Internet Protocols
Fall 2005

Lecture 13

Transport Protocols-UDP

Andreas Terzis

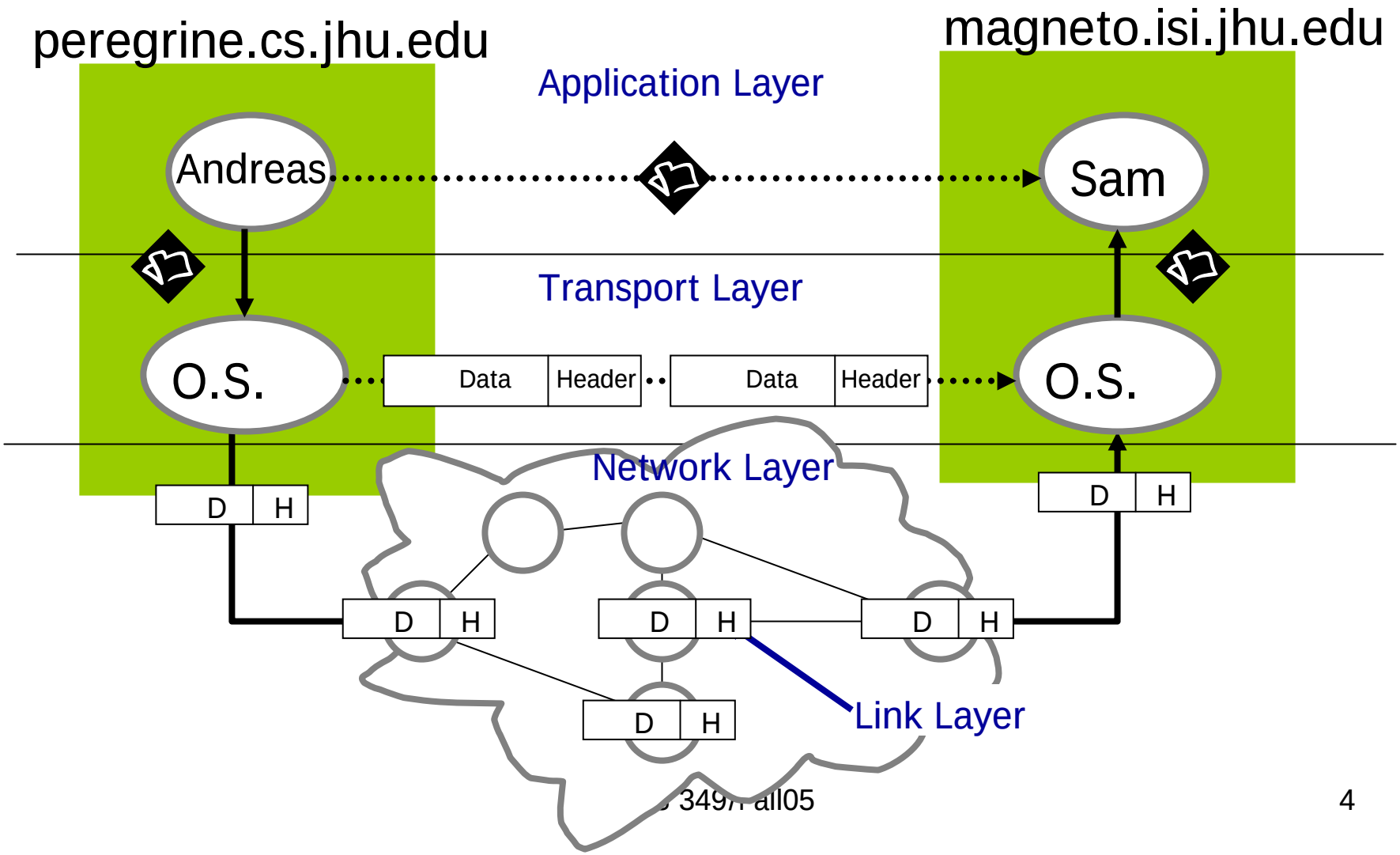
Outline

- Transport Layer Functions
  - De-multiplexing
  - Reliability
  - Flow & Congestion Control
- UDP
  - UDP Checksum
  - IP Fragmentation
- TCP
  - Connection Establishment and Termination

Motivation

- IP provides a weak, but efficient service model (*best-effort*)
  - Packets can be delayed, dropped, reordered, duplicated
  - Packets have limited size (why?)
- IP packets are addressed to a host
  - How to decide which application gets which packets?
- How should hosts send into the network?
  - Too fast is bad; too slow is not efficient

Review of the transport layer



Transport Layer Functions

- De-multiplexing
  - Deliver packets to/from diff applications on the same host
- Reliability
- Flow Control
- Congestion Control

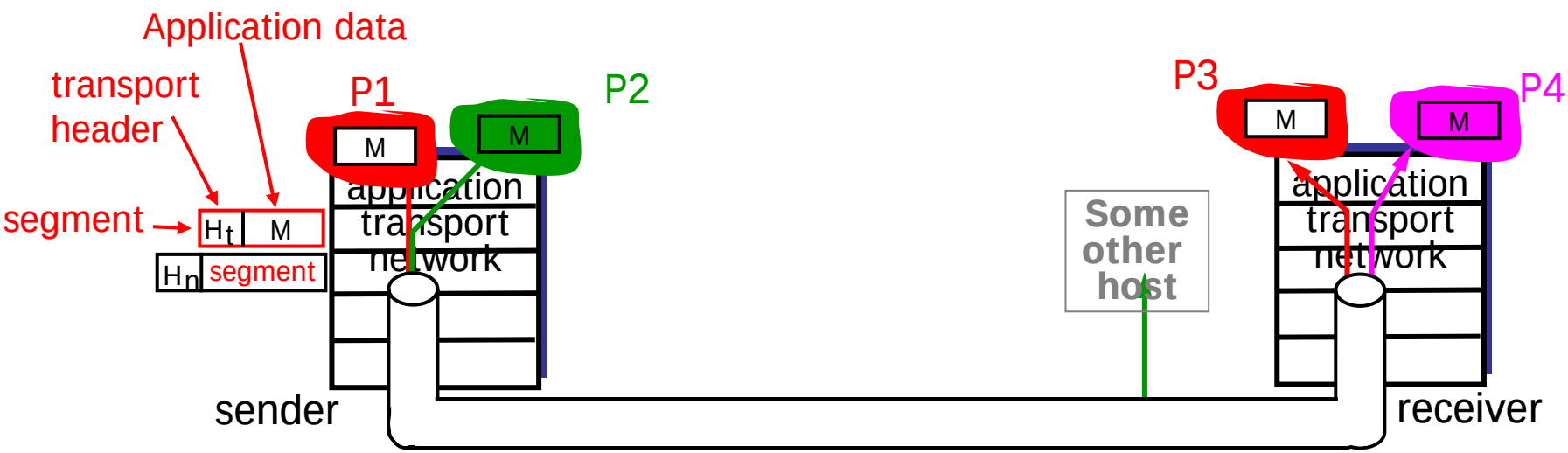
Multiplexing/demultiplexing

Multiplexing

data segments from multiple app processes is sent to lower layer for transmission

Demultiplexing

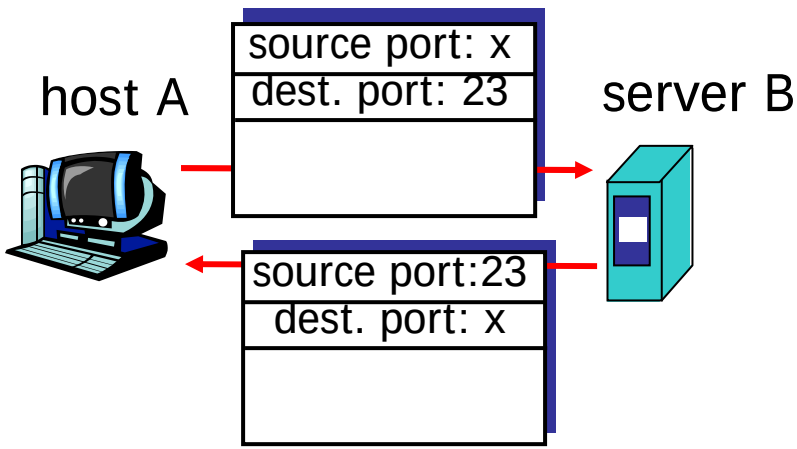
delivering received data segments to corresponding upper layer protocols/apps



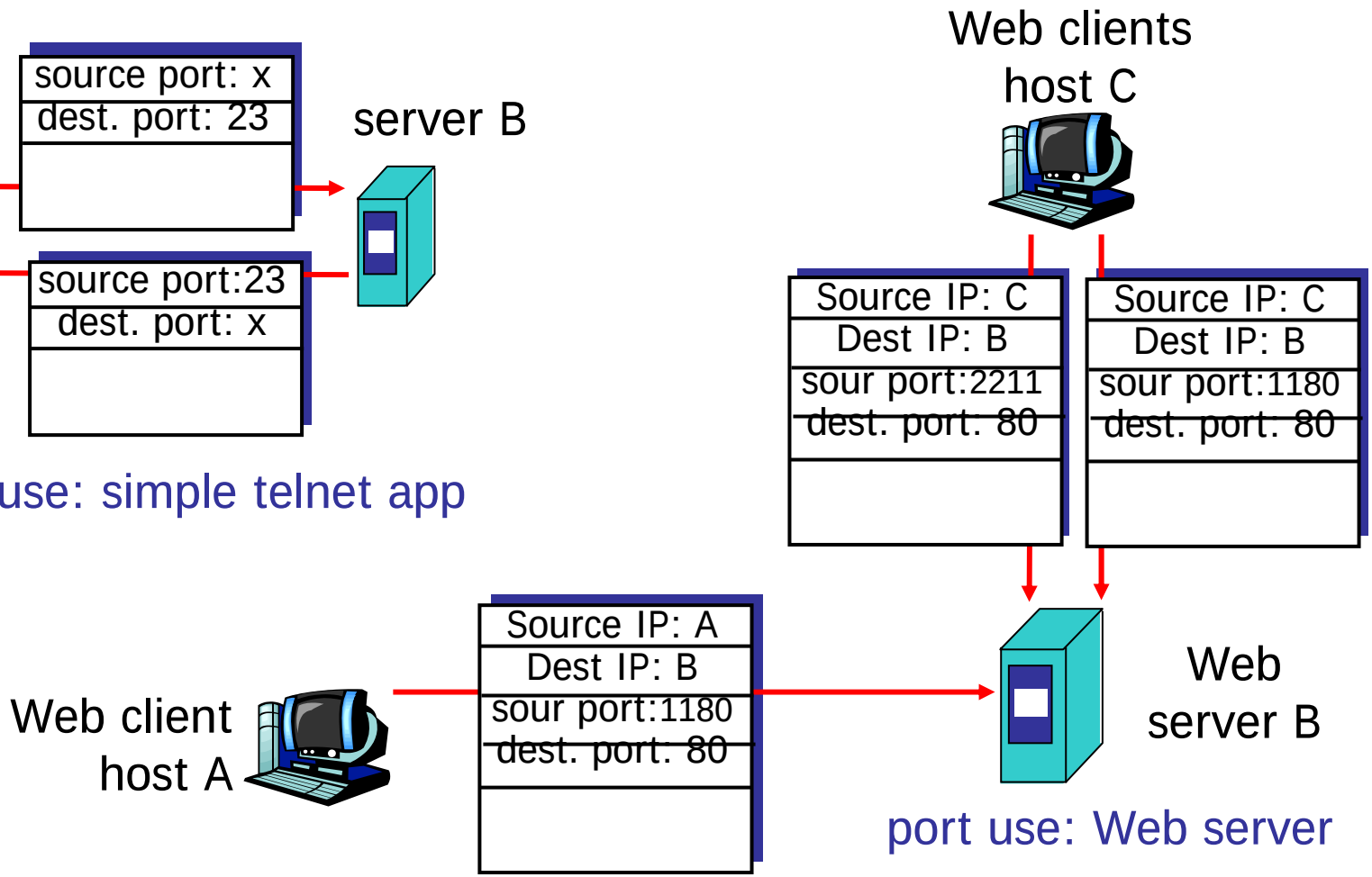
Ports

- Need to decide which application gets which packets
- Solution: map each socket to a *port*
- Client must know server's port
- Separate 16-bit port address space for UDP and TCP
  - (src_IP, src_port, dst_IP, dst_port) uniquely identifies TCP connection
- *Well known ports* (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80
  - on UNIX, must be root to gain access to these ports (why?)
- Ephemeral ports (most 1024-65535): given to clients

Multiplexing/demultiplexing: examples



port use: simple telnet app

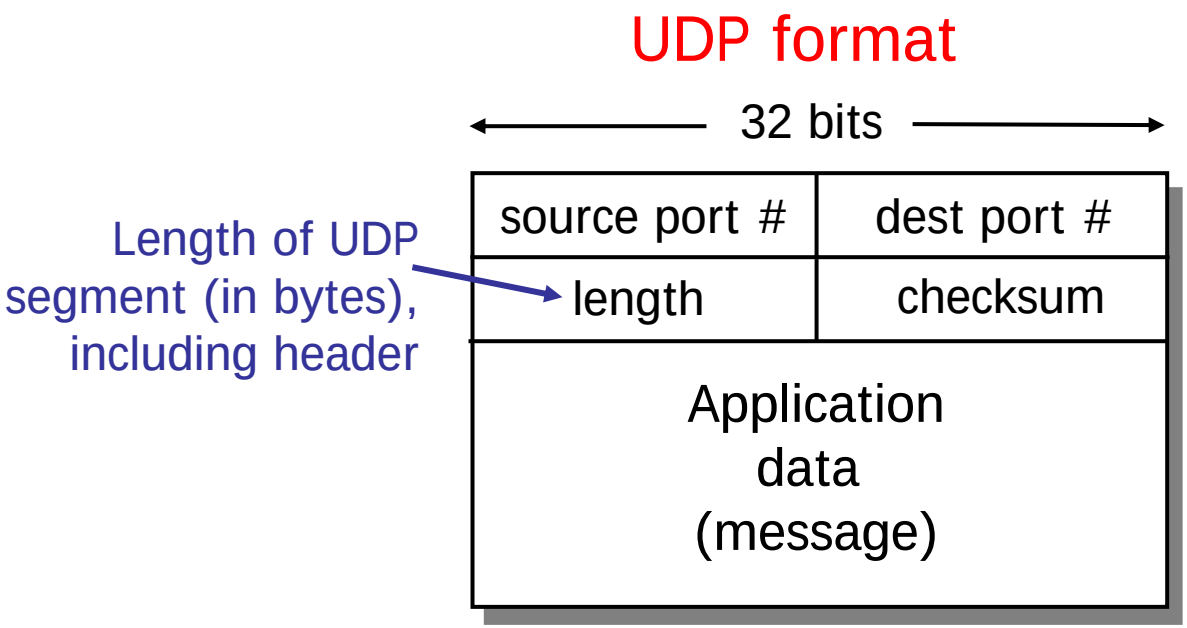


port use: Web server

UDP Characteristics

- UDP is a connectionless datagram service.
  - There is no connection establishment: packets may show up at any time.
- UDP packets are self-contained.
- UDP is unreliable:
  - No acknowledgements to indicate delivery of data.
  - Contains no mechanism to detect missing or mis-sequenced packets.
  - No mechanism for automatic retransmission.
  - No mechanism for flow control, and so can over-run the receiver.

UDP Header


UDP format
32 bits
source port #
dest port #
length
checksum
Application
data
(message)
Length of UDP
segment (in bytes),
including header

UDP checksum

Goal: detect bit errors (e.g., flipped bits) in transmitted segment

- Sender:
- treat data in the segment as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- puts checksum value into UDP checksum field

- Receiver:
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected

UDP Checksum Calculation

- UDP header
  - Length: # of bytes (including both header & data)
  - checksum: computed over
    - the pseudo header, and
    - UDP header and data.
    - if the field is 0, no checksum

UDP header format

32 bits

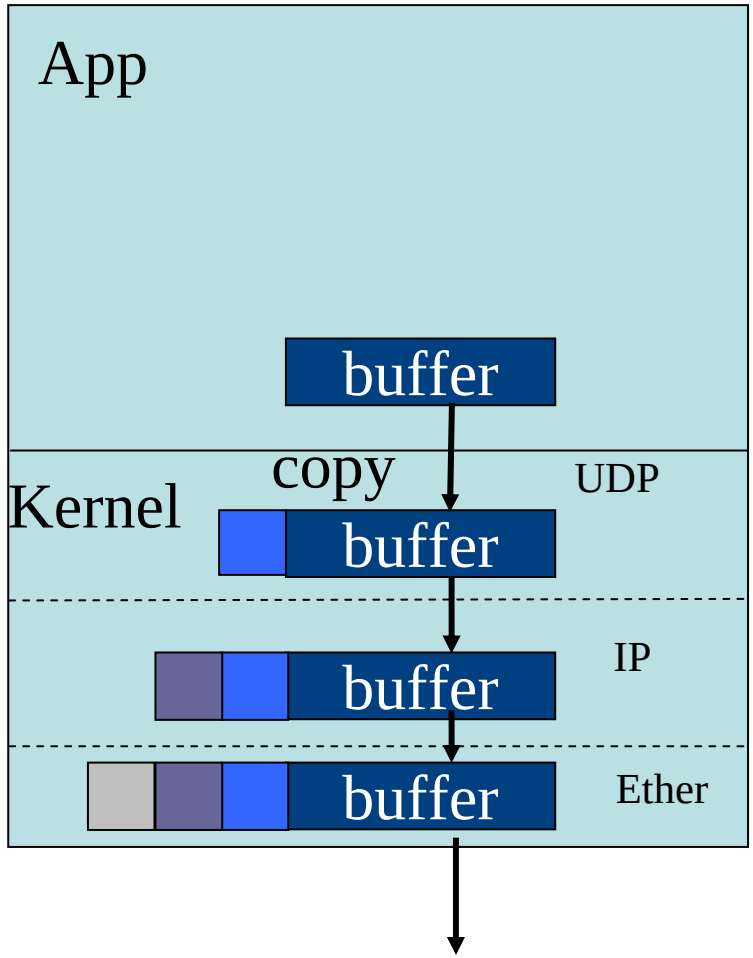
source port #	dest port #
length	checksum
Application data (message)	

- pseudo header: UDP's self-protection against mis-delivered IP packets

source IP address		
destination IP address		
zero	**protocol**	**UDP length**

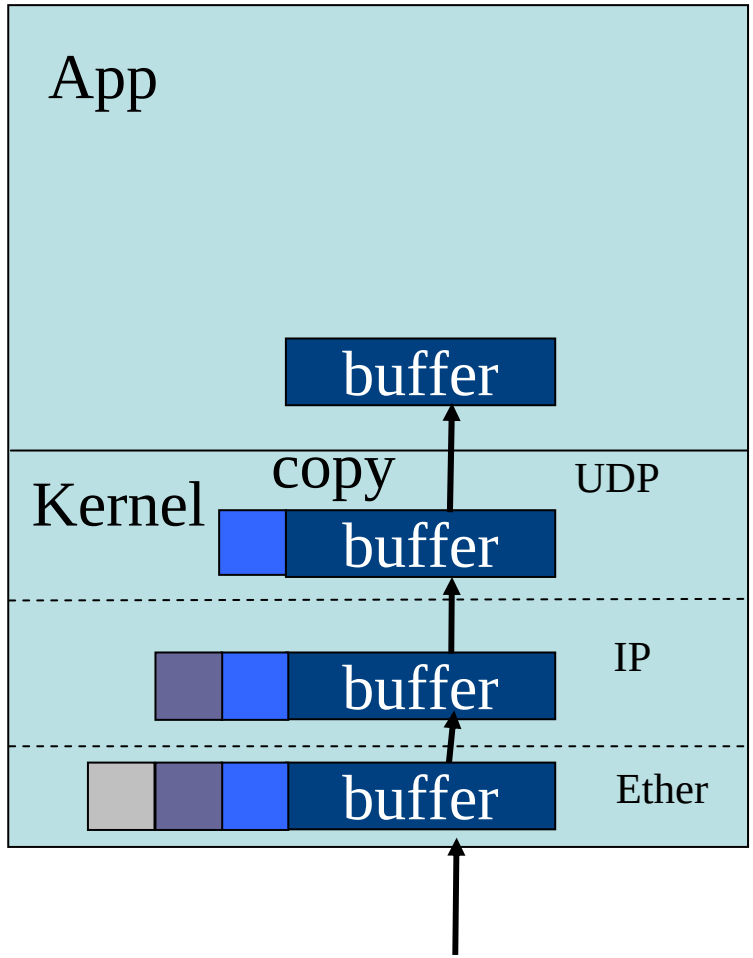
Implementation (sender side)

- App calls `socket()`
- App writes data in `buffer`
- App calls `Sendto(&buffer, bufferlen)`
- Data is copied to kernel



Implementation (recv. side)

- App calls `socket()`
- App calls `bind()`
- Data is received from net and stored in kernel buffer
  - What happens if buffer gets full?
- App calls `recvfrom(&buffer,bufferlen)`
- Data is copied to application `buffer`



Fragmentation

- UDP application can send buffer up to 8000 bytes
- Most links have MTU size ~ 1500 bytes
  - UDP segment is fragmented to multiple IP datagram fragments
  - Fragments reassembled at receiver
  - Expensive process

Path MTU Discovery

- Alternative:
  - Discover max path MTU on the sender side
  - How:
    - Send IP datagrams with DF flag set
    - Routers reply with ICMP Unreachable Error (Frag Required)
      - Contains the MTU of next hop
    - Continue process with the smaller MTU

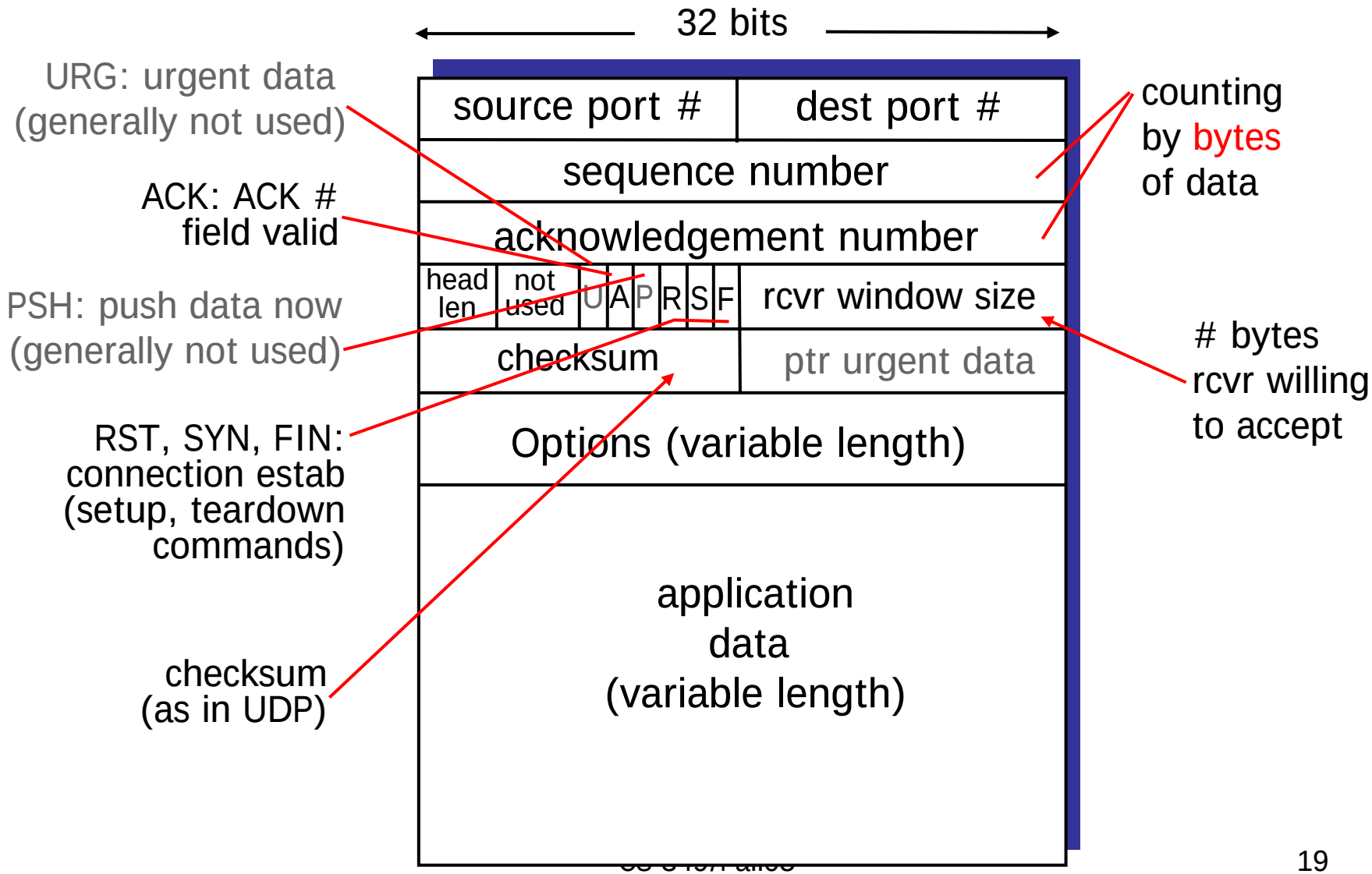
TCP

- Transmission Control Protocol
- Reliable, in-order, and at most once delivery
- Messages can be of arbitrary length
- Provides multiplexing/demultiplexing to IP
- Provides congestion control and avoidance
- Application examples: file transfer, chat, web

TCP Service

1) Open connection
2) Reliable byte stream transfer from (IPa, TCP Port1) to (IPb, TCP Port2)
   - Indication if connection fails: Reset
3) Close connection

TCP segment structure



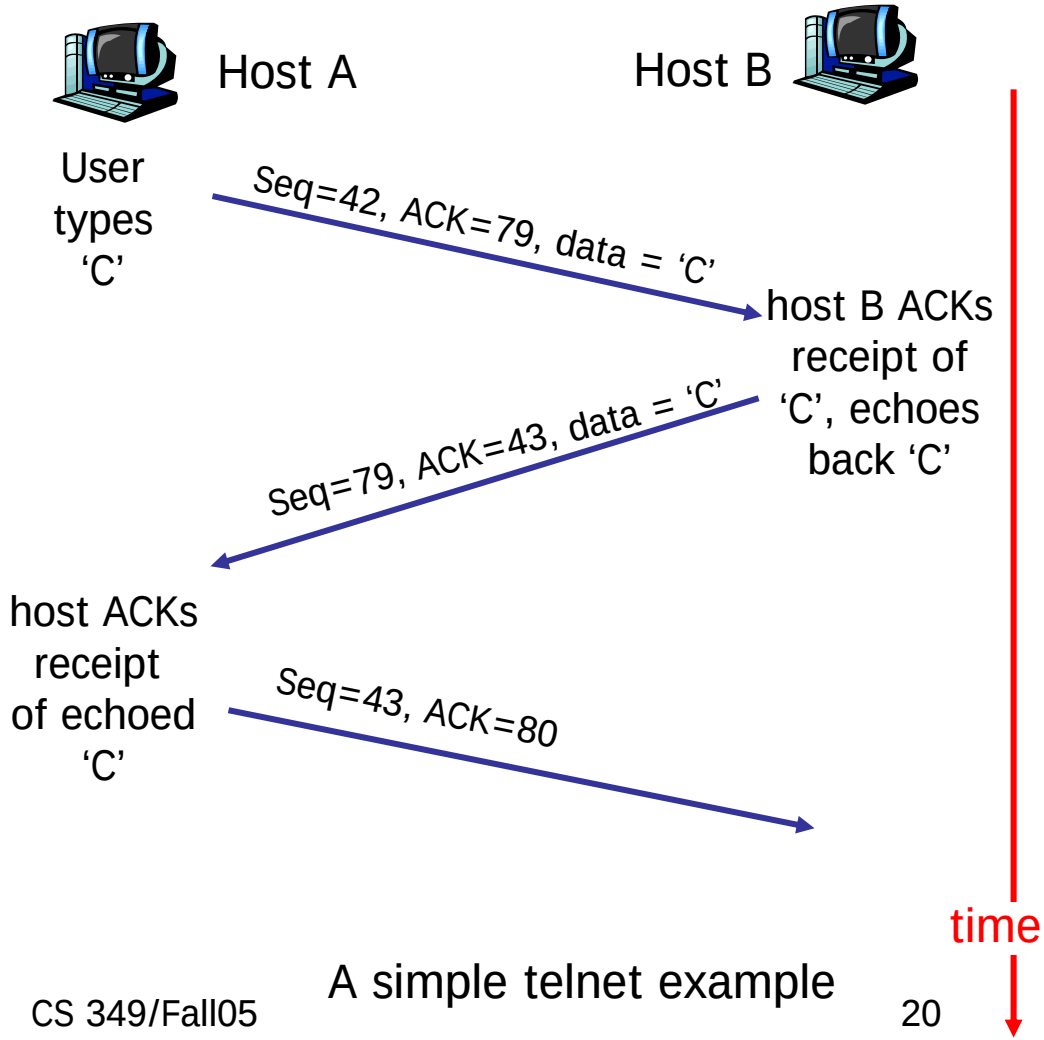
TCP's seq. #s and ACK #s

Seq. #:

- The number of first byte in segment's data

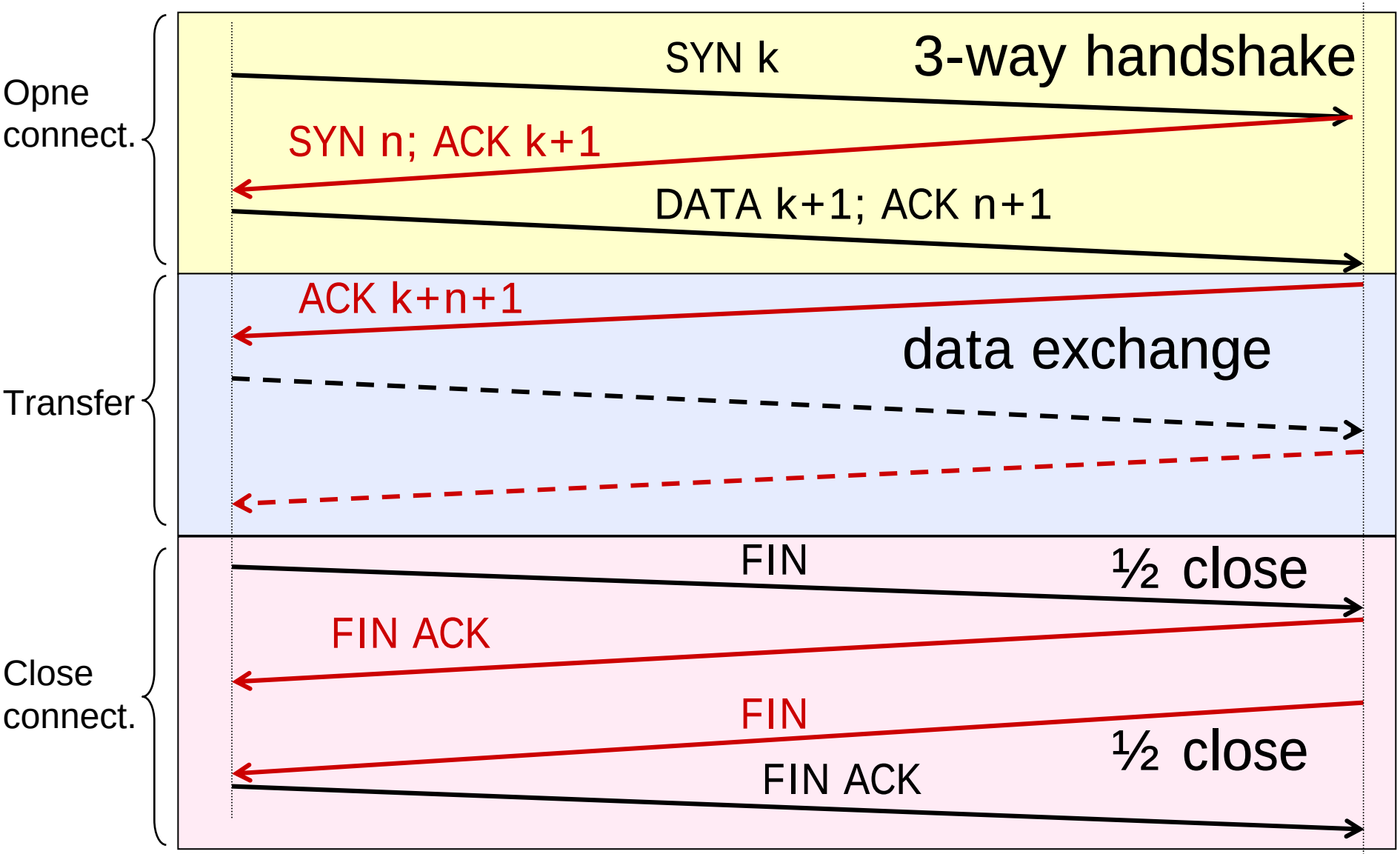
ACK #:

- seq # of next byte expected from other side
- cumulative ACK



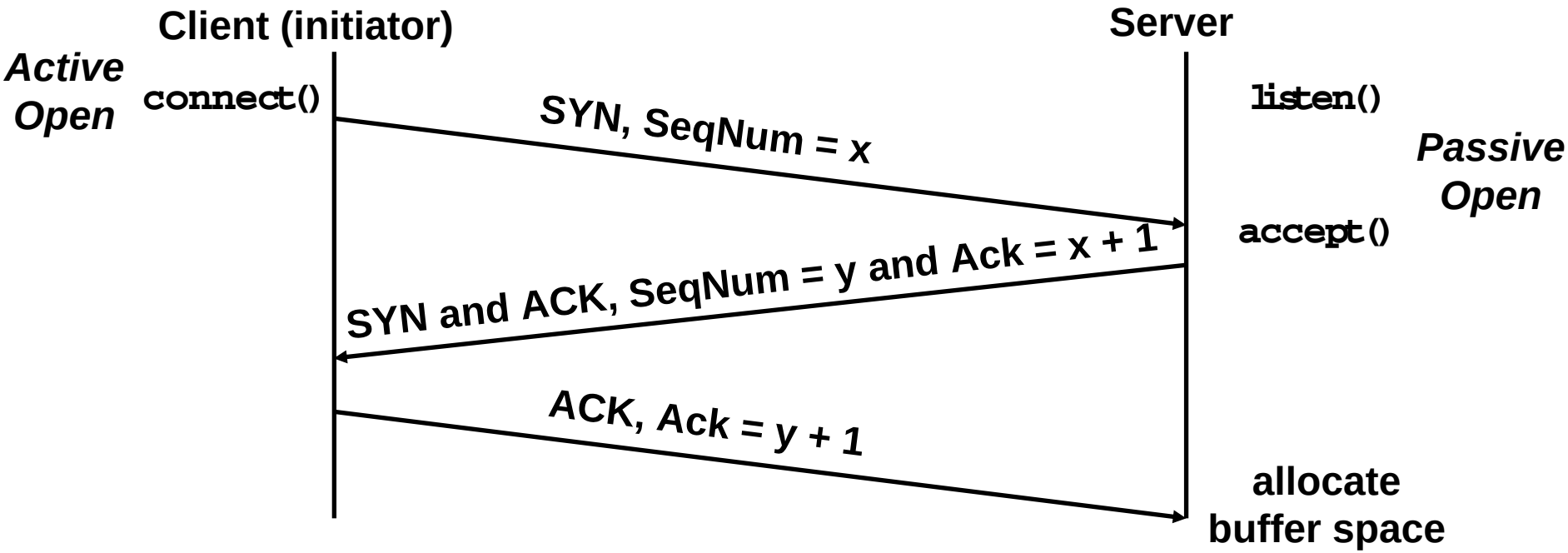
A simple telnet example

Timing Diagram



Open Connection: 3-Way Handshaking

- Goal: agree on a set of parameters: the start sequence number for each side
  - Starting sequence numbers are random.

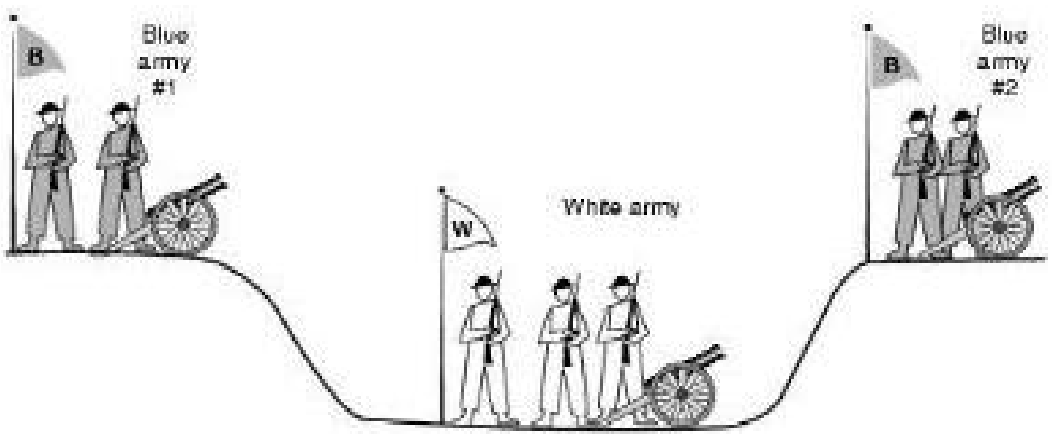


3-Way Handshaking (cont'd)

- Three-way handshake adds 1 RTT delay
- Why?
  - Congestion control: SYN (40 byte) acts as cheap probe
  - Protects against delayed packets from other connection (would confuse receiver)

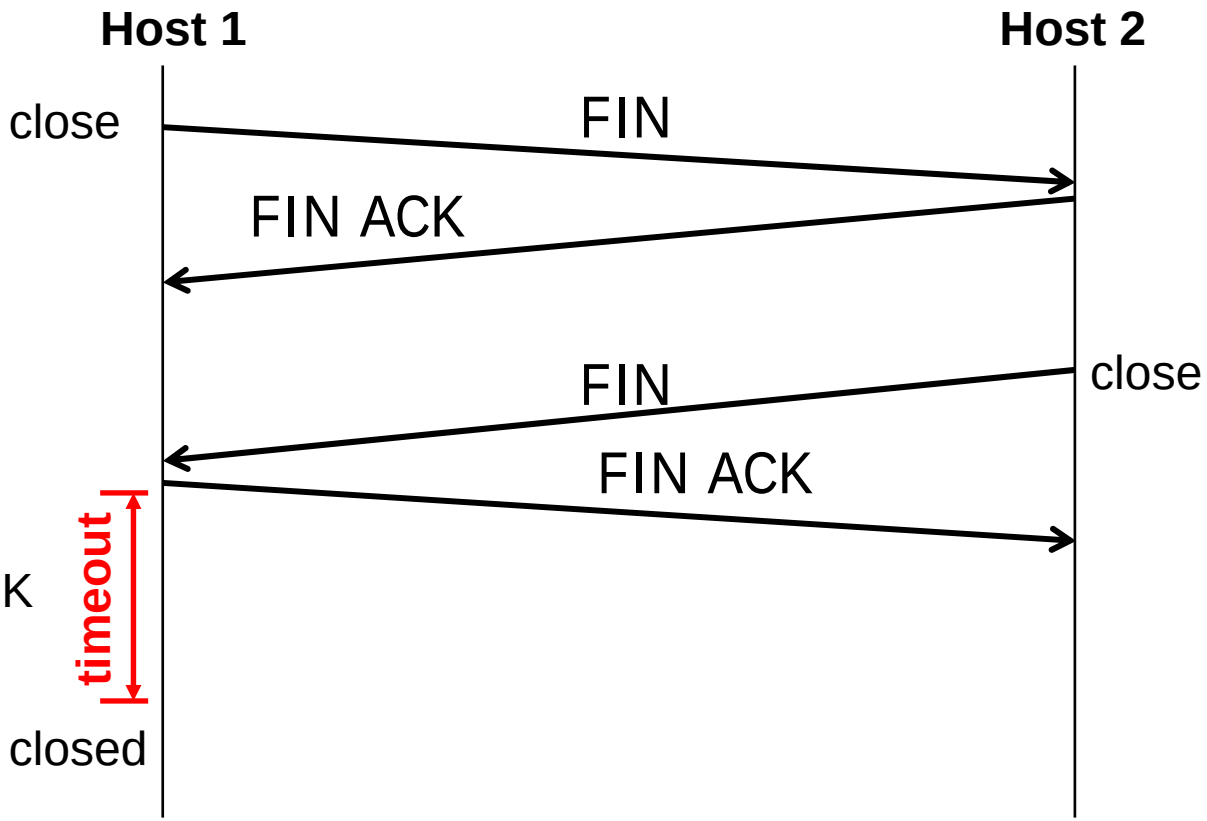
Close Connection (Two-Army Problem)

- Goal: both sides agree to close the connection
- Two-army problem:
  - "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."
- What is the solution?



Close Connection

- 4-ways tear down connection



- Avoid reincarnation
- Can retransmit FIN ACK if it is lost