COSC264

Introduction to Computer Networks and the Internet

Transport Layer Protocols: UDP and TCP

Dr. Barry Wu

Wireless Research Centre

University of Canterbury

barry.wu@canterbury.ac.nz

Outline

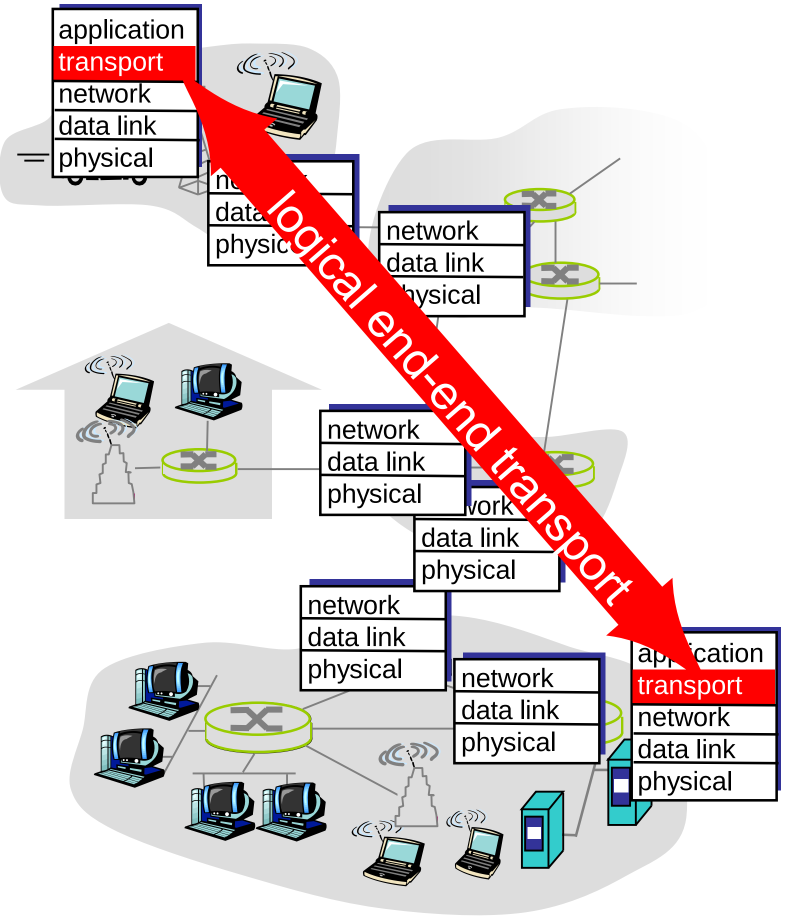
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP

Role of Transport Layer

- Application layer
  - Communication for specific applications
  - e.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)
- **Transport layer**
  - Communication between processes (e.g., socket = IP address and Port)
  - Relies on network layer and serves the application layer
  - e.g., TCP and UDP
- Network layer
  - Logical communication between nodes
  - Hides details of the link technology
  - e.g., Internet Protocol (IP)

Transport Protocols

- Provide *logical communication* between application processes running on different hosts
- Run on end hosts
  - Sender: breaks application messages into segments, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP
    - Other: Datagram Congestion Control Protocol (DCCP), Stream Control Transmission Protocol (SCTP)



Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Household analogy:

12 kids sending letters to 12 kids

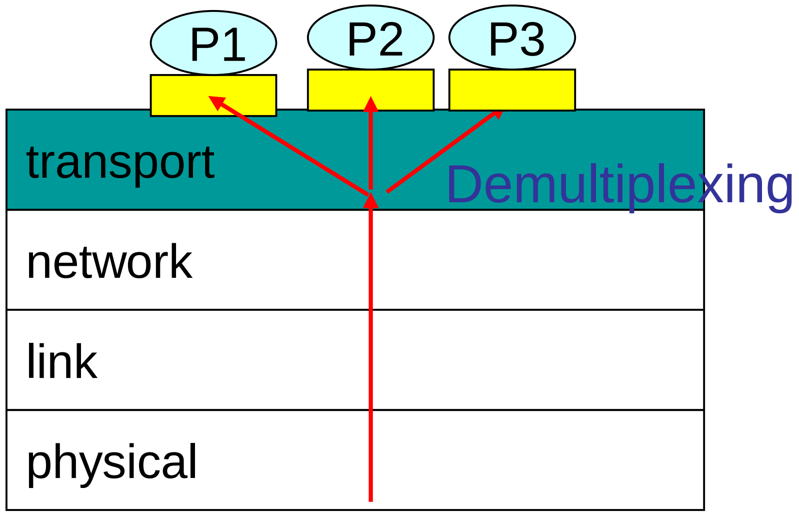
- app messages = letters in envelopes
- processes = kids
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

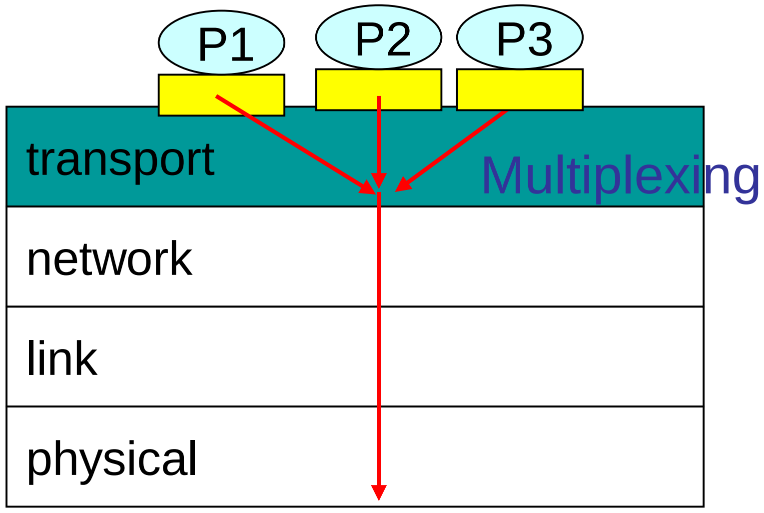
Outline

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP

Multiplexing/demultiplexing – Definition

- Demultiplexing
  - To deliver the data in a transport-layer segment to the correct socket
- Multiplexing
  - To create segments and pass them to the network layer.

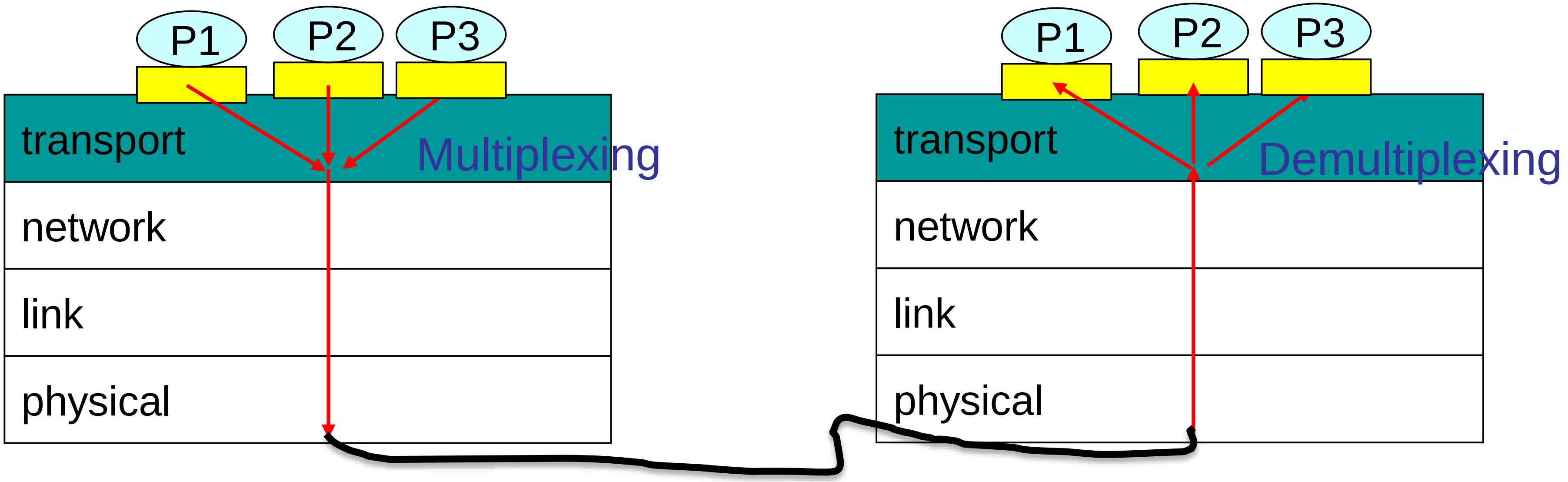




Multiplexing/demultiplexing – Why?

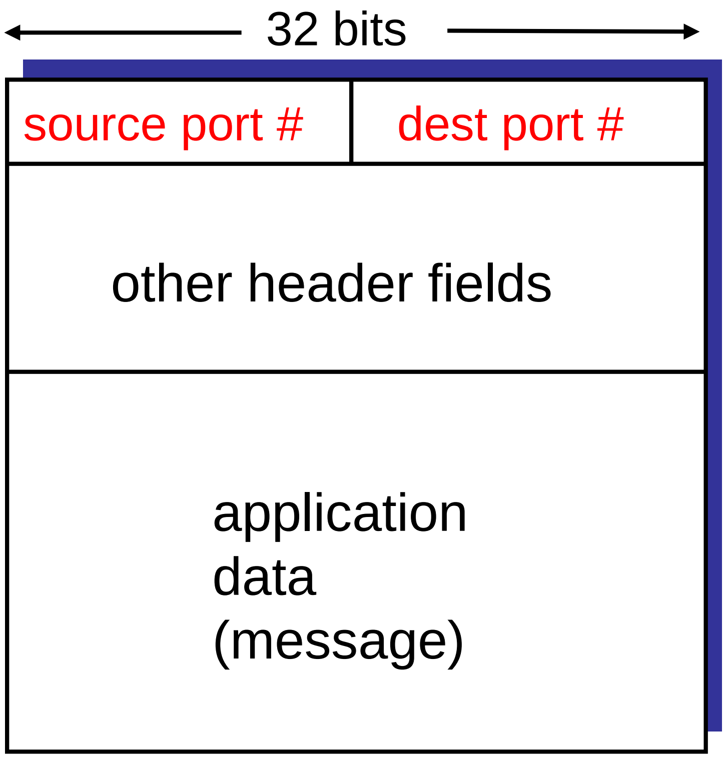
- Why need demultiplexing?
  - Now, the transport layer needs a way to determine which application the packet needs to be delivered. This is the demultiplexing problem.

- Why need multiplexing?
  - To facilitate demultiplexing!



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
   DatagramSocket(99111);
DatagramSocket mySocket2 = new
   DatagramSocket(99222);
```

- UDP socket identified by two-tuple:

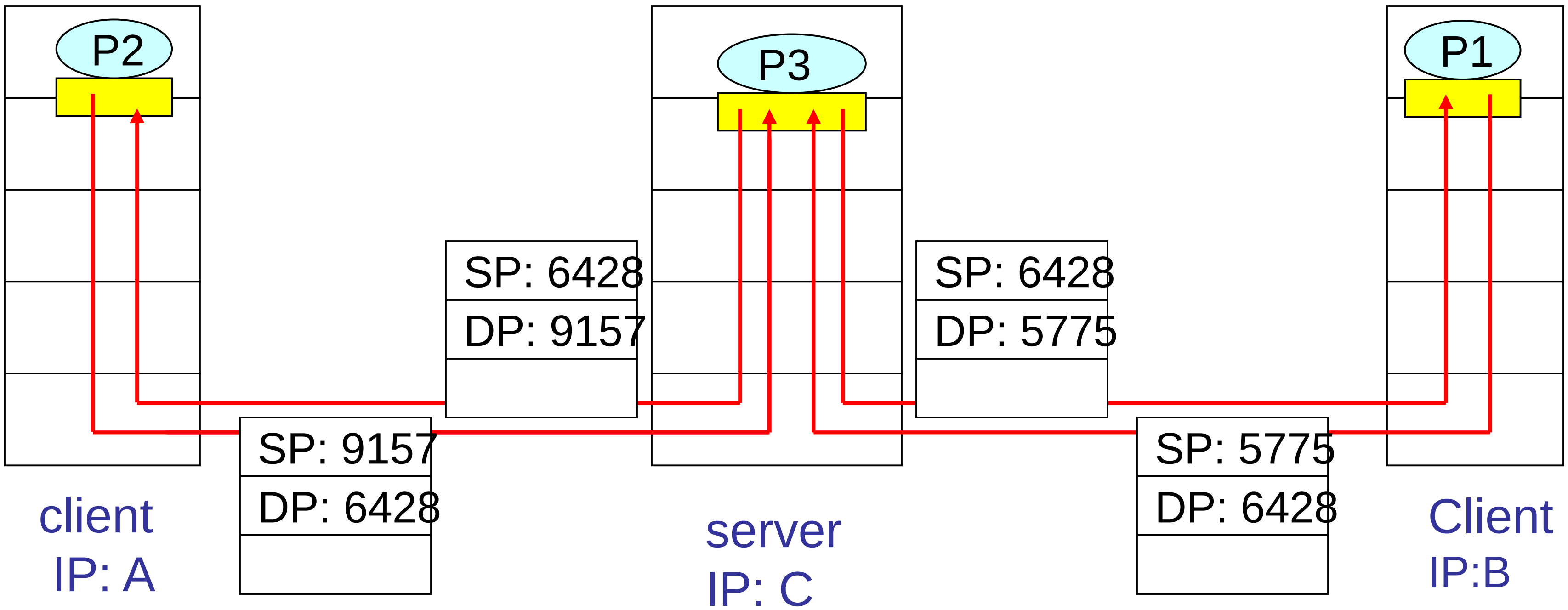
(dest IP address, dest port number)

Why do we need source port # ?

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers, but with the same destination IP and port number, will be directed to same socket.

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



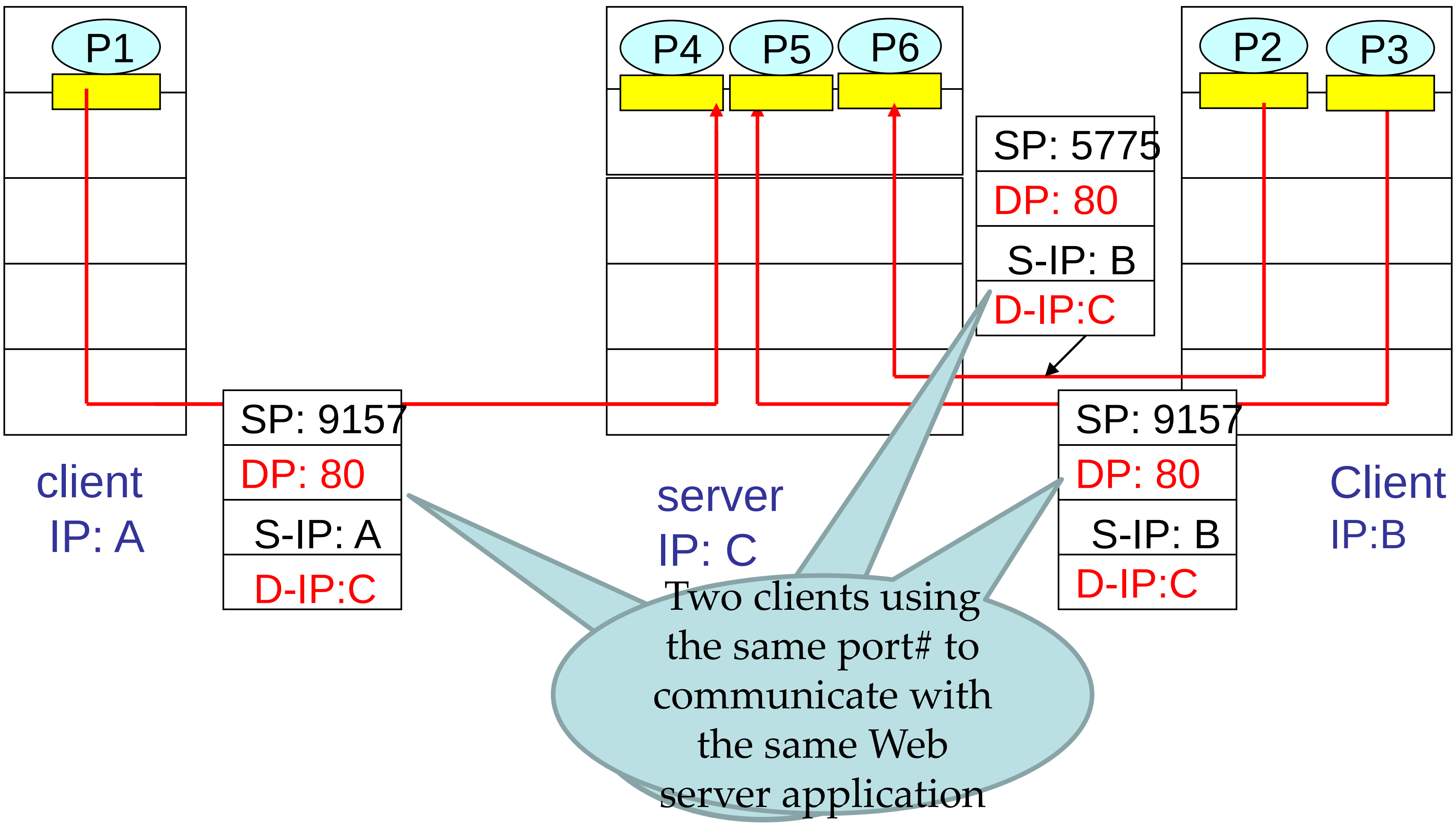
SP provides “return address”

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket

- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)



Outline

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP

UDP: User Datagram Protocol [RFC 768]

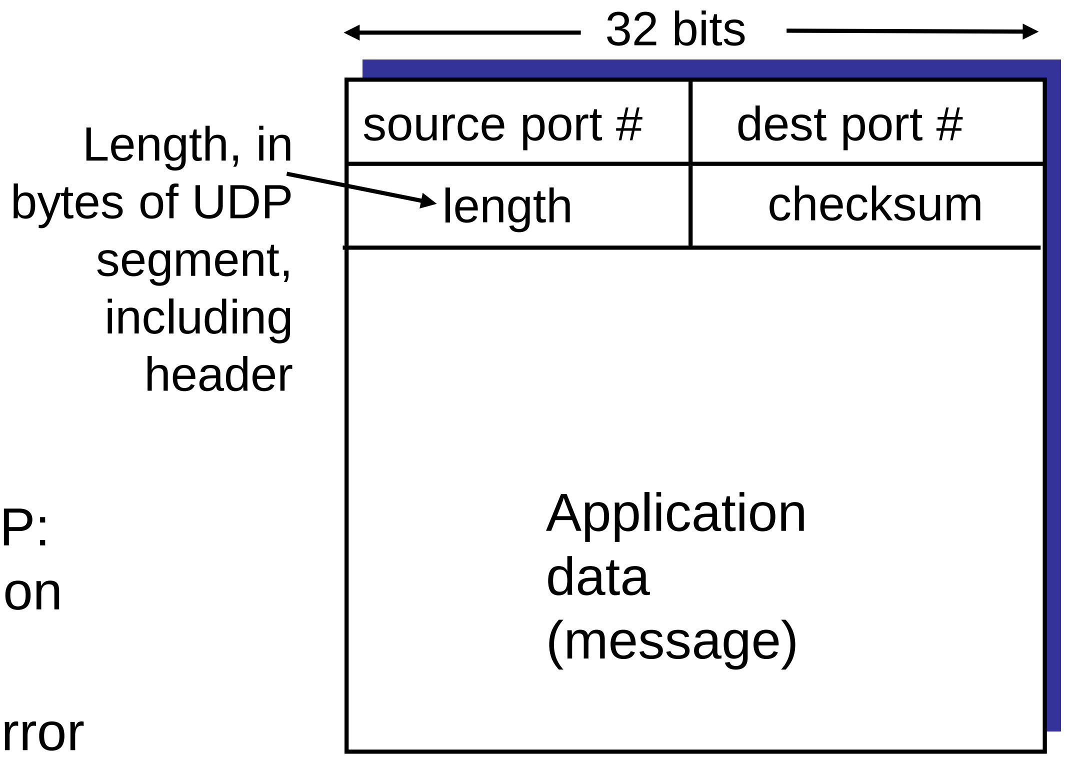
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

Popular Internet Applications and their underlying transport protocols

Application	Application-layer protocol	Underlying transport protocol
Email	SMTP	TCP
Web	HTTP	TCP
Routing protocol	BGP	TCP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP
Streaming multimedia	Typically proprietary	Typically UDP
Internet telephony	Typically proprietary	Typically UDP

UDP checksum

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

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender 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. *But maybe errors nonetheless?* More later ….

Internet Checksum Example

- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

		1	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**
		1	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
wraparound	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
sum		**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
checksum		**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**

A UDP segment captured by Wireshark

12	0.508950	10.34.40.169	132.181.2.225	DNS
13	0.509041	132.181.2.225	10.34.40.169	DNS
14	0.509462	10.34.40.169	132.181.2.225	DNS
15	0.510280	132.181.2.225	10.34.40.169	DNS
16	0.510354	132.181.2.225	10.34.40.169	DNS
17	33.503303	10.34.40.169	132.181.2.225	DNS
18	33.504666	132.181.2.225	10.34.40.169	DNS
19	61.002465	10.34.40.169	132.181.2.225	DNS
20	61.004045	132.181.2.225	10.34.40.169	DNS
21	61.013117	10.34.40.169	132.181.2.225	DNS
22	61.014578	132.181.2.225	10.34.40.169	DNS

```
▷ Frame 12: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
▷ Ethernet II, Src: IntelCor_b6:fe:63 (80:19:34:b6:fe:63), Dst: JuniperN_ef:61:00 (2c:21:31:ef:61:00)
▷ Internet Protocol Version 4, Src: 10.34.40.169, Dst: 132.181.2.225
◢ User Datagram Protocol, Src Port: 63507, Dst Port: 53
    Source Port: 63507
    Destination Port: 53
    Length: 54
    Checksum: 0xe576 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 6]
▷ Domain Name System (query)
```

Why Would Anyone Use UDP?

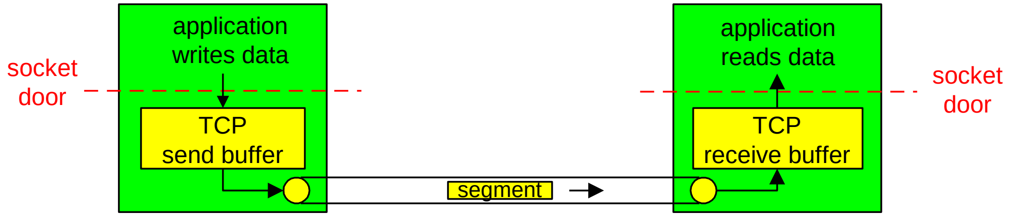
- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
  - … making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only eight-bytes long

Outline

- Connection-oriented transport: TCP
  - Overview and segment structure
  - Connection management
  - Reliable data transfer
  - Flow control

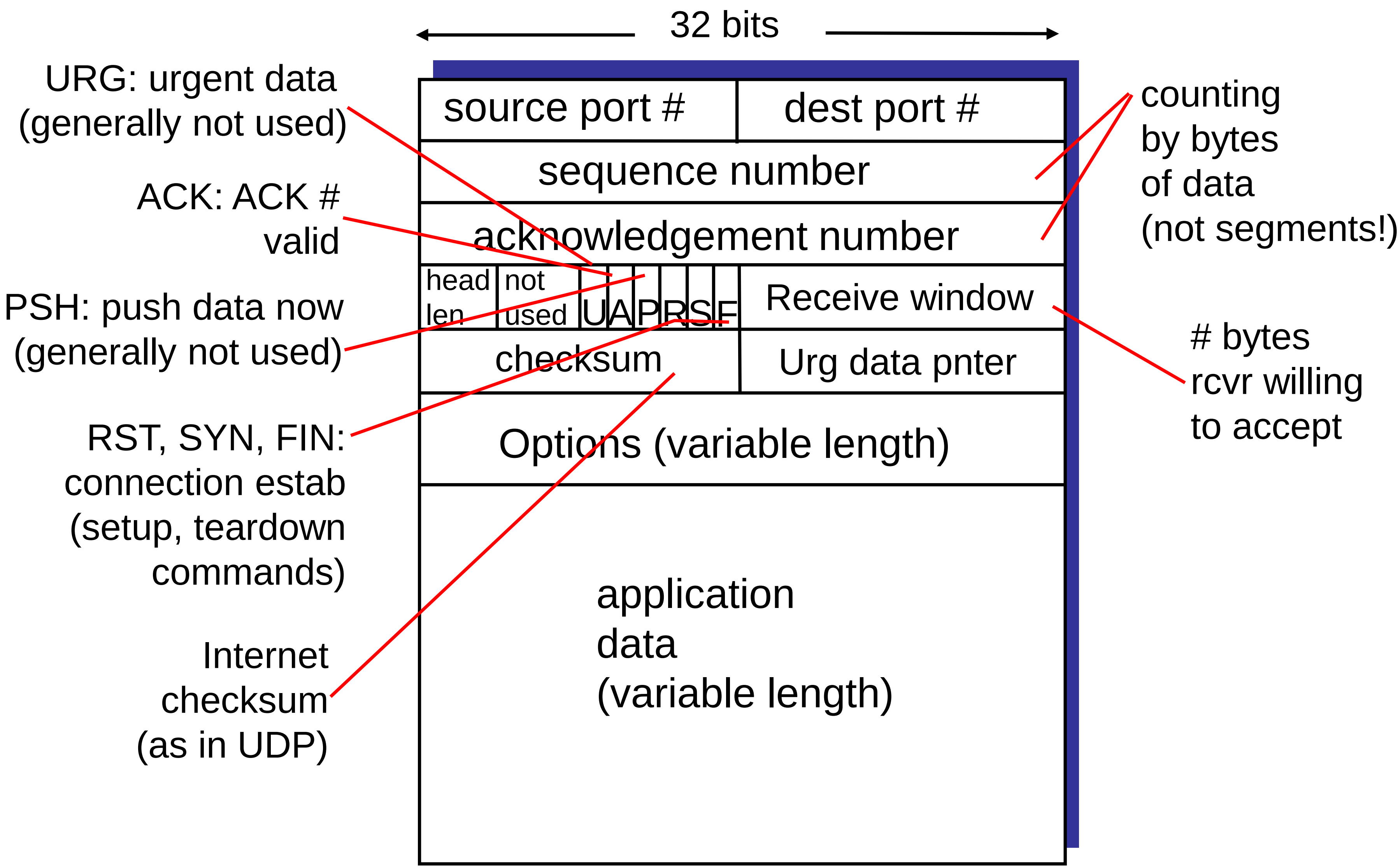
TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no “message boundaries”
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*

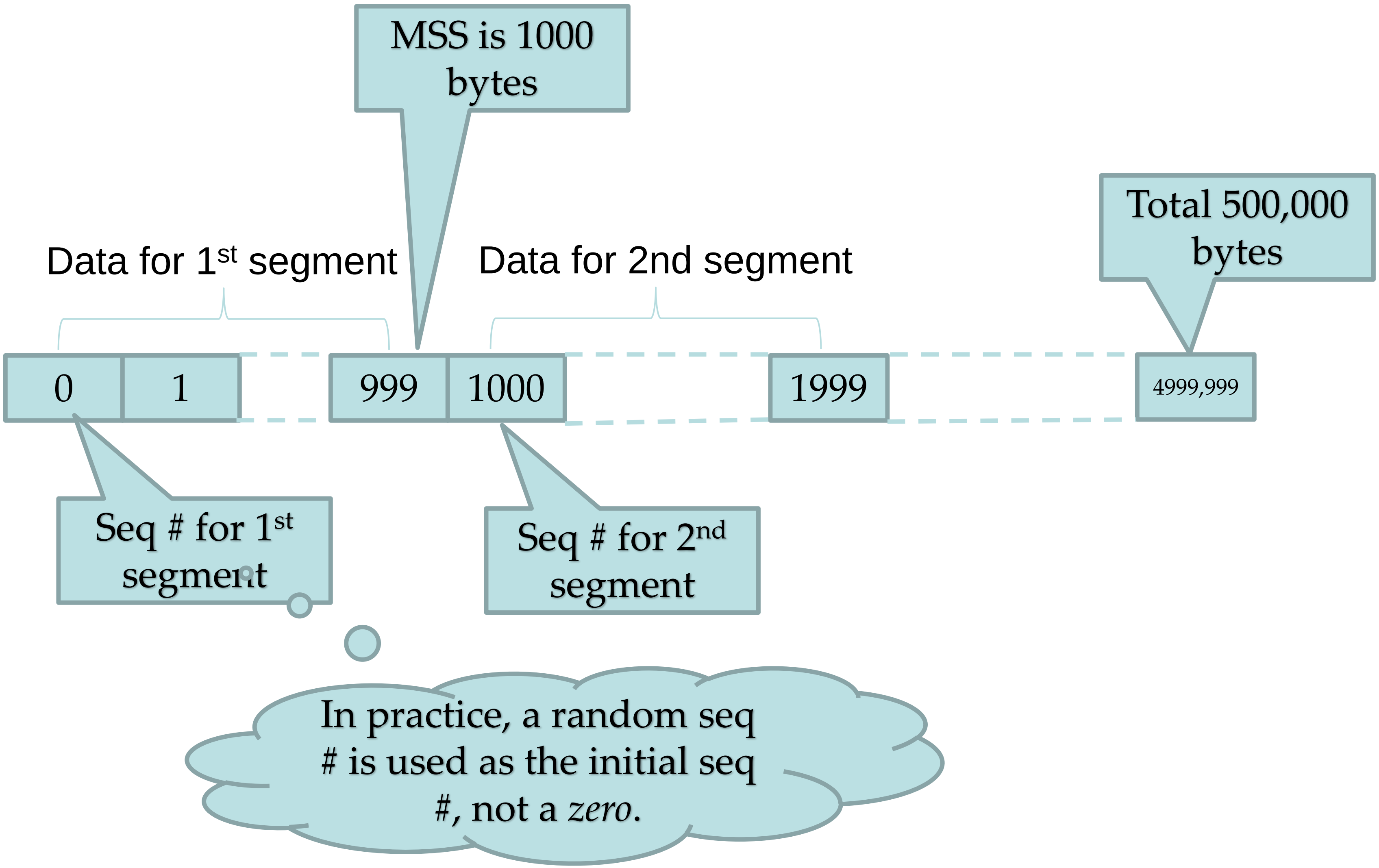


- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



TCP segments



TCP seq. #'s and ACKs

Seq. #'s:

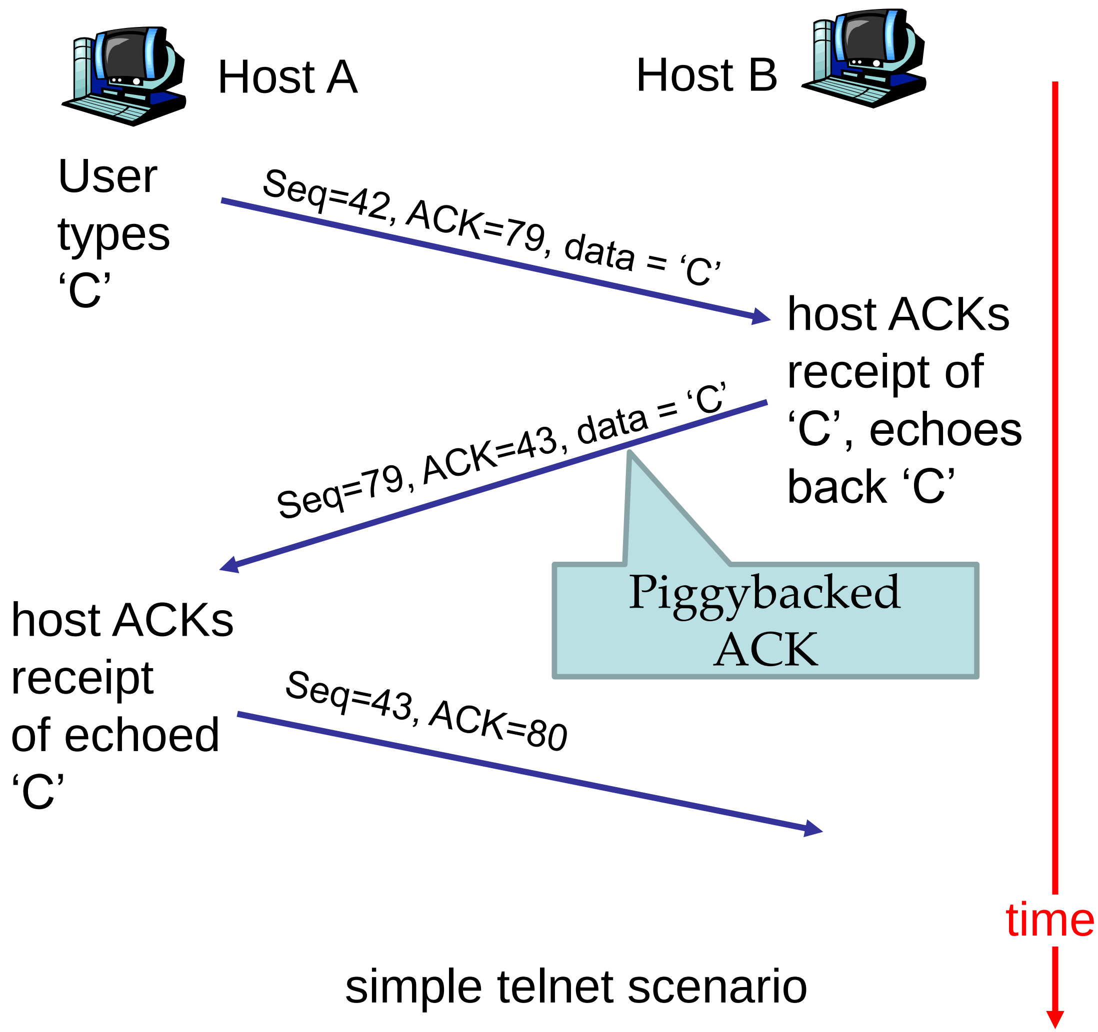
- byte stream "number" of first byte in segment's data

ACKs:

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

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



simple telnet scenario

TCP Connection Management

Three way handshake:

Step 1: client host sends TCP SYN segment to server

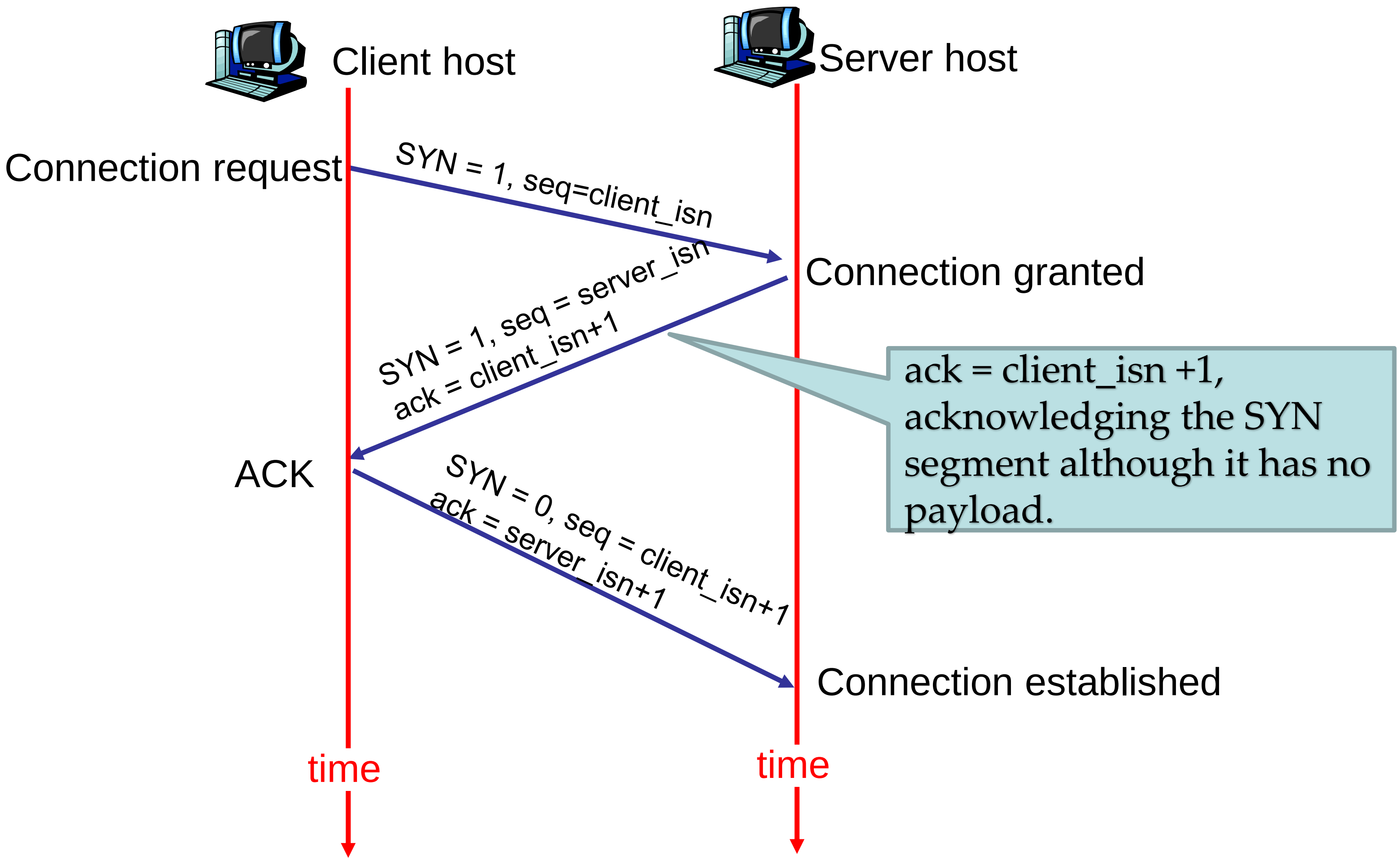
- specifies initial seq #
- no data

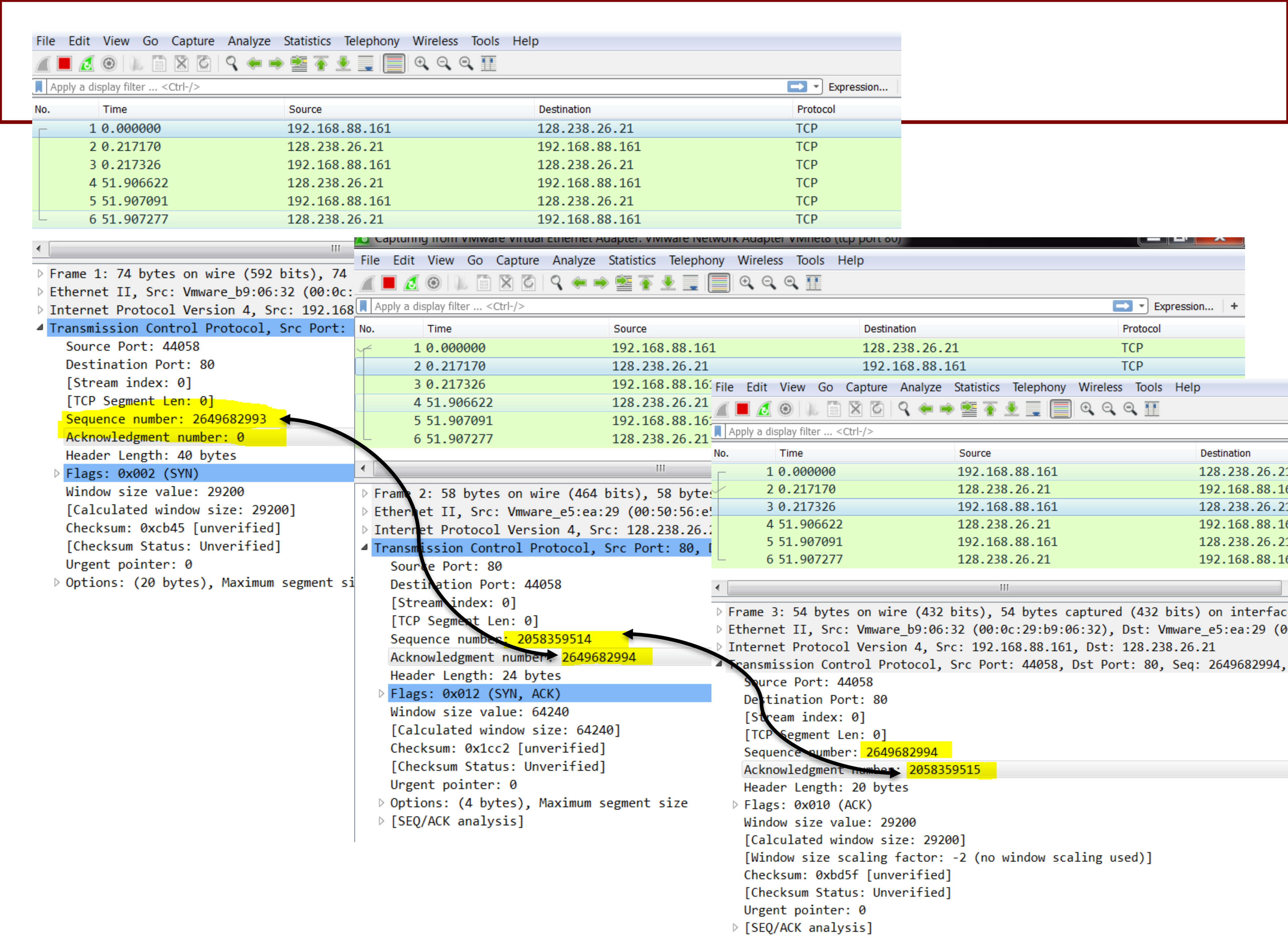
Step 2: server host receives SYN, replies with SYNACK segment

- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

TCP 3-way handshaking





File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol
1 0.000000 192.168.88.161 128.238.26.21 TCP
2 0.217170 128.238.26.21 192.168.88.161 TCP
3 0.217326 192.168.88.161 128.238.26.21 TCP
4 51.906622 128.238.26.21 192.168.88.161 TCP
5 51.907091 192.168.88.161 128.238.26.21 TCP
6 51.907277 128.238.26.21 192.168.88.161 TCP
Frame 1: 74 bytes on wire (592 bits), 74
Ethernet II, Src: Vmware_b9:06:32 (00:0c:
Internet Protocol Version 4, Src: 192.168
Transmission Control Protocol, Src Port:
Source Port: 44058
Destination Port: 80
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 2649682993
Acknowledgment number: 0
Header Length: 40 bytes
Flags: 0x002 (SYN)
Window size value: 29200
[Calculated window size: 29200]
Checksum: 0xcb45 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (20 bytes), Maximum segment si
Capturing from VMware Virtual Ethernet Adapter: VMware Network Adapter VMnet8 (tcp port 80)
Frame 2: 58 bytes on wire (464 bits), 58 byte
Ethernet II, Src: Vmware_e5:ea:29 (00:50:56:e
Internet Protocol Version 4, Src: 128.238.26.
Transmission Control Protocol, Src Port: 80, D
Source Port: 80
Destination Port: 44058
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 2058359514
Acknowledgment number: 2649682994
Header Length: 24 bytes
Flags: 0x012 (SYN, ACK)
Window size value: 64240
[Calculated window size: 64240]
Checksum: 0x1cc2 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (4 bytes), Maximum segment size
[SEQ/ACK analysis]
Frame 3: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interfac
Ethernet II, Src: Vmware_b9:06:32 (00:0c:29:b9:06:32), Dst: Vmware_e5:ea:29 (0
Internet Protocol Version 4, Src: 192.168.88.161, Dst: 128.238.26.21
Transmission Control Protocol, Src Port: 44058, Dst Port: 80, Seq: 2649682994,
Source Port: 44058
Destination Port: 80
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 2649682994
Acknowledgment number: 2058359515
Header Length: 20 bytes
Flags: 0x010 (ACK)
Window size value: 29200
[Calculated window size: 29200]
[Window size scaling factor: -2 (no window scaling used)]
Checksum: 0xbd5f [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]

TCP Connection Management (closing)

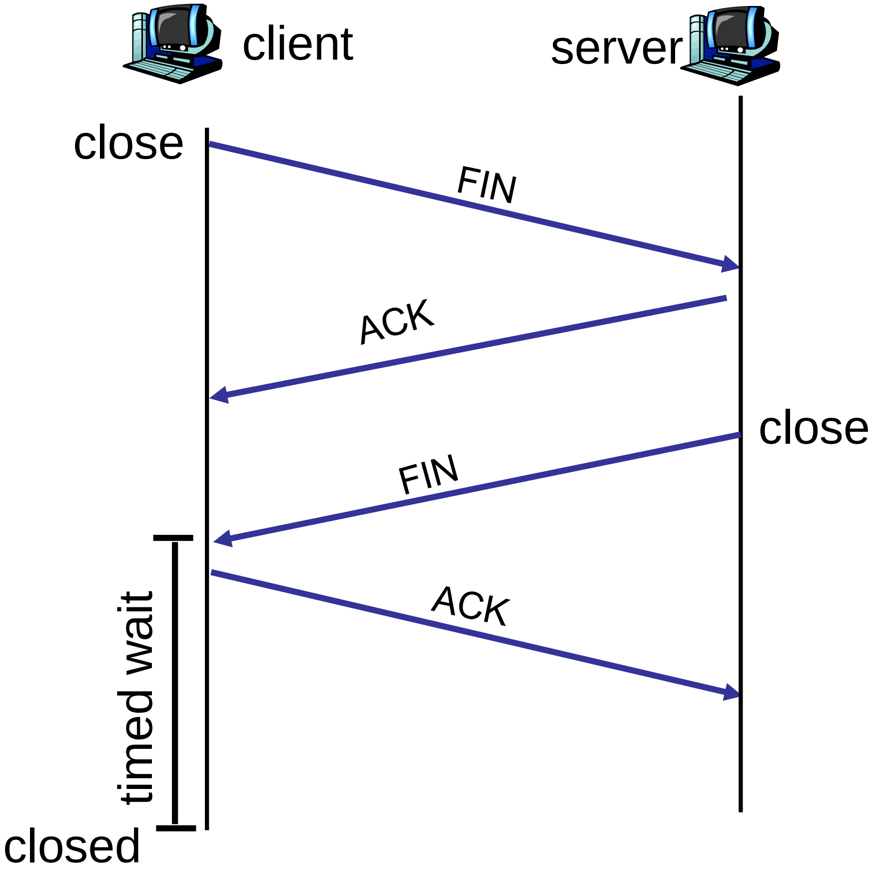
All good things must come to an end:

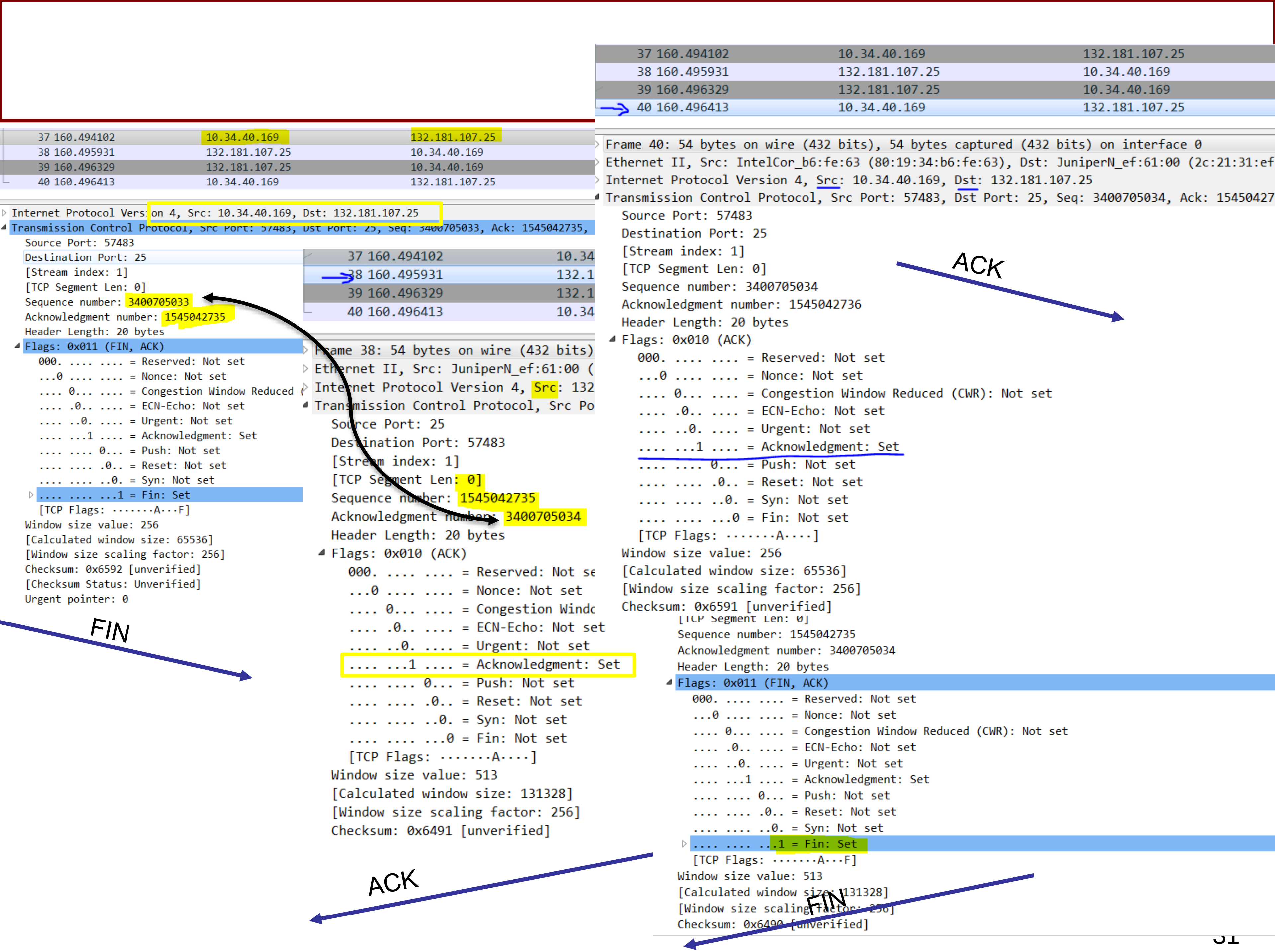
Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.

Step 3: client receives FIN, replies with ACK.

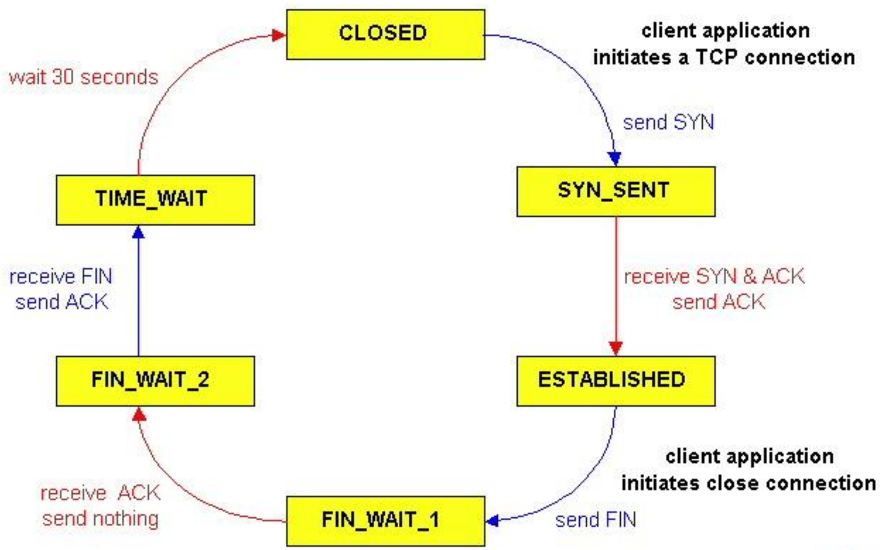
Step 4: server, receives ACK. Connection closed.





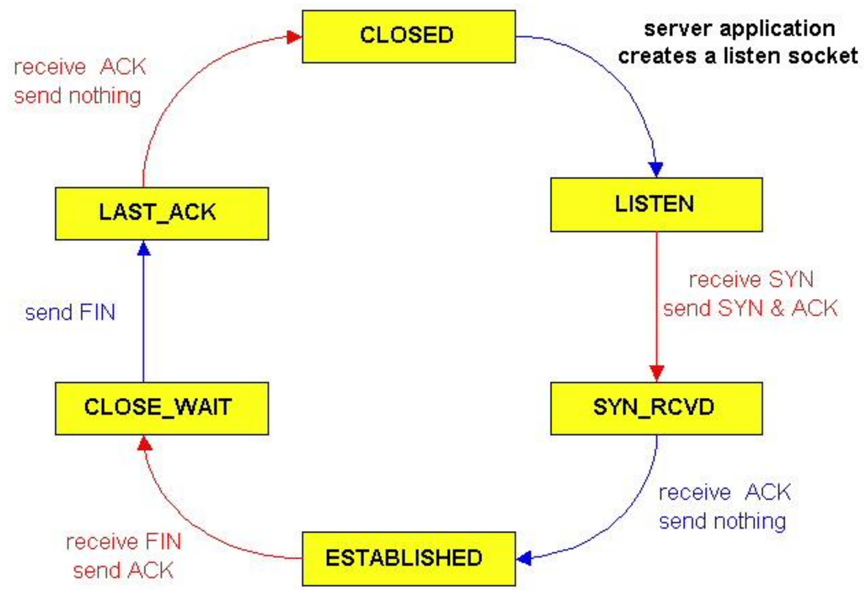
37 160.494102 10.34.40.169 132.181.107.25
38 160.495931 132.181.107.25 10.34.40.169
39 160.496329 132.181.107.25 10.34.40.169
40 160.496413 10.34.40.169 132.181.107.25
Internet Protocol Version 4, Src: 10.34.40.169, Dst: 132.181.107.25
Transmission Control Protocol, Src Port: 57483, Dst Port: 25, Seq: 3400705033, Ack: 1545042735,
Source Port: 57483
Destination Port: 25
[Stream index: 1]
[TCP Segment Len: 0]
Sequence number: 3400705033
Acknowledgment number: 1545042735
Header Length: 20 bytes
Flags: 0x011 (FIN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....1 = Fin: Set
[TCP Flags: ·······A···F]
Window size value: 256
[Calculated window size: 65536]
[Window size scaling factor: 256]
Checksum: 0x6592 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
FIN
Frame 38: 54 bytes on wire (432 bits)
Ethernet II, Src: JuniperN_ef:61:00 (
Internet Protocol Version 4, Src: 132
Transmission Control Protocol, Src Po
Source Port: 25
Destination Port: 57483
[Stream index: 1]
[TCP Segment Len: 0]
Sequence number: 1545042735
Acknowledgment number: 3400705034
Header Length: 20 bytes
Flags: 0x010 (ACK)
000. .... = Reserved: Not se
...0 .... = Nonce: Not set
.... 0... .... = Congestion Windc
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....0 = Fin: Not set
[TCP Flags: ·······A····]
Window size value: 513
[Calculated window size: 131328]
[Window size scaling factor: 256]
Checksum: 0x6491 [unverified]
ACK
Frame 40: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: IntelCor_b6:fe:63 (80:19:34:b6:fe:63), Dst: JuniperN_ef:61:00 (2c:21:31:ef
Internet Protocol Version 4, Src: 10.34.40.169, Dst: 132.181.107.25
Transmission Control Protocol, Src Port: 57483, Dst Port: 25, Seq: 3400705034, Ack: 15450427
Source Port: 57483
Destination Port: 25
[Stream index: 1]
[TCP Segment Len: 0]
Sequence number: 3400705034
Acknowledgment number: 1545042736
Header Length: 20 bytes
Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....0 = Fin: Not set
[TCP Flags: ·······A····]
Window size value: 256
[Calculated window size: 65536]
[Window size scaling factor: 256]
Checksum: 0x6591 [unverified]
ACK
Sequence number: 1545042735
Acknowledgment number: 3400705034
Header Length: 20 bytes
Flags: 0x011 (FIN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....1 = Fin: Set
[TCP Flags: ·······A···F]
Window size value: 513
[Calculated window size: 131328]
[Window size scaling factor: 256]
Checksum: 0x6490 [unverified]
FIN

TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



Outline

- Connection-oriented transport: TCP
  - Overview and segment structure
  - Connection management
  - Reliable data transfer
  - Flow control

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim’s slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic’s lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm