CS 455/555
Intro to Networks and Communications

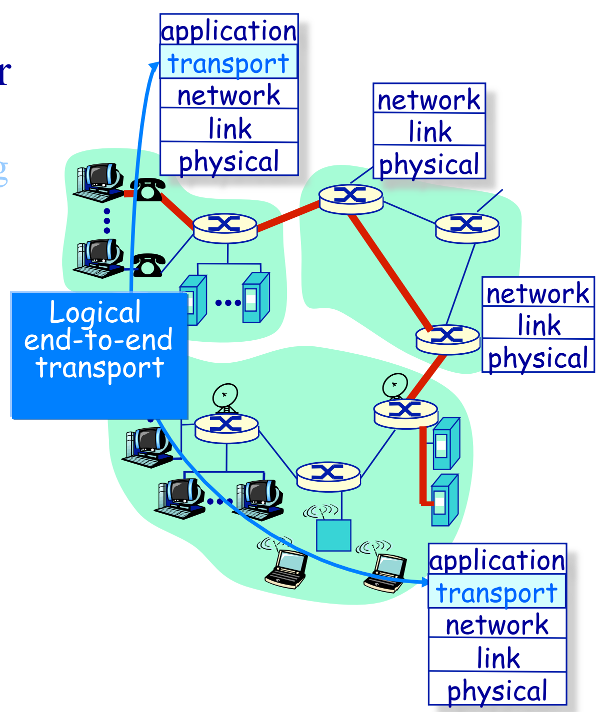
The Transport Layer
Reliable data delivery & flow control in TCP

Dr. Michele Weigle
Department of Computer Science
Old Dominion University
mweigle@cs.odu.edu

http://www.cs.odu.edu/~mweigle/CS455-S13

Transport Layer Protocols & Services
Outline

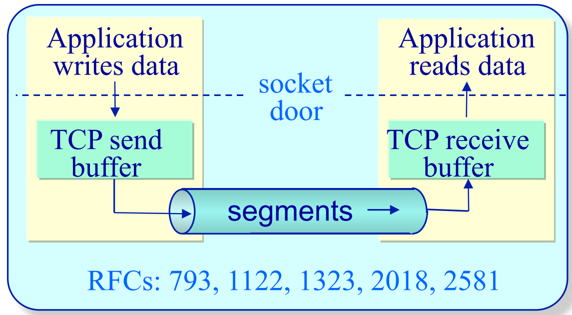
- Fundamental transport layer services
  - » Multiplexing/Demultiplexing
  - » Error detection
  - » Reliable data delivery
  - » Pipelining
  - » Flow control
  - » Congestion control
- Internet transport protocols
  - » UDP
  - » TCP



TCP Overview

TCP is…

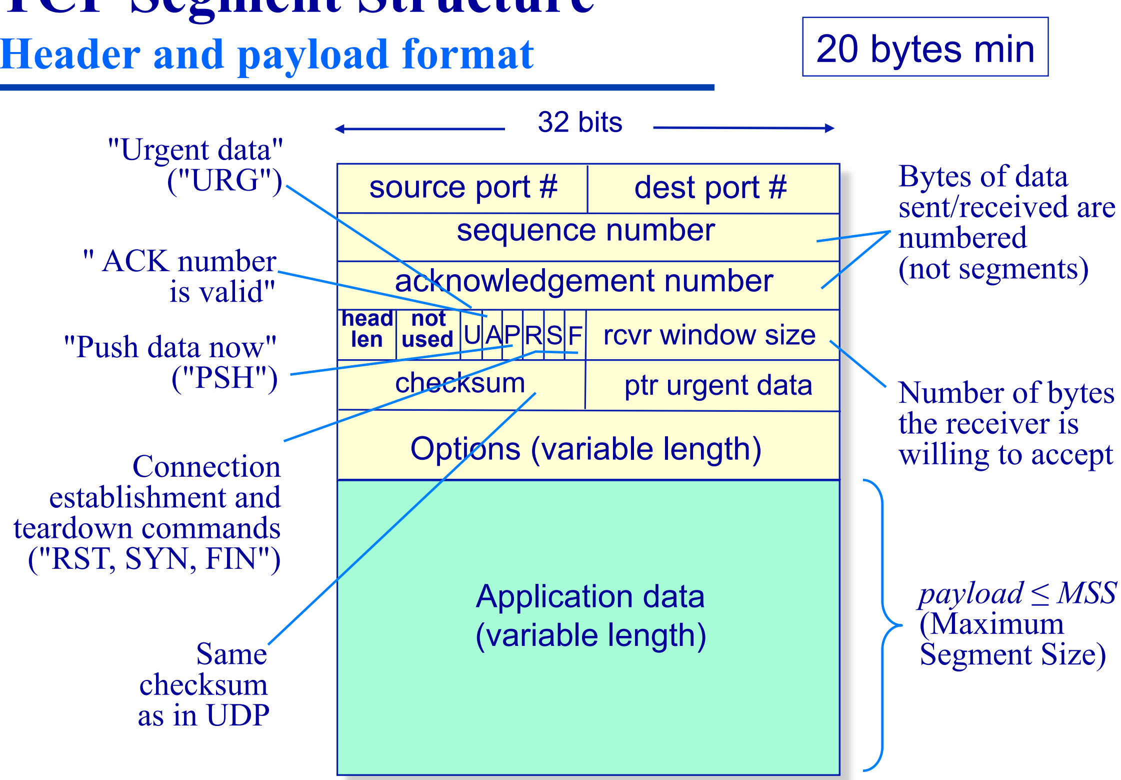
- Point-to-point, full-duplex
  - » Bi-directional data flow within a connection
- Reliable, in-order *byte stream*
  - » No "message boundaries"
- Connection-oriented
  - » Handshaking initializes sender and receiver state before data exchange
- Pipelined
  - » Congestion and flow control determine window size
  - » Each endpoint has *two* buffers: a send and receive buffer



- Congestion controlled
  - » Internet would cease to function without this!
- Flow controlled
  - » Sender and receiver have synchronized windows to ensure receiver is not overwhelmed

TCP Segment Structure

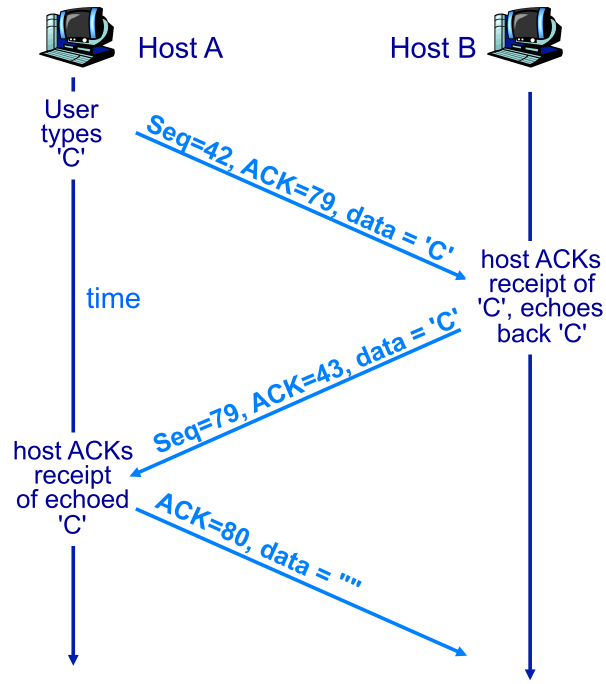
Header and payload format



TCP Sequence Numbers and ACKs

Telnet example

- Sequence numbers:
  - » Byte stream "index" of the first byte in the segment's payload
- ACKs:
  - » Sequence number of next byte expected from the other side
  - » ACKs are cumulative
- How does receiver handle out-of-order segments?
  - » TCP spec doesn't say, it's up to the implementor



TCP Intro

Setting the ACK timer

- How large should the ACK timeout value be?
  - » Too short: Premature timeouts result in unnecessary retransmissions
  - » Too long: Slow reaction to loss results in poor performance because the sender's windows stops advancing
- Timer should be longer than the RTT, but how do we estimate RTT?
  - » Measure the time from segment transmission until receipt of ACK ("`SampleRTT`")
    - Ignore retransmissions - Karn's algorithm
    - Measure only one segment's RTT at a time
  - » `SampleRTT` will vary, so we compute an average RTT based on several recent RTT samples

TCP Intro

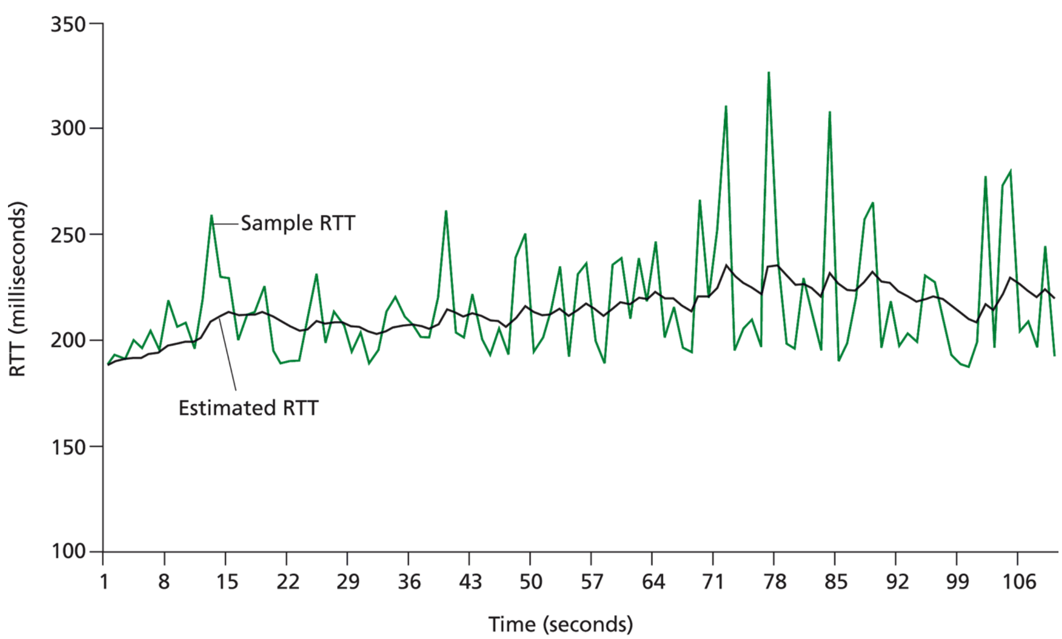
Estimating round-trip-time

```
EstimatedRTT = (1-α)*EstimatedRTT + α *SampleRTT

      Timeout = EstimatedRTT + 4*Deviation

    Deviation = (1-β)*Deviation +
                β * |SampleRTT-EstimatedRTT|
```

- The estimated RTT is an exponential weighted moving average (EWMA)
  - » Computes a "smooth" average
  - » Influence of a given sample decreases exponentially fast
  - » Typical value of α is 0.125
  - » Typical value of β is 0.25
- Timeout is `EstimatedRTT` plus "safety margin"
- Large variation in `EstimatedRTT` results in a larger safety margin

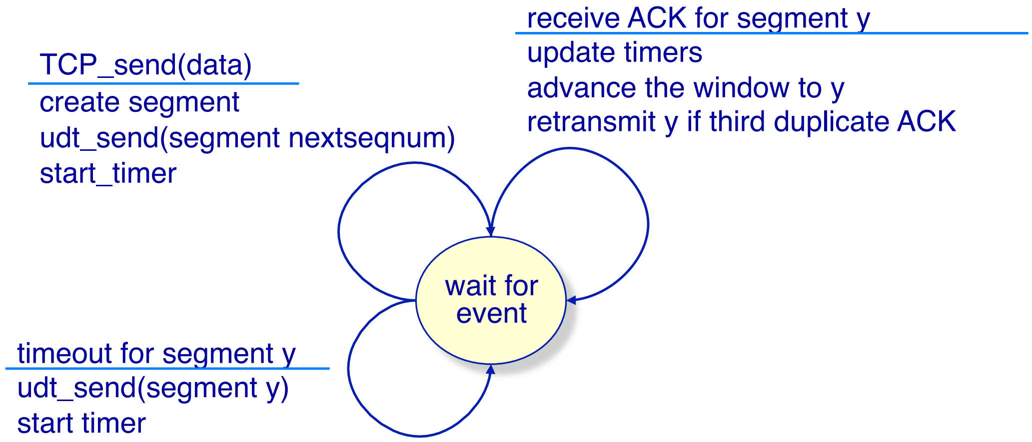
TCP Intro

Estimating round-trip-time



Reliable Data Transfer in TCP

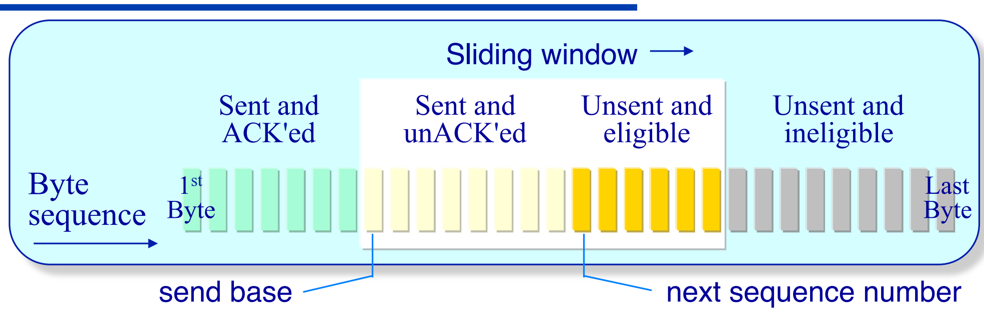
Sender's state machine



- TCP retransmits segments if:
  - » An expected ACK times out
  - » 3 duplicate ACKs for a segment are received

Reliable Data Transfer in TCP

Simplified sender's state machine



sendbase = initial_sequence number
nextseqnum = initial_sequence number

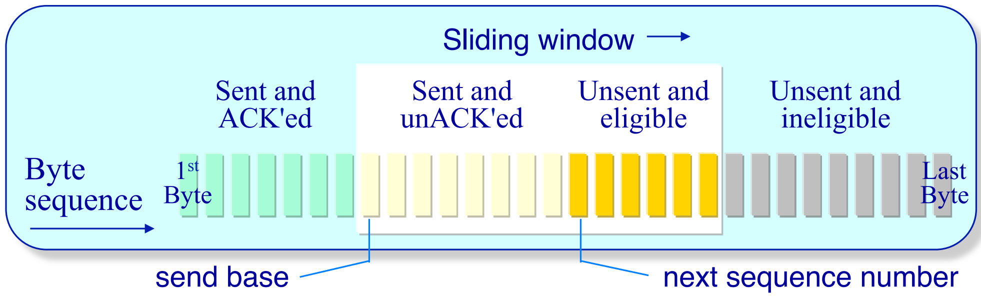
loop (forever) {
switch(event)
event: data received from application above
event: timer timeout for segment with sequence number y
event: ACK received with value y
}

create TCP segment with sequence number nextseqnum
start timer for segment with nextseqnum
pass segment to IP
nextseqnum = nextseqnum + length(data)

retransmit segment with sequence number y ("segment y")
compute new timeout interval for segment y
restart timer for segment y

Reliable Data Transfer in TCP

Simplified sender's state machine




```
sendbase      = initial_sequ
nextseqnum  = initial_sequ

loop (forever) {
  switch(event)
    event: data received fro
              above
    event: timer timeout for
               sequence numbe
    event: ACK received wi
}
```

```
if (y > sendbase) {     /* Cumulative ACK of all data up to y */
   cancel timers for segments with sequence numbers ≤ y
   sendbase := y
 }
else if ( y == sendbase) {                    /* A duplicate ACK */
   increment number of duplicate ACKs received
   if (number of duplicate ACKS received == 3)
    {                                                /* Fast retransmit */
      resend segment with sequence number sendbase
      restart timer for segment sendbase
    }
 }
```

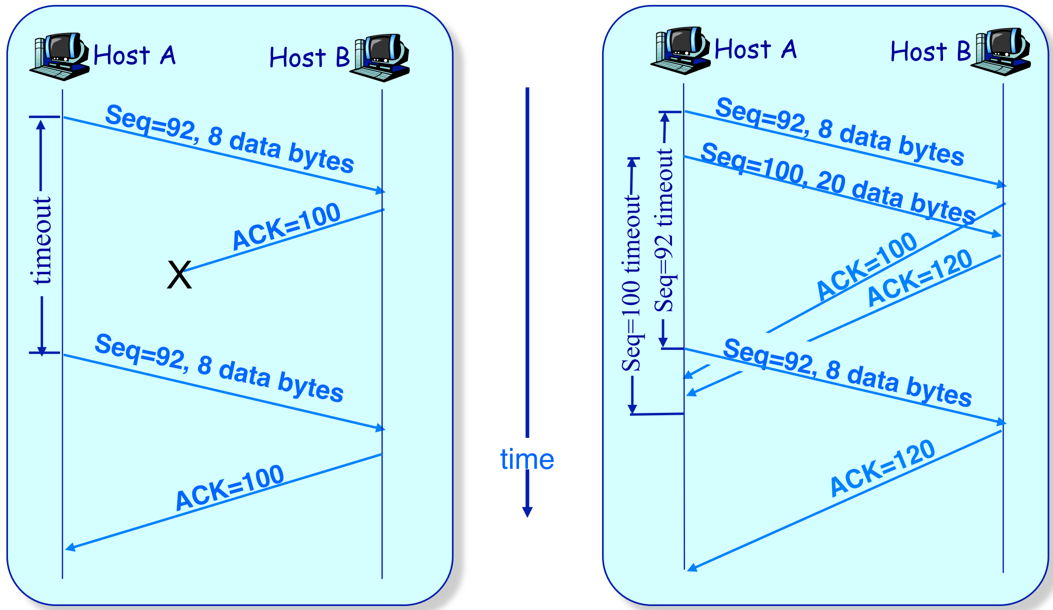

Reliable Data Transfer in TCP

ACK generation rules [RFC 1122, RFC 2581]

Event	TCP Receiver action
In-order segment arrival, no gaps, all previous data already ACKed	**Delayed ACK.** Wait 200 *ms* (up to 500 *ms* allowed) for next segment. If no segment received, send ACK
In-order segment arrival, no gaps, one delayed ACK pending	Immediately send single cumulative ACK
Out-of-order segment arrival (higher than expected sequence number) — Gap detected	Send duplicate ACK, indicating sequence number of next expected byte
Arrival of segment that partially or completely fills gap	Immediate ACK if segment starts at lower end of gap

Reliable Data Transfer in TCP

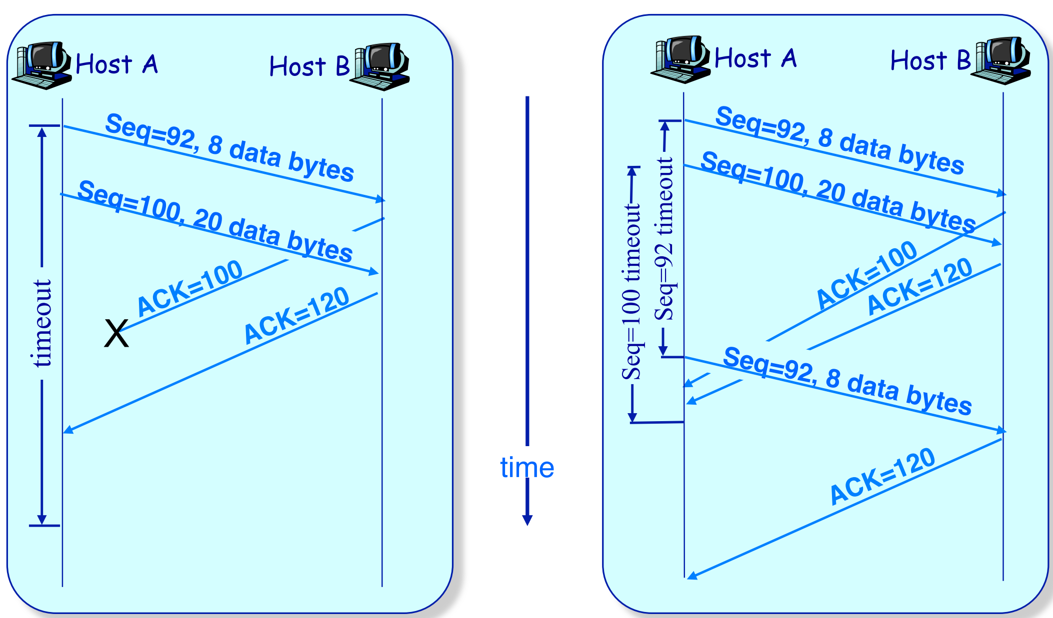
Retransmission examples



- Lost ACK scenario
- Premature timeout

Reliable Data Transfer in TCP

Retransmission examples



- Cumulative ACKs potentially avoid retransmissions
- Premature timeout

Reliable Data Transfer in TCP
Timeout Interval

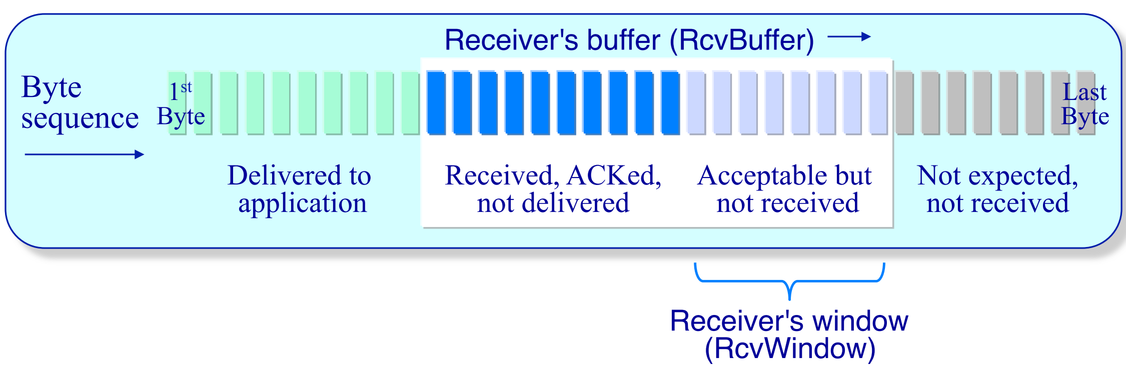
Whenever a timeout occurs

» TCP retransmits the non-yet-acknowledged segment with the smallest sequence number

» Sets the timeout interval to twice the previous value

- timeout interval grows *exponentially* after every consecutive retransmission

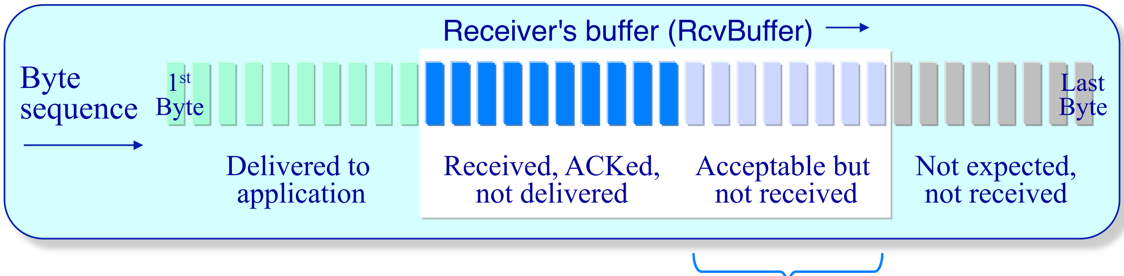
TCP Flow Control
Window control



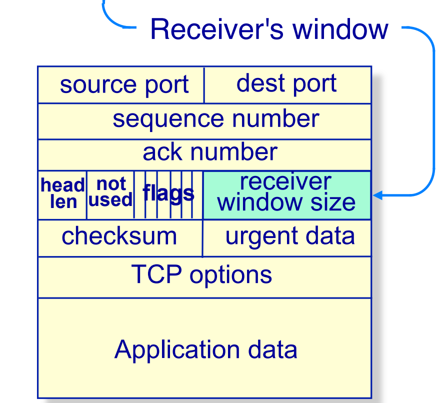
- Flow control is the problem of ensuring the receiver is not overwhelmed
  - » The receiver can become overwhelmed if the application reads too slow or the sender transmits too fast
- The receiver's window represents its remaining buffer capacity
- The window advances as the application reads received data

TCP Flow Control

Window control

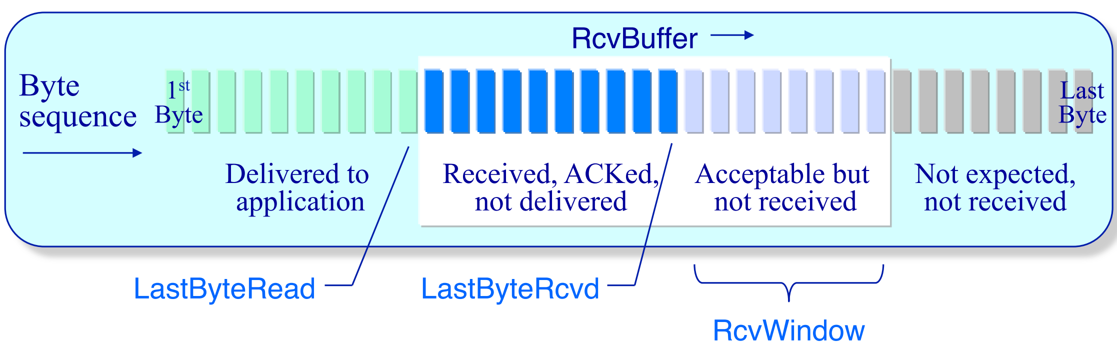


- The receiver explicitly informs the sender of the amount of free buffer space in `RcvBuffer`
  - » `RcvWindow` field in TCP segment
- The sender keeps the amount of transmitted, unACKed data less than most recently received `RcvWindow`



TCP Flow Control

Window control



- The goal is to ensure:

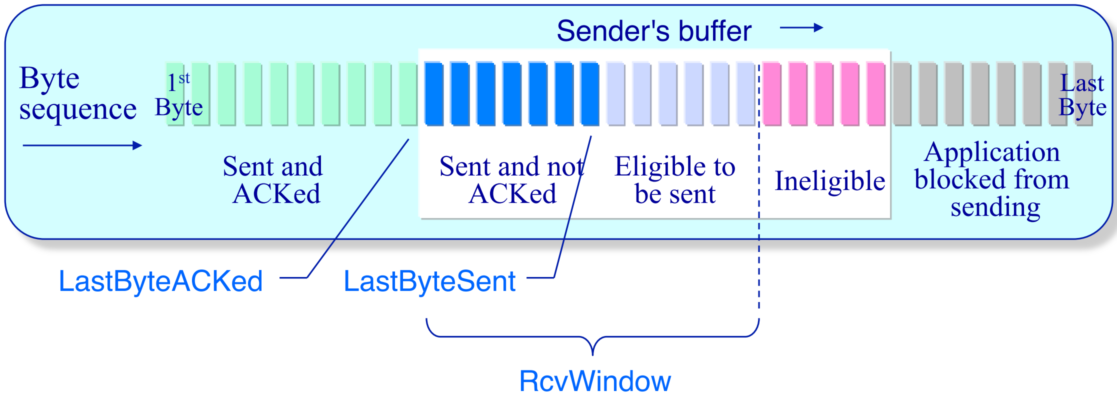
  `LastByteRcvd - LastByteRead ≤ RcvBuffer`

- Sender is sent:

  `RcvWindow = RcvBuffer - (LastByteRcvd-LastByteRead)`

TCP Flow Control

Window control

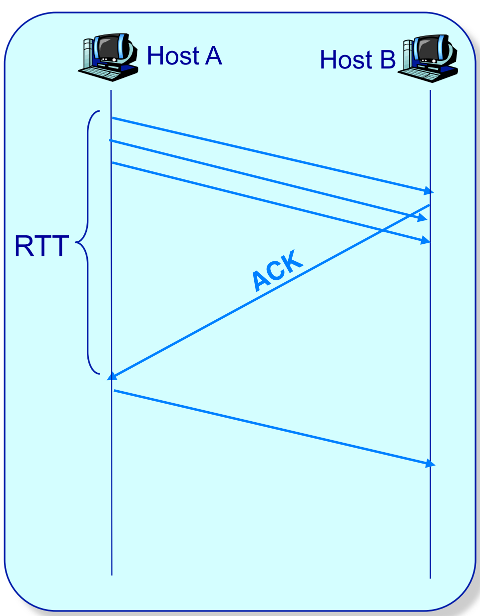


- The sender ensures:

  `LastByteSent - LastByteACKed ≤ RcvWindow`

TCP Flow Control

Example: How big should receiver window be?



TCP's bandwidth-delay product (BDP):
bottleneck bandwidth x RTT

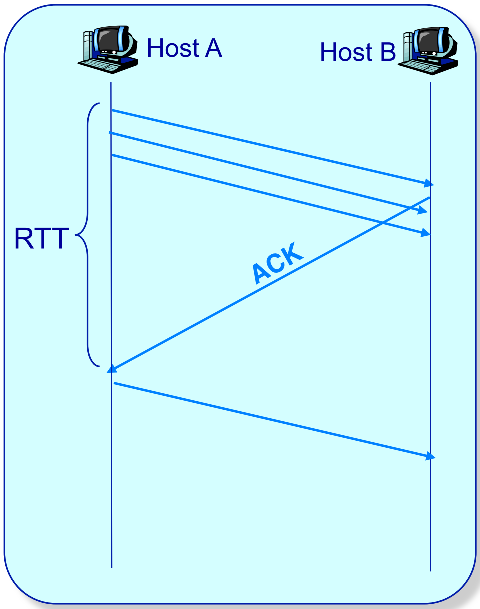
10 Mbps bottleneck bandwidth
RTT = 20 ms

BDP = 10 Mbps x 20 ms = **25,000 B**

So, receiver's window size (W) should be at least 25,000 bytes.

TCP Flow Control

Example: What's max TCP throughput?



$$R \times RTT = W$$

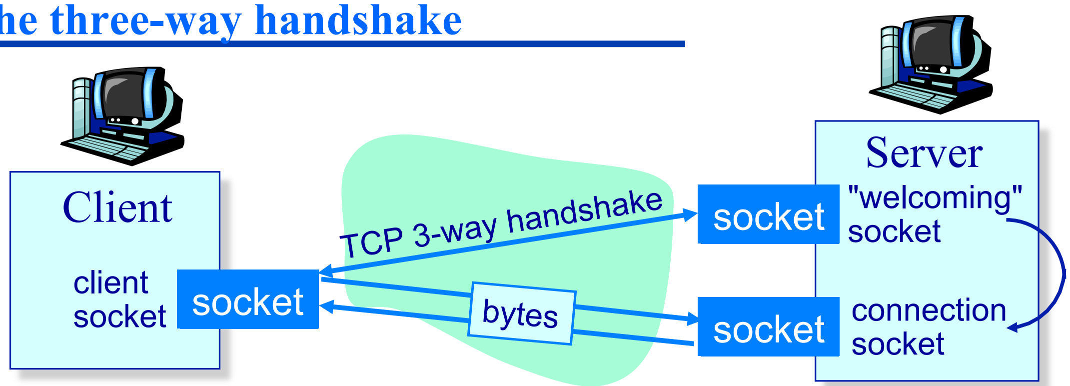
2 KB (2048 B) receive window
20 ms RTT

R = W / RTT
R bps = 2048 B / 20 ms
= 16,384 b / 0.02 seconds
= **819,200 bps**

So, max TCP throughput is
819.2 kbps

TCP Connection Management

The three-way handshake

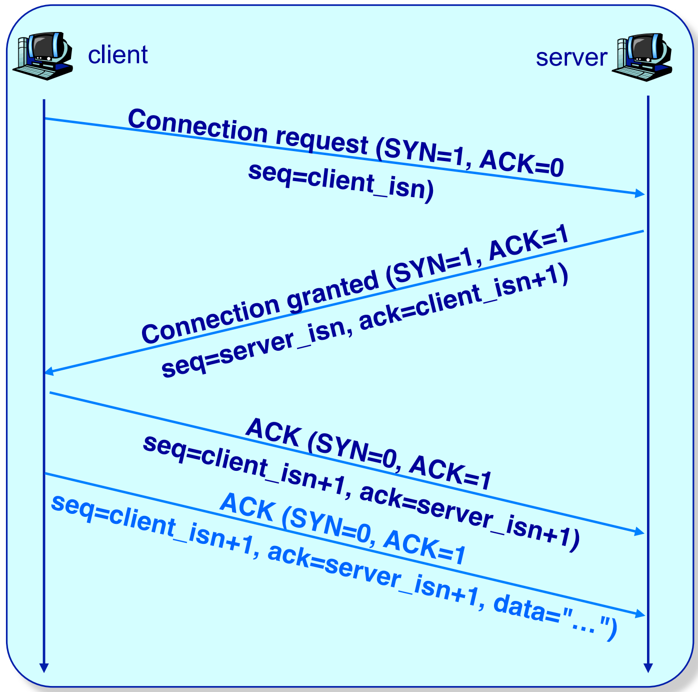


- TCP endpoints establish a "connection" before exchanging data segments
  - » *client:* connection initiator
    ```
    mySocket = new Socket (hostname, port);
    ```
  - » *server:* contacted by client
    ```
    clientSocket = welcomeSocket.accept();
    ```

TCP Connection Management

The three-way handshake

- Client sends SYN segment to server
  - » The SYN specifies the client's initial sequence number
- Server receives SYN, replies with SYN+ACK segment
  - » ACKs received SYN
  - » Allocates buffers
  - » Specifies server's initial sequence number
- Third segment may be an ACK only or an ACK+data



Reliable Data Transfer in TCP

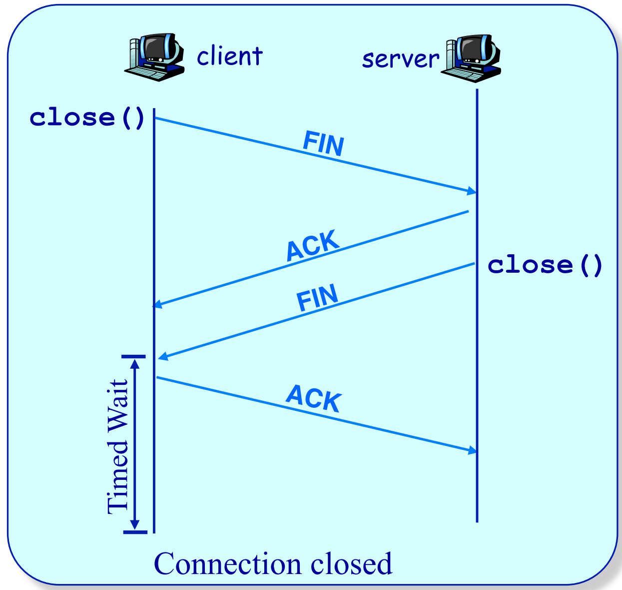
Timeout Interval

```
EstimatedRTT = (1-α)*EstimatedRTT + α *SampleRTT

Timeout = EstimatedRTT + 4*Deviation
```

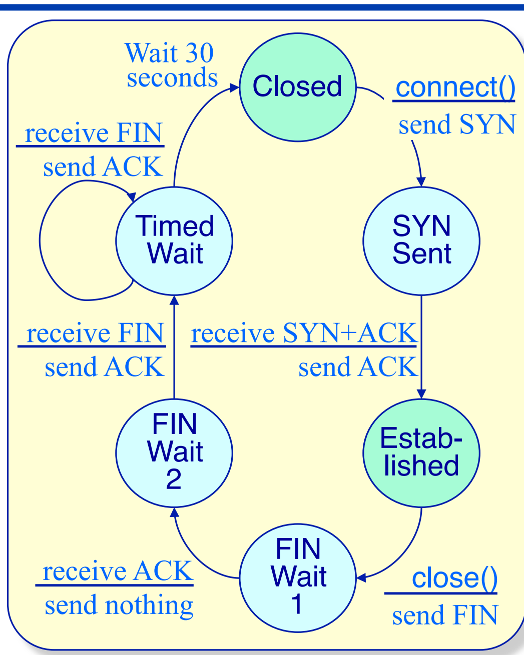
- What happens if the SYN or SYN/ACK is lost?
  - » Initial value of `Timeout` is 3 seconds
- What happens if the retransmission of the SYN is lost?
  - » Exponential increase

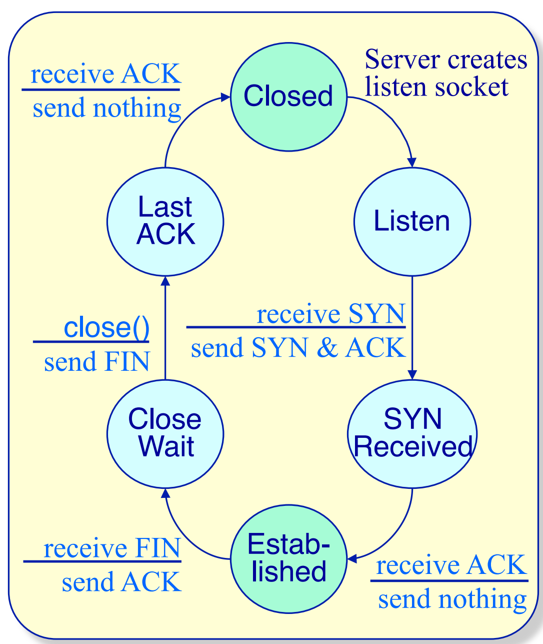
TCP Connection Management
Closing a connection

- Client sends FIN segment to server
- Server receives FIN, replies with ACK
  - » Server closes connection, sends FIN
- Client receives FIN, replies with ACK
- Client enters "timed wait" state
  - » Client will ACK any received FIN



TCP Connection Management
Client/Server lifecycles





- TCP client lifecycle
- TCP server lifecycle

TCP Connection Management
SYN Flood Attack

- Setup: TCP Connection Setup
  - » TCP allocates and initializes connection variables and buffers in response to a received SYN
  - » If TCP never receives 3rd part of handshake, it will deallocate buffers (after a minute or more)
- Attack: SYN Flood
  - » Bad guy sends a large number of SYN segments, but never completes the handshake
  - » Once resources are exhausted, legitimate connections are refused
- Remedy: SYN Cookies
  - » Server does not allocate buffers on receipt of SYN
  - » Server sends SYN/ACK packet with special sequence number ("cookie")
  - » Client returns ACK with SYN/ACK seqno+1 in ackno field (like normal)
  - » Server verifies that this is valid and allocates buffers

TCP Connection Management
Port Scanning

nmap port-mapping tool sends SYN to a particular port on a host)

1. source receives a TCP SYN/ACK -- port is open
2. source receives a TCP RST segment -- SYN reached the host, but the port is closed (not blocked by firewall)
3. source receives nothing -- blocked by firewall

Wireshark Example

- See handout
- Wireshark
  - » network protocol analyzer
  - » http://www.wireshark.org/
- Capture examples
  - » http://wiki.wireshark.org/SampleCaptures/