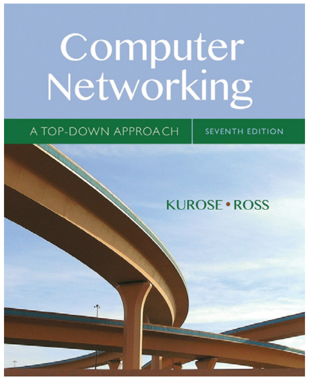
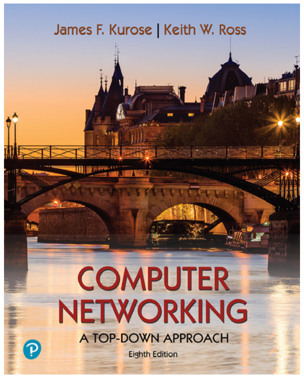
Chapter 3
Transport Layer

Courtesy to the textbooks' authors and Pearson Addison-Wesley because many slides are adapted from the following textbooks and their associated slides.



Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2016.



Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 8th Edition, Pearson, 2020.

Transport layer: overview

Our goal:

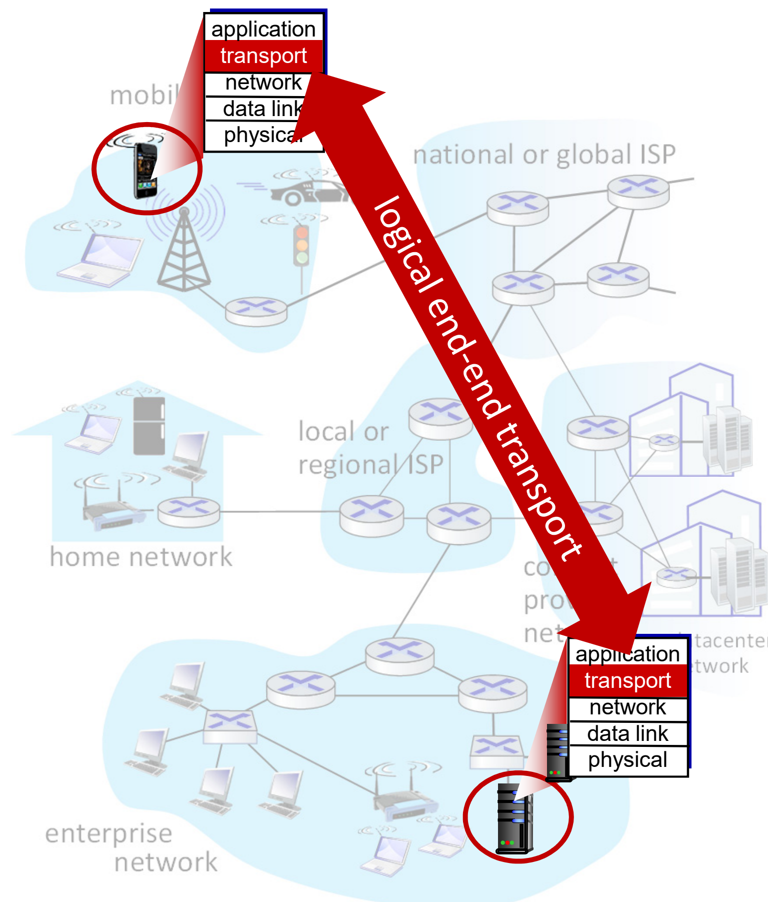
- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Transport layer: roadmap

- Review of transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- ≥2 transport protocols available to Internet applications
  - TCP, UDP



Transport vs. network layer services and protocols

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

household analogy:

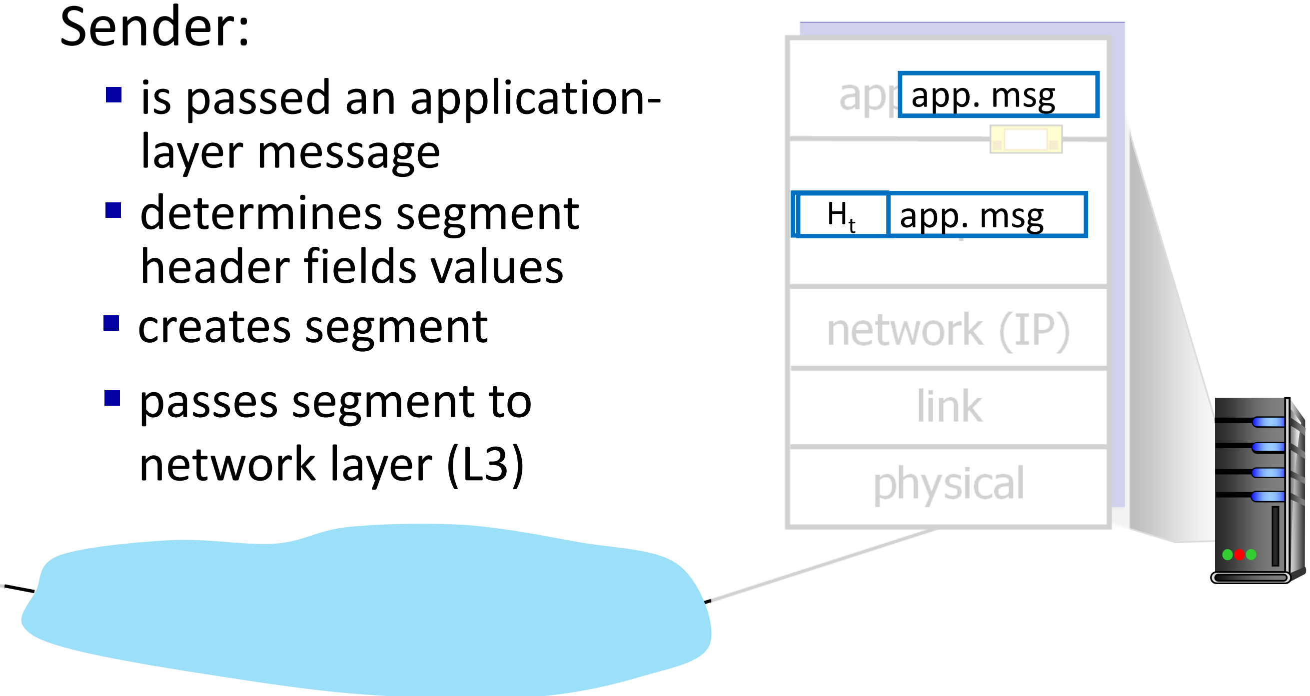
There are multiple persons (Alice, Bob, …) in a house:

- hosts = houses
- processes = persons
- network-layer protocol = postal service
  - to a certain house
- transport protocol = in-site manager
  - to a certain in-house person

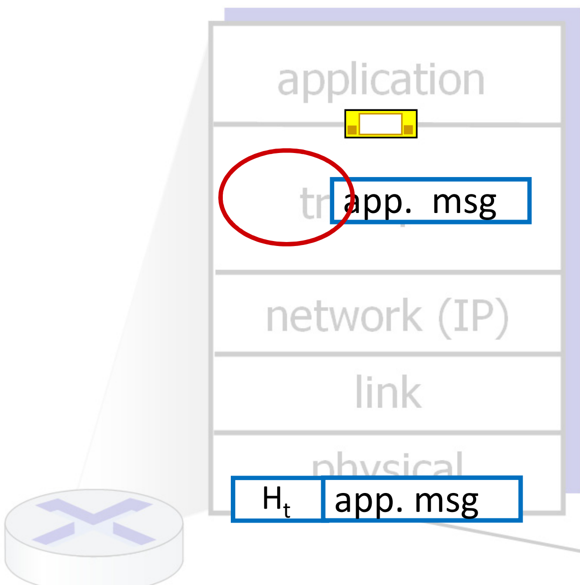
Transport Layer Actions

Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to network layer (L3)

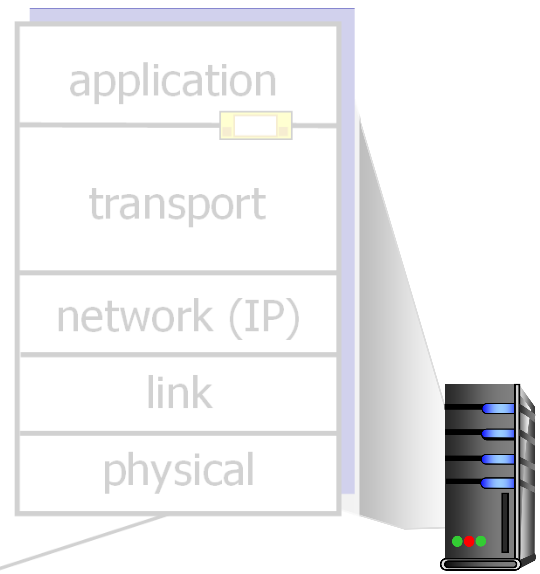


Transport Layer Actions



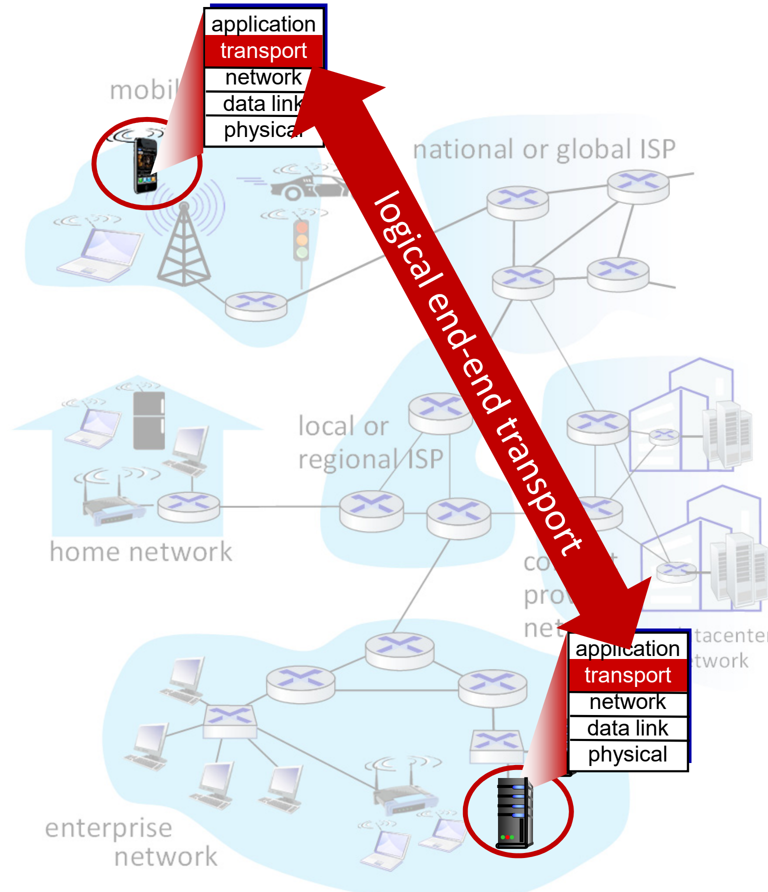
Receiver:

- receives segment from L3
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



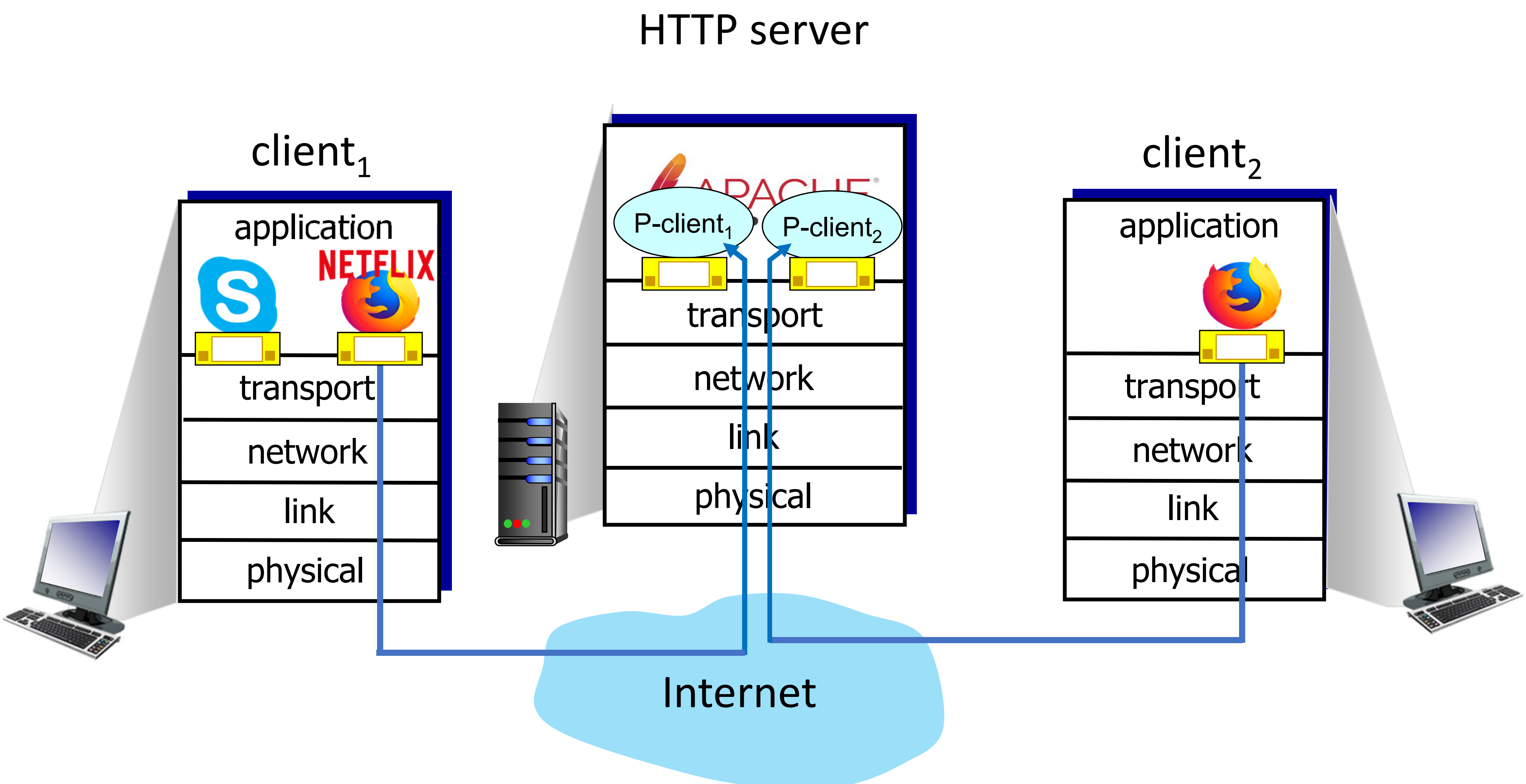
Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Chapter 3: roadmap

- Review of transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality


HTTP server
client1
client2
application
transport
network
link
physical
P-client1
P-client2
application
transport
network
link
physical
application
transport
network
link
physical
Internet

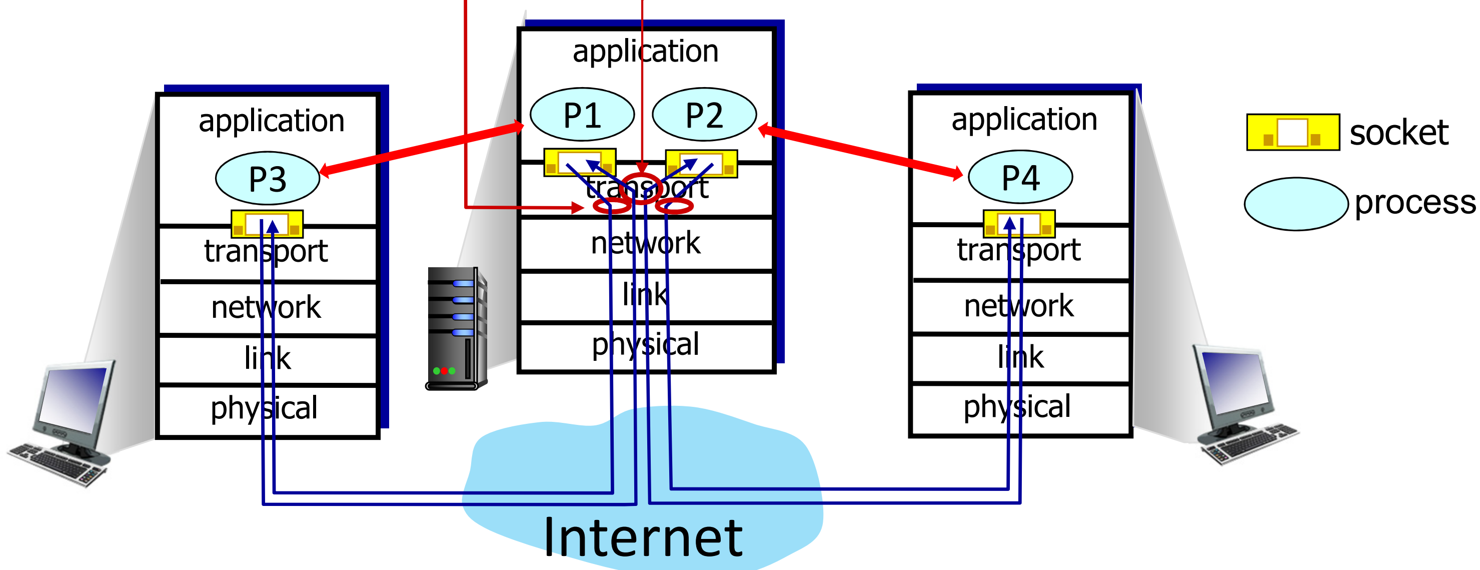
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

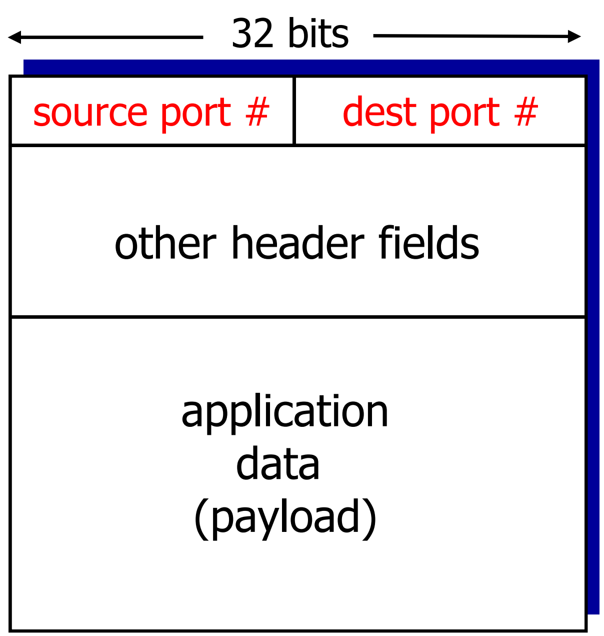
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

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



TCP/UDP segment format

Connectionless (UDP) demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

- when app at the source side creates datagram to send into UDP socket, it must specify
  - destination IP address
  - destination port #

- when receiving host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

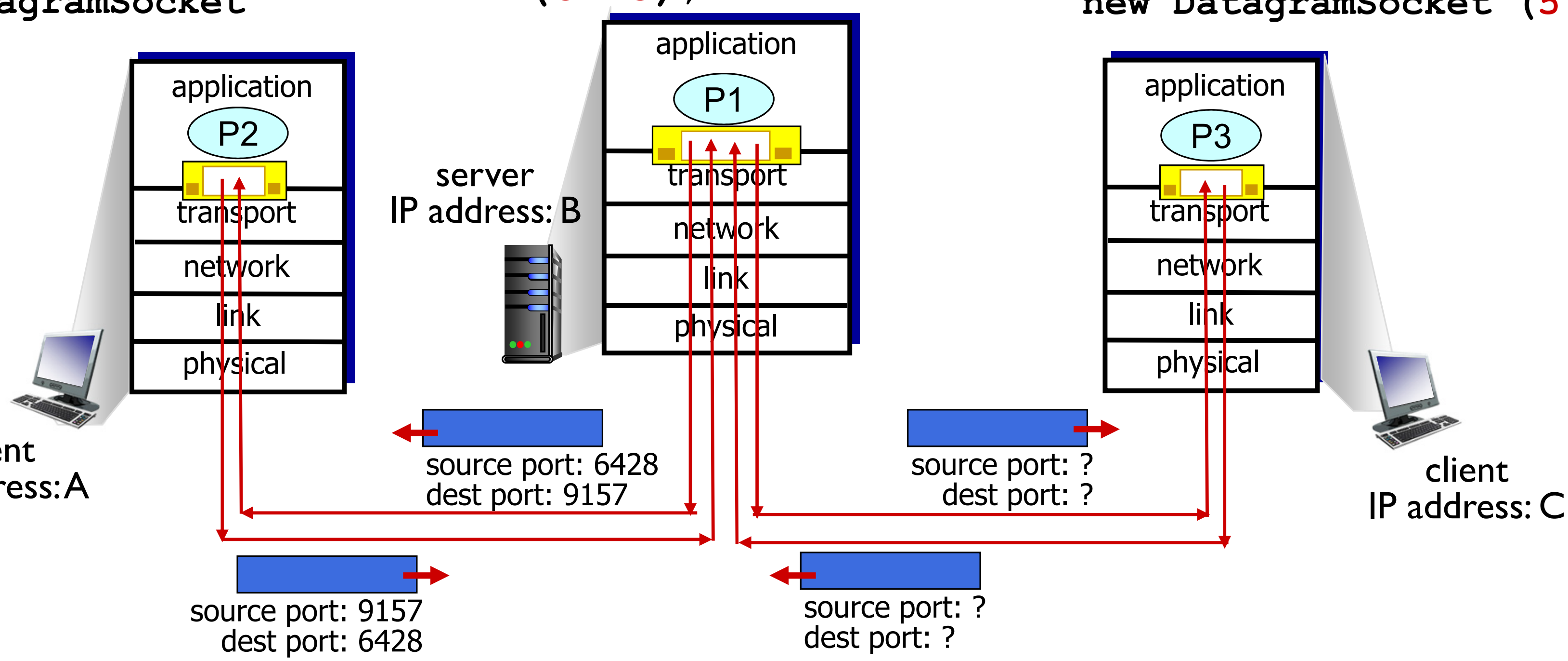
UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

`DatagramSocket serverSocket = new DatagramSocket (6428);`

`DatagramSocket mySocket2 = new DatagramSocket (9157);`

`DatagramSocket mySocket1 = new DatagramSocket (5775);`

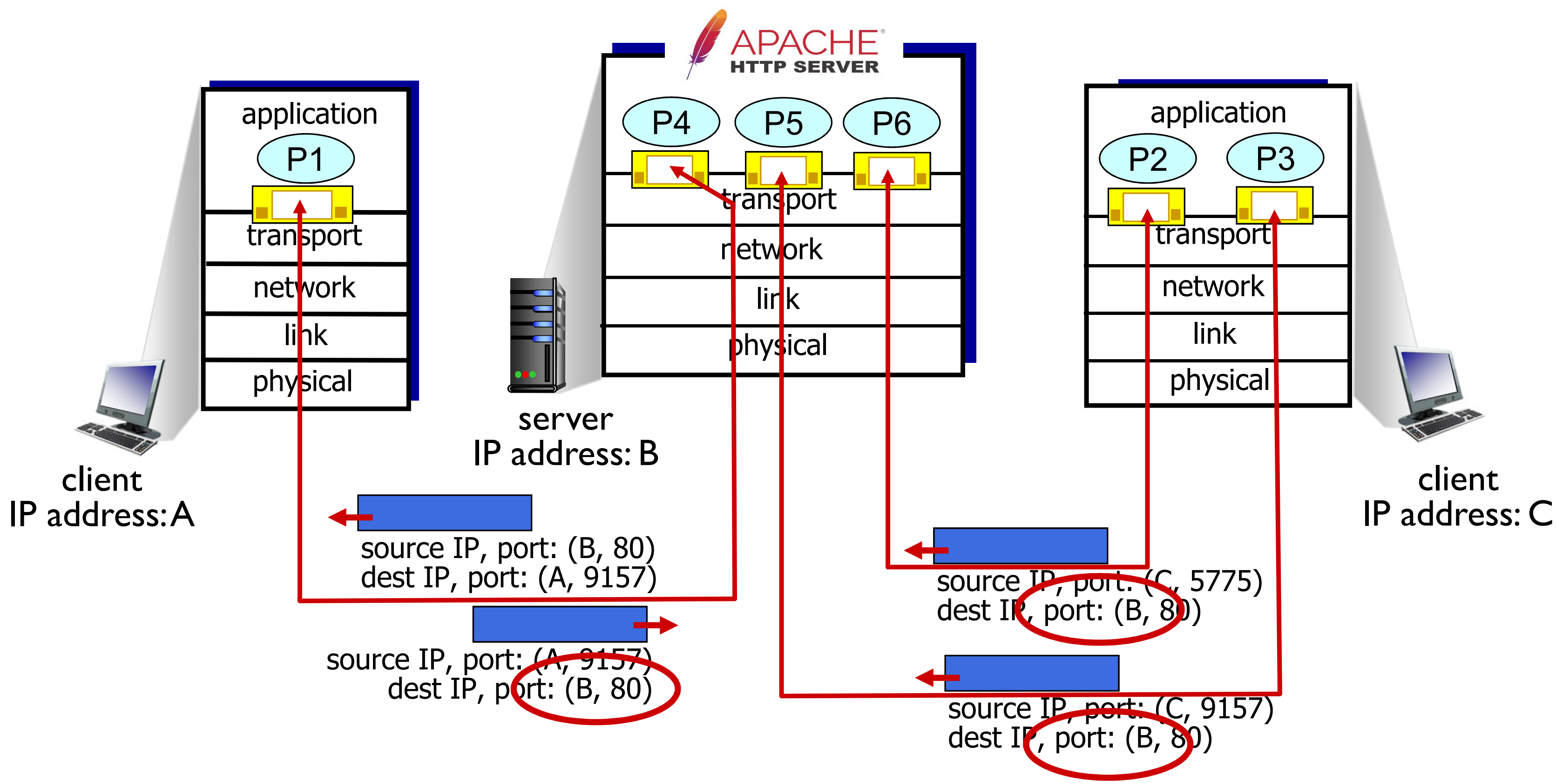


Connection-oriented (TCP) demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Connection-oriented demultiplexing: example



Three segments, all destined to (B,80) are demultiplexed to *different* sockets

Summary

- Multiplexing and demultiplexing are based on header field values
- **UDP:** destination host demultiplexes segments using destination port number (only)
  - a UDP socket is uniquely identified by (dest IP, dest port #)
- **TCP:** demultiplexing using 4-tuple
  - IP addresses and port numbers of source and destination
- Multiplexing/demultiplexing happen at *all* layers

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

UDP: User Datagram Protocol

- "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 and receiver
  - each UDP segment is handled independently of others

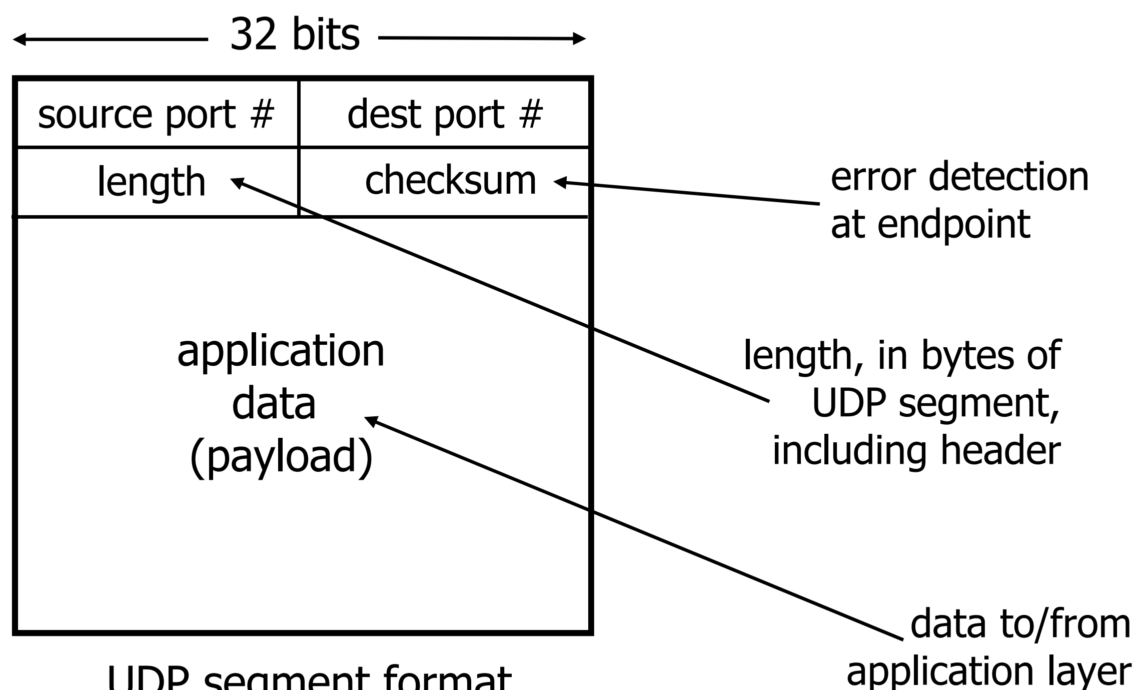
Why is there a UDP?

- no connection establishment
  - which can add RTT delay
- simple (no connection state at endpoint)
  - server can support more active clients when running over UDP
- small header overhead
- finer application-layer control
  - no congestion control
  - UDP can blast away as fast as desired!

UDP: User Datagram Protocol

- UDP was ratified in 1980. [RFC 768]
- UDP is used by:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
  - HTTP/3 (over QUIC over UDP)
- if reliable transfer needed over UDP (e.g., HTTP/3):
  - add needed reliability at application layer
  - add congestion control at application layer

UDP segment format



UDP segment format

UDP checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

receiver-computed checksum ≠ sender-computed checksum (as received)

UDP checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of UDP segment as sequence of 16-bit integers
  - including pseudo header (IP addresses, …)
- checksum:
  - add segment content (1's complement sum)
  - and then take 1's complement
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check whether computed checksum equals checksum field value:
  - Not equal – error detected
  - Equal – no error detected. *But maybe errors nonetheless?* More later ….

Internet checksum: an example

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

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Internet checksum: weak protection!

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

0 1

1 0

Even though numbers have changed (bit flips), *no* change in checksum!

Summary: UDP

- "no frills" protocol:
  - segments may be lost, delivered out of order
  - best effort service: "send and hope for the best"
- UDP has its pluses:
  - no setup/handshaking needed (no RTT incurred)
  - smaller header overhead
  - finer application-layer control
  - helps with reliability (checksum)
- If needed, additional functionality can be built on top of UDP in application layer (e.g., HTTP/3)