Computer Networking

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

TCP

Transmission Control Protocol

Recap

Past Topics

- Overview of networking and layered architecture
- Wireshark packet sniffer and Scapy packet manipulation
- Wired LAN, Wireless LANs, VLANs
- IPv4, IPv6 ARP, ICMP
- Transport Layer: UDP, Sockets
- Application Layer: HTTP, DHCP

Today's Topics

- Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP)

- Standard transport layer protocols on the Internet
  - User Datagram Protocol (UDP) - Appropriate for *loss-tolerant* and/or *delay-intolerant* applications
  - Transmission Control Protocol (TCP) - Appropriate for *loss-intolerant* and/or *delay-tolerant* applications
- TCP uses different techniques to provide these services:
  - *Sequence numbers* to distinguish between packets and provide in-order delivery
  - *Timeout/retransmission* in order to avoid packet loss
  - *Checksum* identifies bit flips
  - *Sliding window algorithm* to maximize throughput without causing congestion

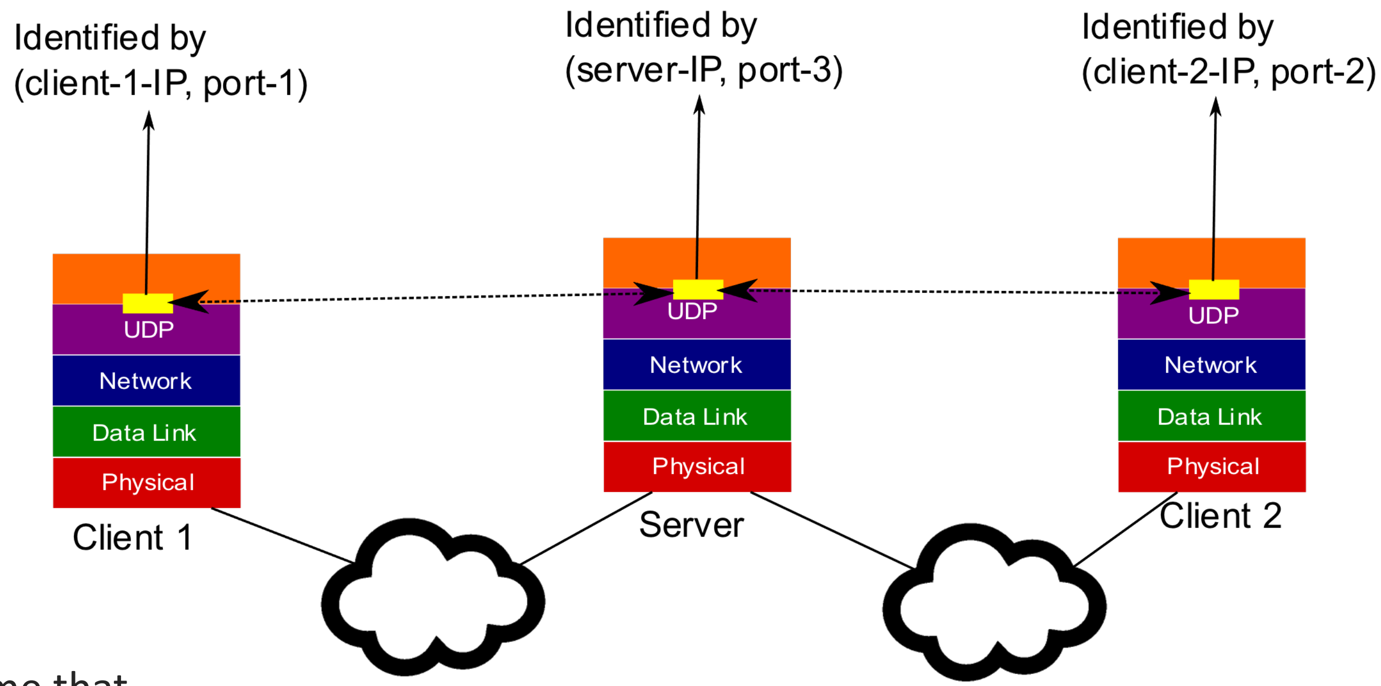
TCP and Applications

- An application that uses TCP follows this scenario:
  - Two ends open a connection (connection-orientation)
  - The sender process writes messages into TCP socket that belongs to that process
  - The receiver process reads messages from the TCP socket that belongs to that process
  - TCP ensures that the transfer of messages is safe and sound
  - Two ends tear down the connection

TCP Sockets

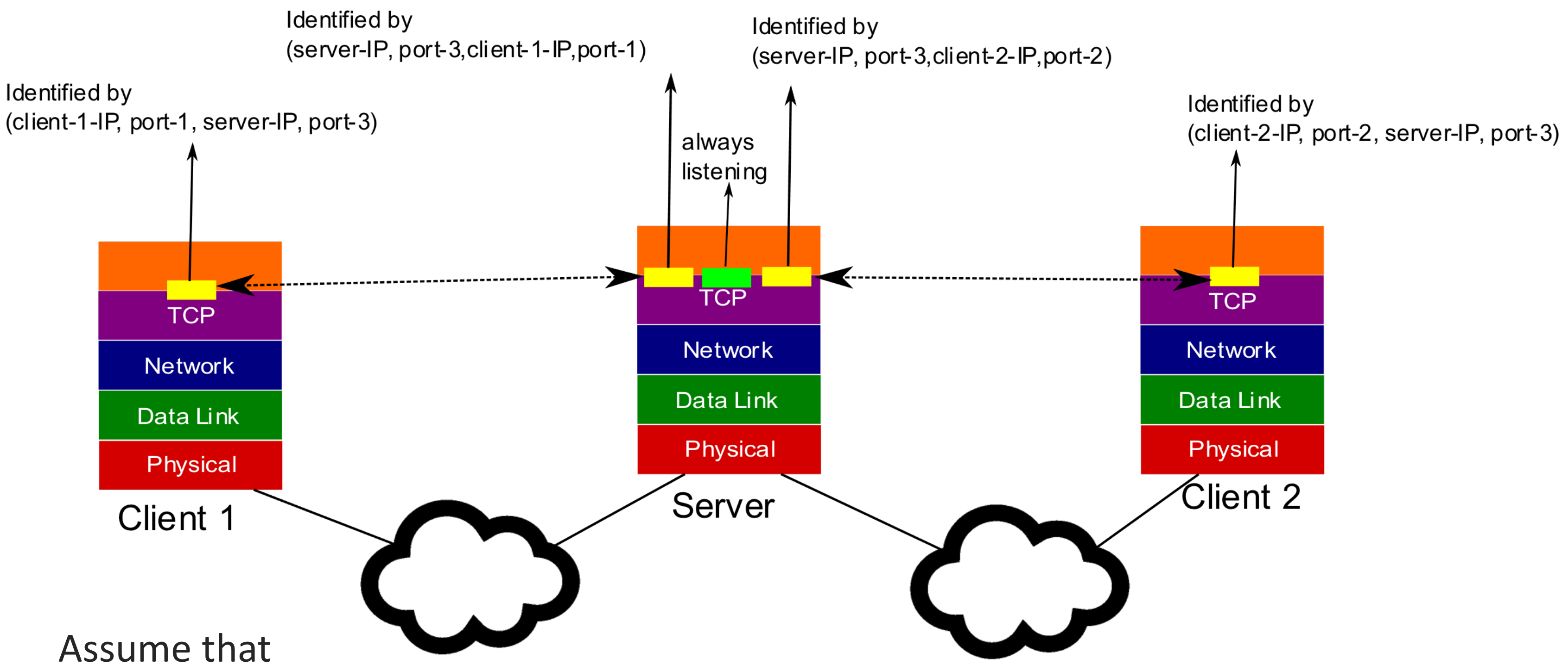
- TCP sockets are treated differently from UDP sockets within a connection
  - Have a UDP socket? You can use it to communicate (via UDP) with <u>any destination</u>
  - Have a TCP socket? That will be bound to a specific connection with a specific destination
- A TCP server application has an *always-listening* socket that is not connected to any endpoint
  - The listening socket does not carry application data to the server
  - The listening socket purpose is to establish connections between endpoints
- To establish a connection, a new socket is created at the server identified by
  - Source IP address
  - Source port #
  - Destination IP address
  - Destination port #
- This socket is bound to the connection between the endpoints and can send/receive application data

UDP Sockets



- Assume that
  - Client 1 is running the process on “port-1”
  - Client 2 is running the process on “port-2”
  - Server is running the process on “port-3”

TCP Socket Connections

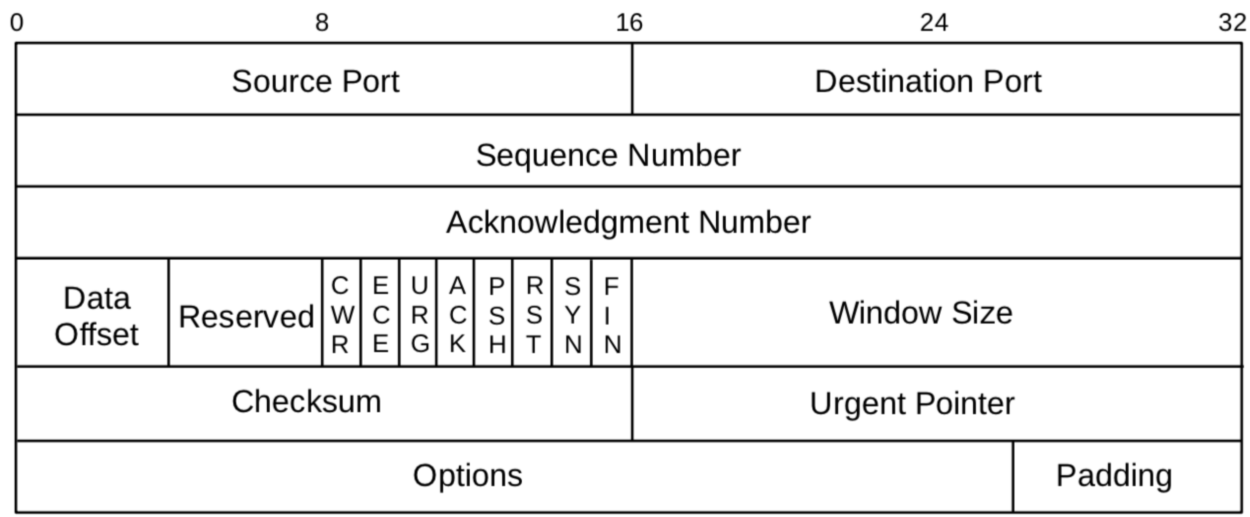


- Assume that
  - Client 1 is running the process on "port-1"
  - Client 2 is running the process on "port-2"
  - Server is running the process on "port-3"

Half-duplex –vs- Full-duplex Connections

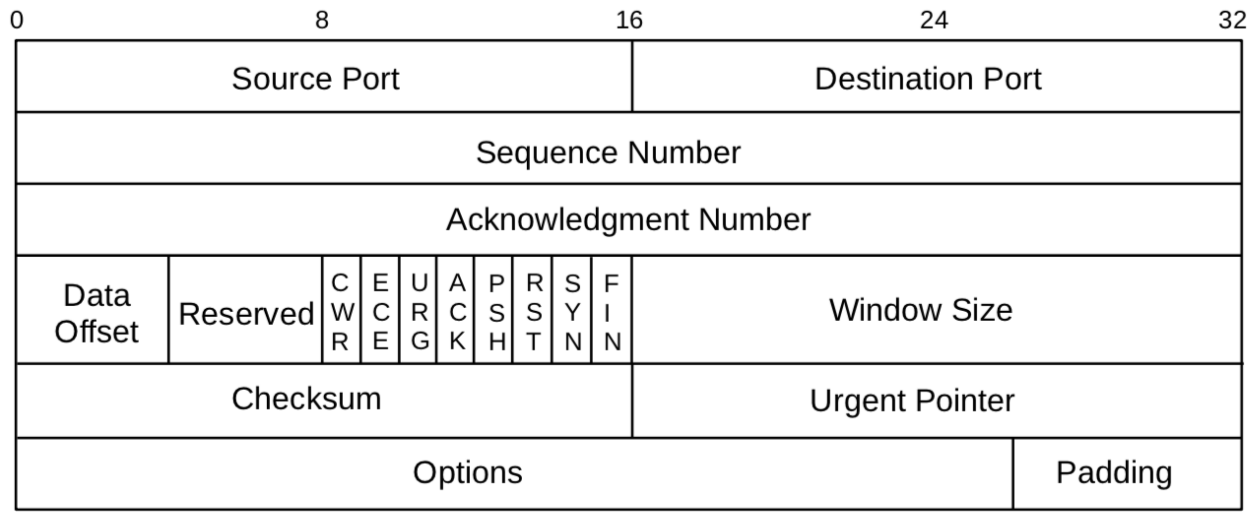
- **Half-duplex connection**
  - One side is the sender of application layer messages and the other side is the receiver of the application layer messages
- **Full-duplex connection**
  - Both sides can be the sender of application layer messages and both sides can be the receiver of the application layer messages, simultaneously
- TCP facilitates *full-duplex connections*, therefore in a single TCP connection
  - Host1 can send message to Host2
  - Host2 receives the message from Host1
  - Host2 responds to Host1 with its own message
  - Host1 receives the message from Host2

TCP Header Format



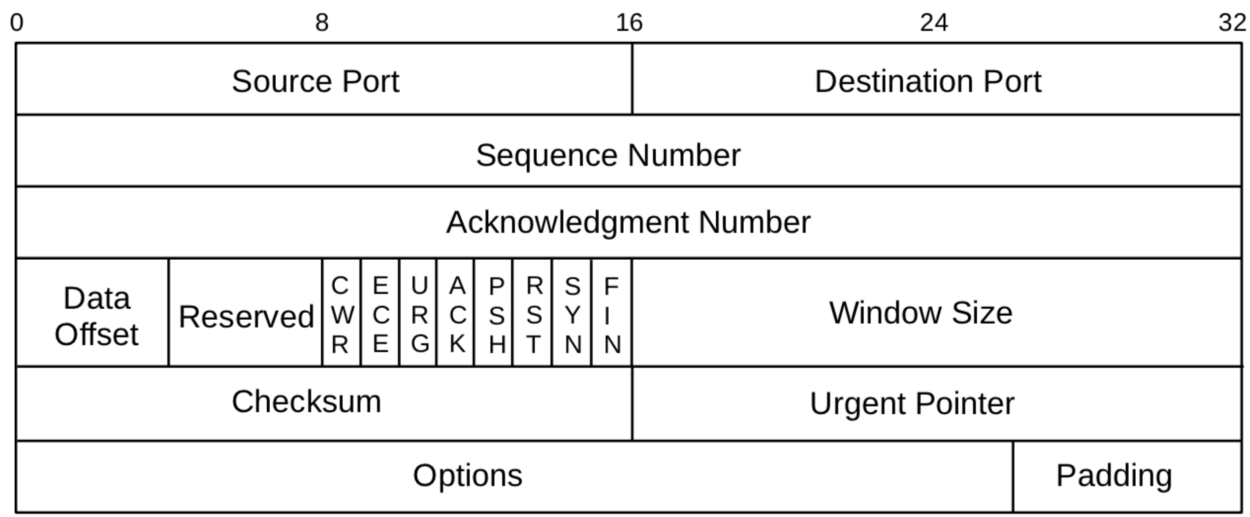
- **Source port** (16 bits): Port number assigned to sender process
- **Destination port** (16 bits): Port number assigned to destination process
  - Like UDP ports, 2^{16} = 65536 port numbers available
  - Port numbers ≤ 1024 are privileged

TCP Header Format



- TCP is a byte stream protocol
  - It treats application messages as a stream of bytes and numbers them.
- TCP establishes a **full-duplex connection**
  - While the sender is transmitting bytes, it is *also* acknowledging the receipt of bytes from the other endpoint for reliable data transfer

TCP Header Format

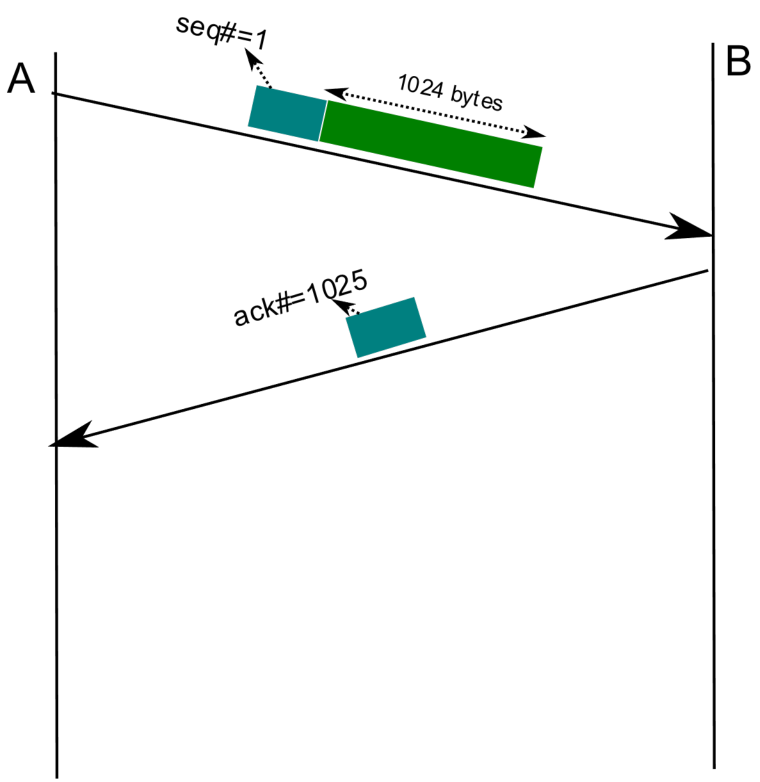


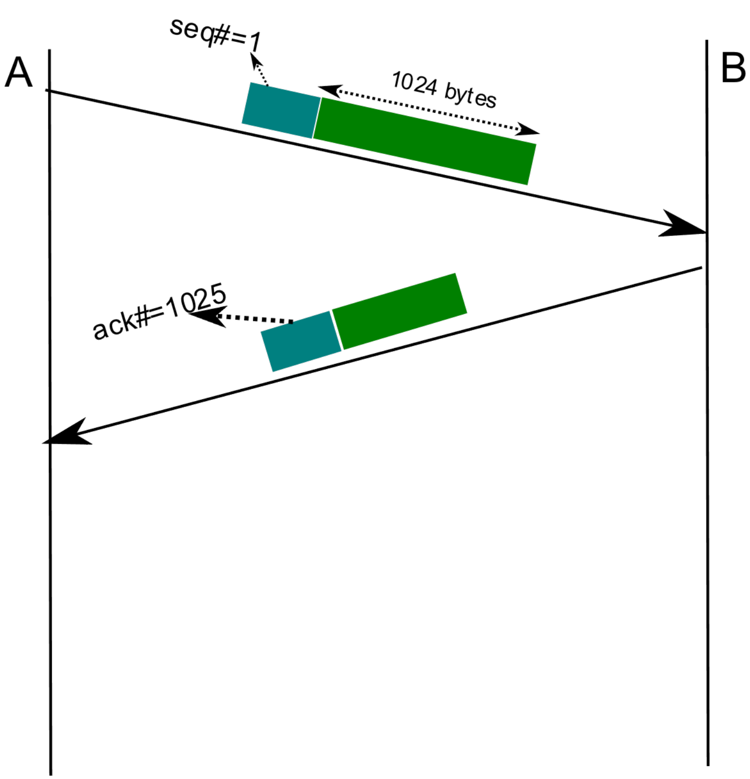
- Sequence number and acknowledgement number fields are used for numbering the bytes of data
  - **Seq #** (32 bits): The number of the first byte in the payload
  - **Ack #** (32 bits): The number of the next byte expected
    - If Ack# = n, then the sender is asserting that it has received every byte with Seq# ≤ (n – 1) correctly

Example: Sequence and Acknowledgement Numbers

- Assume that *A* wants to send an application message of size 1024 bytes to *B*, using TCP
- Bytes within this message can be numbered from 1 to 1024
- *A* encapsulates application message in a TCP header with Seq# = 1
  - Why? The first byte of TCP payload (application message) is numbered 1.
- When *B* receives this packet without any errors (corruption), it responds back with a TCP packet in which the Ack# is 1025.
  - Why? B has received every byte **through** 1024 correctly
- *Since TCP supports full-duplex connections, B can send its own application message encapsulated in a TCP packet while also acknowledging the receipt of A's packet*

Example: Sequence and Acknowledgement Numbers

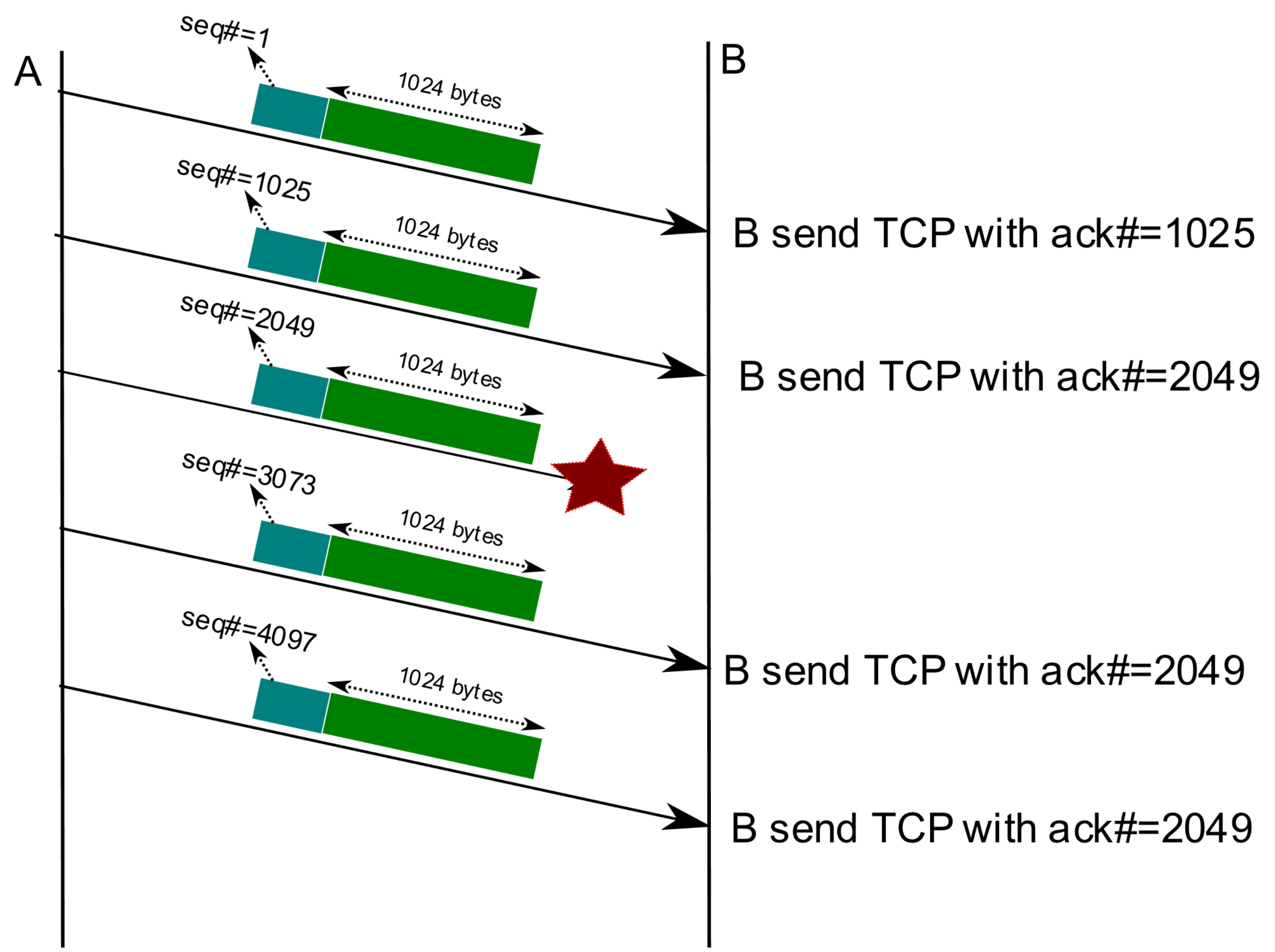




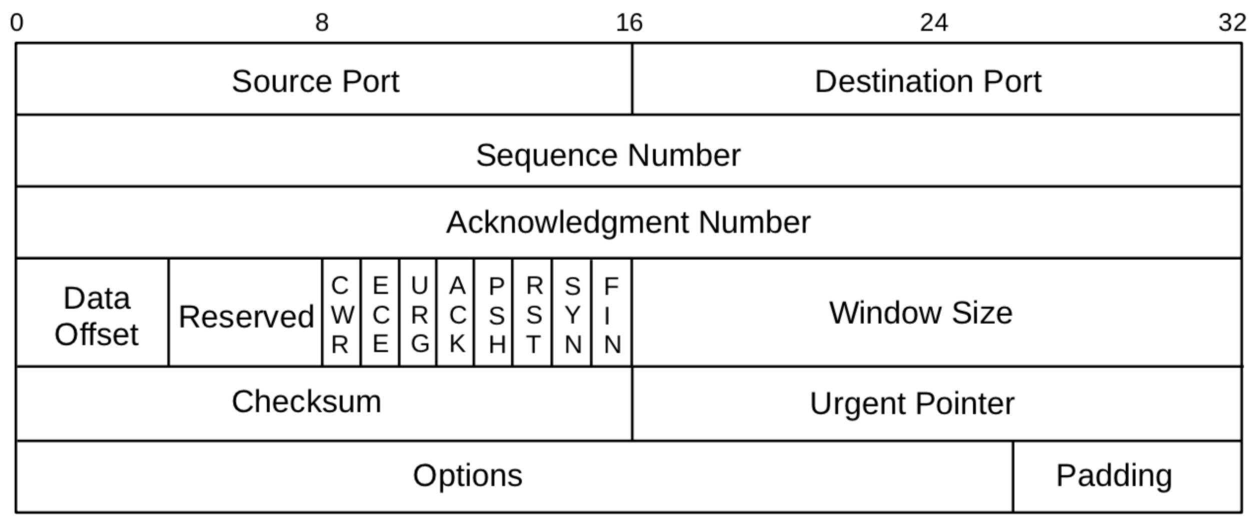
Acknowledgements

- Acknowledgements are used for *reliable data transfer*
- Acknowledgment are *cumulative* in TCP
  - The receiver of the TCP segment with Ack#=n learns that every byte up to n−1 is received by the other end
  - The sender of the TCP segment with Ack#=n is asserting that every byte up to n−1 has been received
- Example:
  - *A* sends 5 TCP packets to *B*
  - 3rd packet is lost in the path, so B doesn't receive it
  - Upon receiving the 4th and 5th packets, *B* still acknowledges the receipt of every byte up to the last byte in the 2nd packet

Example: Sequence and Acknowledgement Numbers

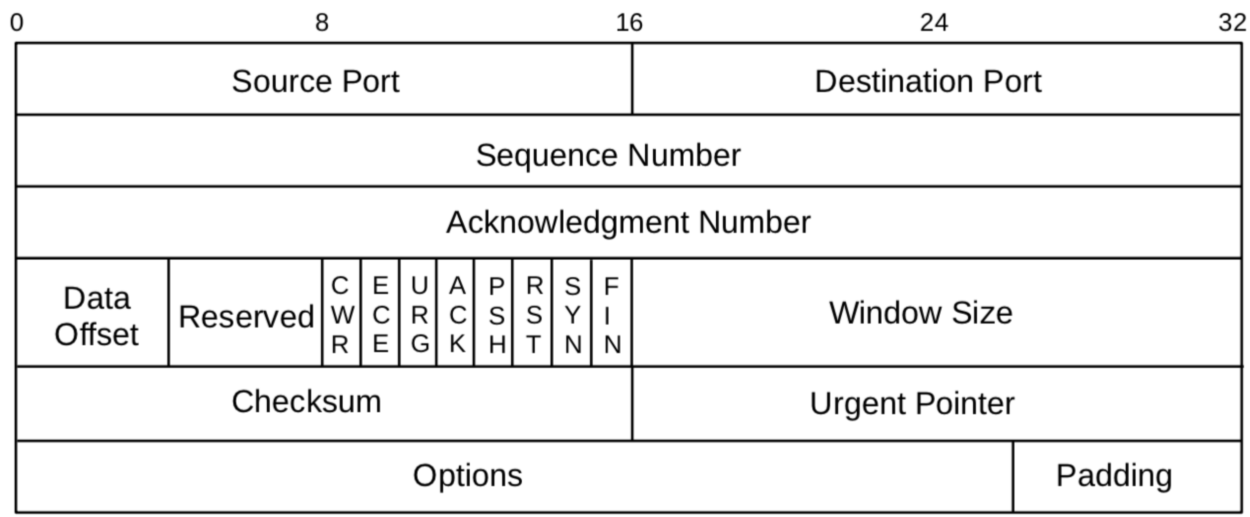


TCP Header Format



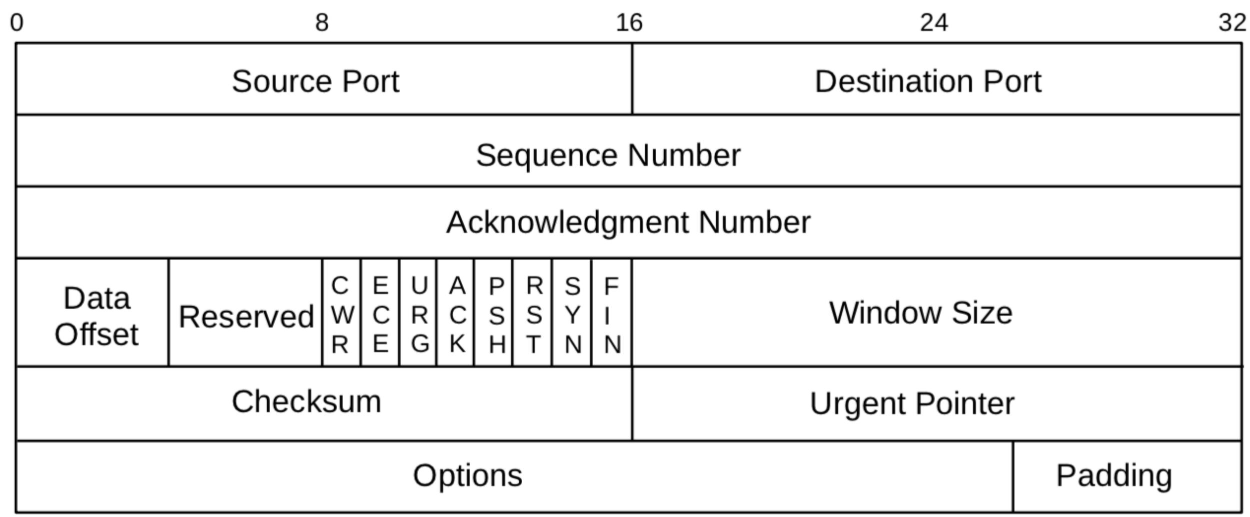
- **Data offset** (4 bits): Specified TCP header size
  - Value is number of 32-bit words (similar to IP IHL header)
- TCP header is variable length (due to options)
  - Default value (no options) is **5** (Five 32-bit words)

TCP Header Format



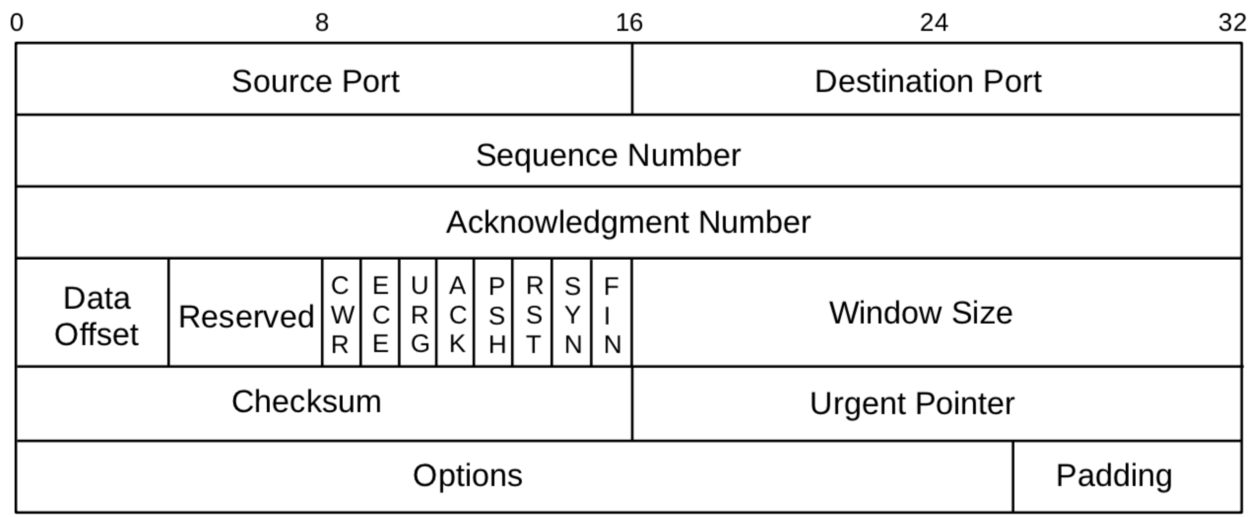
- Flags:
  - **SYN** bit: used in connection establishment (“synchronize”)
    - If set, the TCP packet is called SYN packet
  - **FIN** bit: used to close the connection (“final”)
    - If set, the TCP packet is called FIN packet

TCP Header Format



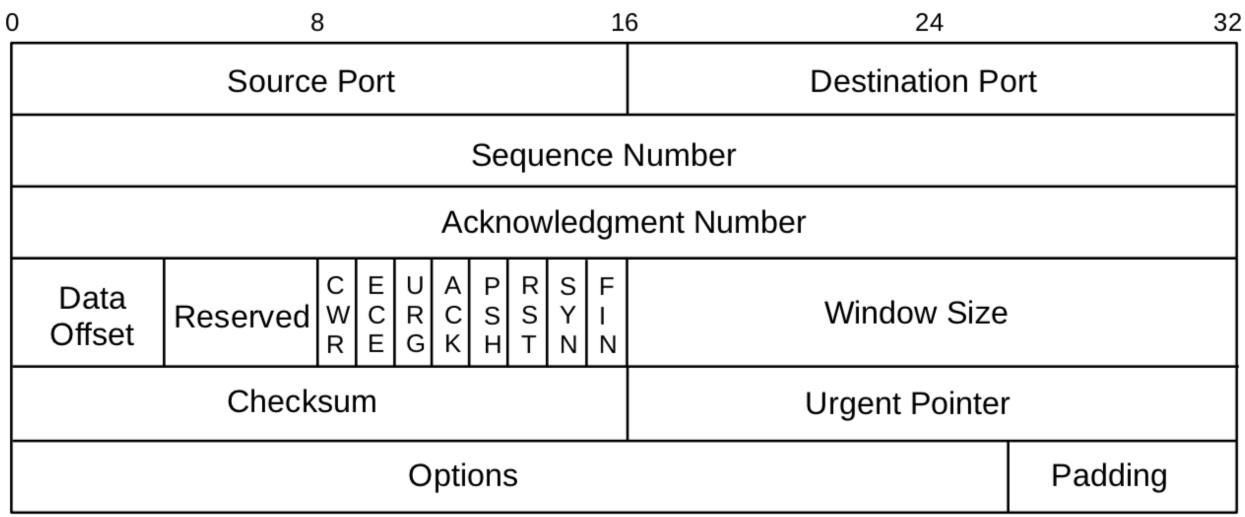
- Flags:
  - **RST** bit: used to reset the connection
    - Used if an error occurs (e.g., the port is closed)
  - **PSH** bit: used to mark packets that need to be delivered to the upper layer process ASAP, at the receiver side

TCP Header Format



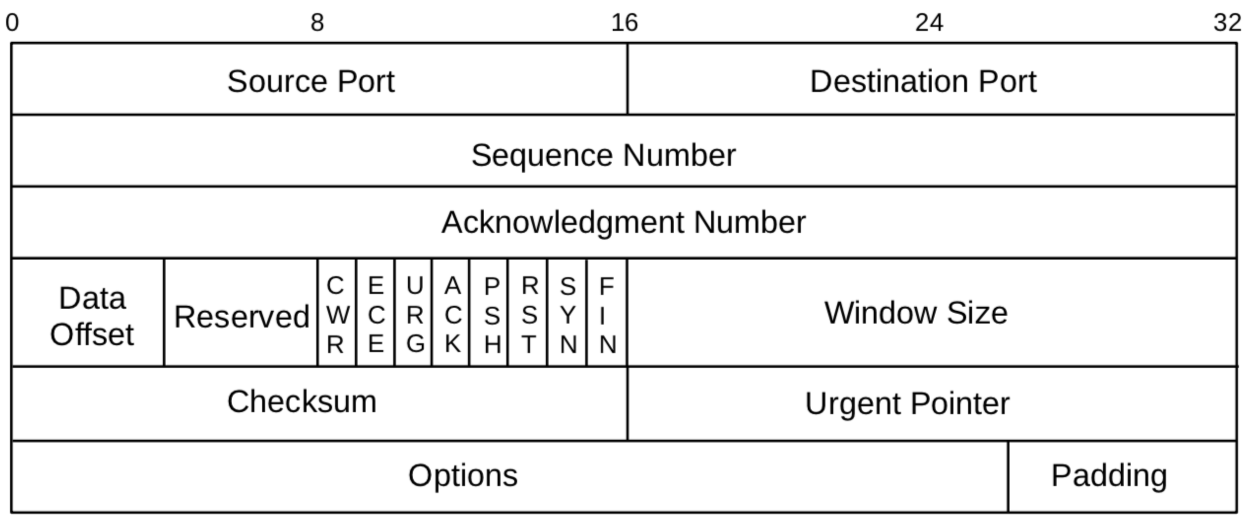
- Flags:
  - **ACK** bit: used to validate the acknowledgement number
    - If this flag is *not* set, the value in the acknowledgment number field is invalid
    - Both the Ack# field and this flag should be set for a valid acknowledgement

TCP Header Format



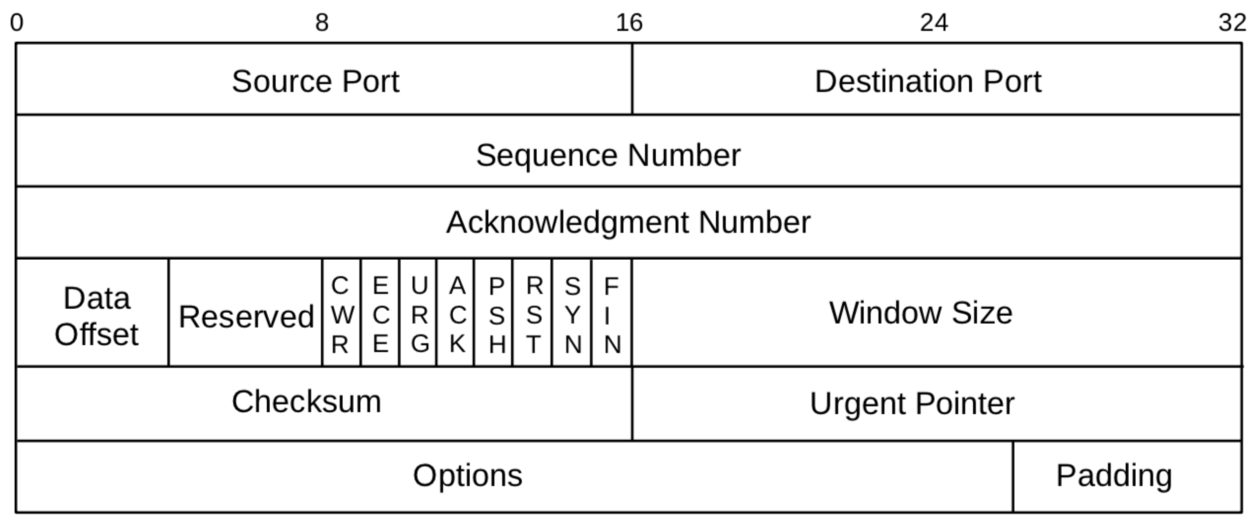
- Flags:
  - **URG** bit: used to mark packets that are urgent to be delivered (high priority)
    - If not set, the value in Urgent Pointer field is invalid
    - Not used in practice frequently

TCP Header Format



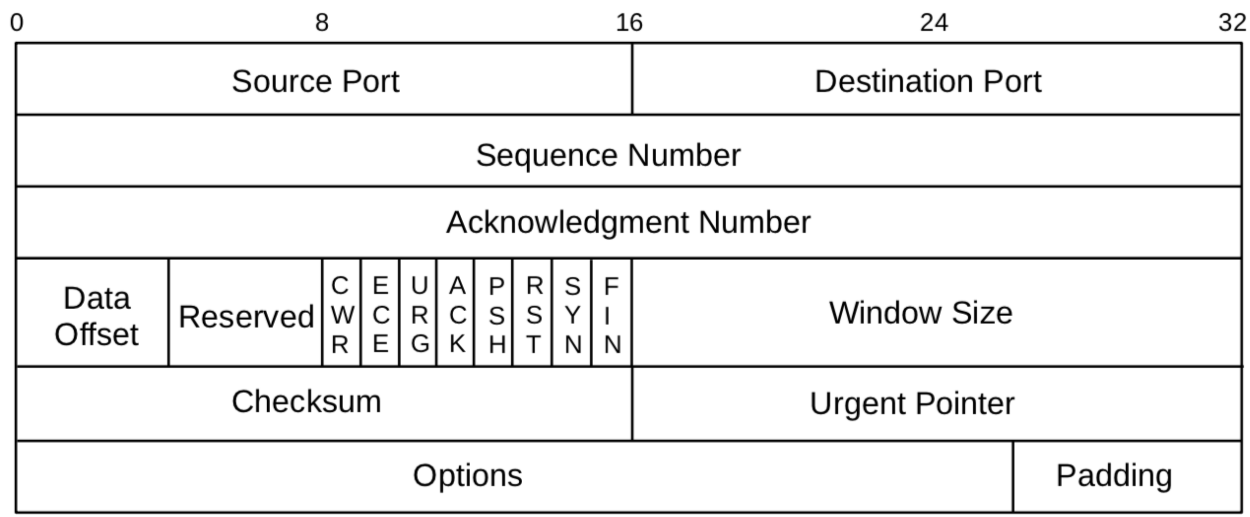
- Flags:
  - **CWR** bit and **ECE** bit: Used in congestion control
    - Explicitly notifies the receiver about congestion in the path
    - Receiver can take measures for congestion control (i.e. slow down)

TCP Header Format



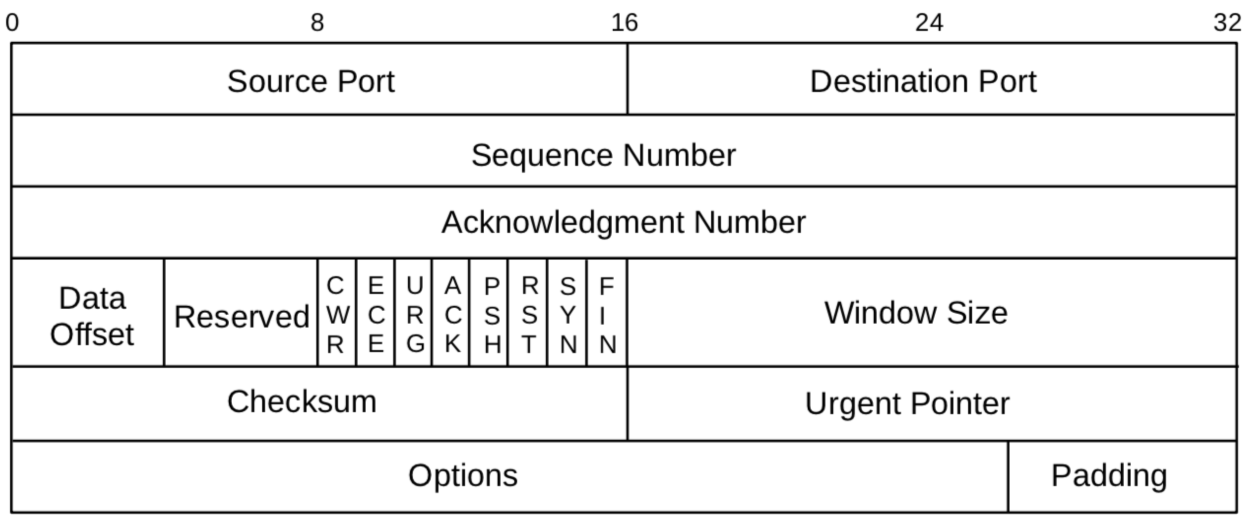
- **Window size** (16 bits): used to report to the other endpoint the available receiving buffer in the host
  - Used in flow control (to avoid overrunning receiver)

TCP Header Format



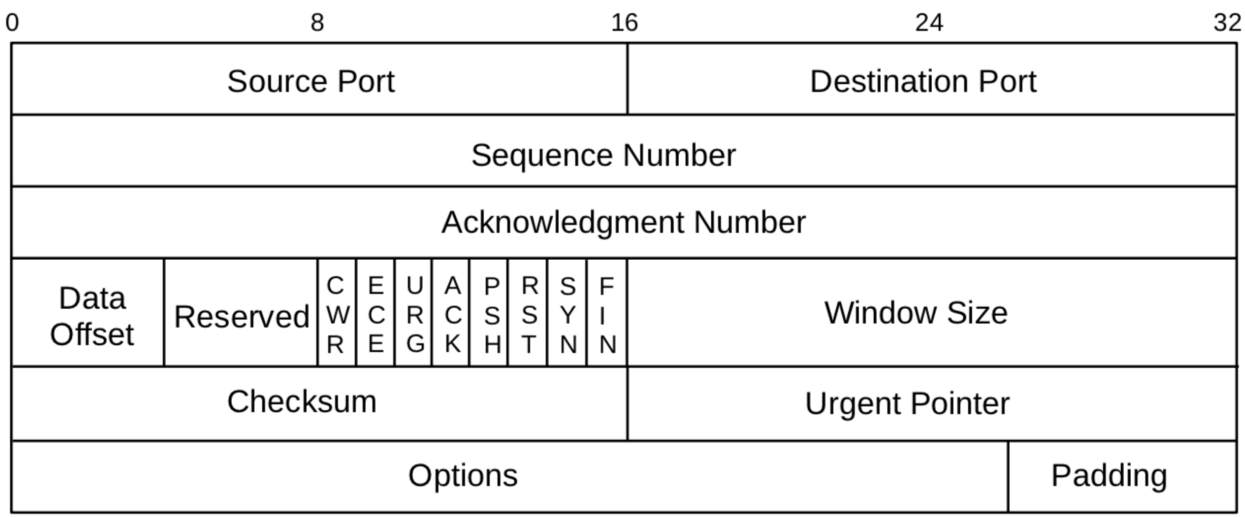
- **Checksum** (16 bits): used to identify bit flips
  - The same algorithm used in UDP and IPv4 header

TCP Header Format



- **Urgent pointer** (16 bits): refers to a specific byte *b* in the payload, so that the receiver should consider delivering the stream of bytes following *b* as soon as possible, rather than buffering them

TCP Header Format



- **Options**: different parameters that can be communicated between the two endpoints
  - Usually provided in the connection establishment phase, e.g., the initial sequence number, timestamps, etc.

Closing Thoughts

Recap

- Today we discussed
  - TCP service model
  - TCP sockets
  - TCP header fields

Next Class

- More TCP

Class Activity

CA.15 – TCP & Wireshark

Due tonight at 11:59pm

Presentation Proposal

Due Nov 4th