Welcome to CPSC 441!

TA: Xifan Zheng

Email: zhengxifan0403@gmail.com

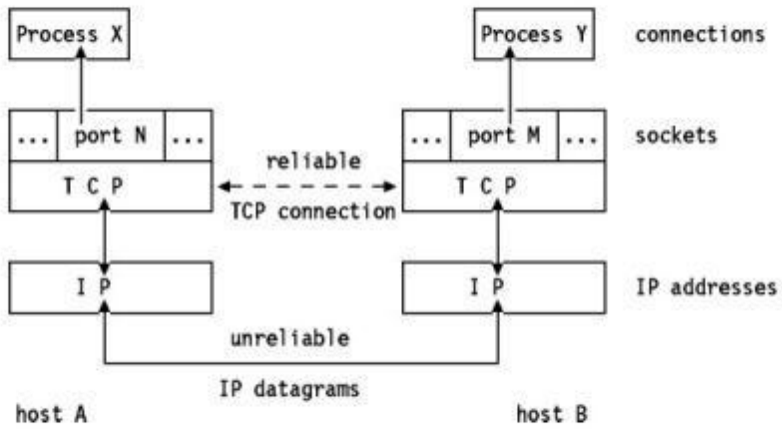
Today's Tutorial: Review of TCP

- Introduction to TCP (Transmission Control Protocol)
- TCP segment structure
- Sequence number/acknowledgement number
- TCP connection management
- Round-trip time estimation and Timeout
- Flow Control
- Congestion Control

Parts of the slide contents are taken from CPSC 641 by Prof. Carey Williamson

The TCP Protocol

- Connection-oriented, point-to-point protocol:
  - Connection establishment and teardown phases
  - ‘Phone-like’ circuit abstraction (application-layer view)
  - One sender, one receiver
  - Called a “reliable byte stream” protocol
  - General purpose (for any network environment)

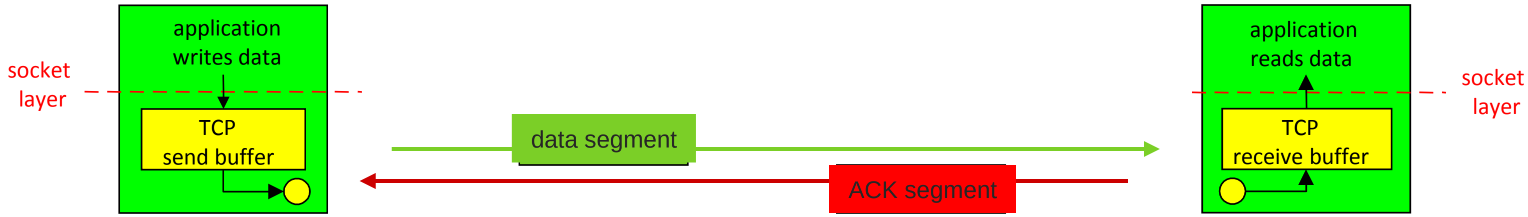


- Connection-oriented: the two applications using TCP must establish a TCP connection with each other before they can exchange data

The TCP Protocol

- TCP provides the following facilities to
  - Stream Data Transfer: Transfers a contiguous stream of bytes in TCP segments
  - Multiplexing: allow for many processes within a single host to use TCP communication facilities simultaneously.
  - Reliability
  - Flow Control and congestion control
- Originally optimized for certain kinds of transfer:
  - Telnet (interactive remote login)
  - FTP (long, slow transfers)
  - Web is like neither of these!

TCP Protocol (cont)

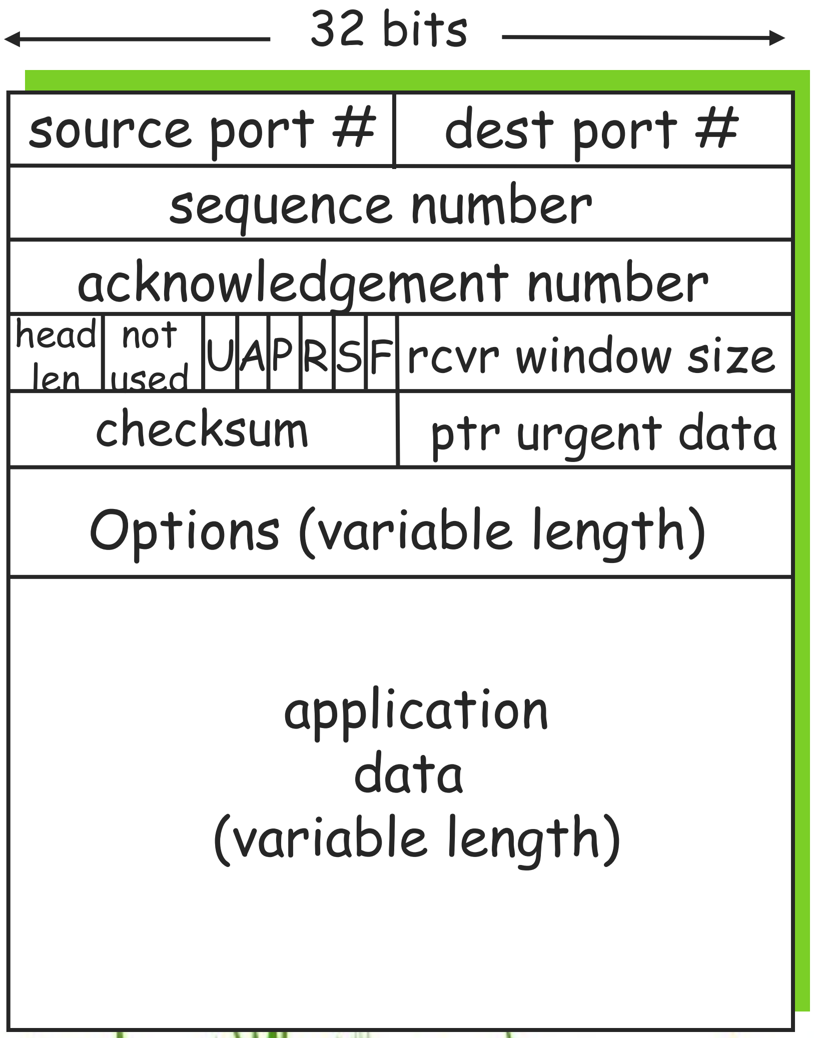


- Provides a reliable, in-order, byte stream abstraction:
  - Recover lost packets and detect/drop duplicates
  - Detect and drop corrupted packets
  - Preserve order in byte stream, no “message boundaries”
  - Full-duplex: bi-directional data flow in same connection
- Flow and congestion control:
  - Flow control: sender will not overwhelm receiver
  - Congestion control: sender will not overwhelm the network
  - Sliding window flow control
  - Send and receive buffers
  - Congestion control done via adaptive flow control window size

The TCP Header

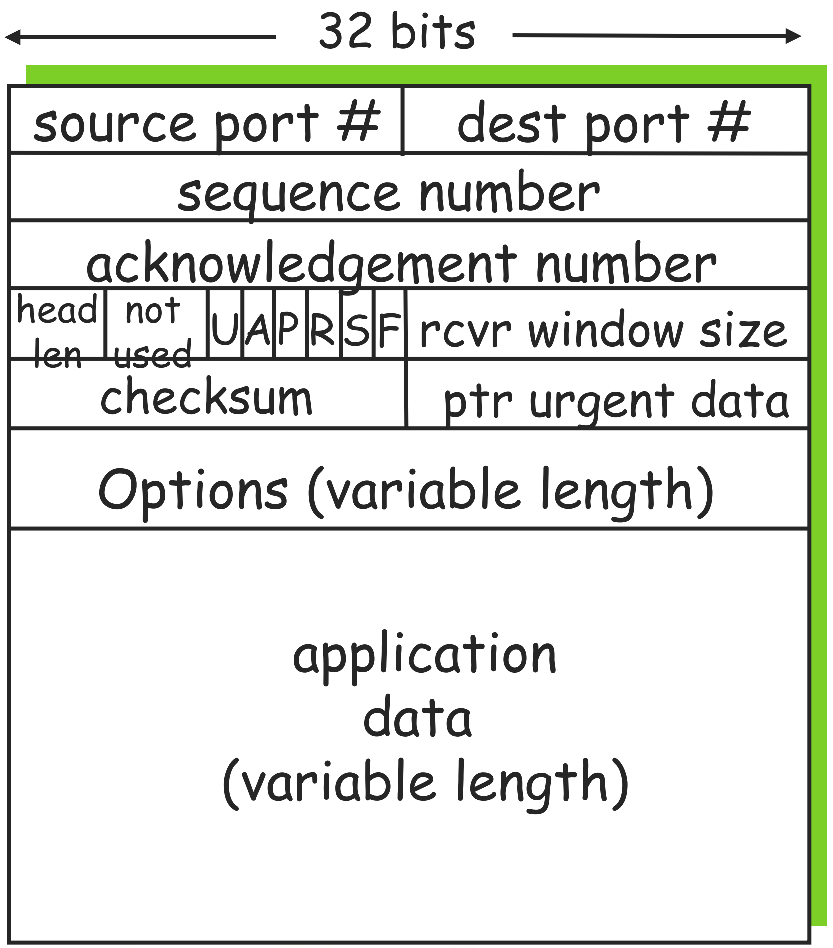
Fields in the header:

- **SrcPort** and **DstPort**: These two fields plus the source and destination IP addresses, combine to uniquely identify each TCP
- **Sequence number**: the byte in the stream of data from the sending TCP to the receiving TCP that the first byte of data in this segment represents
- **Acknowledgement number:** contains the next sequence number that the sender expects to receive. This acknowledges receipt of all prior bytes. This field is valid only if the ACK flag is on.



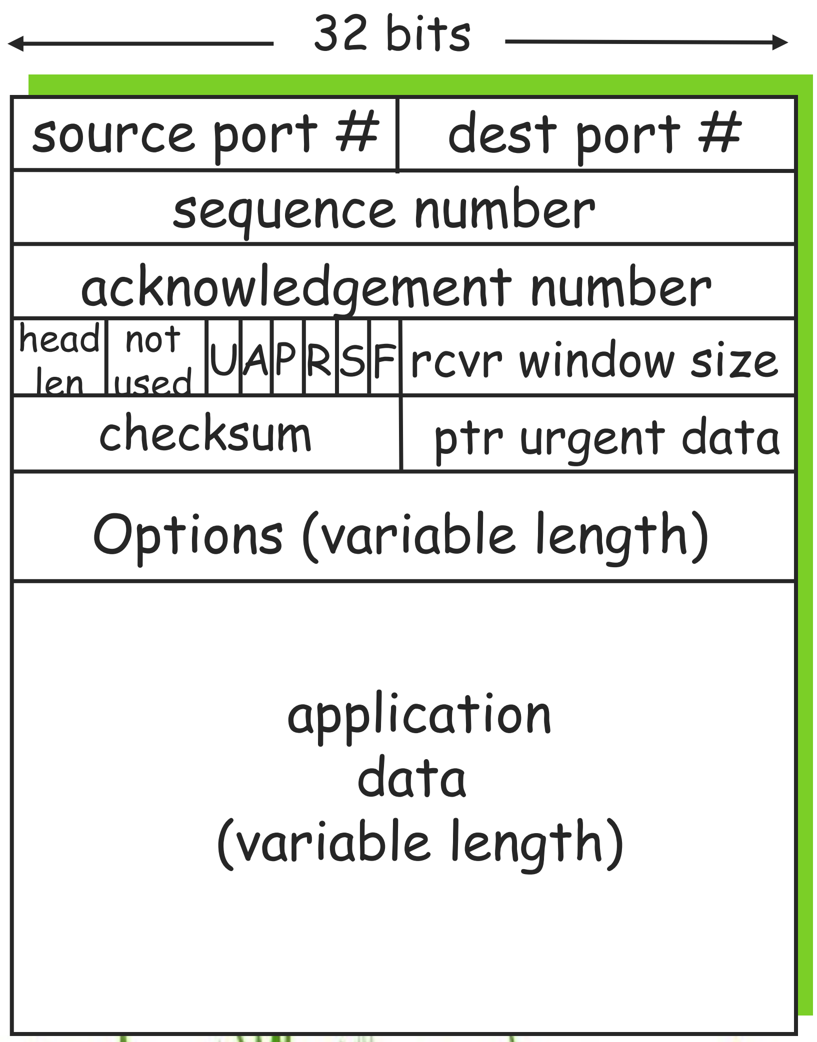
The TCP Header

- **header length:** gives the length of the header in 32-bit words
- **Flags (6 bits):**
  - URG – this segment contains urgent data. When this flag is set, the UrgPtr field indicates where the non-urgent data contained in this segment begins
  - ACK – indicates that the Acknowledgment field is significant. **All packets after the initial SYN packet** sent by the client should have this flag set.
  - PSH – Push function. Asks to push the buffered data to the receiving application.
  - RST– Reset the connection
  - SYN – Synchronize sequence numbers. **Only the first packet** sent from each end should have this flag set.
  - FIN – No more data from sender



The TCP Header

- **Not used:** for future use and should be set to zero
- **Receive window:** the number of bytes that the receiver is currently willing to receive
- **Checksum:** The 16-bit checksum field is used for error-checking of the header and data
- **Option field:** has many different options. For example, maximum segment size option, called the MSS. It specifies the maximum sized segment the sender wants to receive
- The **data** portion of the TCP segment is optional.

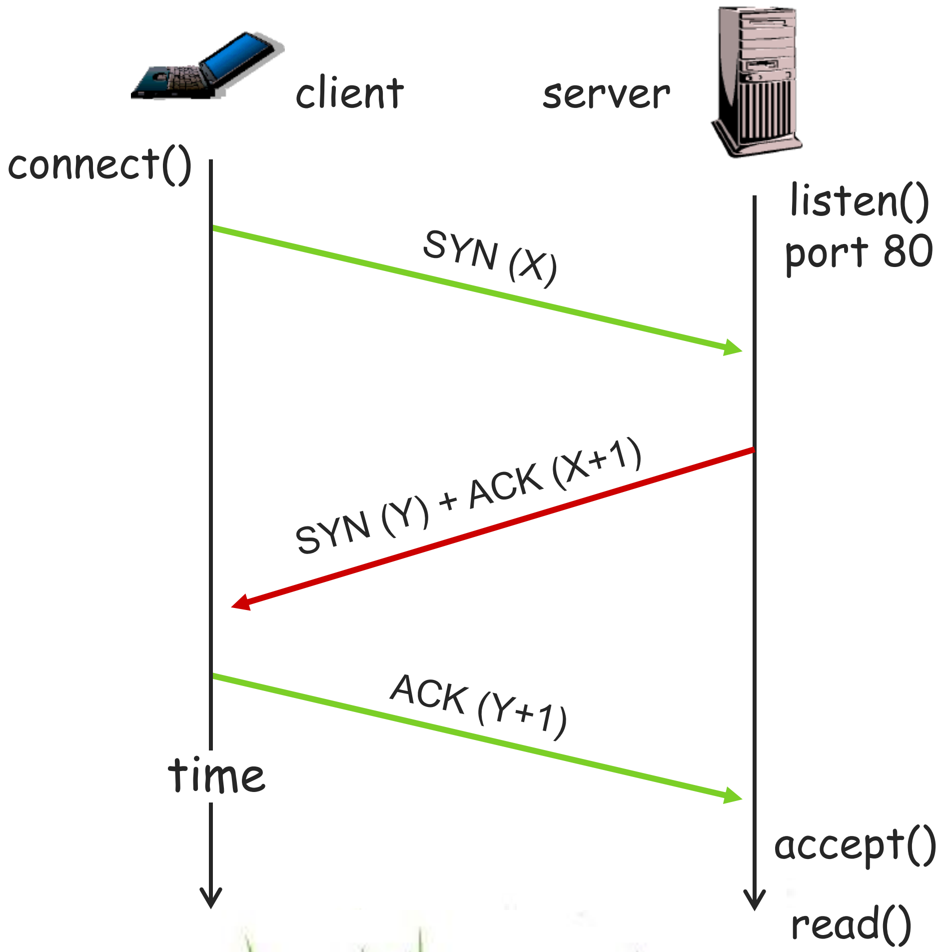


Sequence number/acknowledgement number

- **Sequence number**: the byte-stream number of the first byte in the segment
- **Acknowledgment number**: the sequence number that the host expect to receive

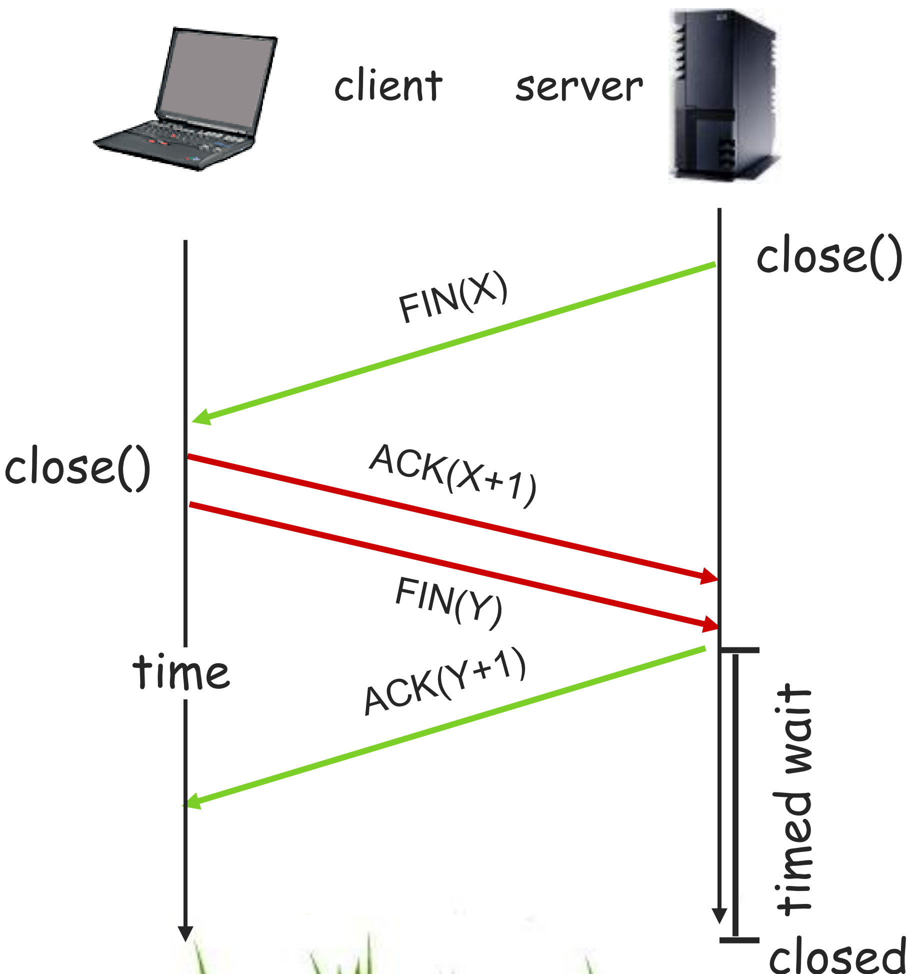
Establishing a TCP Connection

- Client sends SYN with initial sequence number (ISN = X)
- Server responds with its own SYN w/seq number Y and ACK of client ISN with X+1 (next expected byte)
- Client ACKs server's ISN with Y+1
- The ‘3-way handshake’
- X, Y randomly chosen
- All modulo 32-bit arithmetic



Connection Termination

- Either side may terminate a connection. (In fact, connection can stay half-closed.) Let's say the server closes (typical in WWW)
- Server sends FIN with seq Number (X) (i.e., FIN is a byte in sequence)
- Client ACK's the FIN with X+1 ("next expected")
- Client sends it's own FIN when ready
- Server ACK's client FIN as well with Y+1.

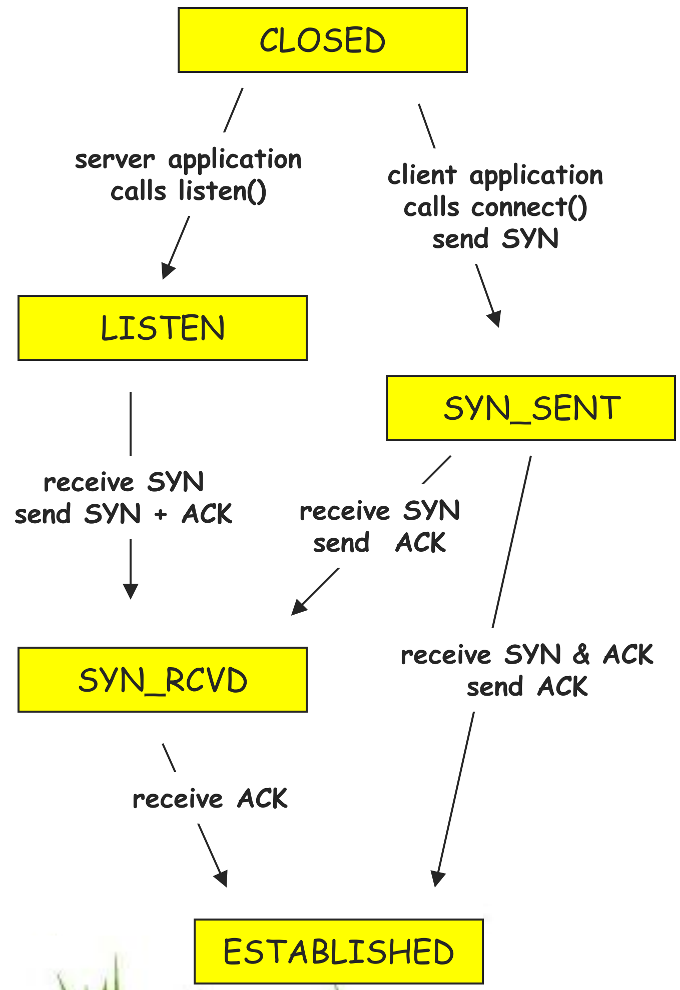


The TCP State Machine

- TCP uses a Finite State Machine, kept by each side of a connection, to keep track of what state a connection is in.
- State transitions reflect inherent races that can happen in the network, e.g., two FIN's passing each other in the network.
- Certain things can go wrong along the way, i.e., packets can be dropped or corrupted. In fact, machine is not perfect; certain problems can arise not anticipated in the original RFC.
- This is where timers will come in, which we will discuss more later.

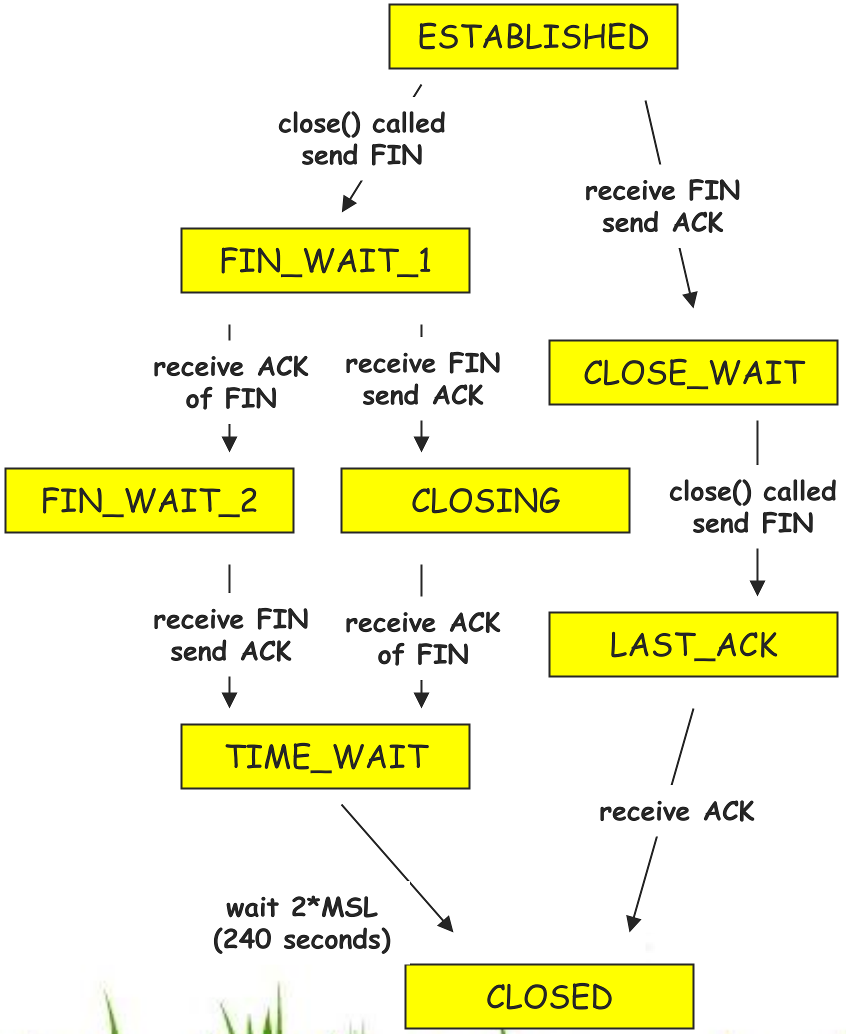
TCP Connection Establishment

- CLOSED: more implied than actual, i.e., no connection
- LISTEN: willing to receive connections (accept call)
- SYN-SENT: sent a SYN, waiting for SYN-ACK
- SYN-RECEIVED: received a SYN, waiting for an ACK of our SYN
- ESTABLISHED: connection ready for data transfer

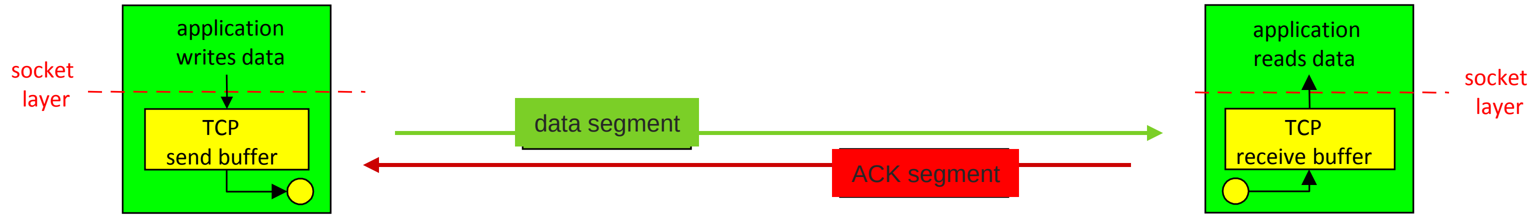


TCP Connection Termination

- FIN-WAIT-1: we closed first, waiting for ACK of our FIN (active close)
- FIN-WAIT-2: we closed first, other side has ACKED our FIN, but not yet FIN'ed
- CLOSING: other side closed before it received our FIN
- TIME-WAIT: we closed, other side closed, got ACK of our FIN
- CLOSE-WAIT: other side sent FIN first, not us (passive close)
- LAST-ACK: other side sent FIN, then we did, now waiting for ACK



Sending Data



- Sender TCP passes segments to IP to transmit:
  - Keeps a copy in buffer at send side in case of loss
  - Called a “reliable byte stream” protocol
  - Sender must obey receiver advertised window
- Receiver sends acknowledgments (ACKs)
  - ACKs can be piggybacked on data going the other way
  - Protocol allows receiver to ACK every other packet in attempt to reduce ACK traffic (delayed ACKs)
  - Delay should not be more than 500 ms. (typically 200 ms)

Example

Three way handshake

- 1629.884415 192.168.1.9 -> 136.159.5.17 44 TCP 1035 80 133227 : 133227 0 win: 32768 S
- 1629.886713 136.159.5.17 -> 192.168.1.9 44 TCP 80 1035 3310607972 : 3310607972 133228 win: 24820 SA
- 1629.888507 192.168.1.9 -> 136.159.5.17 40 TCP 1035 80 133228 : 133228 3310607973 win: 32768 A
- *Timestamp Src IP → Dst IP* ***IP pkt size*** *Protocol SrcPort DstPort* ***Seq num*** ***ACK num*** *win size flag*
- 1629.948462 192.168.1.9 -> 136.159.5.17 418 TCP 1035 80 133228 : 133606 3310607973 win: 32768 PA
- 1629.952320 136.159.5.17 -> 192.168.1.9 40 TCP 80 1035 3310607973 : 3310607973 133606 win: 24820 A
- 1629.955295 136.159.5.17 -> 192.168.1.9 329 TCP 80 1035 3310607973 : 3310608262 133606 win: 24820 PA
- 1629.959145 136.159.5.17 -> 192.168.1.9 1500 TCP 80 1035 3310608262 : 3310609722 133606 win: 24820 A
- 1629.960963 136.159.5.17 -> 192.168.1.9 1500 TCP 80 1035 3310609722 : 3310611182 133606 win: 24820 PA
- 1629.962090 192.168.1.9 -> 136.159.5.17 40 TCP 1035 80 133606 : 133606 3310609722 win: 31019 A

Data transmission

How are losses recovered? – estimate RRT

What if packet is lost (data or ACK!)

- Coarse-grained Timeout:
  - Sender does not receive ACK after some period of time
  - Event is called a retransmission time-out (RTO)
  - RTO value is based on estimated round-trip time (RTT)
  - RTT is adjusted over time using exponential weighted moving average:

    RTT = (1-x)*RTT + (x)*sample

    (x is typically 0.125)

Flow Control V.S. Congestion Control

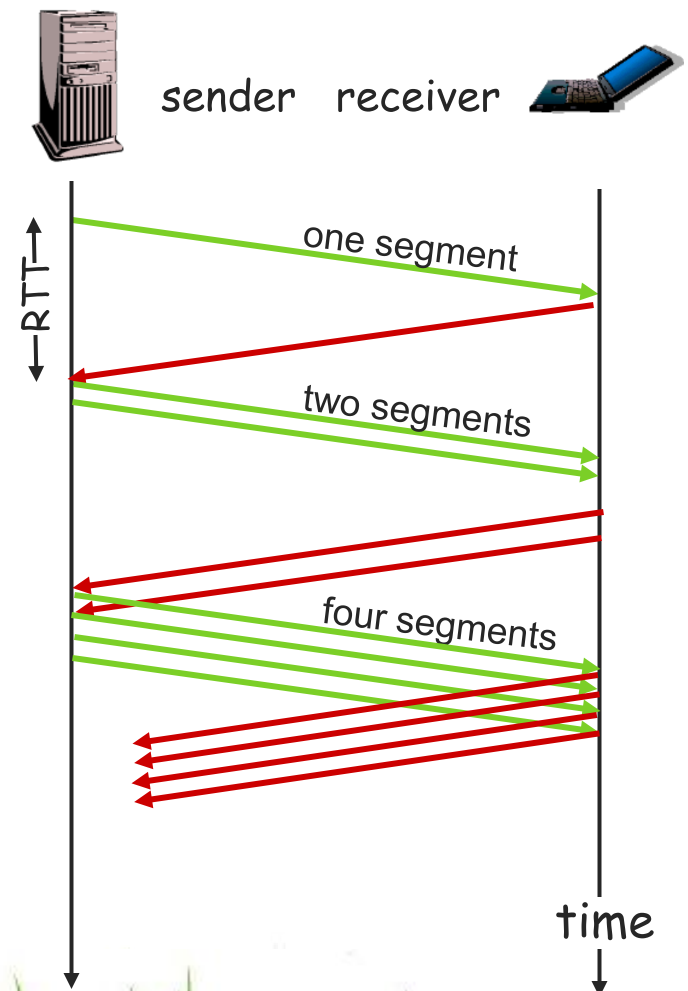
- **Flow Control**: matching the rate at which the sender is sending against the rate at which the receiving application is reading
- **Congestion Control**: preventing a TCP sender overfeed the IP network

Preventing Network Congestion

- Sender may not only overrun receiver, but may also overrun intermediate routers:
  - No way to explicitly know router buffer occupancy, so we need to infer it from packet losses
  - Assumption is that losses stem from congestion, namely, that intermediate routers have no available buffers
- Sender maintains a congestion window:
  - Never have more than CW of un-acknowledged data outstanding (or RWIN data; min of the two)
  - Successive ACKs from receiver cause CW to grow.
- How CW grows based on which of 2 phases:
  - Slow-start: initial state.
  - Congestion avoidance: steady-state.
  - Switch between the two when CW > slow-start threshold

Slow Start

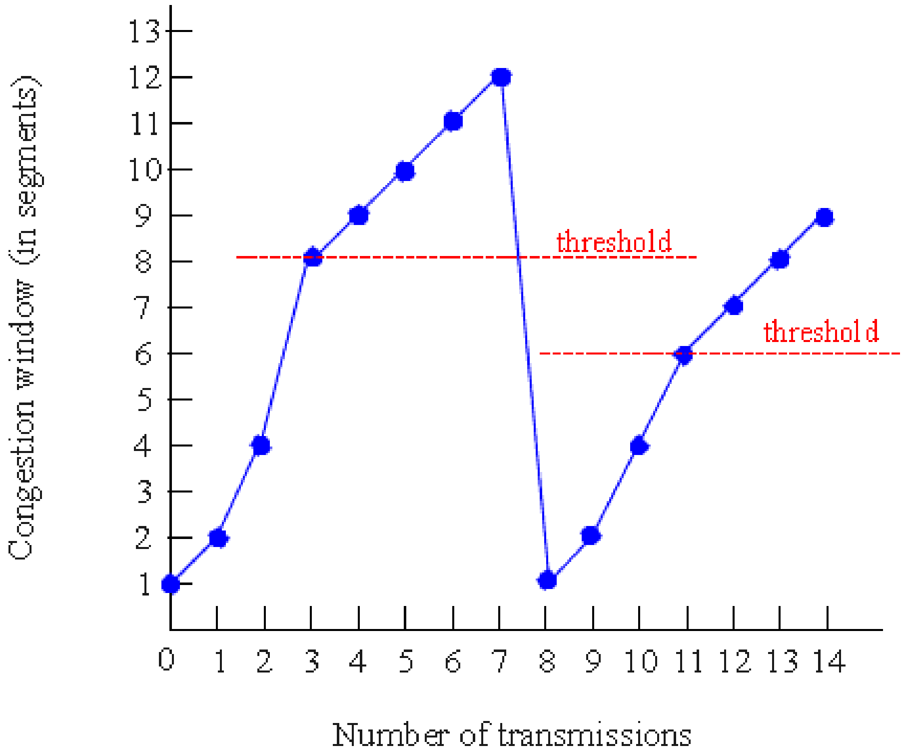
- Initial CW = 1.
- After each ACK, CW += 1;
- Continue until:
  - Loss occurs OR
  - CW > slow start threshold
- Then switch to congestion avoidance
- If we detect loss, cut CW in half
- Exponential increase in window size per RTT



Congestion Avoidance

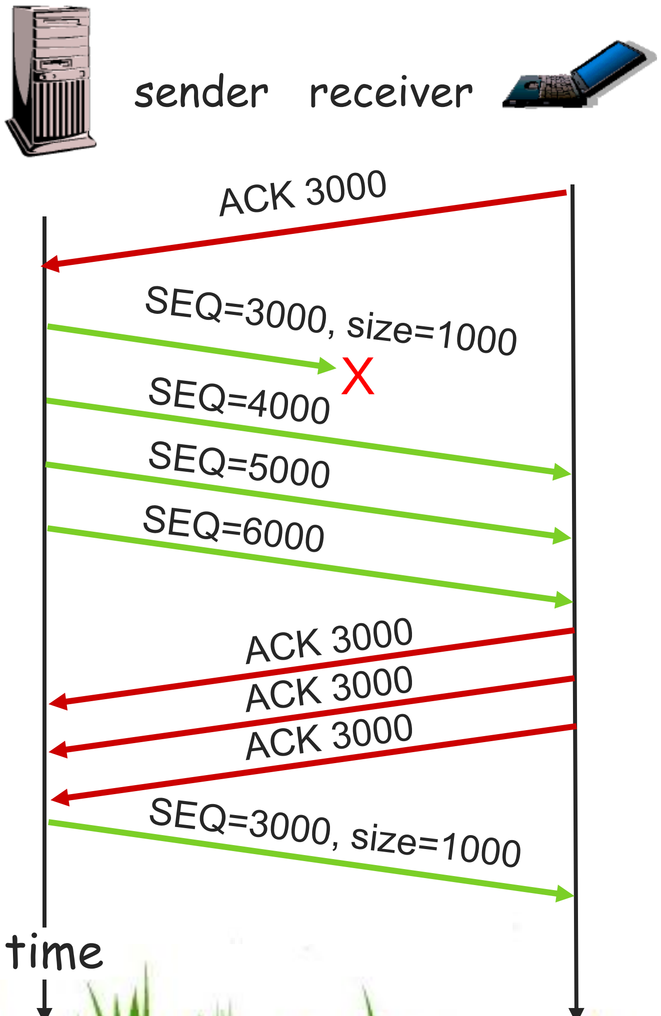
Until (loss) {
 after CW packets ACKed:
   CW += 1;
}
ssthresh = CW/2;
Depending on loss type:
  SACK/Fast Retransmit:
    CW/= 2; continue;
  Course grained timeout:
    CW = 1; go to slow start.

(This is for TCP Reno/SACK: TCP Tahoe always sets CW=1 after a loss)



Fast Retransmit

- Receiver expects N, gets N+1:
  - Immediately sends ACK(N)
  - This is called a duplicate ACK
  - Does NOT delay ACKs here!
  - Continue sending dup ACKs for each subsequent packet (not N)
- Sender gets 3 duplicate ACKs:
  - Infers N is lost and resends
  - 3 chosen so out-of-order packets don't trigger Fast Retransmit accidentally
  - Called "fast" since we don't need to wait for a full RTT



Introduced in TCP Reno

Thanks for attending!