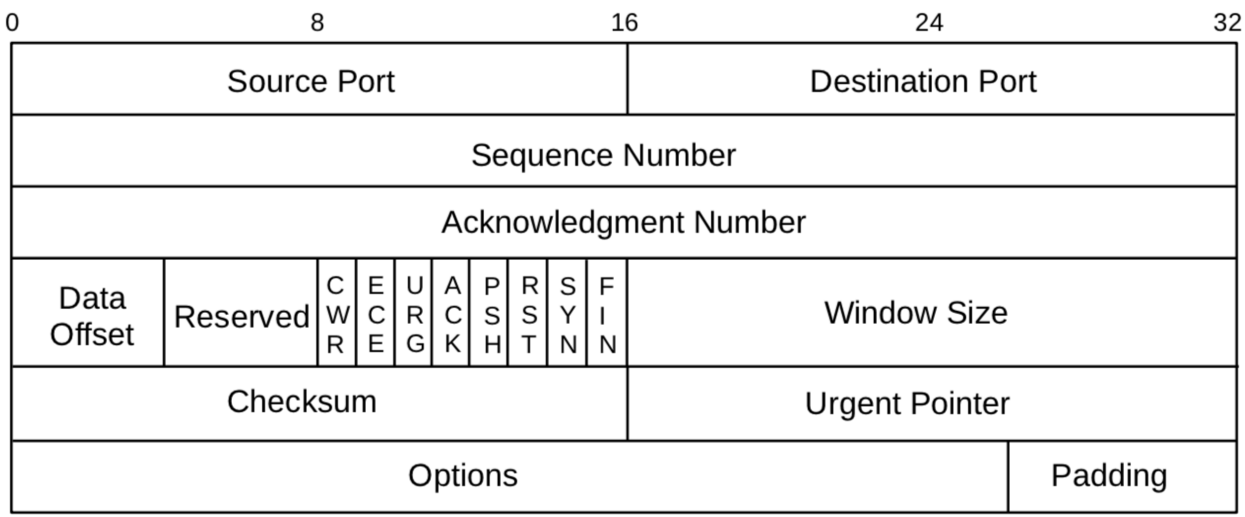
Computer Networking

COMP 177 | Fall 2020 | University of the Pacific | Jeff Shafer

TCP (2)

Transmission Control Protocol

Transmission Control Protocol (TCP)



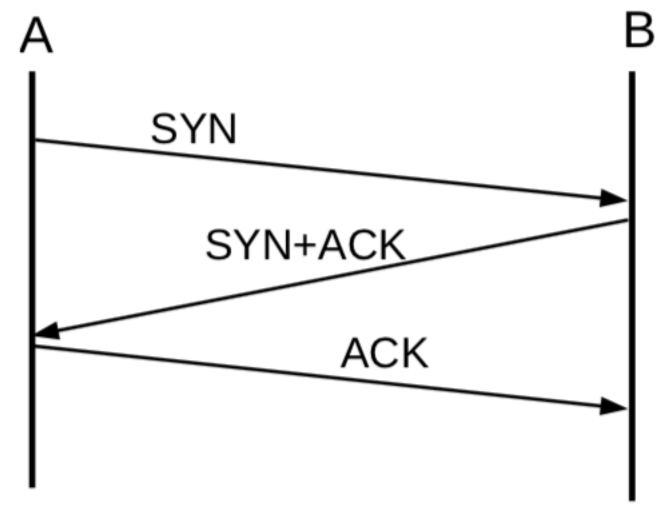
- Connection oriented
- Byte streaming
- Full duplex
- Reliable data transport
- Congestion control
- Flow control

Connection Establishment

- TCP is a *connection-oriented* service
  - A connection between the two endpoints must be established before application layer communication
- TCP connections are established using an exchange called *3-way handshake*.
- Suppose A wants to establish a TCP connection to B. 3-way handshake:
  - A sends a TCP packet to B asking to establish a connection from itself to B
  - B responds back to A, acknowledging the establishment of connection from A to B, and requesting a connection from B to A
  - A responds back to B, acknowledging the establishment of connection from B to A
- Result: *full-duplex* connection between A and B

3-Way Handshake

- Suppose A wants to establish a TCP connection to B. 3-way handshake takes place:
  - A sends B a TCP packet in which *SYN flag* is set. (SYN packet)
  - B responds with a TCP packet in which *SYN and ACK flags* are set. (SYN-ACK packet)
    - The Ack# field in this packet is Seq# of SYN packet + 1.
  - A sends back a TCP packet in which *ACK flag* is set. (ACK packet)
    - The Ack# field in this packet is the Seq# of SYN-ACK packet + 1.

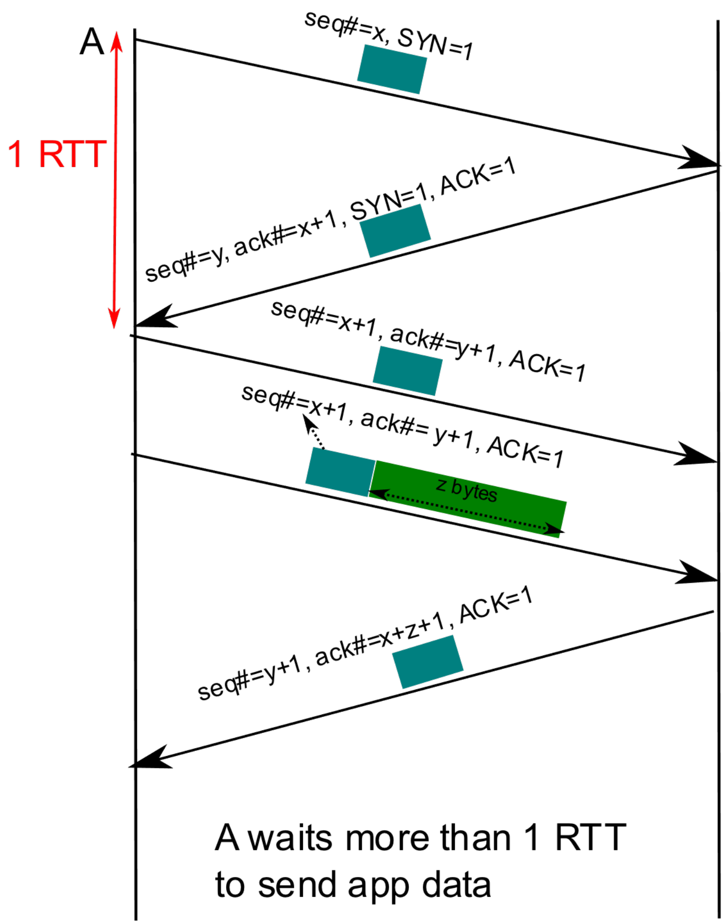


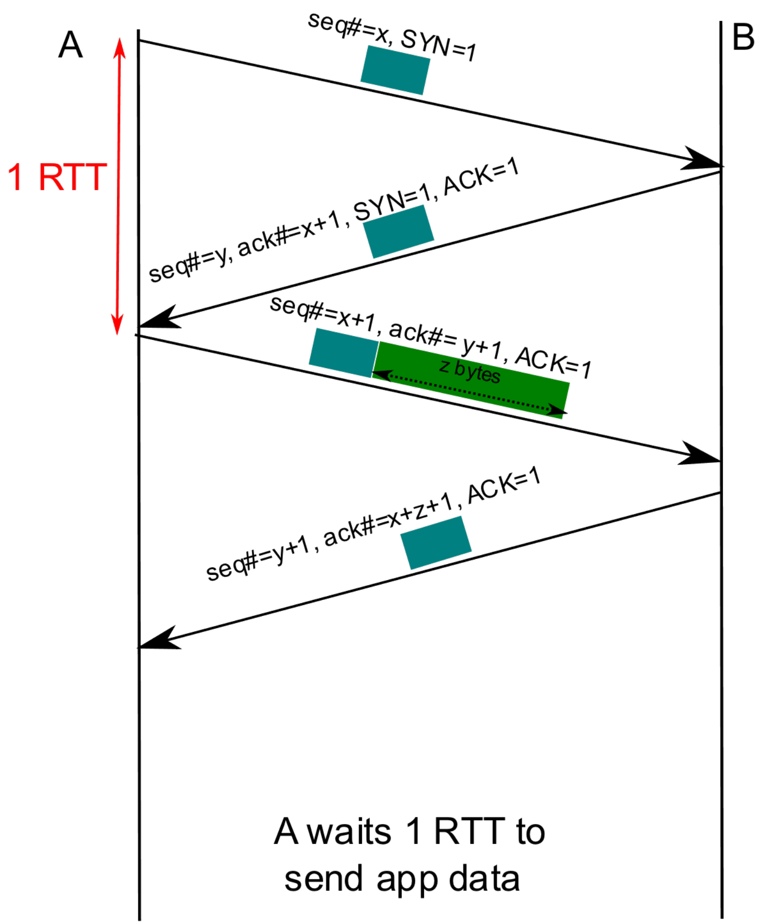
TCP three-way handshake

3-Way Handshake

- 3-way handshake is triggered by an *application-layer process request* to connect to another process
- Application-layer messages (between processes) cannot be communicated until 3-way handshake is completed
- This means that the sender process needs to *at least* wait for one round-trip time (RTT) before app data communication

3-Way Handshake

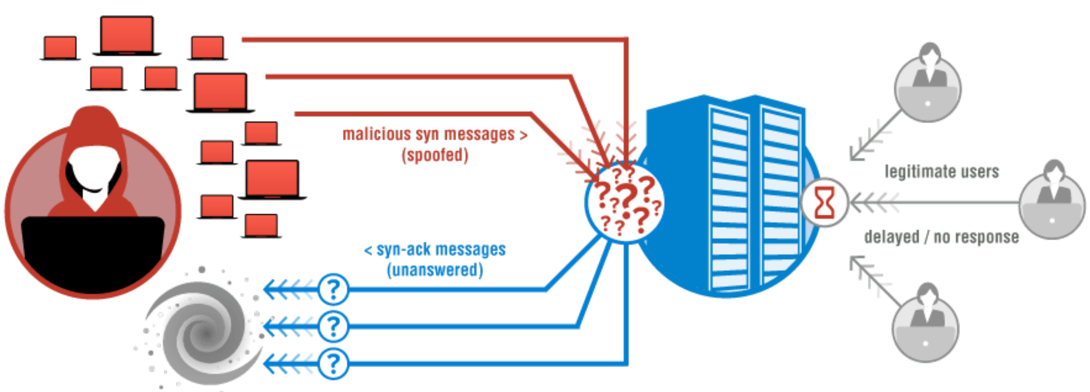




SYN Flooding

- Upon receiving a SYN packet, the server allocates some resources in the system for the upcoming connection, and then sends the SYN-ACK packet to the client
  - The resources include different buffers, state variables, etc.

- This opens the door for a classic *denial of service attack*, called **SYN Flooding**
  - The attacker machines (bots) flood the victim server with SYN packets
    - The source IP on the SYN packet is usually spoofed
  - The victim server allocates resources for each of the received SYN packets, and sends back a SYN-ACK packet to the spoofed IP address
  - The host with the spoofed IP address discards the received SYN-ACK
    - Why?
  - Result: **resource exhaustion** at the server side

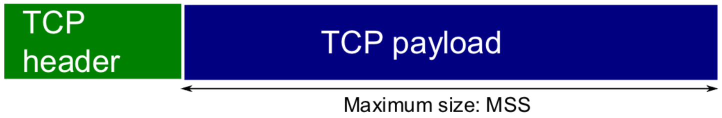


SYN Cookies

- Solution: **SYN cookies**
  - When the server receives the SYN packet, it does not allocate resources yet
  - The server, rather than choosing a random seq# in SYN-ACK, computes the cryptographic hash of
    - Source/destination IP addresses
    - Source/destination port numbers
    - Some data that server knows (e.g., local timestamp)
  - This hash is called a SYN cookie
- The seq# on SYN-ACK is set to be SYN cookie.
  - If the server receives ACK, then recomputes SYN cookie and compares it with Ack# - 1. If equal, then allocates resources

Options in 3-way Handshake

- Different options are negotiated within the 3-way handshake
- **Maximum Segment Size** (**MSS**): Each side may announce its preferred maximum TCP payload size (application layer data), known as MSS.
  - Note that MSS does not include TCP header (only payload)



- **Selective Acknowledgement** (**SACK**):
  - By default the receiver *cumulatively* acknowledges the receipt of packets in Ack# field
  - With SACK option, the receiver can inform the sender about all ranges of bytes arrived successfully, so the sender need retransmit only the segments that have been lost
    - To acknowledge selectively, a left edge and a right edge are specified in the options field
    - All bytes between the two edges are received successfully

Options in 3-way Handshake

- **Timestamp**, consisting of two parts:
  - Timestamp value: report local time of sender
  - Timestamp echo reply: report the timestamp value of the bytes that are being acknowledged
    - This is used to compute RTT

- **Window Scale**: Used to scale the reported size of window in Window Size field
  - If Window Scale is *n*, then the real window size is what is reported in Window Size field $\times 2^n$
  - Represents number of bytes "in flight" across network

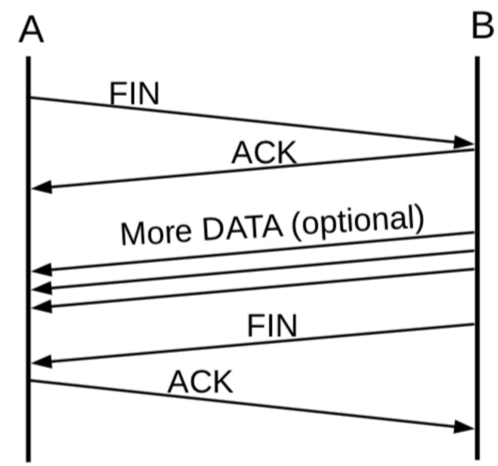
- **No Operation** (**NOP**): used to pad out another option that was used to 32-bit word boundary

Connection Closure

- TCP is a *connection-oriented* service
  - An established connection between the two endpoints TCP services has to be closed after application layer communication
- Suppose A wants to close an already-established TCP connection between itself and B
  - A sends a TCP packet to B, requesting to close the connection from A to B
  - B responds back to A, accepting the closure of the connection from A to B
    - Note that TCP connection between A and B is full duplex
    - At this point, one way of the connection is closed: from A to B
    - *B can still send application layer messages to A, but A cannot send any application message to B*
- At some point, B also realizes that it is time to close its connection to A. So it sends a TCP packet to A, requesting the termination of connection
  - A responds back to B, accepting the closure of the connection from B to A
  - Both directions of the connection are closed

Connection closure: FIN/ACK Handshakes

- Suppose A wants to close an already-established TCP connection to B
  - A sends B a TCP packet in which FIN flag is set (*FIN packet*)
  - B responds with a TCP packet in which ACK flag is set (*ACK packet*)
    - The Ack# field in this packet is Seq# of FIN packet + 1
    - At this point, one way of the connection is closed: from A to B
    - B can still send application layer messages to A
    - A cannot send any application message to B
  - At some point, B sends A a TCP packet in which FIN flag is set (*FIN packet*)
  - A responds with a TCP packet in which ACK flag is set (*ACK packet*)
    - The Ack# field in this packet is the Seq# of previous FIN packet + 1
  - TCP connection closure consists of two FIN/ACK handshakes.



A typical TCP close

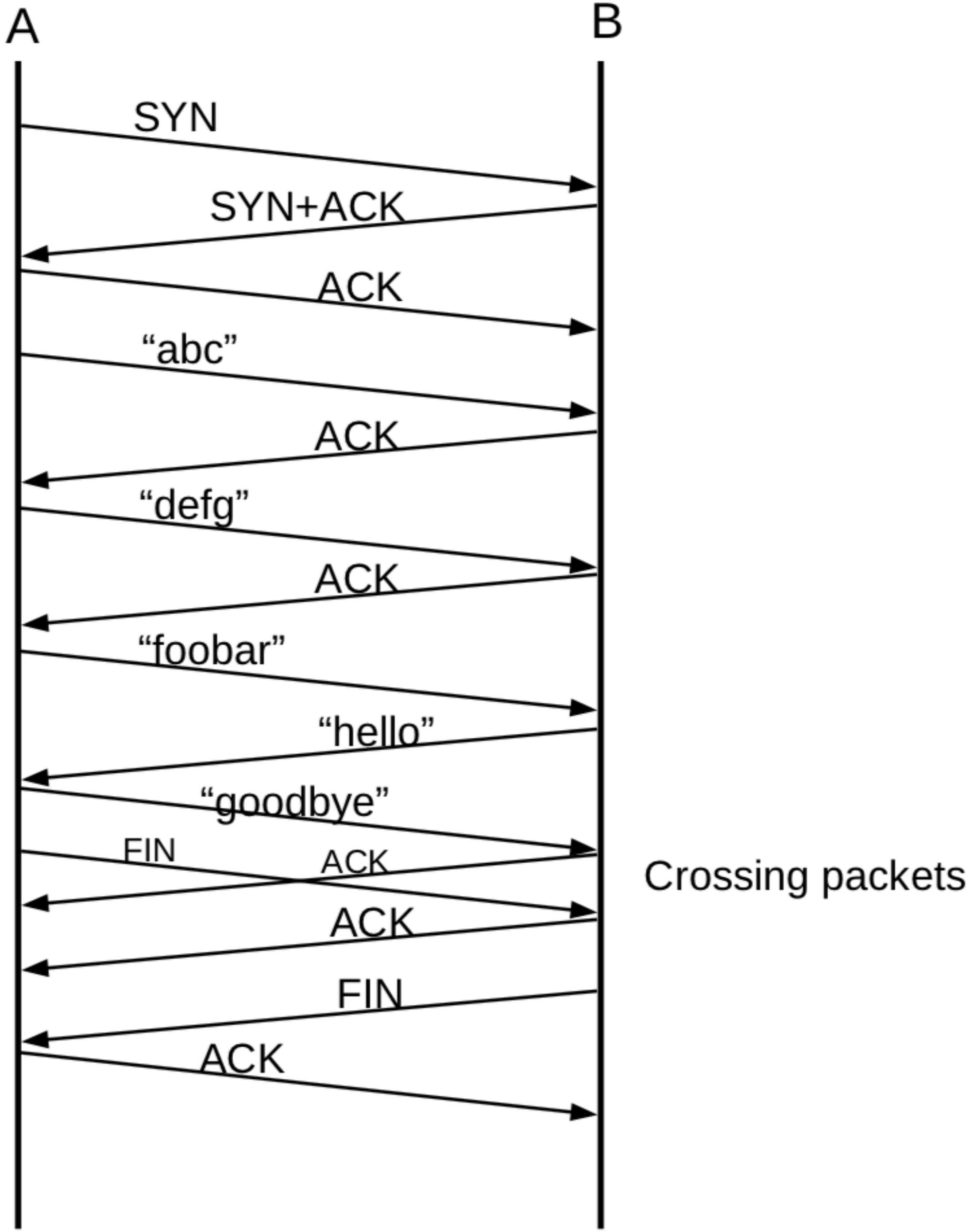
Example: A TCP session

- Suppose a process on host A wants to communicate with a process on host B
- A sends messages “abc”, “defg”, and “foobar”
- B sends back message “hello”.
- A responds with “goodbye” and closes the connection.
- B closes the connection as well
- In the following example, relative sequence & acknowledgement numbers are used (like Wireshark)

Example: A TCP session

	A sends	B sends
1	SYN, **seq=0**	
2		SYN+ACK, seq=0, **ack=1** (expecting)
3	ACK, **seq=1**, ack=1 (ACK of SYN)	
4	"abc", **seq=1**, ack=1	
5		ACK, seq=1, **ack=4**
6	"defg", **seq=4**, ack=1	
7		seq=1, **ack=8**
8	"foobar", **seq=8**, ack=1	
9		seq=1, **ack=14**, "hello"
10	**seq=14**, ack=6, "goodbye"	
11,12	**seq=21**, ack=6, FIN	seq=6, **ack=21** ;; ACK of "goodbye", crossing packets
13		seq=6, **ack=22** ;; ACK of FIN
14		seq=6, **ack=22**, FIN
15	**seq=22**, ack=7 ;; ACK of FIN	

Example: A TCP session



Initial Sequence Number

- TCP does not enforce any specific value for initial sequence number (ISN)
  - ISN can be any 32-bit number
  - Selected by each endpoint and sent to other side in initial 3-way handshake
    - *Sequence number in the SYN and SYN-ACK packets*
- In the following example, ISN is 1000 for A and 7000 for B

Initial Sequence Number

- In the following example, ISN is 1000 for A and 7000 for B

	A, ISN=**1000**	B, ISN=7000
1	SYN, **seq=1000**	
2		SYN+ACK, seq=7000, **ack=1001**
3	ACK, **seq=1001**, ack=7001	
4	“abc”, **seq=1001**, ack=7001	
5		ACK, seq=7001, **ack=1004**
6	“defg”, **seq=1004**, ack=7001	
7		seq=7001, **ack=1008**
8	“foobar”, **seq=1008**, ack=7001	
9		seq=7001, **ack=1014**, “hello”
10	**seq=1014**, ack=7006, “goodbye”	

Closing Thoughts

Recap

- Today we discussed
  - TCP connection establishment
  - TCP SYN flooding attack
  - TCP options
  - TCP connection closure

Next Class

- More TCP

Class Activity

CA.16 – TCP & Wireshark

Due tonight at 11:59pm