UNIVERSITY OF WATERLOO

CS 456/656 Computer Networks

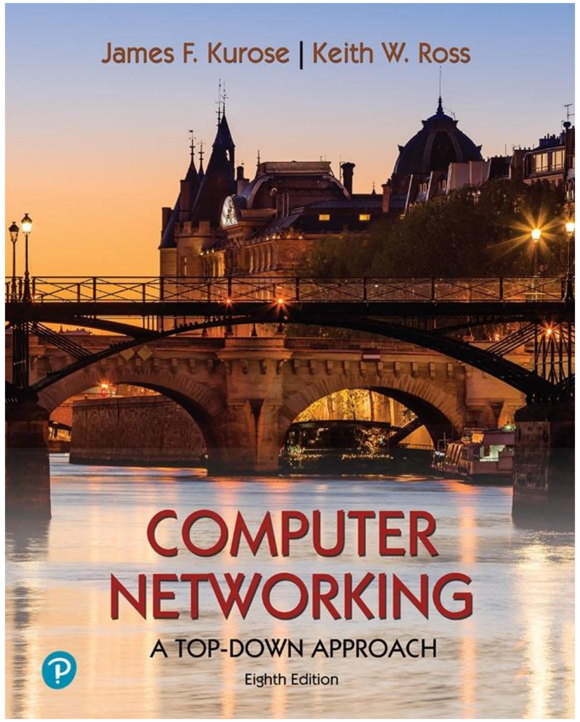
Lecture 5: Transport Layer – Part 1

Mina Tahmasbi Arashloo and Bo Sun

Fall 2024

A note on the slides

Adapted from the slides that accompany this book.



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Transport layer: roadmap

- Transport-layer overview
- 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 layer: roadmap

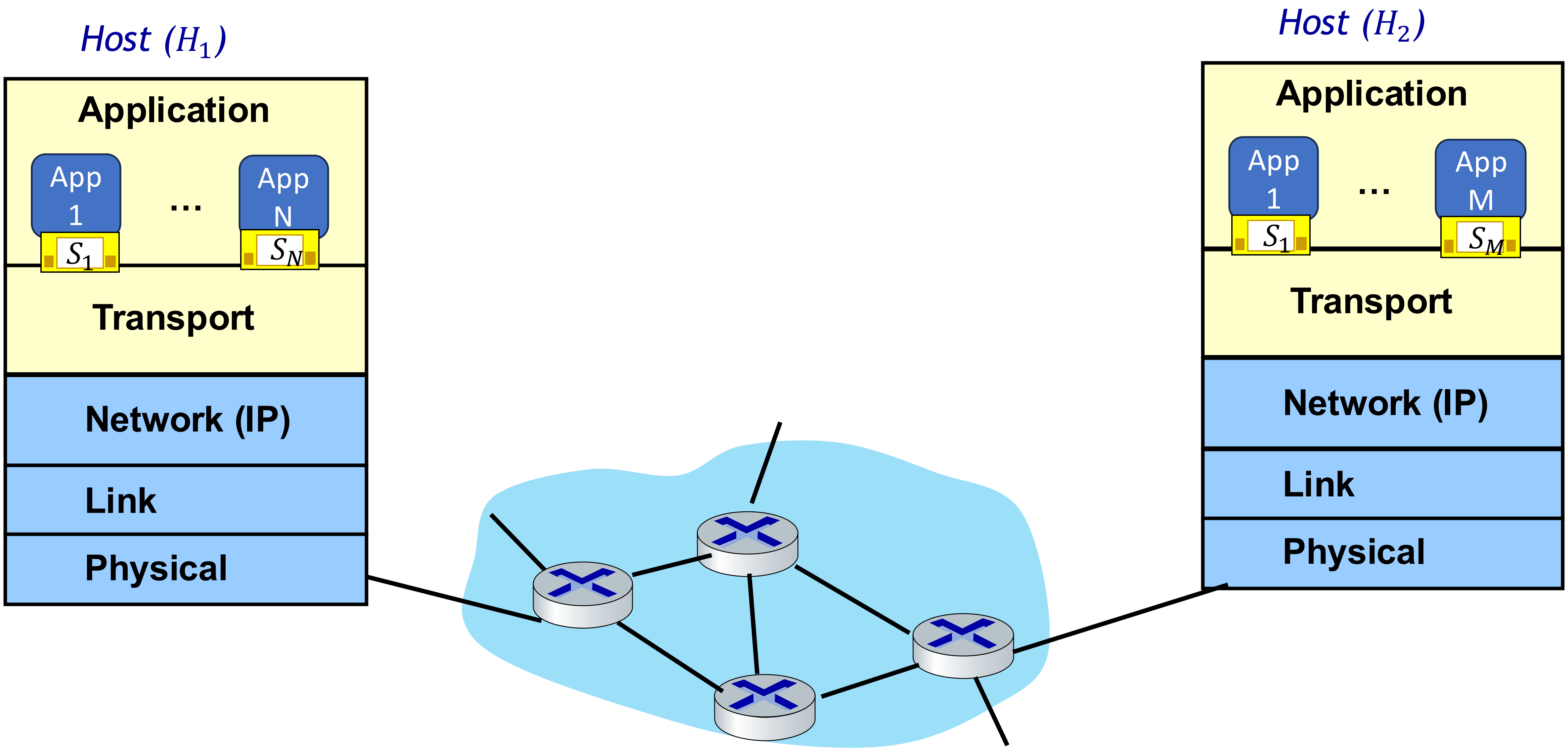
- Transport-layer overview
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Transport layer: overview

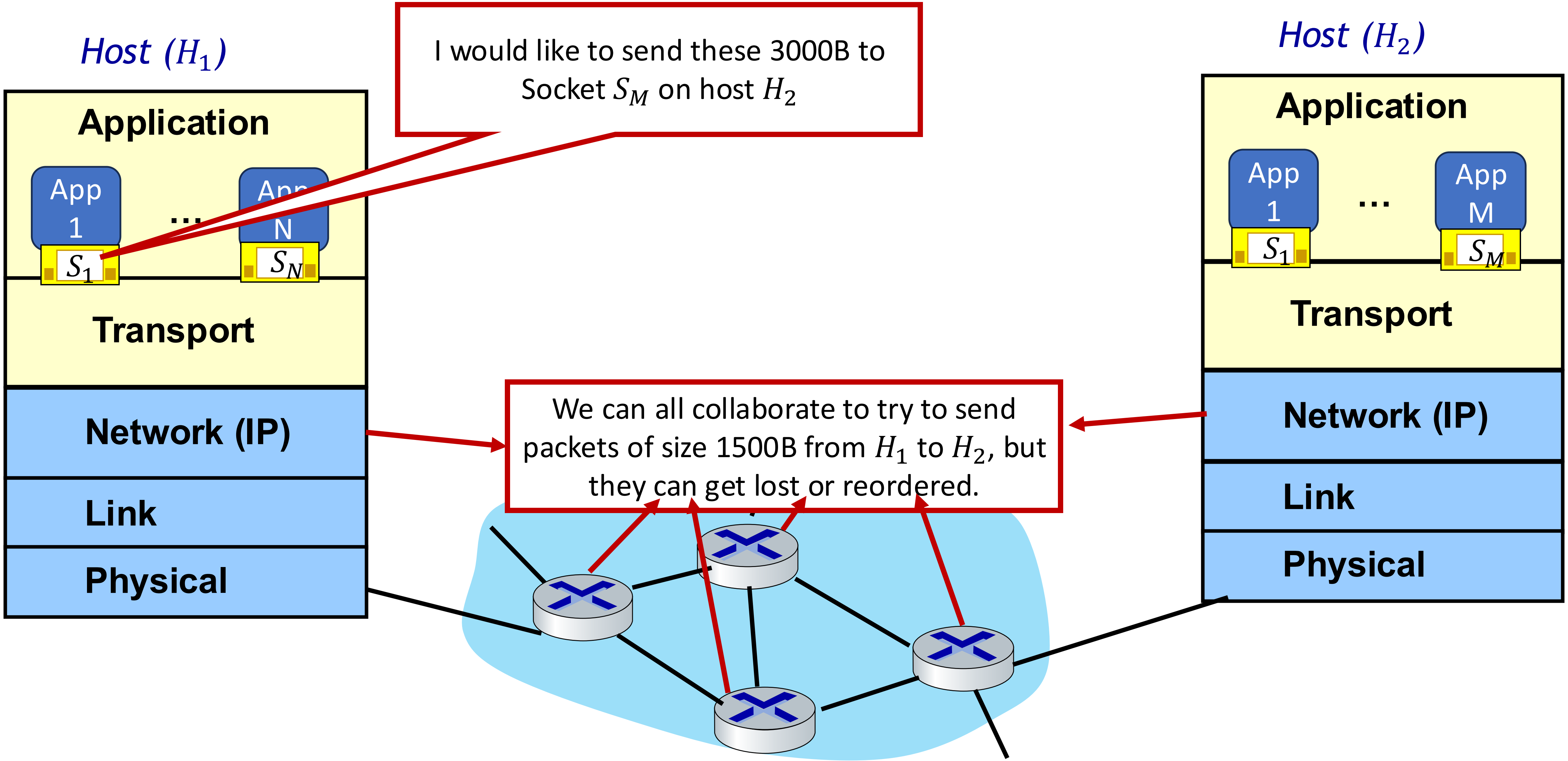
- Provide service to the application layer
  - Transport to Application: “If you give me some data and the ID of the other communication endpoint (e.g., the (IP, port) for the destination socket), I will get the data to that communication endpoint.”

- Using the services of the network layer
  - Network to transport: “If you give me some data and the ID of the computer (host) that is the destination (e.g., the IP address for the host), I will get the data to that destination host."

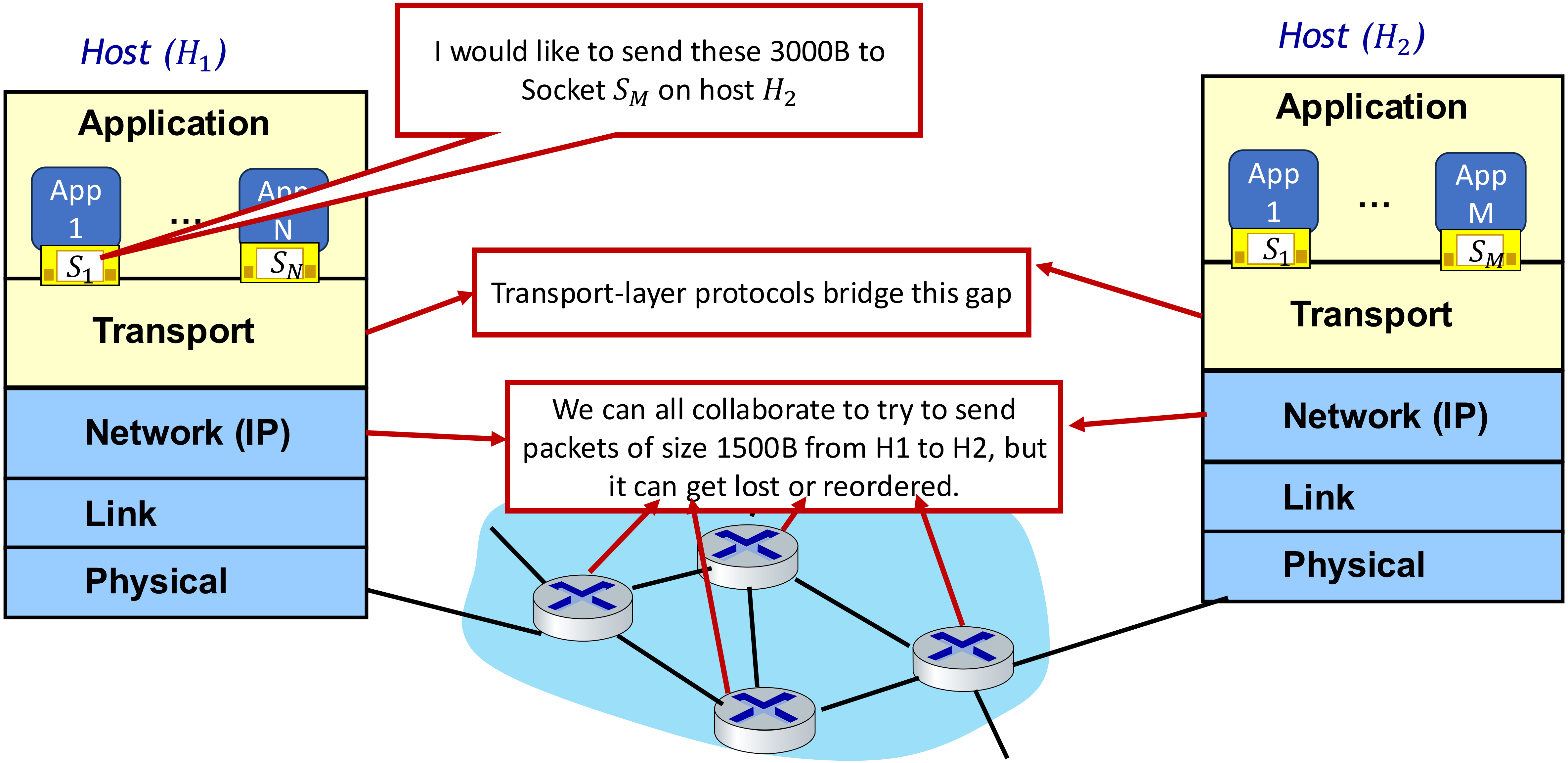
Transport layer: in the Internet



Transport layer: in the Internet



Transport layer: in the Internet



Transport layer: in the Internet

- Application on host H_1: send these 3000B through socket S_1 to socket S_M on host H_2
- The network layer: I'll do my best to get packets of size 1500B from H_1 to H_2, but it may get lost or corrupted, or get to H_2 later than some earlier packets you send from H_1.
- Transport-layer protocol:
  - How can I distinguish between traffic from different sockets?
  - How do I break data into packets and put it back together?
  - How do I make sure all bytes are delivered reliably?

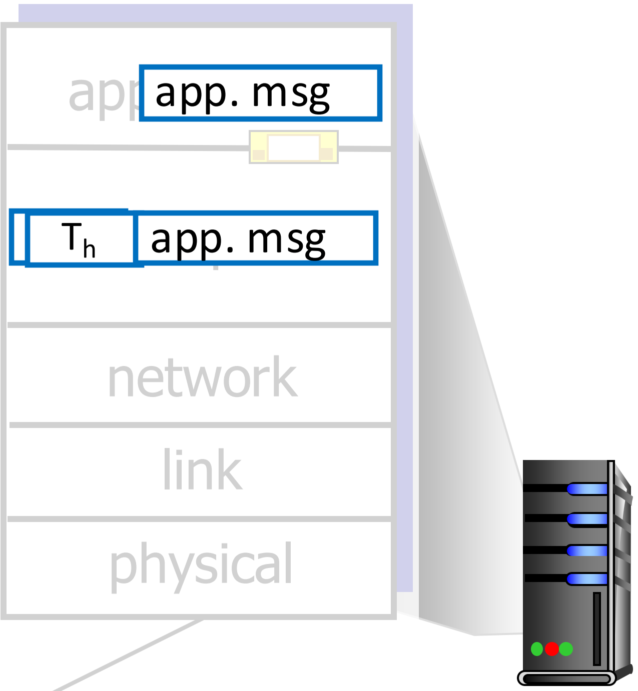
Transport layer: in the Internet

- Application on host H_1: send these 3000B through socket S_1 to socket S_M on host H_2
- The network layer: I'll do my best to get packets of size 1500B from H_1 to H_2, but it may get lost or corrupted, or get to H_2 later than some earlier packets you send from H_1.
- Transport-layer protocol:
  - How can I distinguish between traffic from different sockets?
  - Port numbers, Multiplexing and Demultiplexing
  - How do I break data into packets and put it back together?
  - Segmentation and reassembly
  - How do I make sure all bytes are delivered reliably?
  - Reliable data transfer

Transport Layer: in the Internet

Sender:

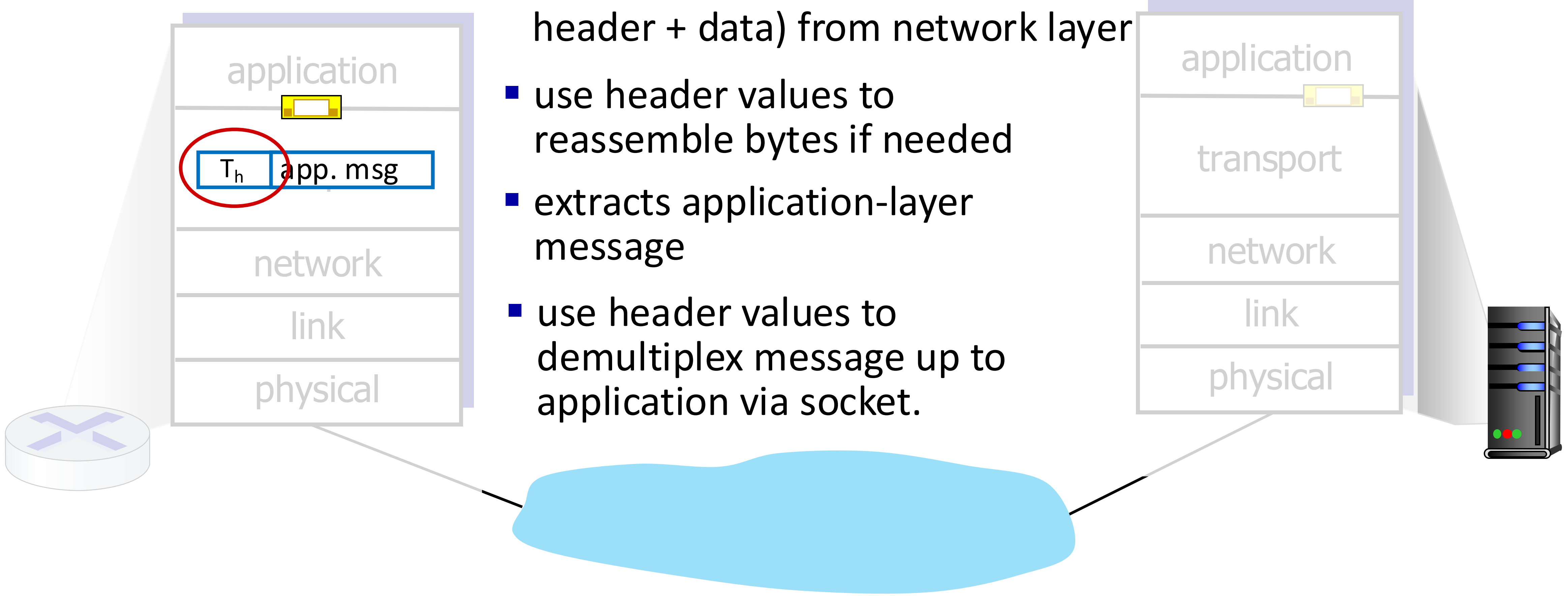
- is passed an application-layer message
- Attached its own metadata (header) to help with (de)mux, segmentation and reassembly, and reliable data delivery
- creates segment (transport header + data)
- passes segment to the network layer



Transport Layer: in the Internet

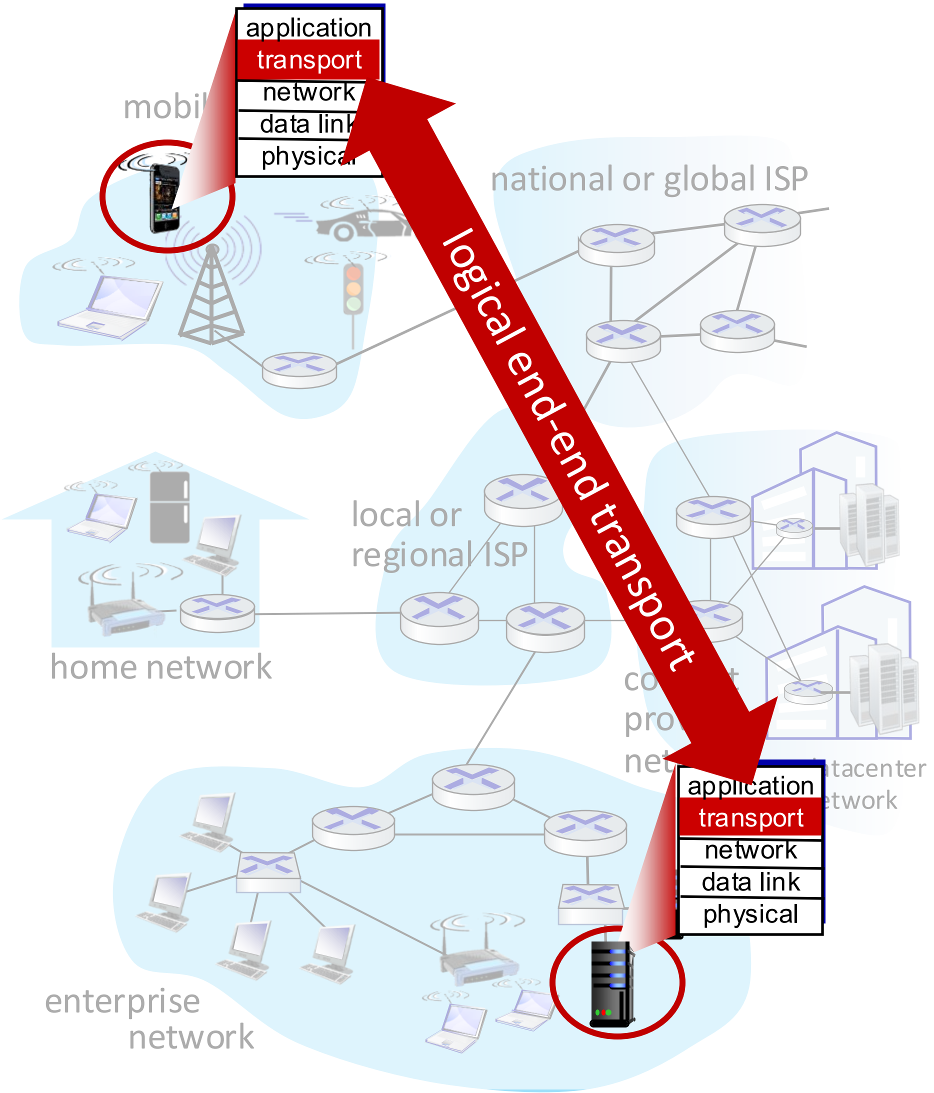
Receiver:

- receives segment (transport header + data) from network layer
- use header values to reassemble bytes if needed
- extracts application-layer message
- use header values to demultiplex message up to application via socket.



Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - segmentation and reassembly
  - reliable, in-order delivery
- **UDP:** User Datagram Protocol
  - no segmentation and reassembly
  - no reliability or ordering guarantees
  - no-frills extension of "best-effort" IP protocol in the network layer
- Both do mux and demux between sockets
- services *not* available:
  - delay guarantees
  - bandwidth guarantees



Transport layer: roadmap

- Transport-layer overview
- 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

Multiplexing/demultiplexing

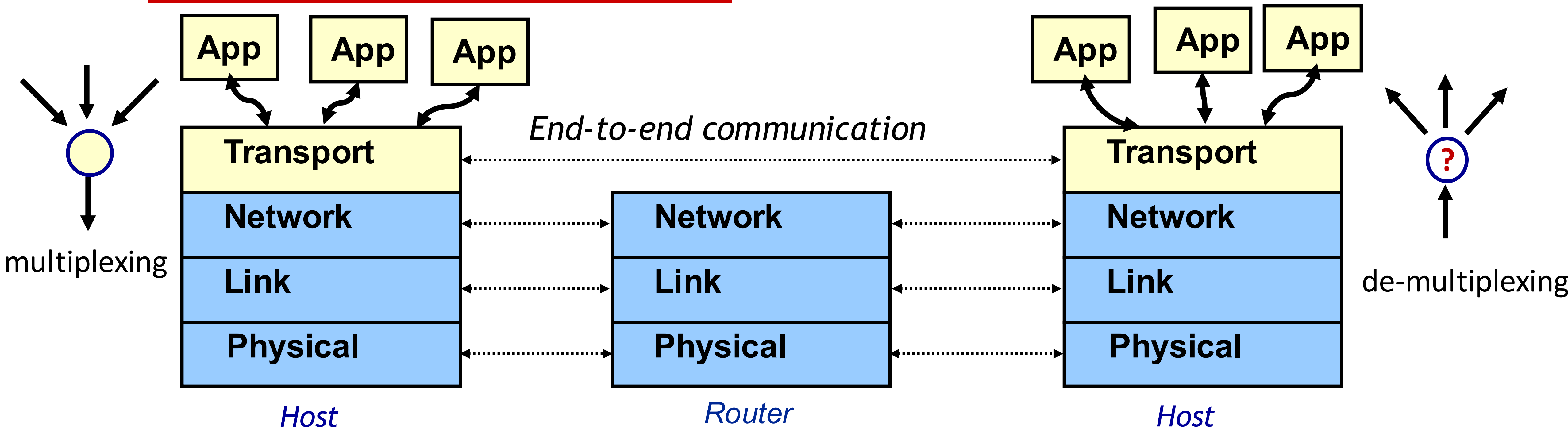
Need an extra identifier!

multiplexing as sender:

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

demultiplexing as receiver:

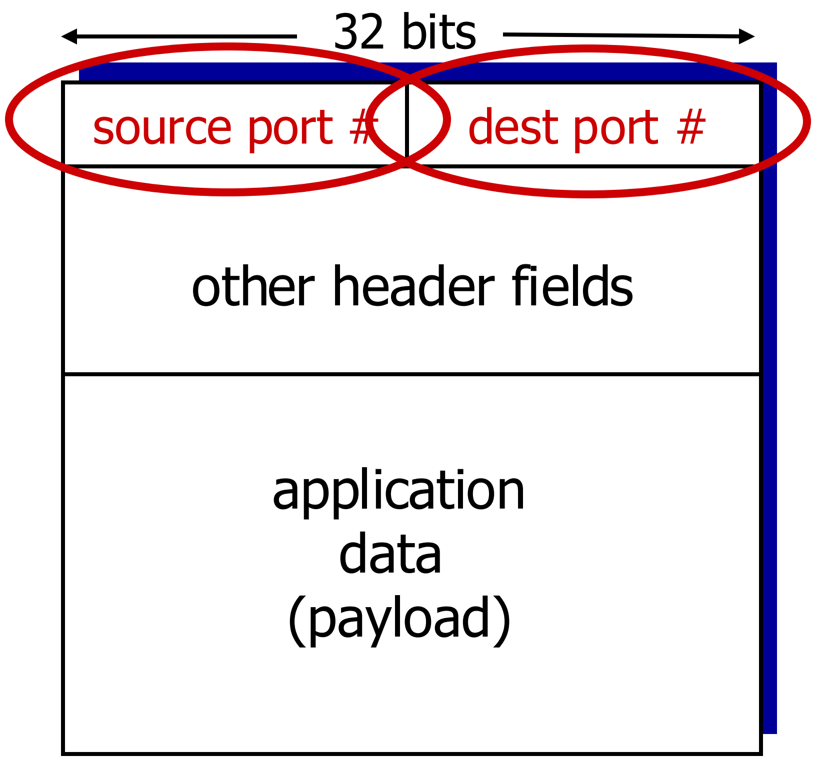
use header info to deliver received segments to correct socket




Multiplexing

How demultiplexing works

- host receives datagrams
  - each datagram has source network address, destination network address (e.g., IP addresses)
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *network addresses (e.g., IP addresses) & port numbers* to direct segments to appropriate sockets



TCP/UDP segment format

Connectionless demultiplexing

Recall:

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

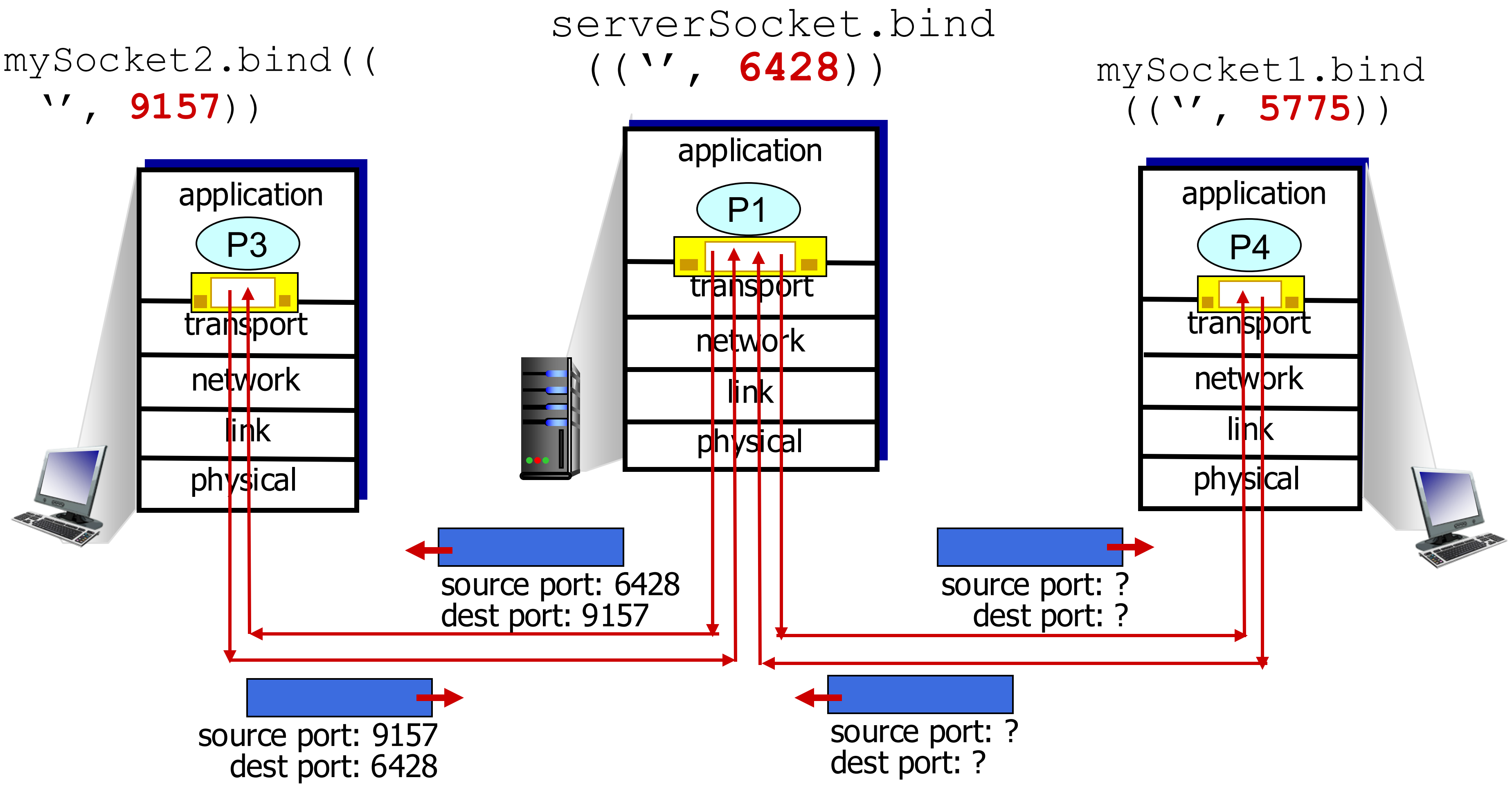
```
serverSocket.bind(('', 12000))
```

- when sending data into UDP socket, must specify
  - destination IP address
  - destination port #
- UDP sockets are identified with a pair of IP and port

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

IP/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

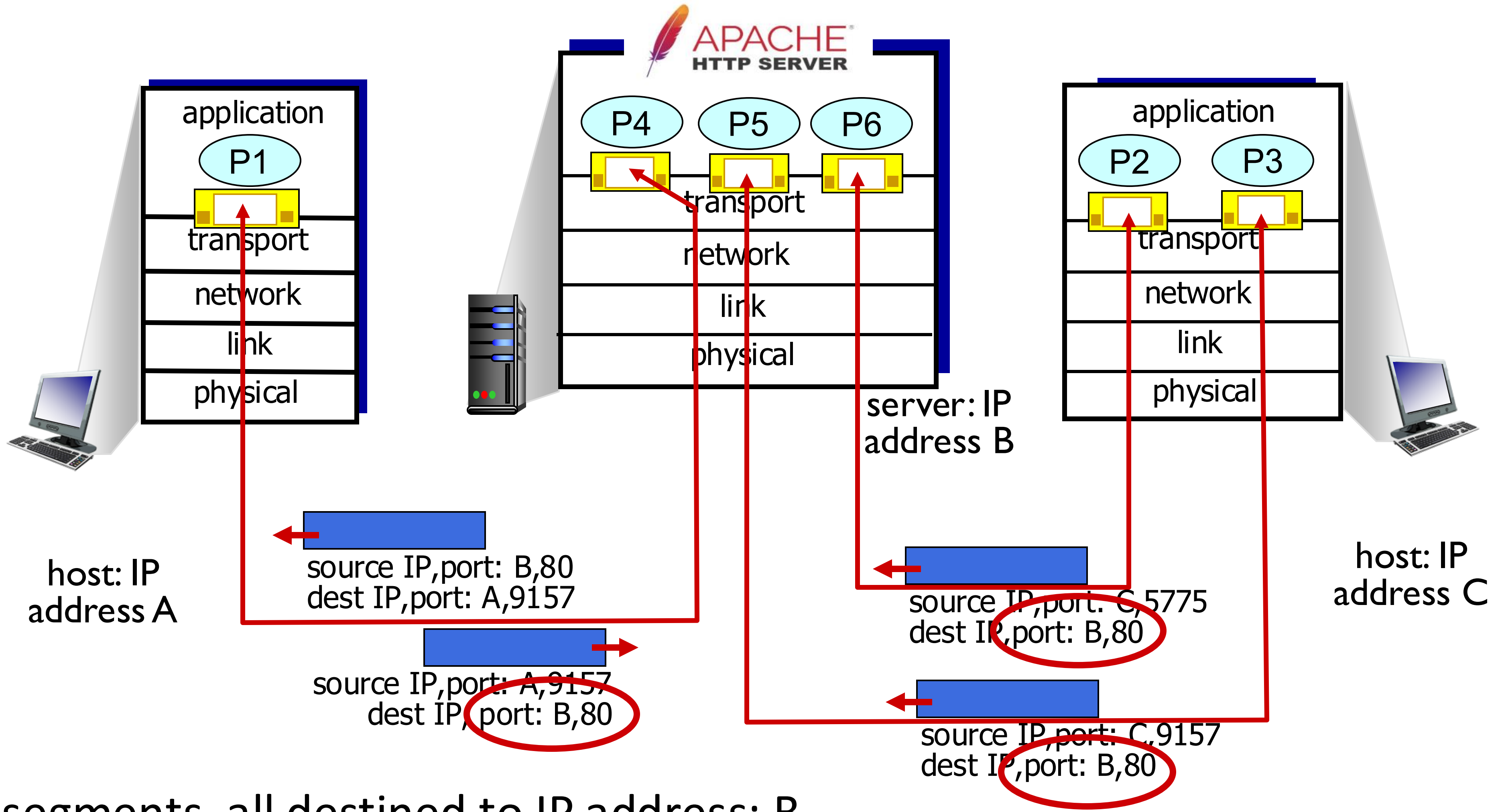


Connection-oriented demultiplexing

- 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 IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Summary of (de)multiplexing

- Multiplexing, demultiplexing: based on transport segment and network datagram header field values
- **UDP:** demultiplexing at the destination host using destination IP and port number (only)
- **TCP:** demultiplexing at the destination host using 4-tuple: source and destination IP addresses, and port numbers

Transport layer: roadmap

- Transport-layer overview
- 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

- How does UDP distinguish between traffic from different sockets?
  - Already covered in (de)multiplexing section
- How does UDP break data into packets and put it back together ?
  - It doesn't! You can only put as much data into a UDP segment that will fit into a single packet. Otherwise, it will give the application an error.
- How does UDP make sure all bytes are delivered reliably?
  - It doesn't!
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be lost or delivered out-of-order to app

Why do we have UDP again?

UDP: User Datagram Protocol

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - HTTP/3
  - Other network apps or protocols like DNS and SNMP (discussed later)
- if reliable transfer needed over UDP:
  - add needed reliability at application layer

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel
ISI

28 August 1980

User Datagram Protocol

Introduction

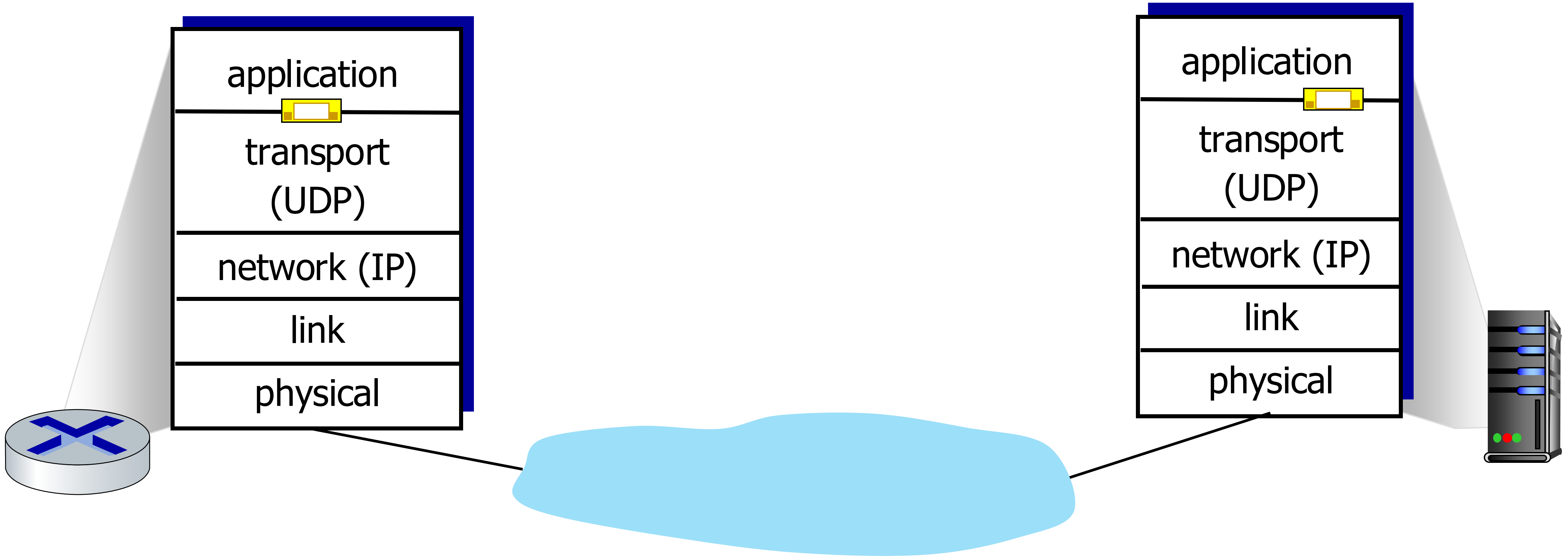
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

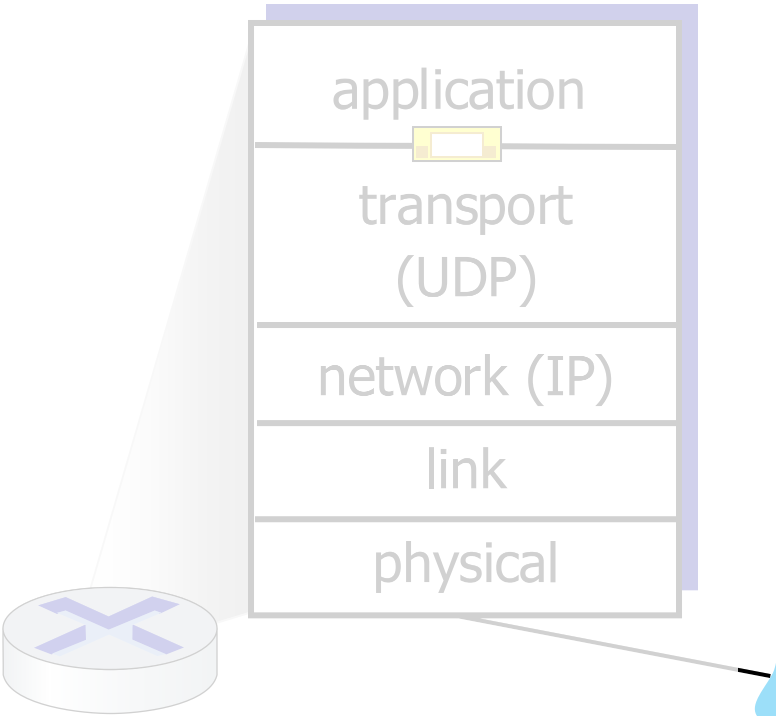
Format

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

UDP: Transport Layer Actions

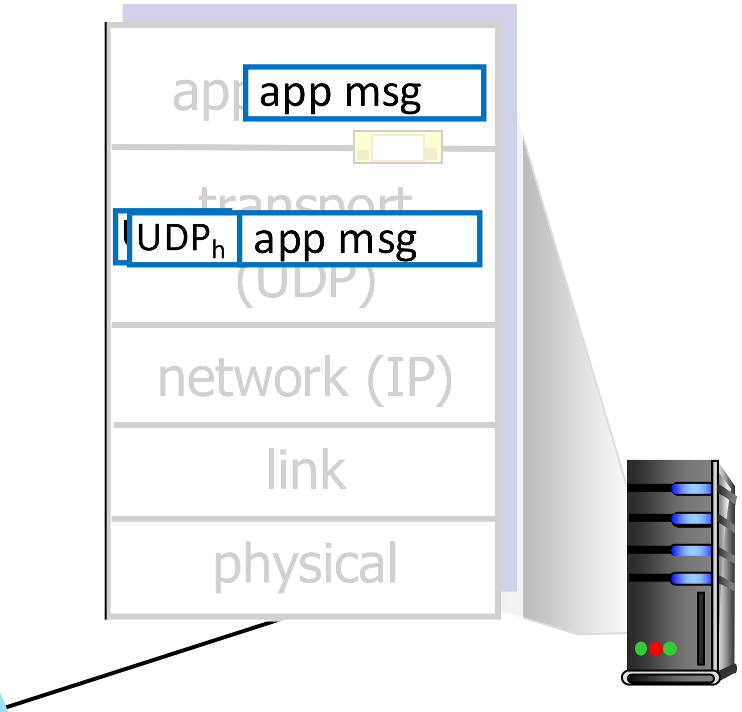


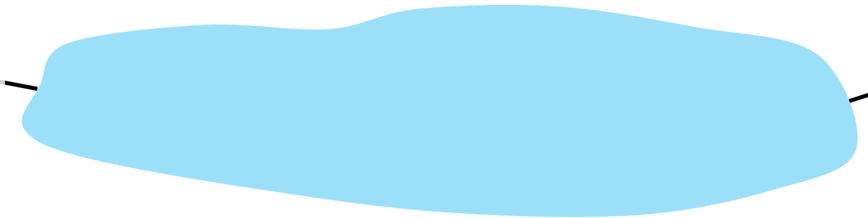
UDP: Transport Layer Actions



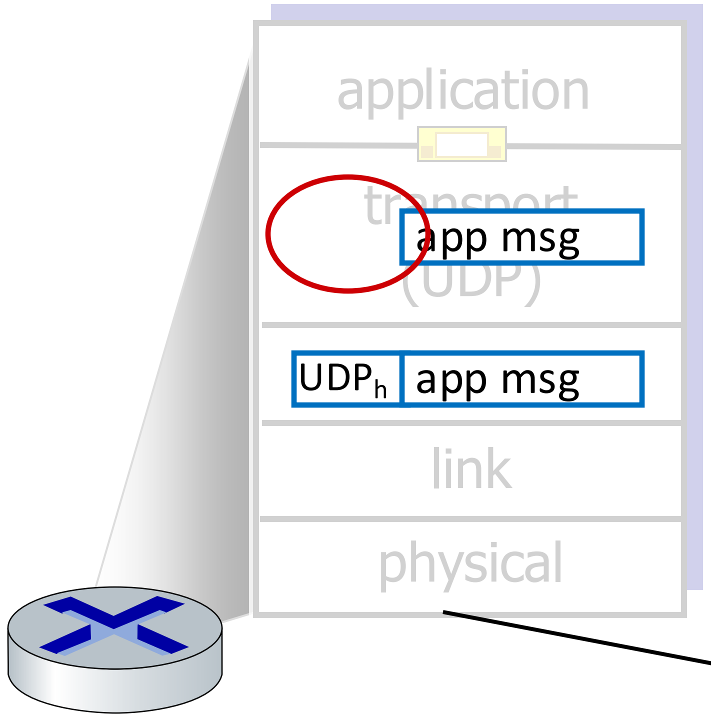
UDP sender actions:

- is passed an application-layer message
- determines UDP segment header fields values
- creates UDP segment
- passes segment to IP



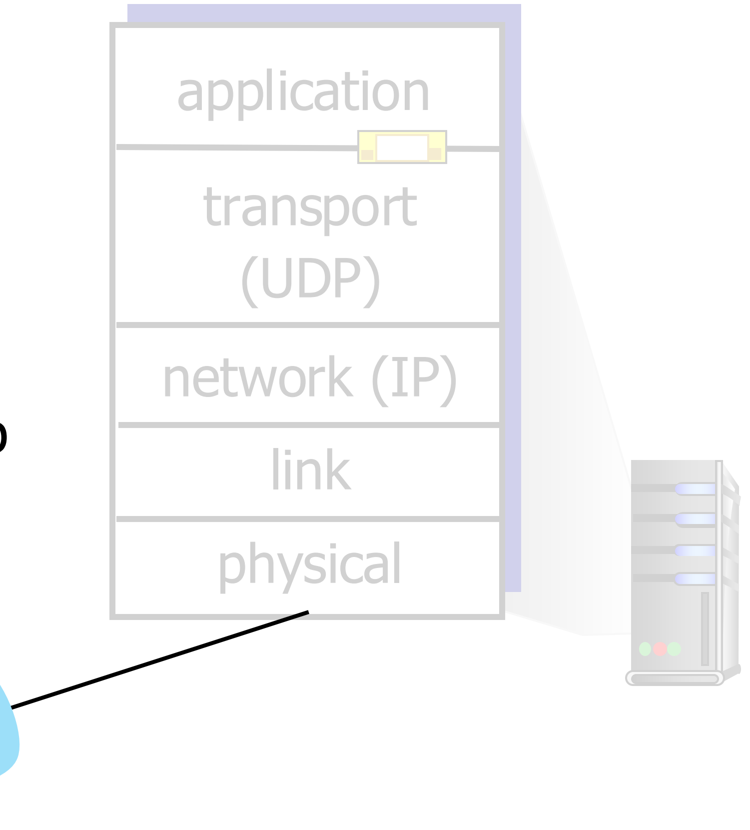


UDP: Transport Layer Actions

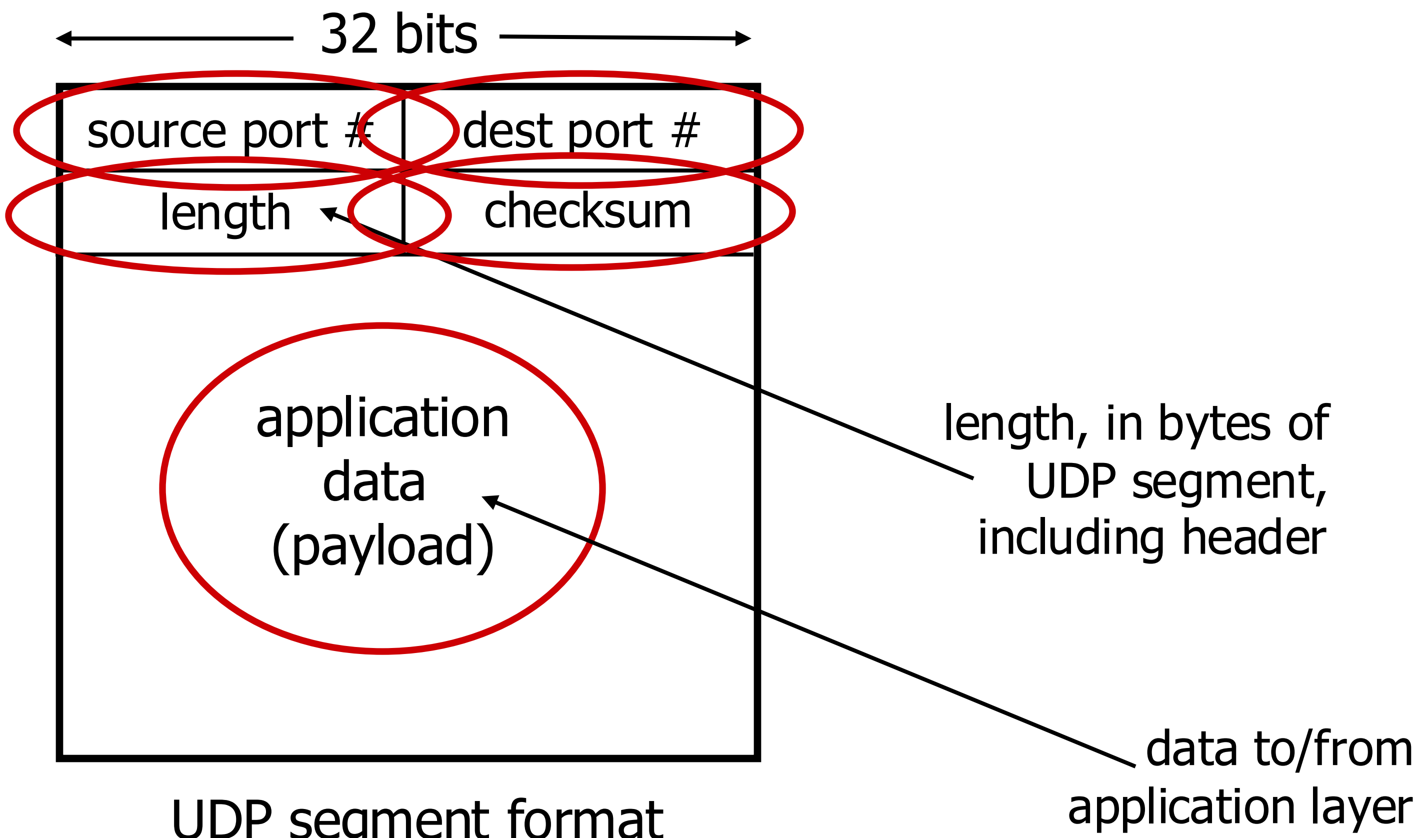


UDP receiver actions:

- receives segment from IP
- checks UDP checksum header value
- extracts application-layer message
- demultiplexes message up to application via socket



UDP segment header



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)

Internet checksum

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

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if 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 benefits:
  - no setup/handshaking needed (no RTT incurred)
  - helps with reliability (checksum)
  - …
- build additional functionality on top of UDP in application layer