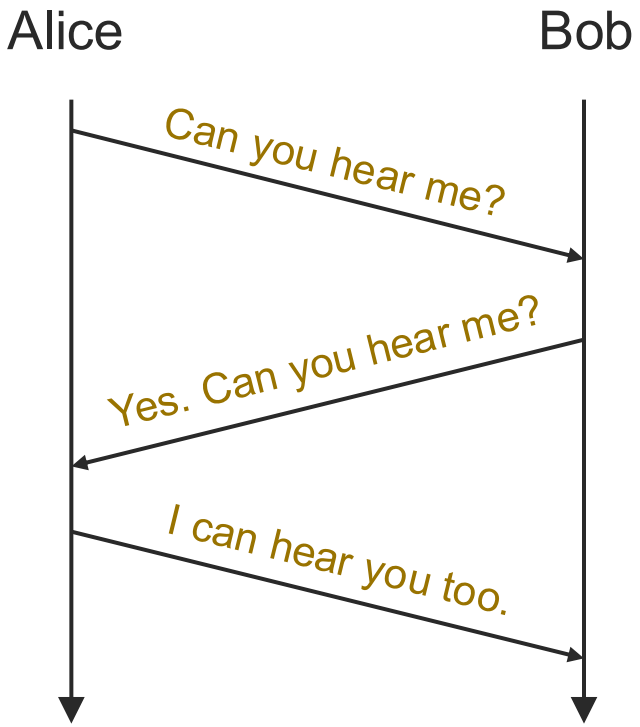
TCP Internals

TCP Usage Model

- Connection setup
- Data transport
  - Sender writes some data
  - TCP ensures reliable communication
    - Breaks data stream into segments
    - Sends each segment over IP
    - Cumulative ACKs
    - Timeout and retransmission
    - Sliding window (flow control, congestion control)
  - Receiver reads some data
- Connection teardown

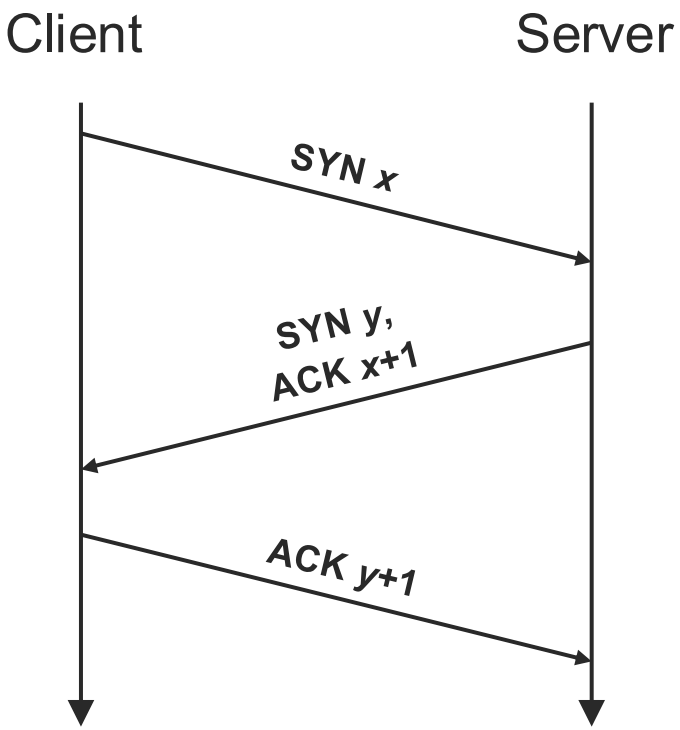
TCP Connection Establishment

- Human protocol analogy
  - Mutual confirmation of an established channel
- The famous TCP 3-way handshake
  - To mutually confirm an established connection
  - To exchange sequence numbers to use



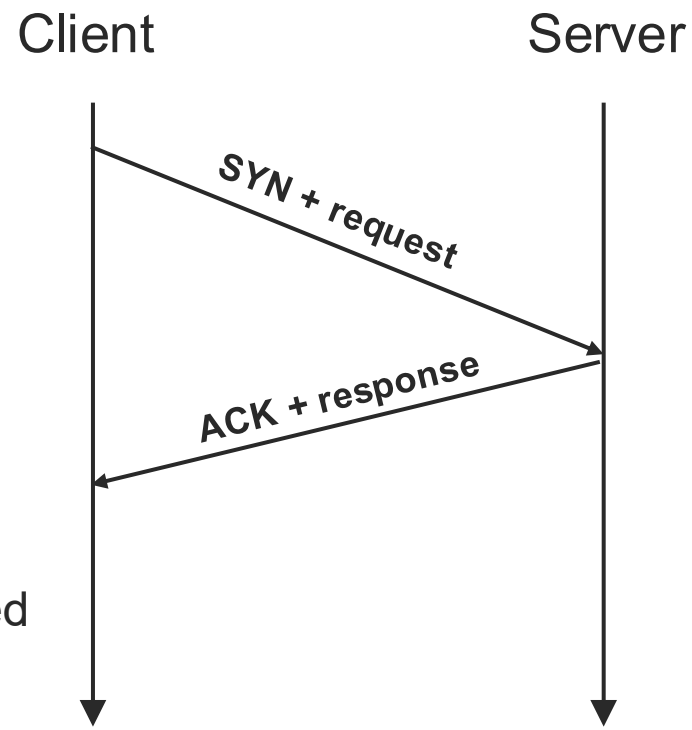
TCP 3-Way Handshake

- Passive open
  - Server listens for connections
- Active open
  - Client initiates the connection
- Message Types
  - SYN (synchronize)
  - ACK
- Client & server maintain *independent* sequence numbers
- cumulative ACK for the *next expected byte*



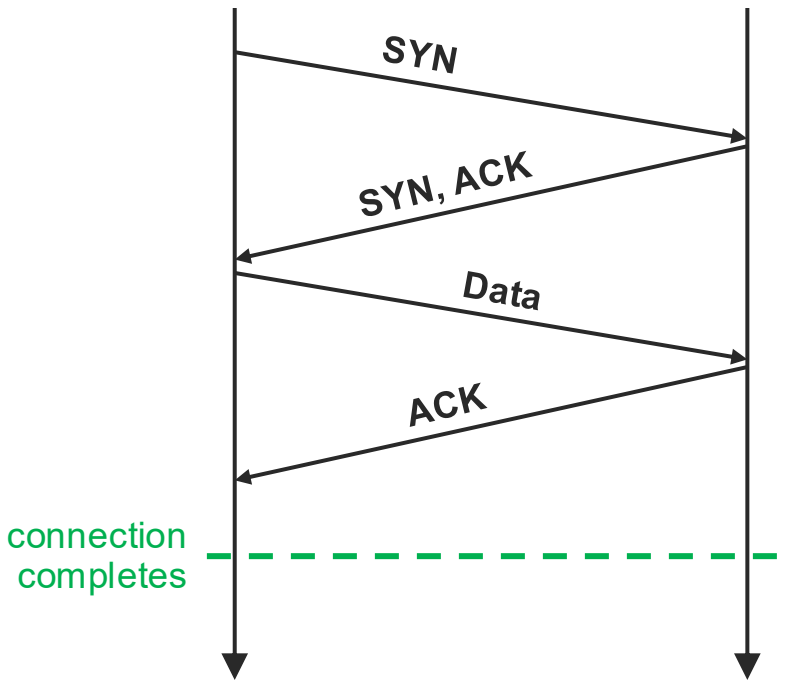
Why not 1-way handshake?

- A (naive) 1-way handshake protocol
  - client sends data (e.g., HTTP request) along with SYN
  - server processes and delivers request to the application
  - server sends ACK + response
- What could go wrong?
  - *Hint:* network is unreliable...
  - Request gets lost → wasted work
  - Request gets delayed & retransmitted → server delivers duplicate request

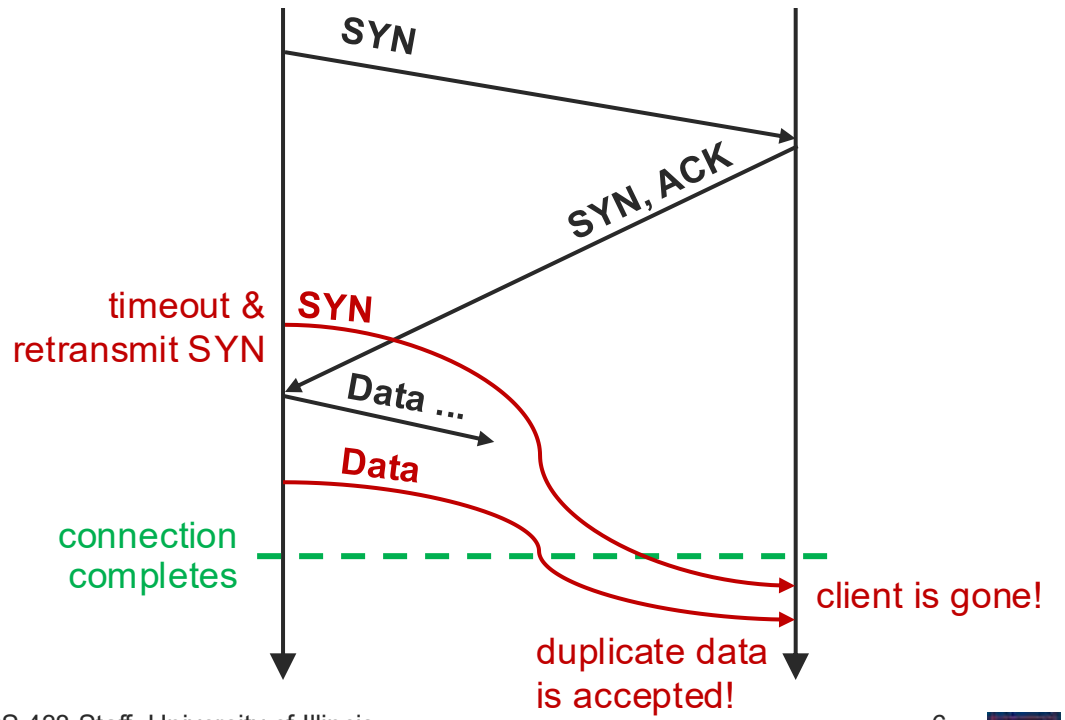


Why not 2-way handshake?

Normal:



Problem:

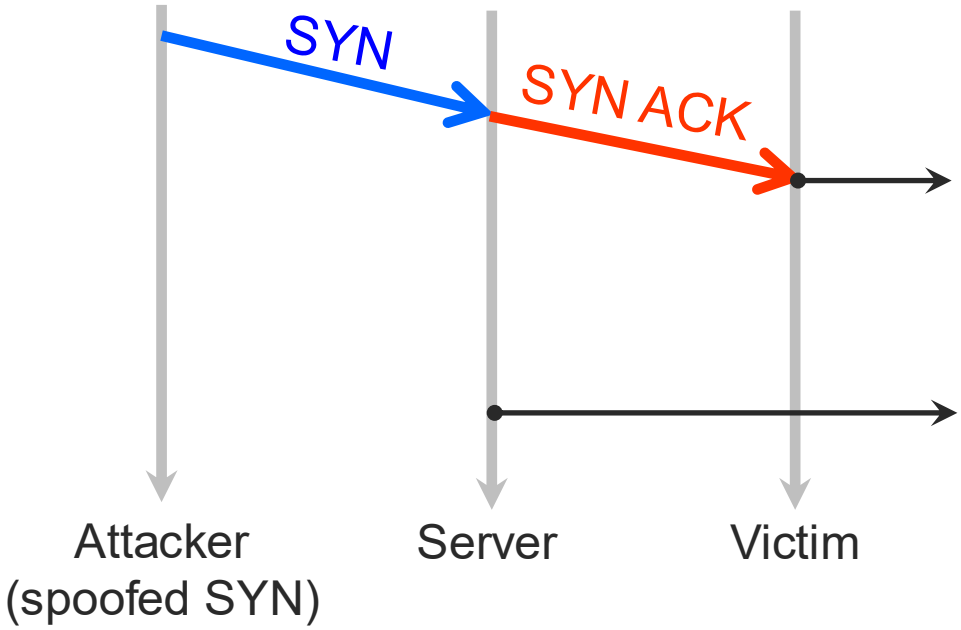


Another purpose of the handshake

- No handshake == security hole
  - Attacker sends request
  - …but spoofs source address, using address of a victim
  - Server happily sends massive amounts of data to victim
  - Attacker repeats for 10,000 web servers
  - Massive denial of service attack, almost free and anonymous for the attacker!
- Used in the largest distributed denial of service (DDoS) attacks in 2008, 2009, and 2010
  - Use services that lack handshake (e.g., DNS over UDP)
  - Amplification factor 1:76 in 2008!

Another purpose of the handshake

- Handshake lets server verify source address is real



Doesn't match a connection initiated by Victim: ignore (or reply with *reset*)

No ACK received after timeout: drop connection without sending data

Q: does this prevent reflection attack?

A: No, but at least it prevents amplification

Handshake is no panacea

- Internet was not designed for accountability
  - Hard to tell where a packet came from
  - ISPs filter suspicious packets: sometimes easy, sometimes hard, and sometimes not done
    - And the Internet is not secure until everyone filters
- More generally, Internet was not designed for security
  - Vulnerabilities in most of the core protocols
  - Even with handshake, early designs are vulnerable
    - Because security was not initial goal of the handshake

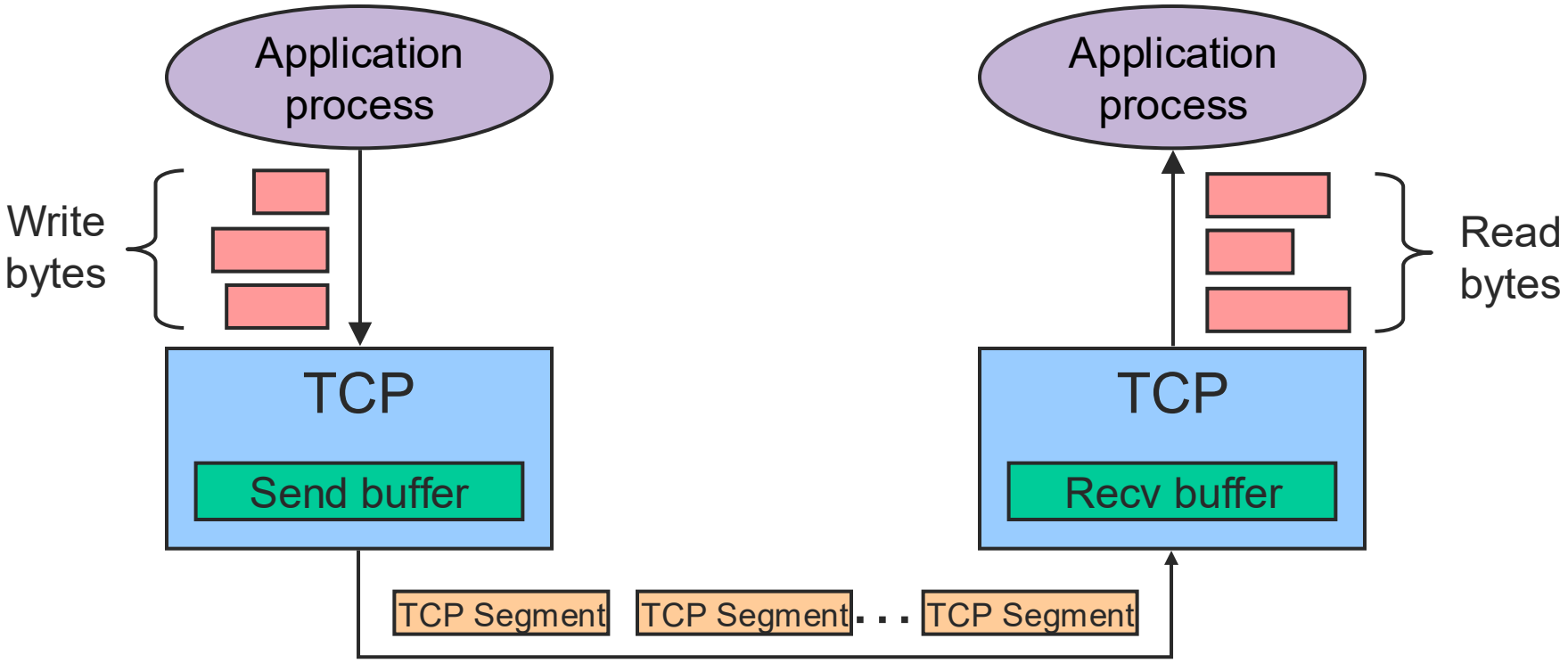
Initial Sequence Number (ISN)

- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … small chance an old packet is still in flight
  - … and might be associated with new connection
- TCP requires (RFC 793) changing ISN
  - Set from 32-bit clock that ticks every 4 microseconds
  - … only wraps around once every 4.55 hours
- What's wrong?
  - ISN is predictable: TCP sequence prediction attacks!

TCP Data Transport

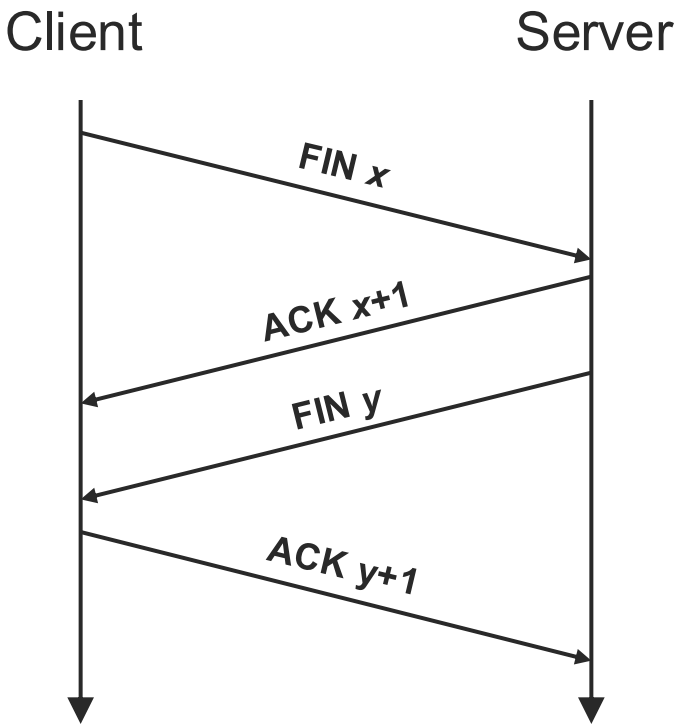
- Data broken into segments
  - Limited by maximum segment size (MSS)
  - MSS negotiable during connection setup
  - Typically set to
    - MTU of directly connected network minus the total size of TCP and IP headers (e.g., 1500 − 20 − 20 = 1460 bytes)

TCP Byte Stream

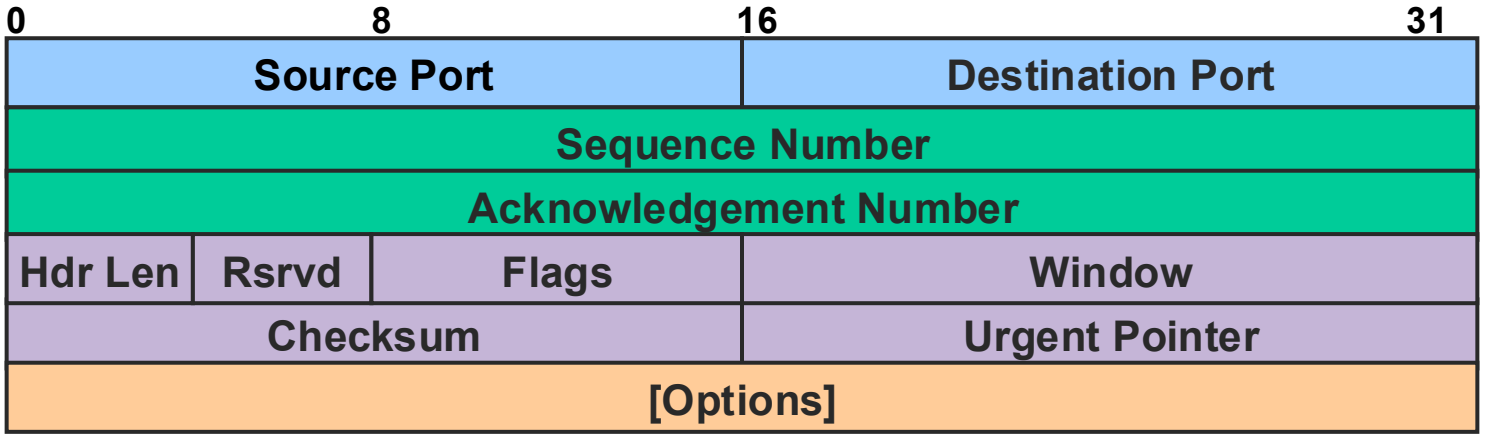


TCP Connection Termination

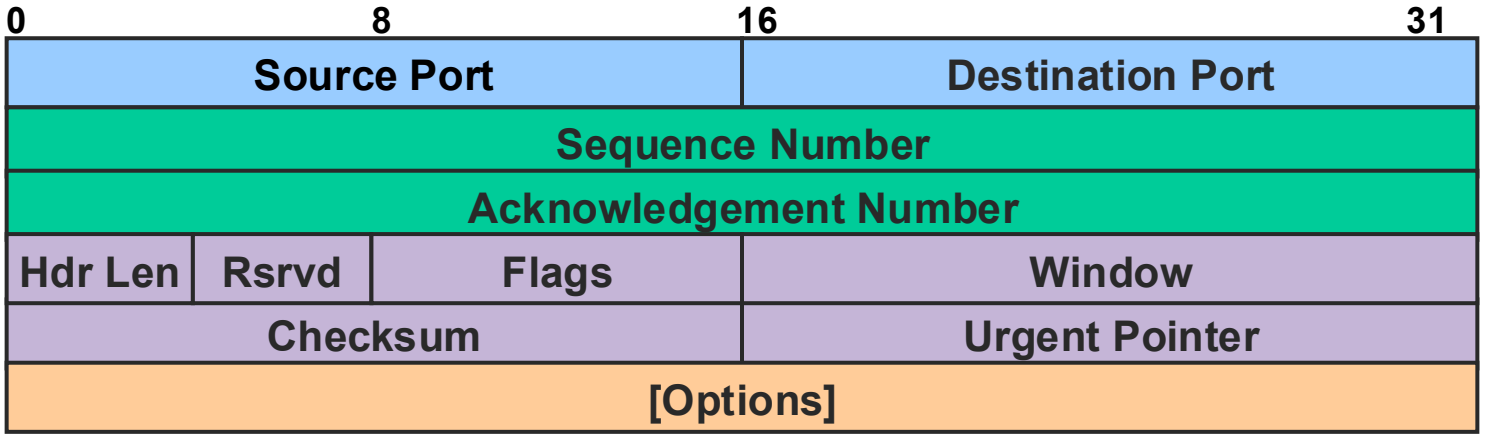
- Message Types
  - FIN (Finished)
  - ACK
- Four steps
  - Connection is full-duplex
  - FIN *x*: client is done sending
  - FIN *y*: server is done sending



TCP Segment Header Format

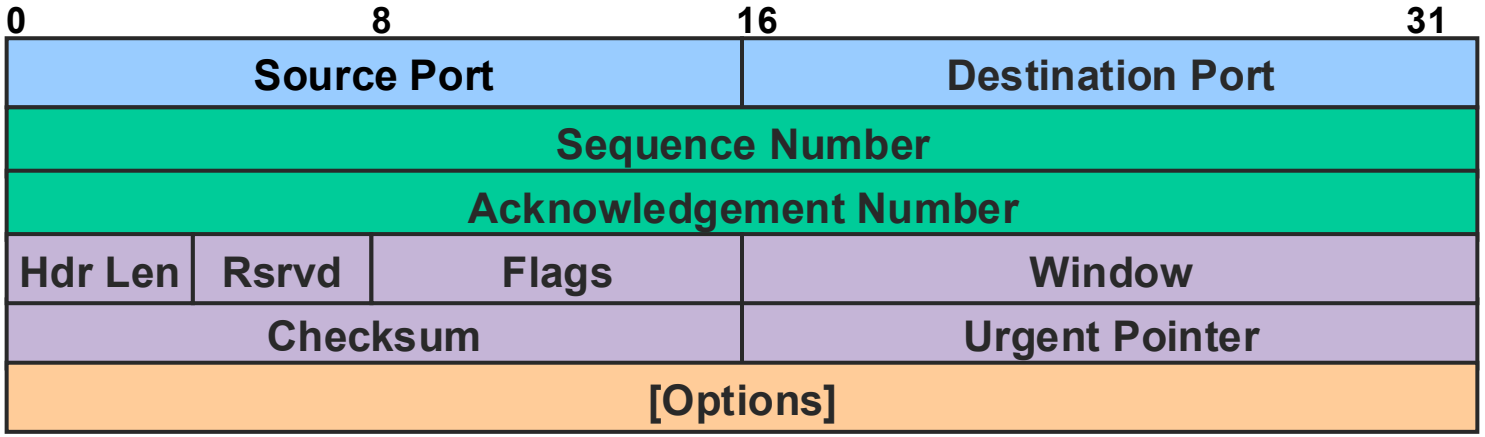


TCP Segment Header Format



- 16-bit source and destination ports

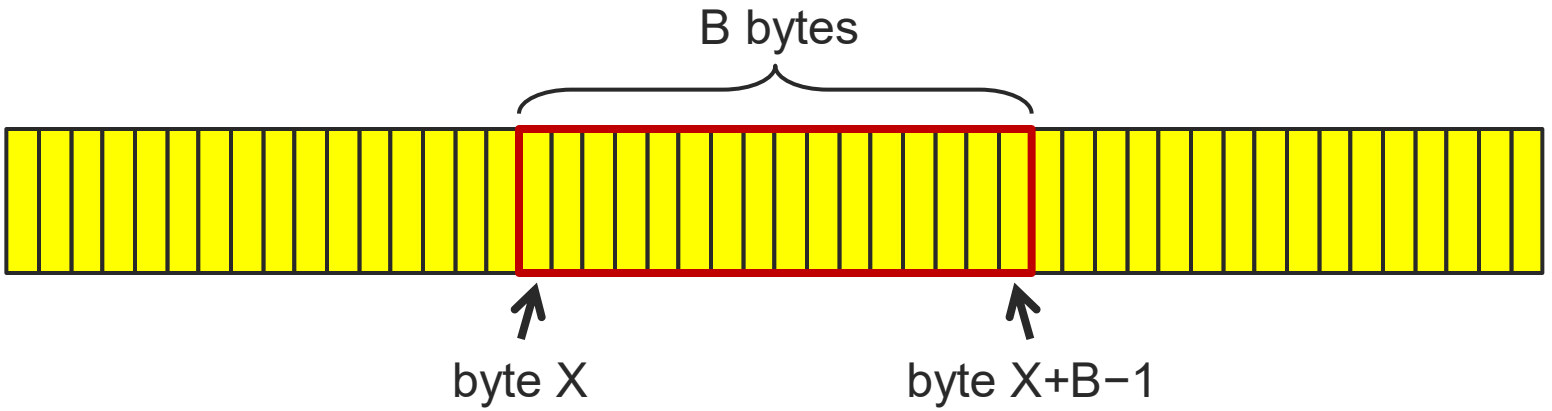
TCP Segment Header Format



- 32-bit sequence number and ACK number
  - Sequence number
    - Index into byte stream
  - ACK number
    - Next byte expected as opposed to last byte received

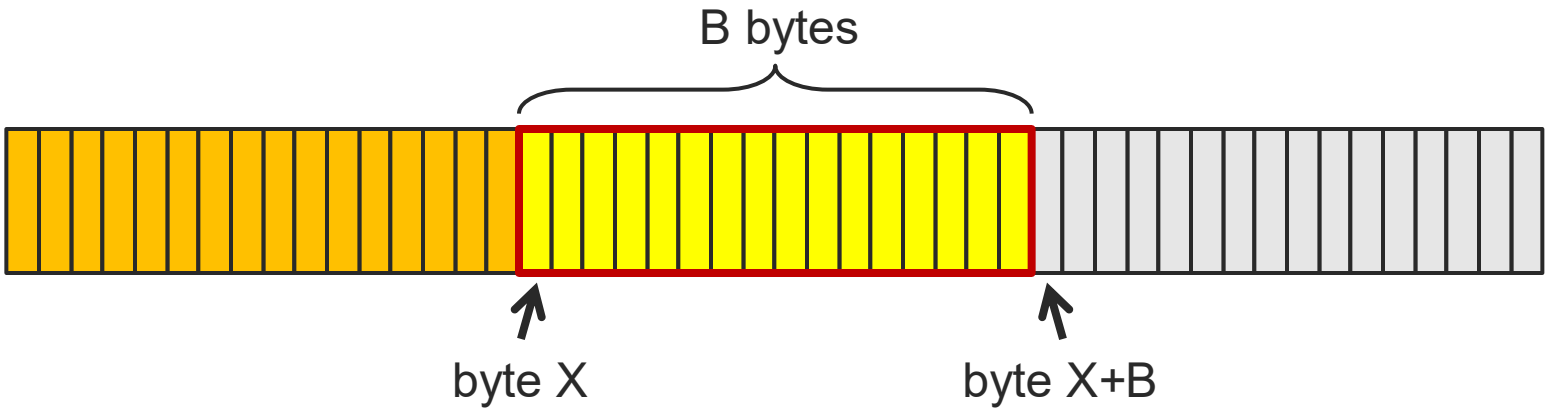
ACKing and Sequence Numbers

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes
    - X, X+1, X+2, ….X+B−1



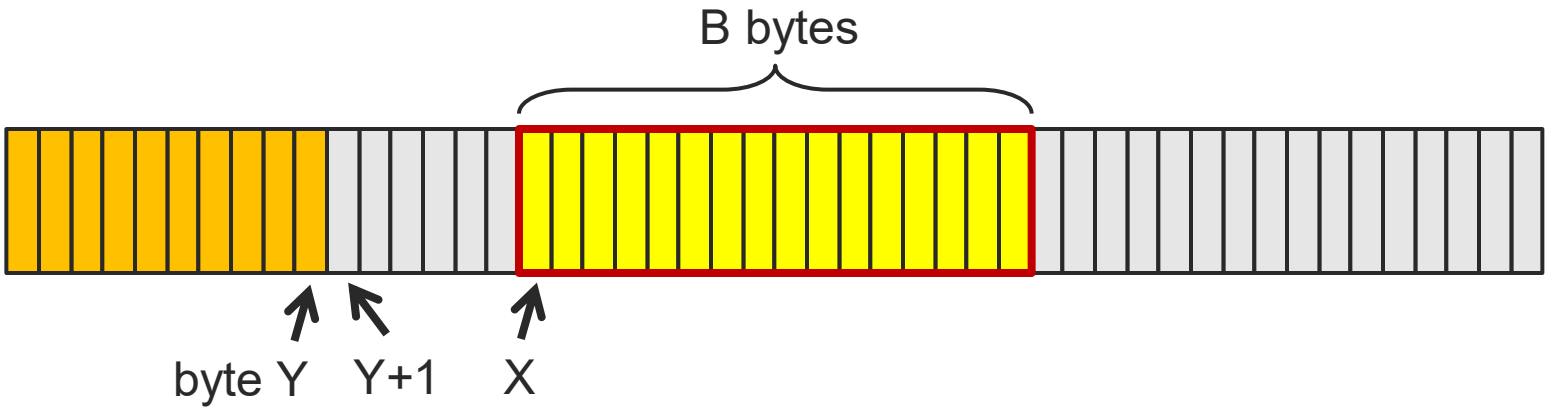
ACKing and Sequence Numbers

- Upon receipt of packet, receiver sends an ACK
- If all data prior to X already received:
  - ACK acknowledges X+B
    (because that is next expected byte)

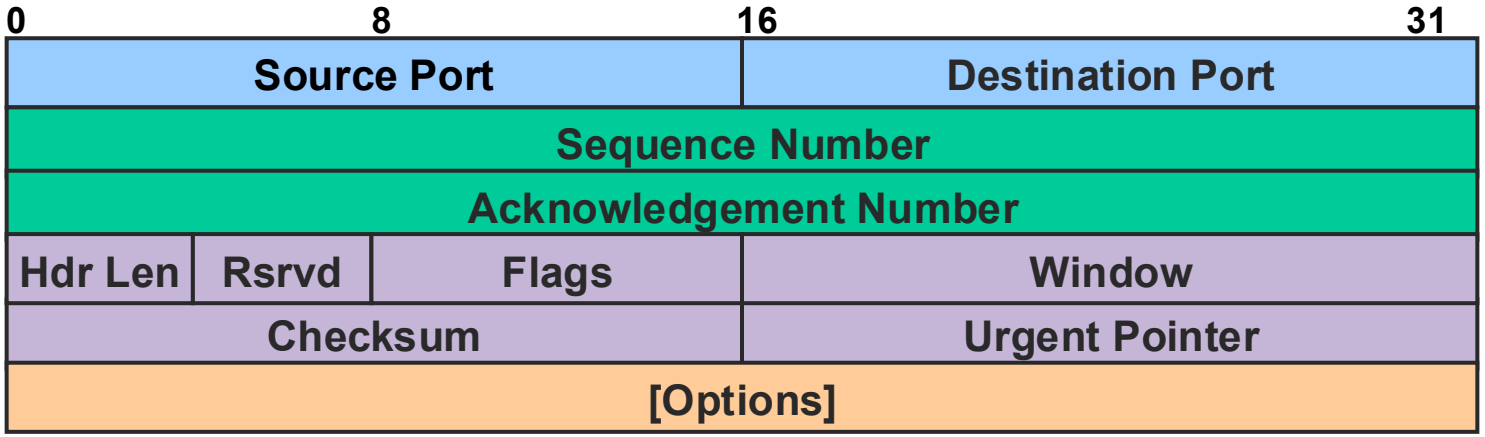


ACKing and Sequence Numbers

- Upon receipt of packet, receiver sends an ACK
- If the highest contiguous byte received is Y:
  - ACK acknowledges Y+1
    (even if this has been ACKed before)

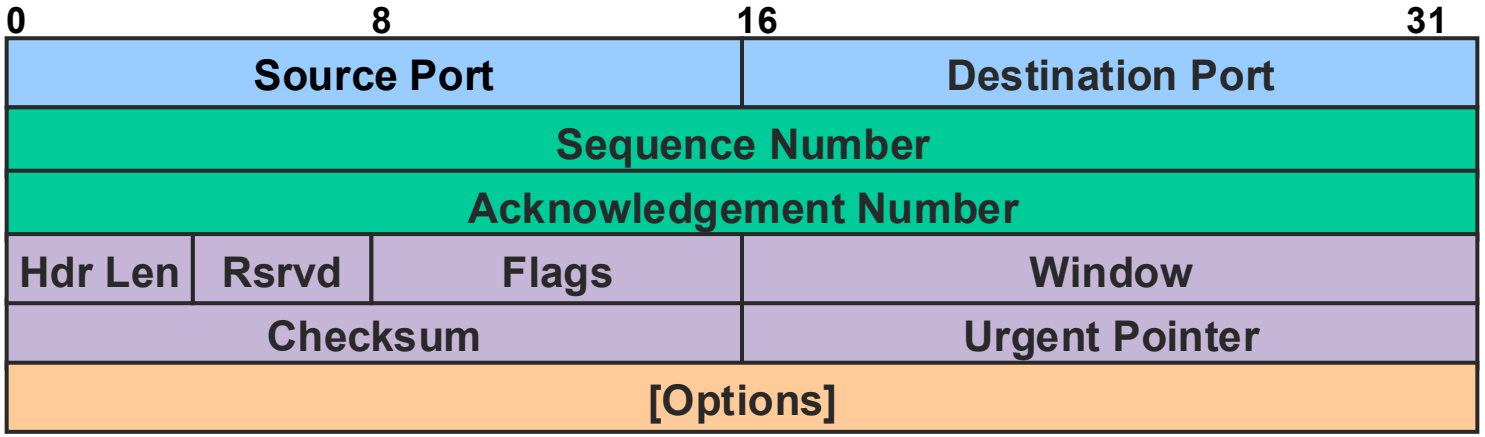


TCP Segment Header Format



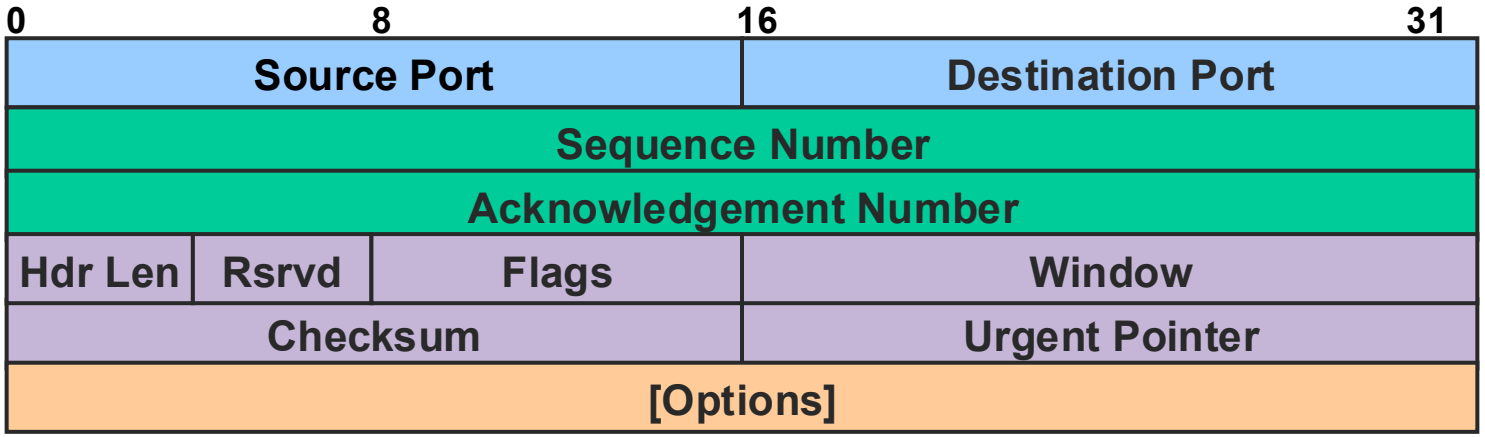
- Header length / data offset (4 bits)
  - Expressed in 4-byte words
  - Minimum 5 words (20 bytes)
  - Offset to first data byte

TCP Segment Header Format



- Reserved (4 bits)
  - Must be 0

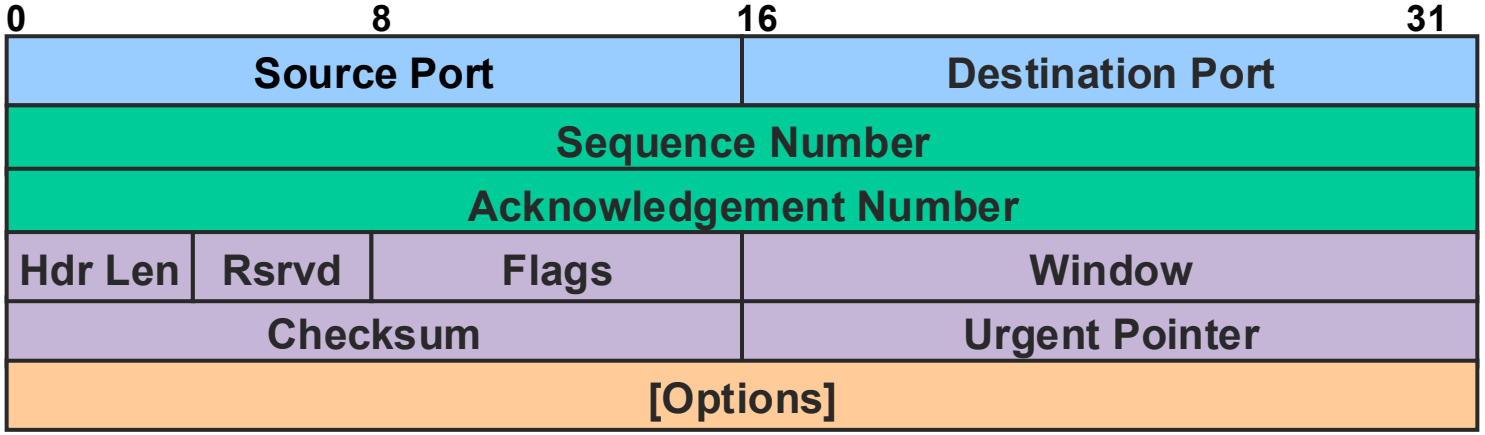
TCP Segment Header Format



- Flag field: eight 1-bit flags

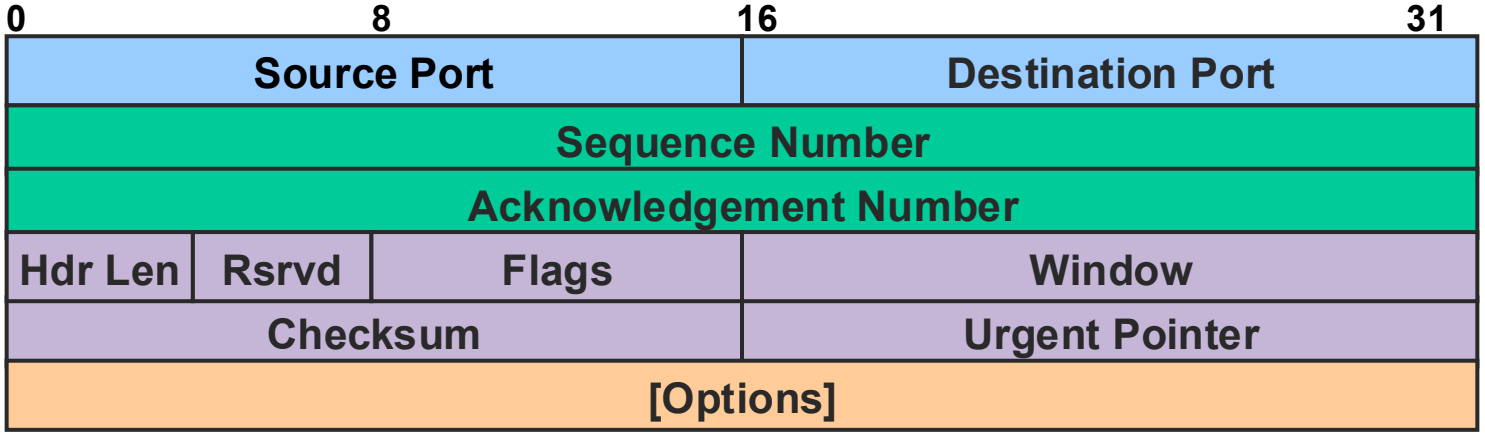
- CWR: Congestion Window Reduced
- ECE: ECN-Echo
- URG: Contains urgent data
- ACK: Indicates ACK number is valid
- PSH: Push buffered data
- RST: Reset the connection
- SYN: Synchronize for setup
- FIN: No more data from me

TCP Segment Header Format

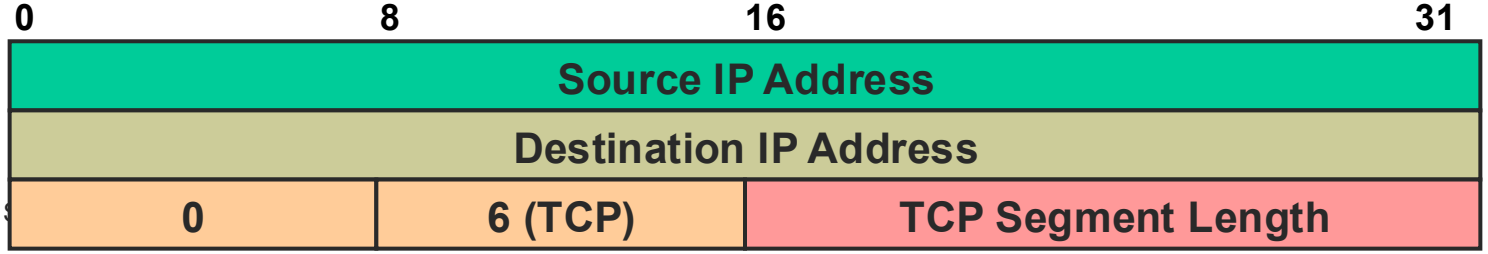


- 16-bit advertised window
  - Space remaining in receive window
  - Used for flow control (more on this later)

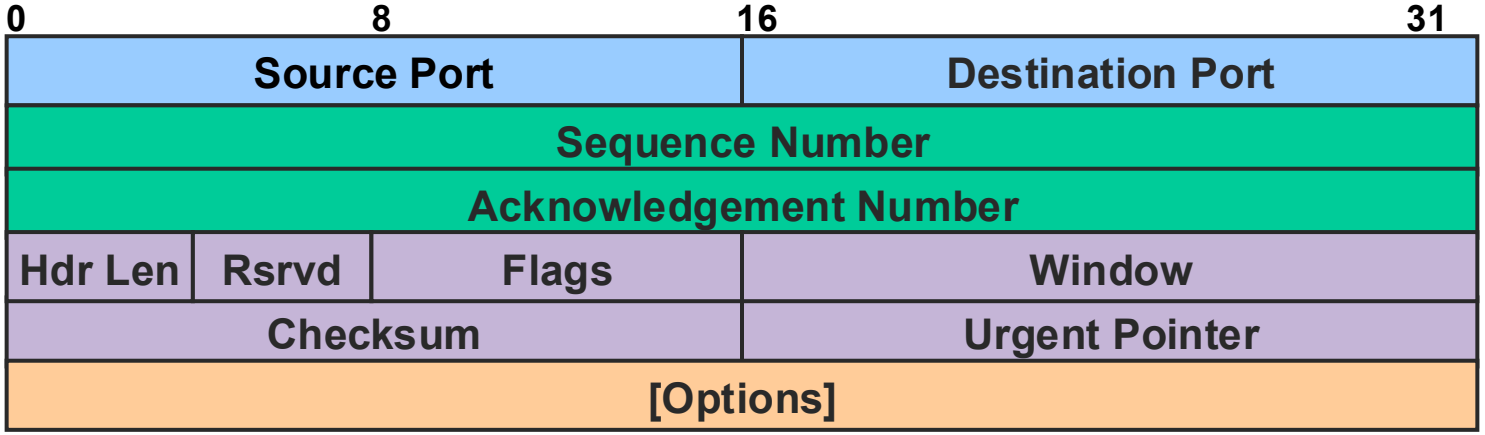
TCP Segment Header Format



- 16-bit checksum
  - Uses IP checksum algorithm
  - Computed on TCP header, TCP data & pseudo IP header



TCP Segment Header Format



- 16-bit urgent data pointer
  - Valid if URG flag is set
  - Offset from the sequence number indicating the last urgent data byte

TCP Options

- Negotiate maximum segment size (MSS)
  - Each host suggests a value
  - Minimum of two values is chosen
  - Prevents IP fragmentation over first and last hops
- Packet timestamp
  - Allows RTT calculation for retransmitted packets
  - Extends sequence number space for identification of stray packets
- Negotiate advertised window granularity
  - Allows larger windows (window scaling)
  - Good for routes with large bandwidth-delay products

TCP Bit Allocation Limitations

- Sequence numbers vs. packet lifetime
  - Assume that packets live less than 120 seconds
  - Can we send 2^{32} bytes in 120 seconds?
  - Only 286.33 Mbps
- Advertised window vs. bandwidth-delay
  - Only 16 bits for advertised window
  - Cross-country RTT = 100 ms
  - Adequate for only 5.24 Mbps!