# TCP Congestion Control

**Congestion control** in TCP avoids overwhelming a network

- how *often* to send ⇒ `timeout`
- how *much* to send ⇒ `window`

# TCP Timeout

TCP timeout uses an estimate of RTT and its variance:

$$\texttt{timeout} = RTT_{est} + 4 \times var_{est}$$

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

$$RTT_{est} = (1-\alpha) \times RTT_{est} + \alpha \times RTT_{measured}$$

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

- $var_{est}$ is similarly weighted:

$$var_{est} = (1-\beta) \times var_{est} + \beta \times |RTT_{measured} - RTT_{est}|$$

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`

- 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`

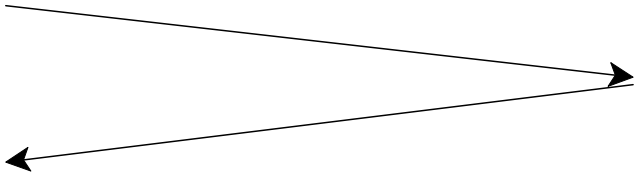
**Congestion control** uses `cwnd` and `window ≤ cwnd`

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

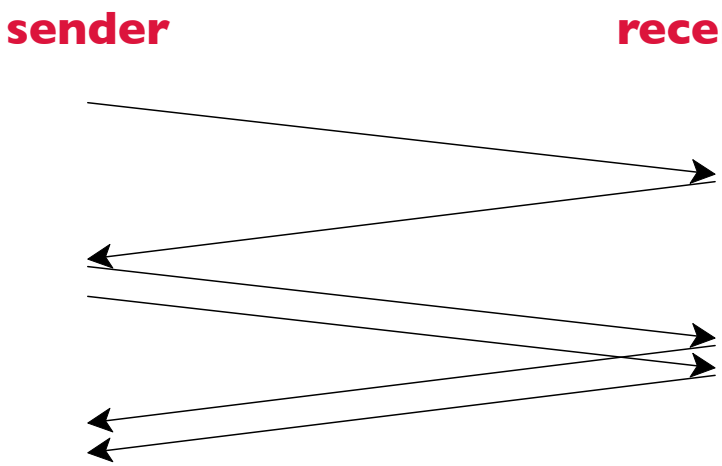
start at 64KB

# TCP Slow Start

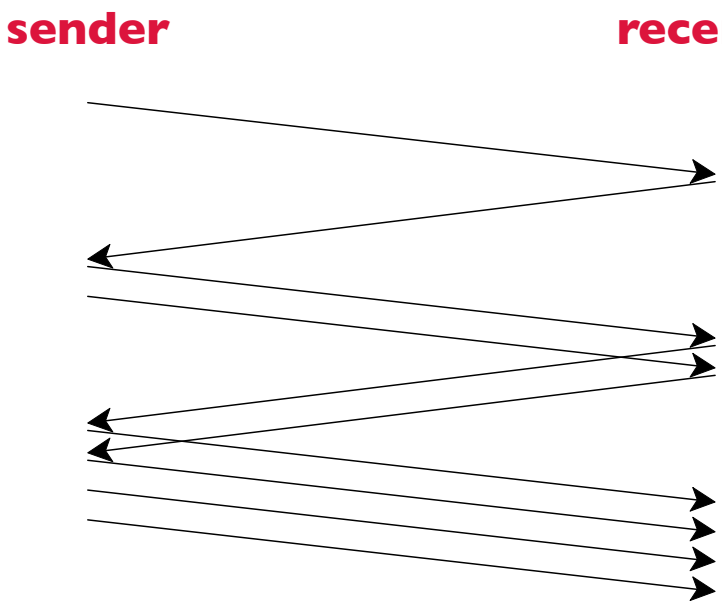
**sender** **receiver**



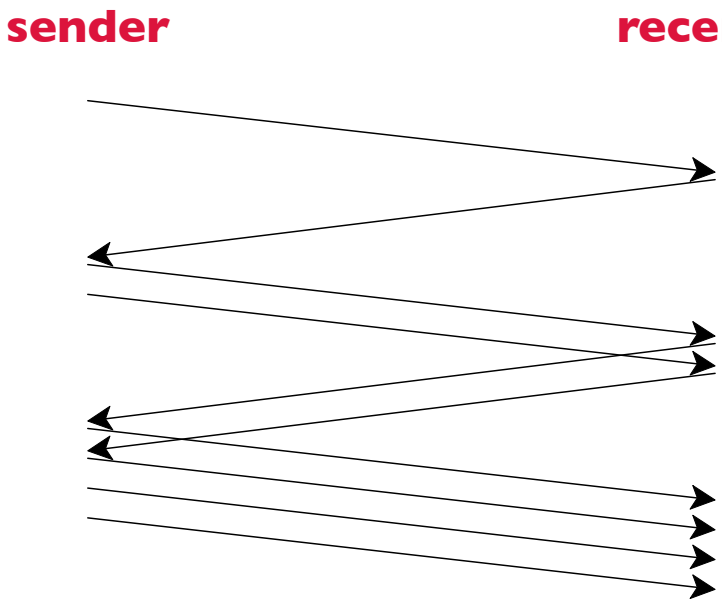
# TCP Slow Start



# TCP Slow Start

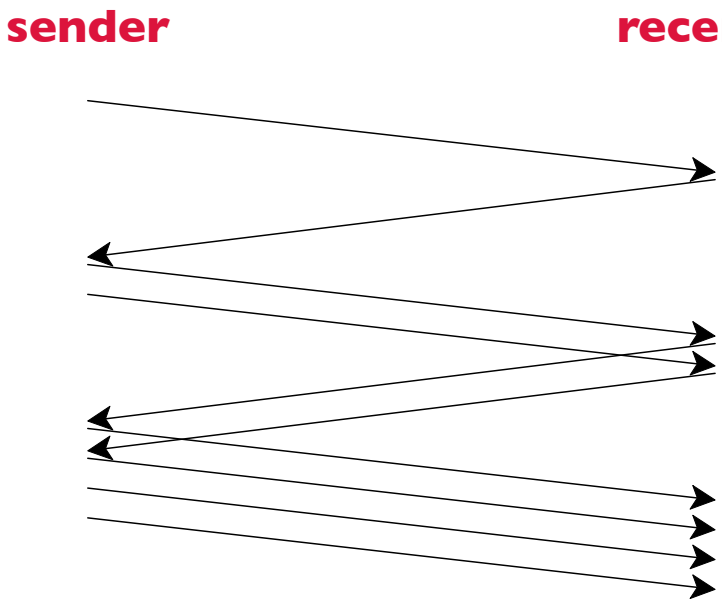


# TCP Slow Start




Initially, `cwnd = MSS`

# TCP Slow Start

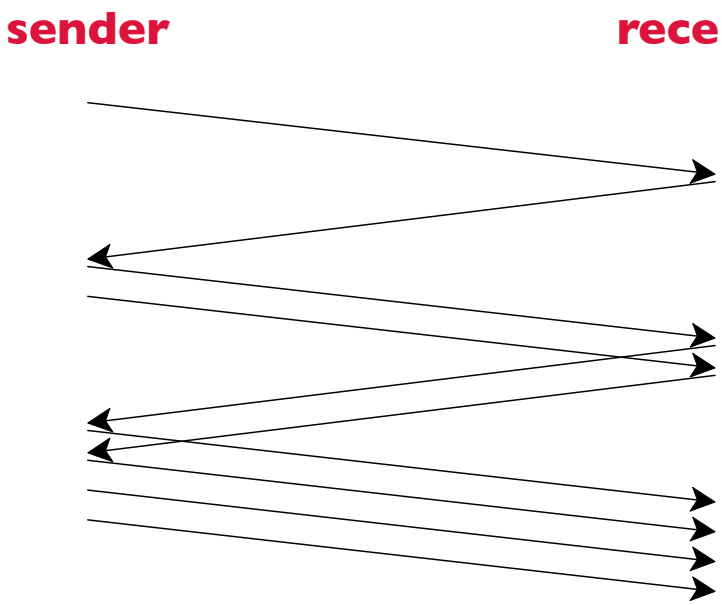



Initially, `cwnd = MSS`

After each ACK, `cwnd += MSS`

⇒ double `cwnd` each RTT

# TCP Slow Start



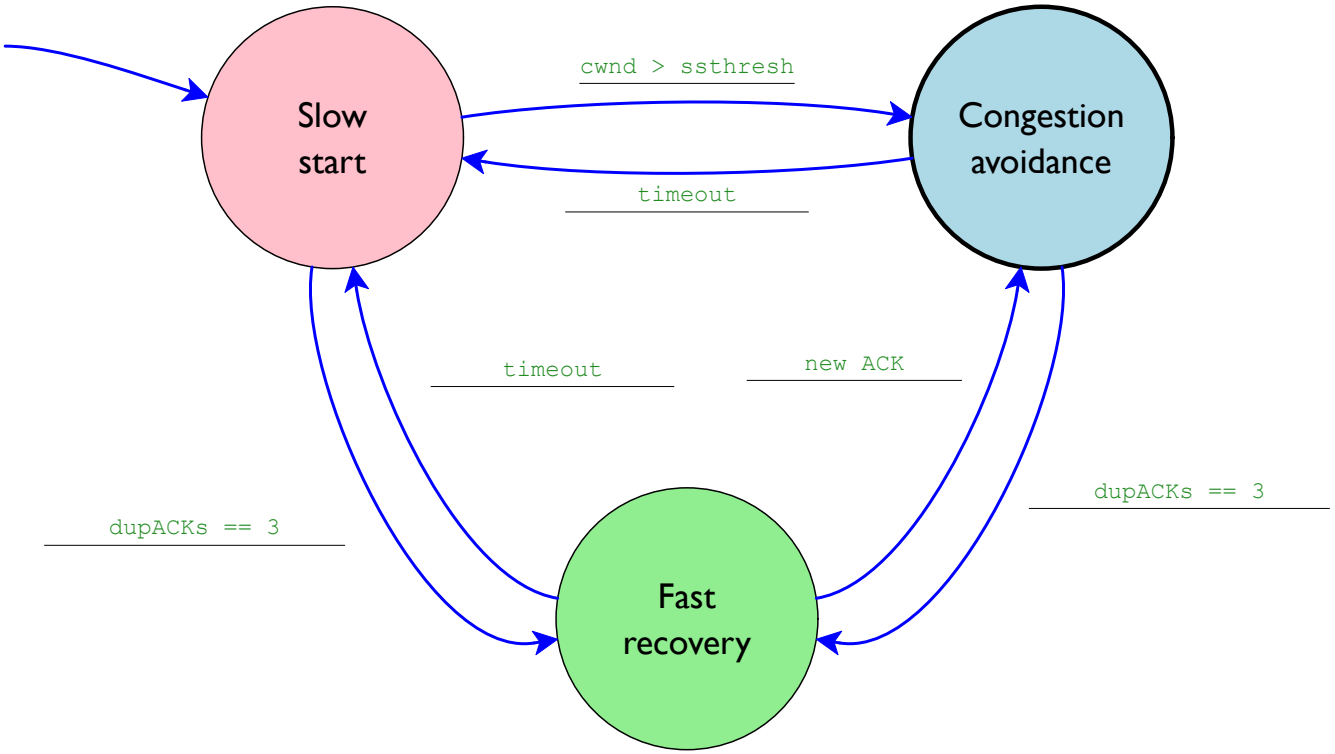

Initially, `cwnd = MSS`

After each ACK, `cwnd += MSS`

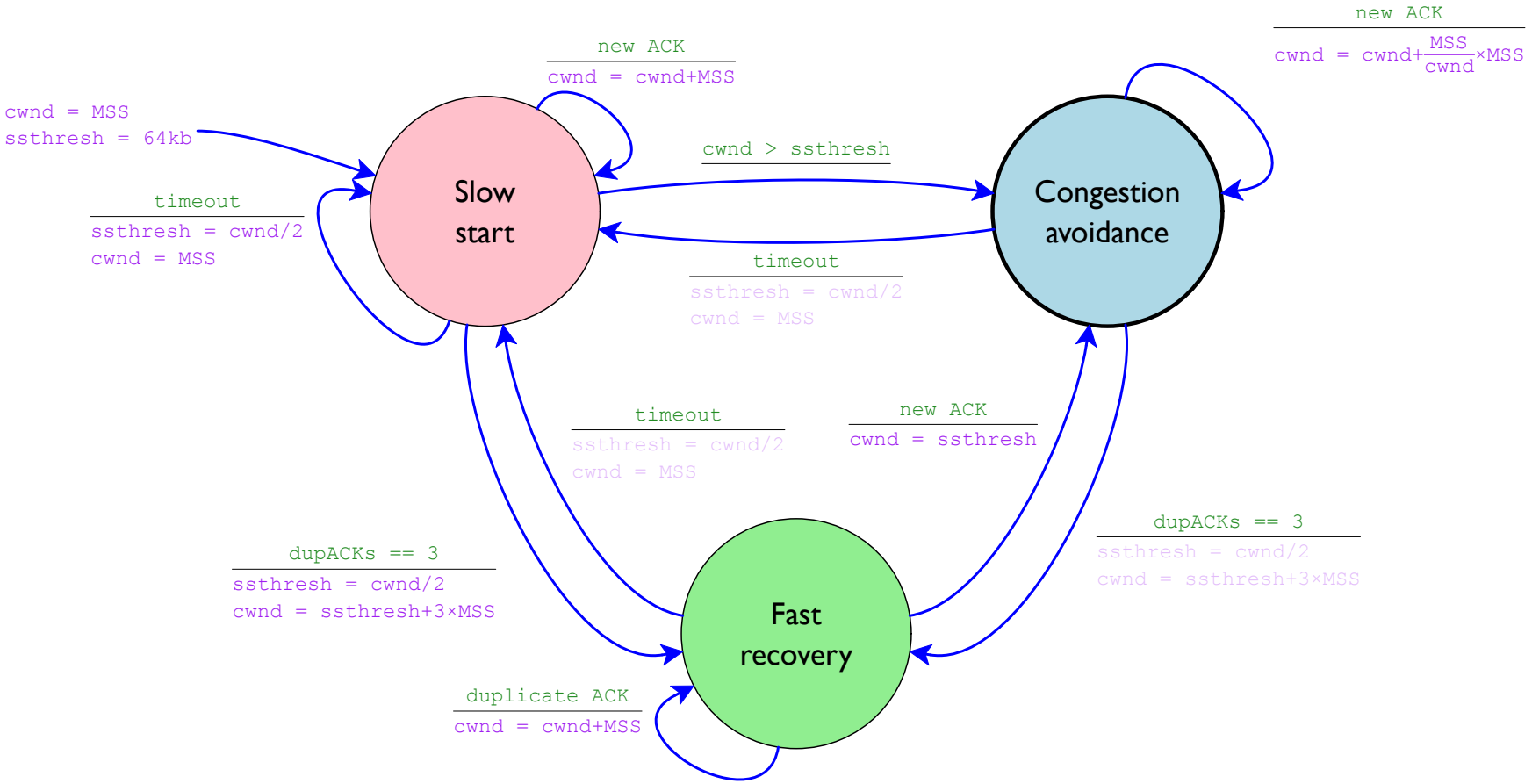
⇒ double `cwnd` each RTT

End slow start when
`cwnd = ssthresh`

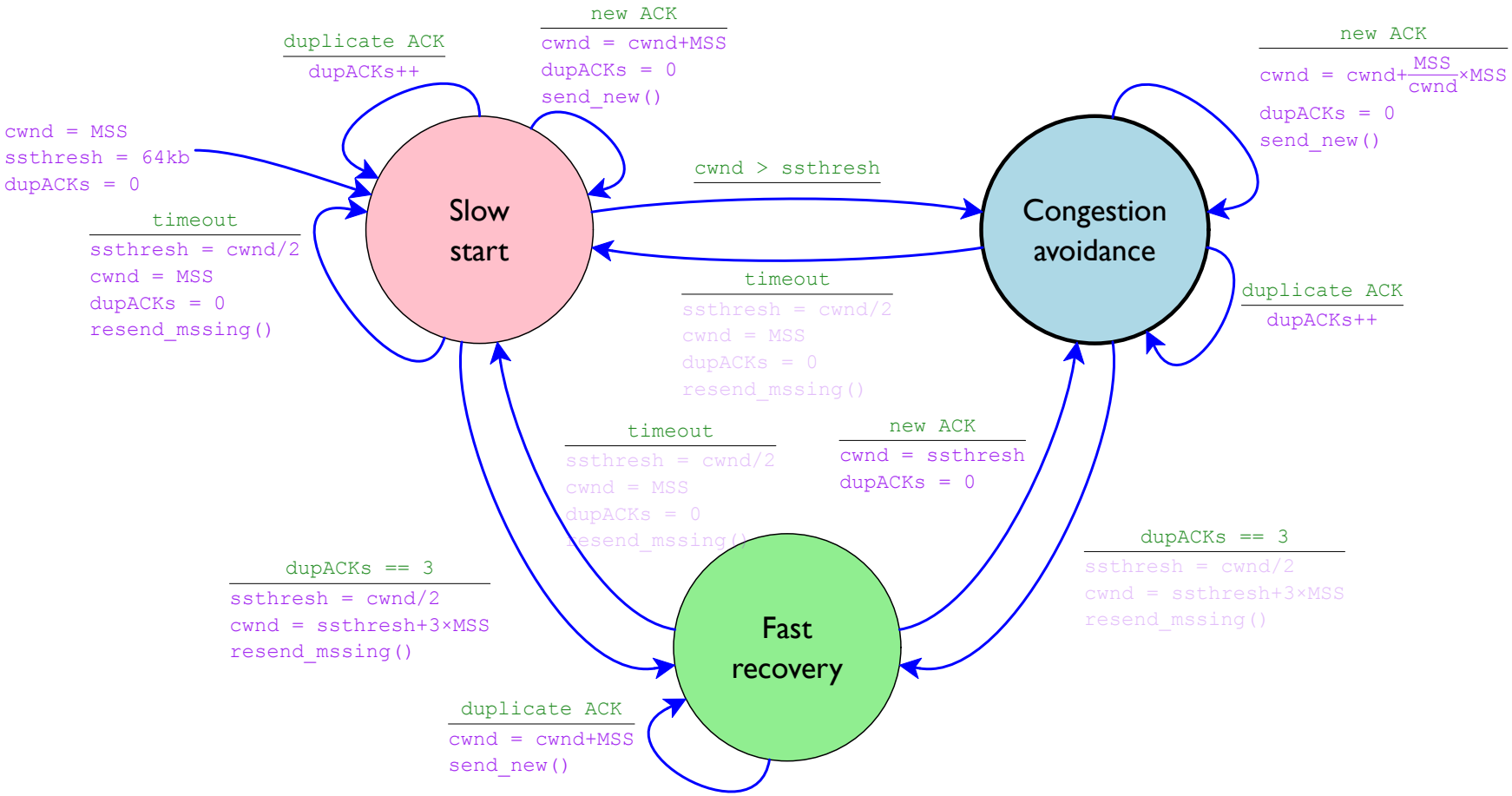
# TCP Connection States



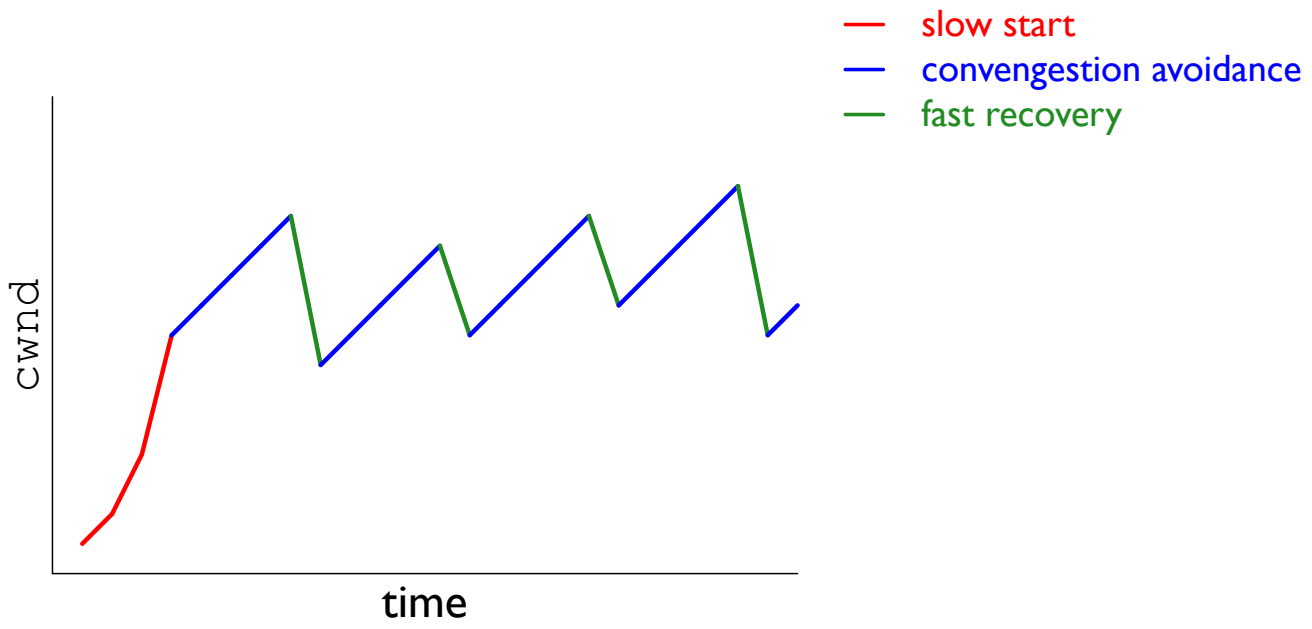
# TCP Connection States



# TCP Connection States

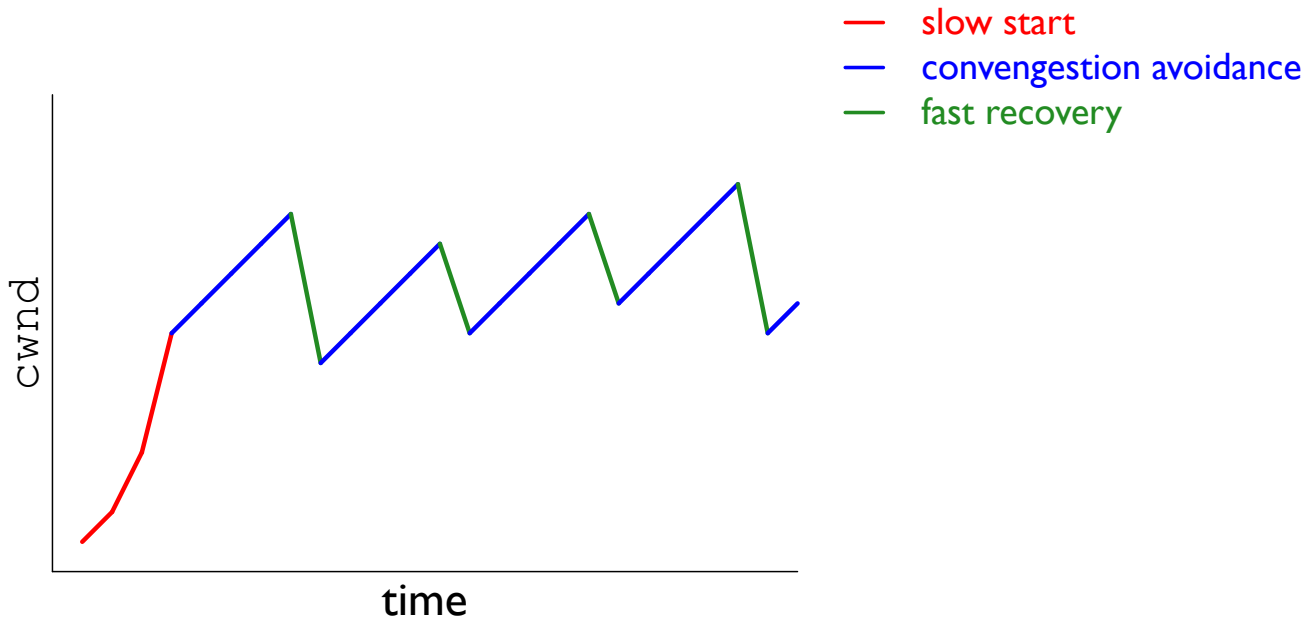


# `cwnd` Adjustment over Time



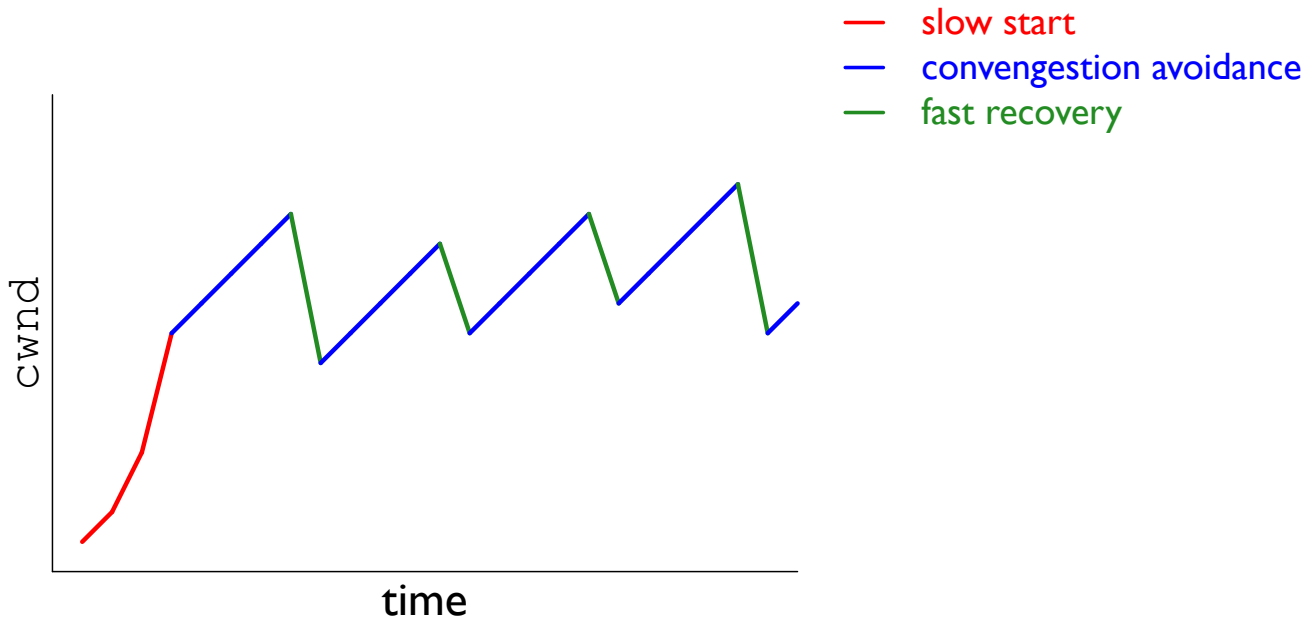

Additive increase, multiplicative decrease (AIMD)

# `cwnd` Adjustment over Time




Expect average `cwnd` to be 75% of maximum

# `cwnd` Adjustment over Time




*Fair*, because multiple senders tend toward same rate

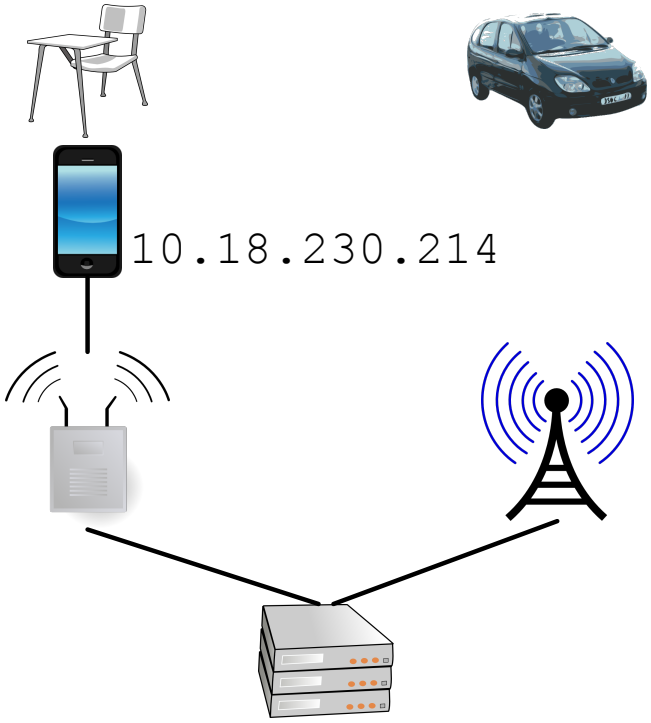
# Issues with TCP

TCP is great for most purposes, but it's not perfect...

... for example, in the case of web services

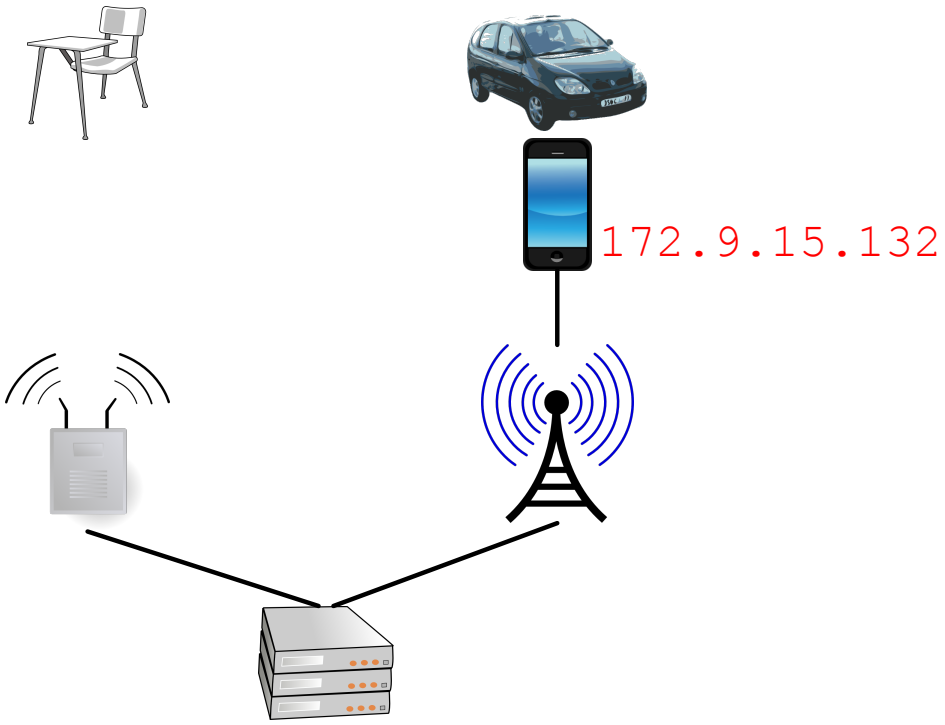
# Issues with TCP

## Parking-lot problem



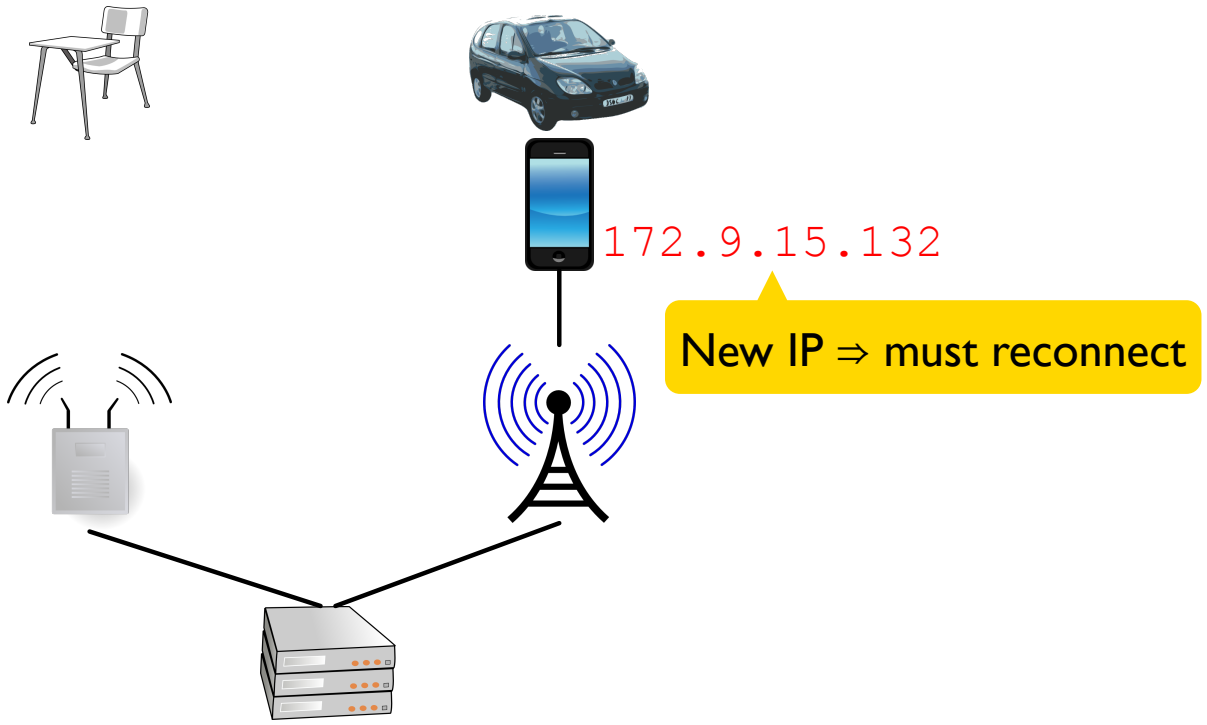
# Issues with TCP

## Parking-lot problem



# Issues with TCP

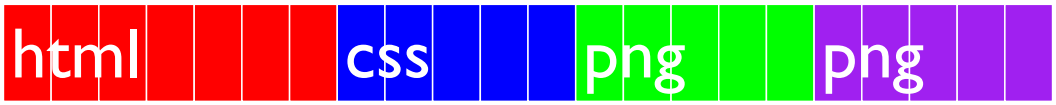
## Parking-lot problem



# Issues with TCP

## Head-of-line problem

A typical web page needs multiple files:

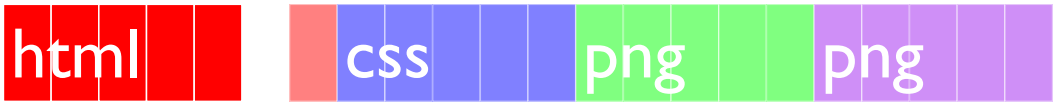



Getting parts in order can delay the whole page

# 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

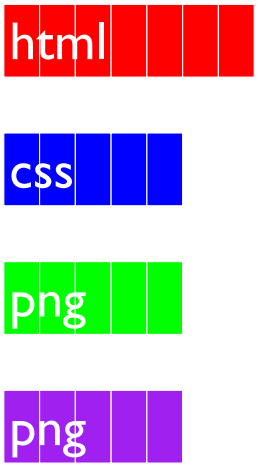
A typical web page needs multiple files:

but that doesn't solve the dropped-packet problem

# Issues with TCP

## Head-of-line problem

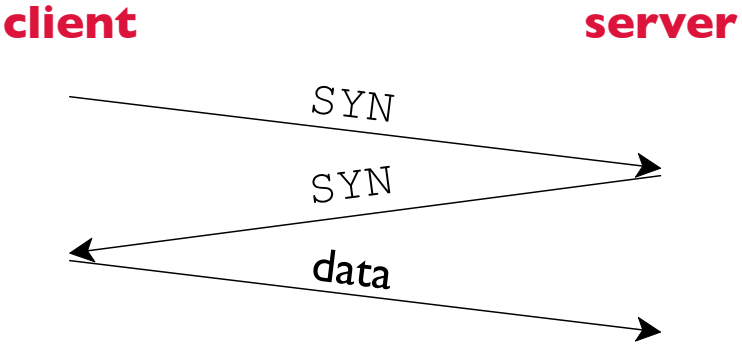
A typical web page needs multiple files:




Multiple connections work, but each takes time to set up

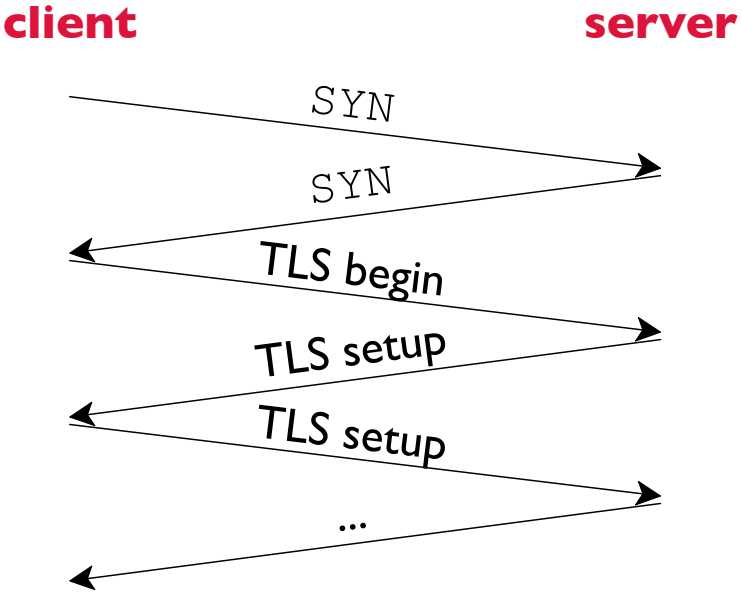
# Issues with TCP

**Handshake hell**



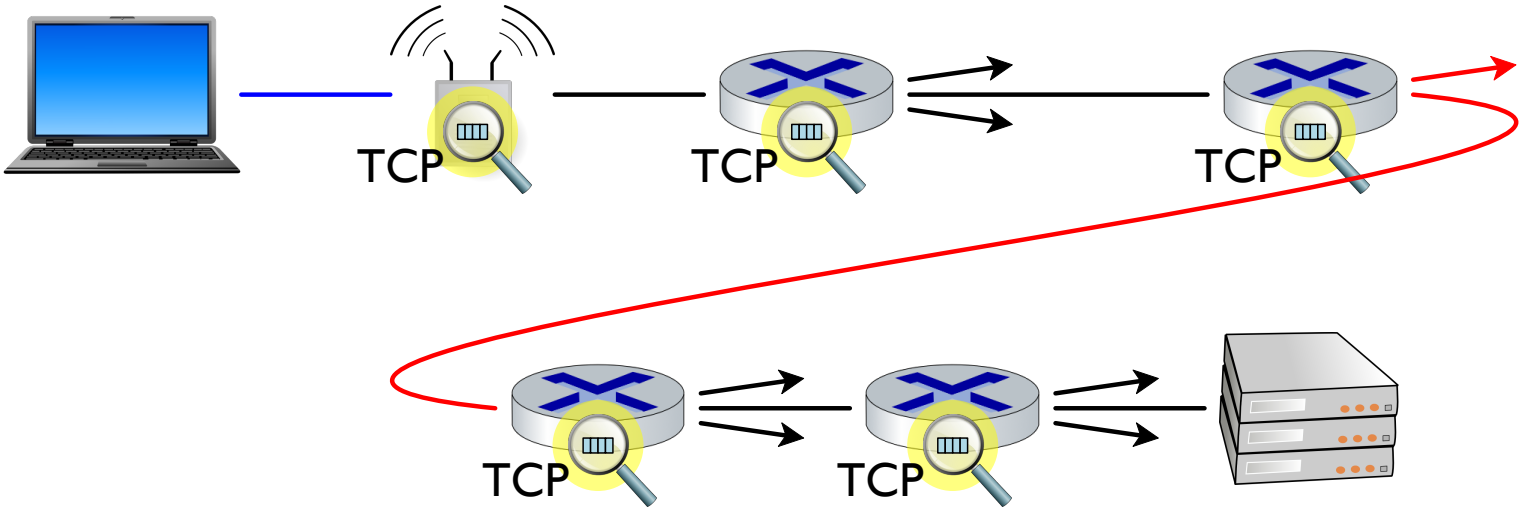
# Issues with TCP

**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

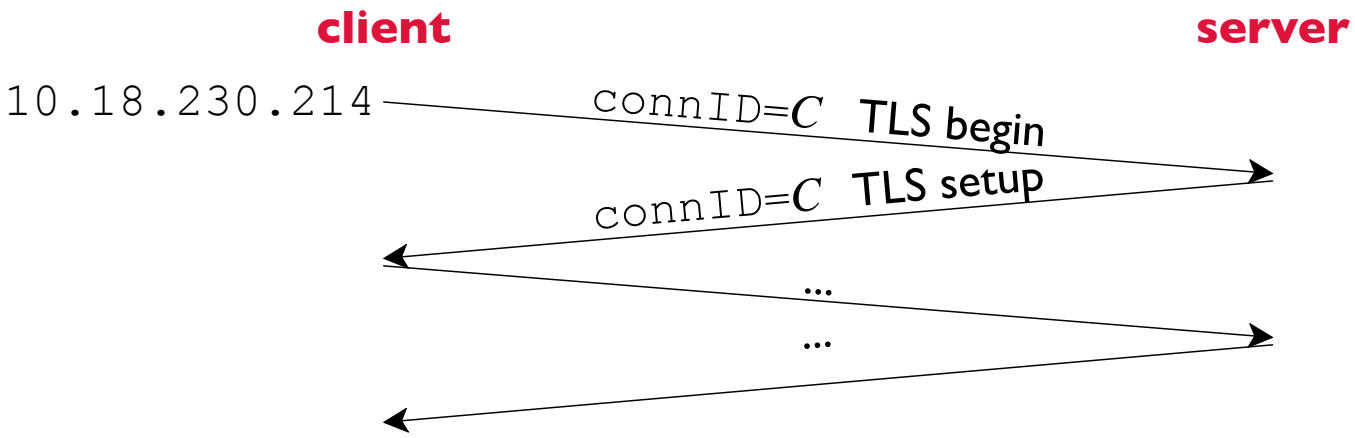
# QUIC

**QUIC**: Quick UDP Internet Connections

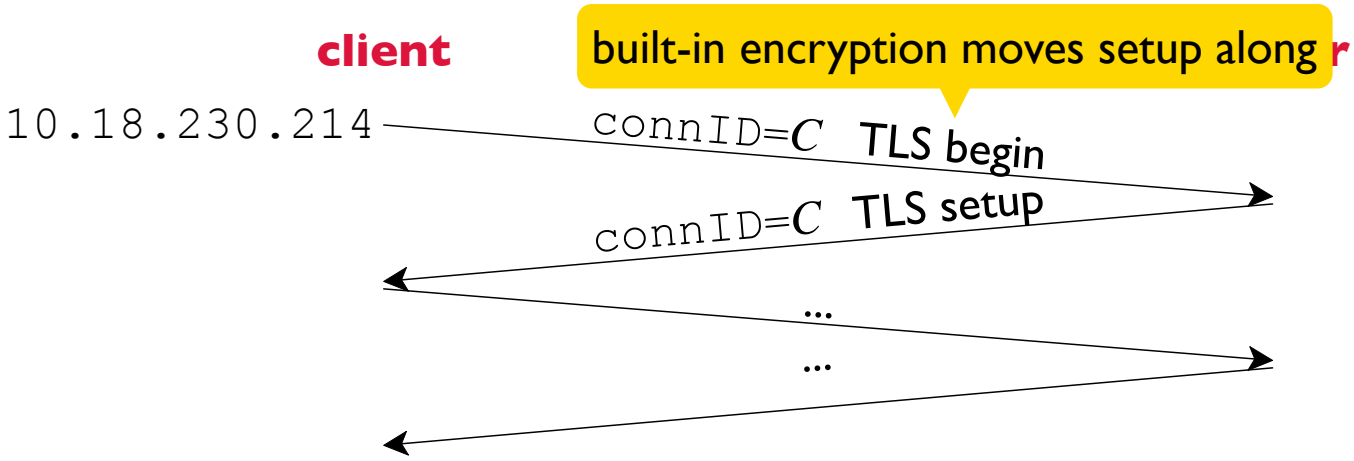
- builds on UDP
- connection-oriented based on a connection ID

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

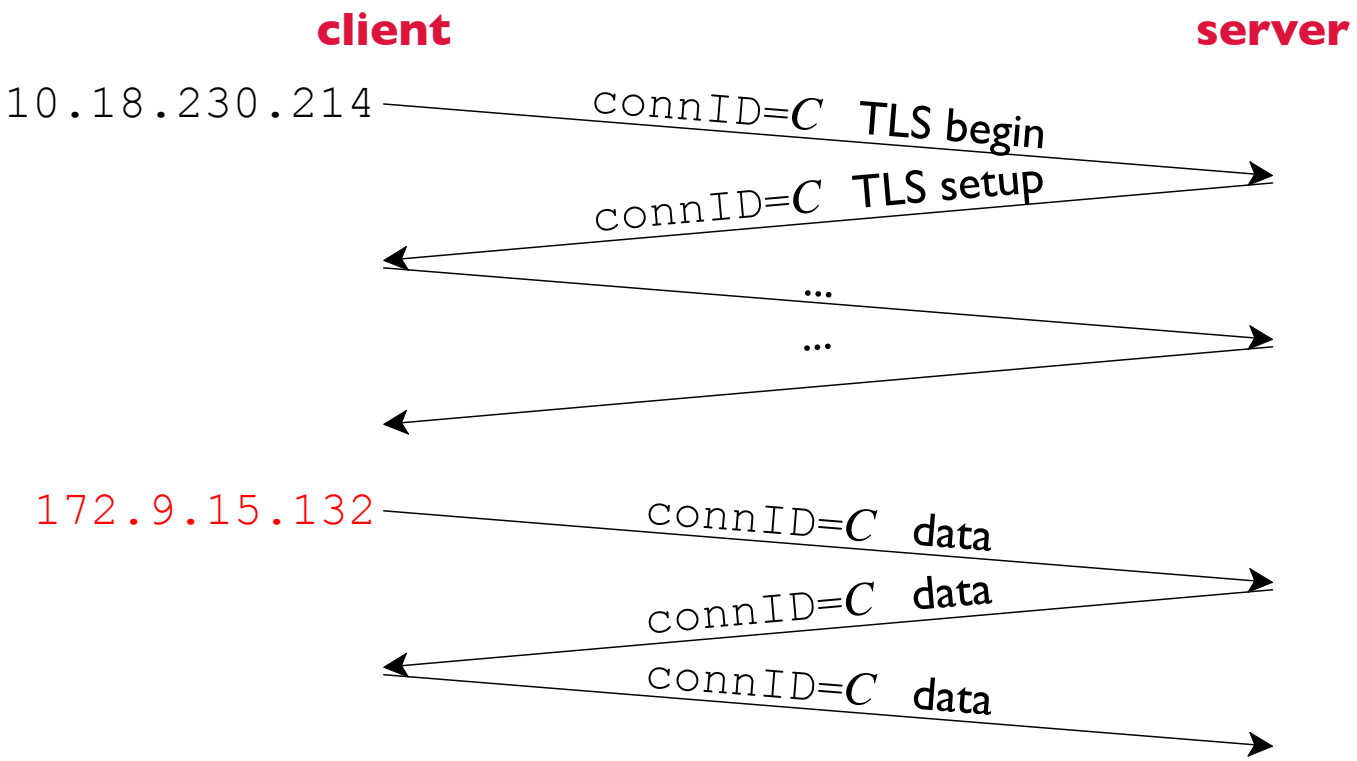
# QUIC


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

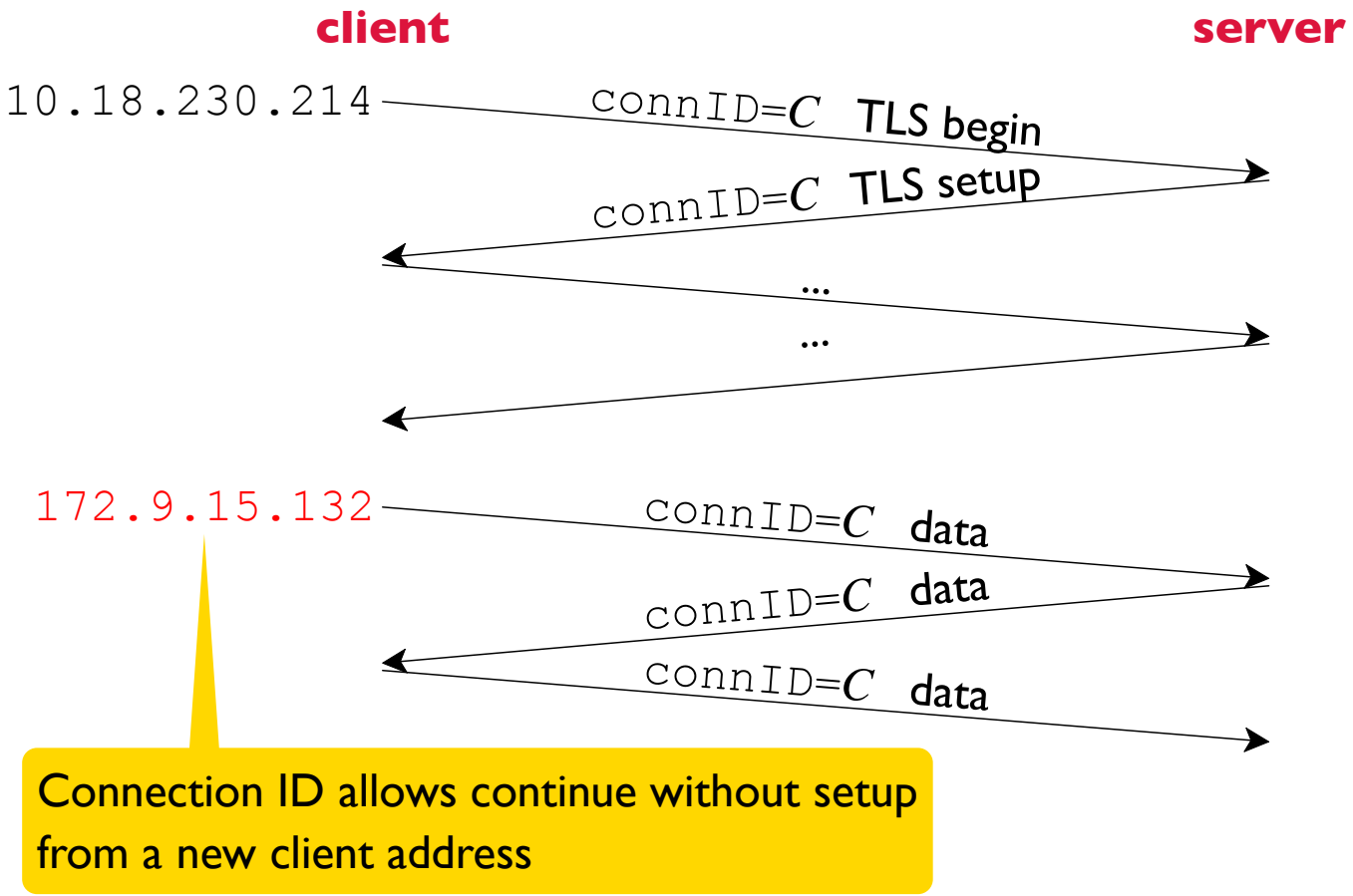
# QUIC


client
built-in encryption moves setup along r
10.18.230.214
connID=C TLS begin
connID=C TLS setup
...
...

# QUIC

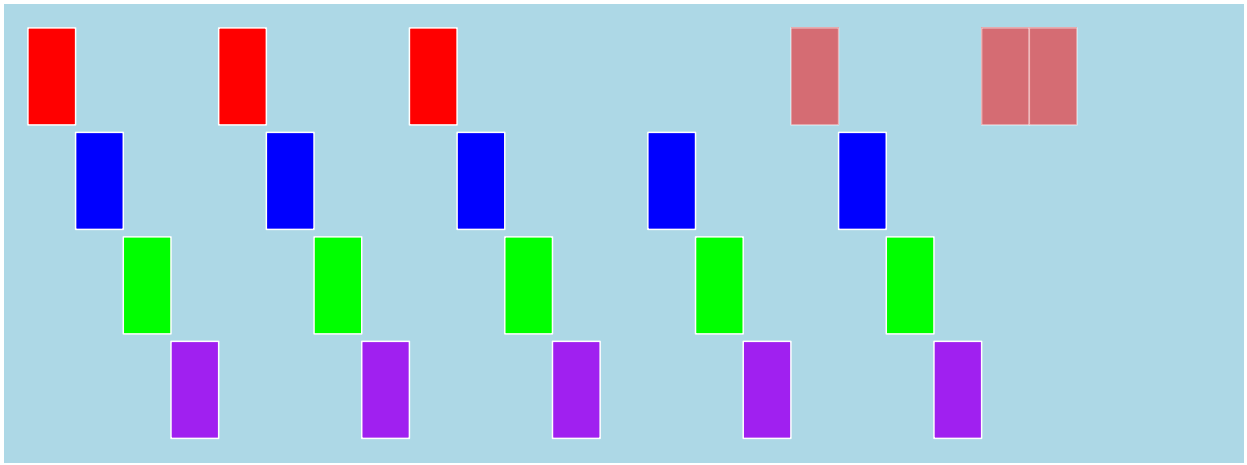

client
server
10.18.230.214
connID=C TLS begin
connID=C TLS setup
...
...
172.9.15.132
connID=C data
connID=C data
connID=C data

# QUIC


client
server
10.18.230.214
connID=C TLS begin
connID=C TLS setup
...
...
172.9.15.132
connID=C data
connID=C data
connID=C data
Connection ID allows continue without setup
from a new client address

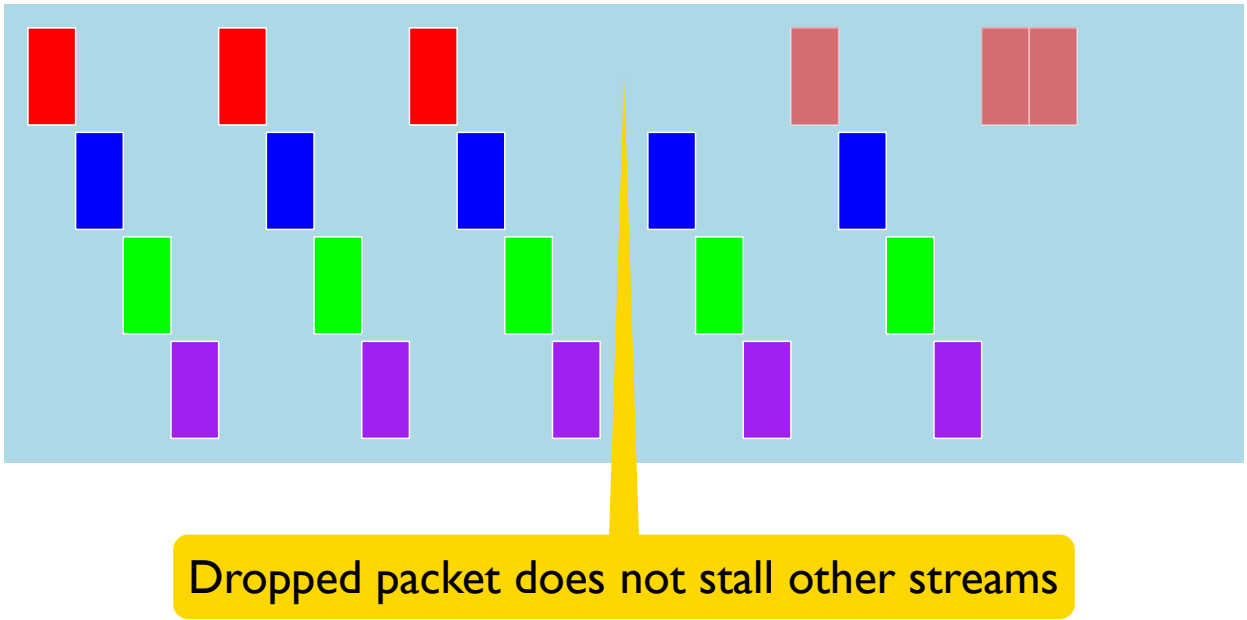
# QUIC

Concurrent streams are handled at the packet level within a connection



# QUIC

Concurrent streams are handled at the packet level within a connection



# Summary

**Flow control** avoids overwhelming the other end of a connection

**Congestion control** avoids overwhelming the network as a whole

TCP states for congestion control:

- **slow start**
- **congestion avoidance**
- **fast recovery**

TCP isn't always the best solution, it and can suffer from
**handshake hell**,
the **parking-lot problem**, and
the **head-of-line problem**.