TCP Multiplexing

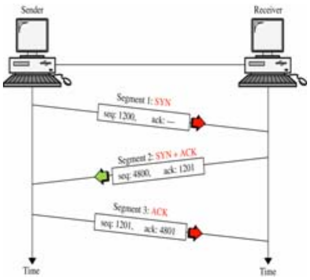
- TCP socket identified by 4-tuple:
  - Source IP address
  - Source port number
  - Destination IP address
  - Destination port number
- Receiver uses all four values to direct segment to appropriate socket
- A server may support many simultaneous TCP sockets:
  - Each socket identified by its own 4-tuple
  - Example:
    * Web servers have different sockets for each connecting client
    * Non-persistent HTTP has a different socket for each request

Reliable Connection Setup

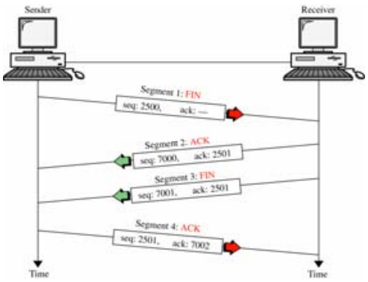
- Frame, packets and primitives in the network may be delayed for one or another reason and unexpectedly appear in their destination at a time where they are unwanted or unexpected
- This in the connection establishment procedure may cause problems, as a delayed connection request shown up suddenly may result in establishing a connection incorrectly
- To solve this problem, Tomlinson in 1975 introduced the three-way handshake
- This establishment protocol allows each side to begin with a different sequence number
- This way delayed primitives can be directly identified and discarded

Three Way Handshake

- Sender executes CONNECT primitive (specifying IP address, port (e.g. 21), the max TCP segment it will accept and other optional user data (e.g. username, password)
- CONNECT primitive sends a TCP segment with the SYN bit on (choosing a sequence number, e.g. 1200) and waits for a response (first arrow)
- When this segment arrives at the receiver, the TCP entity checks to see if there is a process that has done a LISTEN on the port 21 and the gives the incoming segment to that process (i.e. an ftp server)
- If it accepts, an acknowledgement segment is sent back, acknowledging 1200 and announcing its own initial sequence number 4800 (second arrow)
- Finally, sender acknowledges receiver's initial sequence number 4800 and sends data with another initial sequence number
- Then the sequence of TCP segments with data follows

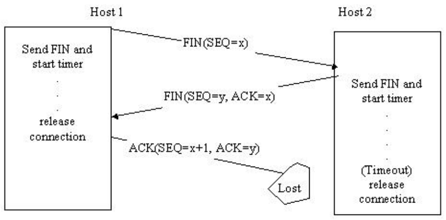


Reliable Connection Shutdown



Shutdown (Final ACK Lost)

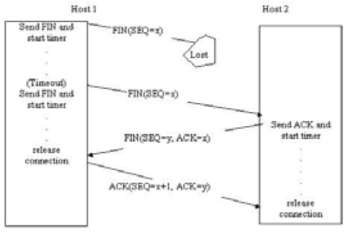
- To cope with lost primitives TCP sets a counter in the receiver when a FIN primitive arrives
- If the final ACK is lost, then the connection is released automatically when it timeouts



5

Shutdown (Request Lost)

- To cope with lost primitives TCP sets a counter in the sender when a FIN primitive is sent
- If the request is lost, then no ACK will ever arrive; a new request is sent automatically when the ACK waiting timeouts



6

TCP Problems and Attacks

- Silly window syndrome (Clark, 1982)

 - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets

 - Solution:

 * Receiver must not advertise small window increases

 * Increase window by min(SMSS, RecvBuffer/2)

7

Nagel’s Algorithm

- Small packet problem:
  - Don’t want to send a 41 byte packet for each keystroke
  - How long to wait for more data?
- Solution:
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged

8

Savage's Attacks

- Congestion control with a misbehaving receiver (greedy Web client)
- Drive a standard TCP sender arbitrarily fast; competing traffic delayed or even discarded

 - ACK division

 - DupACK spoofing

 - Optimistic ACKing

- http://www.cs.washington.edu/homes/savage/papers/CCR99.pdf

9

ACK Division (1/2)

- TCP's error control based on byte offsets within a byte stream
- Congestion control implicitly defined in terms of segments rather than bytes
- Attack

 - Upon receiving a segment of N bytes the receiver calculates the resulting ACK

 - Receiver divides this ACK into M, where M <= N, separate ACKs

 - Each covers one of M distinct pieces of the received segment

10

ACK Division (2/2)

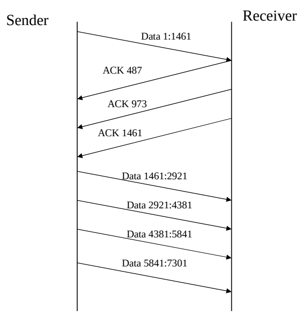
• Sender begins with cwnd=1

• Incremented for each of the three valid ACKs received

• After one RTT cwnd=4 instead of the expected value cwnd=2

• Each ACK is valid, covers data that was sent and previously unknowledged

• Sender's grows cwnd at a rate M faster than usual



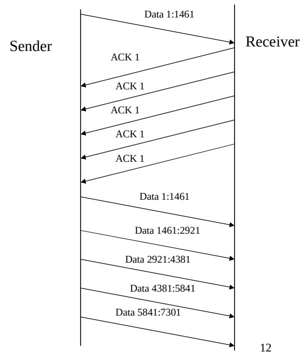
11

DupACK Spoofing

• Upon receiving a data segment the receiver sends a long stream of ACKs for the last sequence number received (start of connection: SYN segment)

• cwnd is increased by SMSS for each additional duplicate ACK

• TCP assumes that duplicate ACKs are sent in response to unique and distinct segments



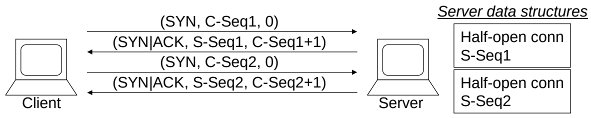
12

Optimistic ACKing

- TCP implicitly assumes that the time between a data segment being sent and an acknowledgement for that segment returning is at least one RTT
- cwnd grown is a function of RTT, sender-receiver pairs with shorter RTTs will transfer data more quickly
- TCP does not use any mechanism to enforce this assumption
- Receiver can emulate a shorter RTT by sending ACKs optimistically for data it has not received yet

13

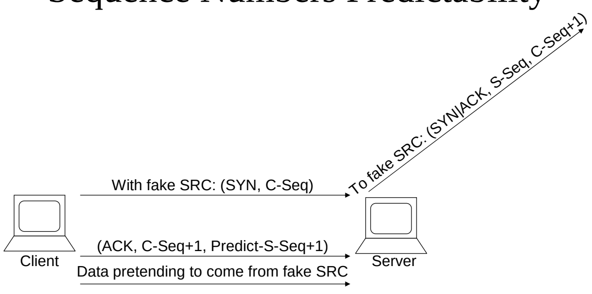
SYN Flooding



- DoS isn't due to traffic volume but to resource exhaustion (memory) in the server
- Early network stacks had a severely limited number of half-open structures available
- Can spoof SRC address with non-existent host
- Solution: SYN cookies – make the SYNACK contents purely a function of SYN contents, therefore, it can be recomputed on reception of next ACK

14

Sequence Numbers Predictability



Particularly dangerous when ``fake SRC'' is a trusted IP address

15