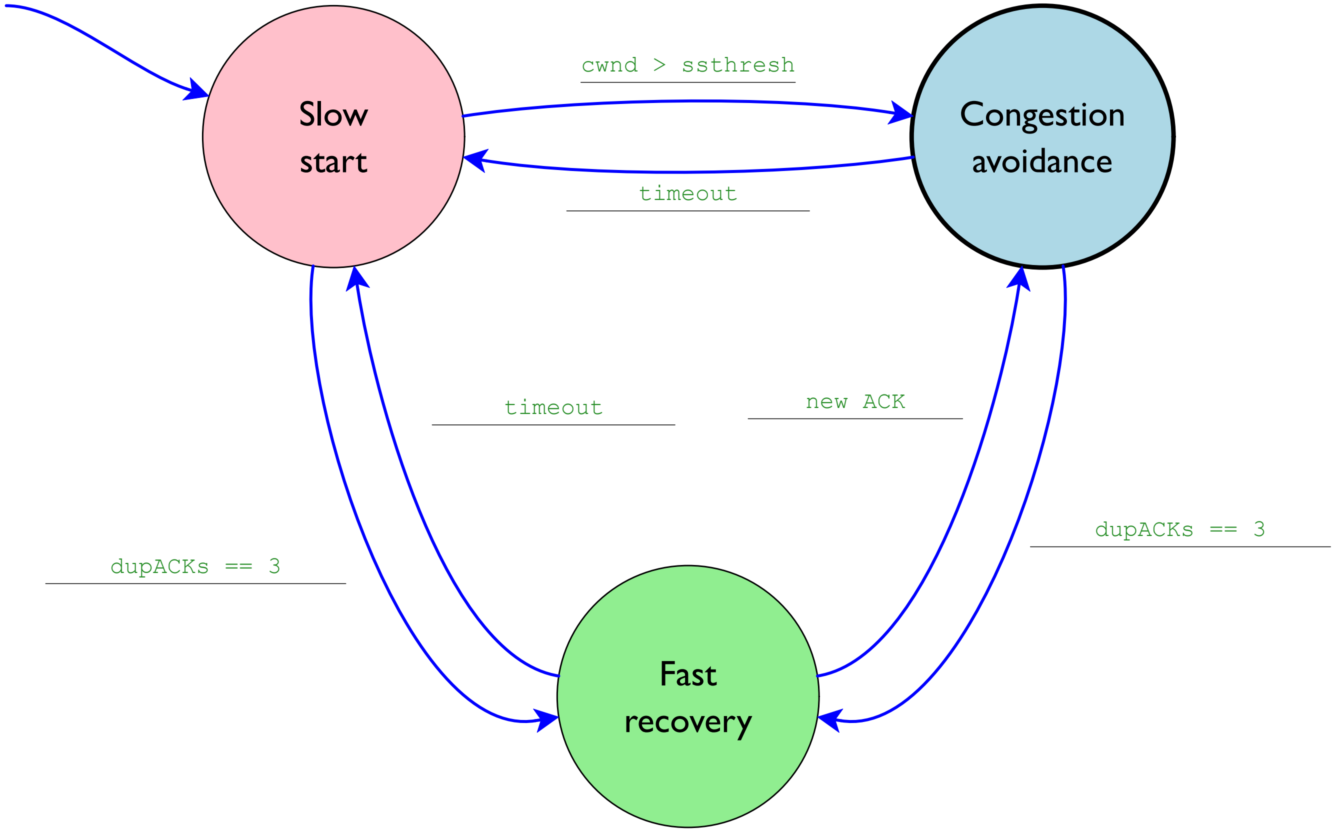
Initially, `cwnd = MSS`

After each ACK, `cwnd += MSS`

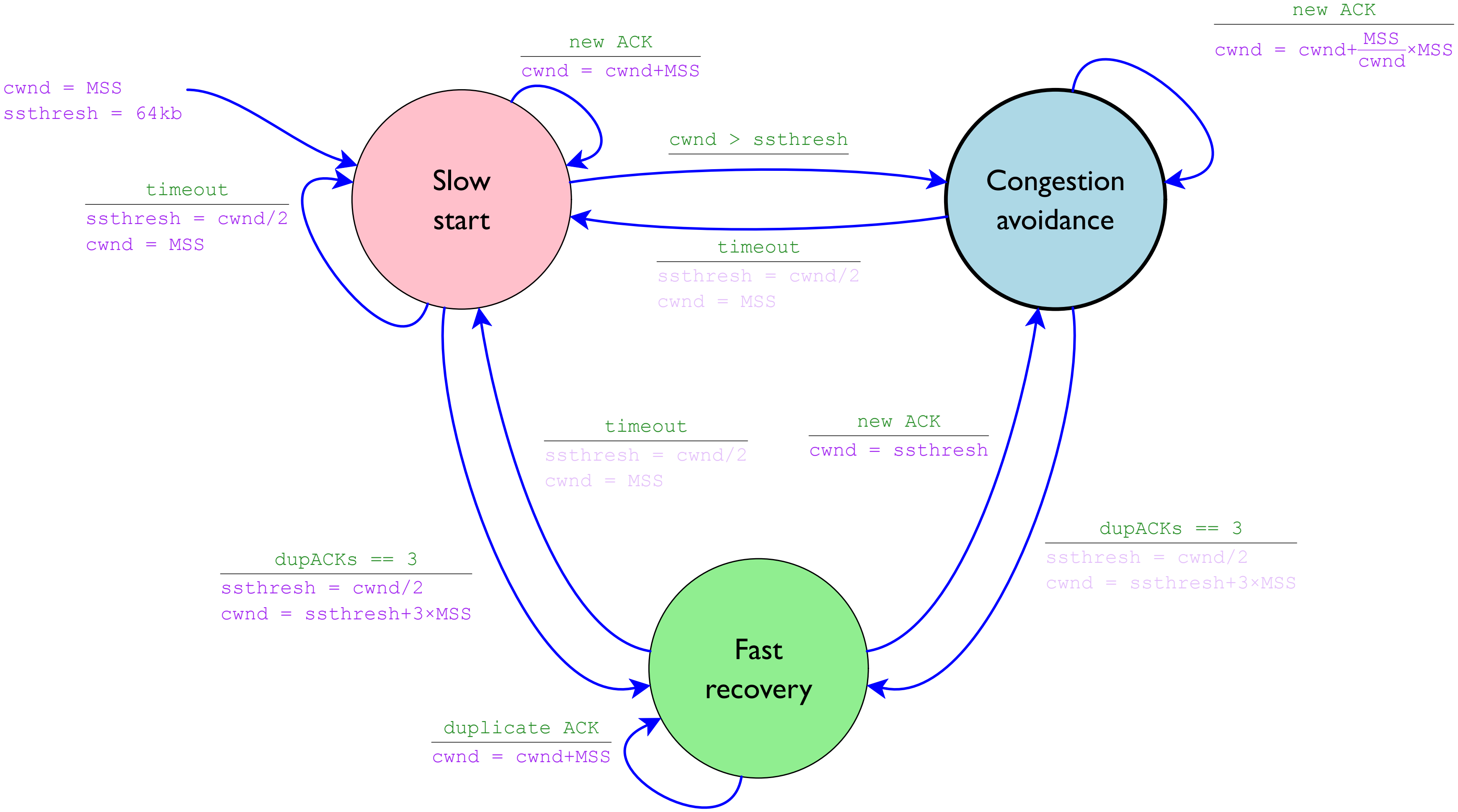
⇒ double `cwnd` each RTT

End slow start when `cwnd = ssthresh`

TCP Connection States



TCP Connection States



Slow start

Congestion avoidance

Fast recovery

cwnd = MSS
ssthresh = 64kb

new ACK
cwnd = cwnd+MSS

new ACK
cwnd = cwnd+$\frac{\text{MSS}}{\text{cwnd}}$×MSS

cwnd > ssthresh

timeout
ssthresh = cwnd/2
cwnd = MSS

timeout
ssthresh = cwnd/2
cwnd = MSS

timeout
ssthresh = cwnd/2
cwnd = MSS

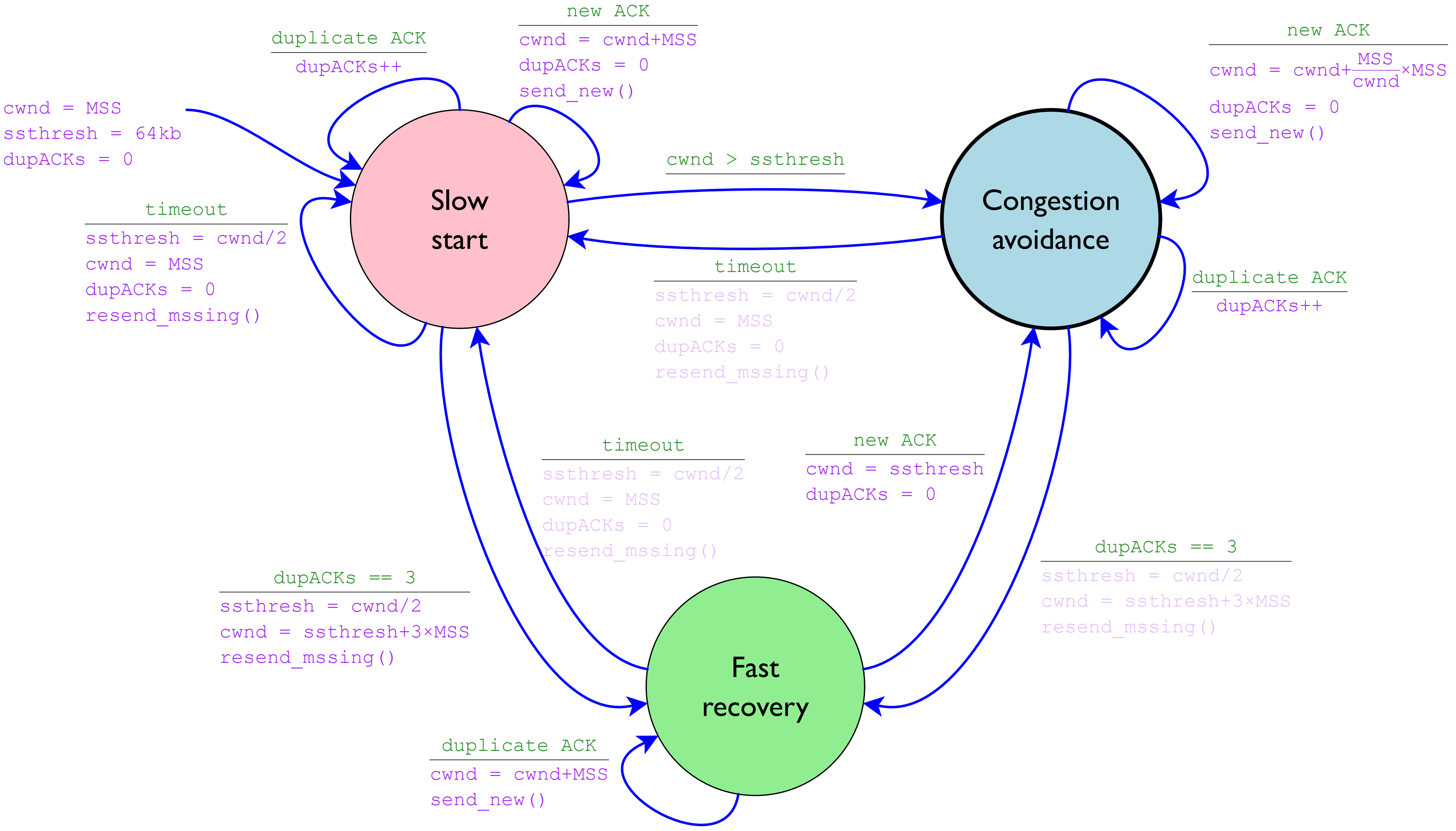
new ACK
cwnd = ssthresh

dupACKs == 3
ssthresh = cwnd/2
cwnd = ssthresh+3×MSS

dupACKs == 3
ssthresh = cwnd/2
cwnd = ssthresh+3×MSS

duplicate ACK
cwnd = cwnd+MSS

TCP Connection States



Slow start

Congestion avoidance

Fast recovery

cwnd = MSS
ssthresh = 64kb
dupACKs = 0

duplicate ACK
dupACKs++

new ACK
cwnd = cwnd+MSS
dupACKs = 0
send_new()

timeout
ssthresh = cwnd/2
cwnd = MSS
dupACKs = 0
resend_mssing()

cwnd > ssthresh

new ACK
cwnd = cwnd+$\frac{MSS}{cwnd}$×MSS
dupACKs = 0
send_new()

timeout
ssthresh = cwnd/2
cwnd = MSS
dupACKs = 0
resend_mssing()

duplicate ACK
dupACKs++

timeout
ssthresh = cwnd/2
cwnd = MSS
dupACKs = 0
resend_mssing()

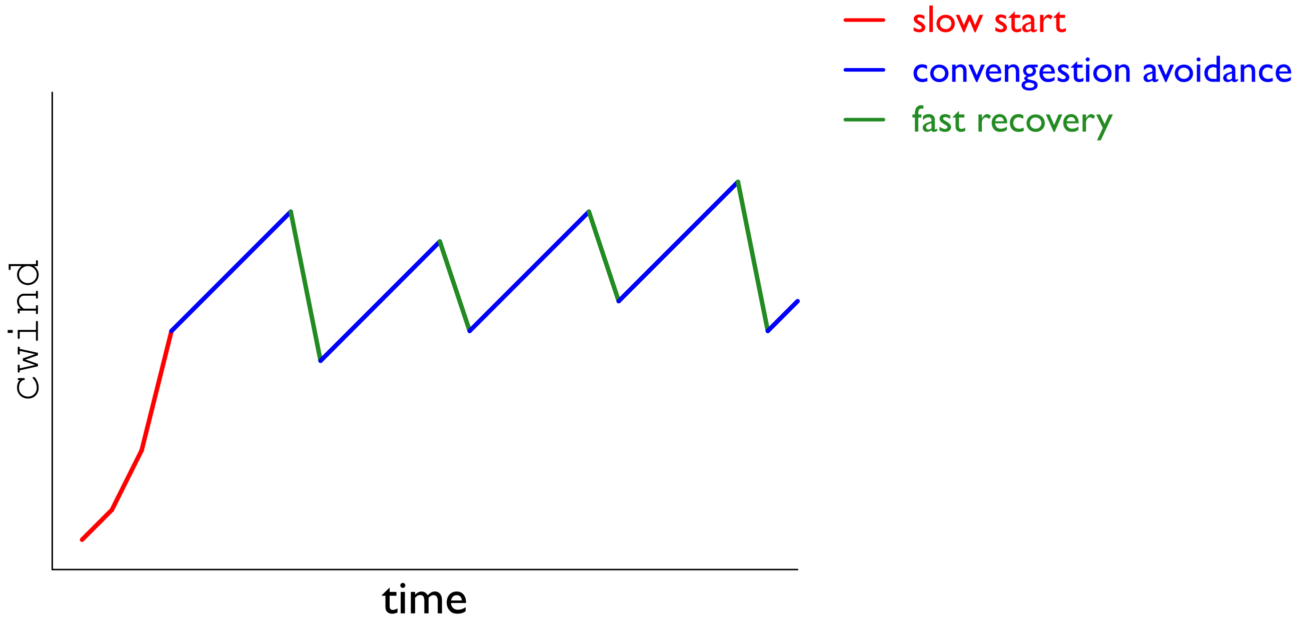
new ACK
cwnd = ssthresh
dupACKs = 0

dupACKs == 3
ssthresh = cwnd/2
cwnd = ssthresh+3×MSS
resend_mssing()

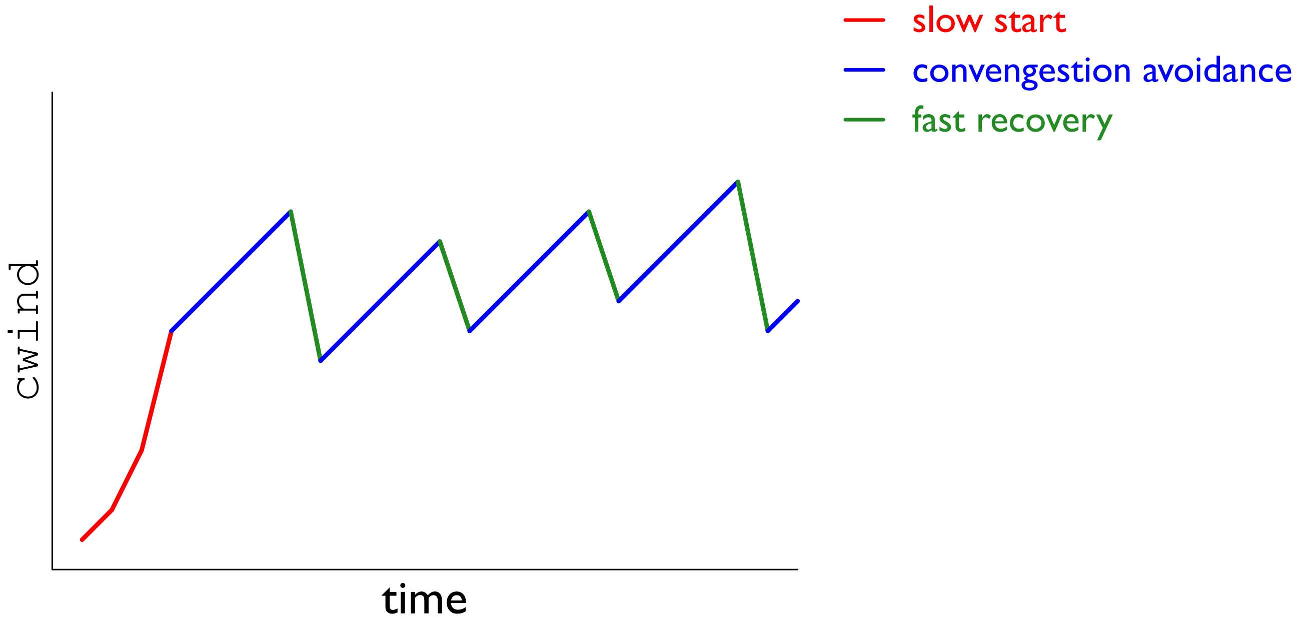
dupACKs == 3
ssthresh = cwnd/2
cwnd = ssthresh+3×MSS
resend_mssing()

duplicate ACK
cwnd = cwnd+MSS
send_new()

`cwind` Adjustment over Time

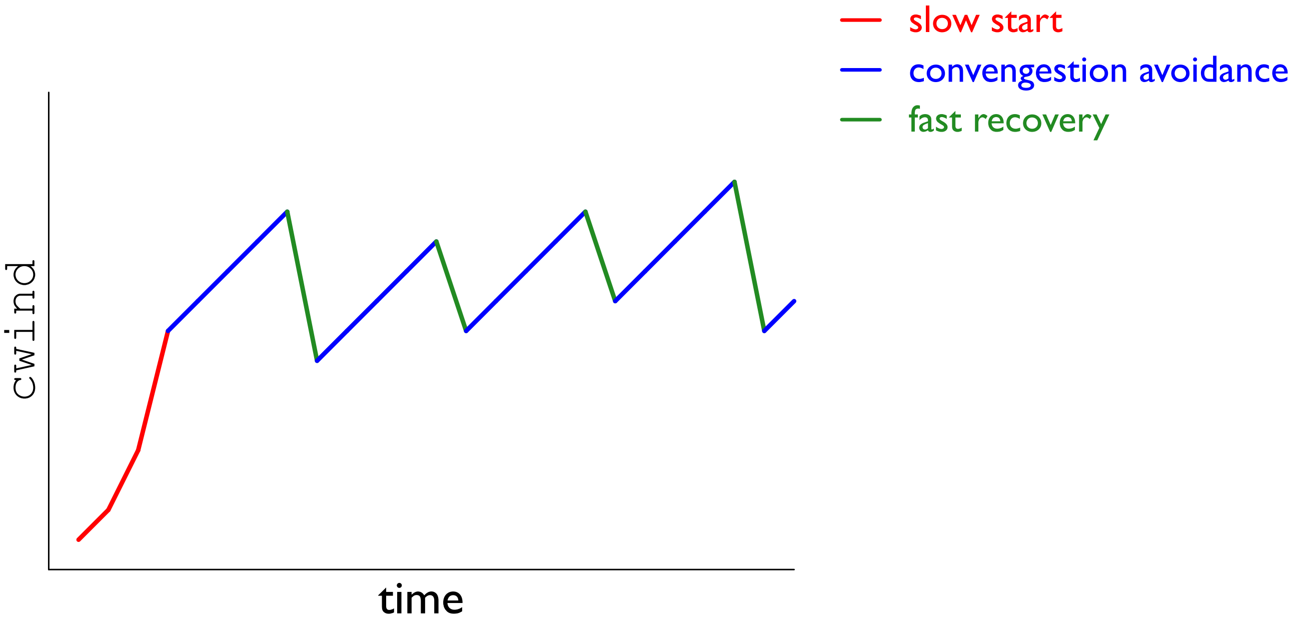


`cwind` Adjustment over Time



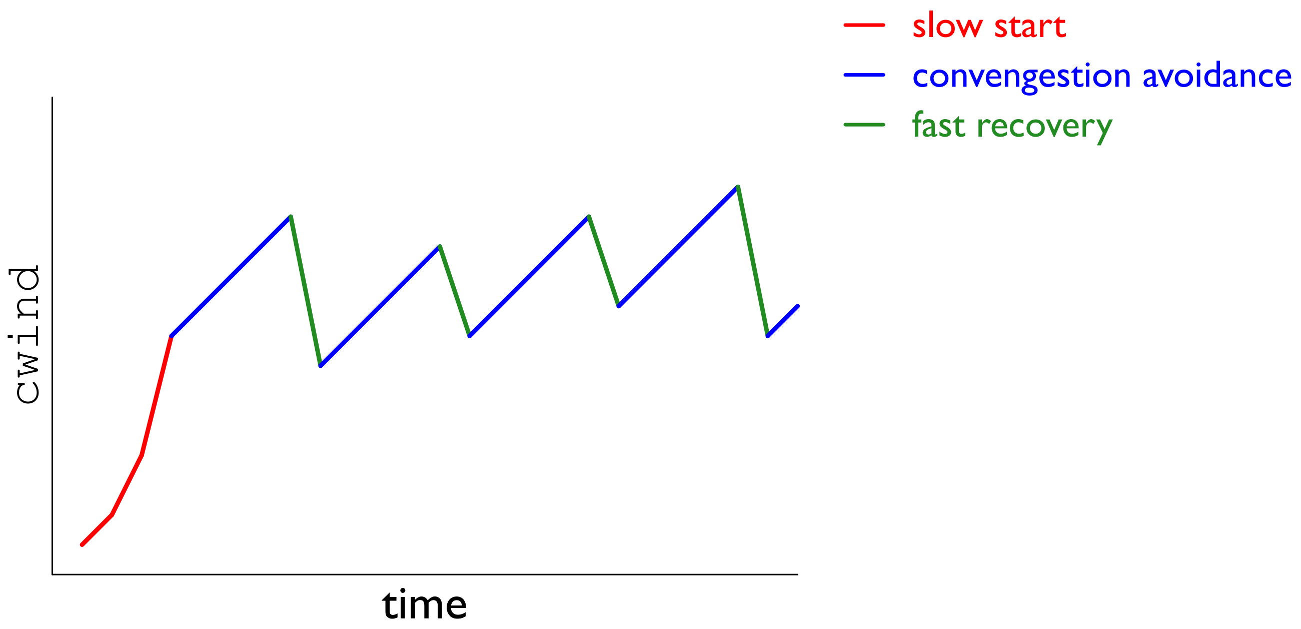
Additive increase, multiplicative decrease (AIMD)

`cwind` Adjustment over Time



Expect average `cwnd` to be 75% of maximum

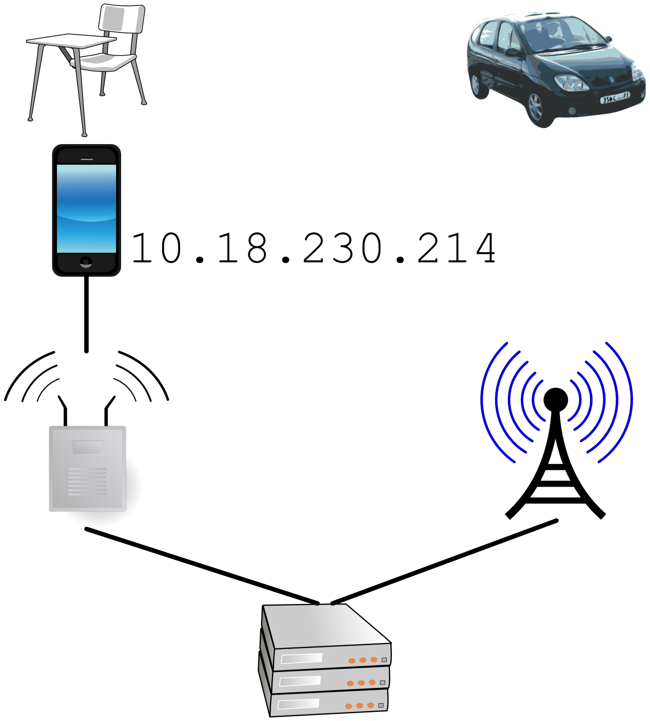
`cwind` Adjustment over Time



Fair , because multiple senders tend toward same rate

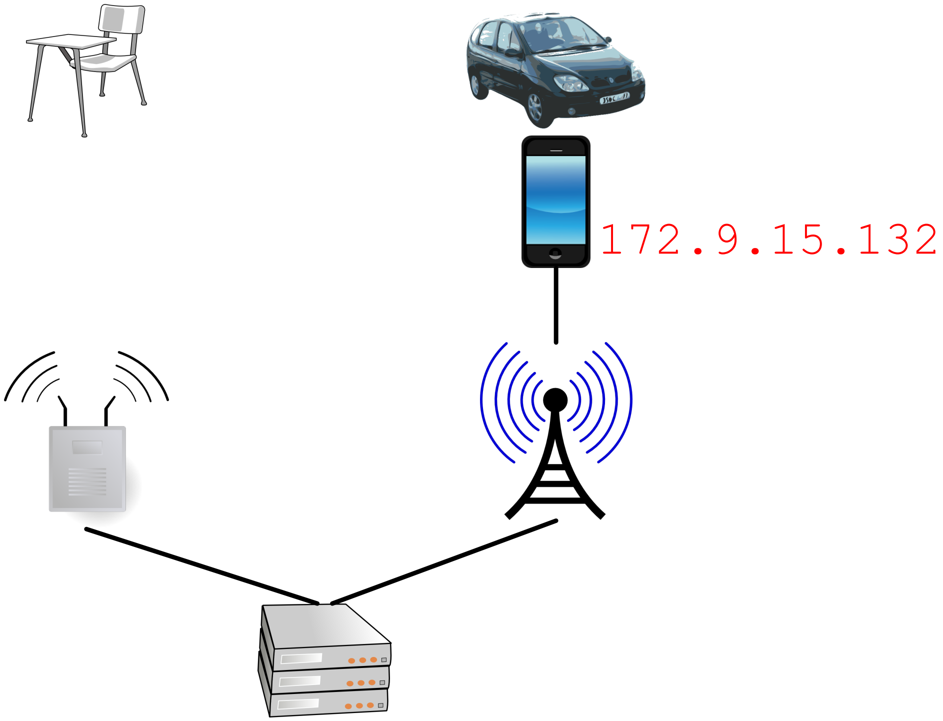
Issues with TCP

Parking-lot problem



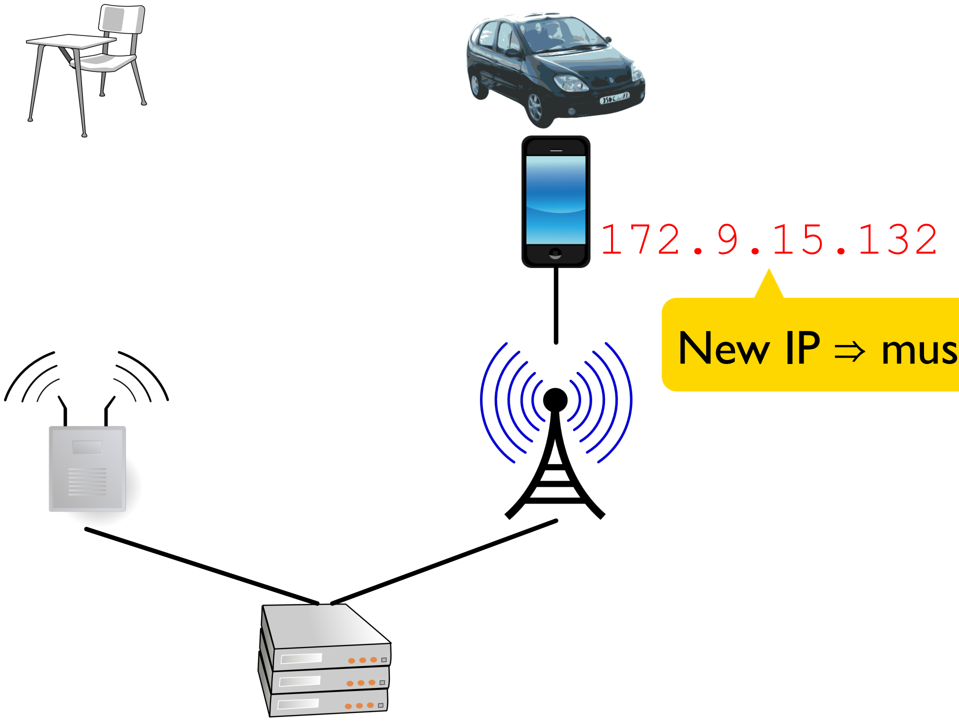
Issues with TCP

Parking-lot problem



Issues with TCP

Parking-lot problem

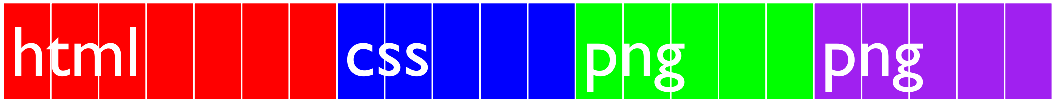


New IP ⇒ must reconnect

Issues with TCP

Head-of-line problem

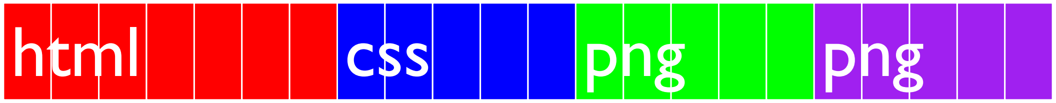
A typical web page needs multiple files:



Issues with TCP

Head-of-line problem

A typical web page needs multiple files:

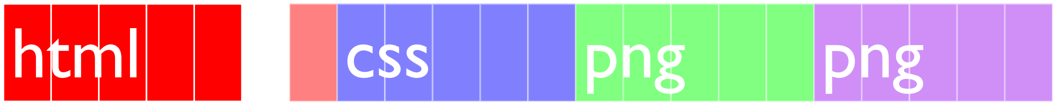


Getting parts in order can delay the whole page

Issues with TCP

Head-of-line problem

A typical web page needs multiple files:

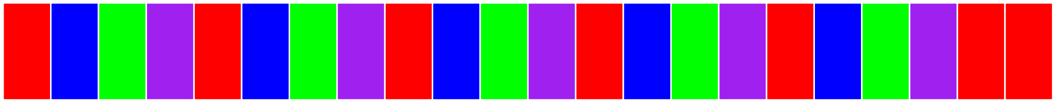


A dropped packet delays everything further

Issues with TCP

Head-of-line problem

A typical web page needs multiple files:

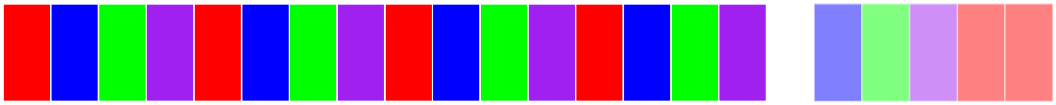


HTTP/2 allows interleaving within a reply...

Issues with TCP

Head-of-line problem

A typical web page needs multiple files:

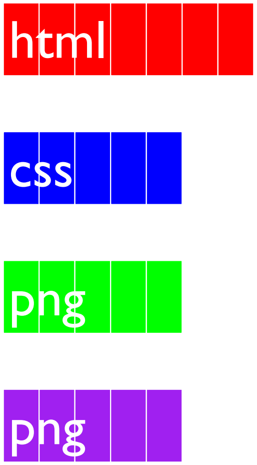


but that doesn't solve the dropped-packet problem

Issues with TCP

Head-of-line problem

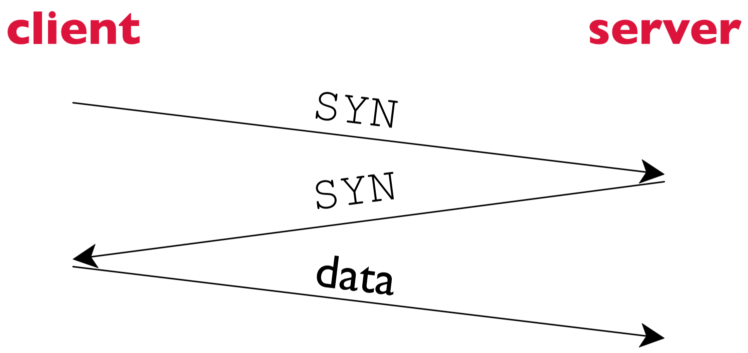
A typical web page needs multiple files:



Multiple connections work, but each takes time to set up

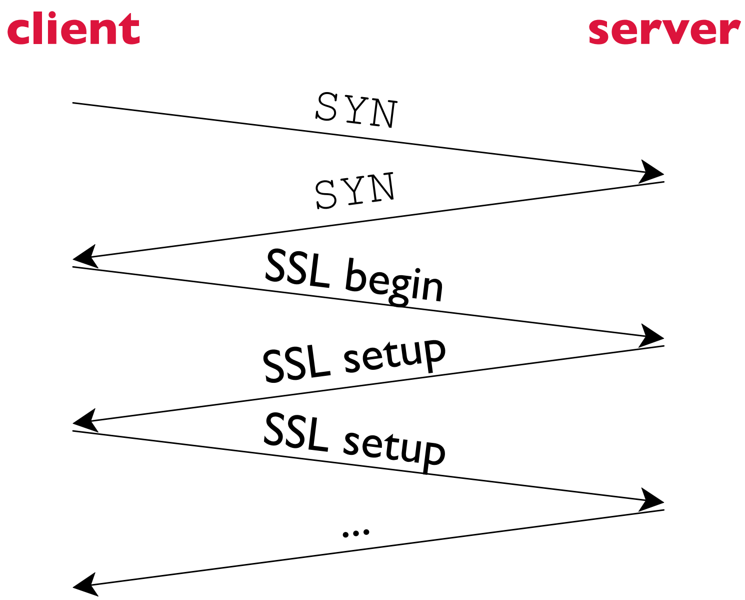
Issues with TCP

Handshake hell



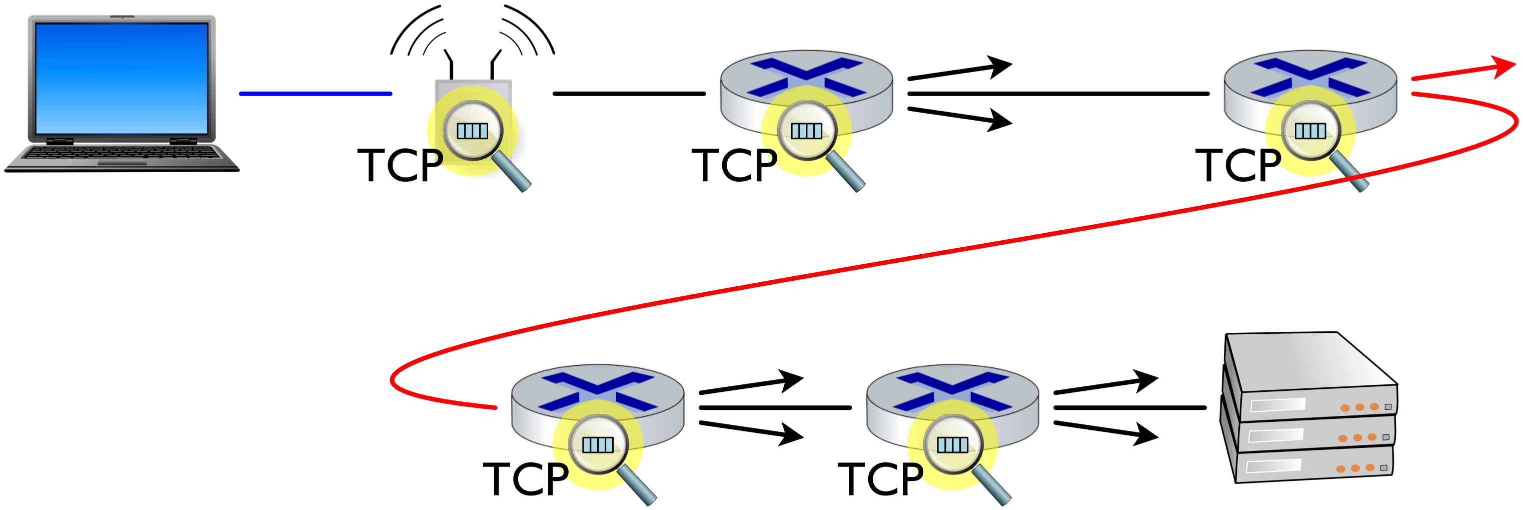
Issues with TCP

Handshake hell



Issues with TCP

Ossification



QUIC

QUIC : Quick UDP Internet Connections

- implemented in Chrome in 2012
- standardized in 2021 as RFC 9000
- implemented in major browsers

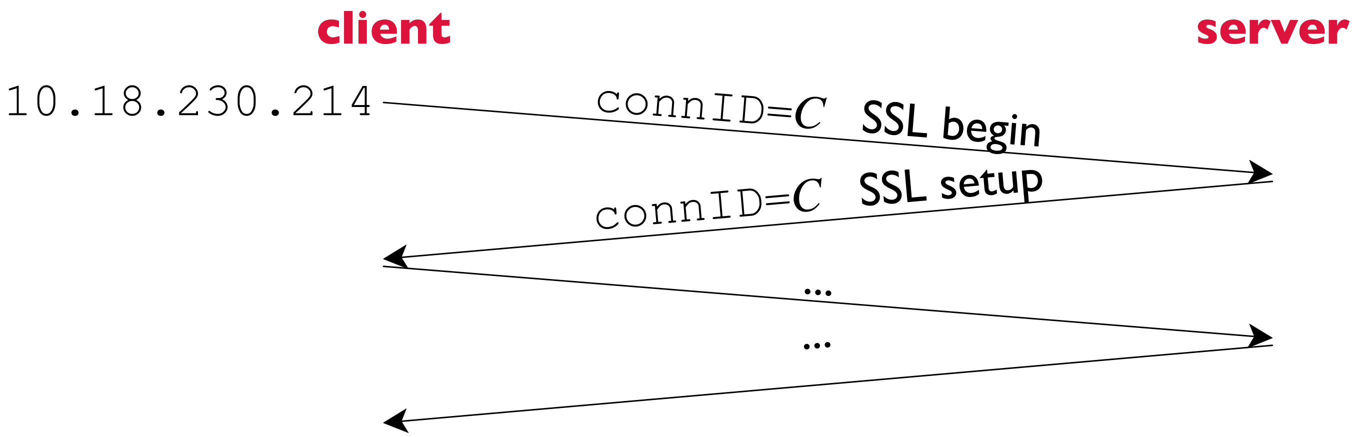
QUIC

QUIC : Quick UDP Internet Connections

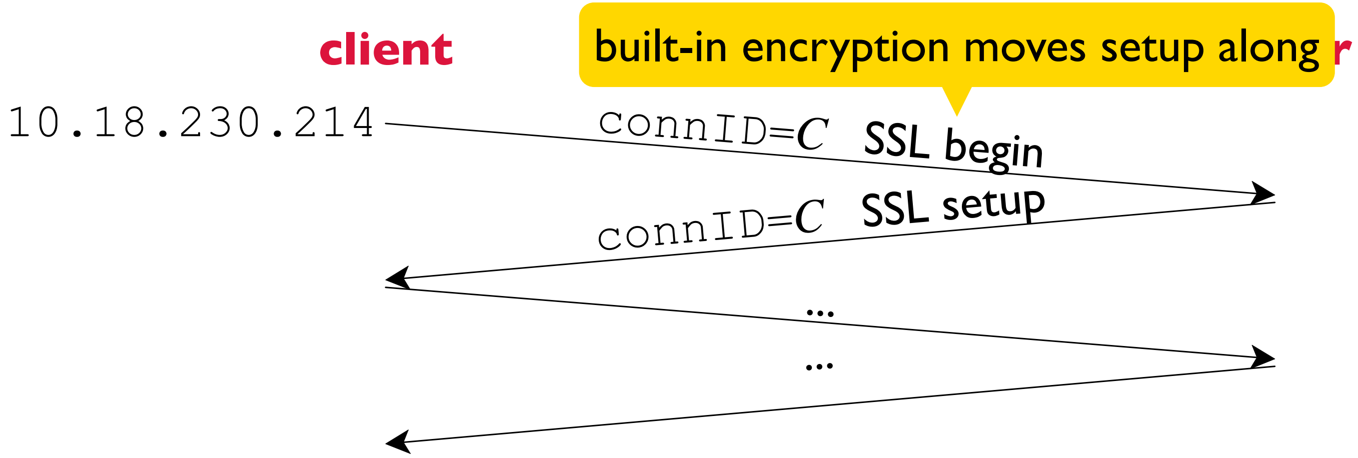
- builds on UDP
- connection-oriented based on a connection ID

  not host and port
- built-in encryption, covers more headers
- can interleave files without dropped-packet interactions

QUIC

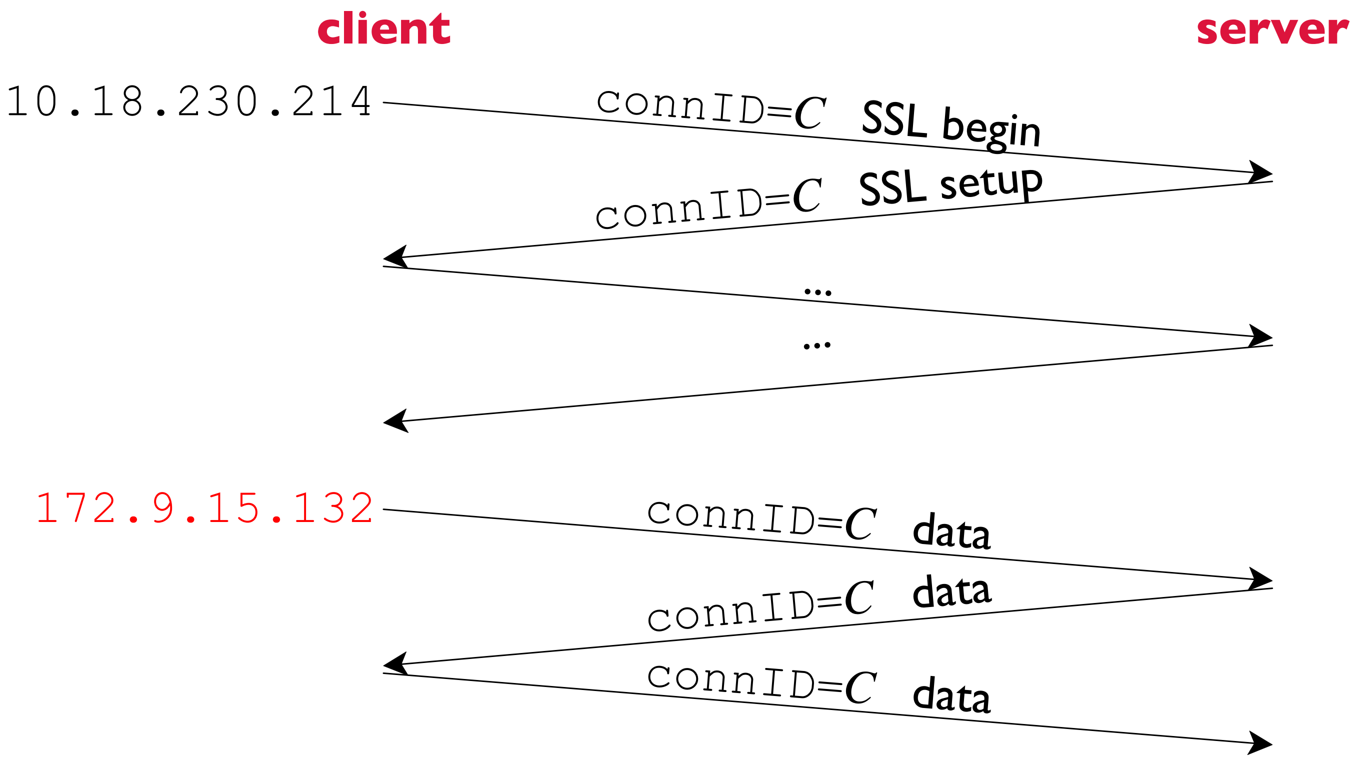

client
server
10.18.230.214
connID=C SSL begin
connID=C SSL setup
...
...

QUIC

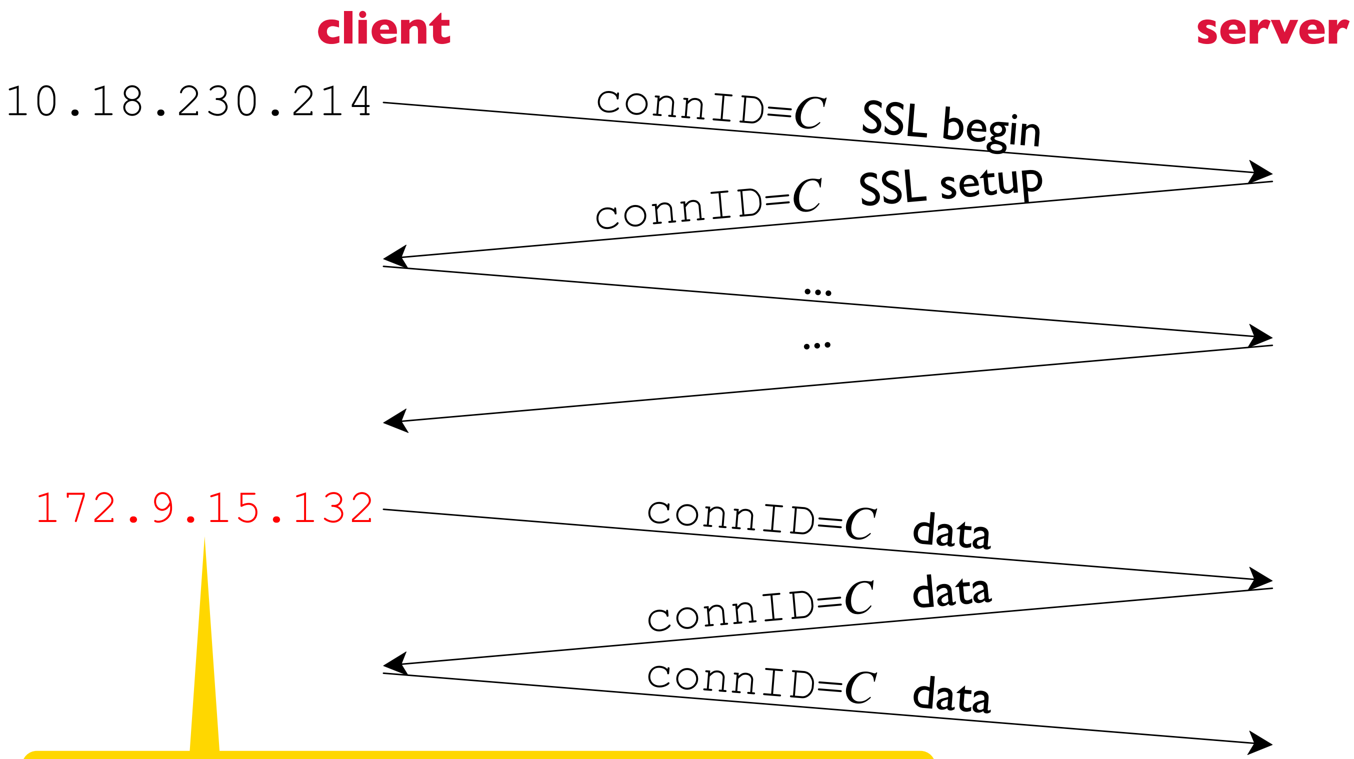


built-in encryption moves setup along

QUIC


client
server
10.18.230.214
connID=*C* SSL begin
connID=*C* SSL setup
...
...
172.9.15.132
connID=*C* data
connID=*C* data
connID=*C* data

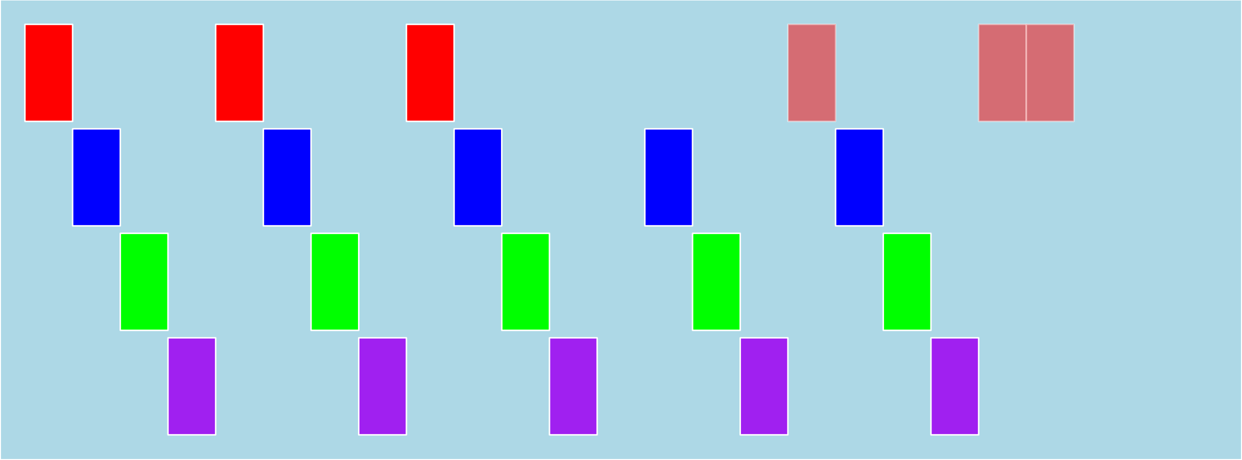
QUIC



Connection ID allows continue without setup from a new client address

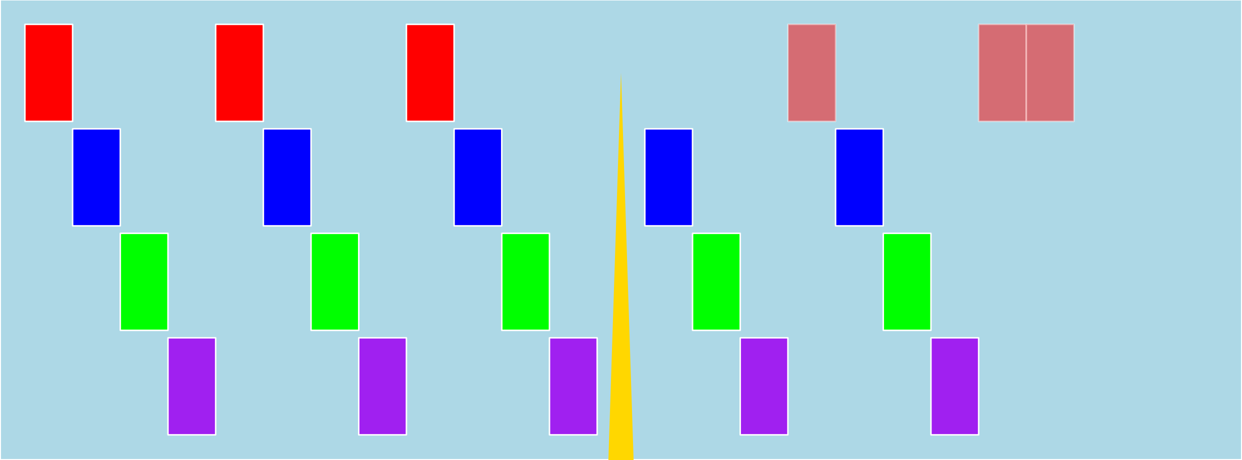
QUIC

Concurrent streams are handled at the packet level within a connection



QUIC

Concurrent streams are handled at the packet level within a connection



Dropped packet does not stall other streams