Lesson 05-01: Transport Layer Intro

CS 326E Elements of Networking

Mikyung Han

mhan@cs.utexas.edu

Internet Reference Model

Example Protocols		Responsible for
FTP, HTTP, SMTP	Application	application specific needs
TCP, UDP	Transport	process to process data transfer
IP	Network	host to host data transfer across different network
Ethernet, WiFi	Link	data transfer between physically adjacent nodes
802.3 PHY	Physical	bit-by-bit or symbol-by-symbol delivery

2

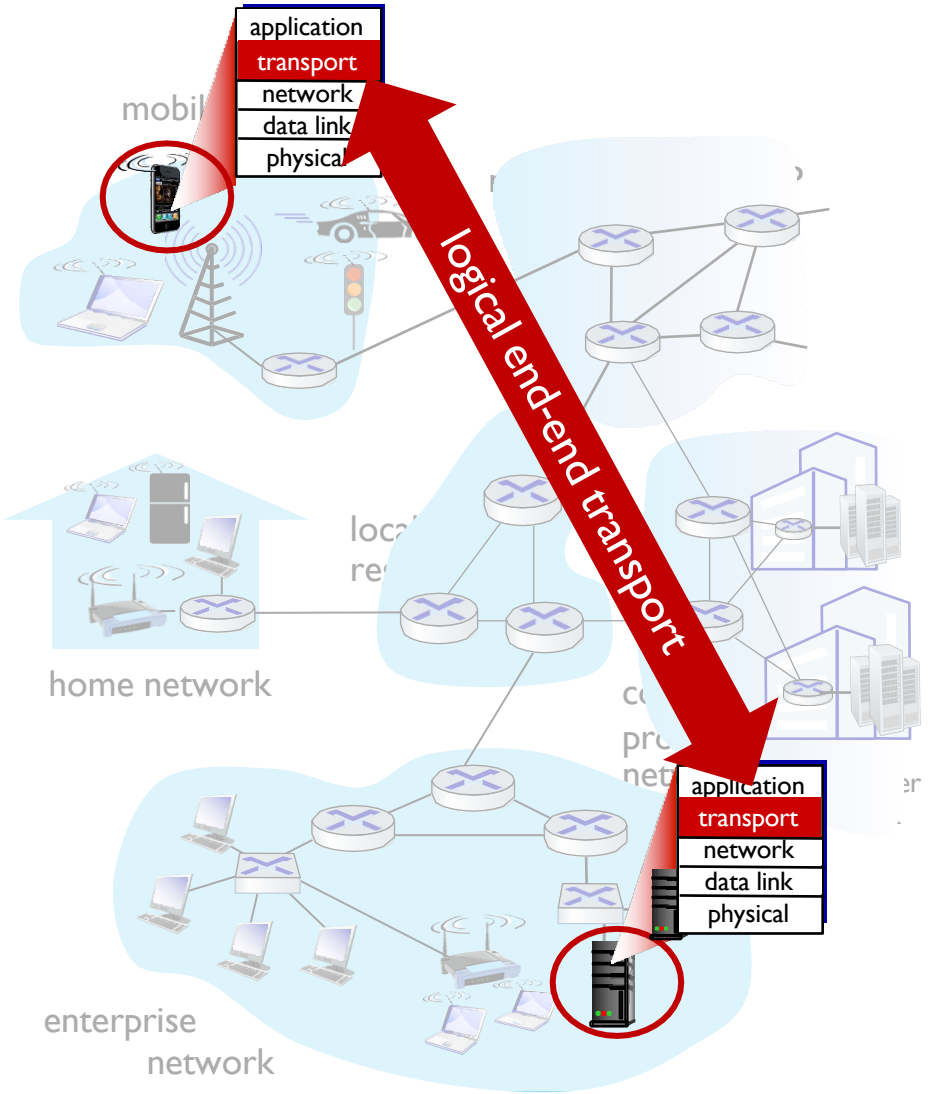
Outline

1. Why Transport Layer?

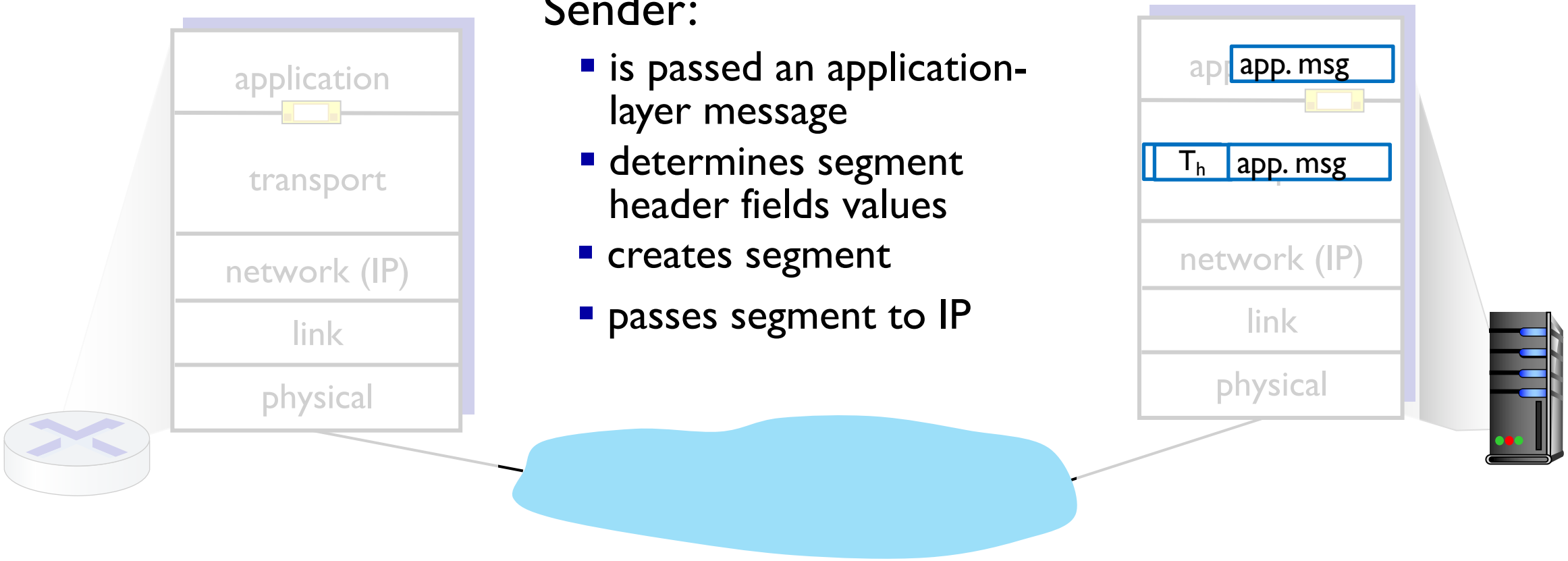
What is transport layer responsible for?

Transport layer's task is to deliver packets to the right application process

- Provides logical one hop between two application processes running on different hosts
  - What are these endpoints called? Sockets!
- Packets in transport layer are called Segments
- What are the two most commonly used protocols in transport layer?
  TCP or UDP



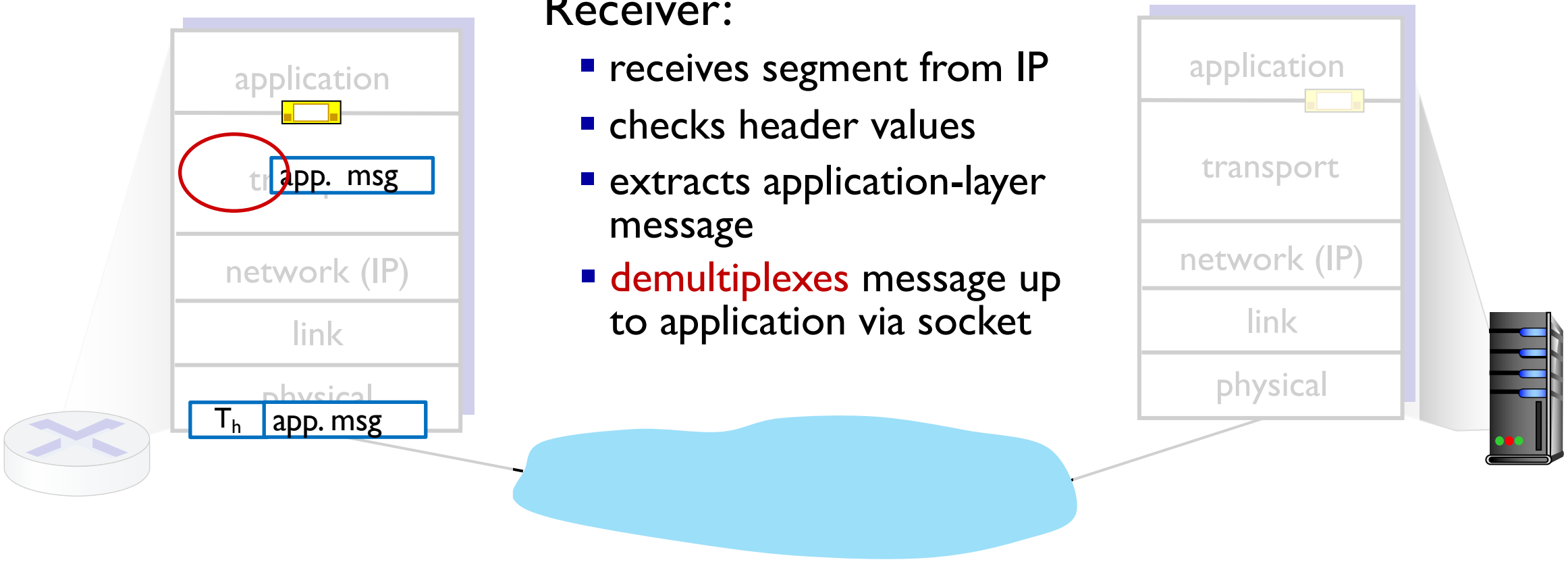
Transport Layer Actions



Sender:

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

Transport Layer Actions



Receiver:

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

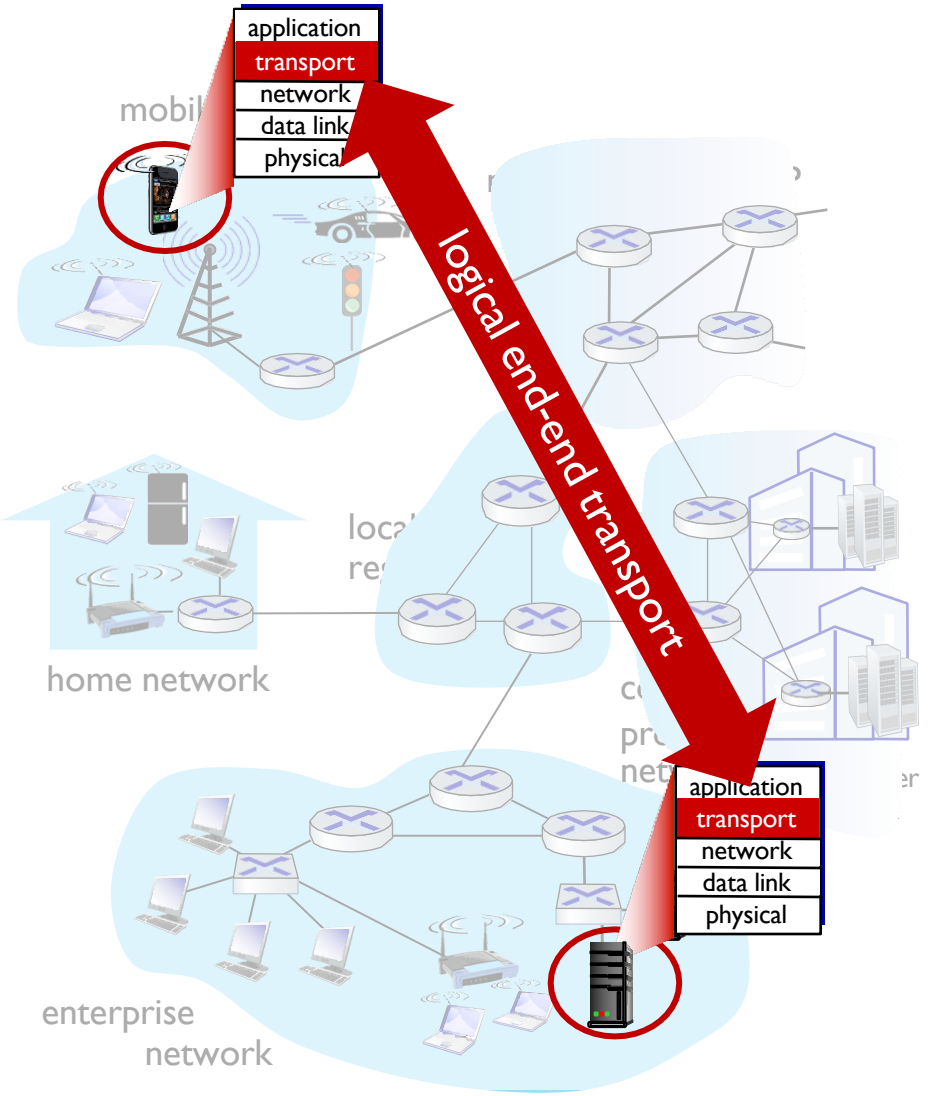
Outline

1. Why Transport Layer?
2. UDP vs TCP

UDP?

TCP vs UDP Features

- 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
- Both has NO guarantee on delay or bandwidth



When would you prefer UDP over TCP?

Another difference is
TCP is connection-oriented while UDP is connection less!

What do we mean by “connection”?

What is connection here?

- **BTW, this is different from Internet connectivity**
  - "Oh I don't have WiFi connection"
  - This is NOT what we are talking about

- **It is a short form of "connection-establishment"**

Analogy: Chocolate Handing Out Protocol (CHOP)

Purpose is to hand out chocolates to people

CHOP 1

- Whoever stops by to your station, you hand out the pre-packaged chocolate no matter what

CHOP 2

- Before you handout chocolate, you ask
  - How many they want
  - What kind they want
  - What time they want
  - Their names and contact
- After the agreement, you then hand out the chocolate accordingly

"Connection" in this context means
establishing agreement prior to actual data exchange

We say the protocol is connection-oriented

- There exists "establishment phase" prior to actual data exchange
  - Aka hand-shake
- Applies to all layers, not just transport layer
- Connection establishment and data exchange can happen over the same "channel"
  - Such as same TCP connection
- Sometime connection establishment can be done over a different mean
  - Connection establishment is done in "control channel"
  - Data exchange can be done separately in "data channel"

Recap: we say the protocol is stateful

- **The protocol saves any state regarding the other party: session state)**
  - At least one side (server side) saves state regarding the other (receiver) side
  - Or both side save info regarding the other side

UDP is stateless and TCP is stateful

There exists some correlation between being state-less/full and connection-less/-oriented

- Bottom line is "how much do I care about the other party"?
  - Anything beyond the src address/port that I need to ask and save?

- Do I need to do something differently based on whom I am talking with?
  - Send more/less traffic
  - Send specific packets
- Establishing how to differ would be connection-oriented part
- Saving that info would be the stateful part

Can connection-oriented protocol be stateless?

In order to be stateless but still connection-oriented… where to save the ”states” related to connection?

- Where do we normally save these states at?
  - Inside server machine (or client machine as well)
- Where else besides server or client to “record” states?

Without anyone saving the states at host,
each party can specify the agreement in the packet header!

Connection-oriented stateless protocol example

- Both server and client agrees to use WiFi channel # 1 to communicate
  - WiFi Channels
- Server and client exchanges data without saving any other info but just specify Ch 1 info in the packet header

UDP is connection-less stateless protocol

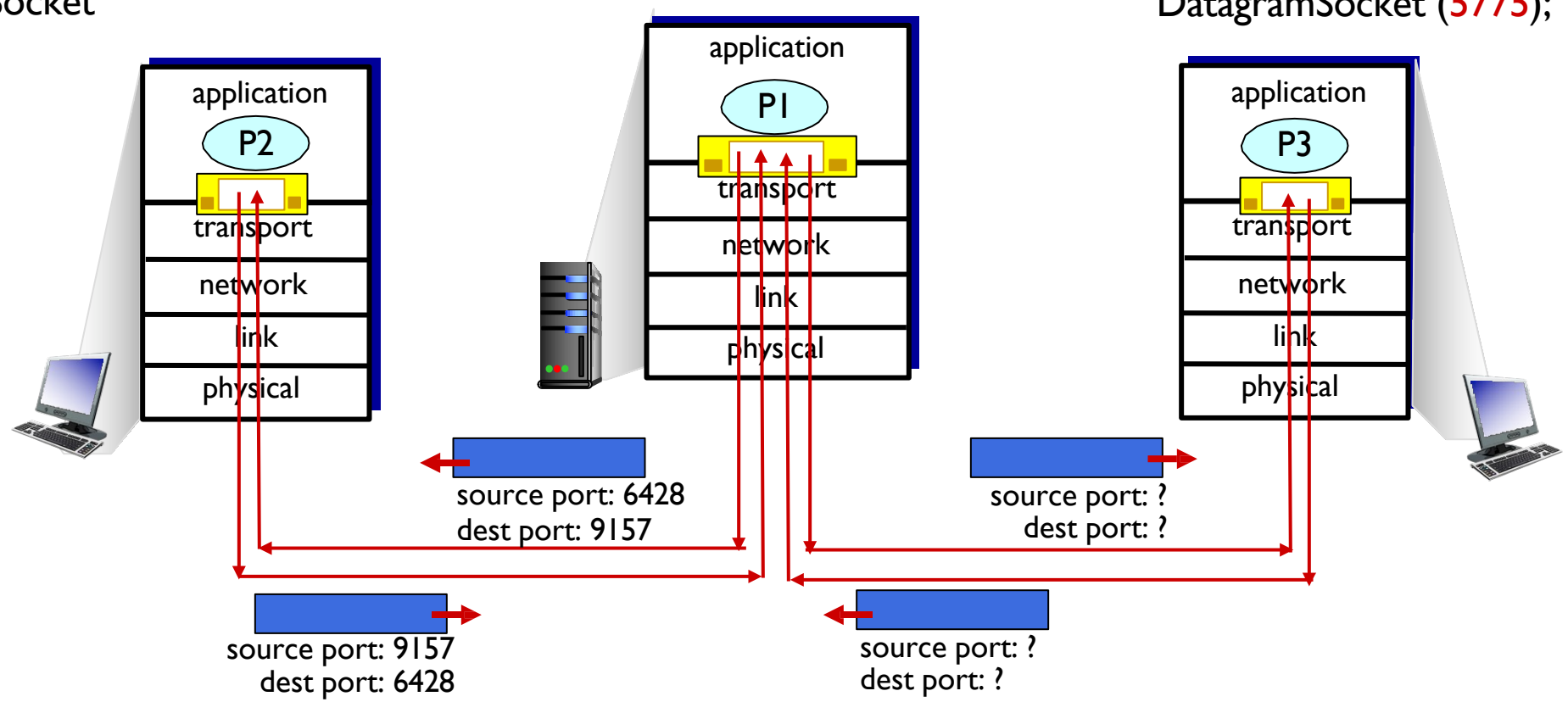
- **UDP doesn't really care who it is talking with**
  - Yes, the receiving end does take a look at IP:port of the source and replies back to that IP:port but that's about it
- **No need to establish custom "channel" for the communication**
- **UDP does not maintain any states of who they are**
  - The upper application layer may care and maintains states but not in transport layer
- **Same socket are shared to receive messages from multiple clients**

UDP demultiplexing

DatagramSocket serverSocket = new DatagramSocket (6428);

DatagramSocket mySocket2 = new DatagramSocket (9157);

DatagramSocket mySocket1 = new DatagramSocket (5775);



TCP is connection-oriented stateful protocol

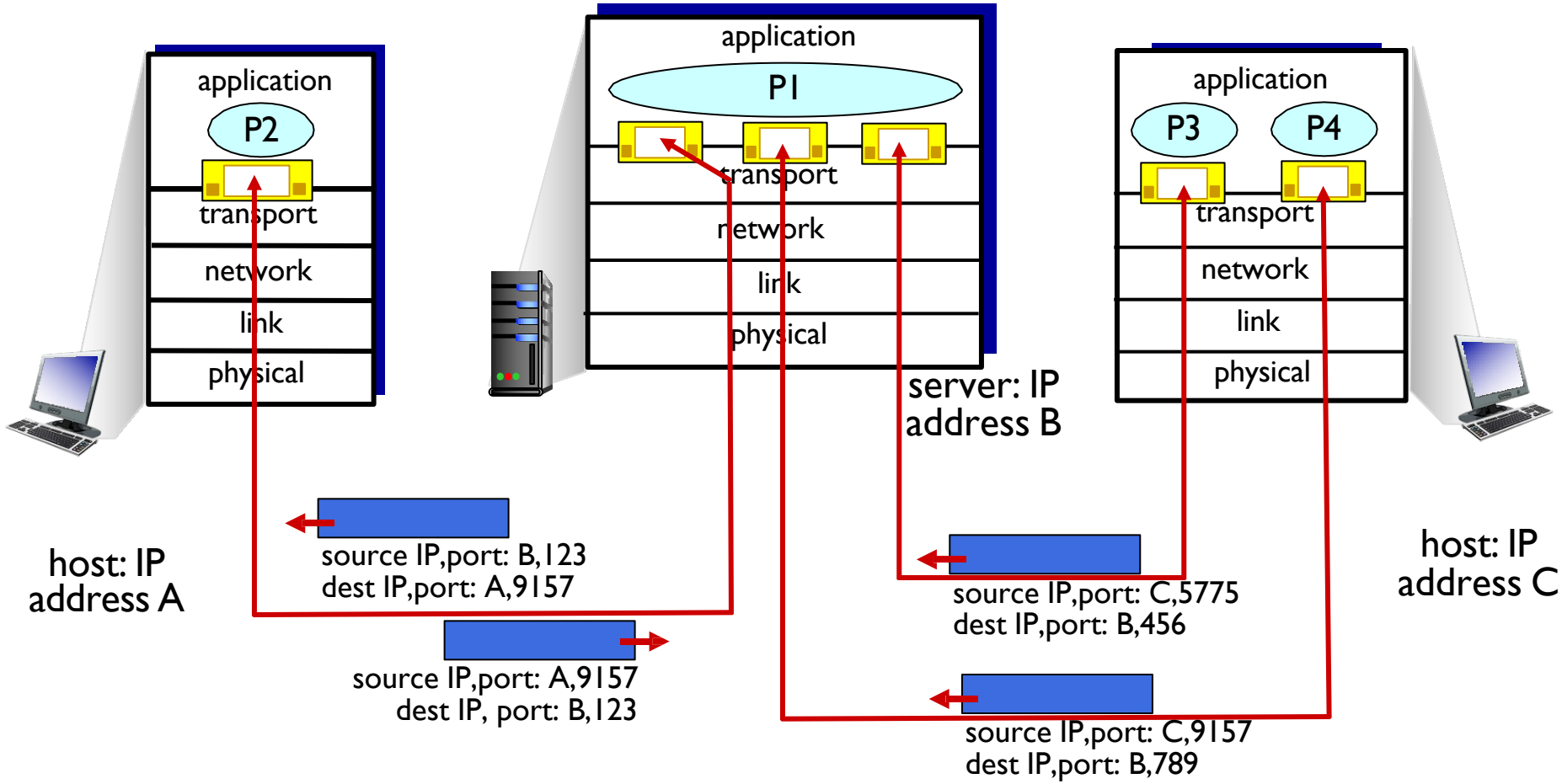
- TCP cares about with whom it's talking
- Pre-establishes agreement for data exchange: TCP hand-shake
- States are maintained per connection
  - ACK, sequence number
- The connection is identified by 4 tuple
  - Each src(IP:port) – dst(IP:port) pair is a "connection"
- Typically, a separate socket is used for each client unlike UDP

TCP uses a separate socket for each client

- Each client has different IP:port
- Server has one listening socket that all new client requests comes in
  - [listening socket] Well-known IP_s:$port_s$
- Server communicates with each client with a separate socket
  - [servicing socket 1] IP_s:$port_s$ – IP_{c1}:$port_{c1}$
  - [servicing socket 2] IP_s :$port_s$ – IP_{c2}:$port_{c2}$
  - [servicing socket 3] IP_s :$port_s$ – IP_{c3}:$port_{c3}$

TCP socket is identified by 4 tuples!

TCP demultiplexing



Three segments, all destined to IP address: B are demultiplexed to different sockets

TCP header vs UDP header

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

UDP socket programming example

```python
# Server configuration
HOST = '0.0.0.0'  # Listen on all available network interfaces
PORT = 12345      # Port to listen on

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address and port
server_socket.bind((HOST, PORT))

print(f"UDP server listening on {HOST}:{PORT}")

while True:
    # Receive data from client (max buffer size 1024 bytes)
    data, client_address = server_socket.recvfrom(1024)
    print(f"Received from {client_address}: {data.decode()}")

    # Send a response back to the client
    response = f"Server received: {data.decode()}"
    server_socket.sendto(response.encode(), client_address)
```

```python
# Server configuration
SERVER_IP = '127.0.0.1'  # Change this to match your server's IP
PORT = 12345             # Same port as the server

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Message to send
message = "Hello, UDP Server!"

# Send the message to the server
client_socket.sendto(message.encode(), (SERVER_IP, PORT))

# Receive response from the server
response, server_address = client_socket.recvfrom(1024)
print(f"Server responded: {response.decode()}")

# Close the socket
client_socket.close()
```

How about DNS?

- Is it stateful or stateless?
- Is it connection-less or connection-oriented?

How about SSH?

- Is it stateful or stateless?
- Is it connection-less or connection-oriented?

Outline

1. Why Transport Layer?
2. TCP vs UDP
3. Project 1

Acknowledgements

Slides are adopted from Kurose' Computer Networking Slides

DNS is is connection-less stateless protocol

- **DNS doesn't really care who is asking the question**
  - The server just take notes on src IP:port and simply replies back to that IP:port. That's about it. No notion of "session"
- **DNS does not maintain any states of who the clients are**
- **No need to establish custom "channel" for the communication in the application layer**
  - Also, no connection establishment in transport layer as well (UDP)
- **Each DNS query is completely independent**