CS6504

Mobile Computing

Dr. Ayman Abdel-Hamid

Computer Science Department

Virginia Tech

TCP Review

Outline

•Review Transmission Control Protocol (TCP)

Based on

➢Behrouz Forouzan, *Data Communications and Networking*, 3rd Ed, McGraw-Hill, 2004

➢W. Richard Stevens, *Unix Network Programming, Networking APIs: Sockets and XTI*, Vol.1, 3rd Ed, Prentice Hall, 2004

TCP

•TCP is a transport-layer protocol offering stream connection-oriented and reliable transport protocol

•Stream Delivery Service

- ➢stream of bytes (in UDP, every datagram is independent from the other)
- ➢Sending process produces a stream of bytes and receiving process consumes (production and consumption have different speeds → buffers for storage)
  - ✓Sending buffer
    - ❑Empty locations that can be filled by sending process
    - ❑Bytes that have been sent but not yet acknowledged
    - ❑Bytes to be sent by sending TCP
  - ✓Receiving buffer
    - ❑Empty locations to be filled with bytes received from the network
    - ❑Bytes that can be consumed by the receiving process

Bytes and Segments

•IP layer sends data in packets, not as stream of bytes

•TCP groups a number of bytes together into a packet called a segment

•TCP adds a header and delivers segment to IP layer for transmission

•Segments encapsulated in an IP datagram and transmitted

•Segments may be received out of order, lost, or corrupted and resent

•Segments are not necessarily the same size

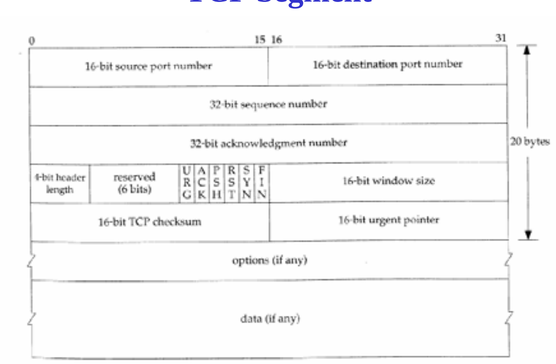
TCP Services

- Full-duplex service
  - ➢Data can flow in both directions at the same time
  - ➢Each TCP has a sending and receiving buffer
- Connection-oriented service
  - ➢When process at site A wants to send and receive data from another process at site B
    - ❑A's TCP informs B's TCP and gets approval from B's TCP
    - ❑A's TCP and B's TCP exchange data in both directions
    - ❑After both processes have no data left to send and the buffers are empty, two TCPs destroy their buffers
  - ➢A virtual connection is created
- Reliable service
  - ➢TCP uses an ACK mechanism to ensure arrival of data

Numbering

- Numbering Bytes
  - ➢Two fields are used sequence number and acknowledgment number. Both refer to byte number and not segment number
  - ➢All data bytes transmitted are numbered (independent in each direction)
  - ➢Numbering does not necessarily start from zero
- Sequence number
  - ➢Sequence number for each segment is the number of the first byte carried in that segment
- Acknowledgment number
  - ➢The ACK number denotes the number of the next byte that this party expects to receive (cumulative)

TCP Segment



Connection Establishment

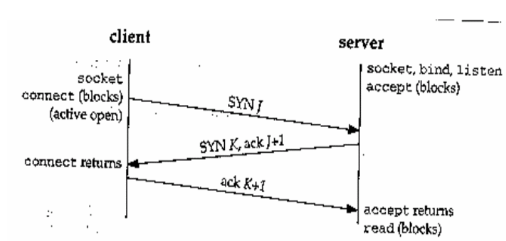


Figure 2.2 TCP three-way handshake.

Connection Termination

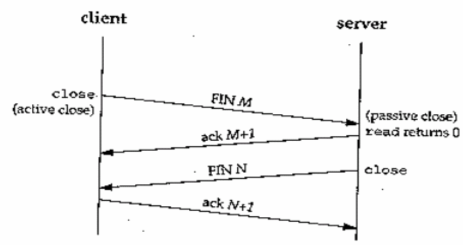
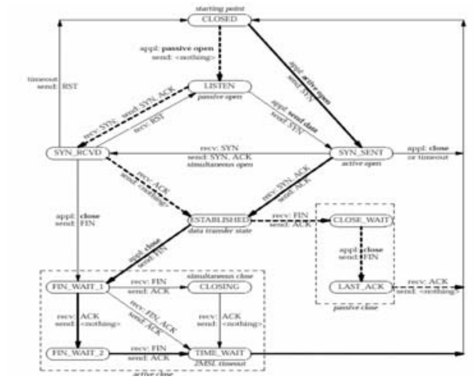


Figure 2.3 Packets exchanged when a TCP connection is closed.

State Transition Diagram



Flow Control

•Sliding window protocol

➢Maintain a window for each connection

➢Window spans a portion of the buffer containing bytes that a host can send before worrying about an ACK from other host

•Receiver window (number of empty locations in receiver buffer)

•Sender window <= receiver window (flow control)

➢Sender window includes bytes sent but not acknowledged and those can be sent

➢Sliding sender window (without a change in receiver's advertised window)

➢Expanding sender window (receiving process consumes data faster than it receives)

➢Shrinking sender window (receiving process consumes data more slowly than it receives)

Error Control

•Mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments

•Tools: checksum, ACK, and time-out

➢Checksum → corrupted at destination

➢ACK → confirm receipt of segments arriving uncorrupted

➢Time-out → one time-out counter per segment

•Lost segment or corrupted segment are the same situation: segment will be retransmitted after time-out

•Duplicate segment (destination discards)

•Out-of-order segment (destination does not acknowledge, until it receives all segments that precede it)

•Lost ACK (loss of an ACK is irrelevant, since ACK mechanism is cumulative)

TCP Timers 1/3

•Retransmission timer

- ➢Retransmission time based on RTT
- ➢Most common, Retransmission time = 2* RTT
- ➢RTT calculated dynamically as
  - ❑RTT = α * old RTT + (1- α)* new RTT where α usually 90%
- ➢Karn's Algorithm
  - ✓A segment not ACKed and retransmitted
  - ✓later, an ACK is received → is ACK for original segment or retransmitted?
  - ✓Do not consider the RTT of a retransmitted segment in calculation

TCP Timers 2/3

•Persistence timer

- ➢Sending TCP stops transmitting segments upon receiving a zero window-advertisement from receiver
- ➢Later, receiving TCP sends an ACK announcing a non-zero window size → this ACK is lost?
- ➢Start a persistence timer when receive zero window advertisement
- ➢After time-out, send a special segment called a probe (1 byte of data, which is not ACKed)
- ➢Value of persistence time set equal to retransmission timer
- ➢If no response from receiver, another probe sent and value of timer doubled and reset
- ➢Repeat until value reaches a threshold (usually 60 seconds)
- ➢Send 1 probe every threshold until window is reopened

TCP Timers 3/3

•Keep-Alive timer

- ➢Prevent long idle connection between 2 TCPs
- ➢Time-out usually around 2 hours
- ➢If does not hear after time-out, send a probe
- ➢If no response after 10 probes (each of which is 75 seconds apart), assume other party is down and terminate connection

•Time-waited timer

- ➢Used during connection termination
- ➢Deal with old duplicates in case of incarnation of previous connection, or to resend final ACK if necessary
- ➢Usually 2 times the expected lifetime of a segment

Congestion Control 1/4

•TCP assumes the cause of a lost segment is due to congestion in the network

•If the cause of the lost segment is congestion, retransmission of the segment does not remove the problem, it actually aggravates it

•The network needs to tell the sender to slow down (affects the sender window size in TCP)

•Actual window size = Min (receiver window size, congestion window size)

- ➢The congestion window is flow control imposed by the sender
- ➢The advertised window is flow control imposed by the receiver

Congestion Control 2/4

•Slow start

➢At start of connection, set congestion window size to maximum segment size

➢For each segment ACKed, increase congestion window size by 1 maximum segment size *until it reaches a threshold of one-half allowable window size*

➢Exponential increase in size

✓Send 1 segment, receive 1 ACK, increase size to 2 segments

✓Send 2 segments, receive 2 ACKs, increase size to 4 segments

✓Send 4 segments, receives 4 ACKs, increase size to 8 segments

Congestion Control 3/4

•Additive Increase (Congestion Avoidance phase)

➢After size reaches threshold, size is increased one segment for each ACK, even if ACK is for several segments (this continues as long as ACKs arrive before time-outs, or congestion window reaches the receiver window value)

•Multiplicative Decrease

➢If a time-out occurs, threshold set to one-half of last congestion window size, and congestion window size starts from 1 (return to slow start)

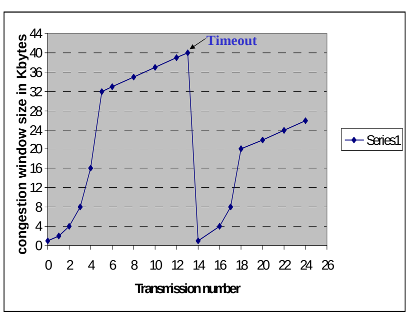
➢Threshold reduced to one-half current congestion window size every time a time-out occurs (exponential reduction)

➢Exponential growth stops when the threshold is hit

➢Afterwards, successful transmissions grow congestion window linearly

•Such congestion control often referred to as TCP Tahoe

Congestion Control 4/4



TCP Variants 1/3

•TCP Tahoe (first implemented in 4.3 BSD, 1988)

➢Slow start + Congestion avoidance and fast retransmit

➢Fast Retransmit

✓Triggers the transmission of a dropped segment if three *dup ACKs* for a segment are received before the occurrence of the segment's timeout

✓TCP required to immediately generate a dup ACK if an out-of-order segment is received (on receiving a dup ACK, cant tell if the reason is a reorder of segments or a lost segment, hence the wait to receive a number of dup ACKs)

✓Fast Retransmit was incorrectly followed by slow start

TCP Variants 2/3

•TCP Reno

➢The above algorithms + fast recovery

➢Fast Recovery

✓Cancel the slow start phase after a fast retransmission

✓Motive: network was able to deliver the 3 dup ACKs

✓Sender window = min (receiver window, congestion window +*ndup*)

❑*ndup* is the number of duplicates, maintained at 0 until the # of ACKs reaches the threshold (3) and then tracks the number of duplicate ACKs

➢Sender waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK it receives

✓Upon receive of an ACK for new data (a recovery ACK), sender exits fast recovery by setting *ndup* = 0

TCP Variants 3/3

•TCP New-Reno

➢Enhancements to fast recovery in case multiple packets are lost from same window

➢TCP Reno will be taken out of fast recovery (deflate usable window size) if partial ACKs are received (ACK some but not all of the packets outstanding at start of fast recovery)

➢New-Reno is not taken out of fast recovery and will retransmit the packet following the partial ACK without the wait for the retransmission timer (effectively retransmitting one lost packet per RTT until all lost packets are retransmitted)

•TCP Vegas

➢Attempt to detect congestion in routers between source and destination before packet loss occurs (detected by observing RTT, longer RTT → greater congestion

➢Lower rate linearly when imminent packet loss occurs

•TCP with Selective Acknowledgment (SACK)

TCP Selective Acknowledgment

•Due to use of cumulative ACK

➢TCP does not provide sender with sufficient information to recover quickly from multiple packet losses within a single transmission window

•SACK added as an option to TCP

•Each ACK contains information about up to *three* noncontiguous blocks of data that *have been received successfully by receiver*

•Each block of data is described by its starting and ending sequence number

•Due to limited number of blocks, inform sender about most recent blocks received

An example of generating SACK options 1/3

•Assume the left window edge is 5000 and that the data transmitter sends a burst of 8 segments, each containing 500 data bytes

•Case 1: The first 4 segments are received but the last 4 are dropped.

➢The data receiver will return a normal TCP ACK segment acknowledging sequence number 7000, with no SACK option.

•Case 2: The first segment is dropped but the remaining 7 are received.

➢Upon receiving each of the last seven packets, the data receiver will return a TCP ACK segment that acknowledges sequence number 5000 and contains a SACK option specifying one block of queued data:

Triggering Segment	ACK	Left Edge	Right Edge
5000	(lost)		
5500	5000	5500	6000
6000	5000	5500	6500
6500	5000	5500	7000
7000	5000	5500	7500
7500	5000	5500	8000
8000	5000	5500	8500
8500	5000	5500	9000

An example of generating SACK options 2/3

•Case 3: The 2nd, 4th, 6th, and 8th (last) segments are dropped.

➢The data receiver ACKs the first packet normally. The third, fifth, and seventh packets trigger SACK options as follows:

Triggering Segment	ACK	First Block	Second Block	Third Block
5000	5500			
5500	(lost)			
6000	5500	6000/6500		
6500	(lost)			
7000	5500	7000/7500	6000/6500	
7500	(lost)			
8000	5500	8000/8500	7000/7500	6000/6500
8500	(lost)			

An example of generating SACK options 3/3

•Case 3: The 2nd, 4th, 6th, and 8th (last) segments are dropped (cont.)

➢Suppose at this point, the 4th packet is received out of order. At this point the data receiver has only two SACK blocks to report. The data receiver replies with the following Selective Acknowledgment:

Triggering Segment	ACK	First Block	Second Block	Third Block
6500	5500	6000/7500	8000/8500	

➢Suppose at this point, the 2nd segment is received. The data receiver then replies with the following Selective Acknowledgment:

Triggering Segment	ACK	First Block	Second Block	Third Block
5500	7500	8000/8500		

An example SACK Implementation (FF96) 1/3

•First block in a SACK option required to report data receiver's most recently received segment

•Additional SACK blocks repeat most recently reported SACK blocks

•Each SACK option has room for 3 SACK blocks

•Congestion control algorithm is a conservative extension of TCP Reno

➢Main difference is when multiple packets are dropped from one window of data

➢SACK TCP enters fast recovery when data sender receives 3 duplicate ACKs

➢Sender retransmits a packet and cuts congestion window in half

An example SACK Implementation (FF96) 2/3

•During fast recovery

➢SACK maintains a variable called *pipe* (estimated number of packets outstanding in the path)

➢Sender only sends new or retransmitted data when estimated number of packets in path less than congestion window

•*pipe*++ when sender sends a new packet or retransmits an old packet

•*pipe*-- when sender receives a duplicate ACK with a SACK option reporting new data has been received at receiver

•Keep a *scoreboard* that remembers ACKs from previous SACK options (infer which segment needs to be retransmitted)

An example SACK Implementation (FF96) 3/3

•When a retransmitted segment is itself dropped, a retransmission timeout takes effect and slow start is entered

•Sender exists fast recover when a recovery ACK is received (ACKing all data that was outstanding when fast recovery was entered)

•How to handle partial ACKs

- ➢Partial ACKs are ACKs received during fast recovery that advance the ACK number of TCP header, but do not take sender out of fast recovery
- ➢For partial ACKs, sender decrements pipe by 2 packets rather than one