Intro to LAN/WAN

Transport Layer

Transport Layer Topics

- ☞ Introduction (6.1)
- ☞ Elements of Transport Protocols (6.2)
- ☞ Internet Transport Protocols: TDP (6.5) ←
- ☞ Internet Transport Protocols: UDP (6.4)

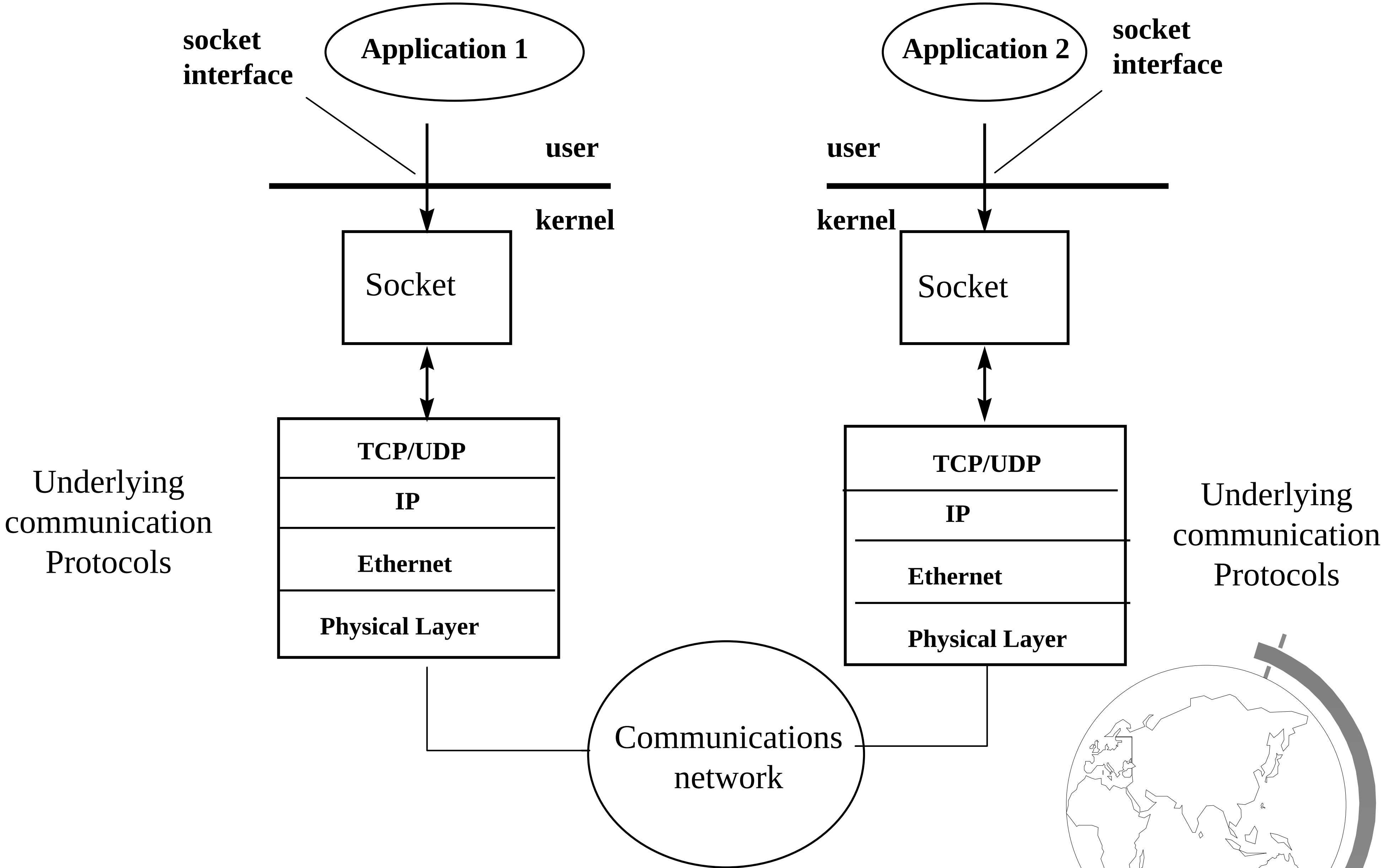

socket
interface
Application 1
user
kernel
Socket
TCP/UDP
IP
Ethernet
Physical Layer
Underlying
communication
Protocols
Application 2
socket
interface
user
kernel
Socket
TCP/UDP
IP
Ethernet
Physical Layer
Underlying
communication
Protocols
Communications
network

Figure 2.16

TCP

- Connection-oriented
- Reliable, end-to-end byte-stream
  - – message boundaries not preserved
- Adapt to a variety of underlying networks
- Robust in the face of failures (IP: no guarantees)
- Break data into *segments*
- Sliding window

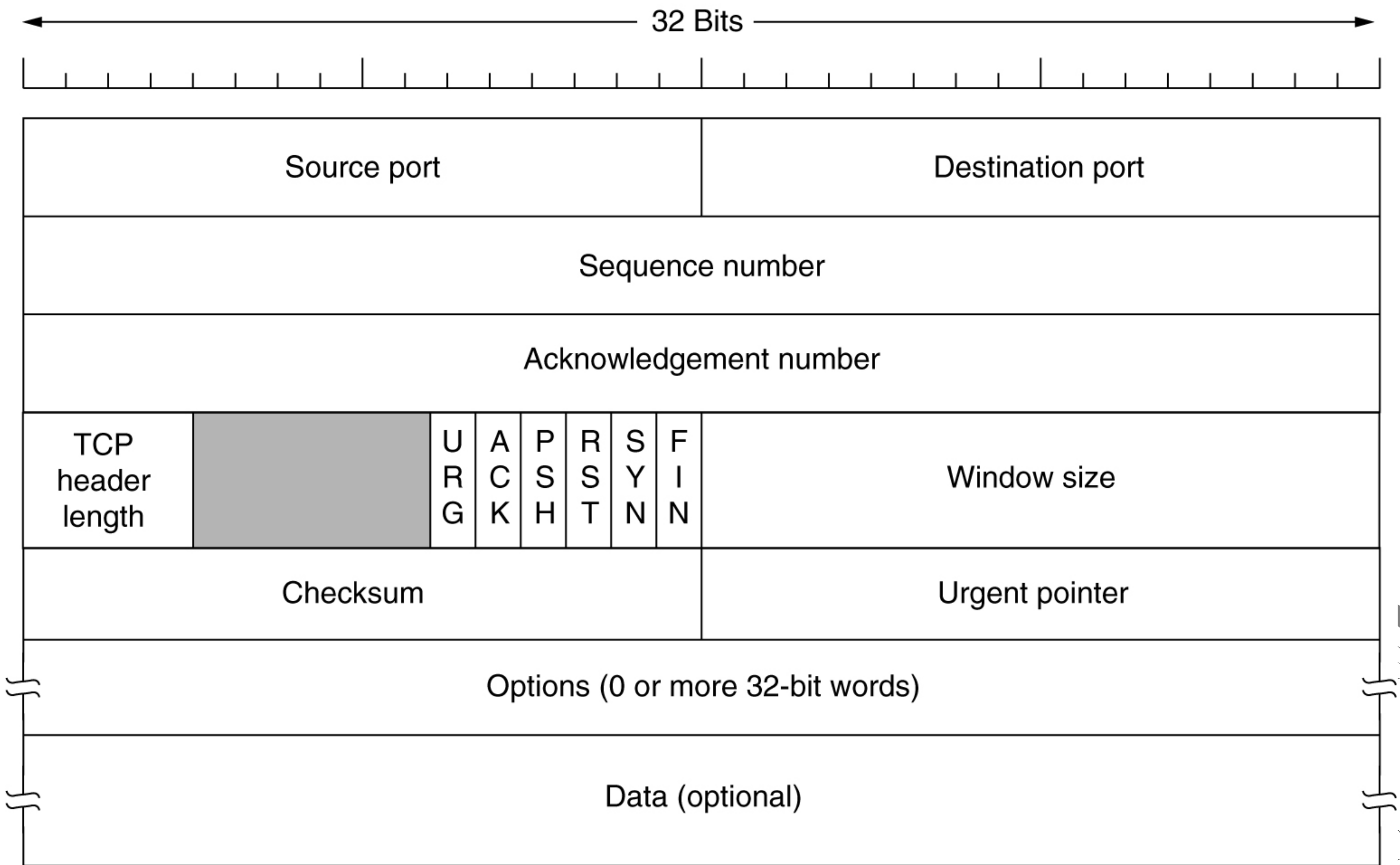
TCP Service Model

- Sender and receiver create end points (sockets)
- One socket may be used for multiple connections
- Well-known services at well known port numbers
  - FTP (port 21)
  - telnet (port 23), etc
- Inetd deamon (UNIX)
  - listens for all connects (FTP, telnet, etc)
  - Forks off new process to handle new connections (at designated ports)
  - Other daemons (FTP, telnet, etc) only active when work to do
  - Inetd learns about what ports to use from config. file

TCP Service Model

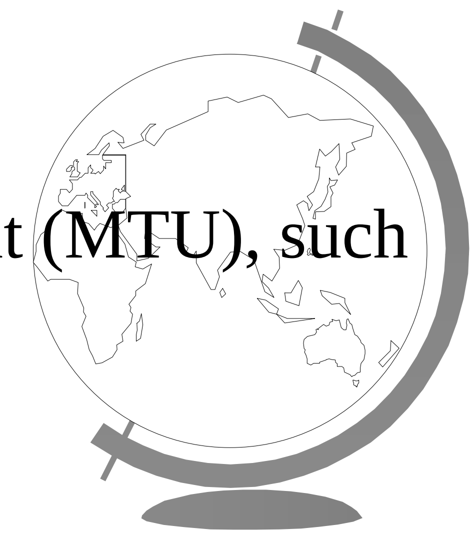
- All TCP connections are
  - full-duplex (can send both ways)
  - Point-to-point: each connection has two end points
  - Does not support multicast or broadcast (need either different protocol or improvement)
  - Multicast (not TCP) protocols used for multicast

TCP Segment Header



TCP Protocol

- TCP entitites (sender, receiver) exchange segments
- TCP segment:
  - 20-byte header (plus optional part)
  - Followed by zero or more data bytes
  - TCP software decides size of segments (fragmentation)
  - Segments can be split up or aggregated
    - must fit into 65,515-byte IP payload
    - Also networks have Maximum Transfer Unit (MTU), such as 1500-byte limit on Ethernet

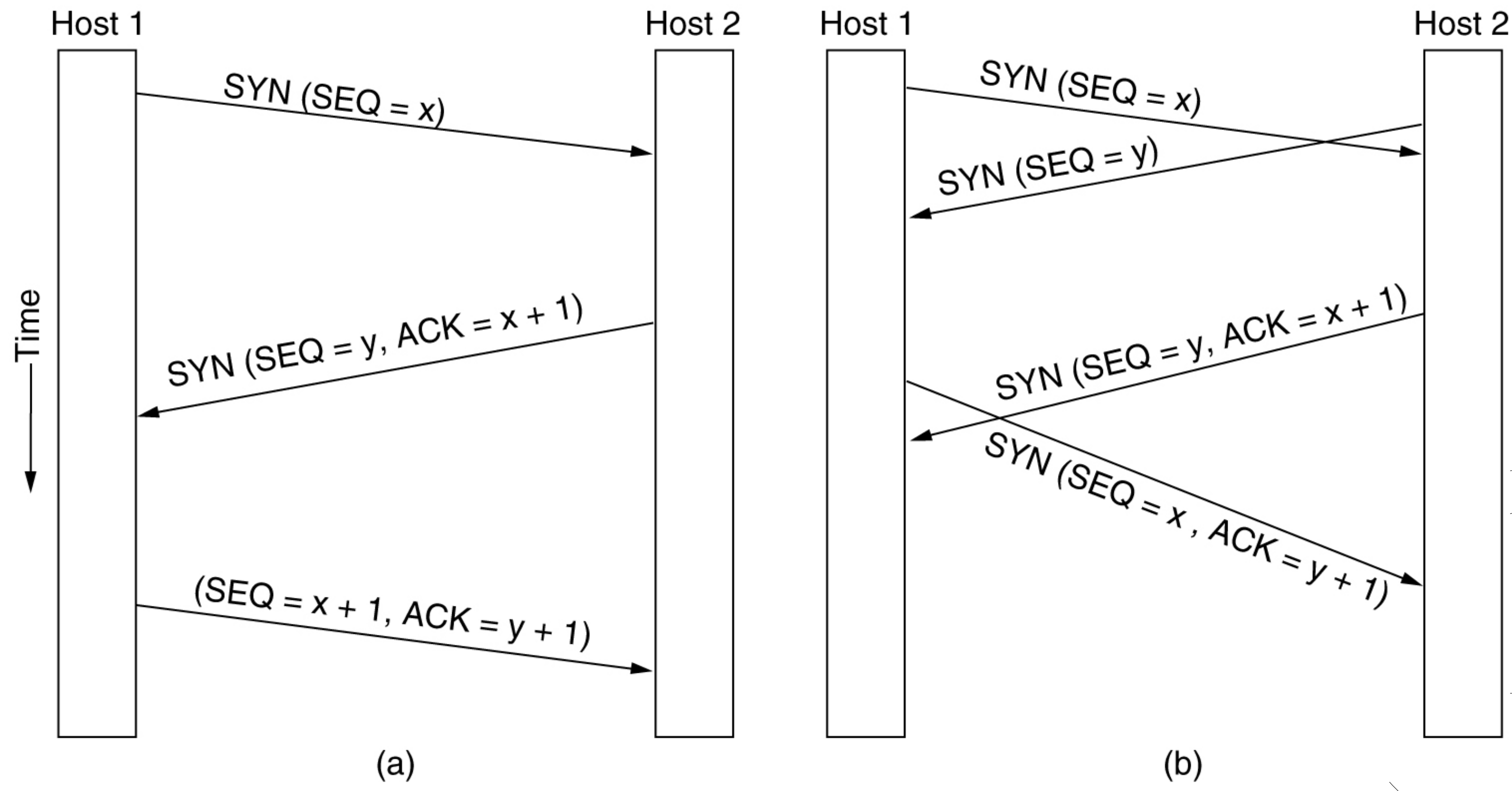


TCP Protocol

- Basic protocol uses sliding window
- Sender starts timer when it sends data
- Receiver can either piggyback ACK or alone
- Sender resends if its timer goes off
- Subtle issues TCP must deal with:
  - Segments can be delayed or arrive out of order (different routes?)
  - In fact, retransmissions may be different byte range from original

TCP Connection Establishment

Uses three-way handshake (similar to that previously discussed)

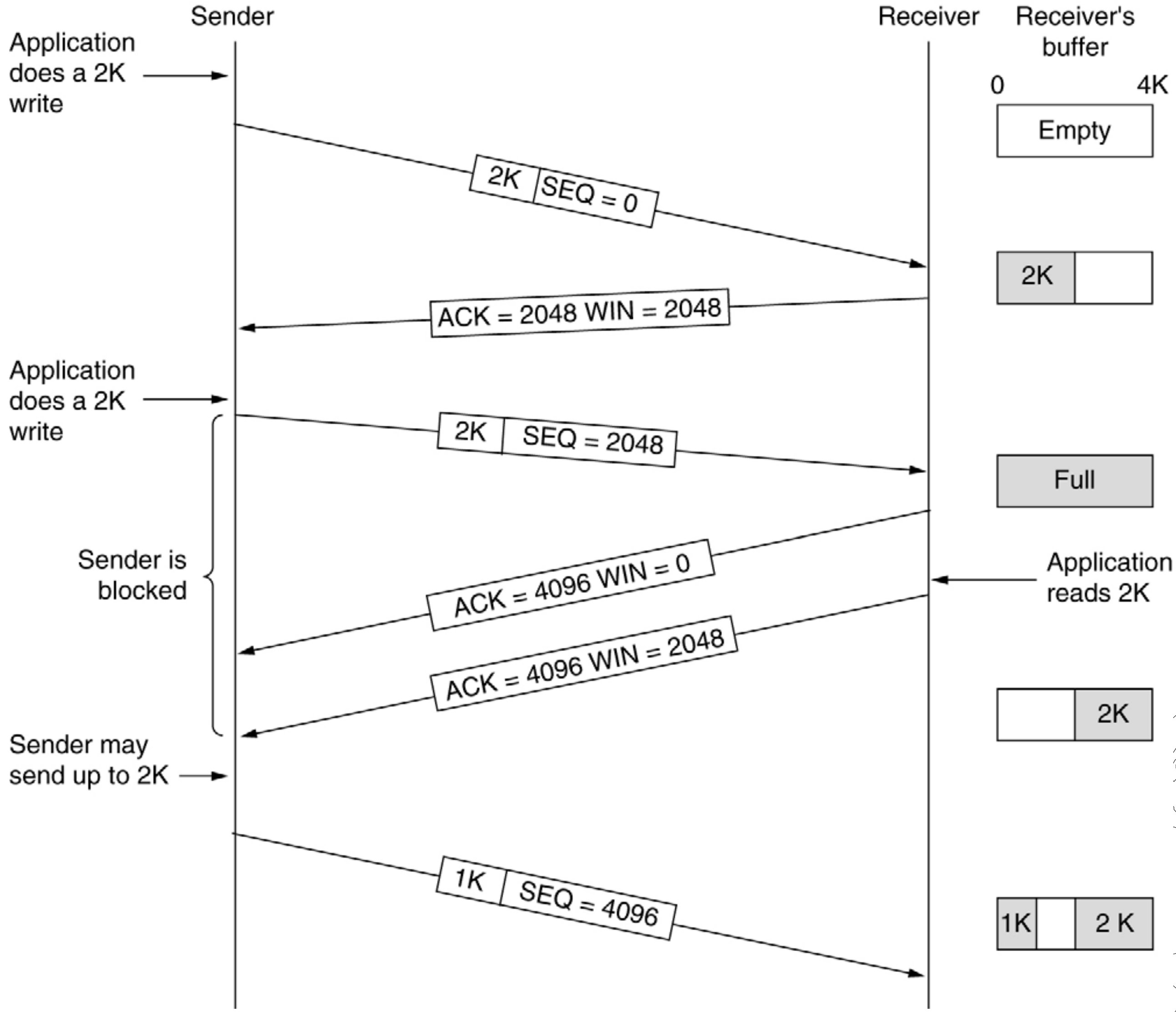


TCP Connection Release

- Although connections are full-duplex, think simplex for connection release
- Either end (sender, receiver) can send segment with *FIN* bit set
- *FIN* acknowledged, that direction is done!!
- Data may continue to flow in other direction
- Process repeated in other direction to close
- Connection closes when both ends close
- Usually two *FIN-ACK* pairs (4 pkts) to close
- May piggyback to reduce packets sent
- Two-army problem: if no ACK within set time, close!!

TCP Transmission Policy

- TCP receiver advertises its window size (remaining buffer space)



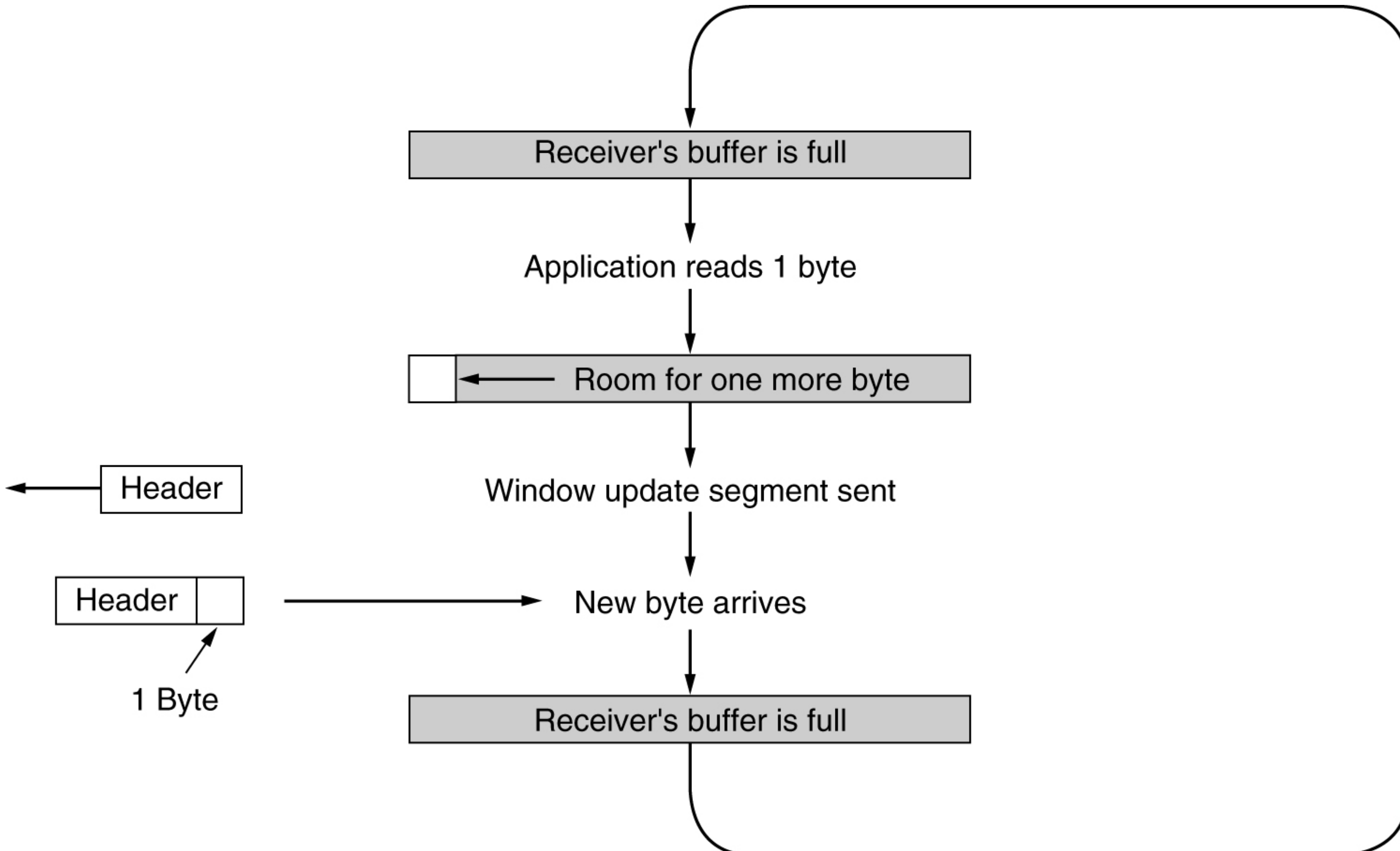
Window management in TCP.

TCP Transmission Policy

- Do not have to send immediately
  - avoid many small packets
- Some TCP implementations delay pkts, ACKs for 500 msec to see if it can get more "stuff" to send
- *Nagle's Algorithm*
  - only 1 outstanding byte at a time
  - fill up, then send
  - time delay, then send
  - bad for some apps (X - with mouse movements)

Silly Window Syndrome

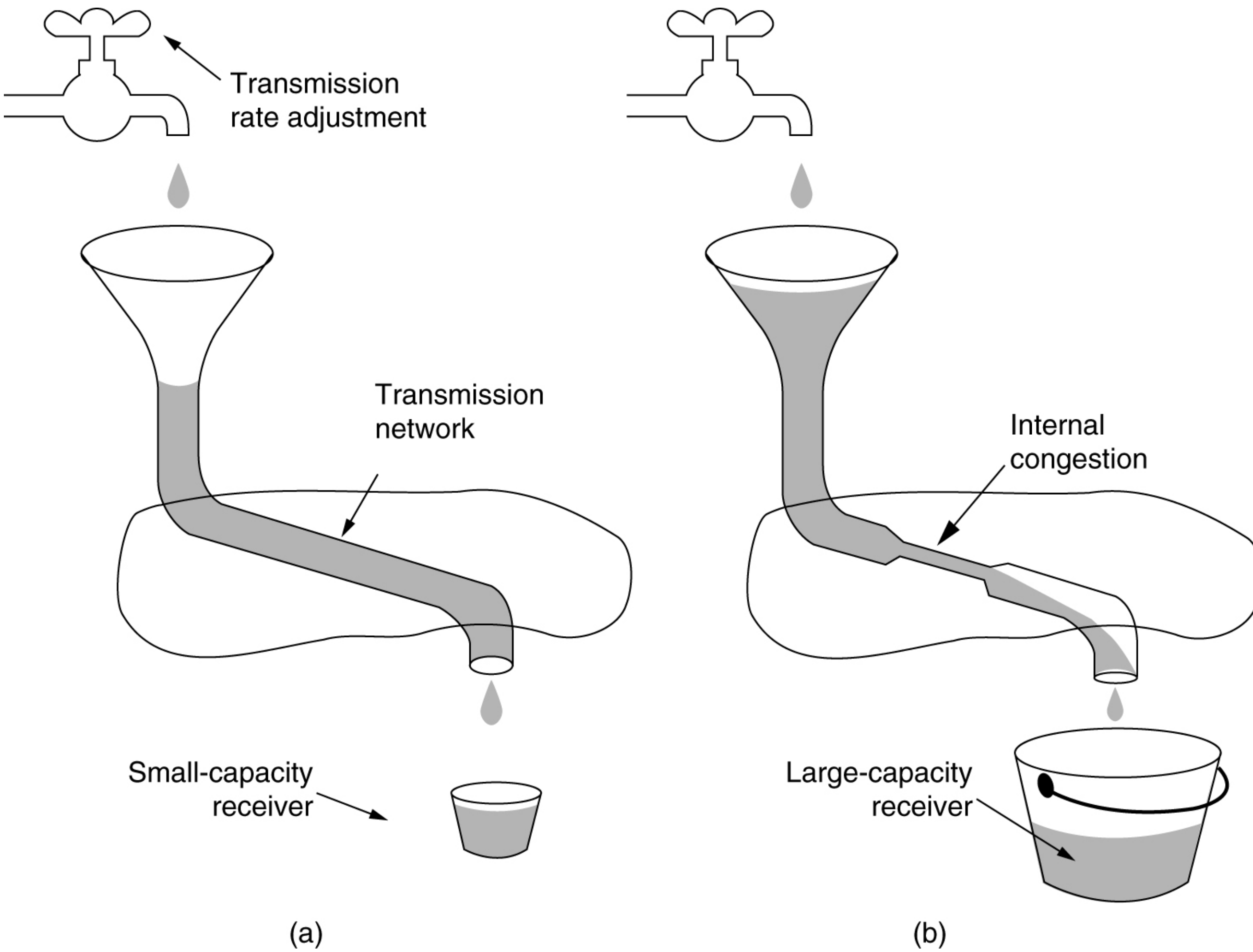
- Sender sends in large chunks
- Application reads 1 byte at a time



- Fix: receiver only advertises send window when 1/2 full

TCP Congestion Control

- ☞ Even if sender and receiver agree, still problems



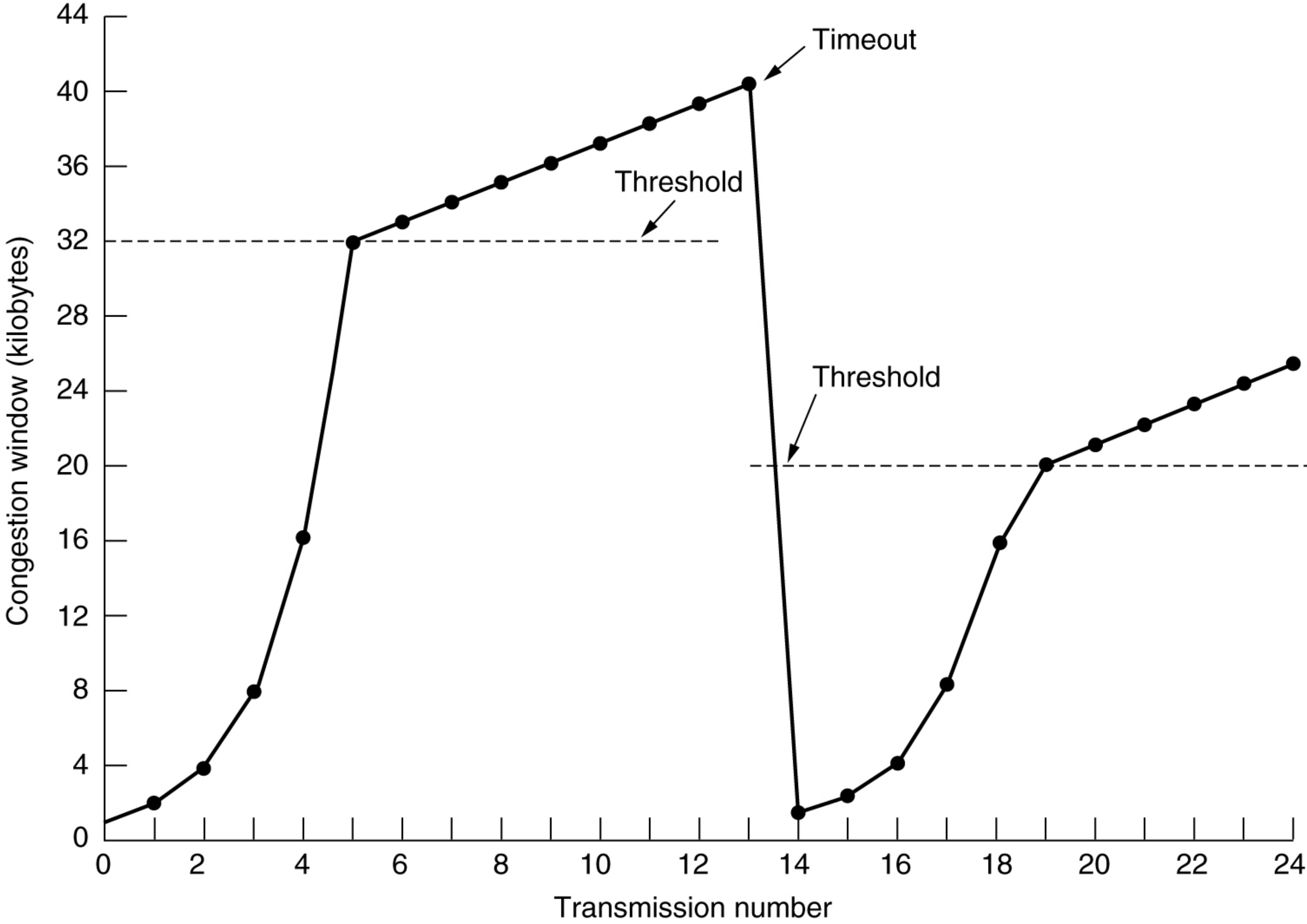
TCP Congestion Control

- Sender tracks two windows
- "Receiver buffer" via receiver's window (via advertisements)
- "Network buffer" via congestion window
- "Effective buffer" is minimum of receiver and network
- Ex:
  - Receiver says "8k", Network says "4k" then 4k
  - Receiver says "8k", Network says "32k" then 8k

Avoiding Congestion

- Network buffer
  - starts at 1 segment
  - increases exponentially (doubles)
  - until timeout or receiver's window reached
  - or threshold (initially 64K), then increases linearly
  - *slow start* (required by TCP, Jacobson 1988)
- Internet congestion includes threshold
  - linear past threshold (called *congestion avoidance*)
  - when timeout, reduce threshold to half of *current window* and restart slow start
    - can go up

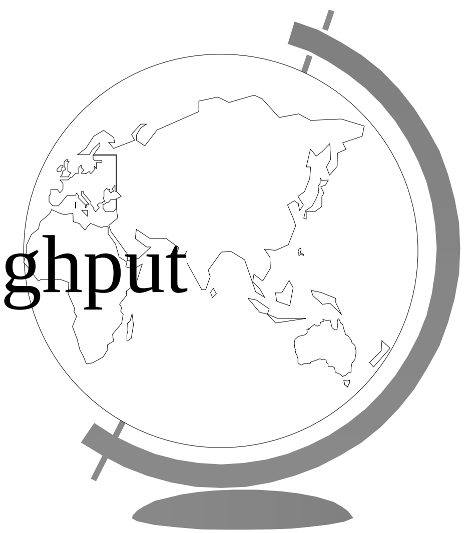
TCP Congestion Control



An example of the Internet congestion algorithm.

TCP Congestion Control Summary

- When below threshold, grow exponentially
  - slow start
- When above threshold, grow linearly
  - congestion avoidance
- When timeout, set threshold to 1/2 current window and set window to 1
- How do you select timer values?
  - Important, since timeouts restrict throughput
  - Timer management

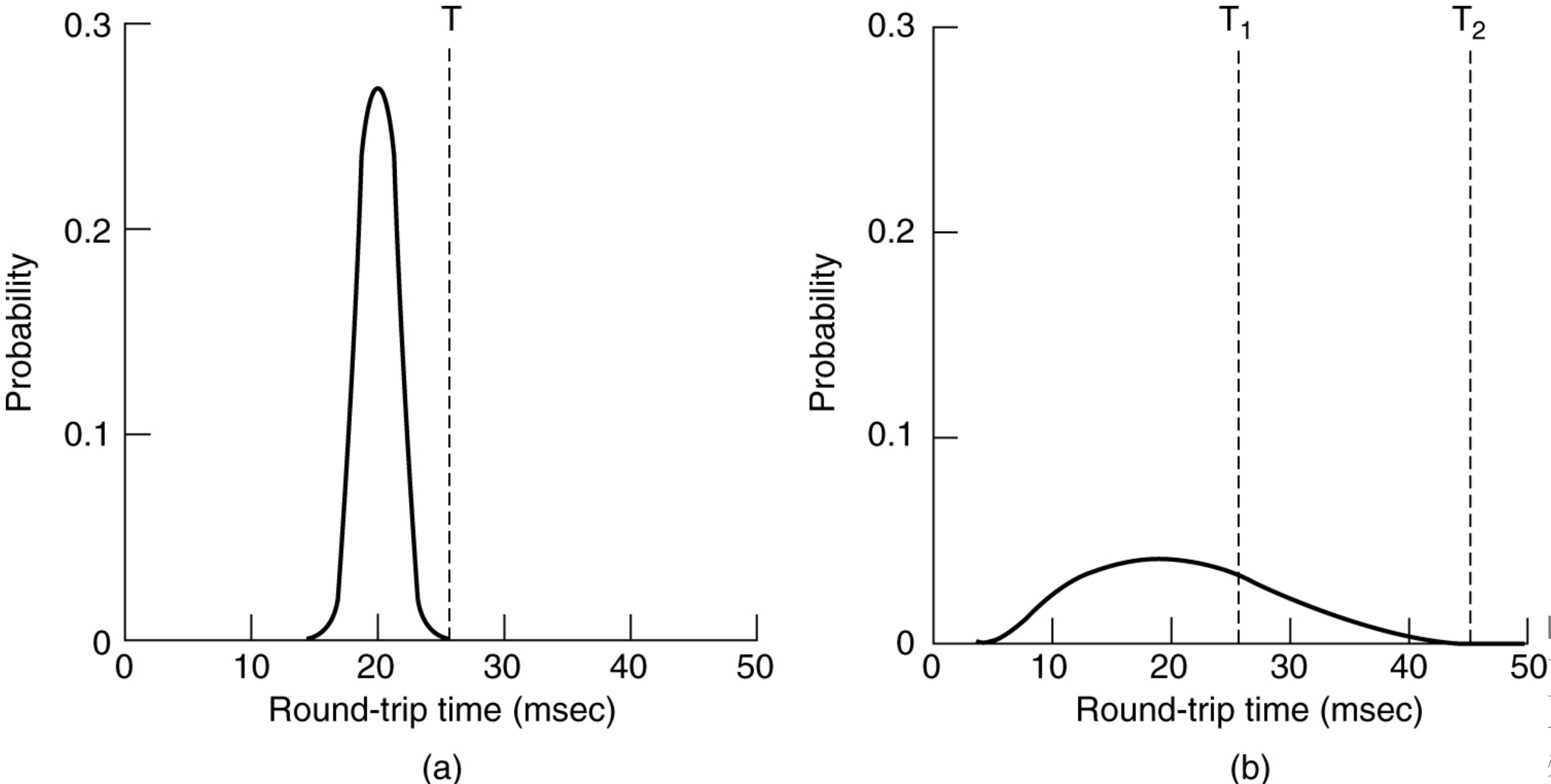


Timer Management

- Retransmission timer: most important in TCP
- Optimal timer setting?
  - Too short, too many retransmissions, packets clog up network
  - Too long, performance suffers
  - Need dynamic algorithm since conditions can change
- Want to set timeout to minimal value where segment is known to be lost (quickly resend)
- Generally set timer as a function of Roundtrip (RTT)
- So, need estimate of round-trip time (RTT)
  - how to get it?
- Why can’t you just measure RTT once and fix timeout timer?

Timer Management

- Difficult when much variance



- RTT = αRTT + (1-α)M   (α = 7/8, M ack time)
- α is smoothing function determining contribution of old RTT
- + add variance, don't update on retransmits