COMP2001J Computer Networks

Lecture 11 – Transport Layer
(TCP&UDP, Connection Management)

Dr. Shen WANG (王燊)

shen.wang@ucd.ie

Warm-up Questions

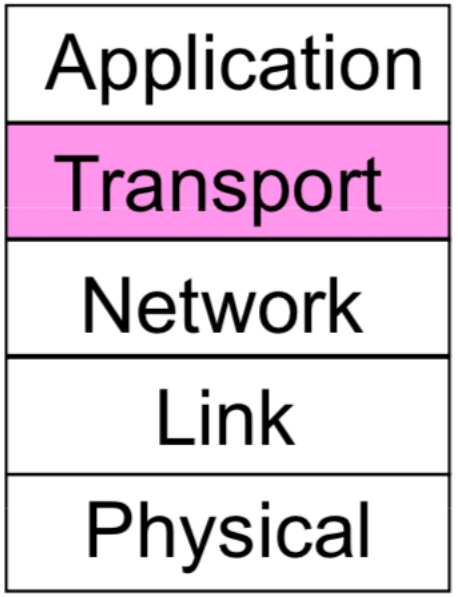
- TCP provides reliable transmission over “unreliable” computer network. What does it mean?

- You probably heard of “3-way handshakes” for TCP connection establishment.
- How does it work exactly?

- Why “4-way handwaves” is needed for releasing TCP connection?

Outline

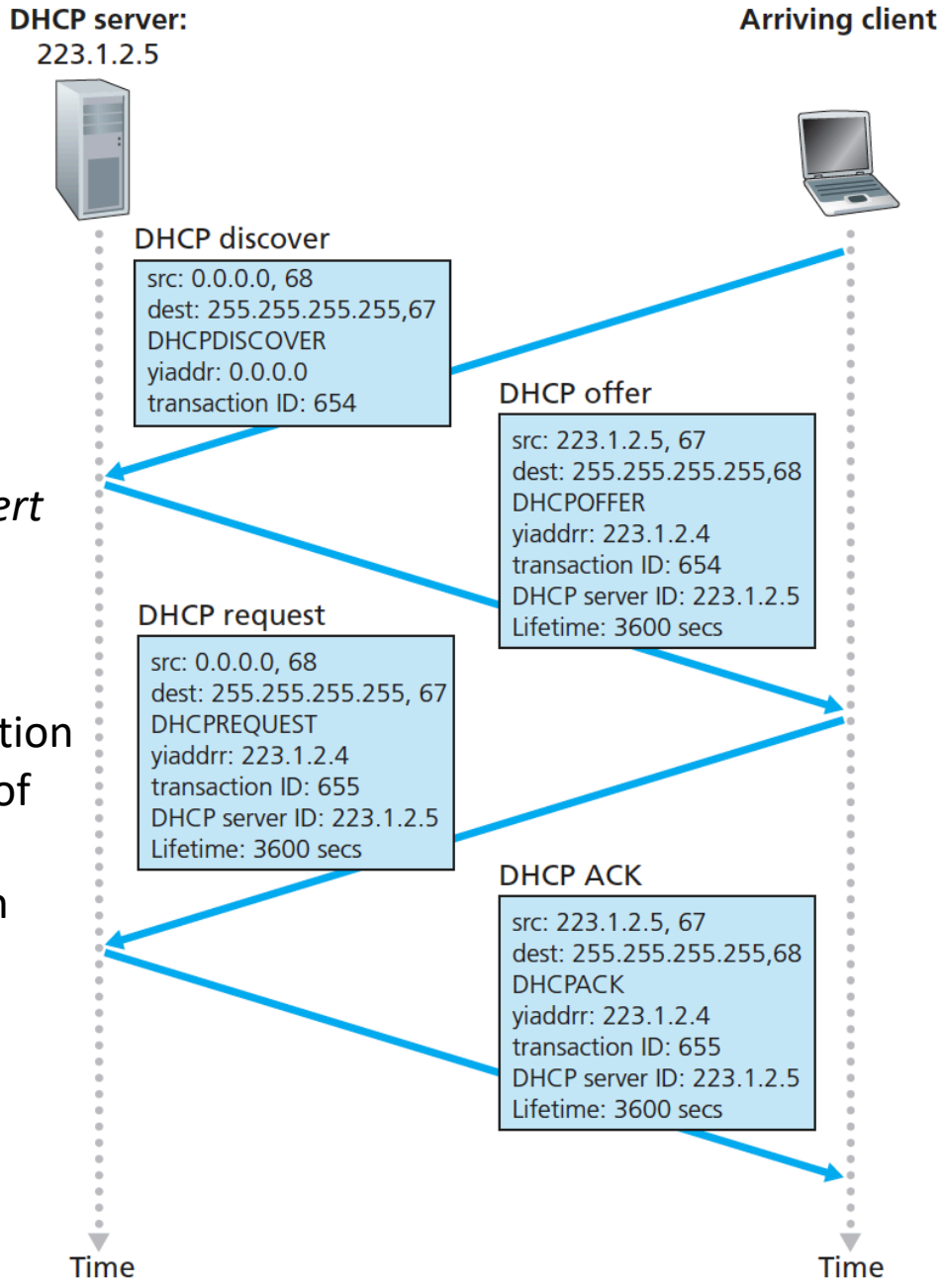
- Background Recap
  - Connection vs. Connectionless
  - Reliable vs. unreliable
- UDP vs. TCP
- TCP Connection Management
  - Establishment
  - Release



About Lab

Q: Why many lines of the output of *tracert* is ** and "request time out"* ?

A: It is blocked for security reasons, like preventing hackers from getting information about open ports and staving off denial of service attacks. When it is blocked, the server doesn't respond at all, resulting in "request timed out" messages.



Connection-Oriented vs. Connectionless

- Service provided by a layer may be kinds of either:
  - Connection-oriented, must be set up for ongoing use (and torn down after use), e.g., phone call
  - Connectionless, messages are handled separately, e.g., postal delivery

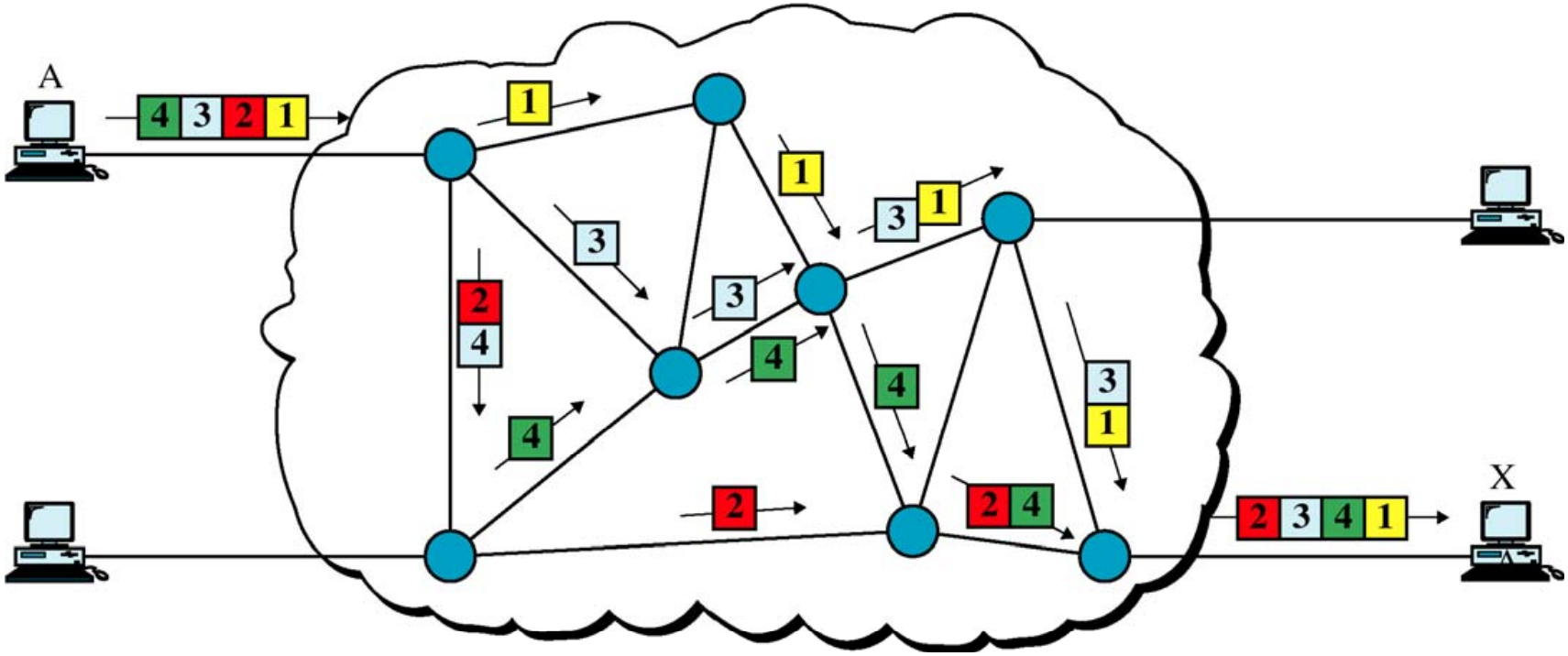
	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Movie download
	Unreliable connection	Voice over IP
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Text messaging
	Request-reply	Database query

Recap-Reliable transmission

- Why the underlying computer network infrastructure is unreliable?
  - As it is based on “Packet Switching”
  - To share network resources with high reliability and low cost

- Review Lecture 2: page 50 -> page 74:
  - Compare “Circuit Switching” and “Packet Switching”
  - In addition, compare “Switching” and “Multiplexing”

Datagram Packet Switching



Packet switching network is unreliable

- Missing packets
  - Send “1, 2, 3, 4” -> Receive “1, 2, 4”
- Delayed packets
  - Assume normal delay is 1 minute
  - Send “1, 2, 3, 4” at 9:00am -> Receive “1, 2, 3, 4” at 9:30am
- Corrupted packets
  - Send “1, 2, 3, 4” -> Receive “1, 8, 3, 4”
- Duplicated packets
  - Send “1, 2, 3, 4” -> Receive “1, 1, 2, 3, 3, 3, 4, 4”
- Out-of-sequenced packets
  - Send “1, 2, 3, 4” -> Receive “3, 1, 4, 2”

Missing packets

- Send “1, 2, 3, 4” -> Receive “1, 2, 4”
- Possible reason
  - being dropped or stuck somewhere over the network due to excessive queueing delay
- Possible solutions
  - Improve network performance
  - Sender receives an “acknowledgement” from the receiver, then sends the the next packet.
  - Sender retransmits the packet, if no “acknowledgement” is received within the duration of its timer.

Delayed packets

- Assume normal delay is 1 minute
- Send “1, 2, 3, 4” at 9:00am -> Receive “1, 2, 3, 4” at 9:30am
- Possible reason
  - Network is congested
  - Some events occurred in network
- Possible solutions
  - Improve network performance
  - Sender sets a timer, retransmit when time out

Corrupted packets

- Send “1, 2, 3, 4” -> Receive “1, 8, 3, 4”
- Possible reason
  - Noisy physical medium, e.g. wireless
- Possible solution:
  - Checksum, if not matched after received, drop it and wait for sender’s retransmission

Duplicated packets

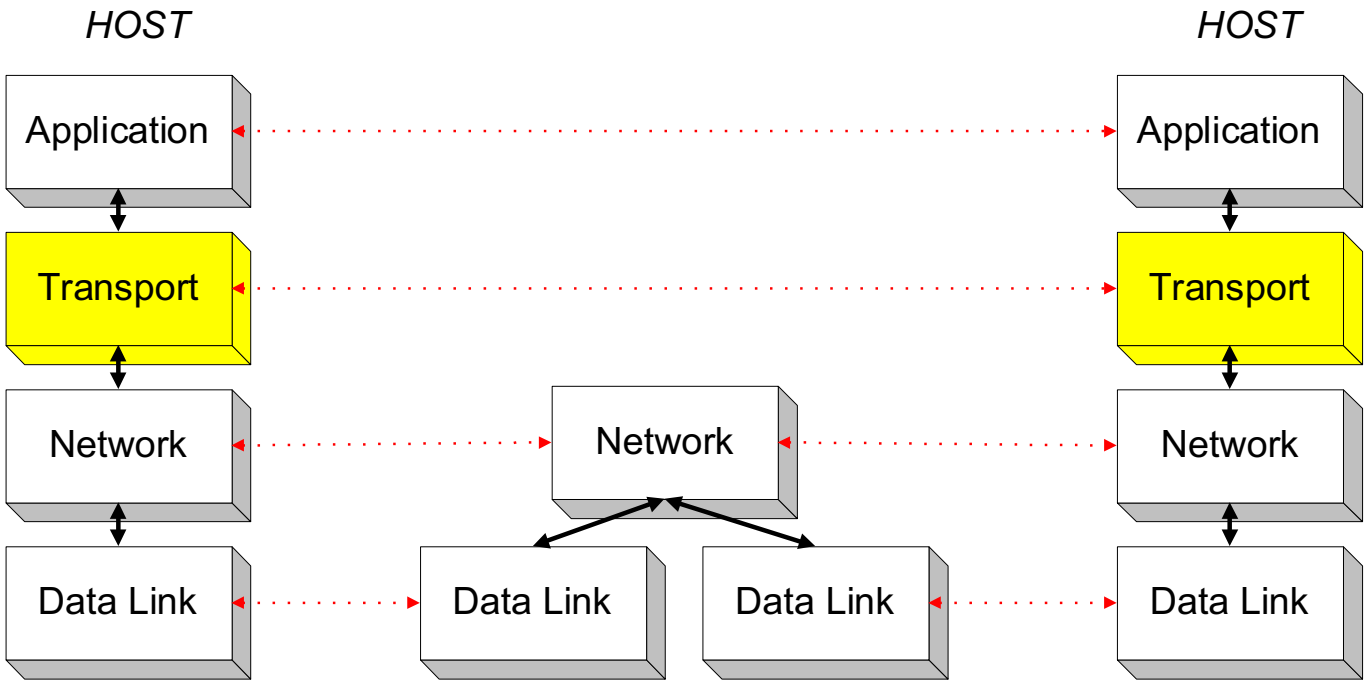
- Send “1, 2, 3, 4” -> Receive “1, 1, 2, 3, 3, 3, 4, 4”
- Possible reason:
  - Too many unnecessary retransmissions
- Possible solutions:
  - Increase to a reasonable duration for the sender’s timer
  - Adding an identification number to allow receiver recognize duplications quickly.

Out-of-sequenced packets

- Send “1, 2, 3, 4” -> Receive “3, 1, 4, 2”
- Possible reason:
  - Different packets travel different routes which have various delays
- Possible solutions:
  - Adding a sequence number, so that receiver can reassemble them in the right order.
- In summary, solutions for achieving reliable transmission over unreliable network
  - Timer, Retransmission, Acknowledgement, Sequence number, Checksum

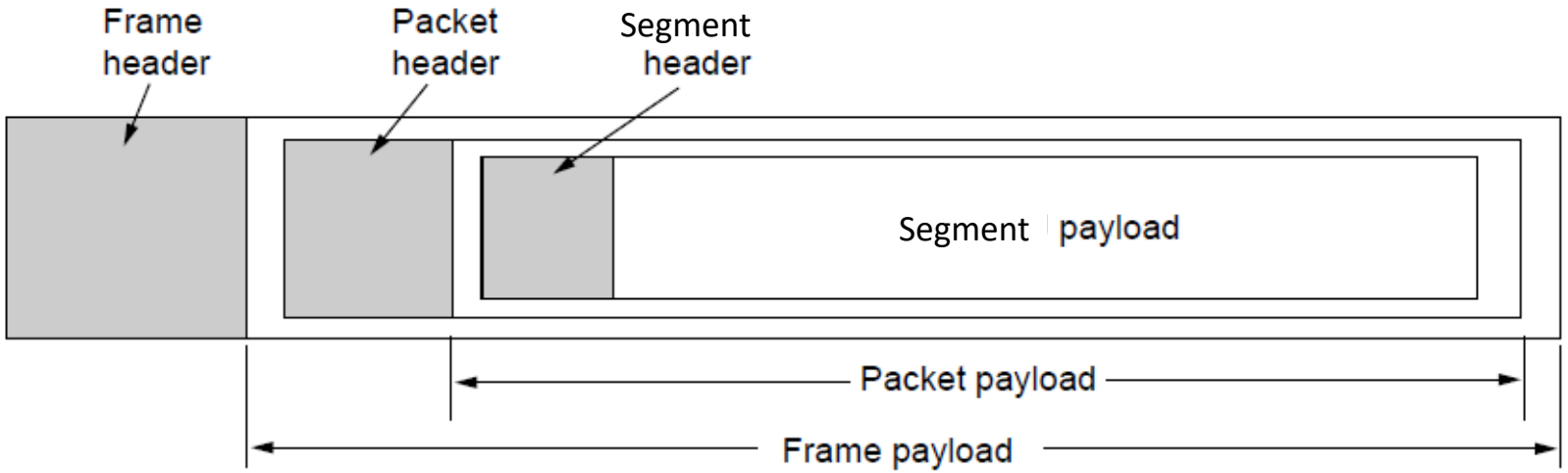
Transport Layer

- Responsible for delivering data across networks with the desired reliability or quality
  - Transport layer protocols are end-to-end protocols
  - They are only implemented at the hosts



Transport Layer

- Transport layer sends segments in packets (in frames)



Transport Protocols

TCP - Transmission Control Protocol

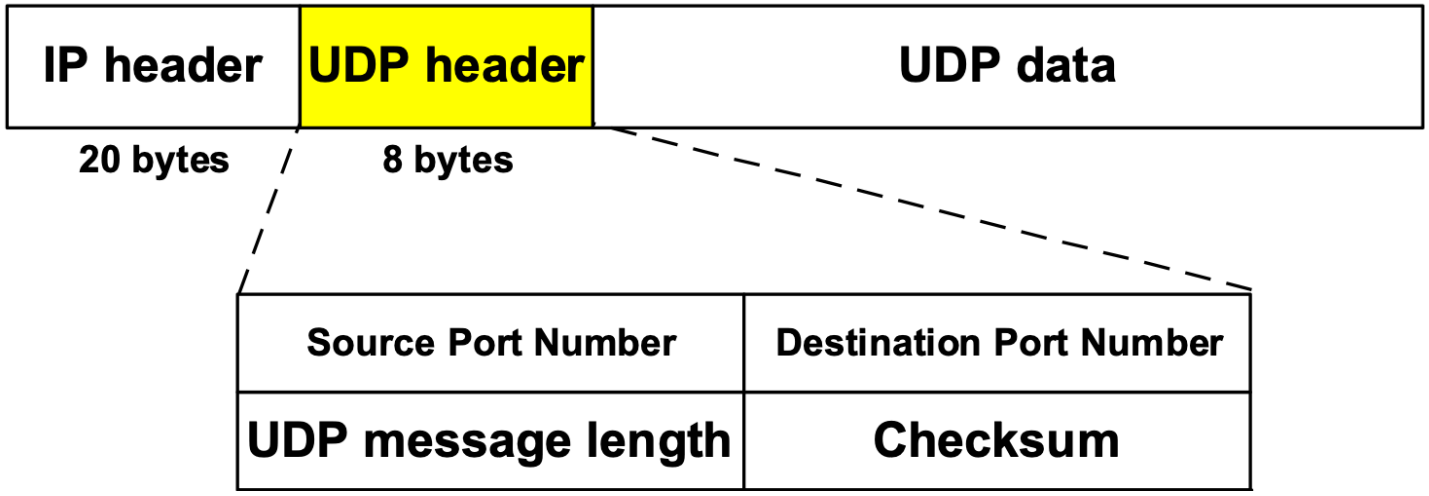
- reliable, connection-oriented
- complex
- unicast
- applications:
  - accurate transmission, but can tolerate delay
  - E.g.: web (HTTP), email (SMTP), file transfer (FTP), terminal (TELNET)

UDP - User Datagram Protocol

- unreliable, connectionless
- simple
- unicast and multicast
- applications:
  - fast transmission, but can tolerate data loss
  - E.g.: real-time video conferencing, network management (SNMP), routing (RIP) ,naming (DNS)

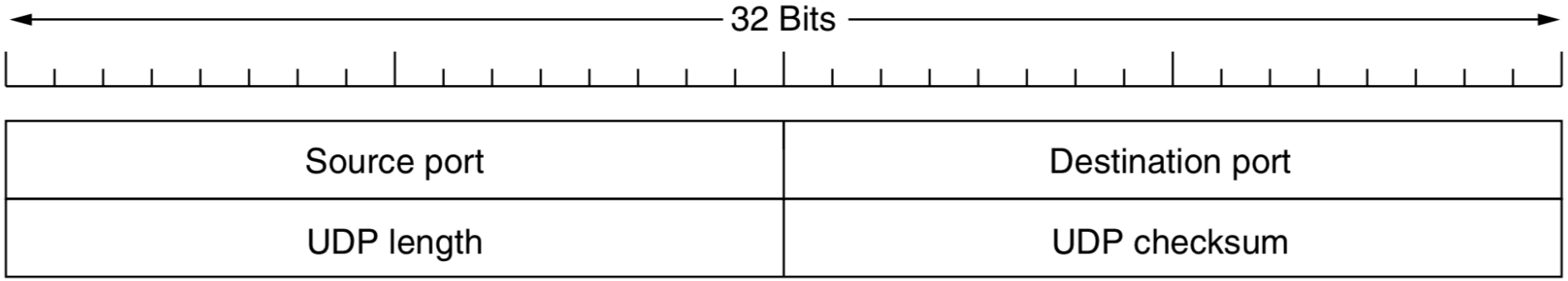
UDP datagram

- UDP length: 8-byte header and the data.
  - Minimum: 8 bytes, only the header
  - Maximum: 65,515 bytes, which is lower than the largest number that will fit in 16 bits because of the size limit on IP packets



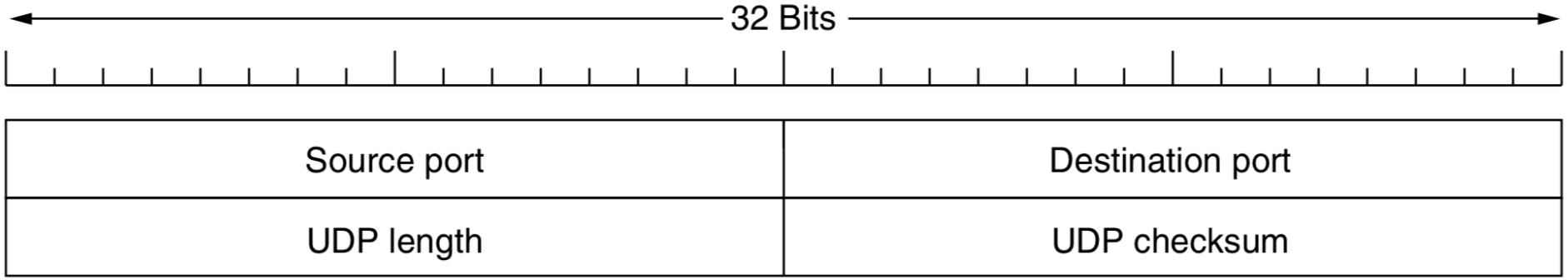
UDP header

- Source/Destination Port:
  - to identify the end-points within the source and destination machines
  - with them, the embedded segment can be delivered to the correct application



UDP header

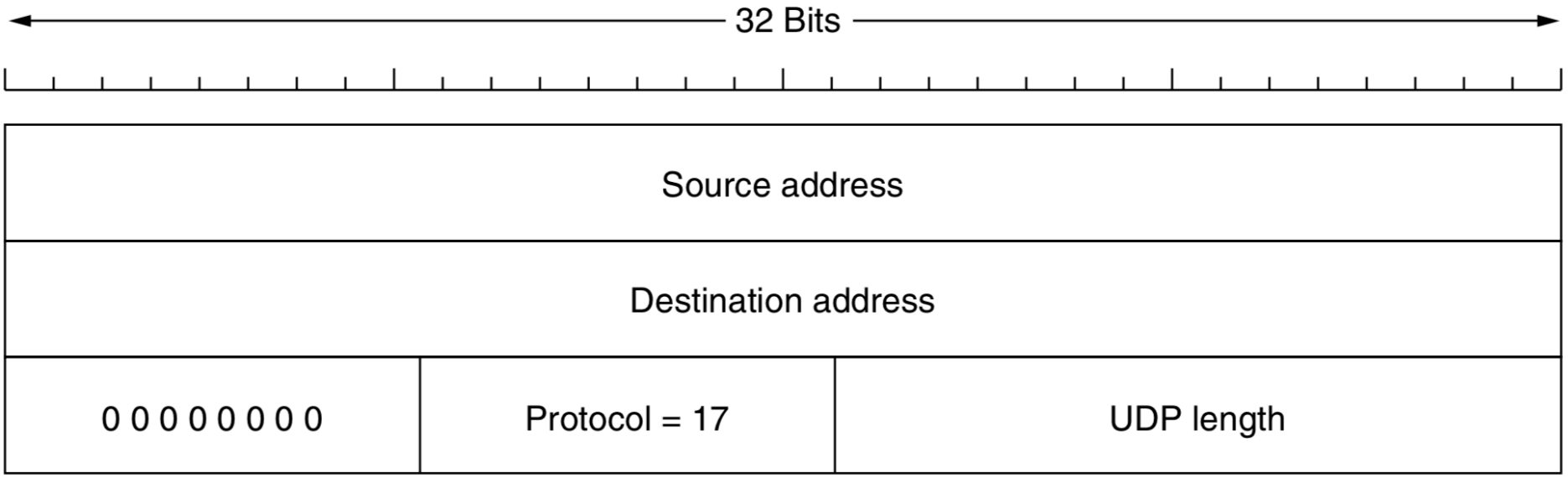
- UDP length: header and data
- UDP checksum: a simplified algorithm to check UDP header, data, and an IPv4 pseudoheader (some parts of IP header including IP address). This is optional for UDP



IPv4 Pseudoheader

1. 32-bit IPv4 addresses of the source and destination machines
2. the protocol number for UDP (17)
3. the byte count for the UDP segment (including the header)

- TCP uses the same pseudoheader for its checksum.



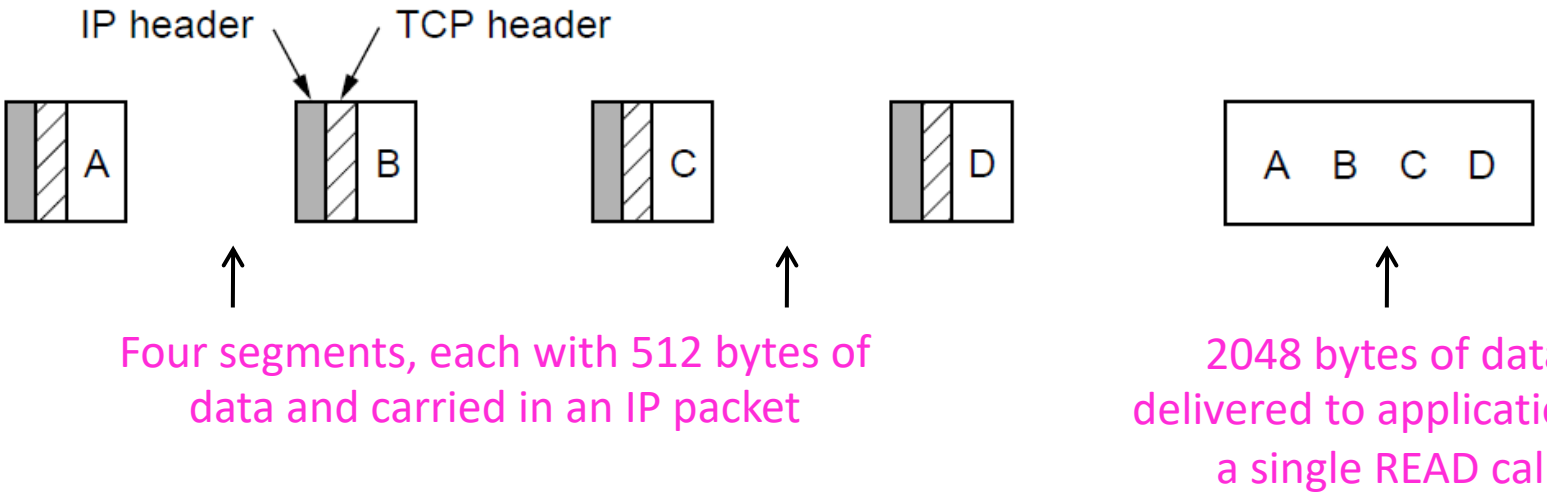
UDP summary

- UDP does do:
  - provide an interface to the IP protocol with the added feature of demultiplexing multiple processes using the ports and optional end-to-end error detection.

- UDP does NOT do:
  - flow control, congestion control, or retransmission upon receipt of a bad segment.
  - all of that is up to the user processes.

The TCP Service Model

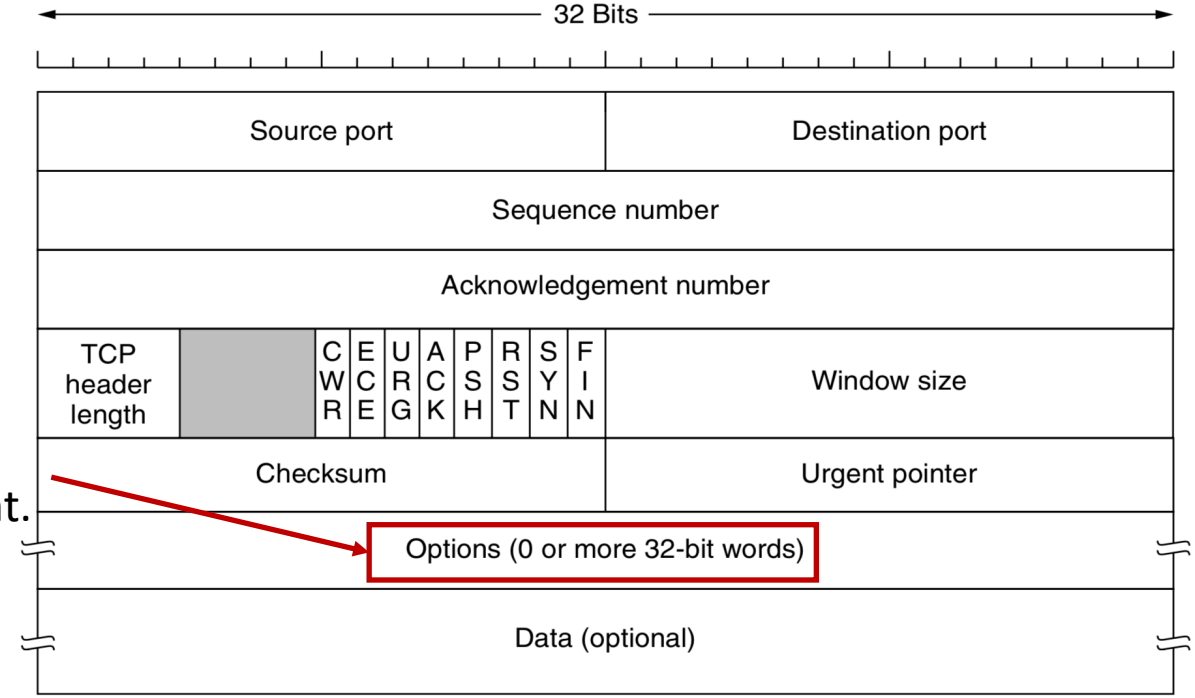
- Applications using TCP see only the byte stream [right] and not the segments [left] sent as separate IP packets



TCP header

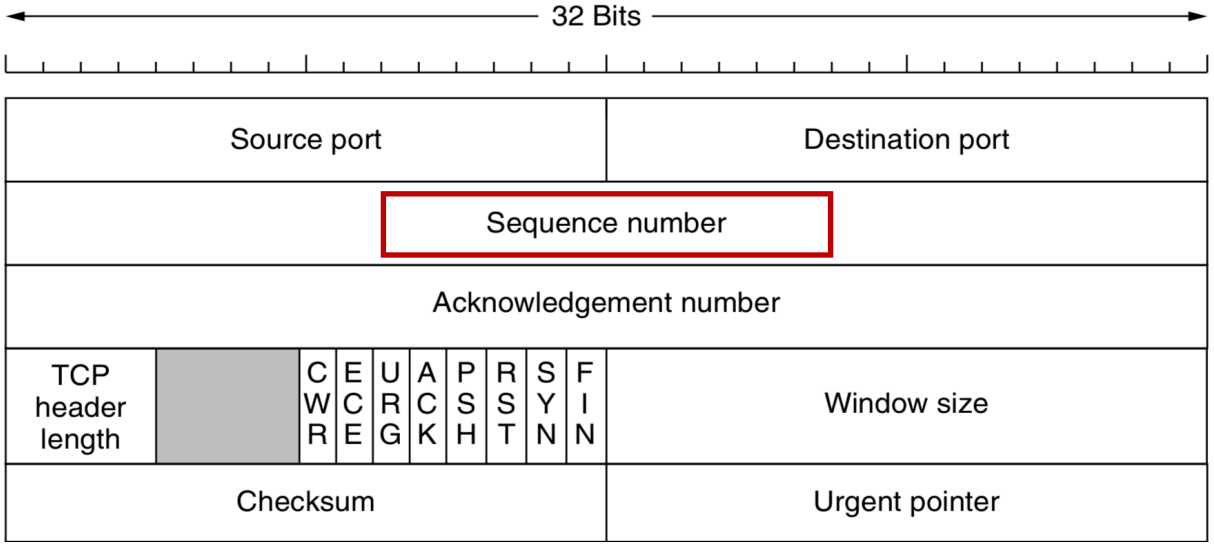
- Every TCP segment begins with a fixed-format, 20-byte header, which may be followed by header options.
- After the options, if any, up to 65,535 − 20 (IP header) − 20 (TCP header) = 65,495 data bytes may follow
- Segments without any data are legal and are commonly used for acknowledgements and control messages.

Often used to define Maximum Segment Size (MSS): the maximum length for data field of TCP segment. Default: 536-byte



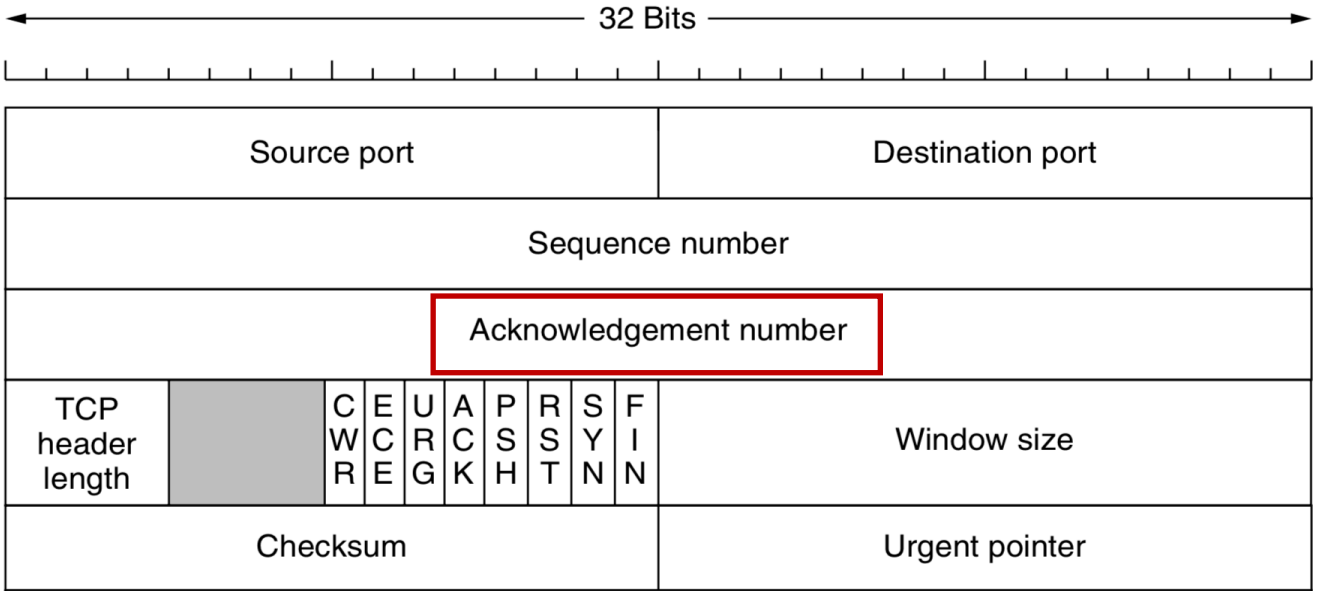
TCP header

- Sequence number:
  - The first byte of a TCP segment to send
  - Every byte of data is numbered in a TCP byte stream
  - It can also be used to determine if packets have been lost over the network.



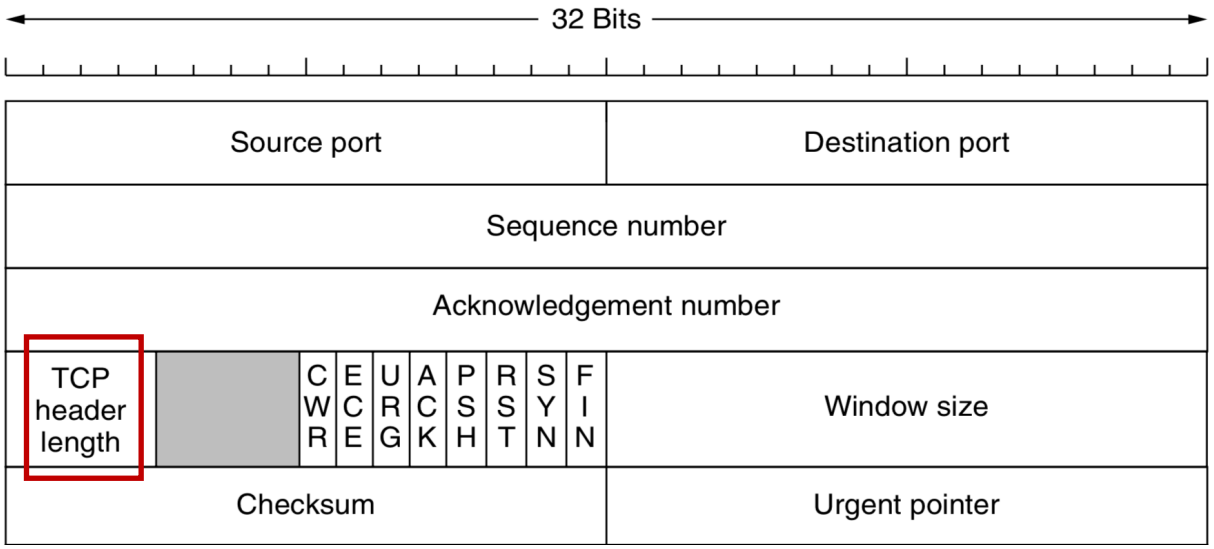
TCP header

- Acknowledgement number:
  - It specifies the next in-order byte expected, not the last byte correctly received.
  - It is a **cumulative acknowledgement** because it summarizes the continuously received data with a single number.



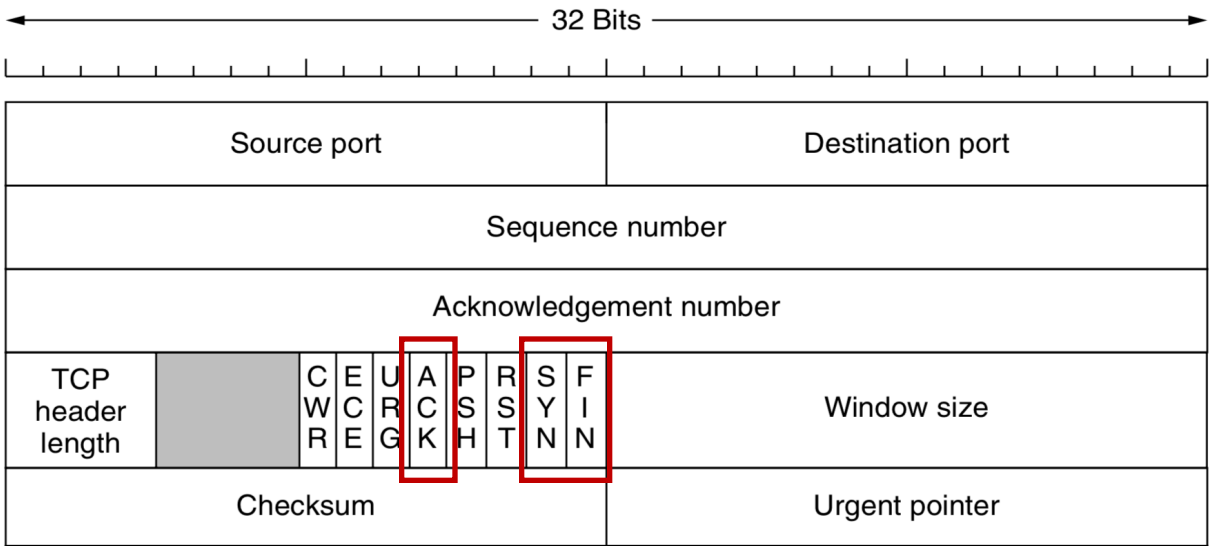
TCP header

- TCP header length:
  - specifies the length of the TCP header in 32-bit words
  - the TCP header can be of variable length due to the TCP options field. (But usually no option field, 20 bytes)



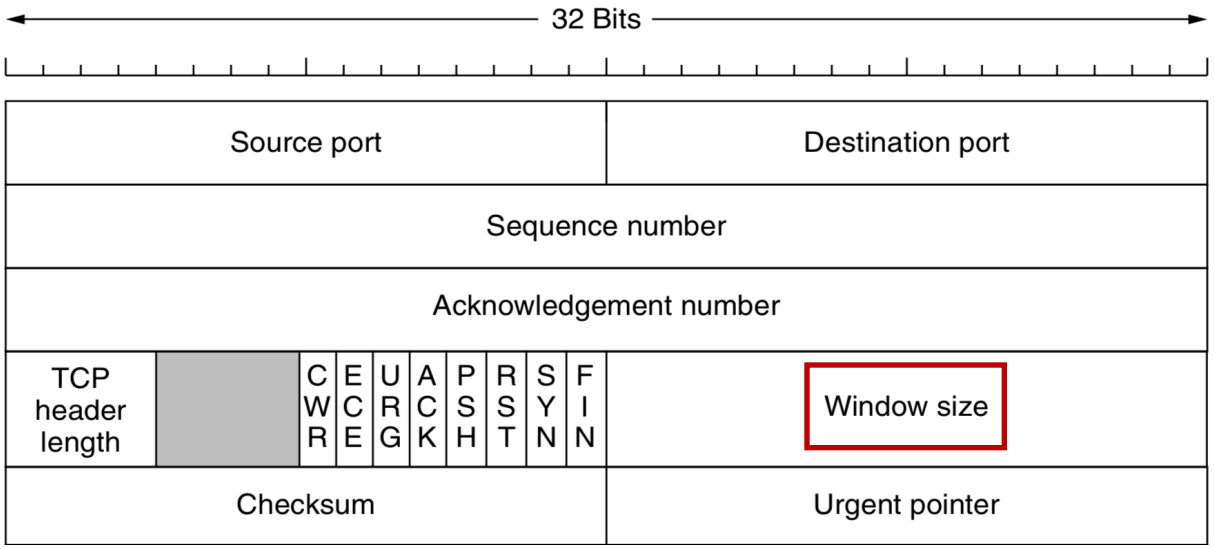
TCP header

- The *ACK* bit is set to 1 to indicate that the *Acknowledgement number* is valid.
- The *SYN* bit is used to establish connections.
- The *FIN* bit is used to release a connection. It specifies that the sender has no more data to *transmit*. (may continue to *receive* data indefinitely)



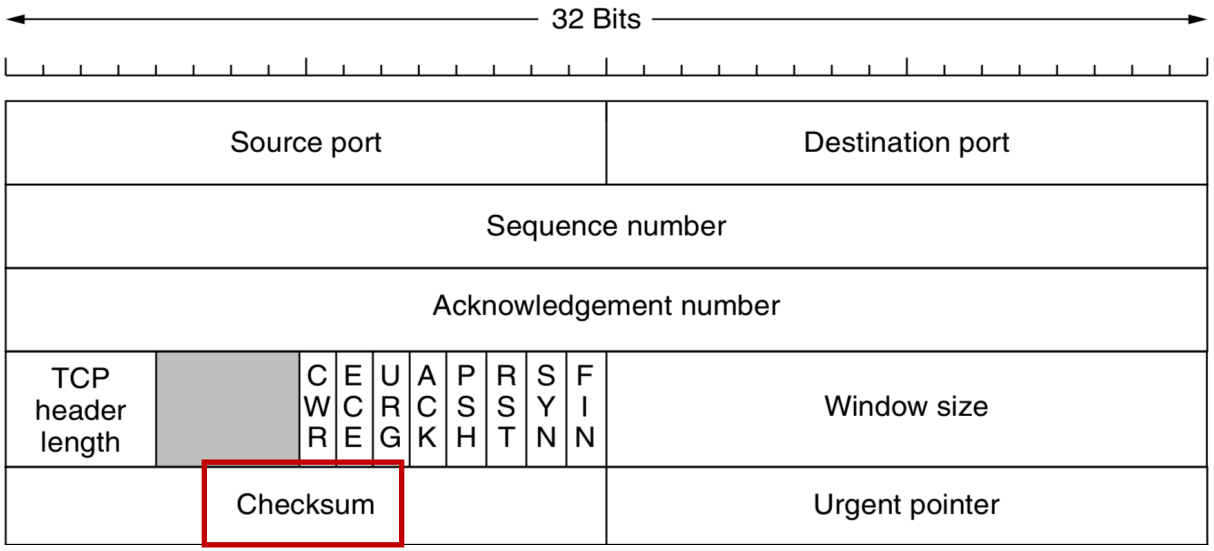
TCP header

- Flow control in TCP is handled using a variable-sized sliding window.
- The *Window size* field tells how many bytes may be sent starting at the byte acknowledged.



TCP header

- It checksums the header, the data, and a IPv4 pseudoheader the same as UDP, except:
  - the pseudoheader has the protocol number for TCP (6)
  - the checksum is mandatory.



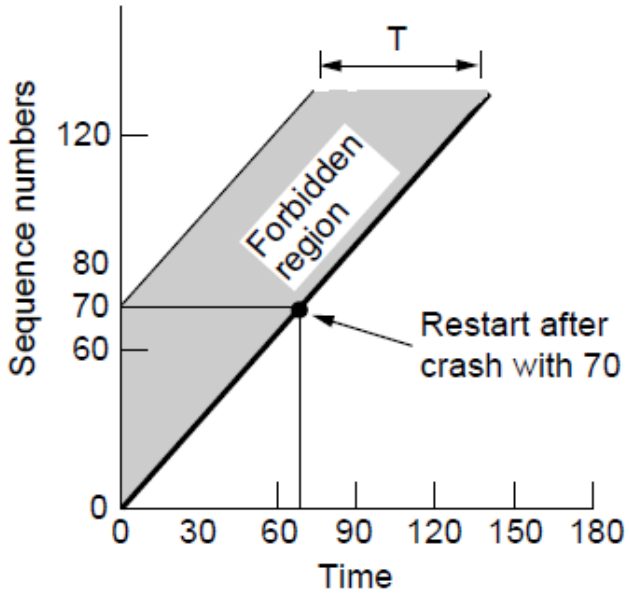
Connection Establishment

- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
  - Don't treat an old or duplicate packet as new
  - (Use ARQ and checksums for loss/corruption)

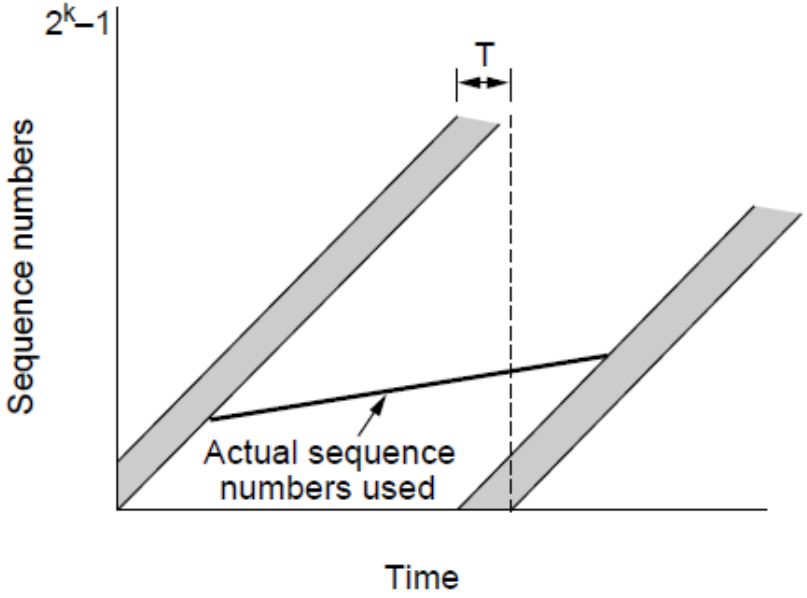
- Approach:
  - Don't reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T=240 secs
  - Three-way handshake for establishing connection

Connection Establishment

- Use a sequence number space large enough that it will not wrap, even when sending at full rate
  - Clock (high bits) advances & keeps state over crash



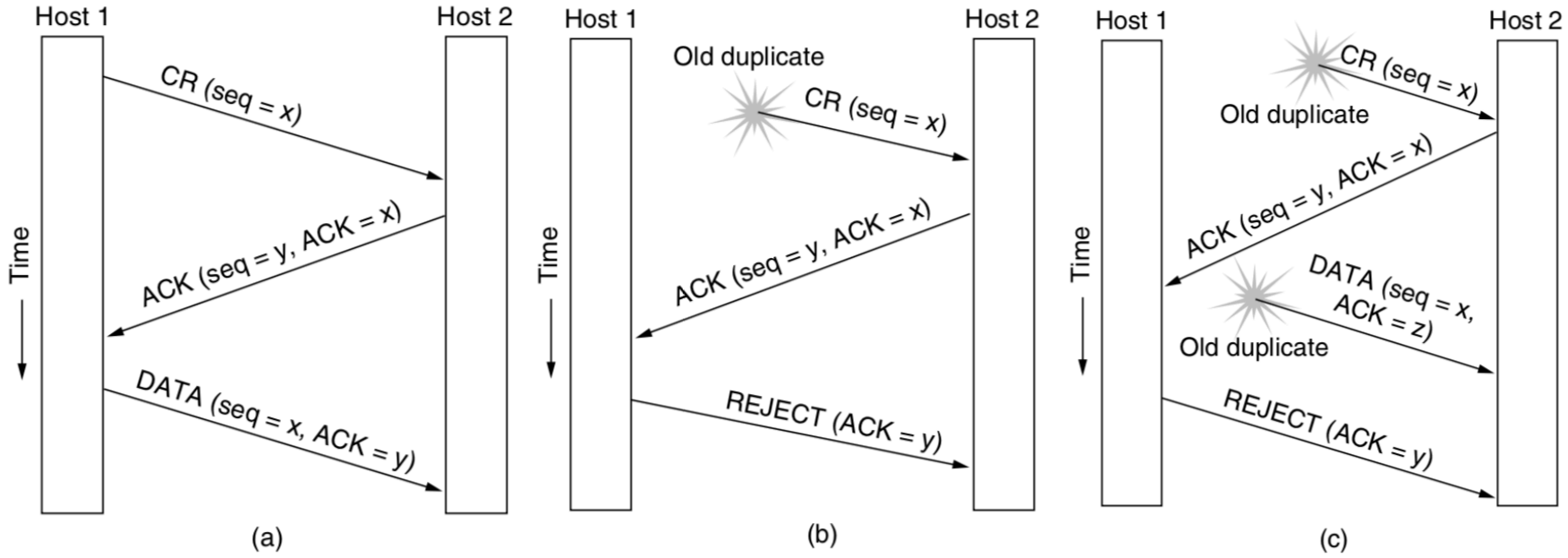
Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

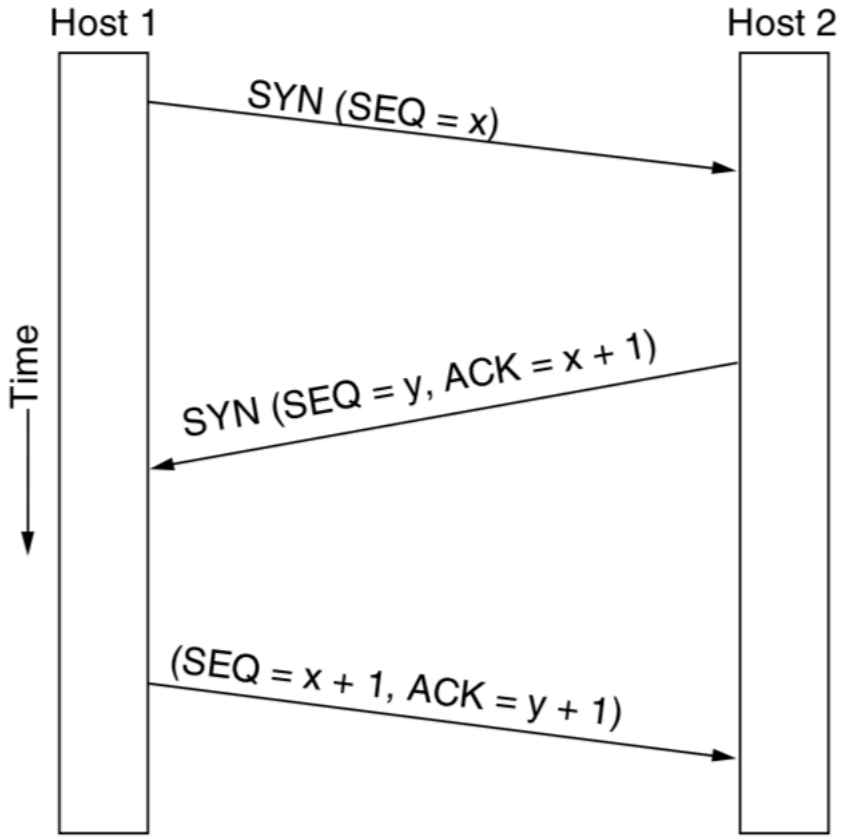
3-Way Handshake – Concept

- CR = CONNECTION REQUEST
- (a): Normal operation. **Robust to duplication and delay.**
- (b): Old duplicate CONNECTION REQUEST appearing out of nowhere.
- (c): Duplicate CONNECTION REQUEST and duplicate ACK.
- Reliable connection can only be established when the network condition is good.



3-Way Handshake – TCP

- ACK: the expected sequence number of the packet to be received
- SYN packets also contain other information about the connection such as **window size** or **MSS** (Maximum Segment Size Default= 536)
- *SYN Flood attack* happens if the third step is "replaced" by the first step.



The *SYN* Flood Attack

- The attacker(s) send a large number of TCP SYN segments, without completing the third handshake step.
- With this deluge of SYN segments, the server's connection resources become exhausted as they are allocated (but never used!) for half-open connections.
- Legitimate clients are then denied service.
- Such SYN flooding attacks were among the first documented Denial of Service (DoS) attacks.
- Fortunately, an effective defense known as **SYN cookies** [RFC 4987] are now deployed in most major operating systems.

SYN cookies

- When the server receives a SYN segment:
  - instead of creating a half-open TCP connection for this SYN
  - the server creates an initial TCP sequence number (“cookie”) that is computed based on source and destination IP addresses and port numbers of the SYN segment, as well as a secret number only known to the server.

- The server then sends the client a SYNACK packet with this “cookie”.
  - Importantly, the server does not remember the cookie or any other state information corresponding to the SYN.

SYN cookies

- A legitimate client will return an ACK segment.
  - When the server receives this ACK, it must verify that the ACK corresponds to some SYN sent earlier.
  - Recall that for a legitimate ACK, the value in the acknowledgment field is equal to the cookie value in SYNACK plus one.
  - The server then creates a fully open connection along with a socket.

- If the client does not return an ACK segment:
  - the original SYN has done no harm at the server
  - since the server hasn't yet allocated any resources in response to the original bogus SYN.

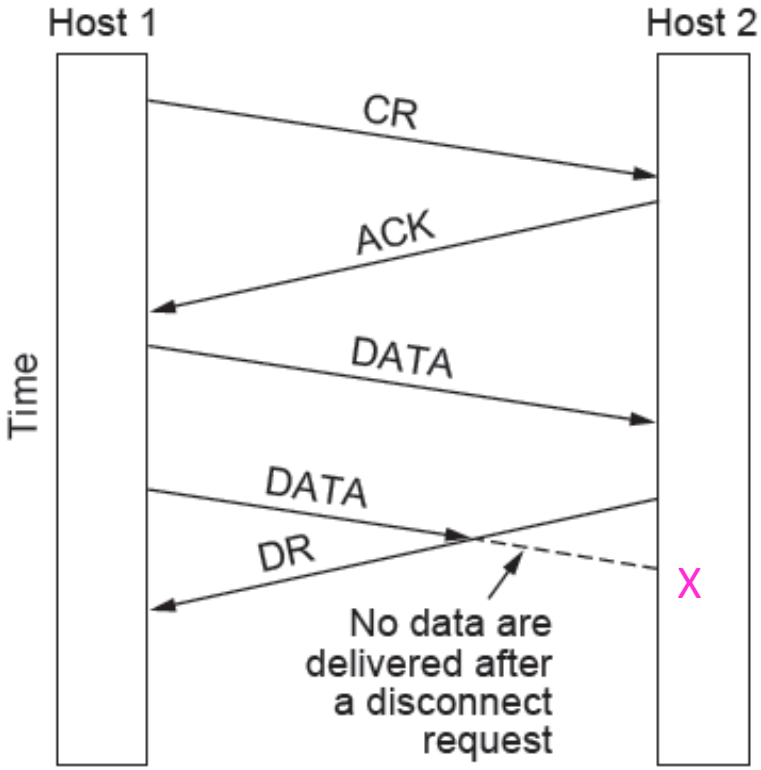
Connection Release

- Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken.

- Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.

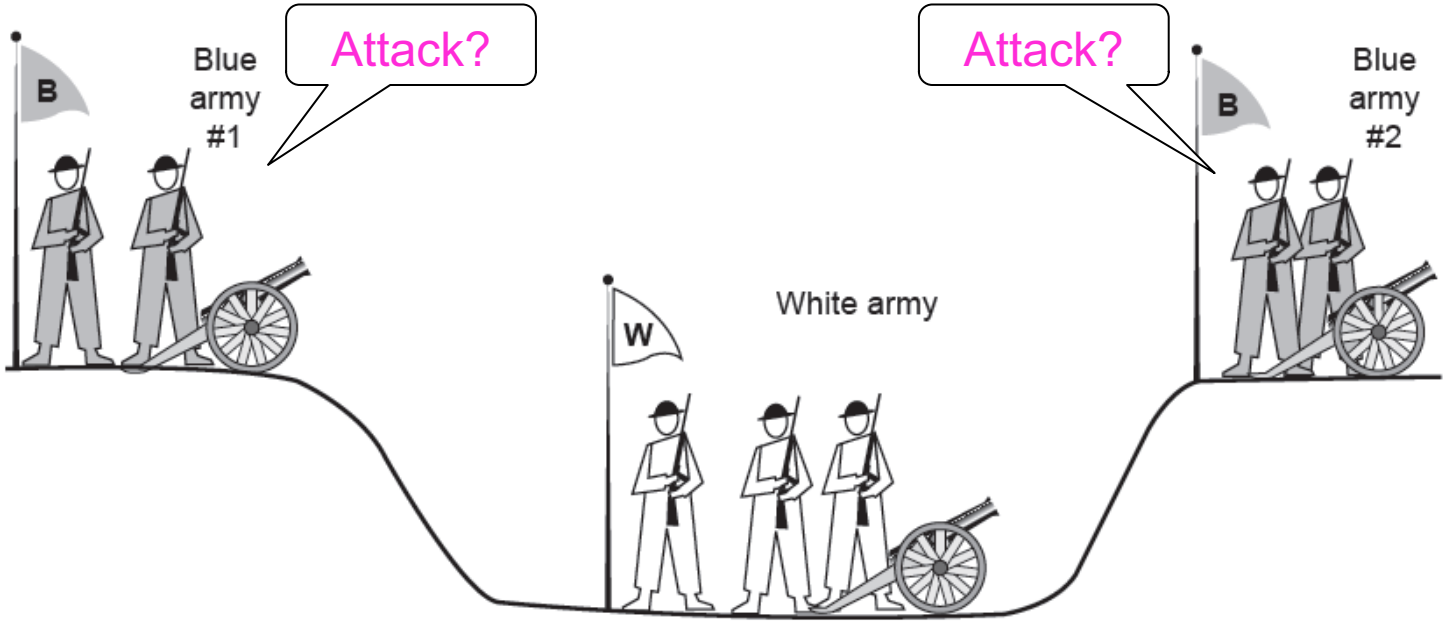
Connection Release

- Key problem is to ensure reliability while releasing
- Asymmetric release (when one side breaks connection) is abrupt and may lose data



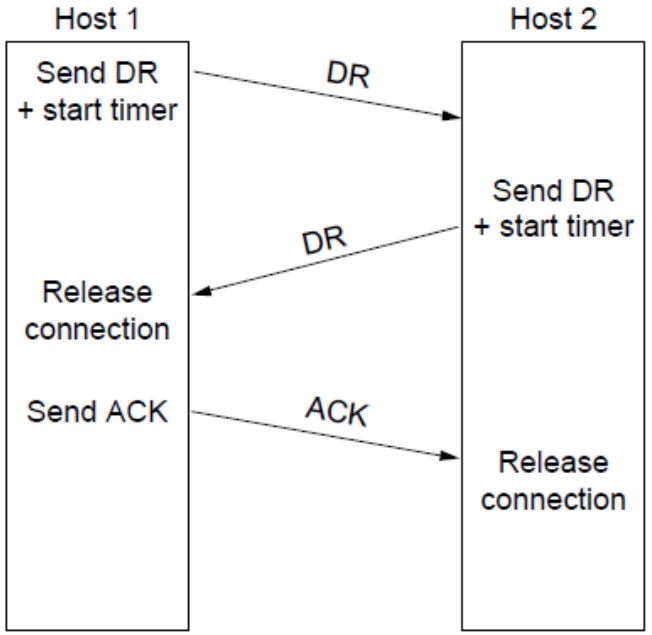
Connection Release

- Symmetric release (both sides agree to release) can't be handled solely by the transport layer
  - Two-army problem shows pitfall of agreement



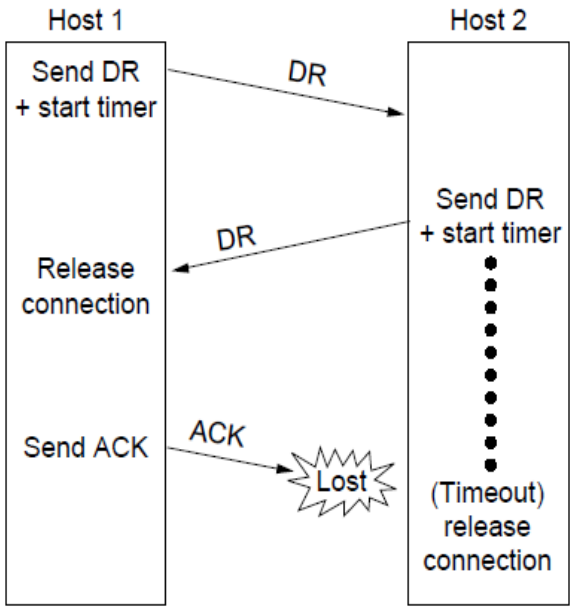
Connection Release

- Normal release sequence, initiated by transport user on Host 1
  - DR=Disconnect Request
  - Both DRs are ACKed by the other side

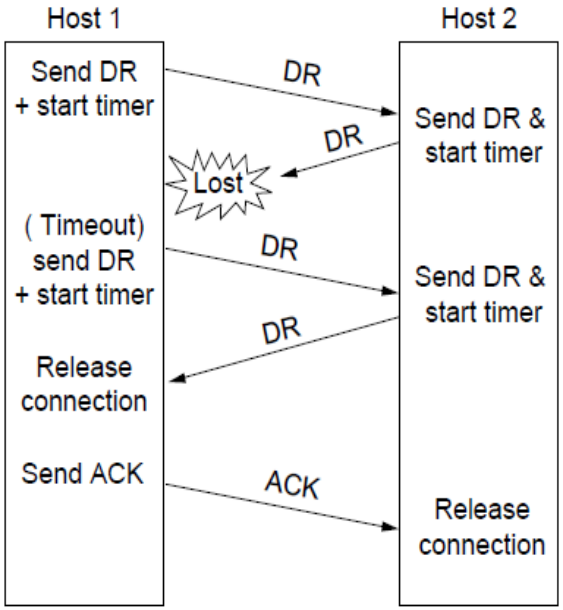


Connection Release

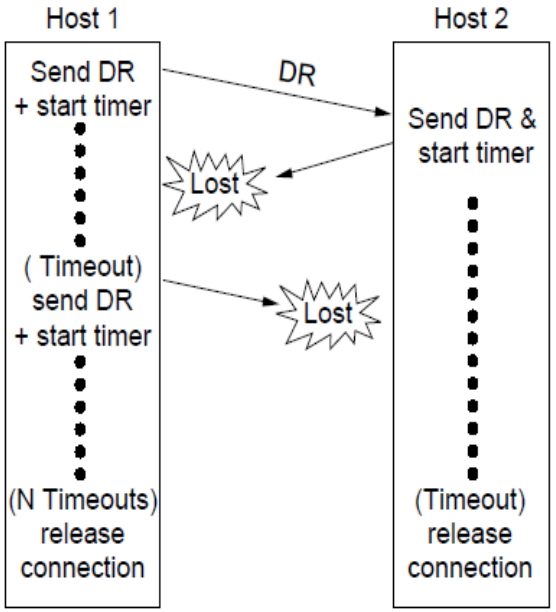
- Error cases are handled with timer and retransmission



Final ACK lost,
Host 2 times out



Lost DR causes
retransmissions



Extreme: Many lost
DRs cause both hosts
to timeout

4-Way “Handwave” – TCP

- Close connections of A -> B, and then B -> A