Redes de Computadores

The Transport Layer

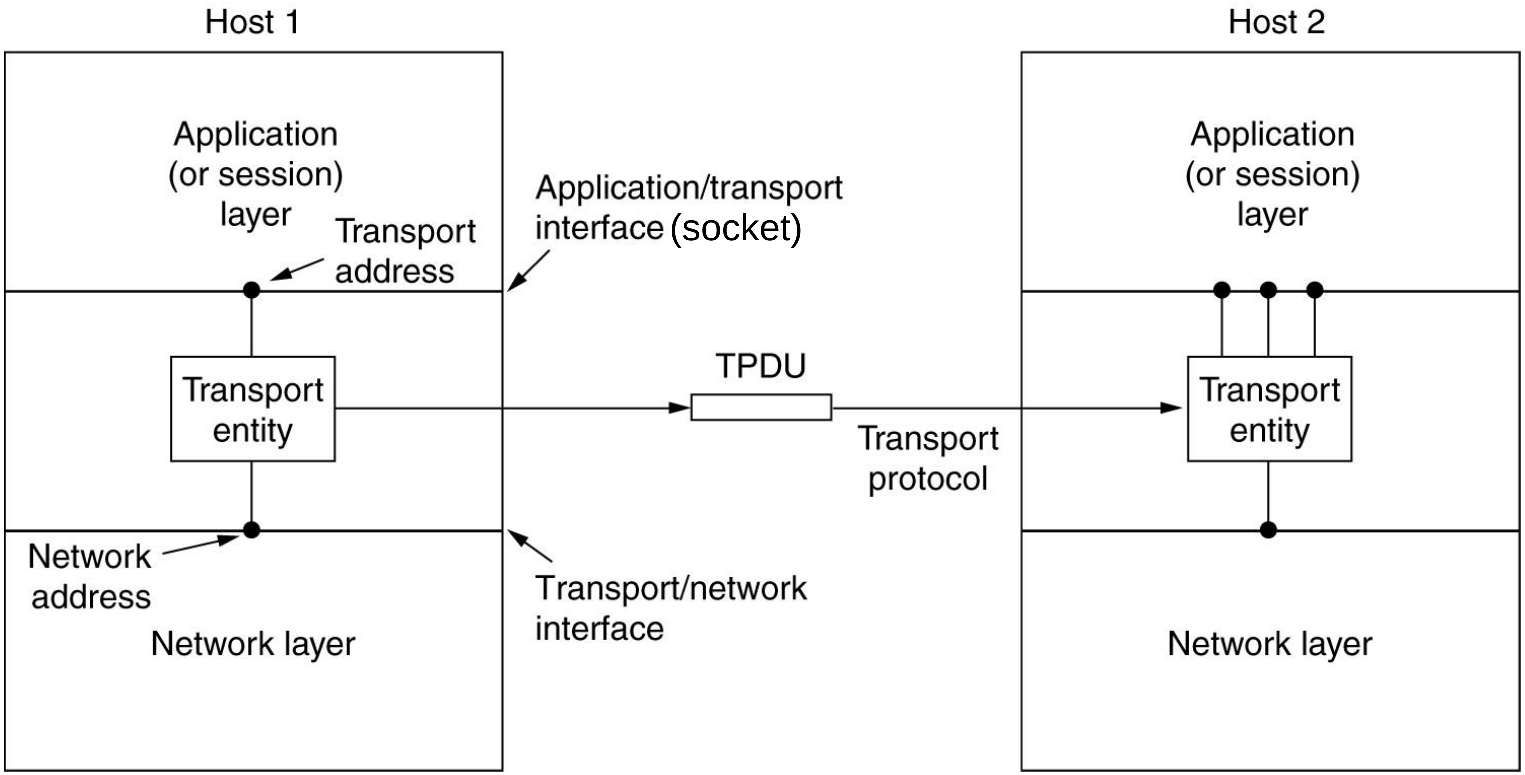
Manuel Ricardo, Rui Prior

Universidade do Porto

- *What are the services provided by the Transport Layer?*
- *What are the transport protocols in the TCP/IP stack?*
- *What are the differences between UDP and TCP?*
- *How is the connection established in TCP?*
- *What is the difference between flow control and congestion control?*
- *How does TCP implement flow control?*
- *What mechanisms does TCP adopt to prevent network congestion control?*
- *Why is it the congestion control mechanism implemented by TCP so important for the behaviour of the Internet?*

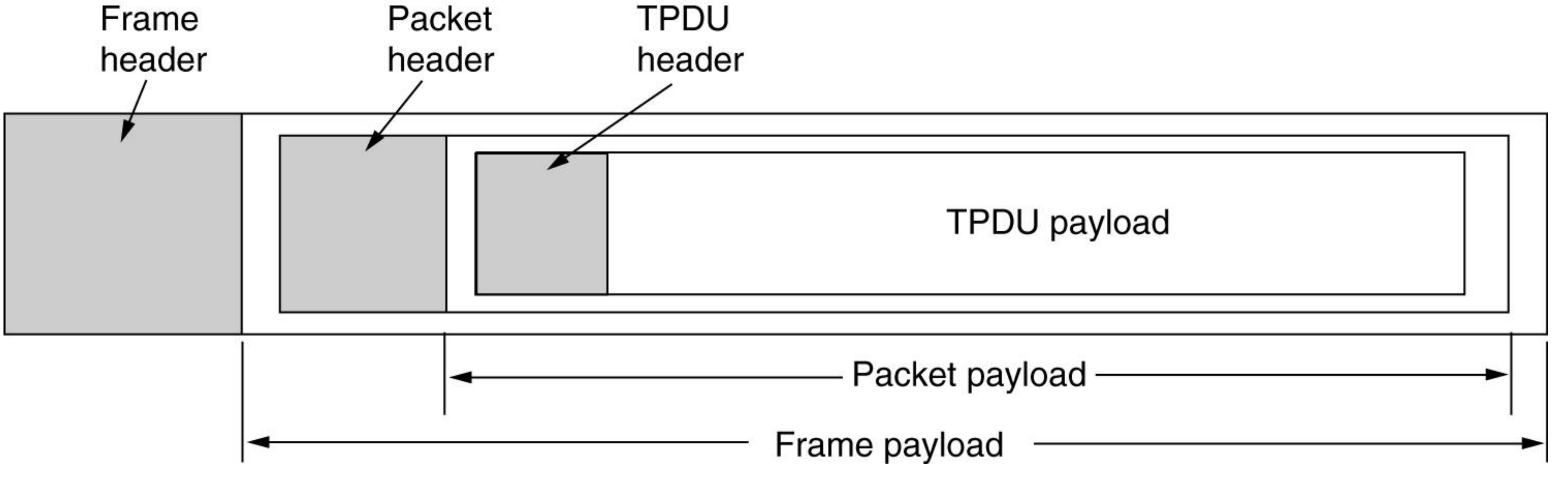
Services Provided to the Upper Layers

The network, transport, and application layers

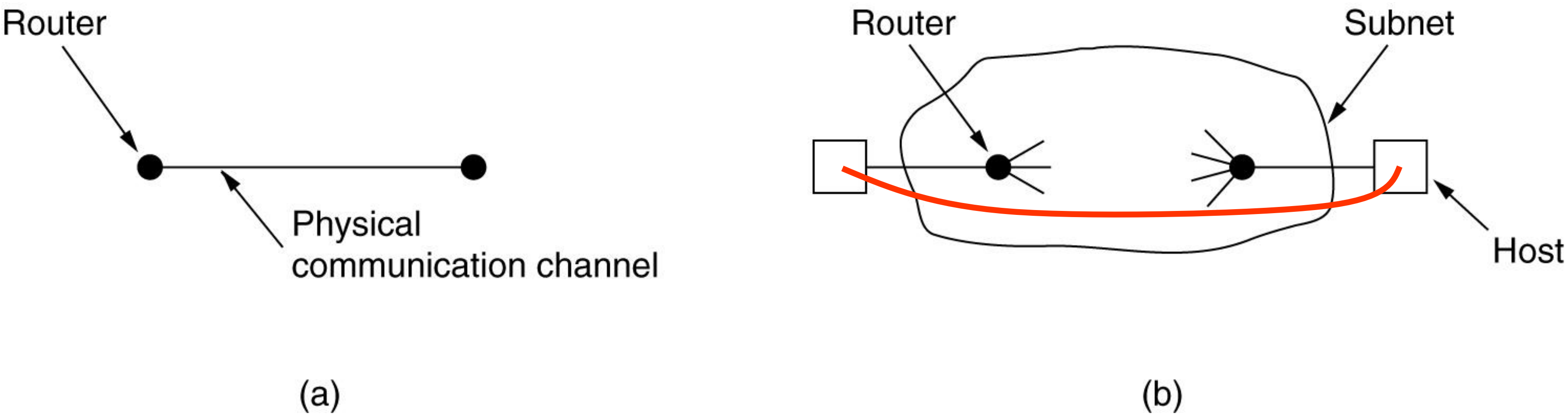


Transport Service Primitives

The nesting of TPDUs, packets, and frames



Transport Protocol

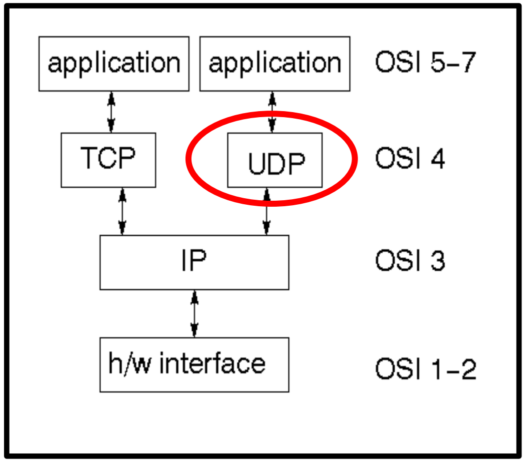


(a) Environment of the data link layer

(b) Environment of the transport layer

UDP - User Datagram Protocol (UDP)

- Datagram oriented
  - » Unreliable ➔ no error control mechanism
  - » Connectionless

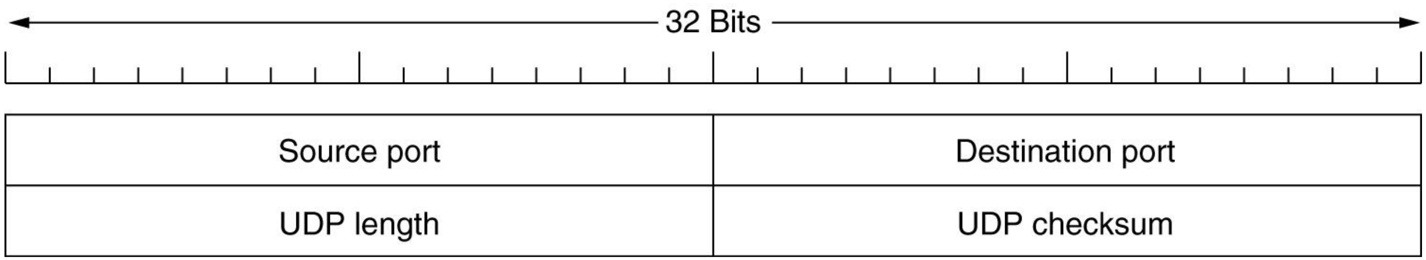


- Allows applications

  to interface directly to IP

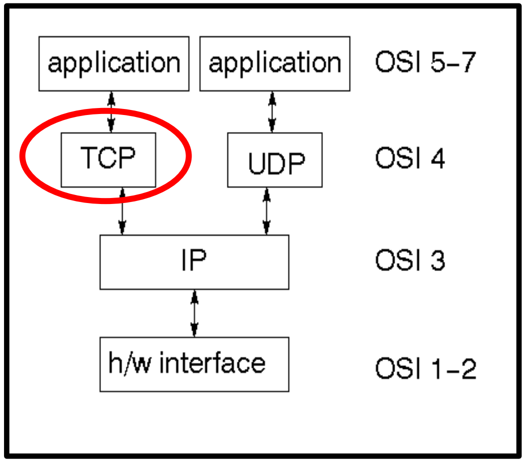
  with minimal additional protocol overhead

- UDP header
  - » Port numbers identify sending and receiving processes
  - » UDP length = length of packet in bytes
  - » Checksum covers header and data; optional

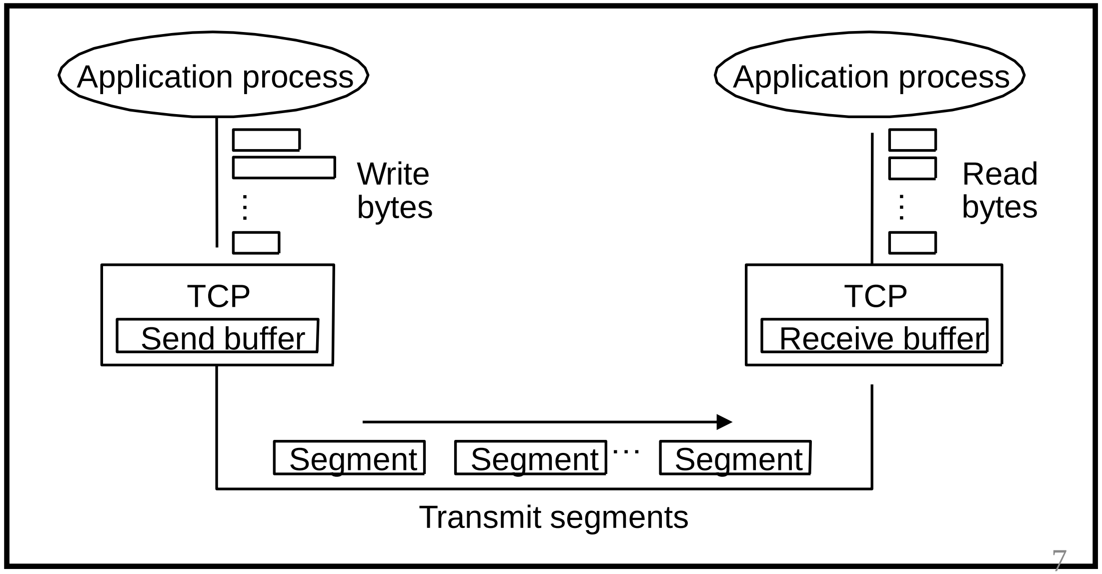


TCP – Transmission Control Protocol

- Connection oriented
- Full-duplex
- Byte stream



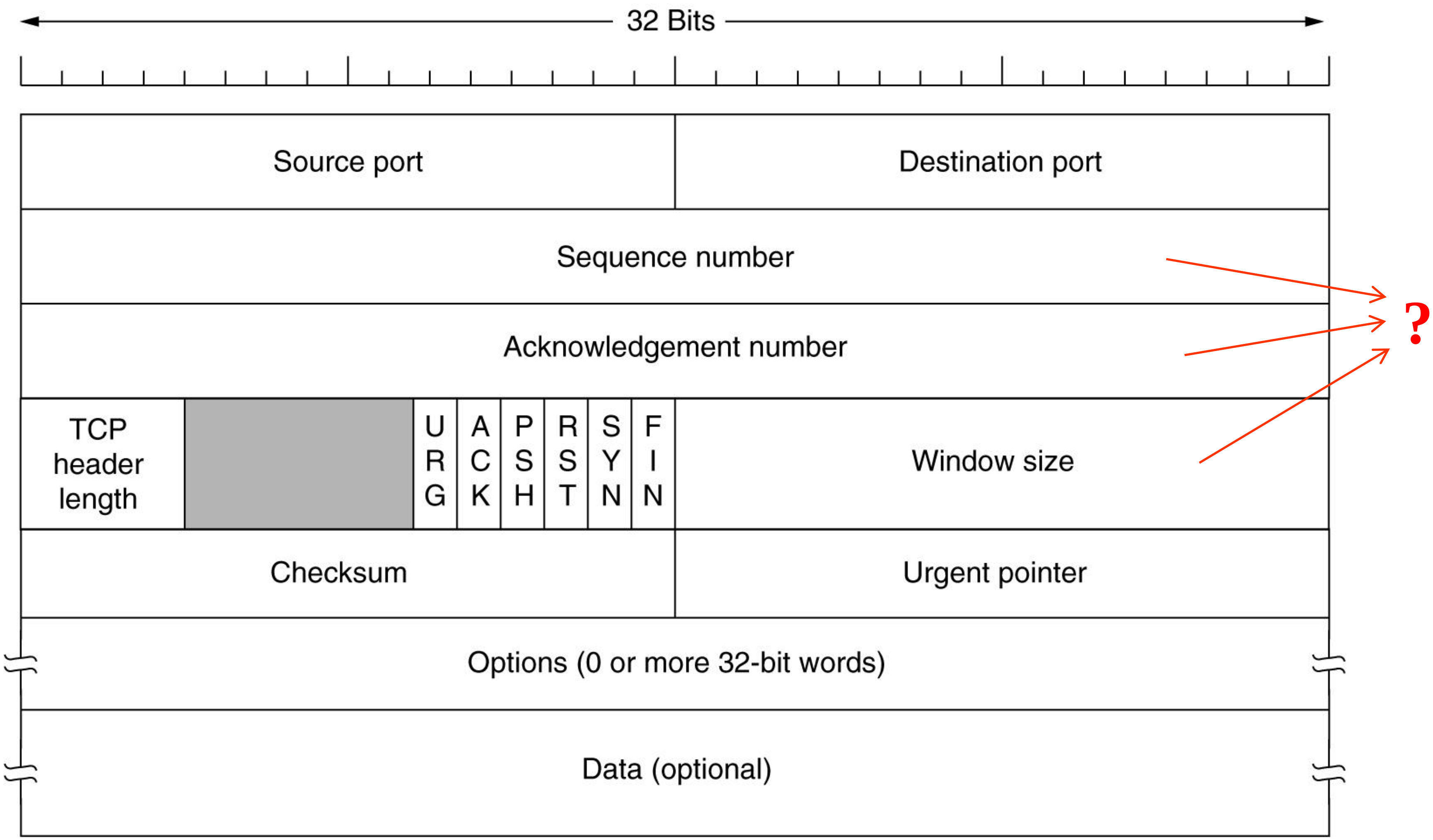
- Reliable
  - » ARQ mechanism
- Flow control
  - » Avoids overloading the receiver
- Congestion control
  - » Avoids overloading the network



Basic TCP Operation

- Sender
  - Application data is broken into segments
  - TCP uses timer while waiting for an ACK of every segment sent
  - Un-ACKed segments are retransmitted
- Receiver
  - Errors detected using a checksum
  - Correctly received data is acknowledged
  - Segments reassembled in proper order
  - Duplicated segments discarded
- Window-based flow control

The TCP Segment Header



TCP Header

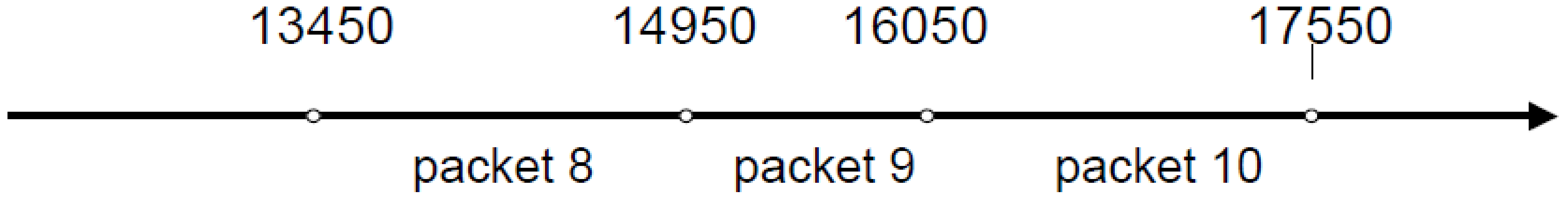
- Port numbers similar to UDP
- 32 bit SeqNumber uniquely identifies the application data contained in the TCP segment
  - » SeqNumber is in bytes
  - » Identifies the first byte of data
- 32 bit AckNumber is used for piggybacking ACKs
  - » AckNumber indicates the next byte the receiver is expecting
  - » Implicit ACK for all preceding bytes
- Window size
  - » Used for flow control (ARQ) and congestion control
    - Sender cannot have more than a window of bytes in the network
  - » Specified in bytes
    - Window scaling used to increase the window size in high-speed networks
- Checksum covers the header and data

Port numbers — TCP vs UDP

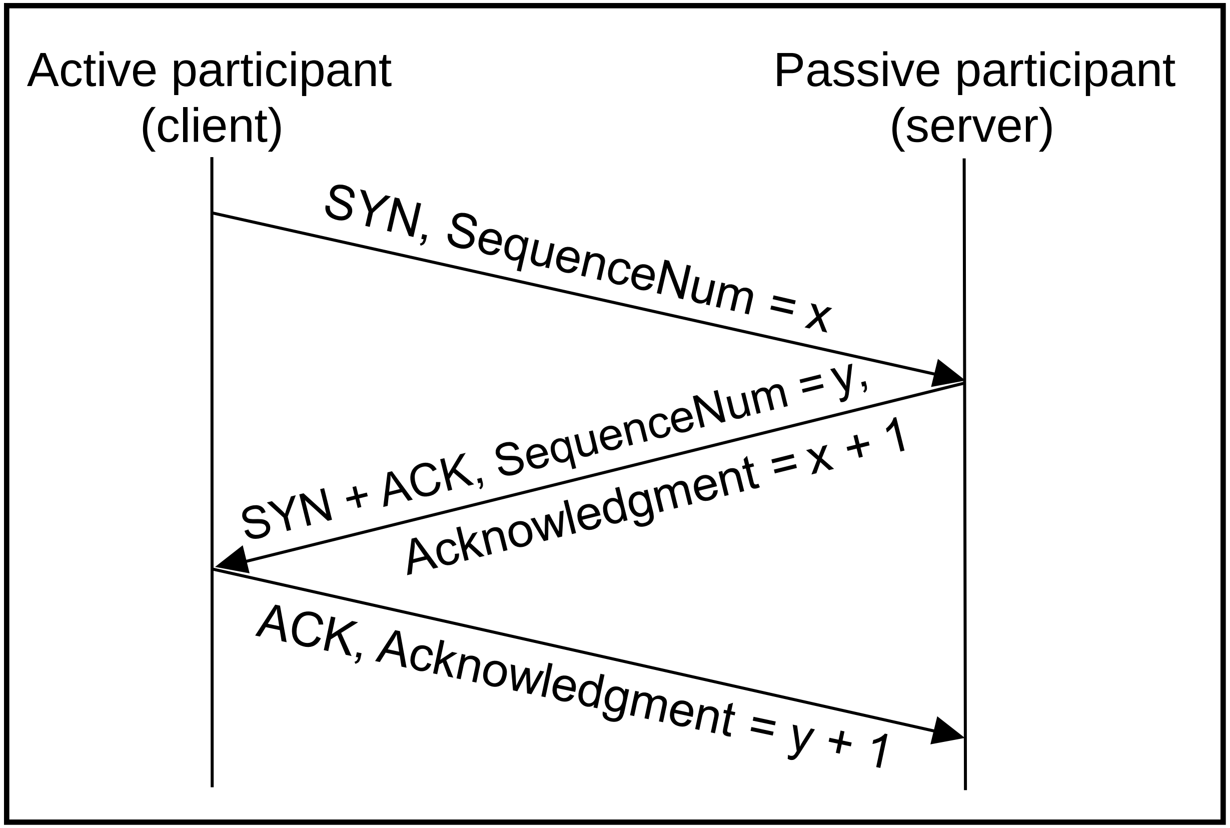
- Port numbers are treated differently in UDP and TCP
- UDP is connectionless
  - » A UDP socket is associated with a local IP address and port number
  - » All UDP packets destined for that IP and port are received by the socket
  - » The socket can be used to send UDP packets to any other IP and port
- TCP is connection-oriented
  - » A TCP connection socket is associated with both endpoints
    - – Local and remote IP addresses and ports
  - » Only TCP packets matching all four parameters are received by the socket
  - » Socket can only send TCP packets to the other endpoint

Sequence Numbers in TCP

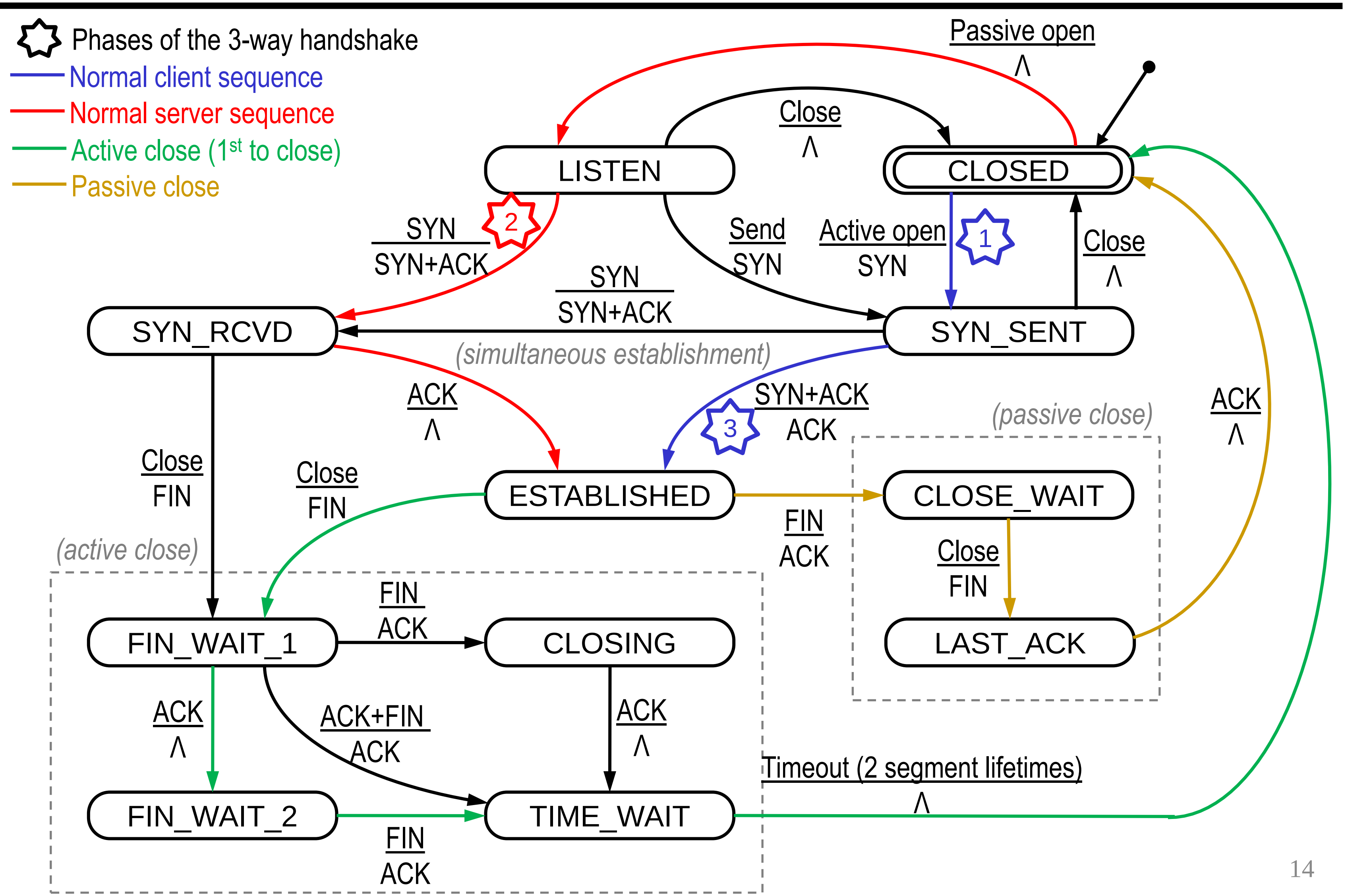
- TCP regards data as a *byte-stream*
  - » each byte in stream is numbered sequentially
- TCP breaks byte stream into segments
  - » size limited by the Maximum Segment Size (MSS)
- Each packet has a sequence number
  - » sequence number of the 1st byte of data transported by the segment
- TCP connection is duplex
  - » data in each direction has different sequence numbers



Connection Establishment

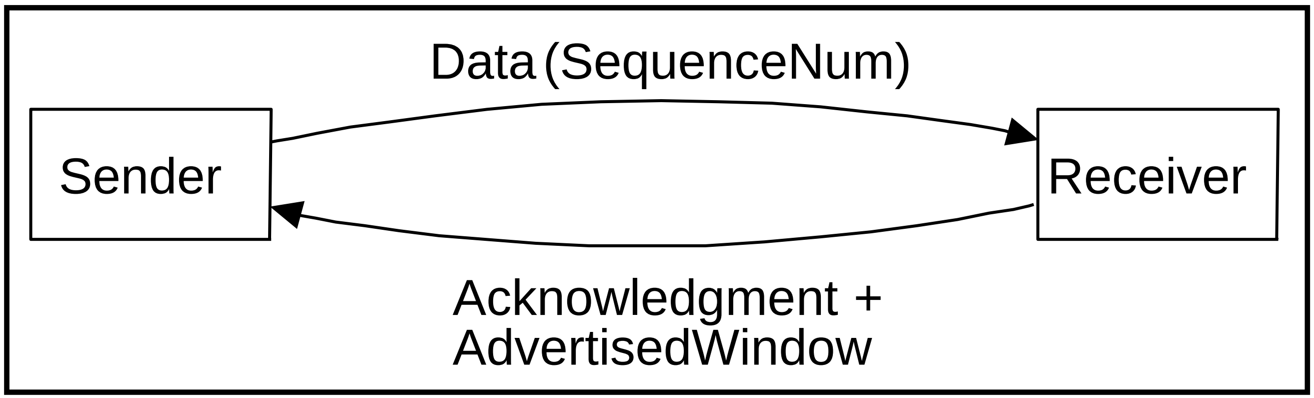


TCP Connection Management

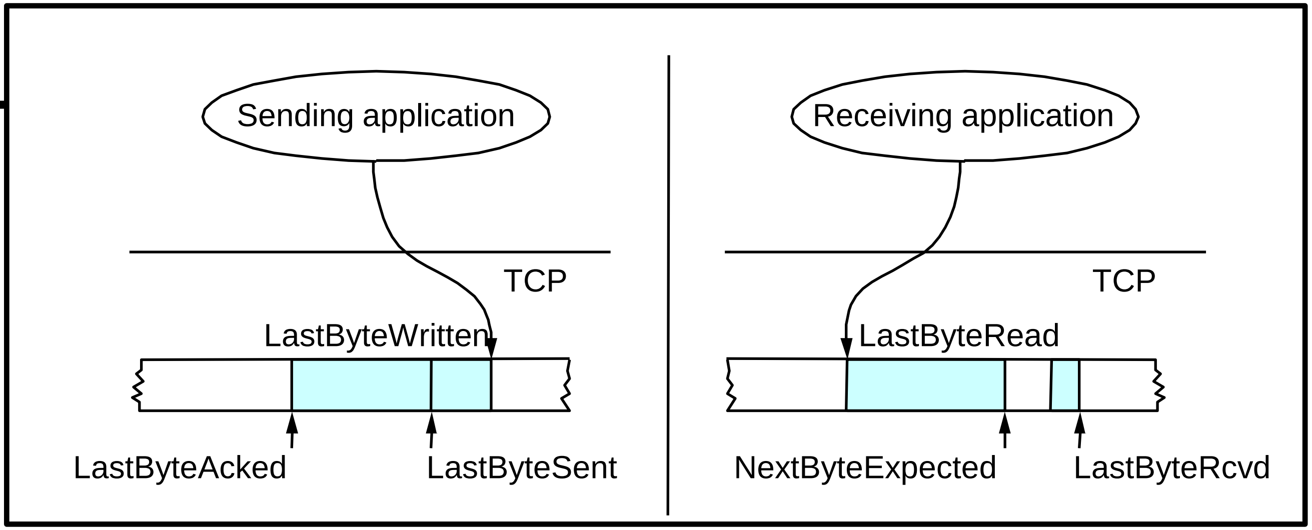


Retransmissions in TCP – A variation of Go-Back-N

- Sliding window
  - ACK contains a single sequence number
  - acknowledges all bytes with a lower sequence number
  - duplicate ACKs sent when out-of-order packet received
- Sender retransmits a single packet at a time
  - optimistic assumption → only one packet is lost
- Error control based on byte sequences, not packets



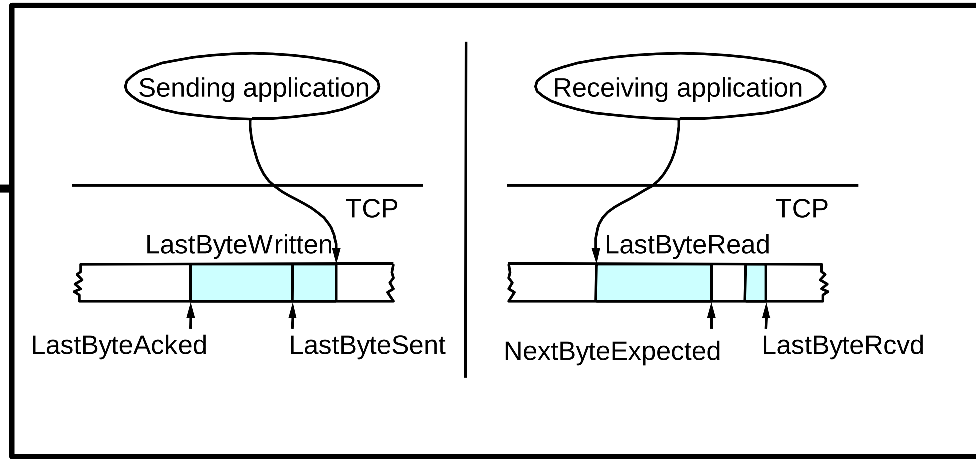
Sliding Window



» Sender
- **`LastByteAcked ≤ LastByteSent`**
- **`LastByteSent ≤ LastByteWritten`**
- Buffer holds bytes between **`LastByteAcked`** and **`LastByteWritten`**

» Receiver
- **`LastByteRead < NextByteExpected`**
- **`NextByteExpected ≤ LastByteRcvd + 1`**
- Buffer holds bytes between **`LastByteRead`** and **`LastByteRcvd`**

Flow Control



- Buffer length
  - Sender → **`MaxSendBuffer`**
  - Receiver → **`MaxRcvBuffer`**
- Receiver

  `LastByteRcvd - LastByteRead` ≤ **`MaxRcvBuffer`**

  $$\texttt{AdvertisedWindow} = \underbrace{\texttt{MaxRcvBuffer - (LastByteRcvd - LastByteRead)}}_{\text{Free space in buffer}}$$

- Sender

  `LastByteWritten - LastByteAcked` ≤ **`MaxSendBuffer`**

  `LastByteSent - LastByteAcked` ≤ **`AdvertisedWindow`**

  `EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)`

- Sending application blocks if it needs to write y bytes and

  `(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer`

- **`ACK`** sent when a segment is received

To Think…

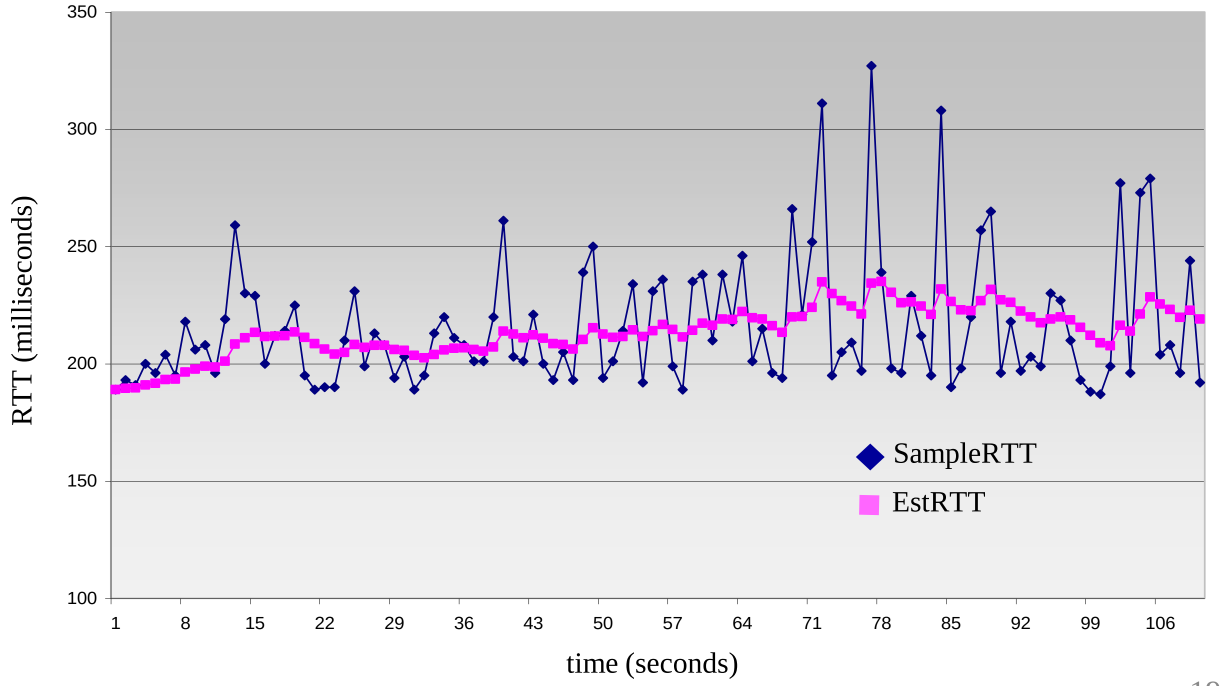
- TCP works on the Internet. How to determine a reasonable value for the retransmission timer?
- What happens if the selected value is unreasonably
  - low?
  - high?

Adaptive Retransmission

- Measure (sample) the Round Trip Time for each segment/ACK pair
- Compute **`EstRTT`** as an Exponentially Weighted Moving Average of the samples

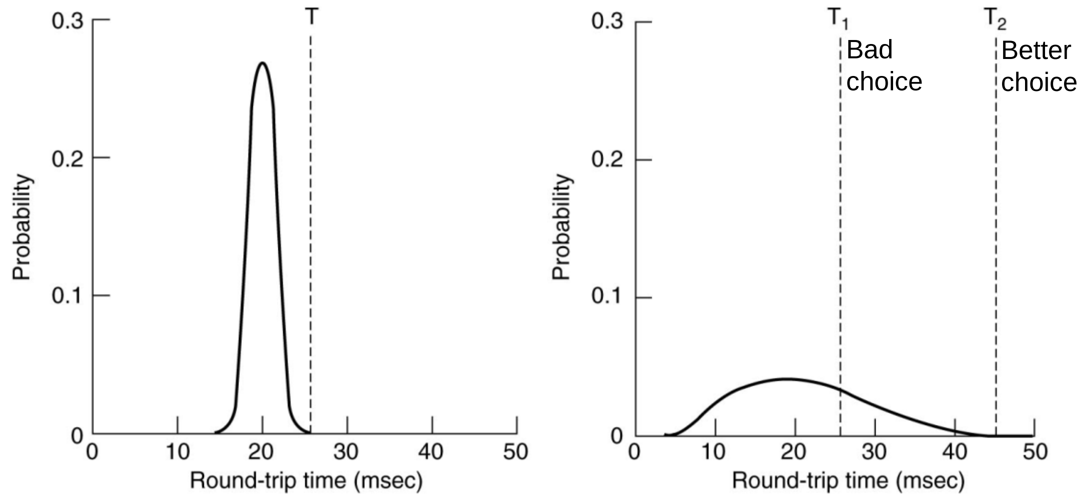
`EstRTT = (1-α)×EstRTT + α×SampleRTT`

α is usually 0.125



Adaptive Retransmission

- But the average is still not enough…



- Solution: also estimate the mean deviation

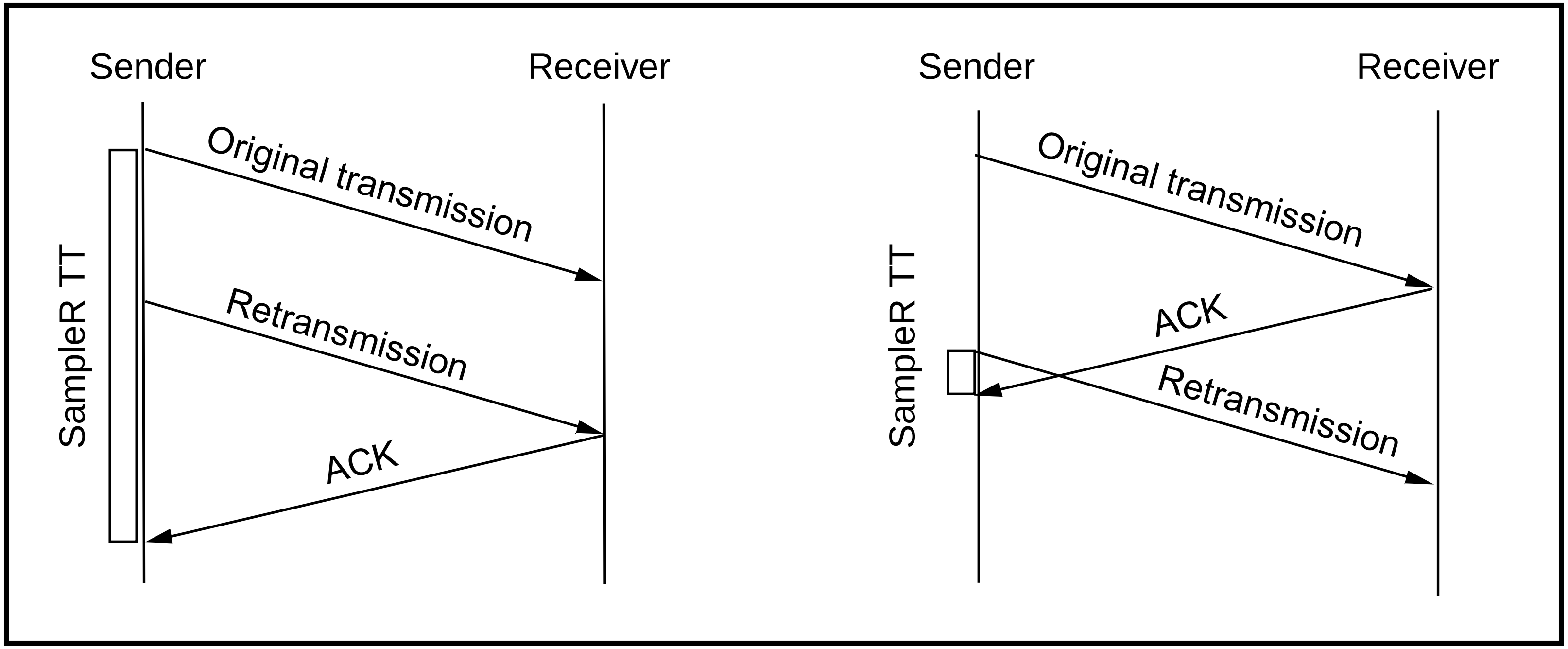
`DevRTT = (1-β)×DevRTT + β×|SampleRTT-EstRTT|`

β is usually 0.25

- Timeout value is **`TimeOut = EstRTT + 4×DevRTT`**

Karn/Partridge Algorithm

- `SampleRTT` not measured in retransmitted segments
  - » avoid ambiguity:



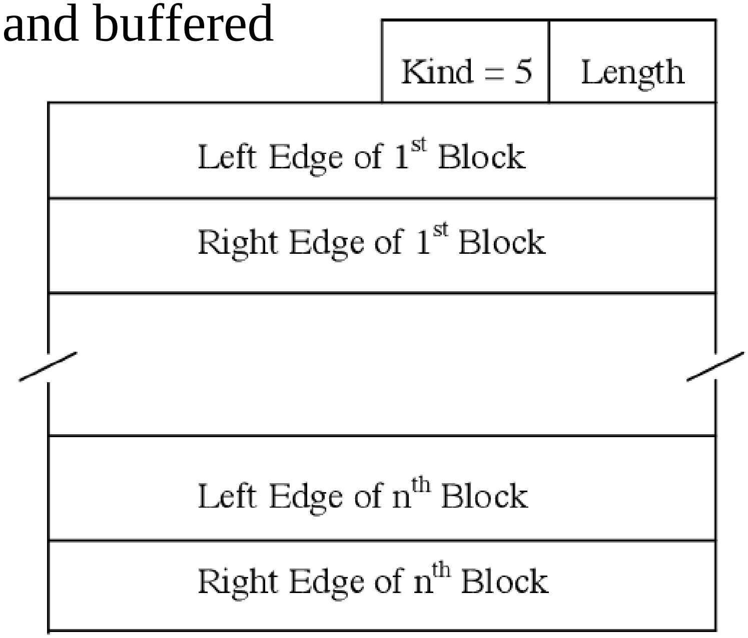
- Timeout doubled for each retransmission

Selective ACK

- Option for selective ACKs (SACK) also widely deployed

- Selective acknowledgement (SACK)
  - » list of blocks of data received out of order and buffered
  - » implemented as a TCP option
  - » negotiated during 3-way handshake

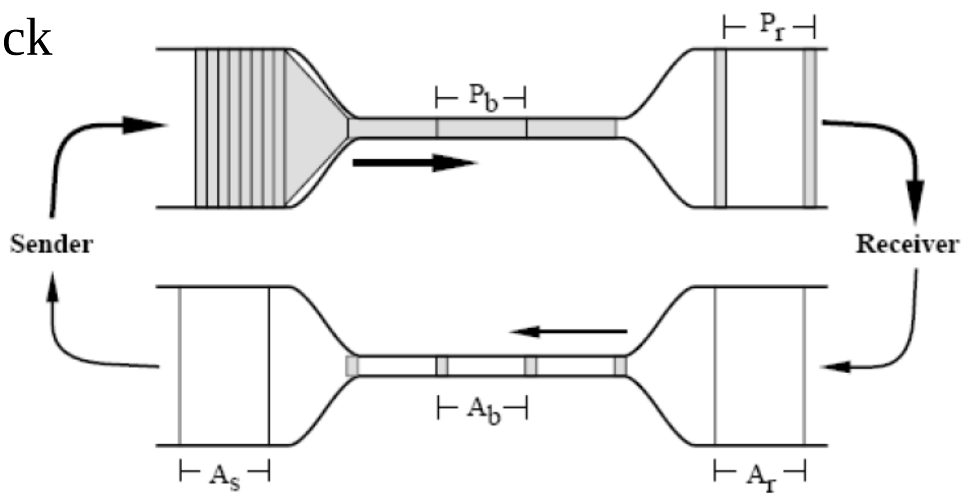
- When to retransmit?
  - » packets may experience different delays
  - » still need to deal with reordering
  - » wait for out of order by 3 packets



TCP SACK Option Format

TCP – Congestion Control

- Main idea
  - » each source determines its capacity
  - » based on criteria enabling
    - – flow fairness
    - – efficiency
- Received **ACKs** regulate packet transmission
  - ➔ they are the source clock



Additive Increase/Multiplicative Decrease

- Changes in channel capacity ➔ adjustment of transmission rate

- New variable per connection → **`CongestionWindow`**
  - » limits the amount of traffic in transit
    - – **`MaxWin = MIN(CongestionWindow, AdvertisedWindow)`**
    - – **`EffWin = MaxWin - (LastByteSent - LastByteAcked)`**

- Objective
  - » If network congestion decreases ➔ **`CongestionWindow`** increases
  - » If network congestion increases ➔ **`CongestionWindow`** decreases

- **`Bitrate (Bytes/s)`** ➔ **`CongestionWindow/RTT`**

Additive Increase/Multiplicative Decrease

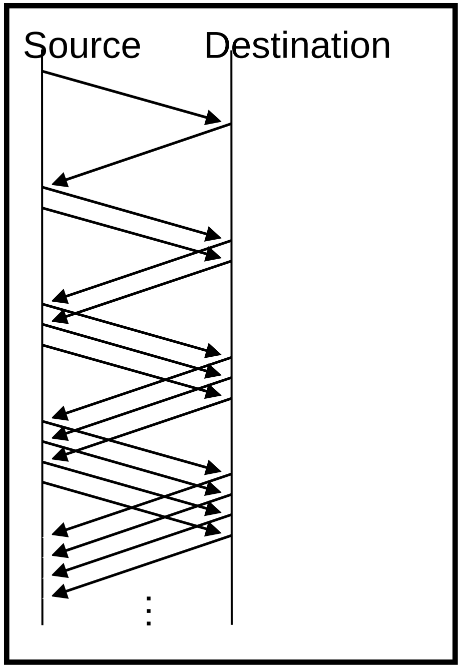
How does the source know if/when the network is in congestion?

➔ Packet losses!

- » packet loss ➔ buffer in router is full ➔ congestion

Additive Increase/Multiplicative Decrease

- Algorithm
  - » increase **`CongestionWindow`** by **`1`** segment
    - – for each **RTT** (Round Trip Time) ➔ additive increase
  - » divide **`CongestionWindow`** by **2**
    - – when there is a packet loss ➔ multiplicative decrease

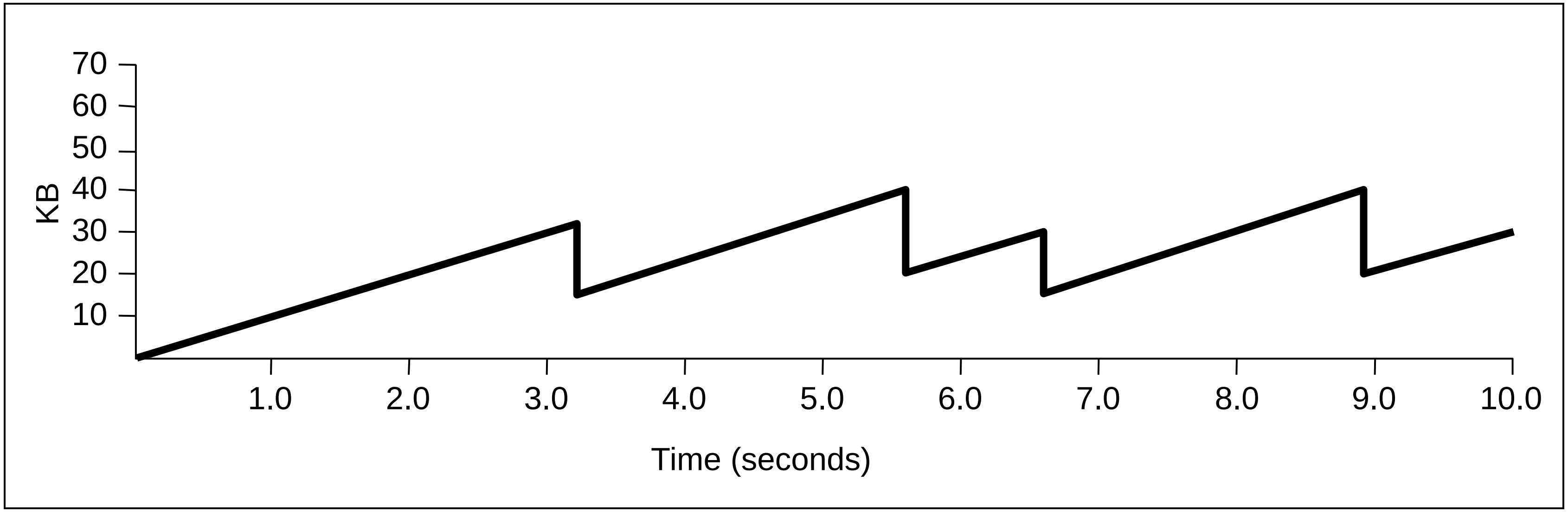


- In practice,
  - » Increases by ACK received
  - » **`Increment = MSS * (MSS / CongestionWindow)`**
  - » **`CongestionWindow += Increment`**

`MSS` → Maximum Segment Size

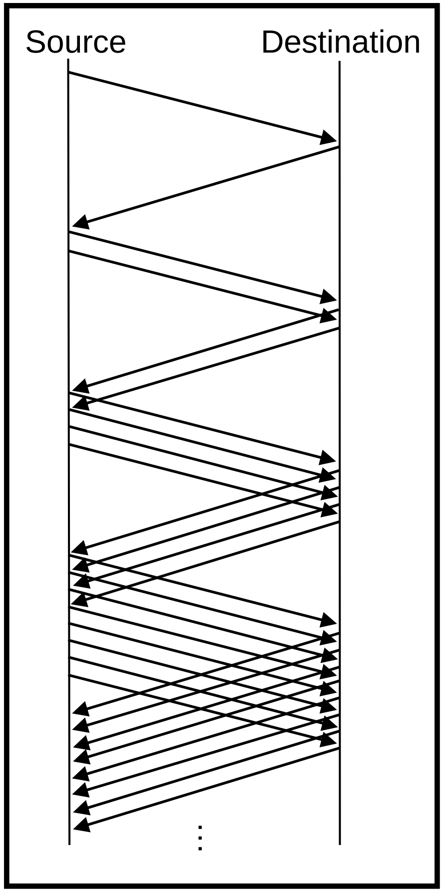
Additive Increase/Multiplicative Decrease

Saw-tooth behavior

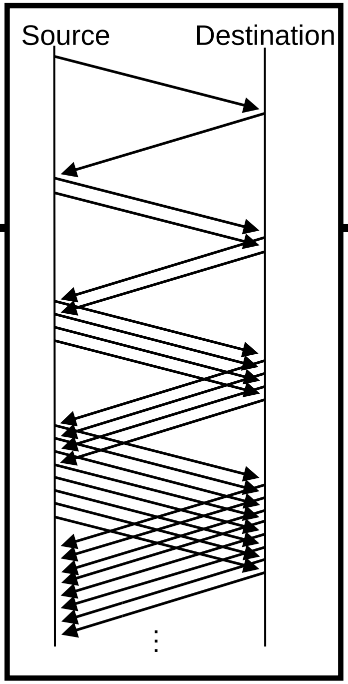


Slow Start ☺

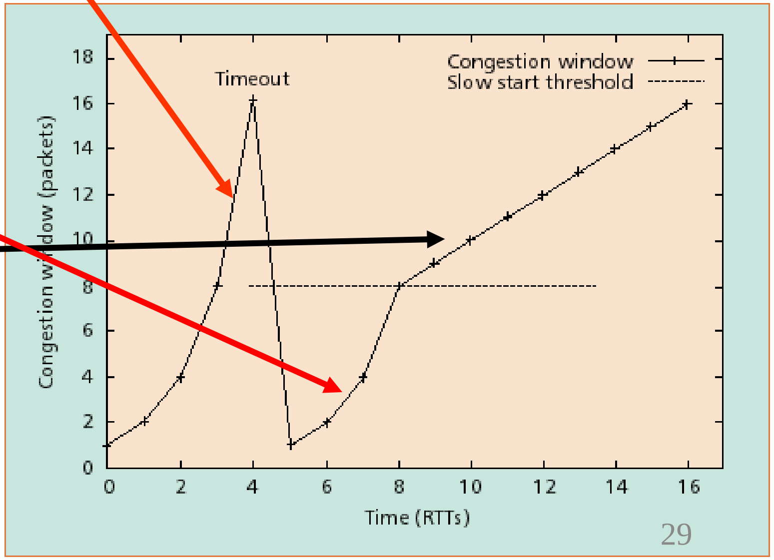
- Objective
  - » determine the available capacity
    quickly
    in a new connection or when the capacity changes
- Behavior
  - » start by **`CongestionWindow = 1 MSS`**
  - » double **`CongestionWindow`** each RTT
    by adding 1 MSS per ACK received
- Exponential growth in effect
  - » window duplicates per RTT
  - » start slow but grow fast



TCP – Slow Start



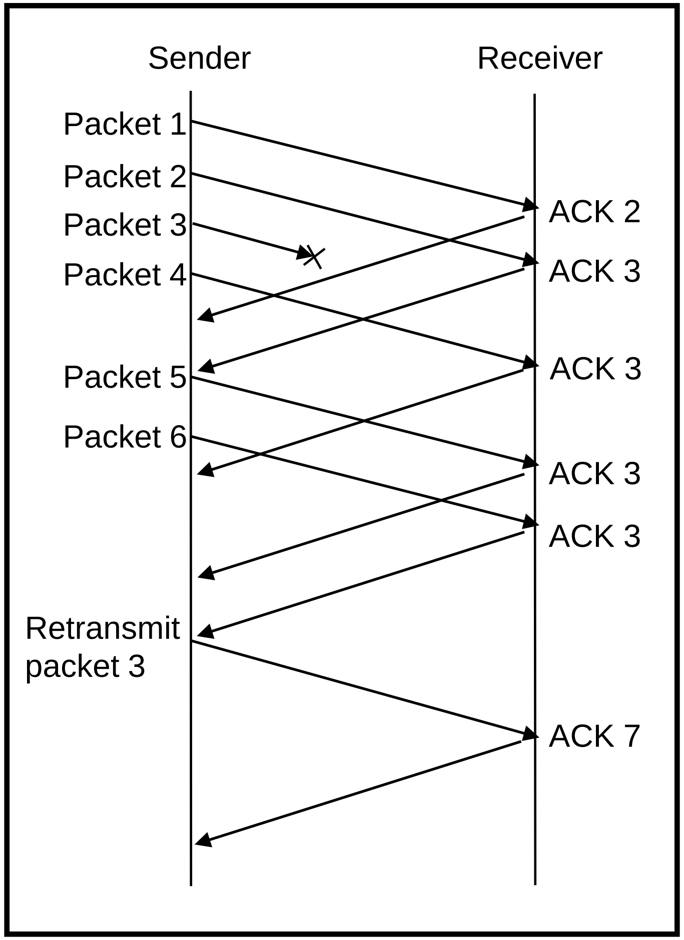
- ***Slow Start***
  - » Sender starts with **`congestionWindow = 1 MSS`**
  - » Doubles **`congestionWindow`** each **`RTT`**
- When a segment loss is detected, by timeout
  - » **`threshold = ½ congestionWindow(*)`**
  - » **`congestionWindow = 1 MSS`**
    (router gets time to empty queues)
  - » Lost packet is retransmitted
  - » ***Slow start*** while
    `congestionWindow < threshold`
  - » Then → ***Congestion Avoidance*** phase



(*) - in fact FlightSize, the amount of outstanding data

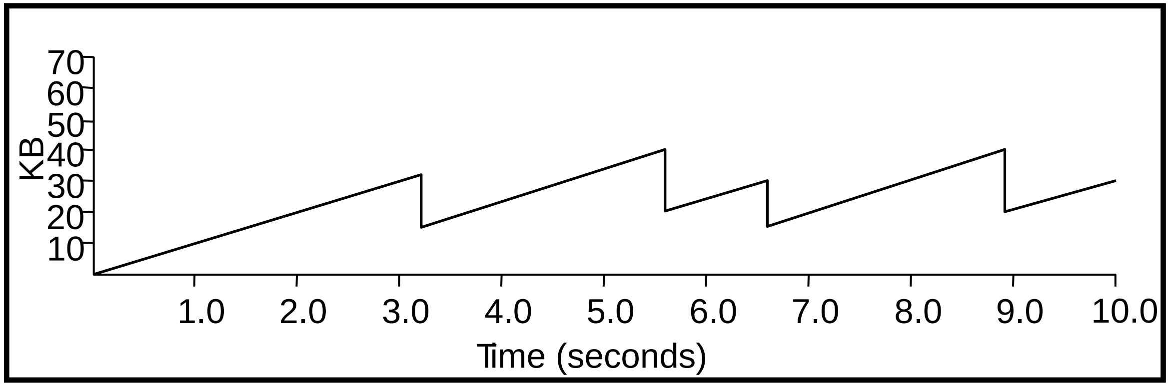
Fast Retransmission

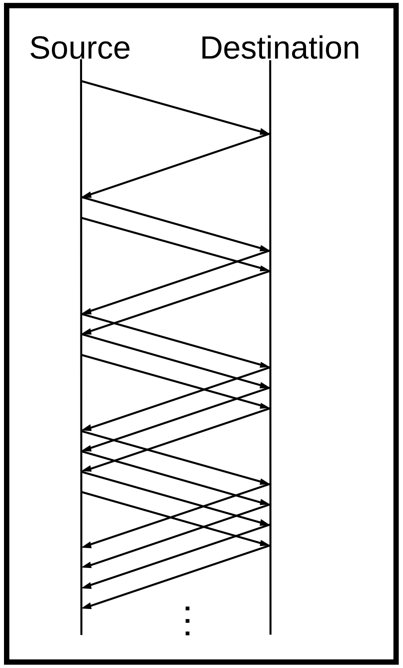
- Problem
  - » if TCP timeout is large
    - → long inactivity period
- Solution
  - » fast retransmission
    - → after 3 repeated ACKs



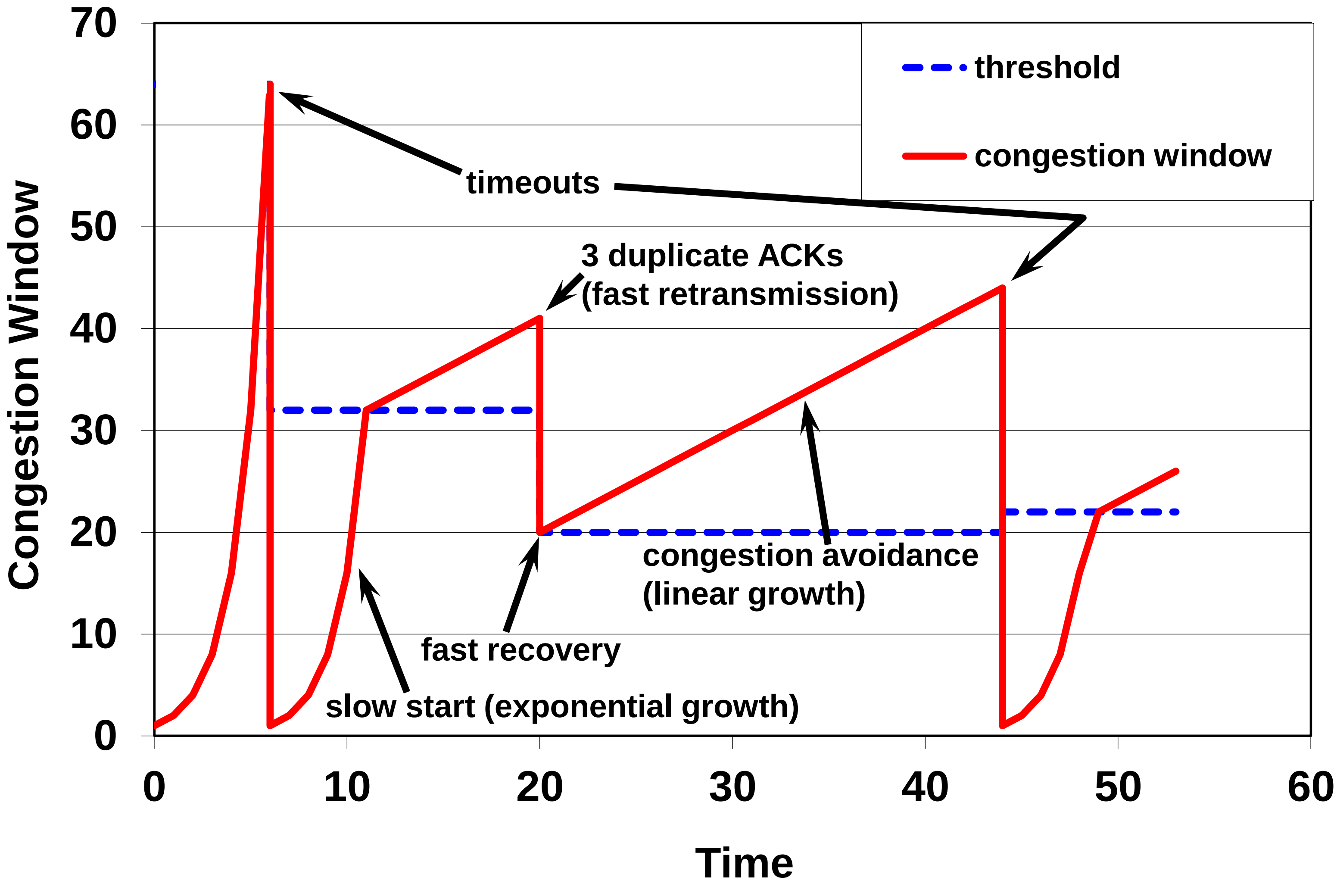
Congestion Avoidance

- ***Congestion Avoidance*** (additive increase)
  - » increments **`congestionWindow`** by 1 MSS per RTT
- Detection of segment loss, by reception of 3 duplicated ACKs
  - » Assumes packet is lost,
    - – Not by severe congestion, since following segments have arrived
  - » Retransmits lost packet
  - » **`congestionWindow = congestionWindow/2`**
  - » ***Congestion Avoidance*** phase





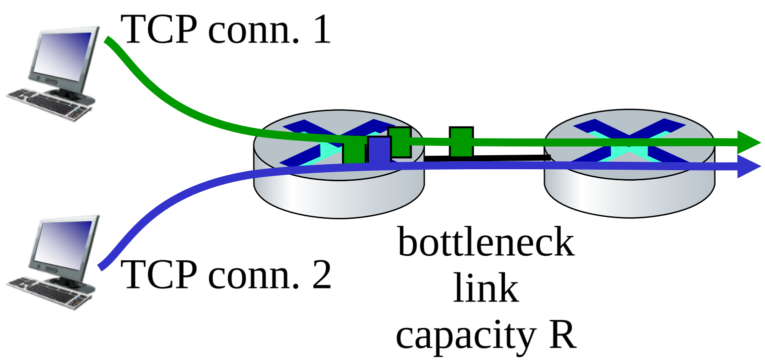
Congestion Control Summary (TCP Reno)



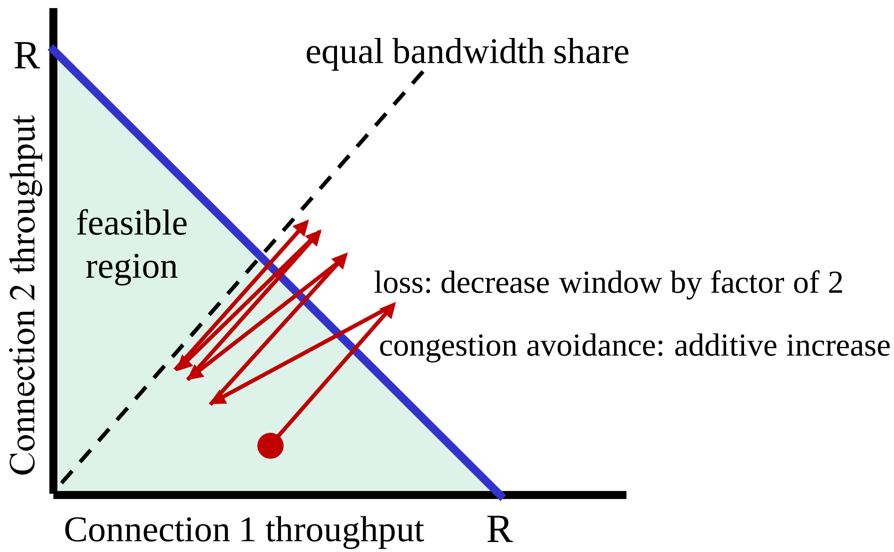
TCP – Congestion Control

- In reality, a bit more complex
- RFC 2581, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”
- Many TCP variants differing on the congestion control algorithm

TCP Fairness



- Example: two competing TCP sessions
- Ideally, each session gets *R*/2
- Additive increase gives slope of 1, as throughput increases
- Multiplicative decrease decreases throughput proportionally (midpoint to origin)



Is TCP fair?

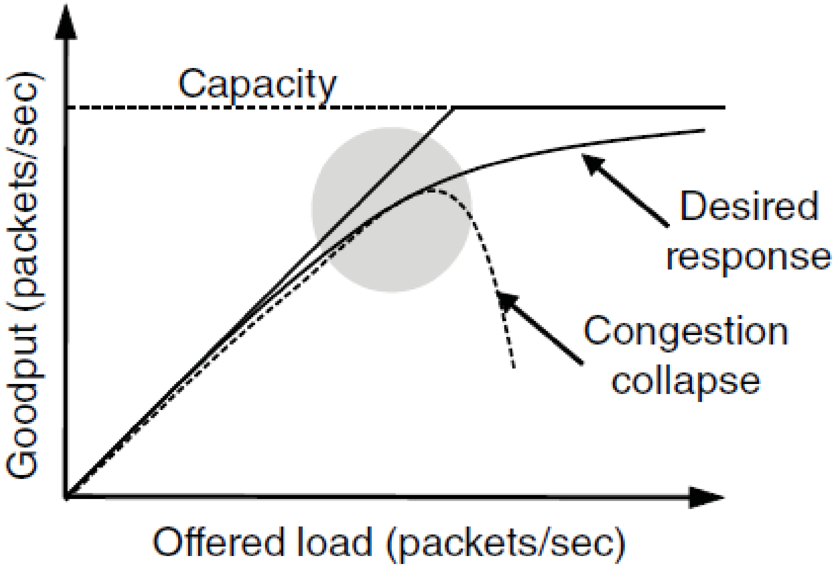
Yes, under idealized assumptions:

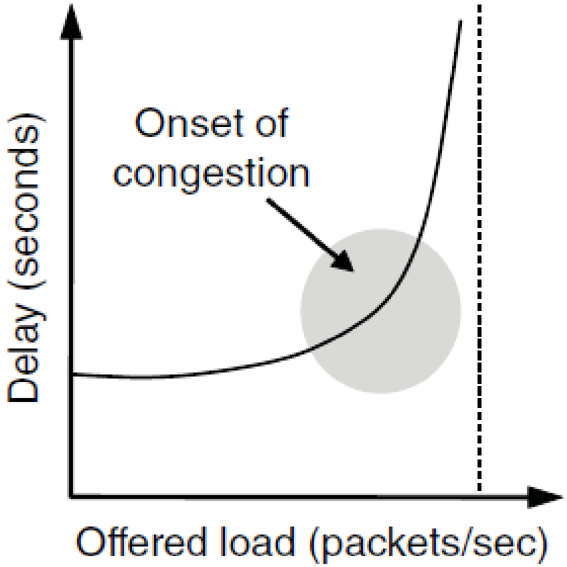
» same RTT

» fixed number of sessions, only in congestion avoidance

Desirable Bandwidth Allocation

Efficient use of bandwidth gives high goodput, low delay





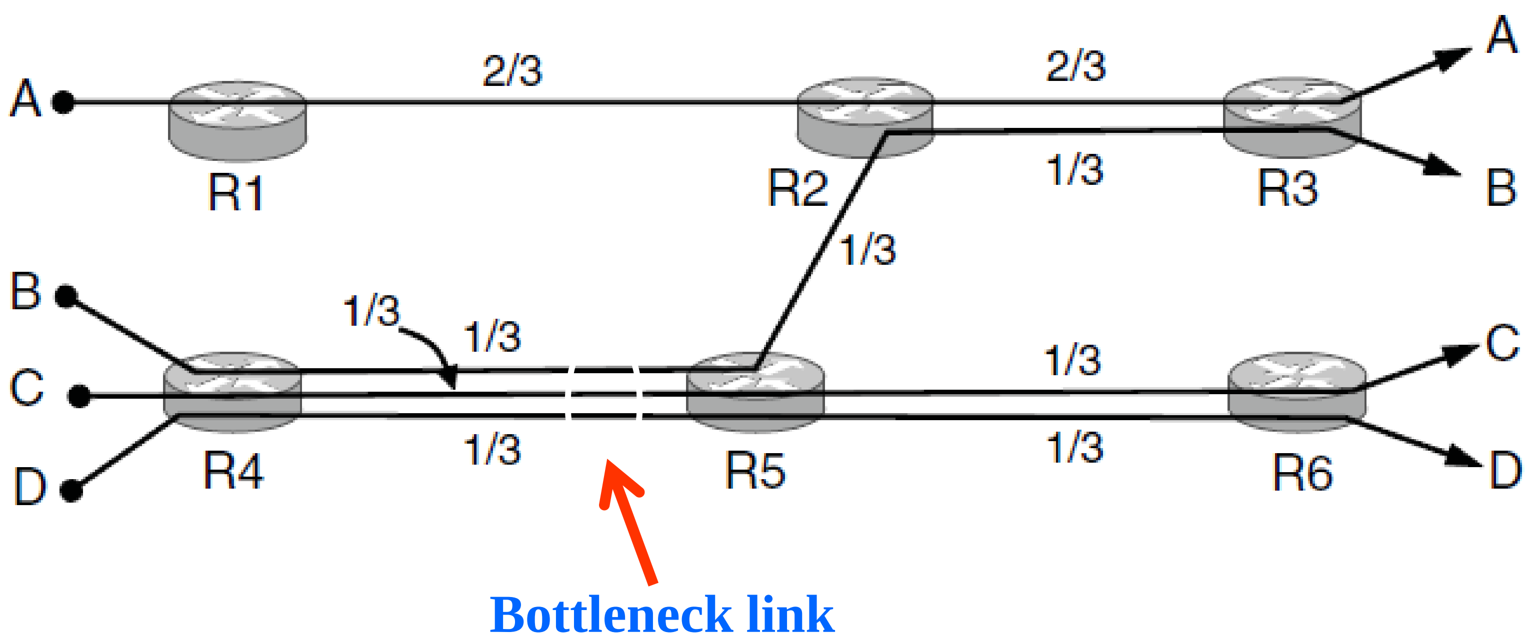
Goodput rises more slowly than load when congestion sets in

Delay begins to rise sharply when congestion sets in

Desirable Bandwidth Allocation –
Max-min fairness

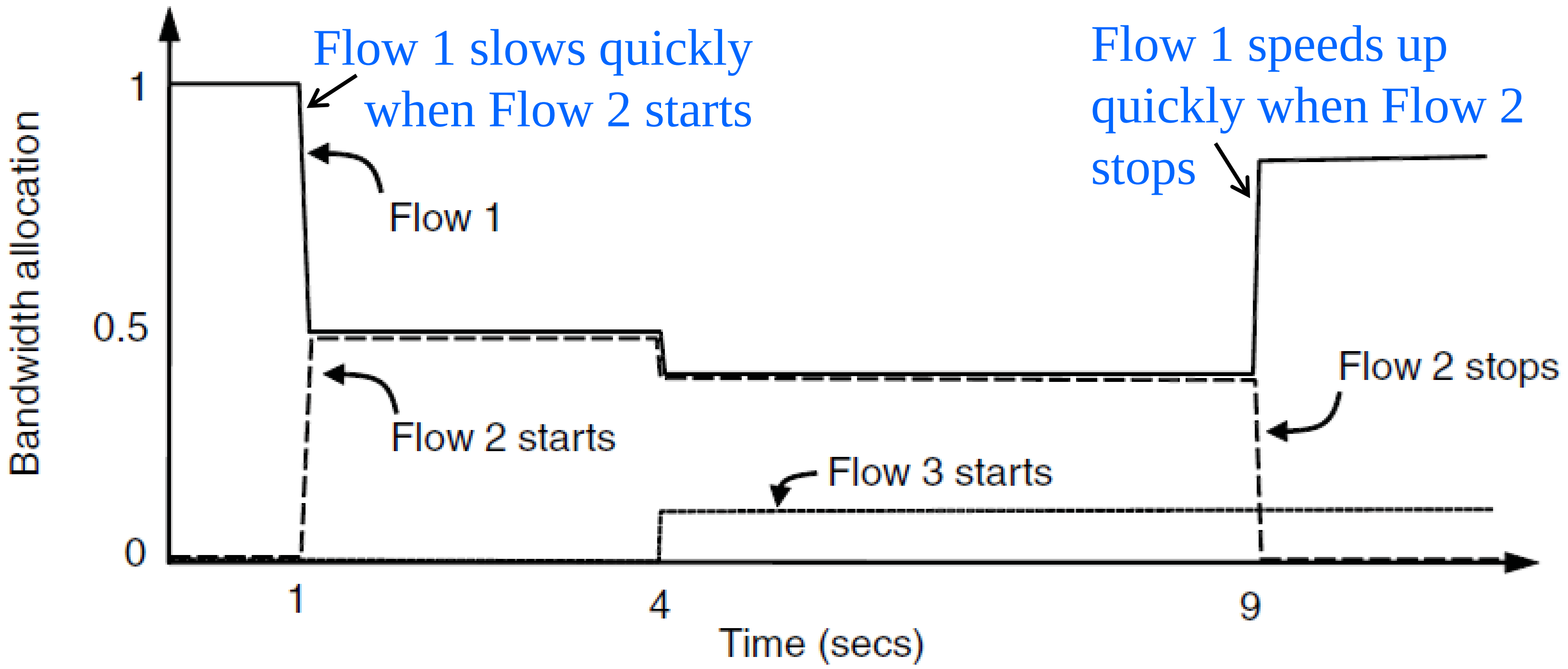
Fair use gives bandwidth to all flows (no starvation)

» **Max-min fairness** gives equal shares of bottleneck



Desirable Bandwidth Allocation – Bitrates along the time

Bitrates must converge quickly when traffic patterns change



Homework

1. Review slides

2. Read from Tanenbaum
   - » Chapter 6 – The Transport Layer

3. Answer questions at moodle