MIEEC

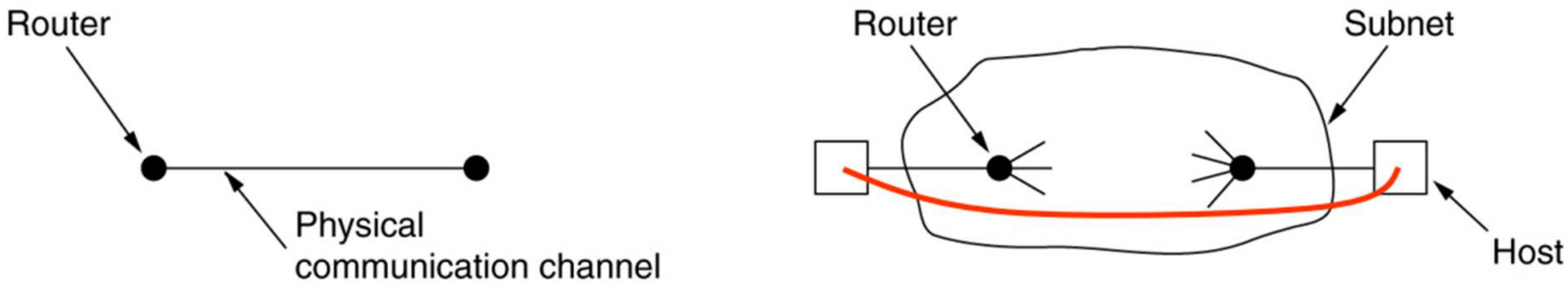
Computer Networks

Lecture note 8

Transport Layer

Transport layer services

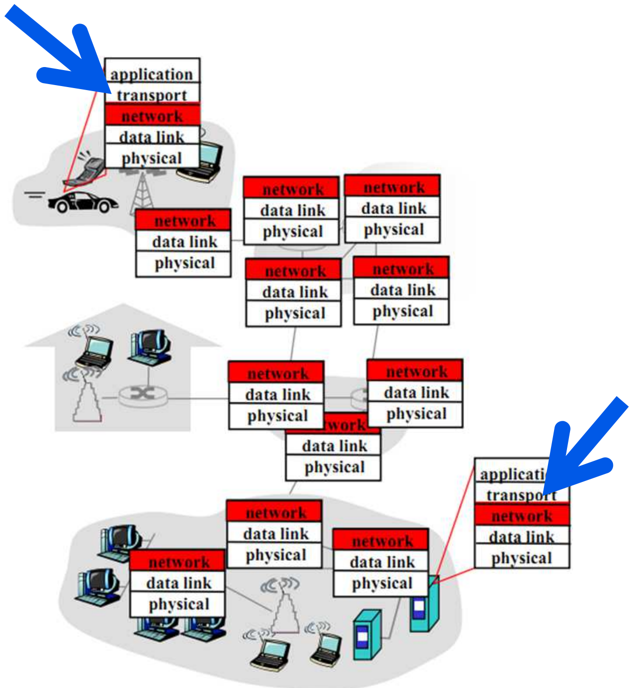
Point-to-point:
Data link vs. Transport

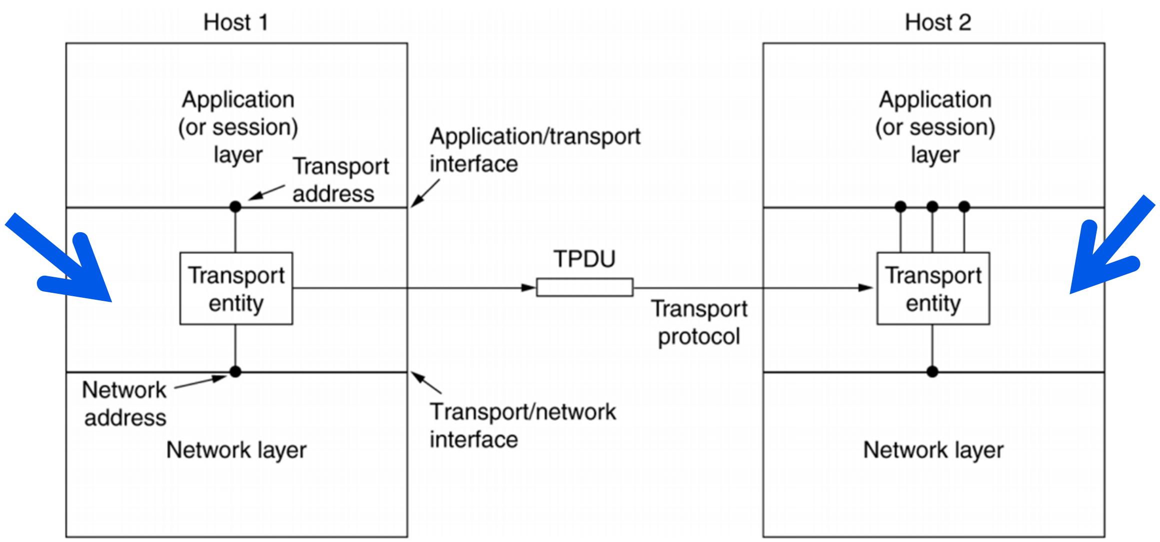


(a) (b)

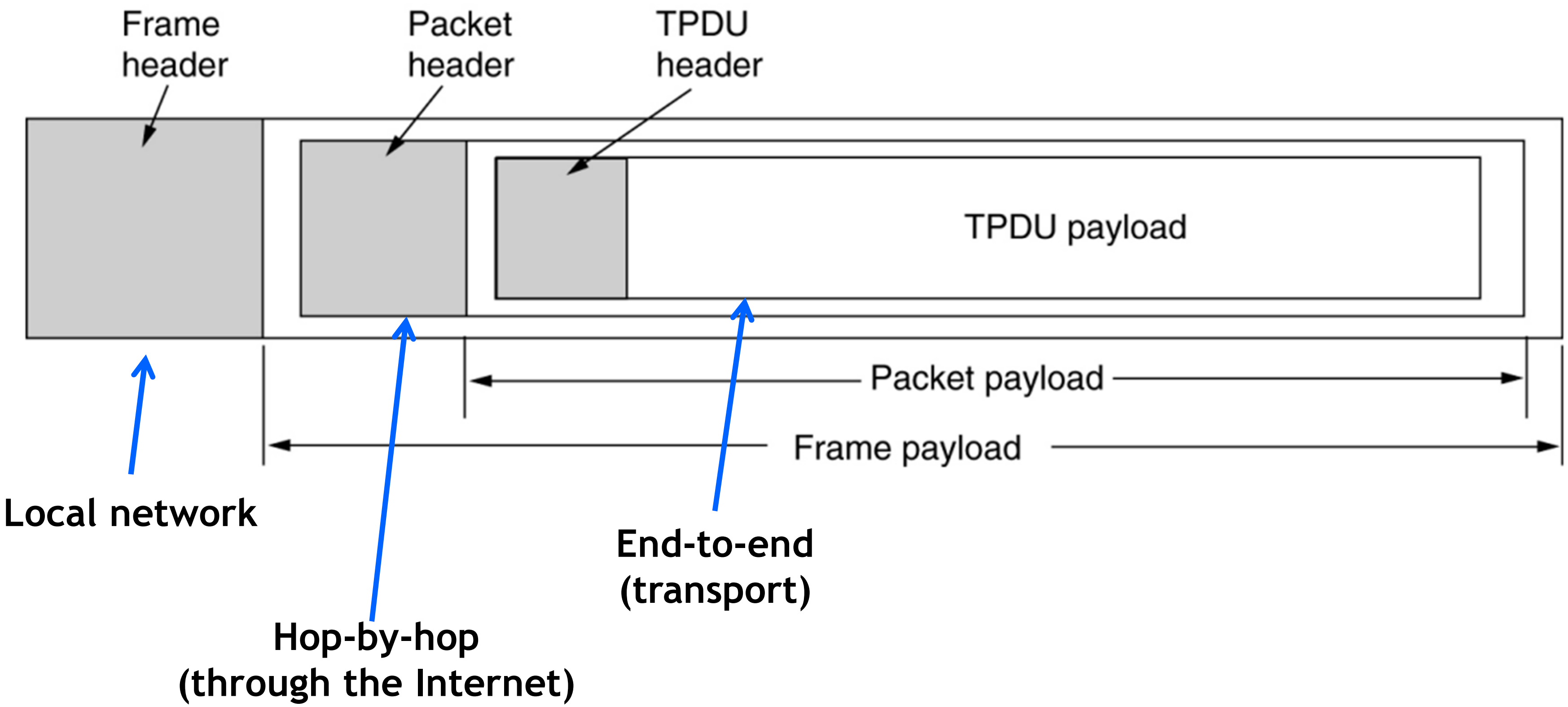
a) P-to-P: Data link layer
b) P-to-P: Transport layer

End-to-end service





Transport datagram encapsulating

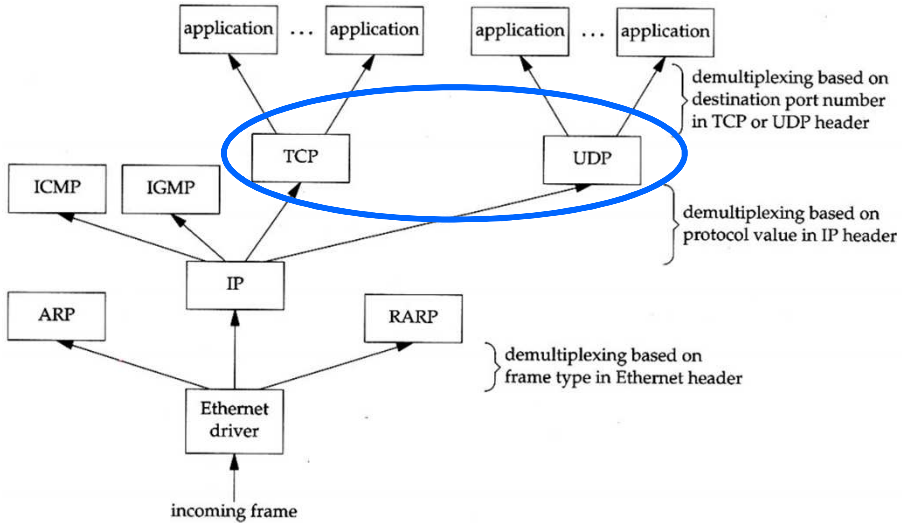


TO THINK

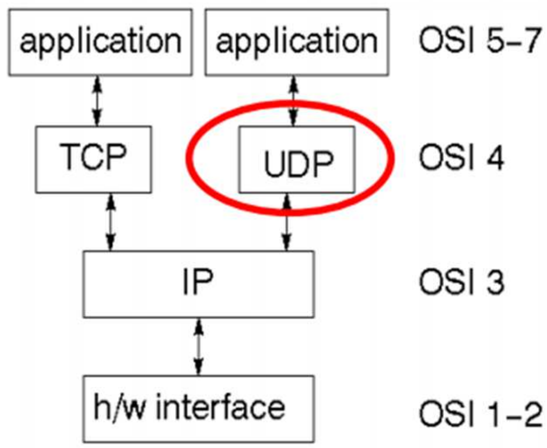
- What do you want out of a transport protocol?
  - Why can’t you just use IP?

Demultiplexing and flows

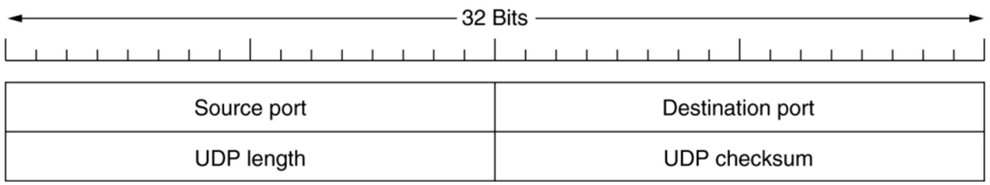
- Flow
  - Source IP
  - Destination IP
  - Source Port
  - Destination Port
  - Transport (TCP/UDP)
- 1 flow => 1 application



UDP
User Datagram Protocol

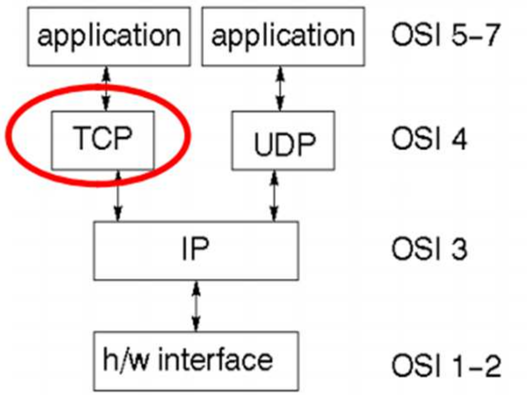


- Datagram oriented
  - Unreliable, no error control
  - Connectionless, no reordering etc.
- Provides applications with “direct”
  - Small protocol overhead
  - De-multiplexing
- UDP header
  - Port numbers identify sending/receiving process
  - UDP length : length of packet in bytes
  - Checksum covers header and data, optional

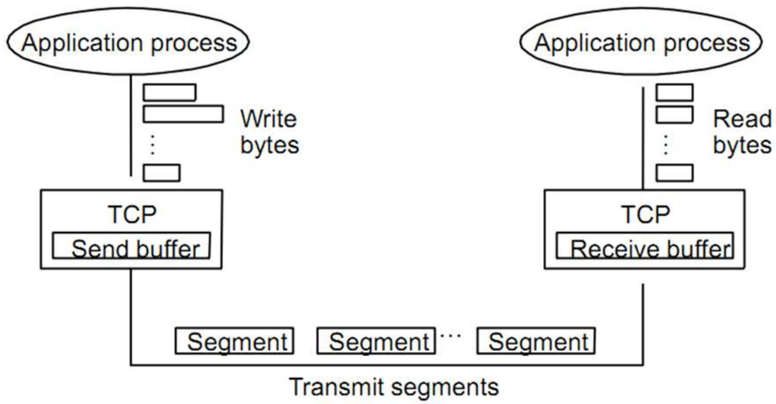


TCP
Transmission control protocol

- Connection oriented
- Full duplex
- Byte stream



- Flow control
  - Reliability
  - Avoids sending more than receiver can handle
  - ARQ
- Congestion control
  - Avoids congestion in the network



TCP

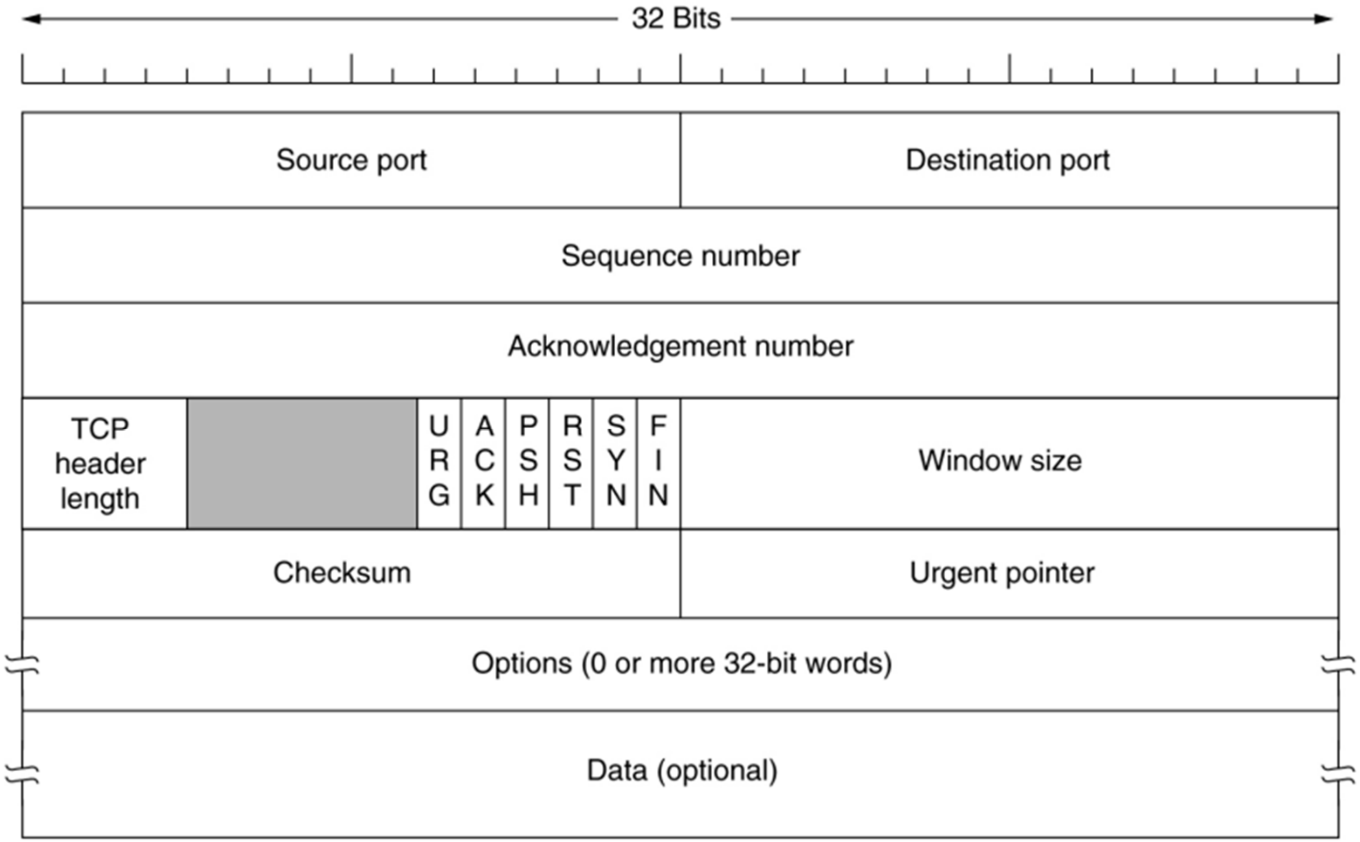
Basic TCP operation

- Sender
  - Application data is broken in segments
  - TCP uses timer while waiting for an ACK for every segment
  - Un-ACKnowledged packets are retransmitted

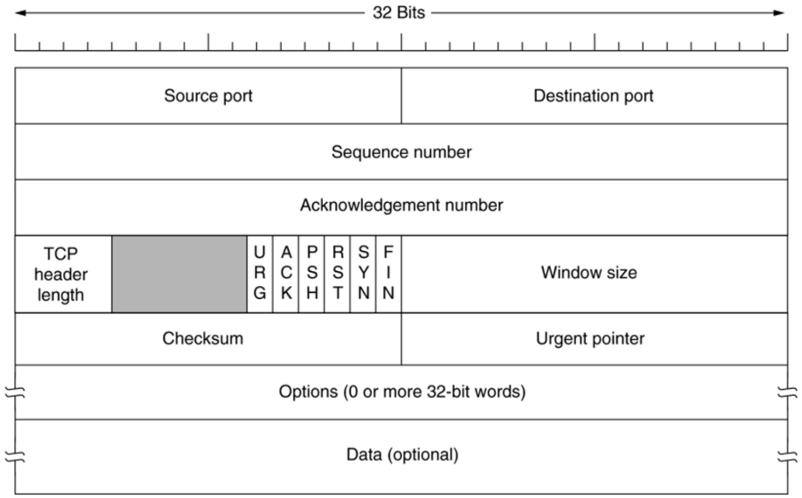
- Receiver
  - Errors detected using checksum
  - Correctly received data is acknowledged
  - Segments are reassembled in order
  - Duplicate segments are discarded

- Window-based flow control
  - ?

TCP segment header



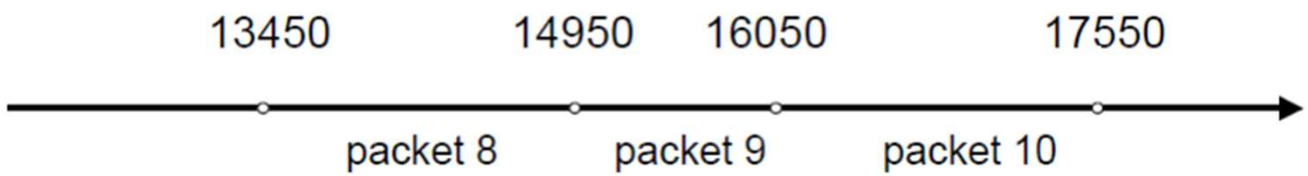
TCP header



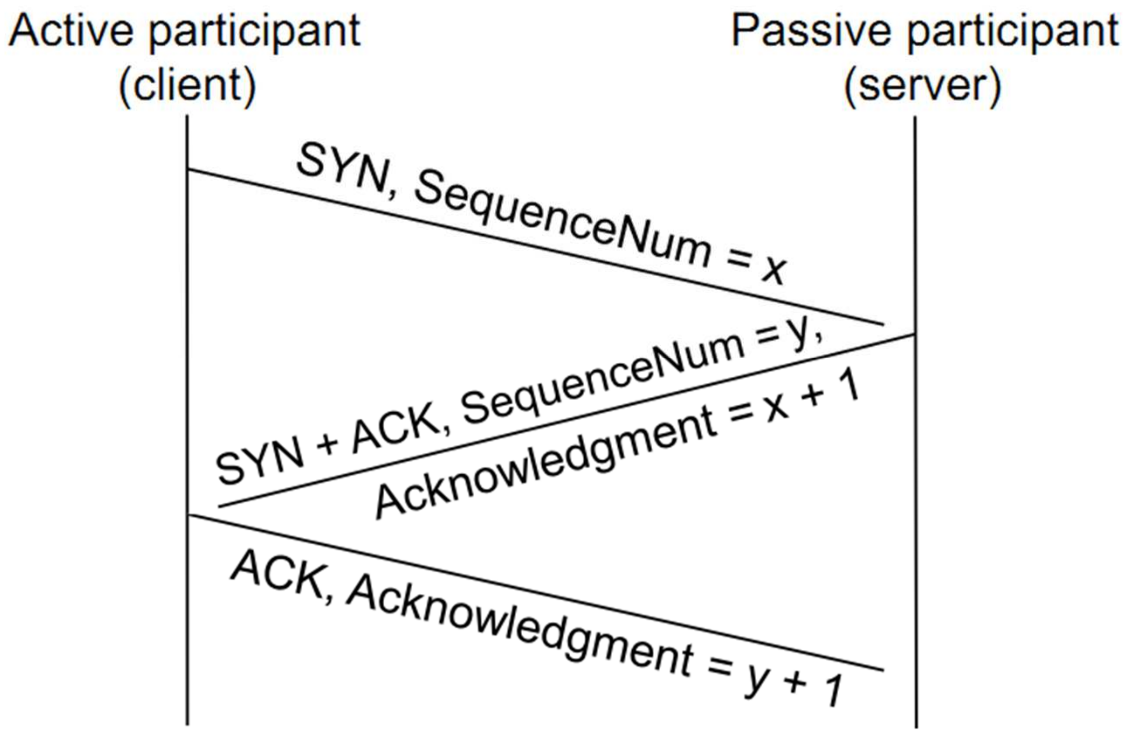
- Port numbers
  - Similar to UDP, demux
- Sequence number
  - Uniquely identifies which application data is contained in the segment
  - SN in bytes, identifies 1st byte
- ACK number, piggybacking ACK's
  - Next byte the receiver is expecting
  - Implicit ACK for all bytes up to that point
- Window size
  - Flow control (ARQ) and congestion control
  - Cannot send more than window size to network
  - In bytes; can increase/decrease depending on network/traffic
- Checksum covers header and data

TCP sequence number

- TCP views data as streams of bytes
  - Bytes are numbered sequentially
- TCP breaks stream in segments
  - Maximum segment size MSS
- Each packet has a sequence number
  - This is the number of the 1st byte in the packet
- TCP connection is duplex
  - Different byte streams and sequence numbers in each direction

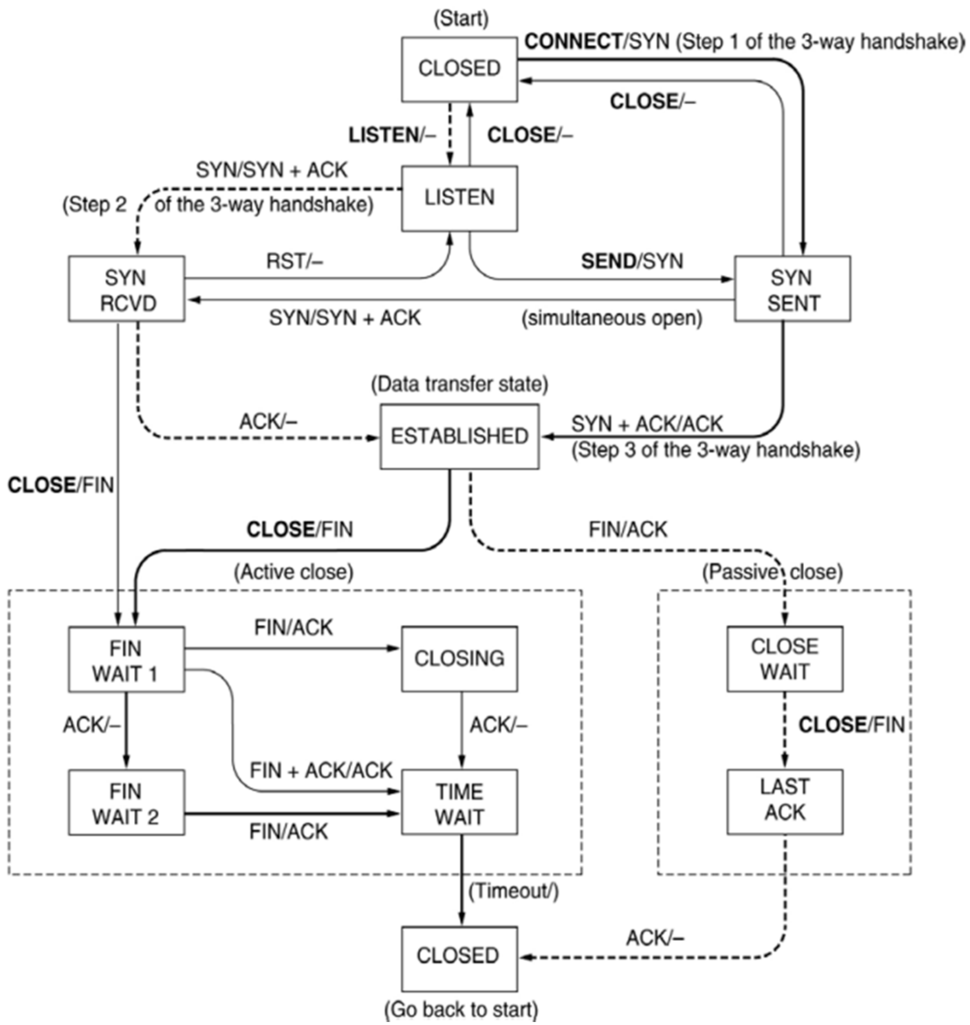


TCP connection establishment



No.	Time	Source	Destination	Protocol	Info
13	1.246280	10.0.0.125	10.0.0.121	TCP	61432 > as-debug [SYN] Seq=0 Win=8192 Len=0
14	1.246601	10.0.0.121	10.0.0.125	TCP	as-debug > 61432 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
15	1.256106	10.0.0.125	10.0.0.121	TCP	61432 > as-debug [ACK] Seq=1 Ack=1 Win=8192 Len=0
16	1.263175	10.0.0.125	10.0.0.121	TCP	61432 > as-debug [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=16
17	1.456773	10.0.0.121	10.0.0.125	TCP	as-debug > 61432 [ACK] Seq=1 Ack=17 Win=16616 Len=0
20	3.174325	10.0.0.125	10.0.0.121	UDP	Source port: domain Destination port: as-debug
24	3.314327	10.0.0.125	10.0.0.121	UDP	Source port: domain Destination port: as-debug

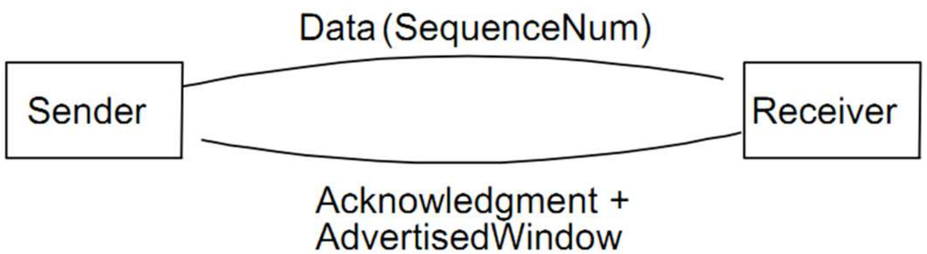
TCP Connection Management



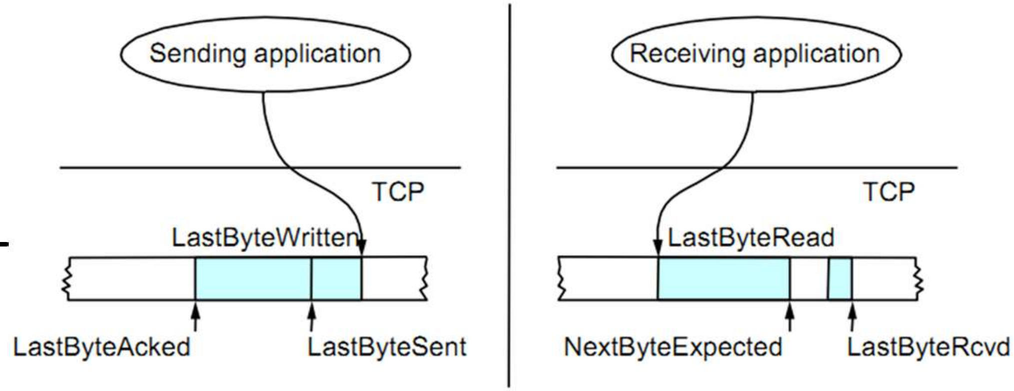
Flow control

Retransmissions in TCP A variation of Go-Back-N

- Sliding window
  - ACK contains single sequence number
  - Acknowledges all bytes with lower sequence number
- Sender retransmits single packet at a time
  - Assumes only one packet is lost (optimist)
- Error control based on byte sequences
  - not packets

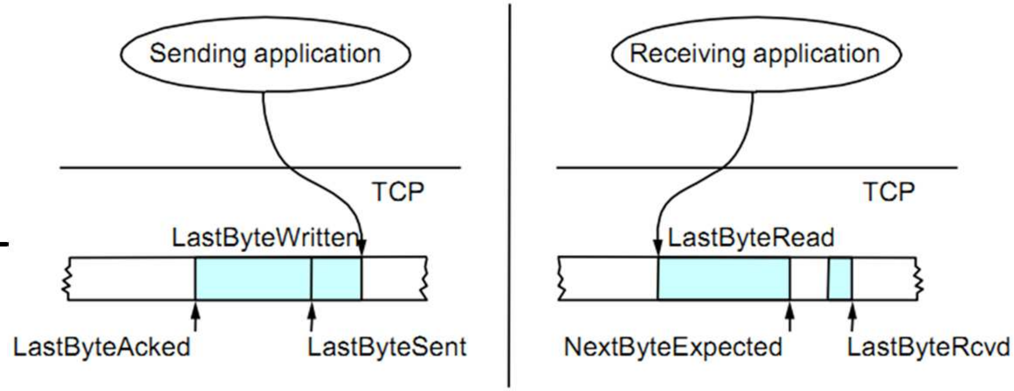


Sliding window



- Sender
  - `LastByteAcked <= LastByteSent`
  - `LastByteSent <= LastByteWritten`
  - Buffers (`LastByteWritten-LastByteAcked`) bytes
- Receiver
  - `LastByteRead < NextByteExpected`
  - `NextByteExpected <= LastByteRcvd + 1`
  - Buffers (`LastByteRcvd-LastByteRead`) bytes

Flow Control



- Buffer size
  - Sender : `MaxSendBuffer`
  - Receiver: `MaxRcvBuffer`
- Receiver
  - `LastByteRcvd - LastByteRead <= MaxRcvBuffer`
  - `AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - LastByteRead)`
- Sender
  - `LastByteWritten - LastByteAcked <= MaxSendBuffer`
  - `LastByteSent - LastByteAcked <= AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow - (LastByteSent – LastByteAcked)`
- Sending application stops
  - If it needs to write y bytes and
    - `LastByteWritten – LastByteAcked + y > MaxSendBuffer`
- ACK sent when segment is received

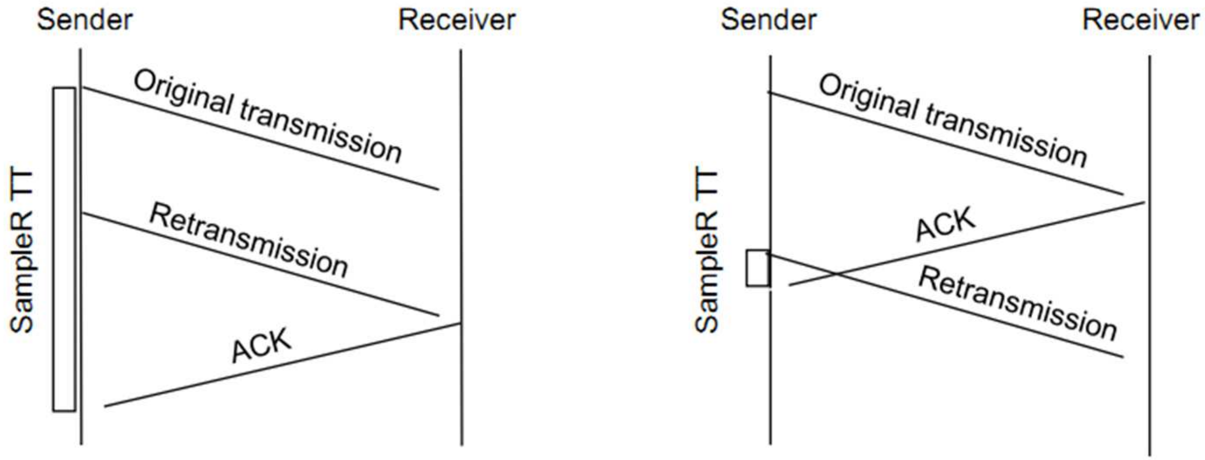
TO THINK

- Retransmission is based on a timer.
  - How does TCP pick a timeout value?

Adaptive retransmission

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

Karn/Partridge algorithm



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

Selective ACK

- Normal ACK
  - confirm all bytes up to this point
- Selective ACK
  - Acknowledges packets that arrive out-of-order
  - Adds bitmask of received packets
  - Implemented as a TCP option
- When to retransmit?
  - Packets may experience different delays
  - Still need to deal with reordering
  - Wait for 3 out of order packets

Congestion Control

Congestion control in TCP

- Congestion
  - More traffic than what the network can take
- Main idea:
  - Each source increases/decreases the traffic it generates
  - Based on criteria allowing
    - Flow fairness
    - Efficiency
- Approach:
  - Received ACKs regulate packet transmission

Congestion control in TCP

- Changes in channel capacity
  => adjustment of transmission rate

- New variable per connection: `CongestionWindow`
  - Limits the amount of traffic in the network
  - `MaxWin = MIN(CongestionWindow, AdvertisedWindow)`
  - `EffectiveWindow = MaxWin – (LastByteSent – LastByteAcked)`

- Goal:
  - If network congestion decreases
    - Increase `CongestionWindow`
  - If network congestion increases
    - Decrease `CongestionWindow`

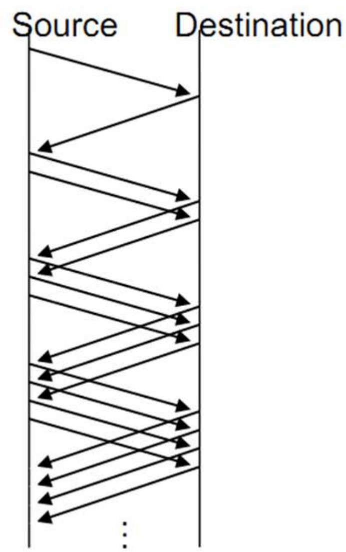
- Bitrate (byte/s) => `CongestionWindow/RTT`

Congestion control in TCP

- Need to measure congestion
  - to update value of congestion window
- How does the source know about congestion?
  - By timeout
  - Wired link => low BER => low FER
  - Timeout => loss of packet
  - Packet loss => buffer in router full => drops packets => congestion
- Wireless link?

Additive Increase / Multiplicative Decrease

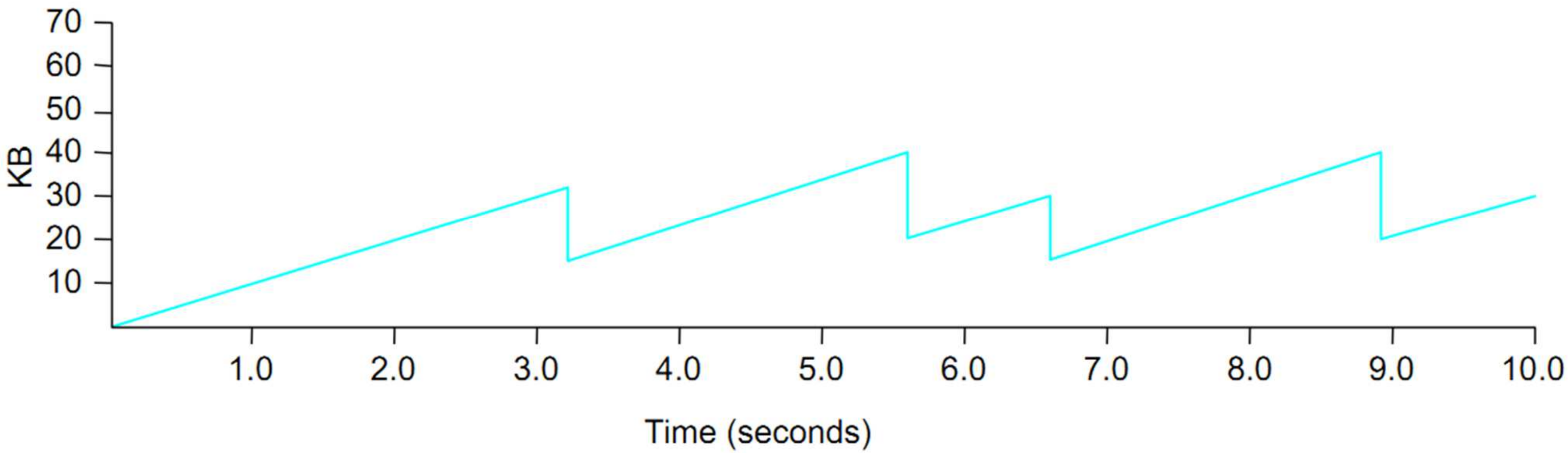
- Source knows about congestion by timeout
- How does it update `CongestionWindow`?



- Algorithm
  - Increases `CongestionWindow` by 1 segment
    - For each RTT => additive increase
  - Divide `CongestionWindow` by 2
    - When there is a packet loss => multiplicative decrease
- In practice, per received ACK
  - `Increment = MSS*(MSS/CongestionWindow)`
  - `CongestionWindow += Increment`
  - `MSS`: Maximum segment size

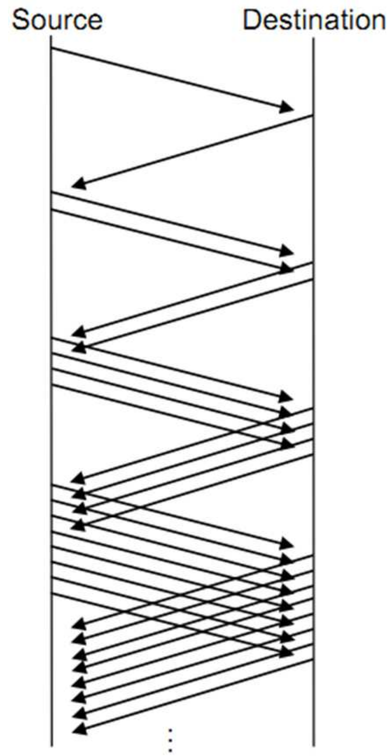
Additive Increase / Multiplicative Decrease

Sawtooth wave behavior



Slow start

- At the beginning of the connection:
  - Increase `CongestionWindow` exponentially
    - Start with `CongestionWindow=1`
    - Double `CongestionWindow` for each ACK

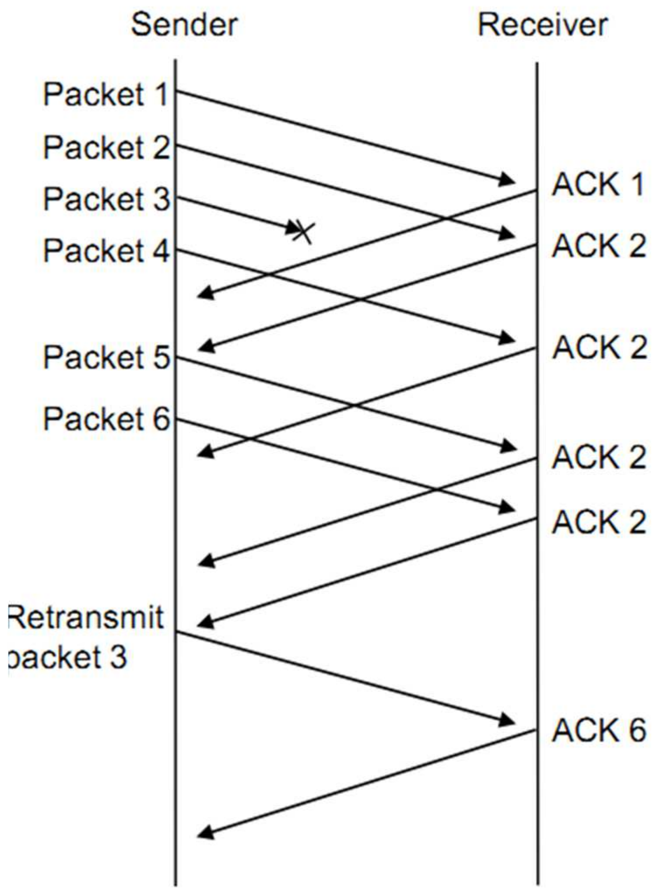


- Goal
  - More quickly determine available capacity

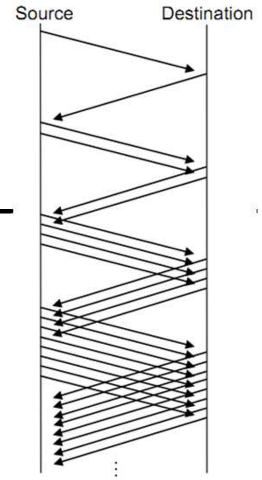
- After first timeout
  - go to linear increase
    (congestion avoidance phase)

Fast retransmission, fast recovery

- If TCP timeout too large
  - Long inactivity period
  - Retransmissions delayed
- Solution
  - Fast retransmission
  - After 3 repeated ACKs

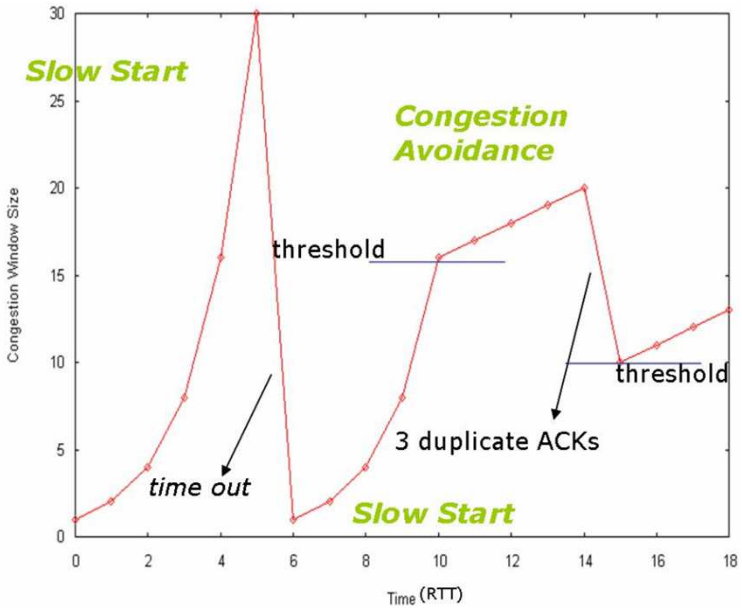


TCP - Slow start



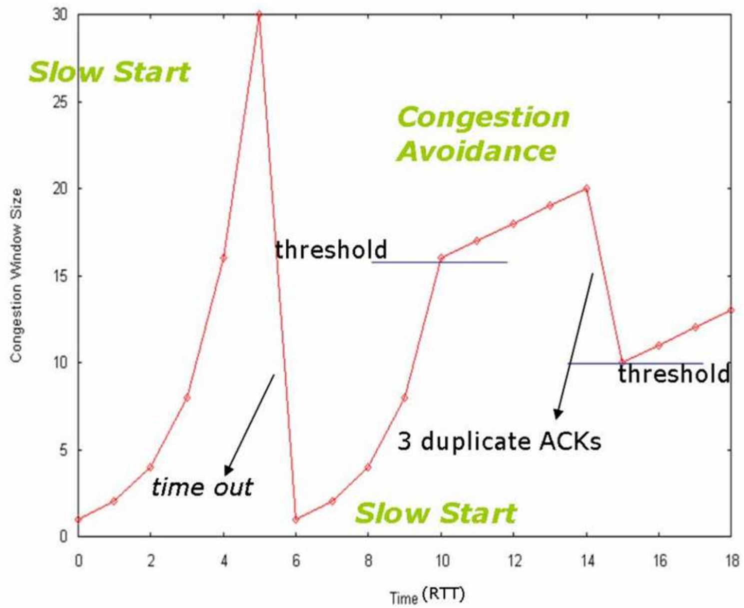
- Slow start
  - Sender starts with `CongestionWindow=1sgm`
  - Doubles `CongestionWindow` every `RTT`

- When segment loss detected by timeout:
  - `threshold = 1/2 CongestionWindow`
  - `CongestionWindow = 1 sgm`
    (so that router has time to process all packets in queue)
  - Lost packet is retransmitted
  - Slow start while
    - `CongestionWindow < threshold`
  - Then Congestion Avoidance phase
    - Linear increase



TCP - Congestion Avoidance

- Additive increase
  - Increments CongestionWindow by 1 sgm per RTT
- Detection of segment loss by 3 repeated ACKs
  - Assumes packet is lost
    - Cause is not severe congestion
      - Some of the following packets have arrived
  - Retransmits lost packet
  - `CongestionWindow=CongestionWindow/2`
  - Congestion Avoidance phase



TCP - Congestion Control

- In reality Congestion Control is a bit more complex
- RFC2581
  - "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms"

- What services are provided by the transport layer?
- What are the transport protocols of the TCP/IP stack?
- What’s the difference between TCP and UDP?
- How is the connection established in TCP?
- What’s the difference between:
  - Flow control
  - Congestion control
- How does TCP implement flow control?
- What are the congestion control mechanisms in TCP?
- Why is TCP congestion control so important to the Internet?

HOMEWORK

- Review slides
- Read “The Transport Layer” from:
  - Tanenbaum – Chap. 5
- Do your Moodle homework