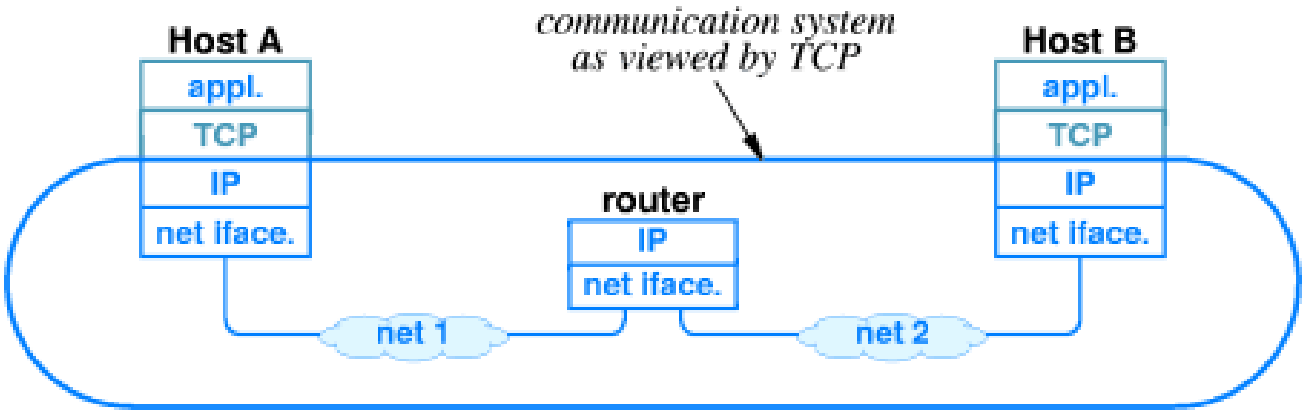
# Transmission Control Protocol (TCP)

# TCP Services Offered to an Application

- Connection orientation
- Point-to-Point
- Reliability
- Full duplex
- Stream interface
- Reliable startup
- Graceful connection shutdown

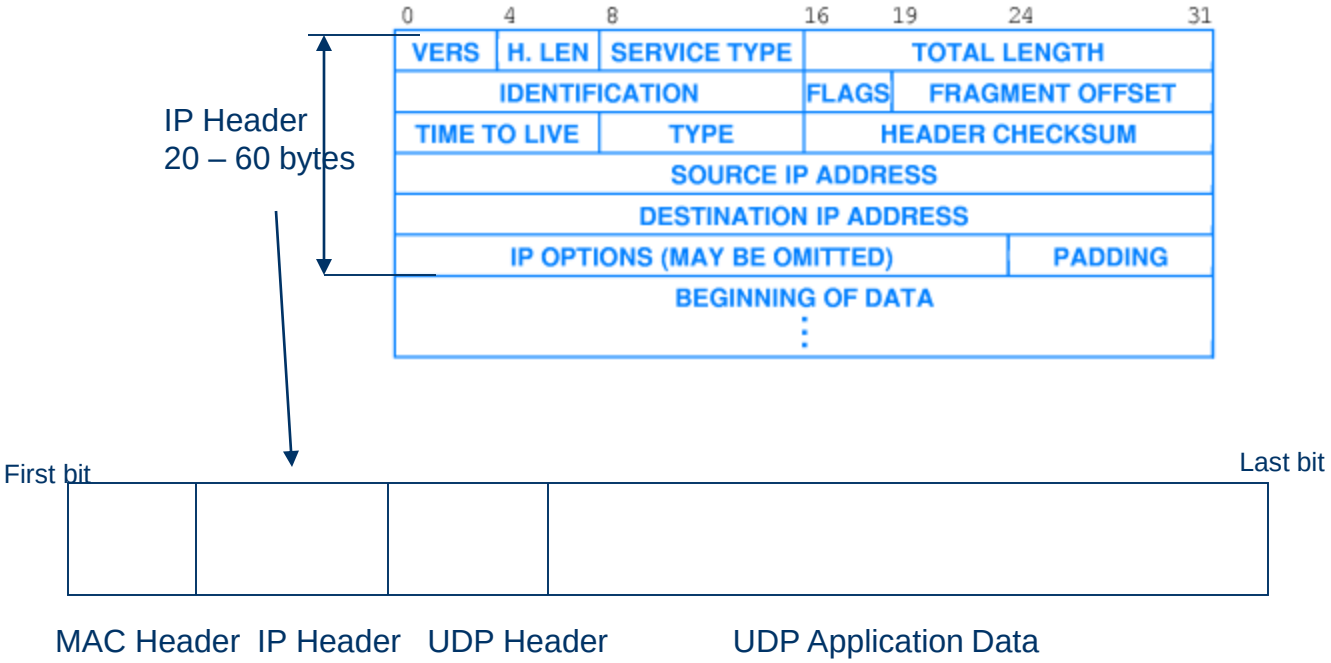
# Transmission Control Protocol (TCP)




•TCP is an end-to-end protocol because it provides a connection directly from one application to another running on a remote computer.

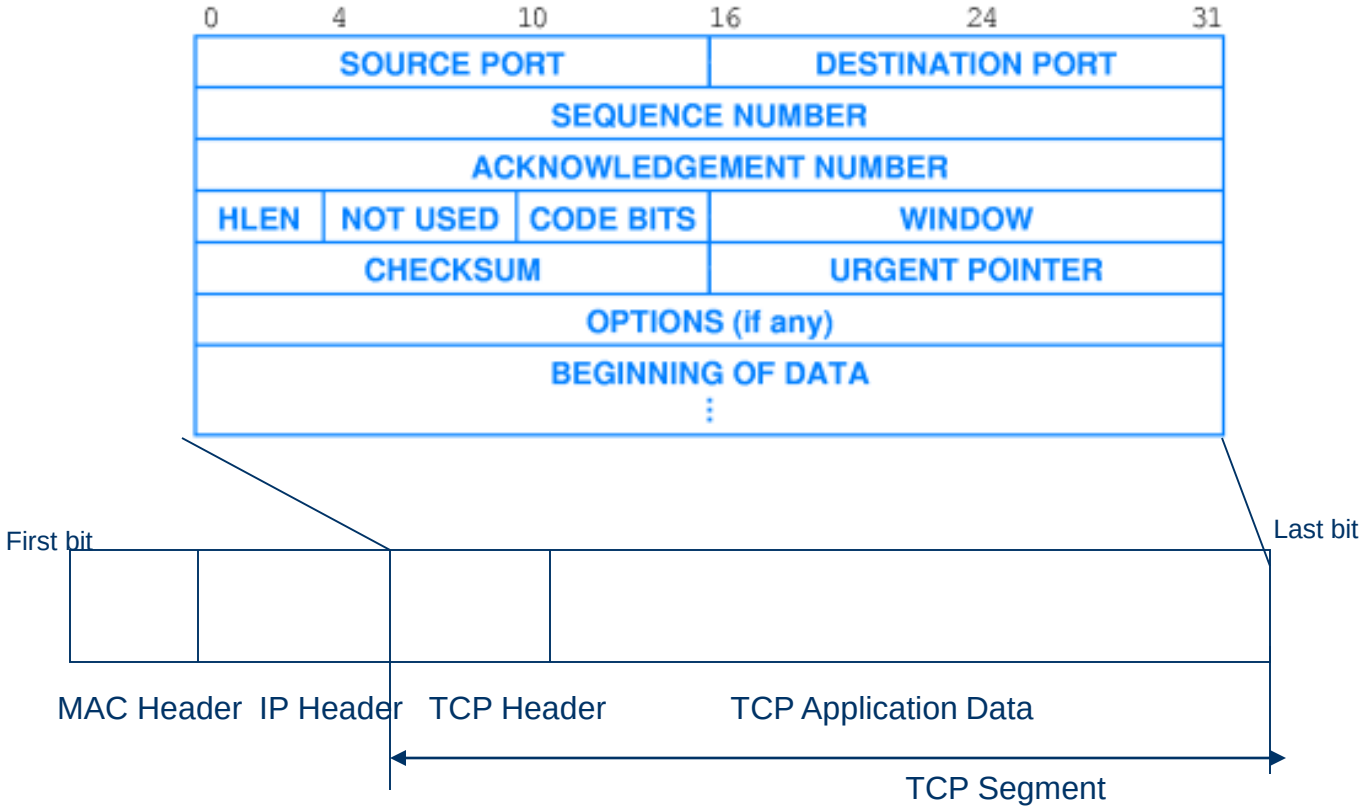
•The connections are virtual connections because they are achieved in software.

# IP Datagram Format: UDP example

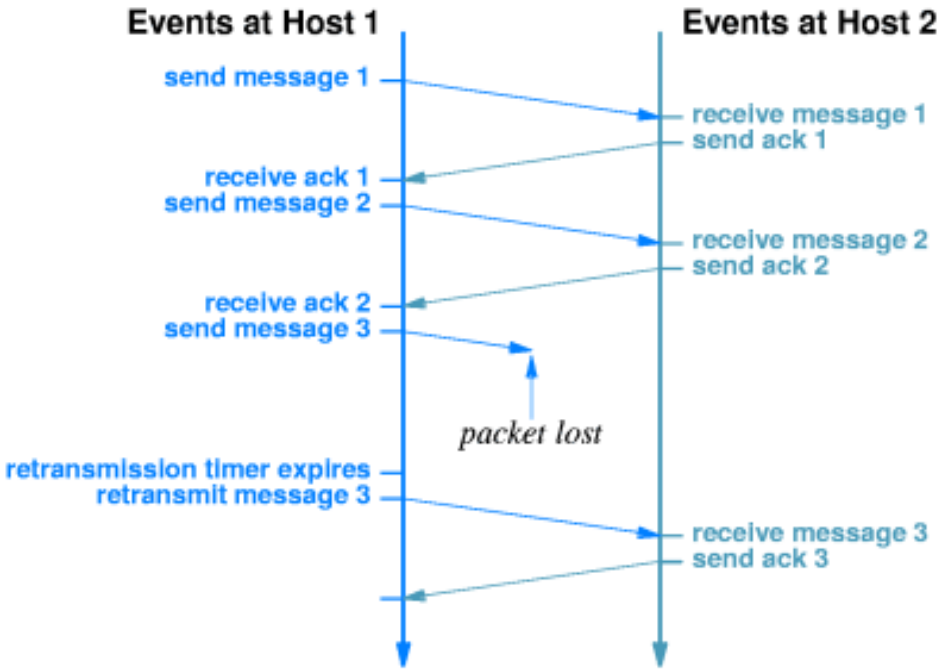



This is what gets sent 'on the wire': A frame which contains an IP Packet

# TCP Datagram Format



# Packet Loss and Retransmission




Example of retransmission. Items on the left correspond to events in a computer sending data, items on the right correspond to events in a computer receiving data, and time goes down the figure. The sender retransmits lost data.

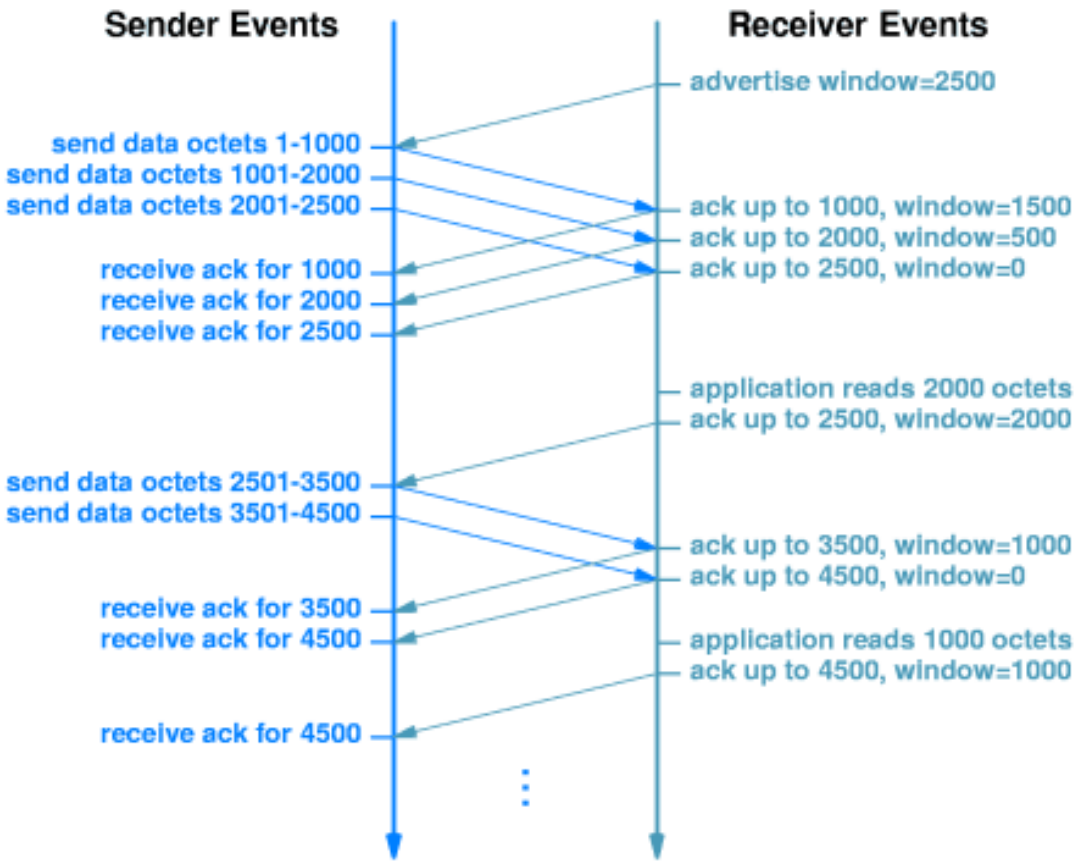
# Transmission Control Protocol (TCP)

- Error detection/recovery
- Adaptive retransmission:
  - If the retransmission timeout is too low, you might retransmit unnecessarily.
  - If the timeout is too high, user perceived performance can be very bad.

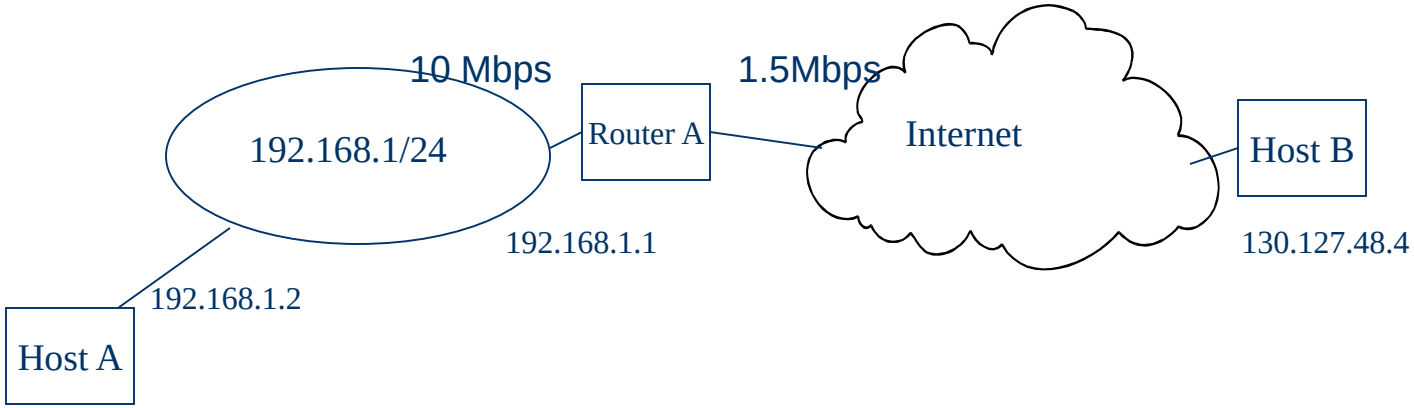
# Flow Control

- A TCP receiver allocates a receive buffer.
- As data arrives (and fills the buffer), the receiver sends ACKs that also specify the remaining buffer size.
- The amount of buffer space available at any time is the window and the notification that specifies the size is the window advertisement.

# Flow Control



# Network Congestion Control




- What if Host A sends many back-to-back UDP Echo Messages to Host B ?
- TCP's base congestion control algorithm based on a dynamic window (slow start)
  - cwnd = 1; //init the congestion window to 1 segment
  - wnd = min(cwnd, adverstised window) //The actual window used…
  - On a timeout, cwnd = 1
  - When a new ACK arrives
    - cwnd += 1;
- Lead's to TCP's 'saw-tooth behavior'

# TCP Sockets Programming

| TCP Client | TCP Server |
| --- | --- |
| | socket()<br>bind()<br>listen()<br>accept() |
| socket()<br>connect() | |
| write() | read() |
| read() | write() |
| close() | read()<br>close() |

# Server Side Calls: Listen and Accept

- Listen
  - Converts an active socket into a passive socket meaning the kernel should accept incoming connection requests.
  - Sets the maximum number of connections the kernel should queue for this socket.

    *int listen (int sockfd, int backlog)*
  - There are two queues:
    - incomplete cx queue
    - completed cx queue
  - the backlog (roughly) indicates the sum of the two queues
- Accept returns a Cx from the completed queue

  *int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addren)*
  - Returns a new socket descriptor.
    - Server will have at least a listenfd and a connectfd socket descriptors

# TCP startup/tear down

- TCP connection setup:  3 way handshake
  - guarantees that both sides are ready to transfer data (handles simultaneous opens)
  - Allows both sides to agree on initial sequence numbers (ISNs).
- TCP connection termination:
  - modified  3 way handshake
  - The TCP close requires 4 flows rather than 3
  - A FIN leads to an EOF to a receiving application
  - Supports a half close:  one side terminates its output but still receives data from the other side.
  - Sockets supports this-  but most applications don't use it.

# TCP State Machine

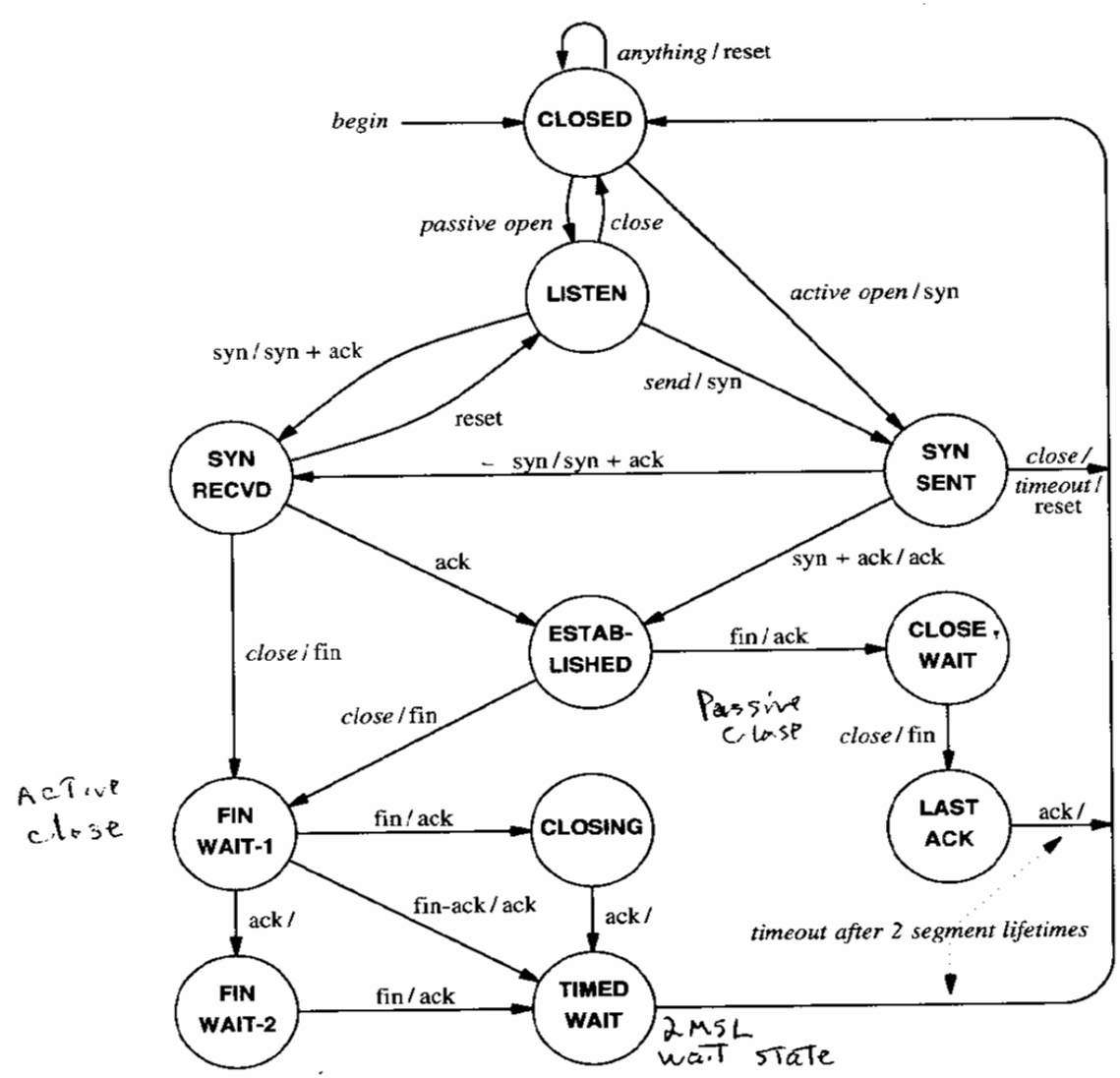
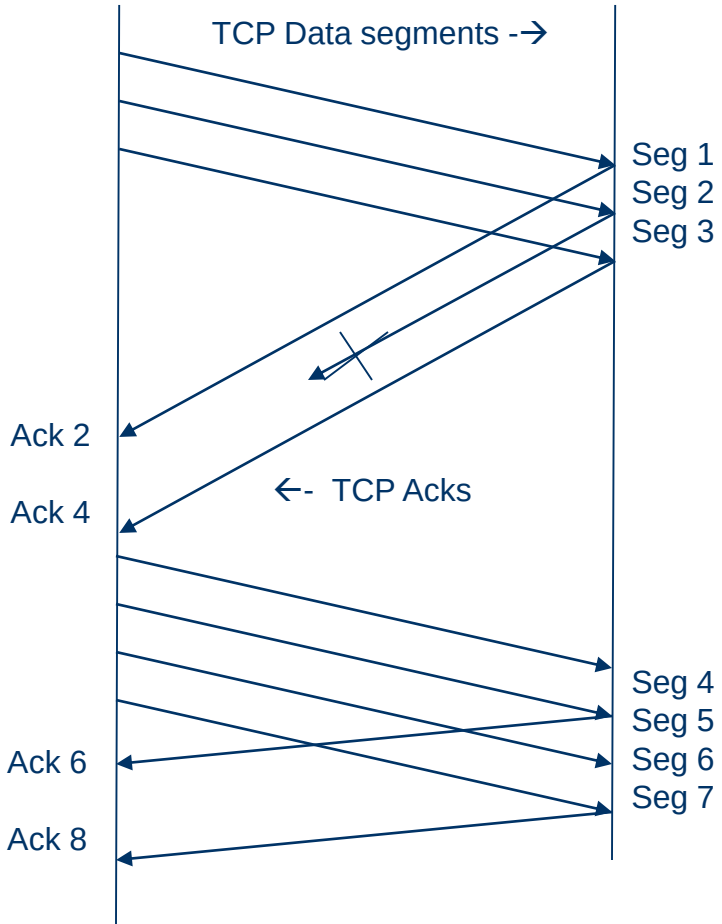


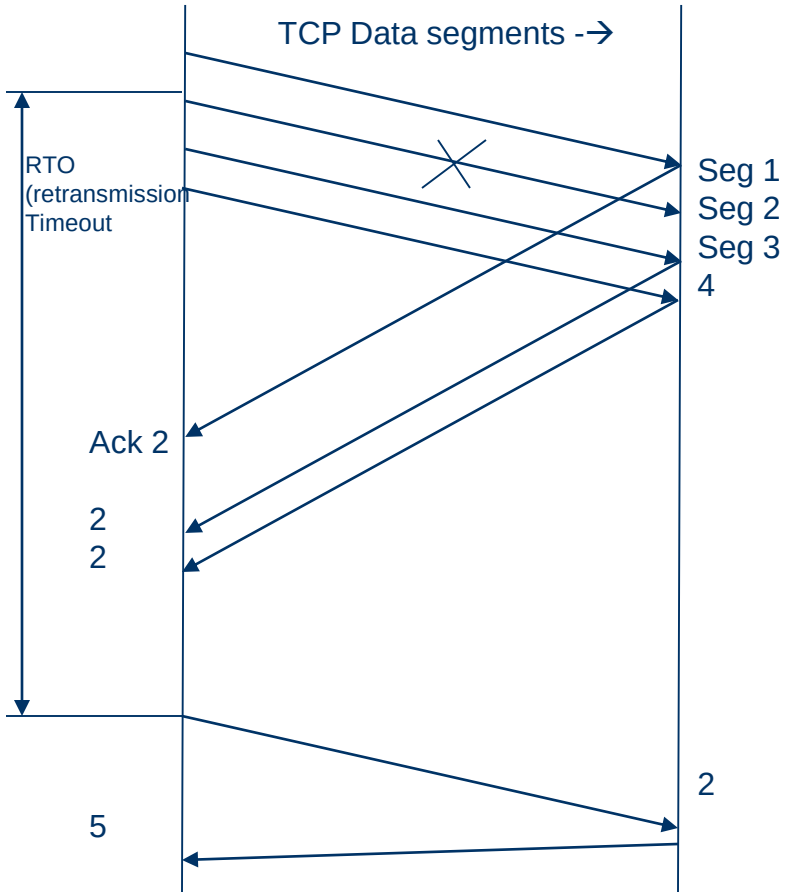

**Figure 13.15** The TCP finite state machine. Each endpoint begins in the *closed* state. Labels on transitions show the input that caused the transition followed by the output if any.

# TCP Acknowledgement Strategy



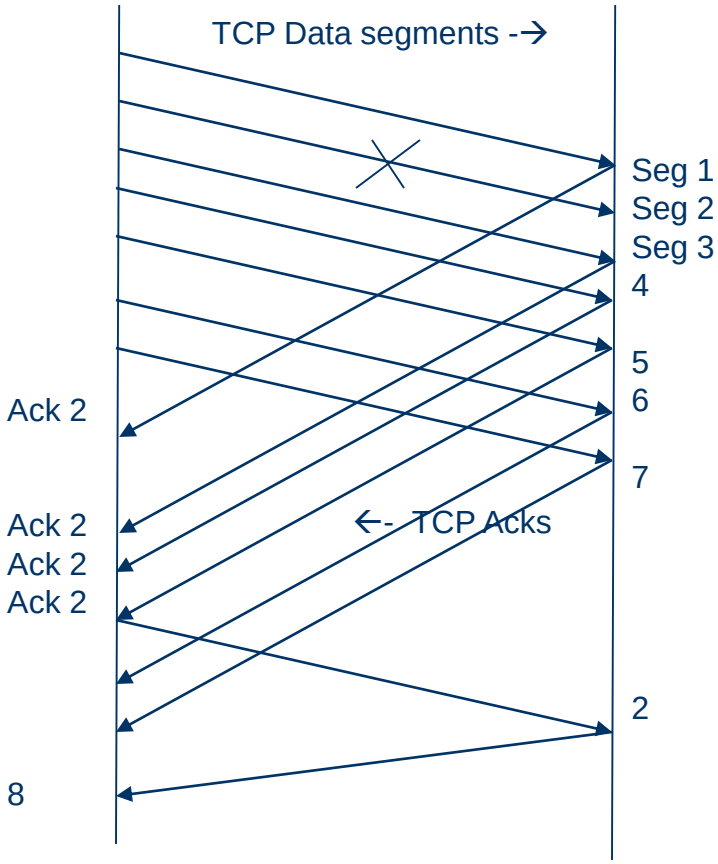

- Acks indicate the next byte the receiver expects (we show the next packet that is expected)
- Acks are cumulative
  - Easy to generate
  - But they are ambiguous.
  - Lost Acks don't necessarily force a retransmission.
- Ack strategy: an Ack is typically generated for every 2 segments that arrive

# Congestion Avoidance Algorithm




- Combined slow start and congestion avoidance algorithm (Tahoe)
- create a second state variable, ssthresh, to switch between the two algorithms.
- Assume the wnd = min(cwnd, adverstised window)
- On a timeout
  - ssthresh = wnd / 2 (min value of 2 segments)
  - cwnd = 1
- When a new ACK arrives
  - if (cwnd <= ssthresh)
    - /*open the window exponentially */
    - cwnd += 1;
  - else
    - /*otherwise do Congestion Avoidance-increment linearly */
    - cwnd += 1/cwnd;

# Fast Recovery, Fast Retransmit Algorithms




•Fast Retransmit:

- •If three or more duplicate ACKs arrive at a sender, this is a strong indicator that a packet was dropped.
- •The sender retransmits without waiting for the retransmit timer.

•Fast Recovery: After a fast retransmit, the sender goes to congestion avoidance rather than slow-start.

•Enhanced algorithm:

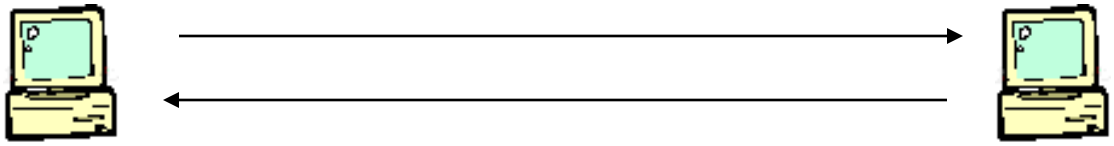
- •When the third duplicate ACK is received
  - set ssthresh = cwnd/2
  - retransmit the segment
  - set cwnd = ssthresh + 3 packets
- •For each additional duplicate ACK (after the third duplicate ACK), increment cwnd by 1 and transmit a new packet (if allowed by the new cwnd value).
- •When the next ACK arrives that acknowledges new data, set cwnd to ssthresh.

# TCPDump Trace

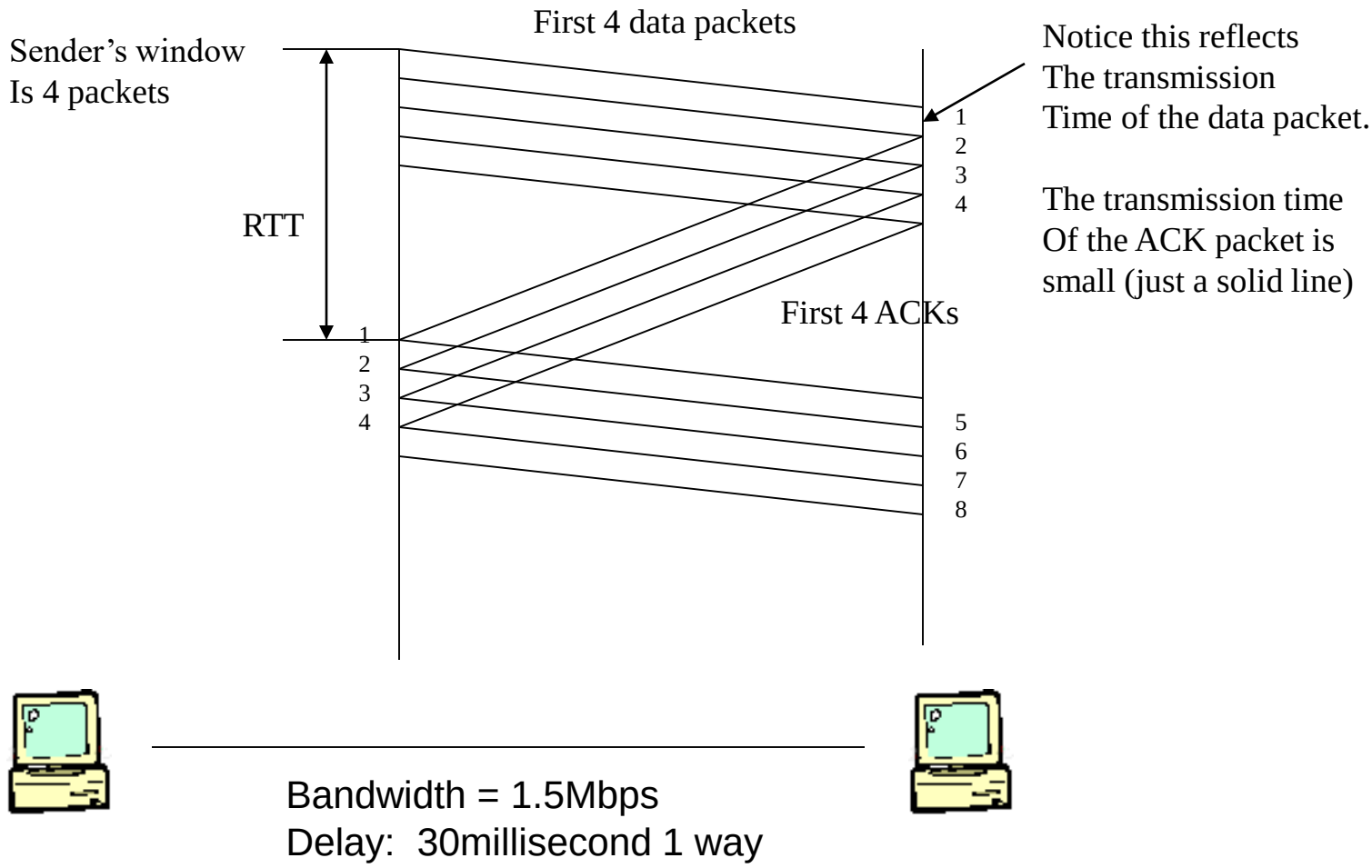
982120155.354974 192.168.1.100.1279 > 212.208.230.29.45912: S 143851464:143851464(0) win 16384 <mss 1460> (DF)
982120155.567089 212.208.230.29.45912 > 192.168.1.100.1279: S 2709967313:2709967313(0) ack 143851465 win 32120 <mss 1460> (DF)
982120155.567271 192.168.1.100.1279 > 212.208.230.29.45912: . ack 1 win **17520** (DF)
982120156.248161 212.208.230.29.45912 > 192.168.1.100.1279: P 1:1461(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.258408 212.208.230.29.45912 > 192.168.1.100.1279: P 1461:2921(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.266857 192.168.1.100.1279 > 212.208.230.29.45912: . ack 2921 win **14600** (DF)
982120156.487286 212.208.230.29.45912 > 192.168.1.100.1279: P 2921:4381(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.496933 212.208.230.29.45912 > 192.168.1.100.1279: P 4381:5841(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.507092 212.208.230.29.45912 > 192.168.1.100.1279: P 5841:7301(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.666886 192.168.1.100.1279 > 212.208.230.29.45912: . ack 7301 win **10220 (**DF)
982120156.936903 212.208.230.29.45912 > 192.168.1.100.1279: P 7301:8761(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.951005 212.208.230.29.45912 > 192.168.1.100.1279: P 8761:10221(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.963396 212.208.230.29.45912 > 192.168.1.100.1279: P 10221:11681(1460) ack 1 win 32120 (DF) [tos 0x10]
982120156.982527 212.208.230.29.45912 > 192.168.1.100.1279: P 11681:13141(1460) ack 1 win 32120 (DF) [tos 0x10]
982120157.066889 192.168.1.100.1279 > 212.208.230.29.45912: . ack 13141 win **4380** (DF)
982120157.299244 212.208.230.29.45912 > 192.168.1.100.1279: P 13141:14601(1460) ack 1 win 32120 (DF) [tos 0x10]
982120157.310435 212.208.230.29.45912 > 192.168.1.100.1279: P 14601:16061(1460) ack 1 win 32120 (DF) [tos 0x10]
982120157.321023 212.208.230.29.45912 > 192.168.1.100.1279: P 16061:17521(1460) ack 1 win 32120 (DF) [tos 0x10]
982120157.466886 192.168.1.100.1279 > 212.208.230.29.45912: . ack 17521 **win 0** (DF)
982120158.988267 212.208.230.29.45912 > 192.168.1.100.1279: . ack 1 win 32120 (DF) [tos 0x10]
982120158.988477 192.168.1.100.1279 > 212.208.230.29.45912: . ack 17521 win 0 (DF)
982120161.846314 212.208.230.29.45912 > 192.168.1.100.1279: . ack 1 win 32120 (DF) [tos 0x10]
982120161.846590 192.168.1.100.1279 > 212.208.230.29.45912: . ack 17521 win 0 (DF)
982120162.258374 192.168.1.100.1279 > 212.208.230.29.45912: . ack 17521 **win 10240** (DF)
982120162.266653 192.168.1.100.1279 > 212.208.230.29.45912: . ack 17521 **win 17520** (DF)

# Network Performance Characteristics

- Bandwidth-delay product
  - The product of the bw * delay measures the volume of data that can be in the network.
  - The capacity of a point-to-point link that directly connects two hosts is 1.5Mbps and the round trip time is 60ms. How much data can fit in the 'pipe' ?



•There's really two meanings of Bandwidth-delay product:
  - •First, a path property
  - •Second, a windowed protocol property




•Path level: BW*Delay = 1.5EXP6 * .03/ (8 * 1500) = 3.75 packets
•Windowed protocol level: If the max window is set to 4 packets, the link will only be 50% utilized (if we are the only sender)
    •So BW*Delay is used to define how big the window should be
        •BW*RTT = just under 8 packets. If the rx advertised window is set for 8 packets, the link will be 100% utilized (i.e., always busy!!!!).