ECE544: Communication Networks-II, Spring 2005

Transport Layer Protocols

Sumathi Gopal

Feb 25th 2005

Lecture Outline

- Introduction to end-to-end protocols
- UDP
- RTP
- TCP
- Programming details

End-To-End Protocols

- Enable communication between 2 or more processes (which may be on different hosts in different networks)

- The Transport Layer is the lowest Layer in the network stack that is an end-to-end protocol

Transport Layer Protocols

- Connectionless protocols considered here
- Basic Function:
  - Enable process-to-process communication via virtual process-hooks called *ports.*

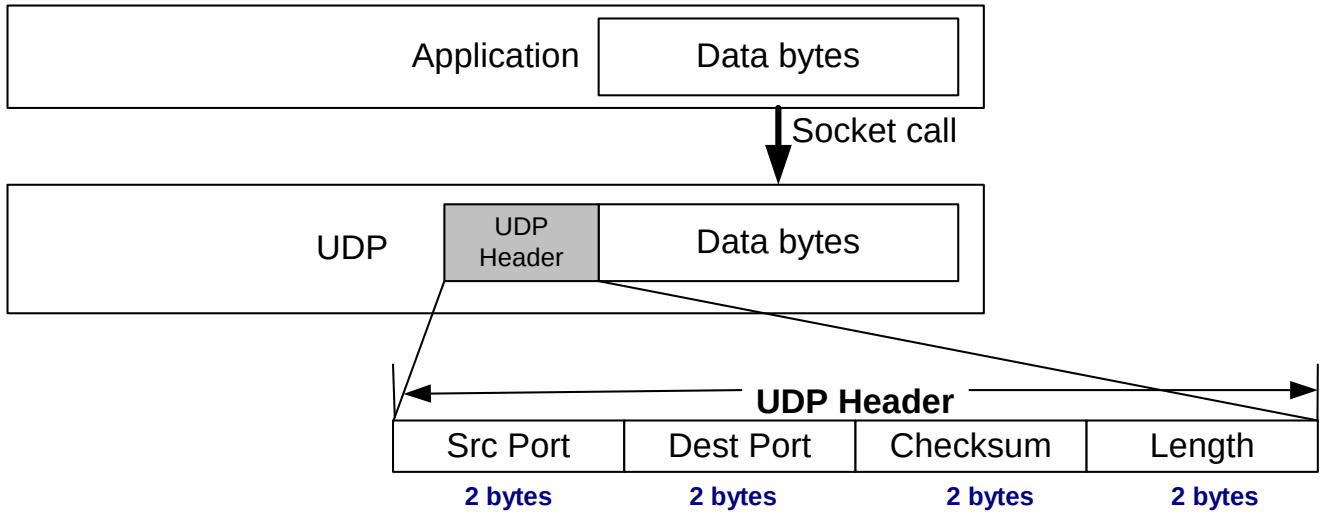
4-Tuple Connection Identifier:
< SrcPort, SrcIPAddr, DestPort, DestIPAddr >

- A transport protocol may provide several features in addition.

Most popular transport protocols

- User Datagram Protocol (UDP):
  - Provides the process identification functionality via ports
  - Option to check messages for correctness with CRC check
- Transmission Control Protocol (TCP):
  - Ensures reliable delivery of packets between source and destination processes
  - Ensures in-order delivery of packets to destination process
  - Other options
- Real Time Protocol (RTP):
  - Serves real-time multimedia applications
  - Header contains sequence number, timestamp, marker bit etc
  - Runs over UDP

User Datagram Protocol (UDP)

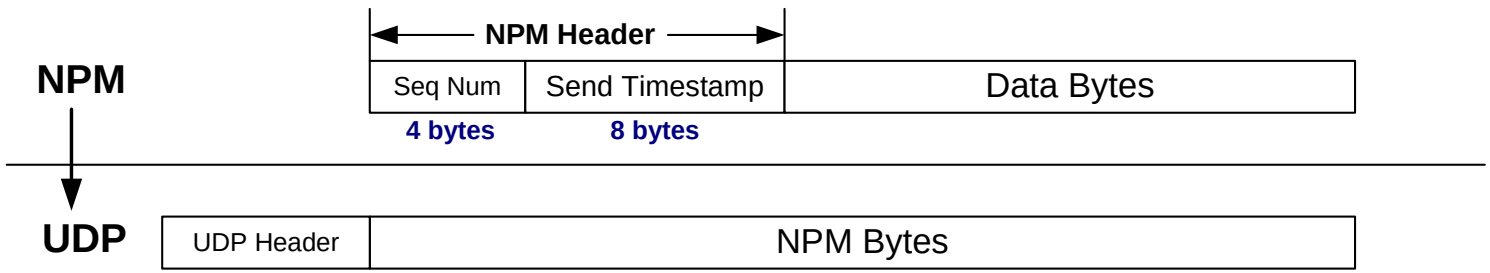


Header Fields:

- **Src Port**: Unique identification number assigned to the source process by the kernel in source node.
- **Dest Port**: The Unique Identification number assigned to the destination process by the kernel in the destination node.
- **Checksum**: Filled on source side. Checked on receiver side to ensure message correctness. Calculated over <Data, UDP hdr, portion of IP hdr>
- **Length**: Total number of bytes in (UDP header + data bytes)

Example of an application using UDP

- My application called Network Performance Monitor (NPM) needs to measure the pattern of packet losses in a network.
- Application needs sequence numbers and timestamps in each packet
- UDP does not provide this facility; So NPM adds its own header to each packet



Application requirements

- Like NPM, most applications need much more from a transport protocol than the basic functionality
- Multimedia applications require tracking of packet loss, delay and jitter.
- Most other applications such as HTTP, Database Management, FTP etc, require reliable data transport

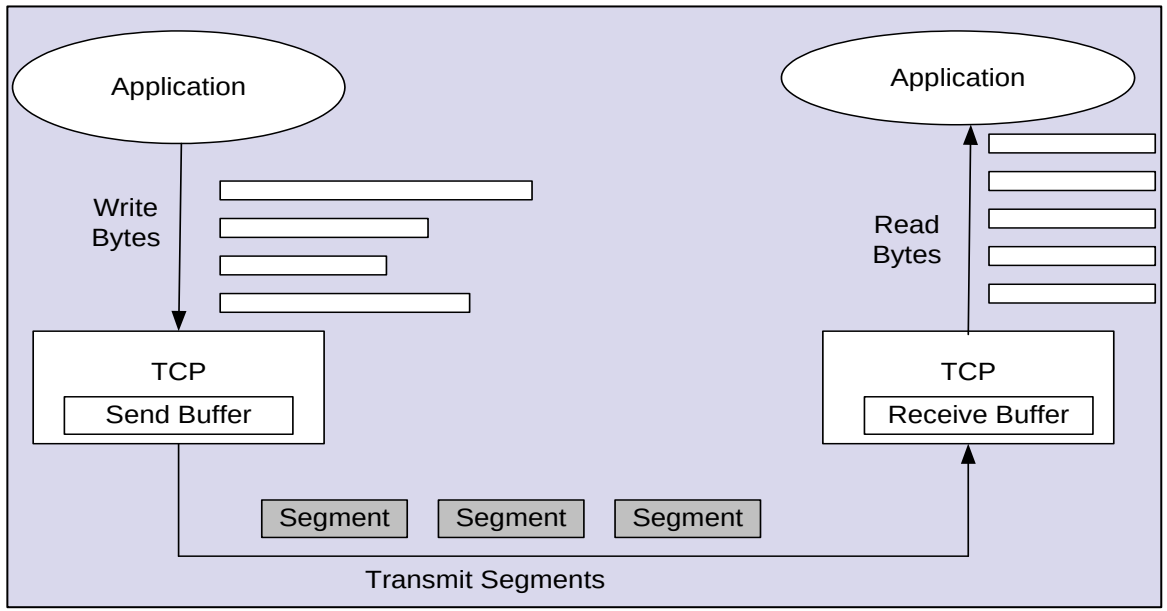
- TCP, UDP and RTP satisfy needs of the most common applications
- Applications requiring other functionality usually use UDP for transport protocol, and implement additional features as part of the application

Introduction to TCP

- The TCP/IP protocol suite has enabled computers of all sizes, from different vendors, different OSs, to communicate with each other.
- Forms the basis for the *worldwide Internet* that spans the globe.

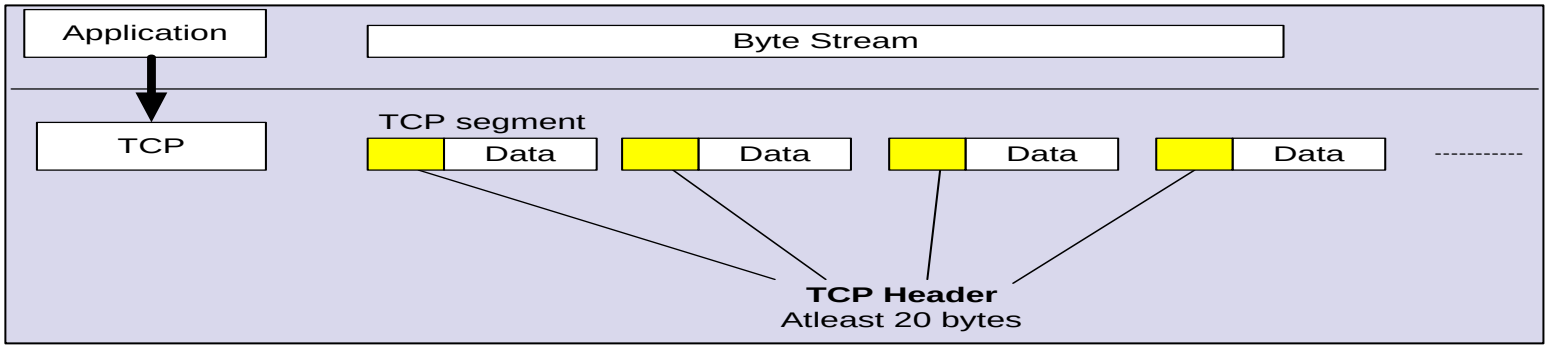
- Reliably delivers data between two processes
  - Assumes unreliable, non-sequenced delivery
  - Divides data passed to it from application process into appropriate sized chunks for the network layer below
  - Acknowledges received packets
  - Sets timeouts to ensure other end acknowledges packets set
- Application can ignore details of reliability

A top-level view of TCP operation

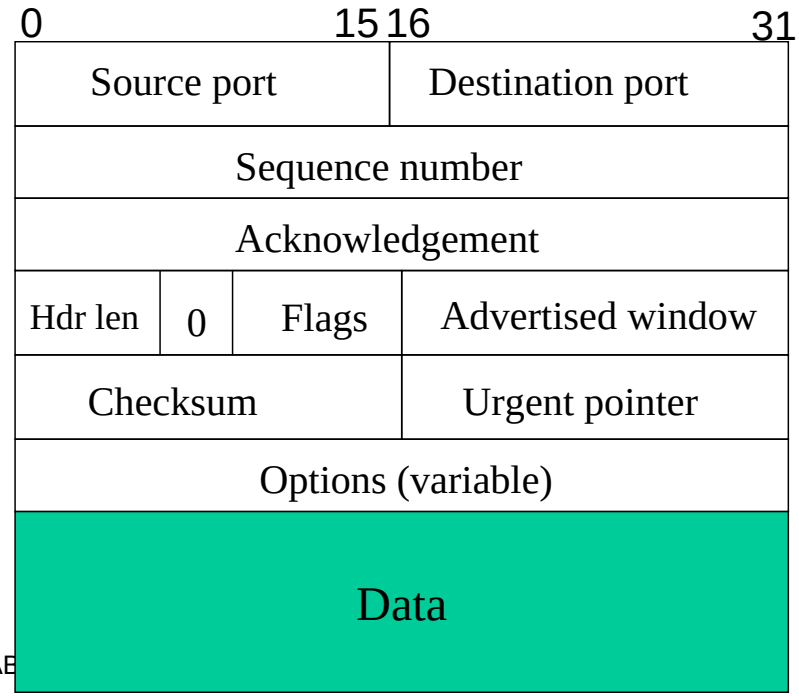


4-Tuple Connection Identifier:
< SrcPort, SrcIPAddr, DestPort, DestIPAddr >

TCP Header Format



Flags:
SYN
FIN
RESET
PUSH
URG
ACK



Summary of TCP's Operation Sequence

- All Operations are sender driven; TCP protocol completely implemented at the ends

Start:

- Connection Establishment by a Three-Way Handshake algorithm
- Consensus on Initial Sequence Number (ISN)

Data Transfer:

- An enhanced sliding-window protocol is the core of TCP operation
- Operate in *slow-start* and *congestion-avoidance* modes
- Receiver acknowledges successful reception of every segment
- Sender continuously estimates round-trip time and maintains several dependent timers to ensure reliability of data transfer

Finish:

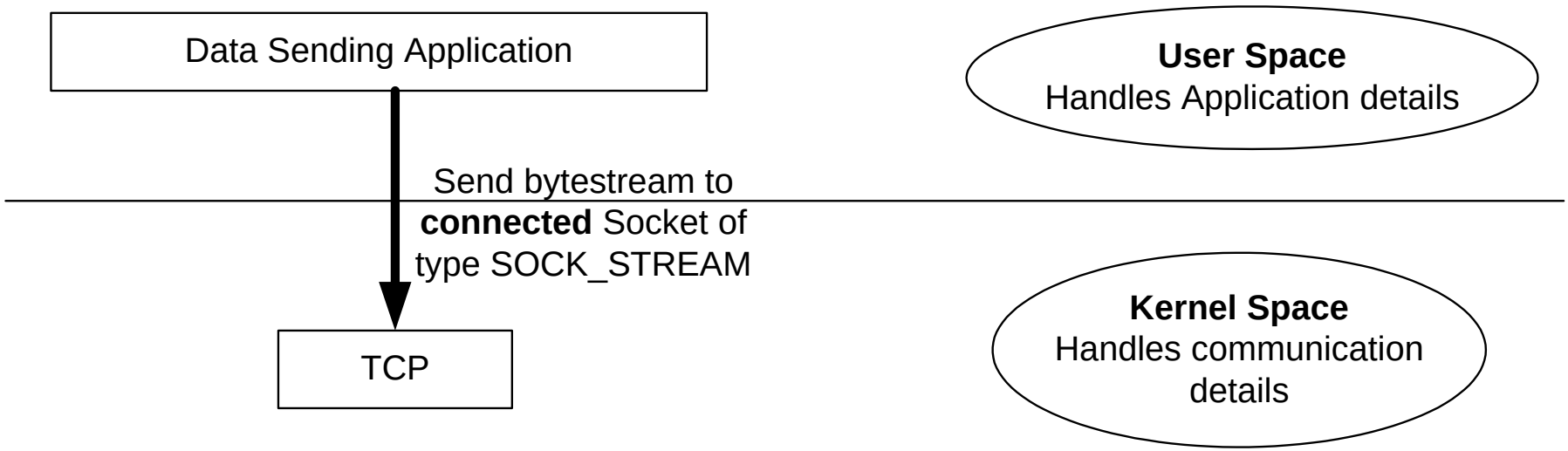
- Connection tear-down by a Three-Way Handshake algorithm
- Both sides independently close their half of the connection

A simple File Transfer Application

- Receiver process waits for connection and data from sender
- Sender process requests receiver for a connection
- Once the two processes are “connected”, the sender process transfers the file and closes.

- Receiver should be started first
- Receiver port Id should be known to the sender

Programming Viewpoint



- Sender application process only needs to provide a bytestream to the kernel
- Kernels on sending and receiving hosts operate TCP processes
- Receiver application process only needs to read received bytes from the assigned TCP buffers

Connection Establishment

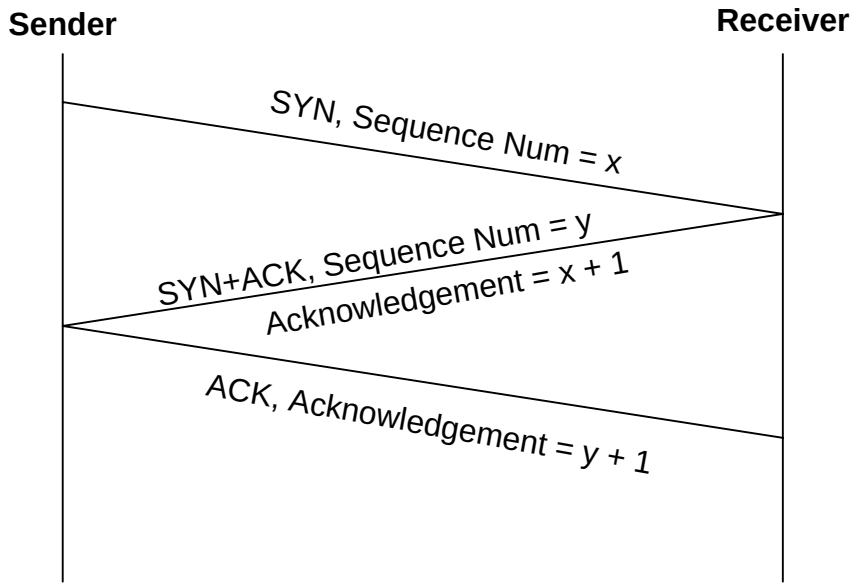
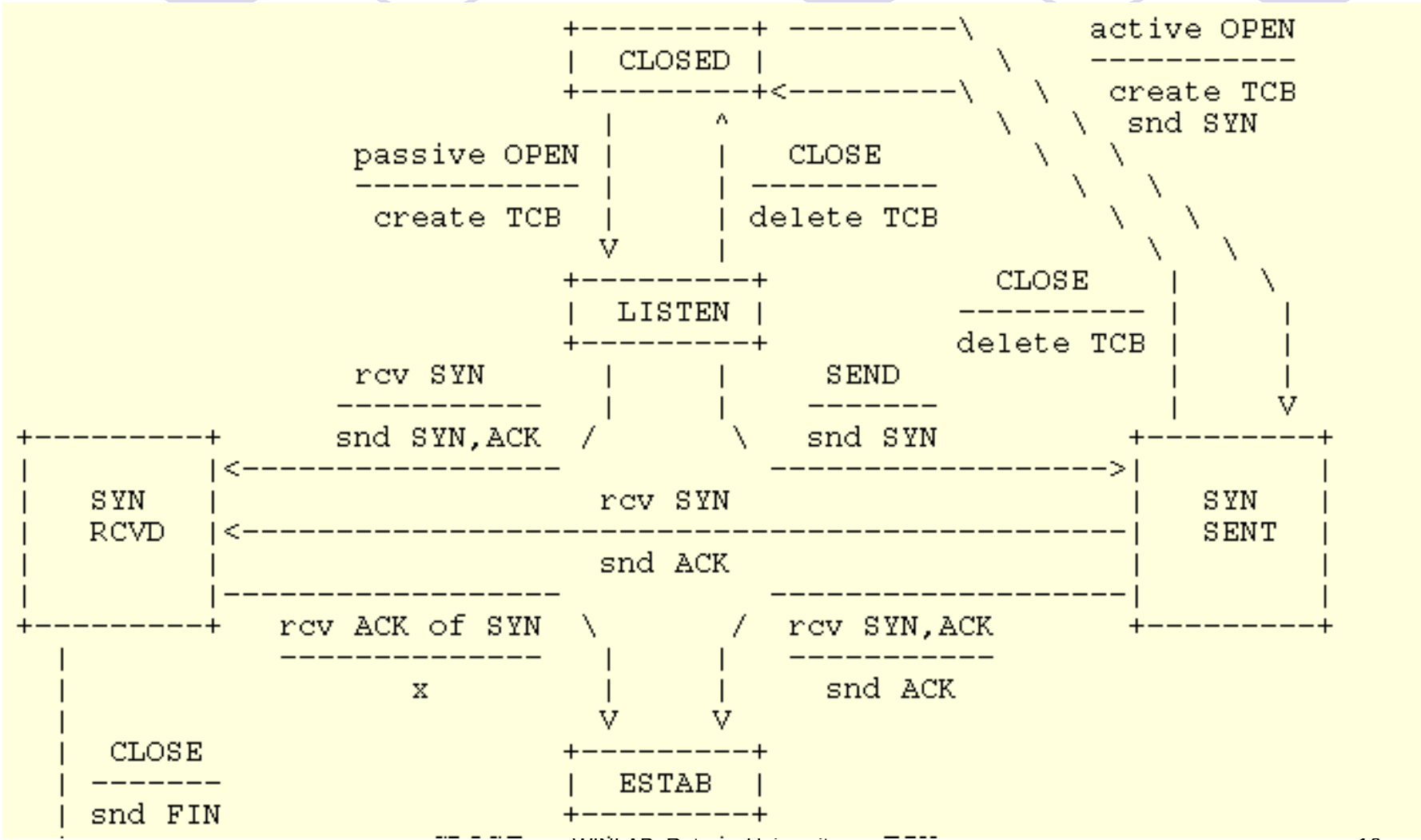


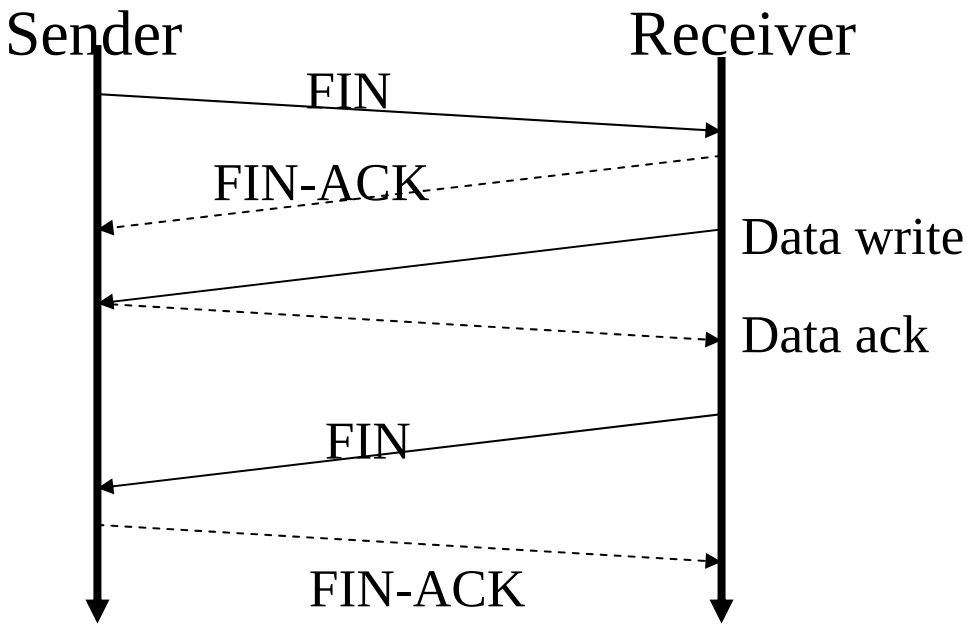
Figure 5.6 from text book

- Three-Way Handshake Algorithm
- SYN and ACK flags in the header used
- Initial Sequence numbers x and y selected at random
- Required to avoid same number for previous incarnation on the same connection

Connection Establishment

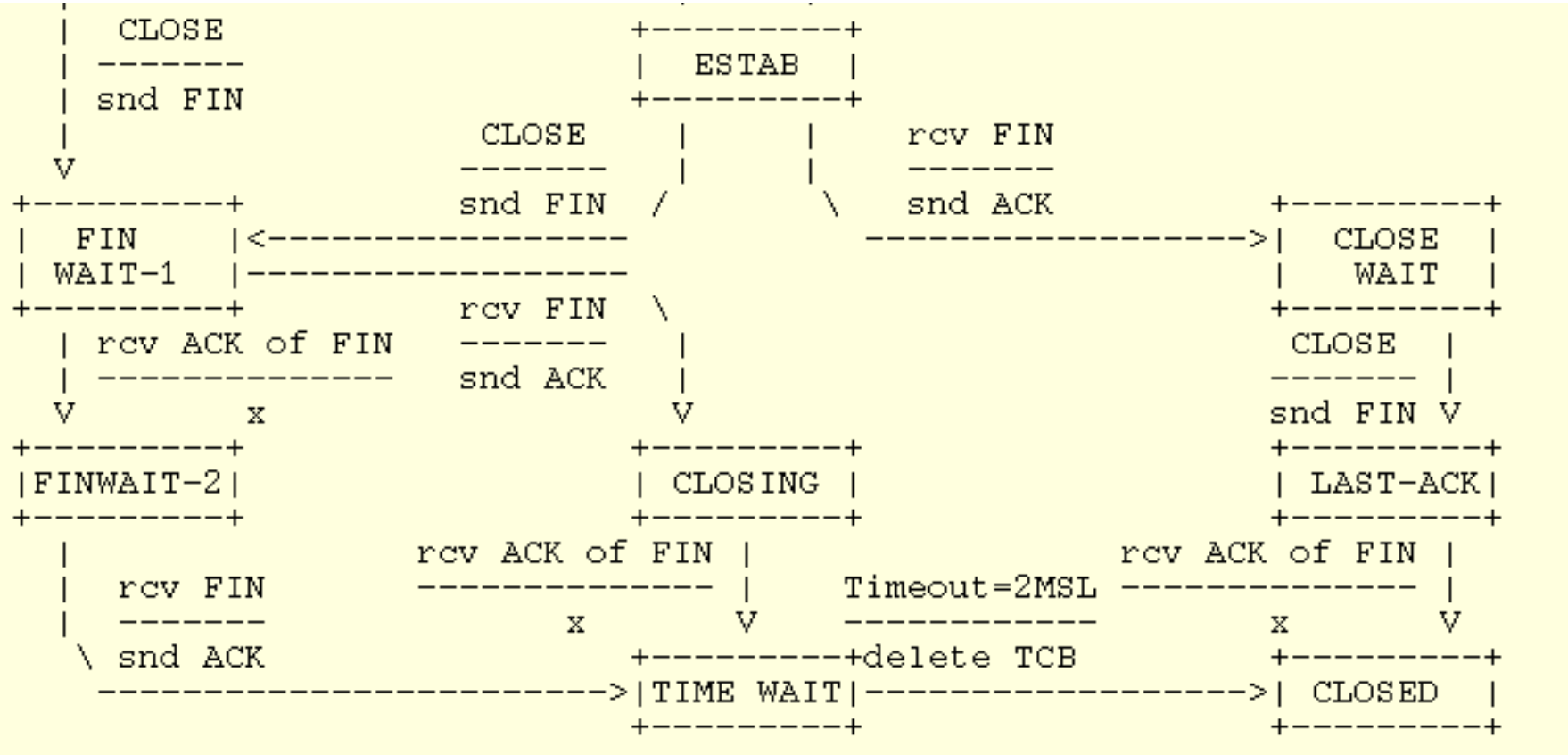


Connection Tear-down



- Each side closes its half of the connection independently

Connection Tear-down State Diagram



Observe that a connection in the TIME_WAIT state can move to CLOSED state only after waiting for **2*Max-TTL**

TCP Data Transfer Operation

Goal of TCP:

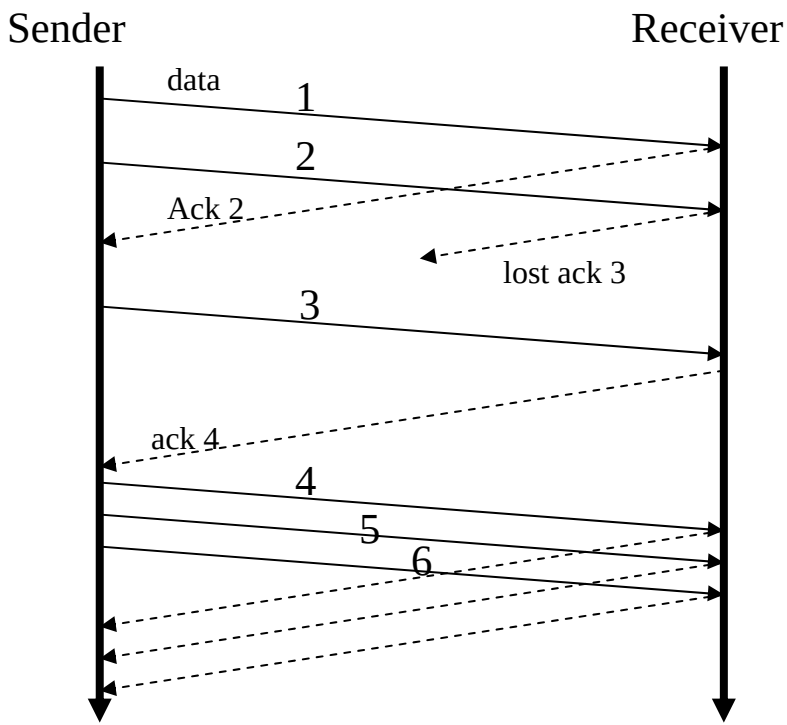
- Deliver data reliably and in order while maximizing end-to-end throughput (Throughput = bytes delivered/ time taken)

- Flow Control:
  - On sender side with *congestion window;* On receiver side with *advertised window*
  - Saturate network 'pipe': (delay X bandwidth) bytes
    - If more bytes sent: packets lost due to overflows
    - If fewer bytes sent: Network resources underutilized
  - Ensure receiver is not flooded with data

- Error Control or Congestion Control:
  - Lost ACK for a packet => lost packet
  - Packet loss interpreted as due to congestion => overflow of a queue in some router along the way.

The Reliability Mechanism

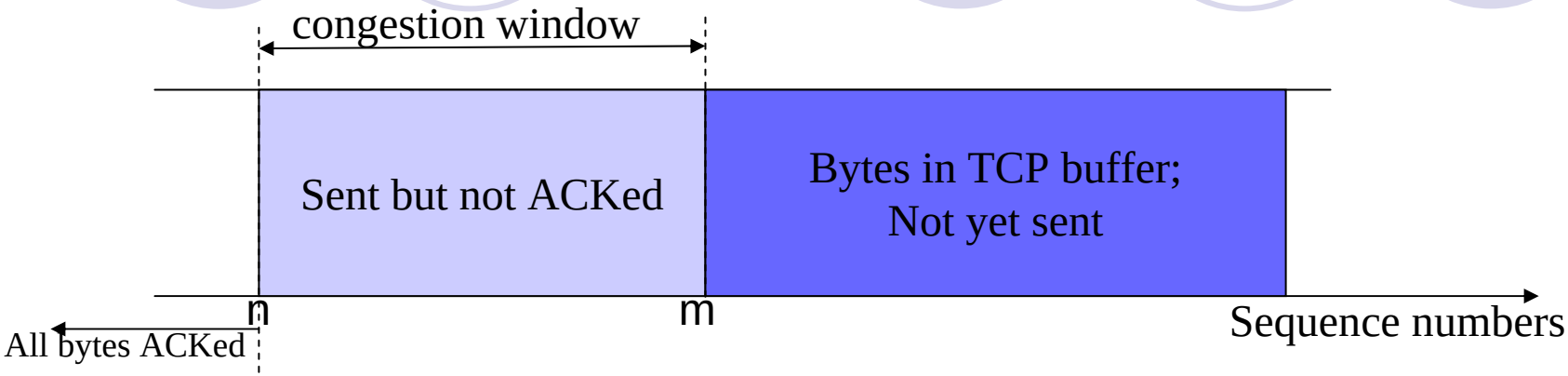
- Receiver generates ACKs *every time a* segment is received
- ACKs are cumulative



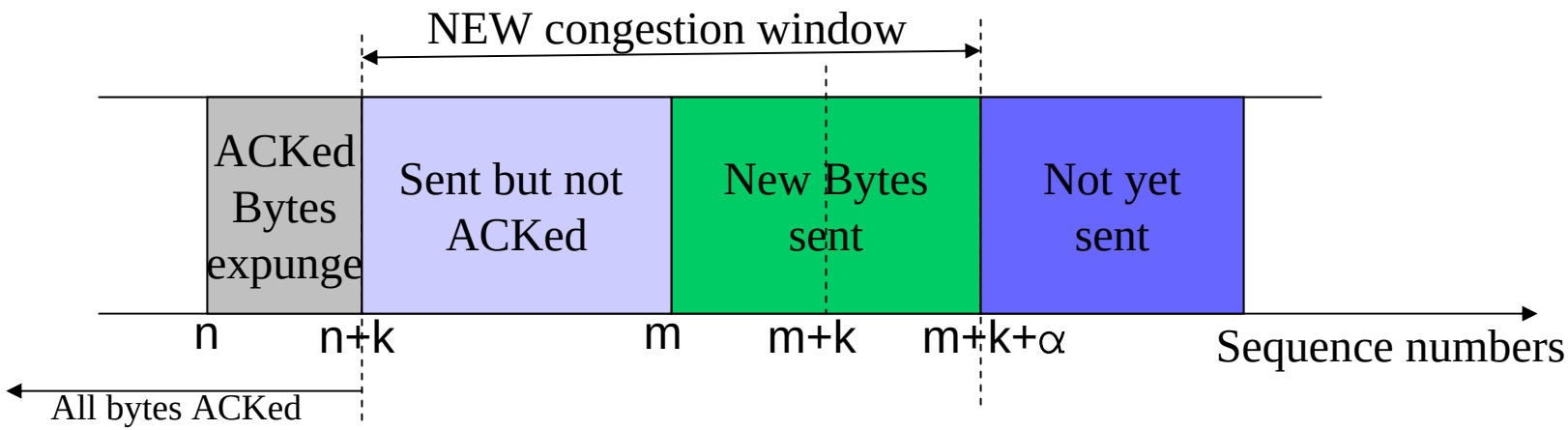
TCP's Congestion Window

- The Congestion Window (*cwnd*) is TCP's main tool of operation
- A sliding window used for both Flow control and Error Control
- Error Control:
  - *cwnd m*aintains all packets not-yet acknowledged
- Flow control:
  - Size of *cwnd* is the burst size that can be sent at one time
- Higher the size of *cwnd,* better the net throughput
- Sequence numbers maintained in bytes (remember, TCP serves a byte stream!)

cwnd Operation



Receive (ACK n+k) => Receiver has received all bytes upto (not incl.) n+k



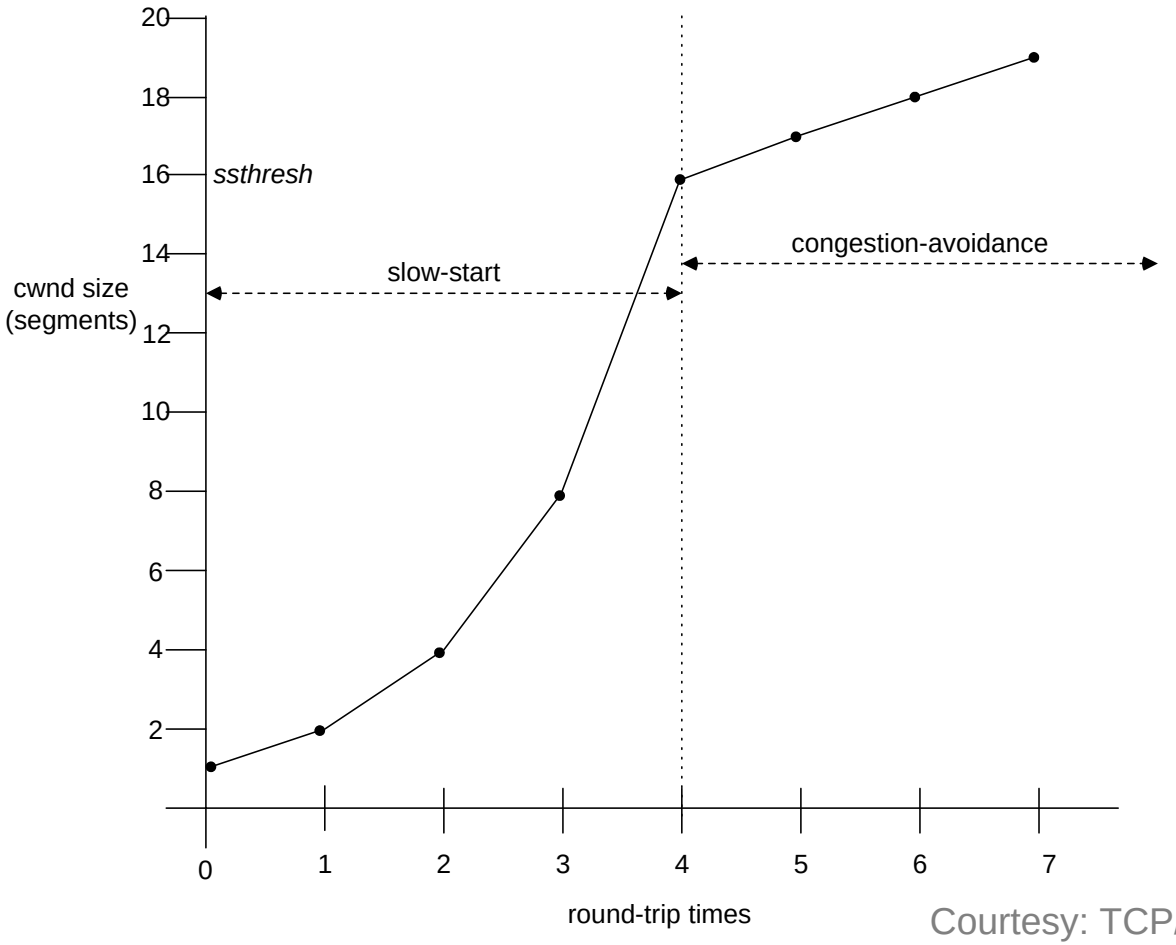
α depends on *mode* of operation

Modes of Operation

- Slow-start mode:
  - *cwnd* growth in this mode when
    - *cwnd* size < slow-start-threshold AND
    - No congestion has been detected
  - *cwnd* increases by a segment with every incoming ACK
  - Exponential increase
  - *cwnd* incremented by the number of ACKs received in one round-trip-time

- *congestion-avoidance mode*
  - *cwnd* growth in this mode in all other cases
  - *cwnd* incremented by (***1/cwnd)* number of bytes acked*** with each incoming ACK
  - Additive increase
  - *cwnd* incremented by at most one segment in each round-trip-time

Visualization of slow-start and congestion avoidance



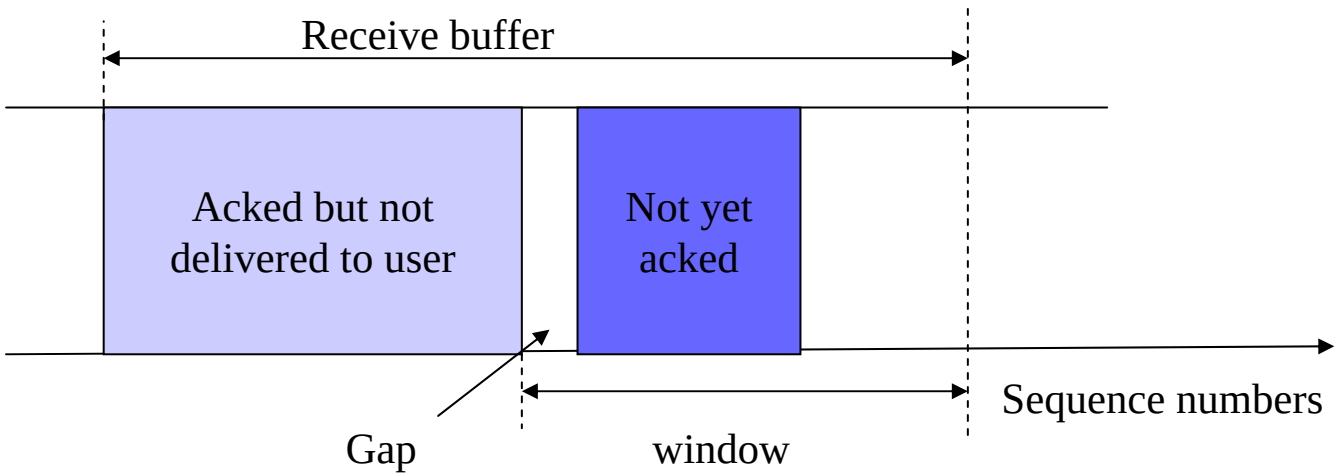
Assumes:

- ssthresh = 16
- All segments are ACKed and there are no packet losses

Courtesy: TCP/IP Illustrated, Vol 1 by W.R.Stevens

Receiver-side flow control

- Avoid flooding receiver with data
  - Notifies sender of number of bytes it can accept in *advertisedWindow* field in ACK header.
- Sender bytes sent = MIN(*cwnd*, *advertisedWindow*)
- Receiver delivers bytes in correct order to application process by maintaining a receive window



TCP's Error Control Mechanism

- Data segments and ACKs may get lost in transit; Losses interpreted as due to network congestion
- TCP sender sets deadlines for ACK arrival using timers;
  - Deadlines a function of estimated RTT
- If deadlines not met:
  - *cwnd* scaled down
  - Segment(s) retransmitted
- Accurate Round-trip Time estimation critical for efficient TCP operation
- Premature timeouts and retransmissions place huge toll on the net throughput

Round-trip time (RTT) Estimation

- Two important timers : Retransmission Timer and RTO Timer depend on accurate RTT estimation
- Round-trip time is variable. Smoothed RTT estimator:
  - $R = \alpha R + (1- \alpha)M$
    - R: smoothed RTT
    - M: new RTT measurement
    - α: smoothing factor (typically = 0.9)
- RTO = function of(smoothed RTT, RTT variance) (look in the book for expression)
- A single RTT estimator active at a time. Cumulative ACKs also considered
- Karn's Algorithm: Retransmitted segments not considered for RTT estimation because of *retransmission ambiguity problem*

Timeout and Retransmission

- RTT Timer Expiry:
  - Duplicate ACKs arrive, but the ACK for a specific segment does not arrive
  - TCP interprets this as loss of a single segment, and a transient network congestion
  - RTT Timer expires and triggers an immediate retransmission of the segment requested in the duplicate ACKs.

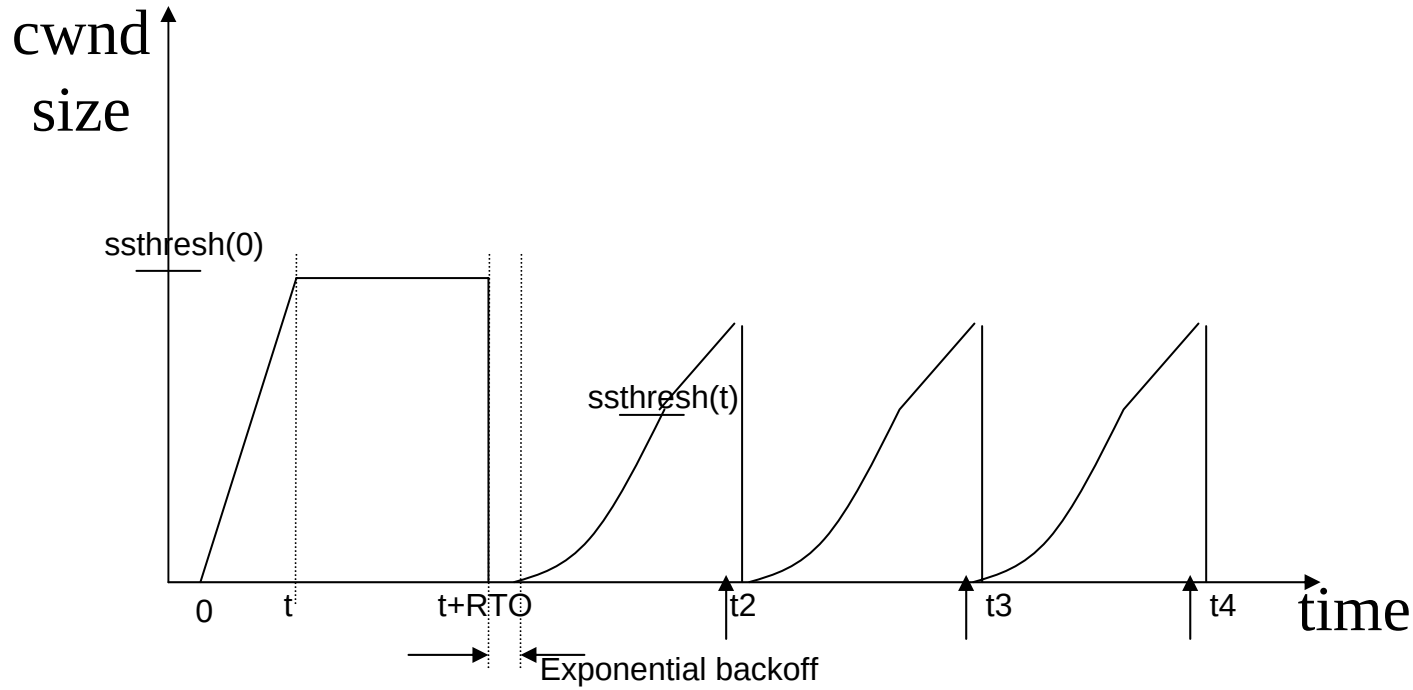
- RTO Timer Expiry:
  - No ACKs arrive at all before RTO timer expires
  - TCP interprets this as heavily congested network
  - *Exponential Backoff* triggered when no segment is transmitted and network is given time to recover from congestion
  - After the backoff duration, the first unacknowledged segment is retransmitted subsequent resumption of data flow only after Backoff duration

- Expiration of either timer sets *cwnd*=1, and *slow-start* mode

Timeout impact on Throughput

- A timeout reduces *cwnd* size to 1 => Just 1 segment transmitted in 1 RTT.
- *cwnd* subsequently grows very cautiously in slow-start mode
- Bad for lossy high bandwidth-delay paths
- All segments following the lost segment are also retransmitted: even if they have been successfully received (out of order) at the receiver
- A possible bulk retransmission of a large portion, may further contribute to network congestion

Ideal behavior of TCP Tahoe



At t2, t3, t4:

- Duplicate ACKs arrive
- Retransmission timer expires
- Single packet retransmitted in slow-start mode at cwnd=1
- Next segments sent based on cumulative ACKs received

Optimizations in various TCP flavors

- Several optimizations for better TCP throughput in the past 30 years.
- Most important among them:
  - Fast Retransmit
  - Fast Recovery
  - Selective Acknowledgement (SACK)
  - Delayed ACKs

Fast Retransmit

- Don’t wait for retransmission timer to expire that causes *cwnd* to drop to 1
- React to duplicate ACKs instead
  - Don’t know if duplicate ACKs are because of packet loss or reordering. Threshold set to 3 duplicate ACKs
  - On receiving 3 duplicate ACKs:
    - Requested segment retransmitted *cwnd* growth continues
    - Set *ssthresh* = *(½* *MIN(*cwnd,* rcvr_adv_window)) bytes
    - Set *cwnd* = (*ssthresh* + num-dup-ACKs*segment_size) bytes
  - *cwnd* continues to grow with arriving dup-ACKs
    - Each duplicate ACK implies that a segment has left the network and reached the receiver
  - New segment transmitted if *cwnd* size permits

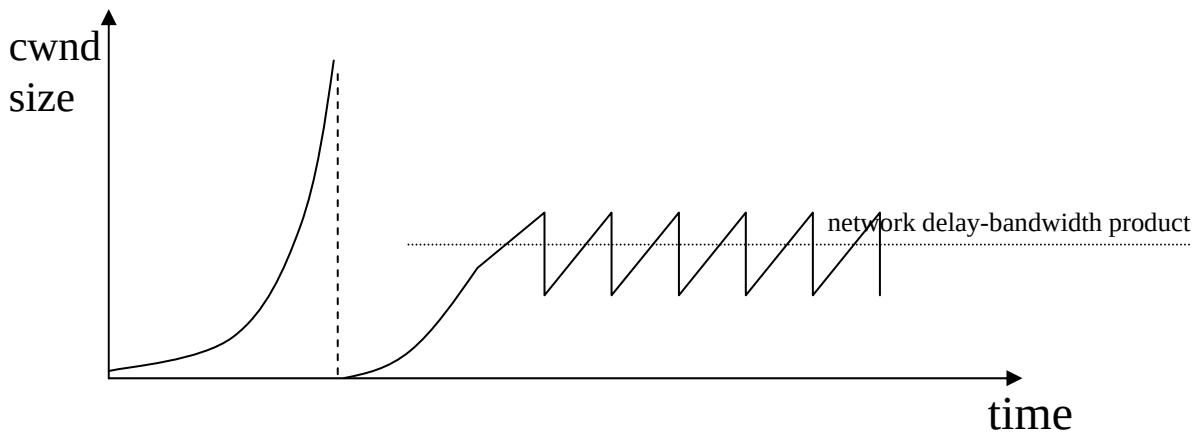
Fast Recovery

- When ACK for retransmission received *cwnd* growth resumes in *congestion avoidance* mode with *cwnd=ssthresh* rather than starting in slow-start mode with *cwnd=1*

- For an example of Fast Retransmit and Fast Recovery refer “TCP/IP Vol 1, W.R.Stevens”, section 21.8, Figures 21.10 & 21.11, Page 315

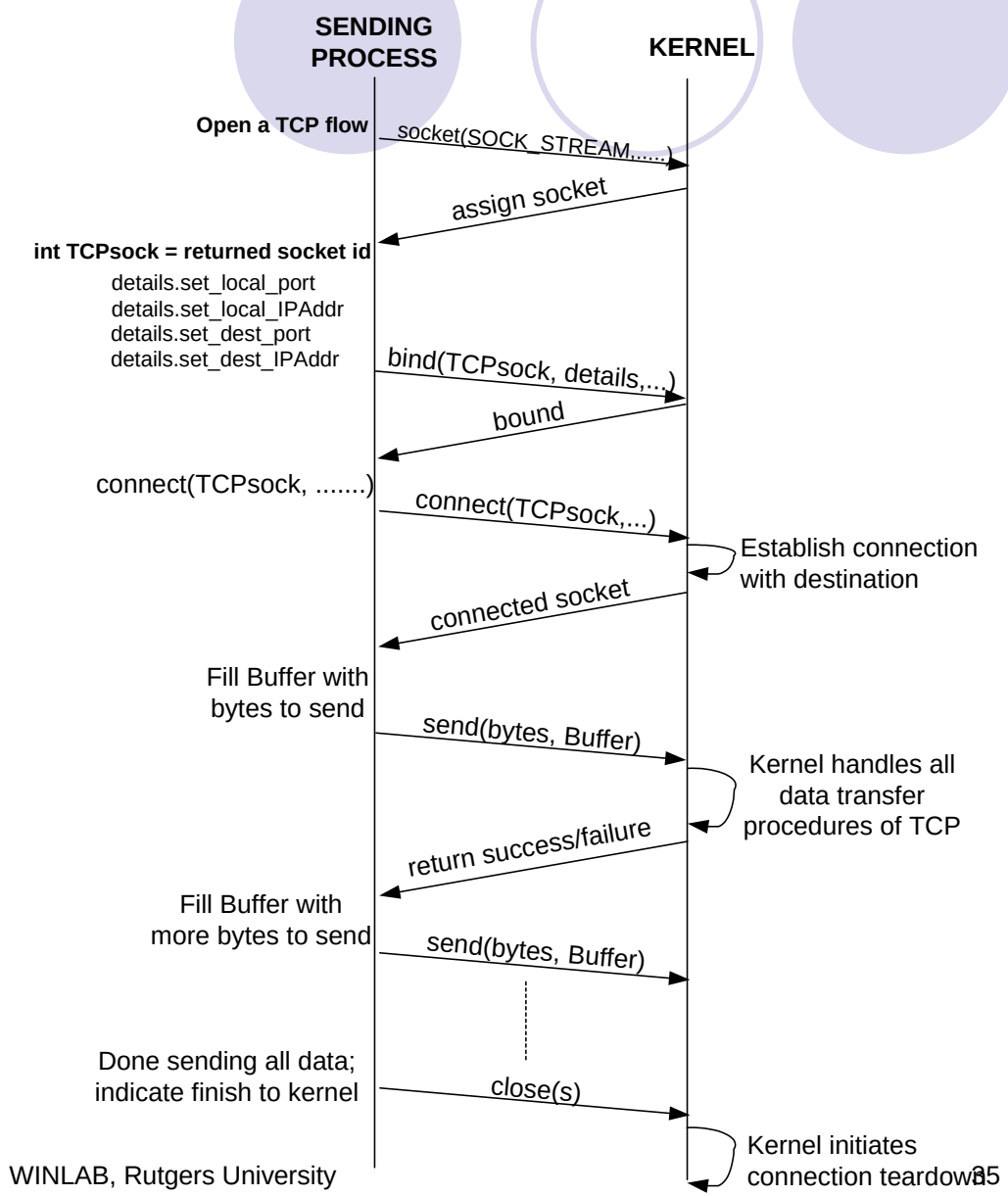
TCP Reno

- Most popular TCP flavor; implemented in most operating systems
- Implements Fast Retransmit and Fast Recovery in addition to default TCP congestion control and flow control mechanisms.



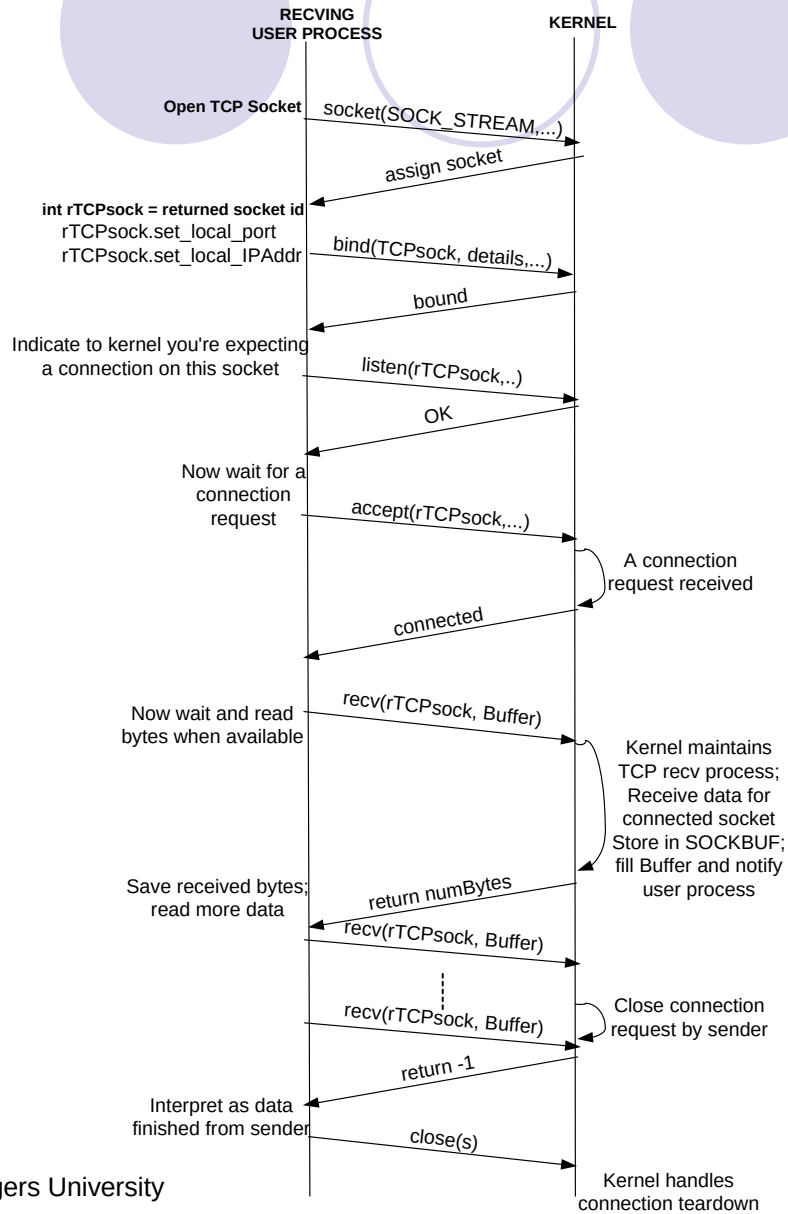
Programming viewpoint

Sender side



Programming viewpoint

Receiver side



TCP is not the ideal for all applications

- TCP optimized for wired networks
- Performance is poor in wireless networks
- Applications with stringent delay requirements do not use TCP, because of possible unbounded delays

Real-Time Protocol

- Quality of Service (QoS) factors: Reliability, Delay and Jitter
- Because of possibly unbounded retransmissions in TCP, large delay and jitter may ensue.
- Applications prefer UDP instead.
- RTP protocol operates over UDP, and with header containing
  - timestamp
  - sequence number
  - A marker bit
  - Packet concatenation etc
- RTP provides no other correction strategies like in TCP; Applications handle all aspects themselves.
- RTP modules run in user-space. RTP libraries included in the application.

Summary

- Numerous transport protocols proposed
- TCP sustained because of its distributed nature and because of the TCP/IP protocol suite that enabled computer systems to connect across boundaries
- Ample scope exists for new transport protocols given proliferation of heterogeneous networks and devices

Homework!

- 5.23
- 5.29
- 5.35
- 5.36
- 5.39