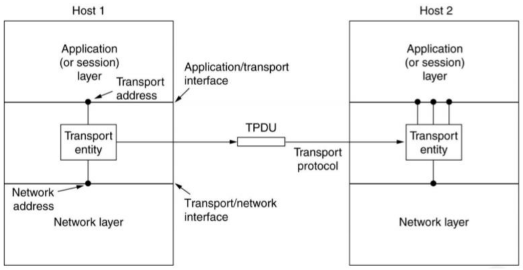
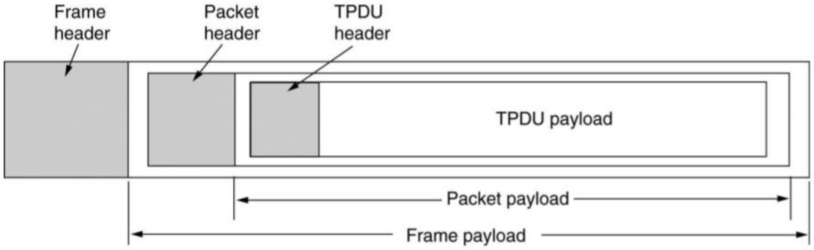
0.1 Services provided to the uppper layers



0.2 Transport Service Primitives

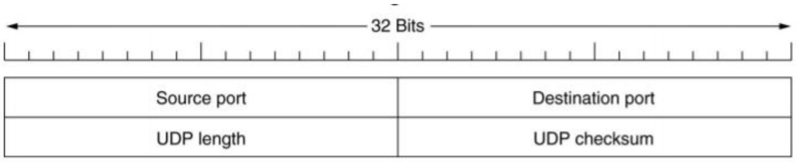


0.3 UDP - User Datagram Protocol

Datagram oriented
Unreliable because 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

0.4 TCP - Transmission Control Protocol

Flow Control
Reliability
ARQ mechanism (error control with ACK)
Avoids receiver's congestion
Congestion Control



Source port	Destination port
UDP length	UDP checksum

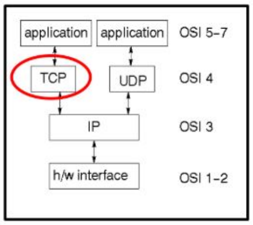
Avoids network's congestion

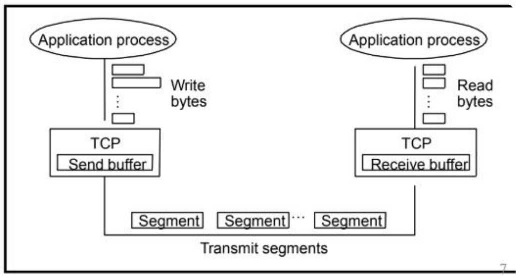
Properties

Connection oriented

Full-duplex

Byte stream





0.5 Basic TCP Operation

-Sender

Application data is broken in 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

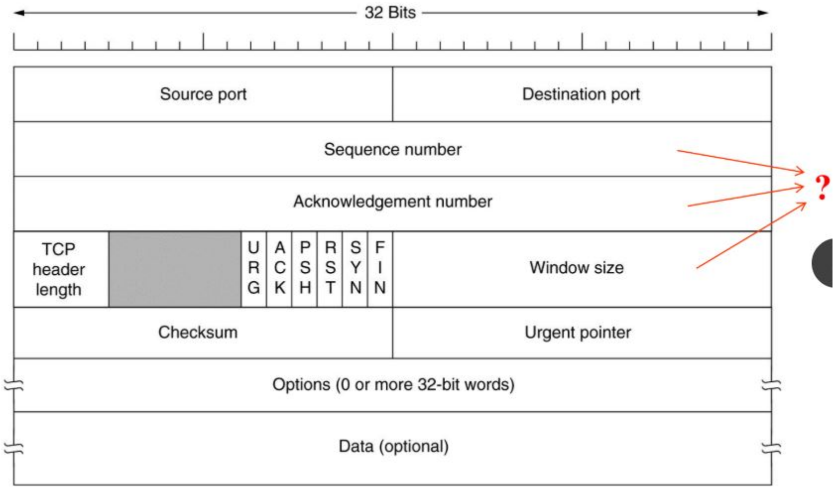
Duplicated segments discarded

-Window based flow control

0.6 The TCP Segment Header

0.7 TCP Header

- Ports number are the same as for UDP



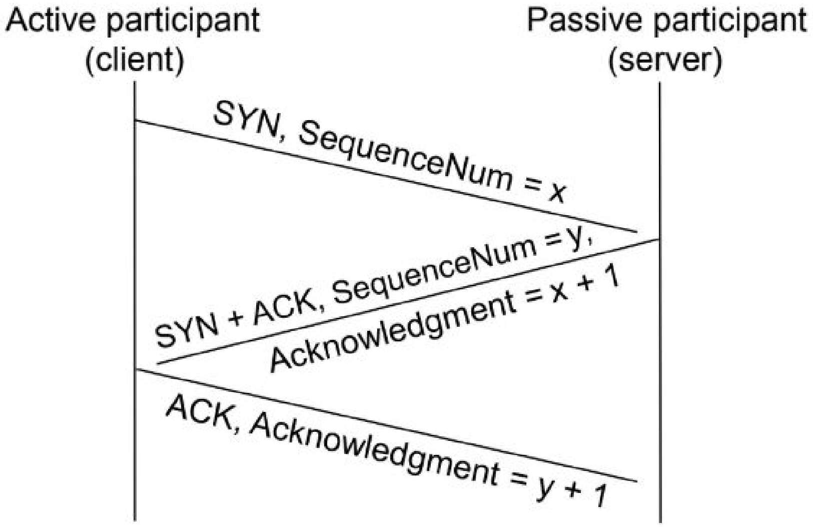
- 32 bit SeqNumber uniquely identifies the application data contained in the TCP segment
  - SeqNumber is in bytes
  - It identifies the first byte of data
- 32 bit AckNumber is used for piggybacking ACKs
  - AckNumber indicates the next byte the receiver is expecting
  - Implicit ACK for all the bytes up to that point
- Window size
  - Used for 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

0.8 Sequence Numbers in TCP

- TCP regards data as byte-stream (each byte is numbered sequentially)
- TCP breaks byte stream into segments (size limited by the Maximum Segment Size - MSS)
- Each packet has a sequence numbe (sequence number of 1st byte of data transported by the segment)

- TCP connection is duplex (data in each direction has different sequence numbers)

0.9 Connection Establishment


Active participant (client)
Passive participant (server)
SYN, SequenceNum = x
SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1
ACK, Acknowledgment = y + 1

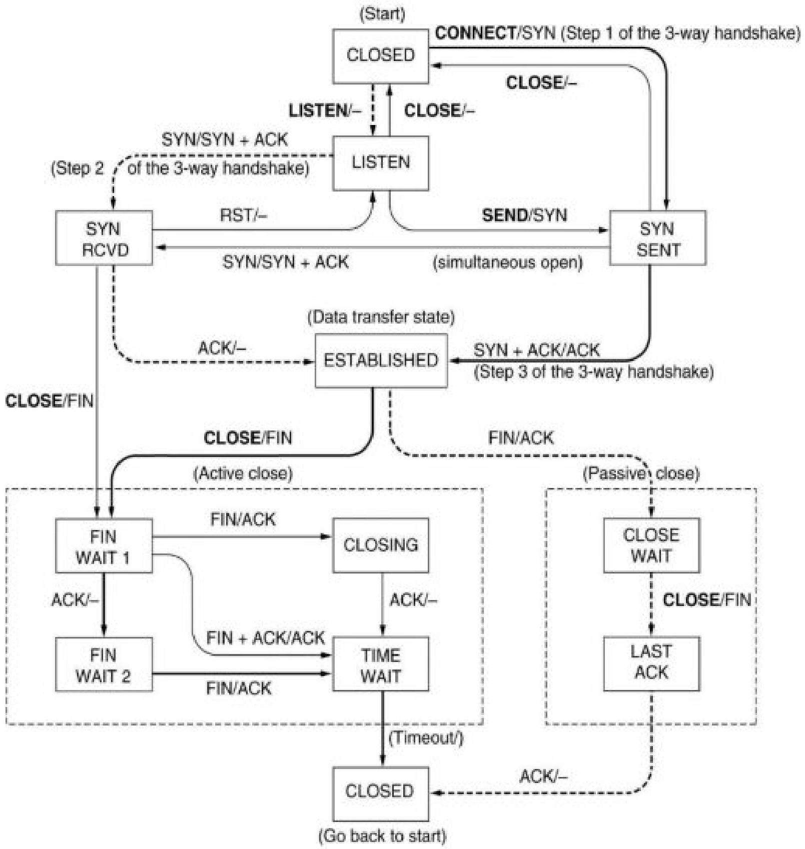
0.10 TCP connection management

0.11 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

0.12 Sliding Window

- Sender
  - LastByteAcked less or equal than LastByteSent
  - LastByteSent less or equal than LastByteWritten

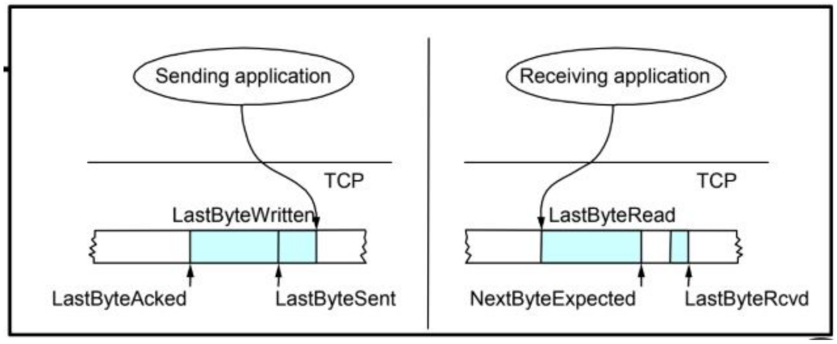


  - Buffers bytes between LastByteAcked and LastByteWritten

- Receiver
  - LastByteRead less than NextByteExpected
  - LastByteExpected less or equal than LastByteRcvd + 1
  - Buffers bytes between LastByteRead and LastByteRcvd

0.13 Flow Control

- Buffer length
  - Sender - MaxSendBuffer
  - Receiver - MaxRcvBuffer
- Receiver
  - LastByteRcvd - LastByteRead less or equal than MaxRcvBuffer



- AdvertisdeWindow equals MaxRcvBuffer - (LastByteRcvd - LastByteRead)

- Sender
  - LastByteWritten - LastByteAcked less or = MaxSendBuffer
  - LastByteSent - LastByteAcked less or = AdvertisedWindow
  - EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)

- Sending application blocks if it needs to write y bytes and (LastByteWritten - LastByeAcked) + y greater than MaxSenderBuffer

- ACK sent when a segment is received

0.14 Adaptive Retransmission (Original Algorithm)

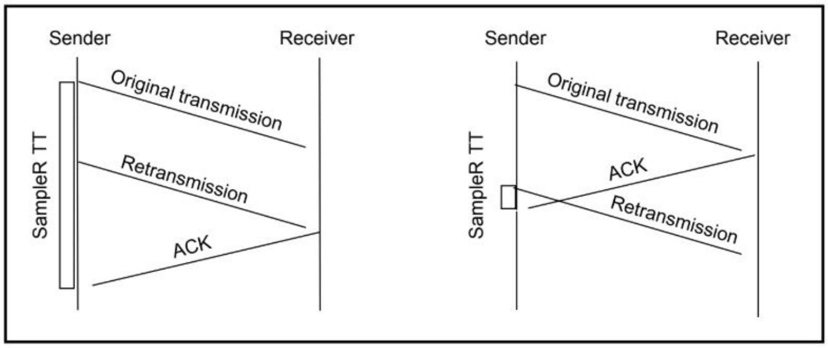
- RTT - Round trip time
- sampleRTT measured for each segment/ACK pair
- Average RTT (RTT = a * RTT + (1-a) * sampleRTT) a in [0.8,0.9]
- Timeout = 2 * RTT

0.15 Karn/Partride Algorithm

- sampleRTT not measured in retransmission
- Timeout doubled for each retransmission

0.16 Selective ACK

- Option for selective ACKs (SACK) also widely deployed
- Selective acknowledgement (SACK)



  - adds a bitmask of packets received
  - implemented as a TCP option

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

0.17 TCP- Congestion Control

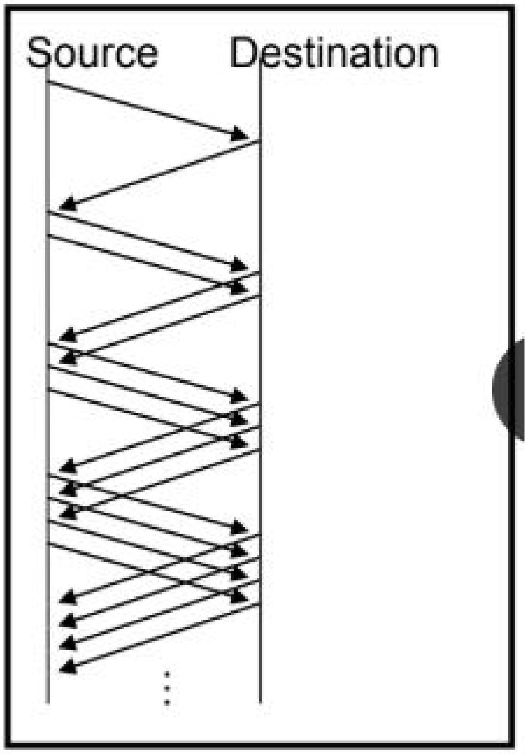
Each source determines its capicity.
Its based on criteria enablind **flow fairness** and **efficiency**.
Received ACKs regulate packet transmission, they are used as the source clock.

0.18 Additive Increase/Multiplicative Decrease

Changes in channel capacity leads to adjustment of transmission rate.
New variable per connection (CongestionWindow)
Limits the amout of traffic in transit :
MaxWin = MIN(CongestionWindow, AdvertisedWindow)
EffWin = MaxWin - (LastByteSent - LastByteAcked)

Objective
If network congestion **decreases** then CongestionWindow **increases**.
If network congestion **increases** then CongestionWindow **decreases**.
Bitrate(byte/s) = CongestionWindow/RTT

How to know if/when network is in congestion? Timeout!
Timeout occurence leads to loss of packet. Packet loss leads to buffer in router is full which leads to congestion. //



0.18.1 Algorithm

- increases CongestionWindow by 1 segment. For each RTT - additive increase
- divide CongestionWindow by 2. When there is a packet loss - multiplicative decrease.

0.18.2 In practice

- Increases by ACK received
- Increment = MSS * (MSS / CongestionWindow)
- CongestionWindow += Increment
- MSS = Maximum Segment Size

0.18.3 Objective

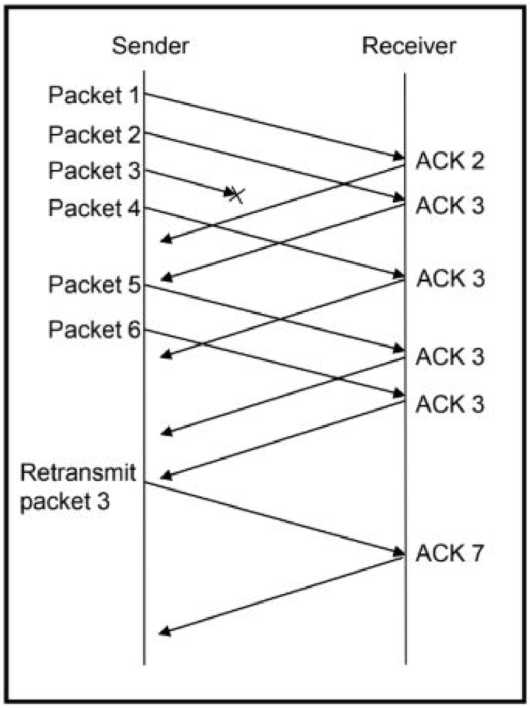
Determine the available capacity

0.18.4 Behaviour

Start by CongestionWindow = 1 segment.
Double CongestionWindow by each RTT.

0.19 Fast Retransmission, Fast Recovery

Problem If TCP timeout is large then long inactivity period

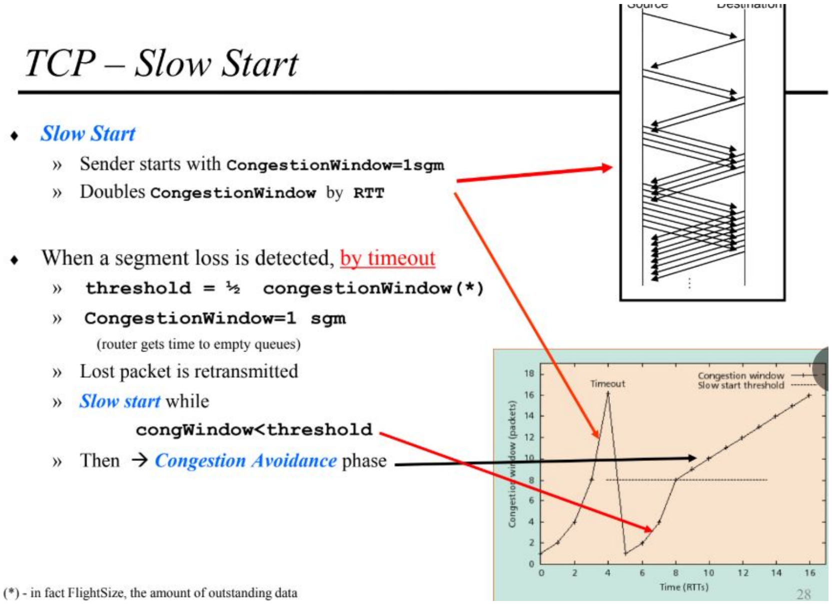


Solution Fast retransmission - after 3 repeated ACK's

0.20 TCP - Slow start

0.21 Congestion Avoidance

- Congestion Avoidance (additive increase)
  - increments congestionWindow by 1sgm, per RTT
- Detection of segment loss, by reception of 3 duplicated ACKs
  - Assumes packet is lost (because following segments have arrived)
  - Retransmits lost packet

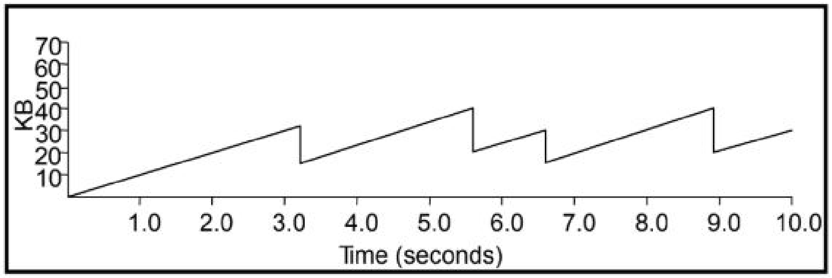

Source
Destination
Timeout
Congestion window
Slow start threshold
Congestion window (packets)
0
2
4
6
8
10
12
14
16
18
Time (RTTs)
0
2
4
6
8
10
12
14
16
28

TCP – Slow Start

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

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

- CongestionWindow = CongestionWindow/2
- Congestion Avoidance Phase

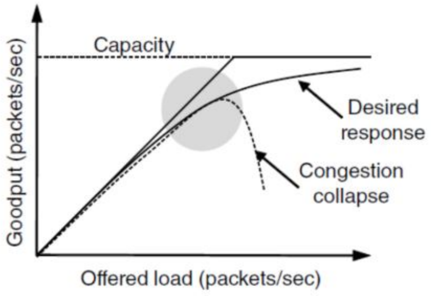


0.22 Desirable Bandwidth Allocation

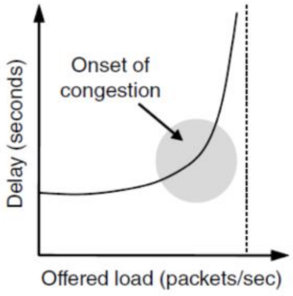
0.23 Max-min fairness

0.24 Bitrates along the time

Efficient use of bandwidth gives high goodput, low delay



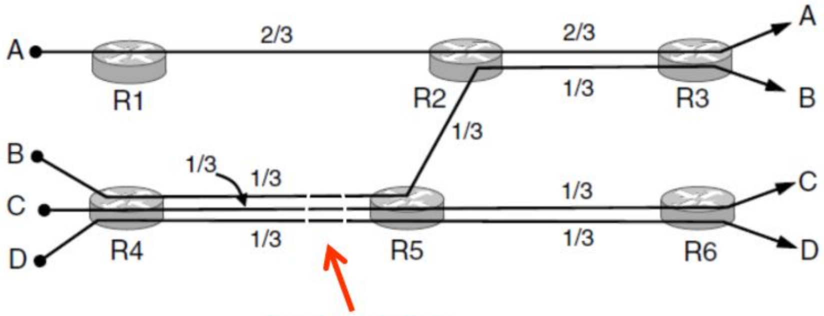
Goodput rises more slowly than load when congestion sets in



Delay begins to rise sharply when congestion sets in

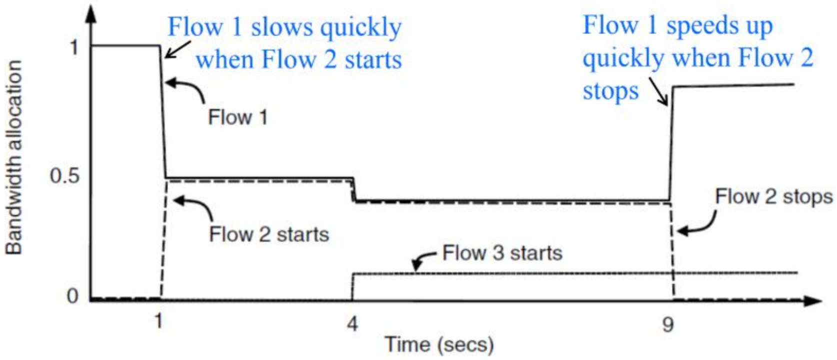
Fair use gives bandwidth to all flows (no starvation)

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



Bottleneck link

Bitrates must converge quickly when traffic patterns change


Flow 1 slows quickly when Flow 2 starts

Flow 1 speeds up quickly when Flow 2 stops