Data Communication and Networks (Fall 2004)

Lecture 8

The Transport Layer

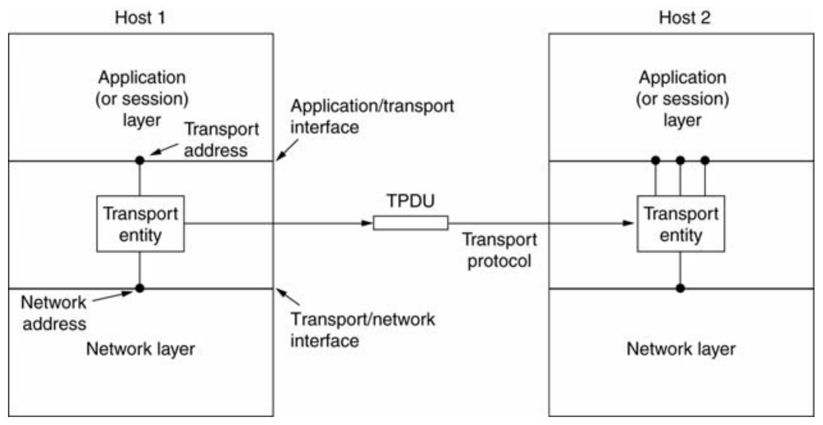
Dr. Sahar M. Ghanem

Outline

The Transport Layer

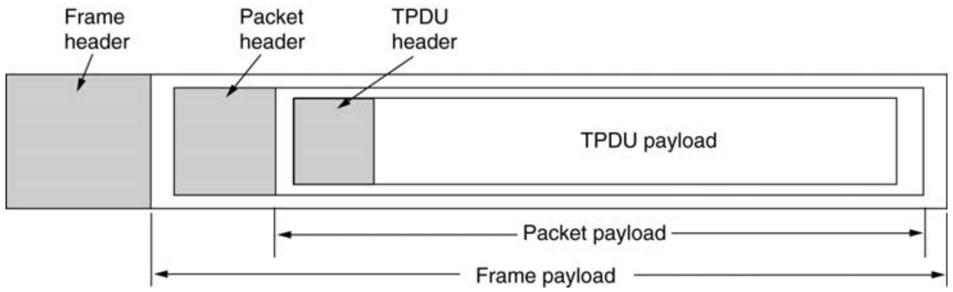
- The *transport layer* receives services from the *network layer* and provides services to the *application layer*
- The transport entity can be a library procedure, a user process, or part of the kernel
- The Internet has two protocols at the transport layer UDP and TCP
  - The data link layer provides node-to-node frame delivery using MAC address
  - The network layer provides host-to-host packet delivery using IP address
  - The transport layer provides process-to-process message delivery using port numbers
- Port numbers are used in multiplexing many processes by the sender's transport entity and in demultiplexing by the receiver's transport entity

The Transport Layer (2)



The network, transport, and application layers.

The Transport Layer (3)



TPDU: Transport Protocol Data Unit

The Transport Layer (4)

- The client-server paradigm: Connect-Accept
  - Identified by a pair of Socket addresses
- Socket (srcIP, srcPort, dstIP, dstPort)
  - Send/Write; Receive/Read; Close
- Well-known port numbers: www.iana.org

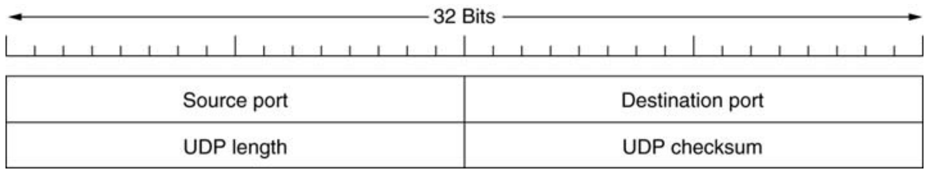
Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

Some assigned TCP ports

User Datagram Protocol (UDP)

- Unreliable connectionless protocol
- Just IP with a short header to identify the *source* and *destination* ports (multiplexing-demultiplexing)
- The *length* field includes the 8-byte header
- The *checksum* filed is optional
  - Includes user data, UDP header, and pseudoheader (similar to TCP)
- Applications
  - Client-server: short request- reply interaction (e.g. DNS, RPC)
  - Real-time Transport Protocol (RTP, RTCP) for Internet multimedia applications

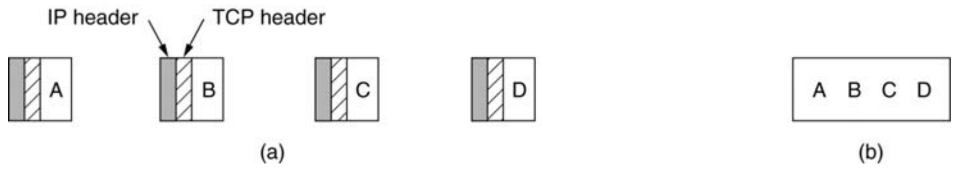
The UDP header



Transmission Control Protocol (TCP)

- Reliable connection-oriented protocol
  - full-duplex point-to-point byte stream protocol
  - Message boundaries are not preserved
- TCP dynamically adapt to properties of the internetwork
  - Flow control
- TCP may send data immediately or buffer it
  - PUSH flag
  - URGENET flag

The TCP Service Model



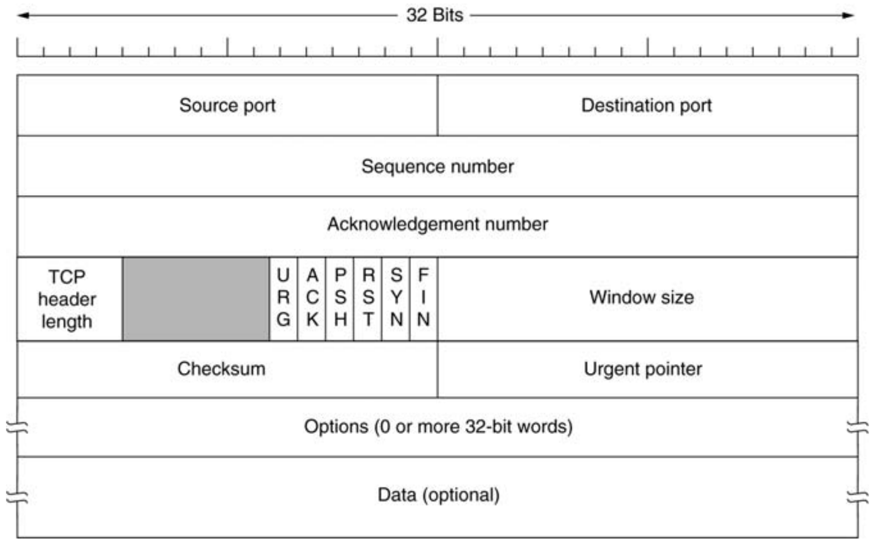
(a) Four 512-byte segments sent as separate IP datagrams.

(b) The 2048 bytes of data delivered to the application in a single READ CALL.

The TCP Protocol

- TCP segment (encapsulated into IP datagram)
  - 20-byte fixed header + optional header + 0 or more data bytes
- Segment size limits
  - IP maximum payload is 65,515 bytes
  - Network Maximum Transfer Unit (Ethernet MTU is 1500 bytes)
- Every byte has its own 32-bit sequence number
- Flow control + Congestion control
  - Go-back-n sliding window protocol (variable-sized window)
- Error control
  - checksum; ACK; time-out
    - ACK number is cumulative and specifies the next byte expected
  - out of order segment delivery; duplicates

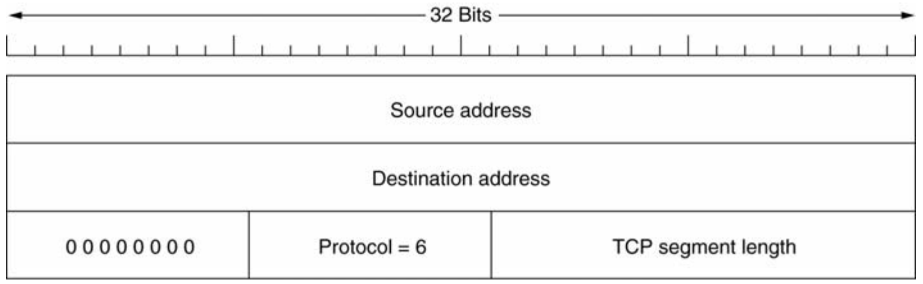
TCP Segment Header



TCP Segment Header (2)

- *Source port, Destination port*: 16-bit numbers
- *Sequence number, Acknowledgment number*: 32-bit numbers
- *header length*: how many 32-bit words
- URG flag is set: *Urgent pointer* is valid (offset from the *sequence number*)
- ACK flag is set: *Acknowledgment number* is valid
- RST flag is set: connection is confused
- SYN flag is set: connection establishment
- FIN flag is set: connection release
- *Window size*: 16-bit number that specifies how many bytes may be sent
- *Checksum:* 16-bit that includes checksum for header + data + pseudoheader
  - Pseudoheader violates protocol hierarchy
- Options:
  - Specify maximum TCP payload (default 536)
  - Window scale to shift window size
  - Selective repeat instead of go back n protocol

TCP Segment Header (3)

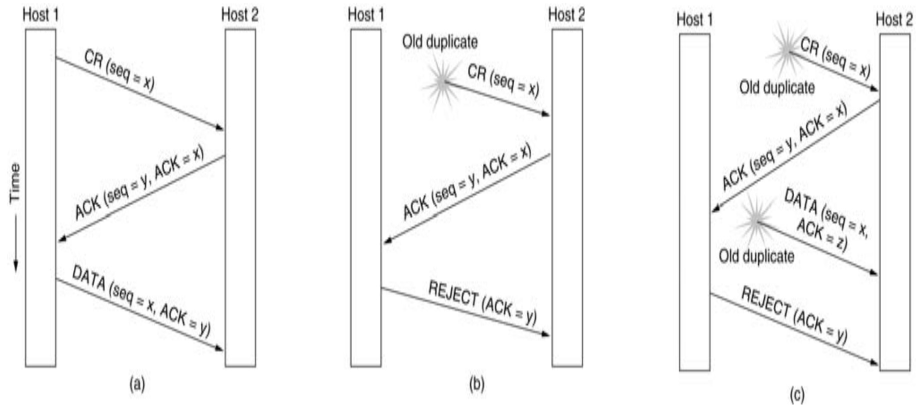


The pseudoheader included in the TCP checksum.

TCP Connection Establishment

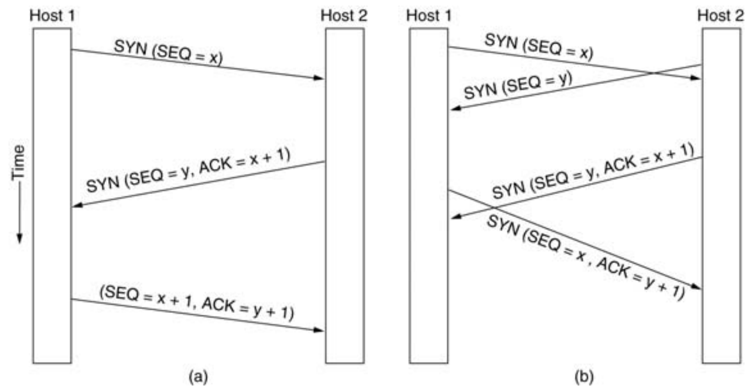
- Network can lose, store, and duplicate packets
  - Bank connection and the delayed duplicate problem
- Initial sequence number is chosen by a clock-based scheme
  - Clock tick every 4 micro-sec
  - Low-order k bits of the clock are used as initial sequence number
  - a host may not reboot after a crash for the maximum packet lifetime
- Three-way handshake protocol
  - Two different initial sequence numbers to avoid synchronization
  - Server executes *Listen*
  - Client executes *Connect* specifying the server IP address and port: TCP segment sent (SYN=1, ACK=0, SEQ = X)
  - Server executes *Accept*: TCP segment sent (SYN = 1, ACK=X+1, SEQ = Y)
  - Client acknowledgement segment (ACK=Y+1, SEQ=X+1, data if any)

Three-way Handshake Protocol Scenarios



CR denotes CONNECTION REQUEST.
(a) Normal operation, (b) Old CR appearing out of nowhere. (c) Duplicate CR and duplicate ACK.

TCP Connection Establishment (2)



Host 1
Host 2
Time
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x , ACK = y + 1)
(b)

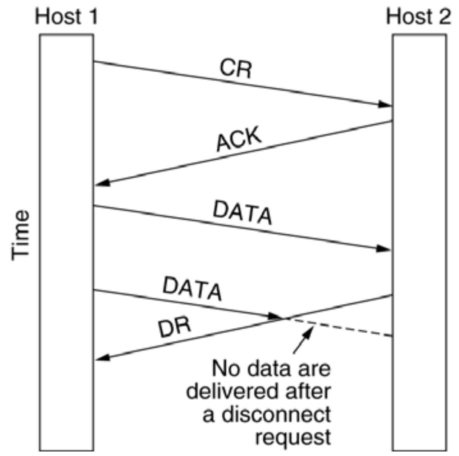
(a) TCP connection establishment in the normal case.

(b) Call collision.

Connection Release Problem

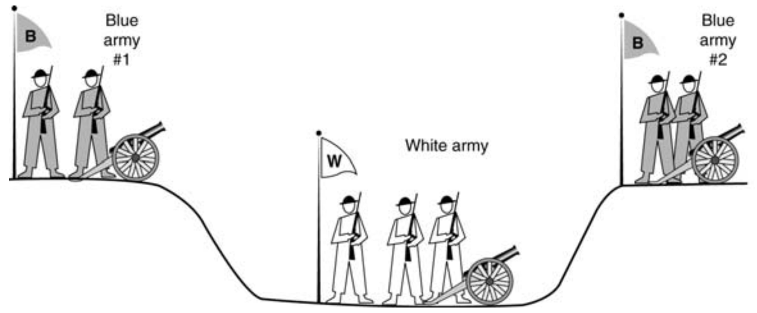
- Asymmetric release may lead to lost data
- Two-army problem
  - no-protocol exits that works
- Three-way handshake for releasing a connection
  - Can fail if initial DR and N retransmission are all lost (half open connection)
  - Half open connections can be killed by having a keepalive timer
- Perfect solution is theoretically impossible

Asymmetric Release

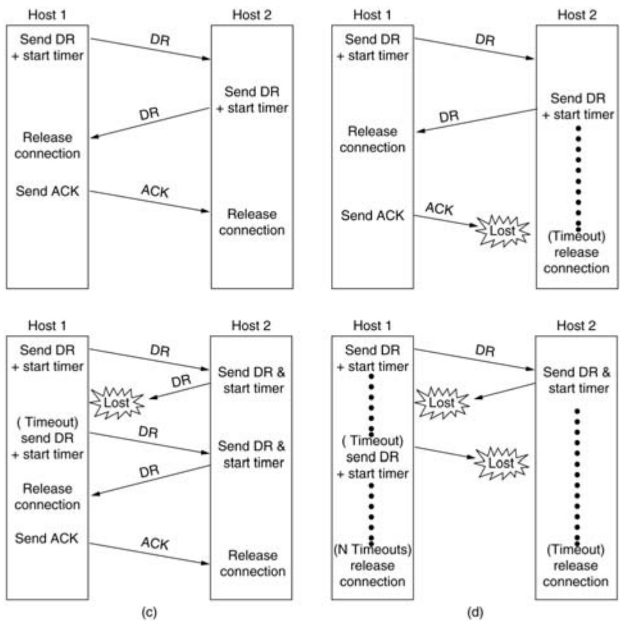


Abrupt disconnection with loss of data.

The two-army problem



Three-way handshake for Disconnection



DR denotes DISCONNECTION REQUEST.

- Normal case.
- final ACK lost.

(c) Response lost.

(d) Response lost and subsequent DRs lost.

TCP Connection Release

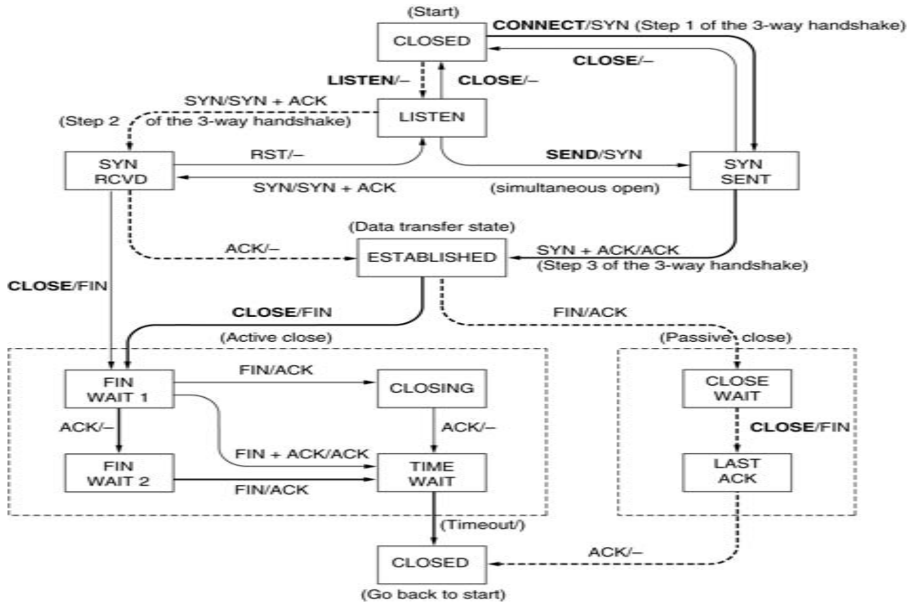
- TCP full duplex connection is a pair of simplex connections that are released independently (symmetric release)
  - When a FIN is acknowledged, that direction is shut down
  - Data may continue to flow on the other direction
  - FIN response timeout = 2 x maximum packet lifetime

TCP Connection Management

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

The states used in the TCP connection management finite state machine.

TCP Connection Management Finite State Machine

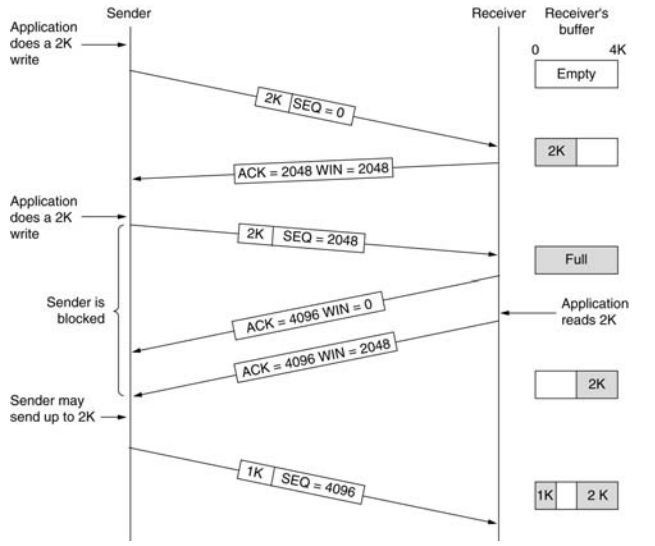


The heavy solid line is the client path. The heavy dashed line is the server path. The light lines are unusual events. Each transition is labeled event/action.

TCP Transmission Policy

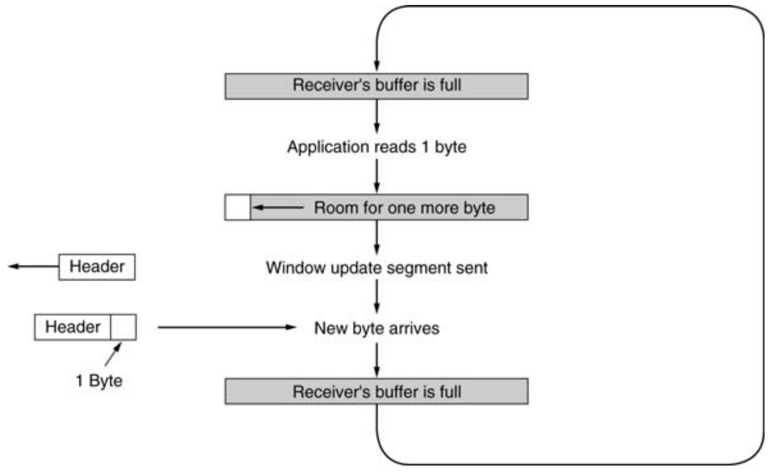
- Acknowledgement and window size are decoupled
- When the receiver window size is 0, the sender can't send except:
  - Urgent data
  - 1-byte to probe the receive to re-announce the next byte expected and the window size
- Silly window syndrome
  - Data are passed to TCP sending entity a byte at a time
  - Data are consumed by the receiver a byte at a time
  - Delay acknowledgment and window updates for 500 msec

TCP Transmission Policy (2)



Window management in TCP.

TCP Transmission Policy (3)



Silly window syndrome.

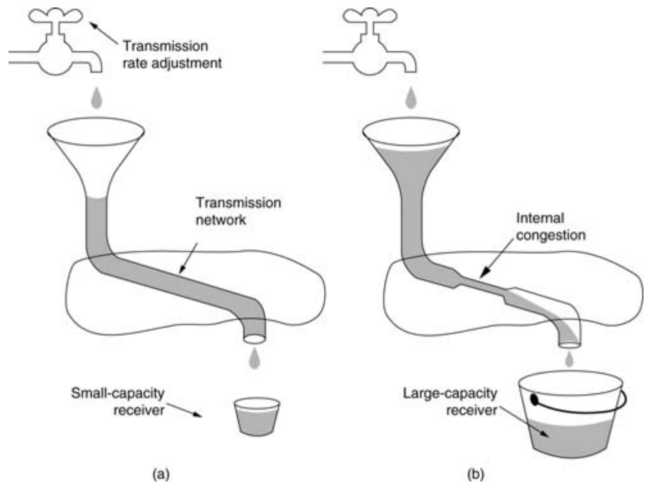
TCP Transmission Policy (4)

- Nagle's sender algorithm
  - A segment can be sent if enough data have trickled in to fill half the window or a maximum segment.
  - When data come into the sender 1 byte at a time, just send the first byte and buffer the rest until the outstanding byte is acknowledged. Then send all the buffered characters in one segment and start buffering again until they are all acknowledged.
  - There are times when it is better to disable it.
- Clark's receiver solution
  - The receiver should not send a window update until it can handle the maximum segment size or until its buffer is half empty (whichever is smaller).

TCP Congestion Control

- The receive specify a window based on its buffer size
- The sender maintains two windows
  - The receiver granted
  - Congestion window (network capacity)
- The sender may send the minimum of the two windows
- Detecting congestion
  - Most transmission timeouts are due to congestion (wireless network!)
- Dynamic congestion window (CW) size

TCP Congestion Control (2)

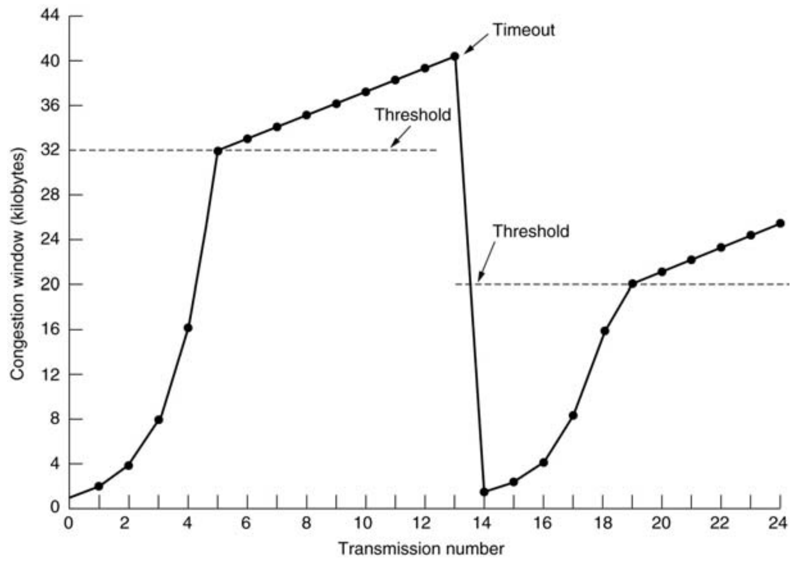


(a) A fast network feeding a low capacity receiver.
(b) A slow network feeding a high-capacity receiver.

TCP Congestion Control (3)

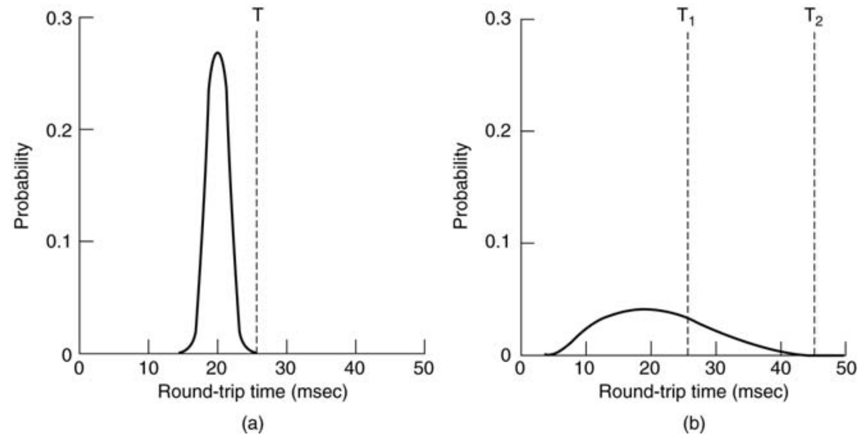
- Initial Threshold = 64 KB
- Initial congestion window (CW) = maximum segment size (S bytes)
- Send CW bytes, if acknowledged before timeout CW = 2 x CW
  - Slow start algorithm (exponential growth)
- When Threshold is hit, CW grow linearly
- Stop when timeout or the receiver window is reached
- If timeout, Threshold = CW/2; CW = S
  - Slow start again

TCP Congestion Control (4)



An example of the Internet congestion algorithm.

TCP Timer Management



(a) Probability density of ACK arrival times in the data link layer.

(b) Probability density of ACK arrival times for TCP.

TCP Timer Management (2)

- Retransmission timer
  - Jacobson: RTT = a RTT + (1- a) M;
    - M time between sending a segment and receiving its ACK
    - a is a smoothing factor , typically 7/8
  - Timeout = b RTT; Initially b = 2
  - Jackson: D = a D + (1 – a) |RTT – M|; Timeout = RTT + 4 D
  - Phil Karn (Karn’s algorithm): do not update RTT on any segment that have been retransmitted. Instead the timeout is doubled on each failure.
- Persistence timer
  - When window size is 0, the sender probes the receiver
- Keepalive timer
- TIMED WAIT state
  - 2 x maximum packet lifetime

Other TCP Issues

- Wireless TCP
- Transactional TCP
- TCP for Gigabit Networks

Reading

- Tanenbaum 6.4.1, 6.5.1-10, 6.2.2, 6.2.3