CS 331: Computer Networks

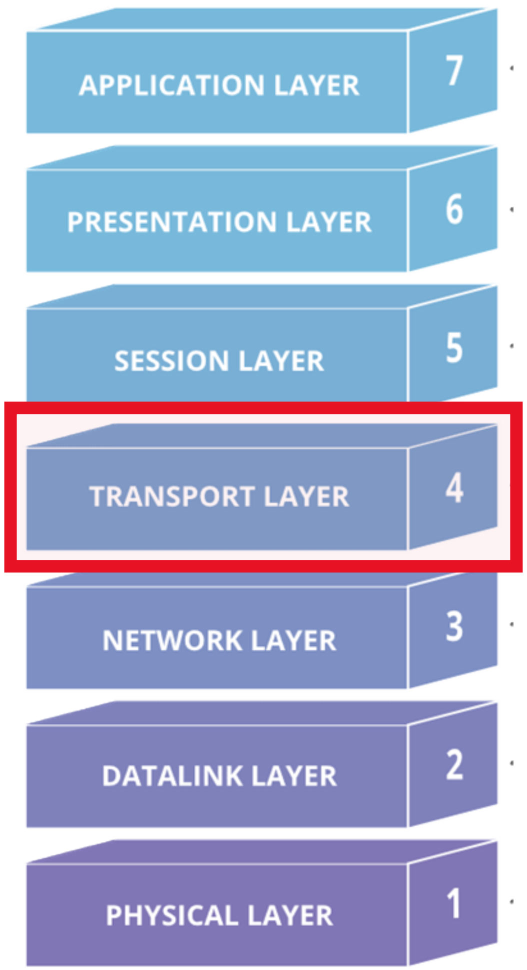
Sameer G Kulkarni

Assistant Professor,

Department of Computer Science and Engineering,

Indian Institute of Technology, Gandhinagar

TRANSPORT LAYER



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

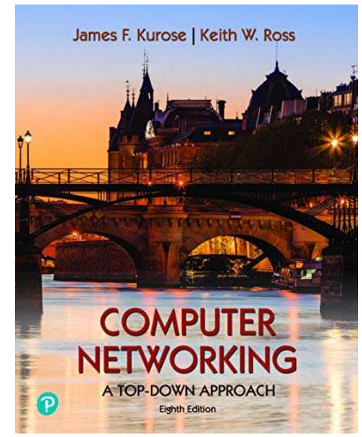
- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR ©

All material copyright 1996-2019
J.F Kurose and K.W. Ross, All Rights Reserved

Few Slides are borrowed from Prof. Rajat Chakraborty and Prof. Sandip Chakraborty, Computer Science Department, IIT Kharagpur

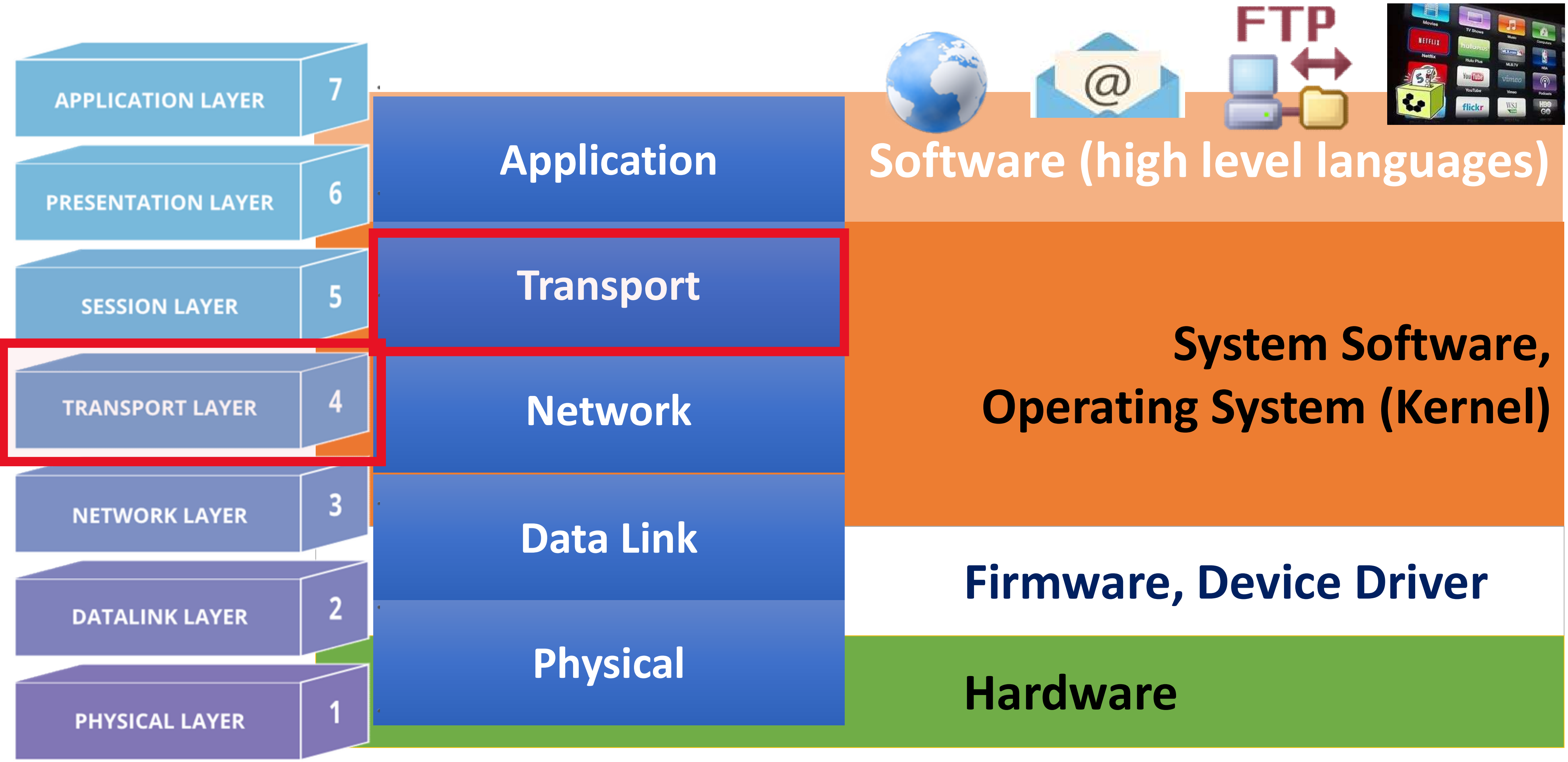
Transport Layer: Study of Protocols: TCP, UDP. Reliable Data transfer, Flow control, congestion control. TCP variants.



Chapter 3

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

Protocol Stack Implementation in a Host



Transport Layer Services

End to end packet delivery

UDP

Reliable and Ordered Data Delivery

Connection Establishment

Flow Control

Congestion Control

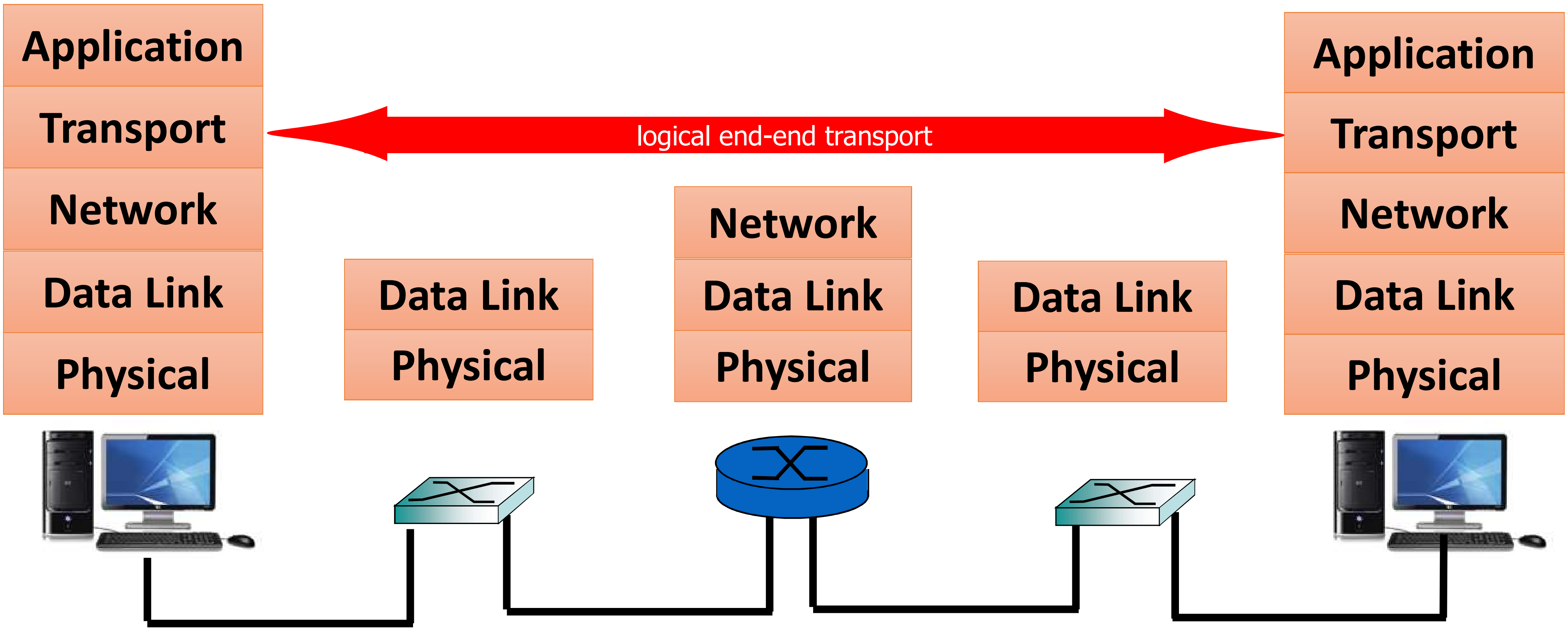
TCP

Transport Layer

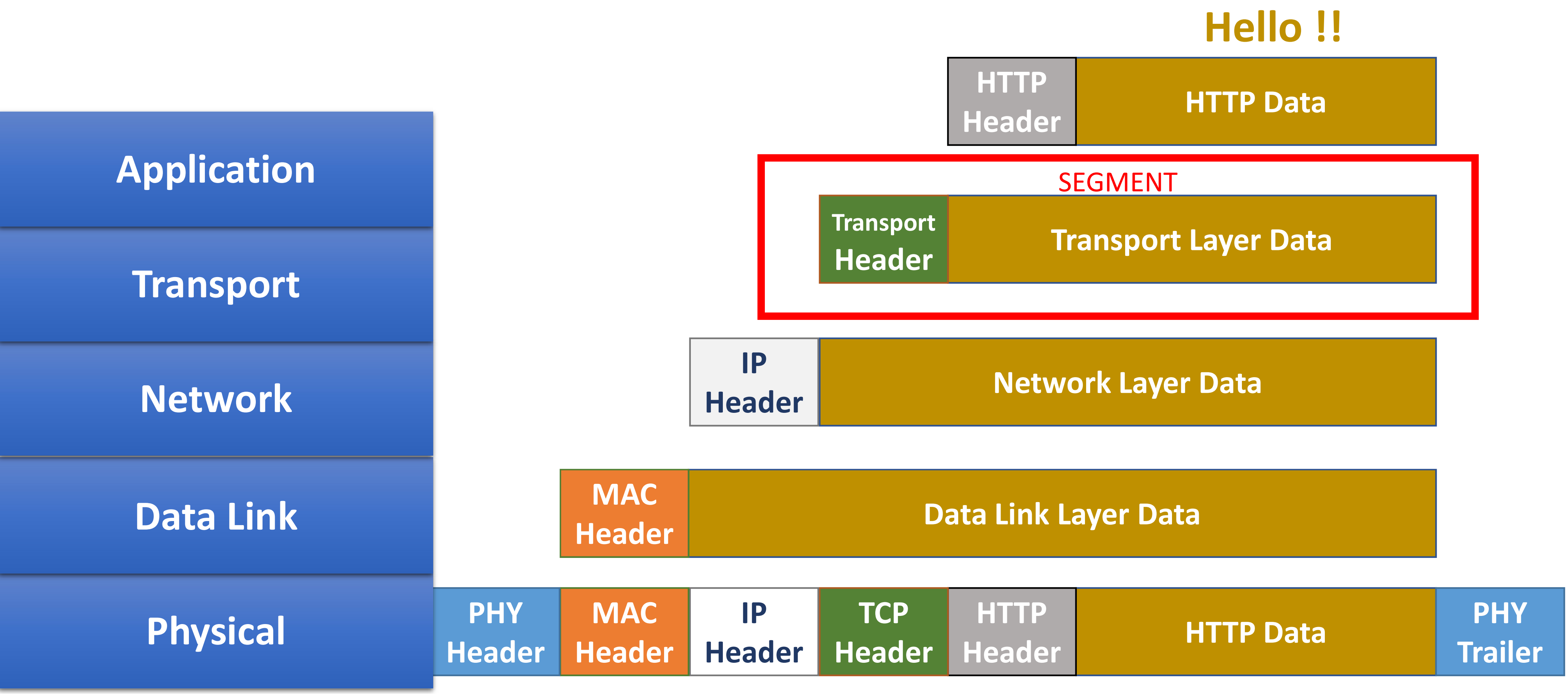
IP Layer

Packet delivery (unreliable)

Data Transfer between Two Remote Machines

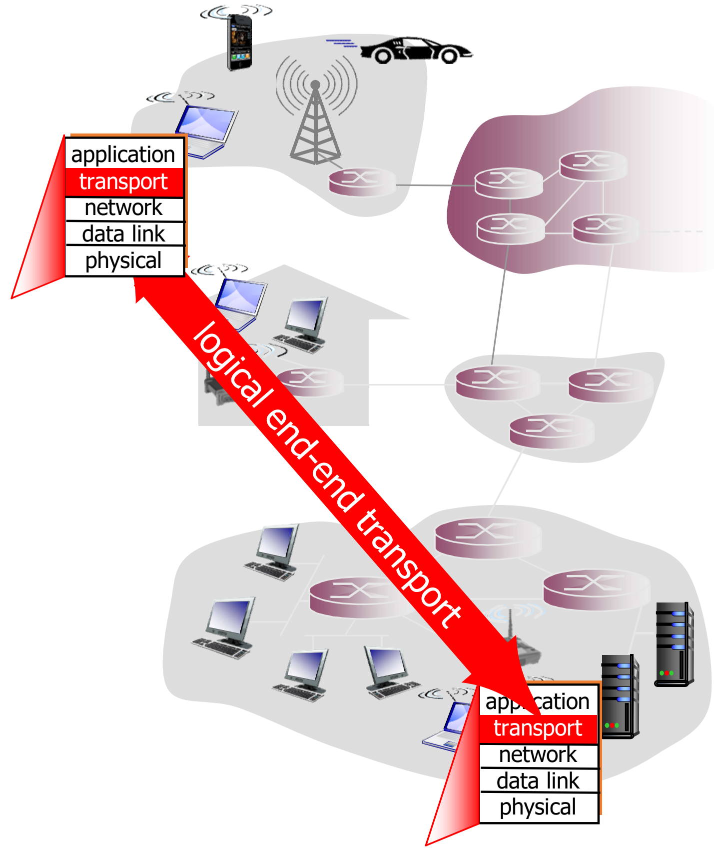


How Application Data Passes Through Different Layers



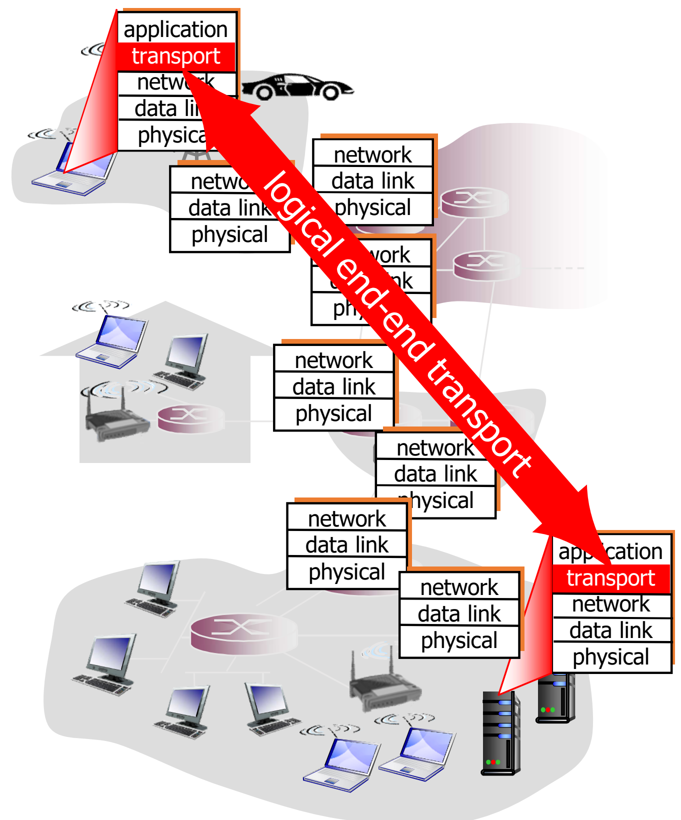
TRANSPORT SERVICES AND PROTOCOLS

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



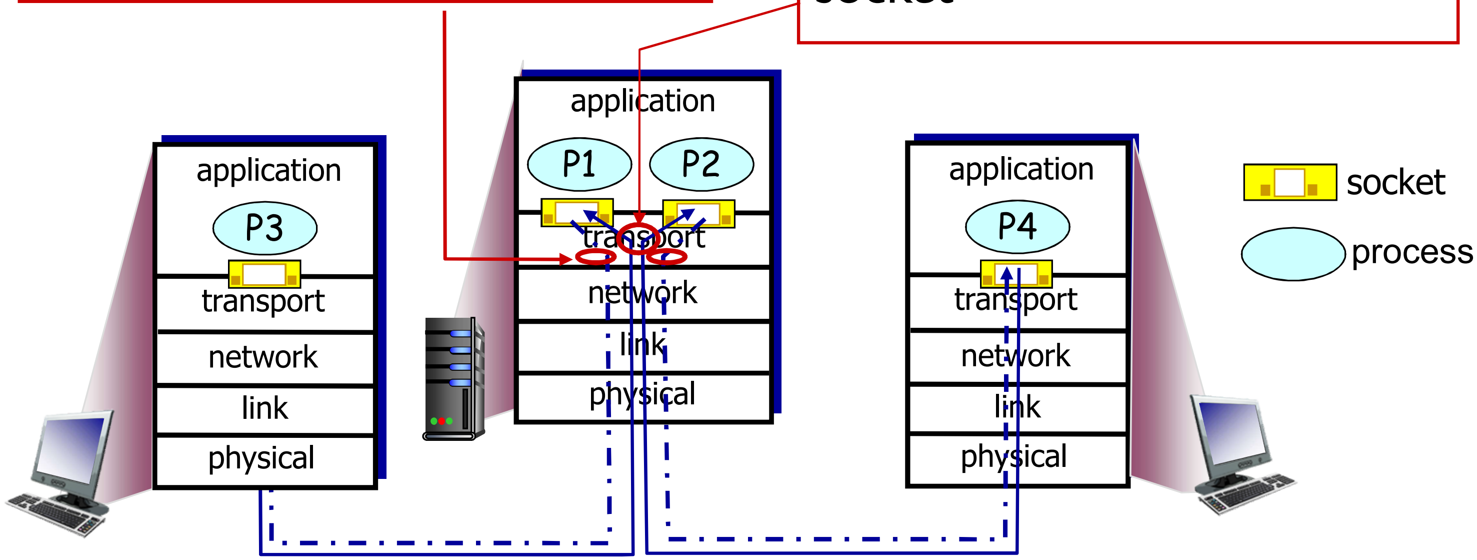
TRANSPORT-LAYER PROTOCOLS

- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- reliable, in-order delivery (TCP)
  - connection setup
  - congestion control
  - flow control
- services not available:
  - delay guarantees
  - bandwidth guarantees

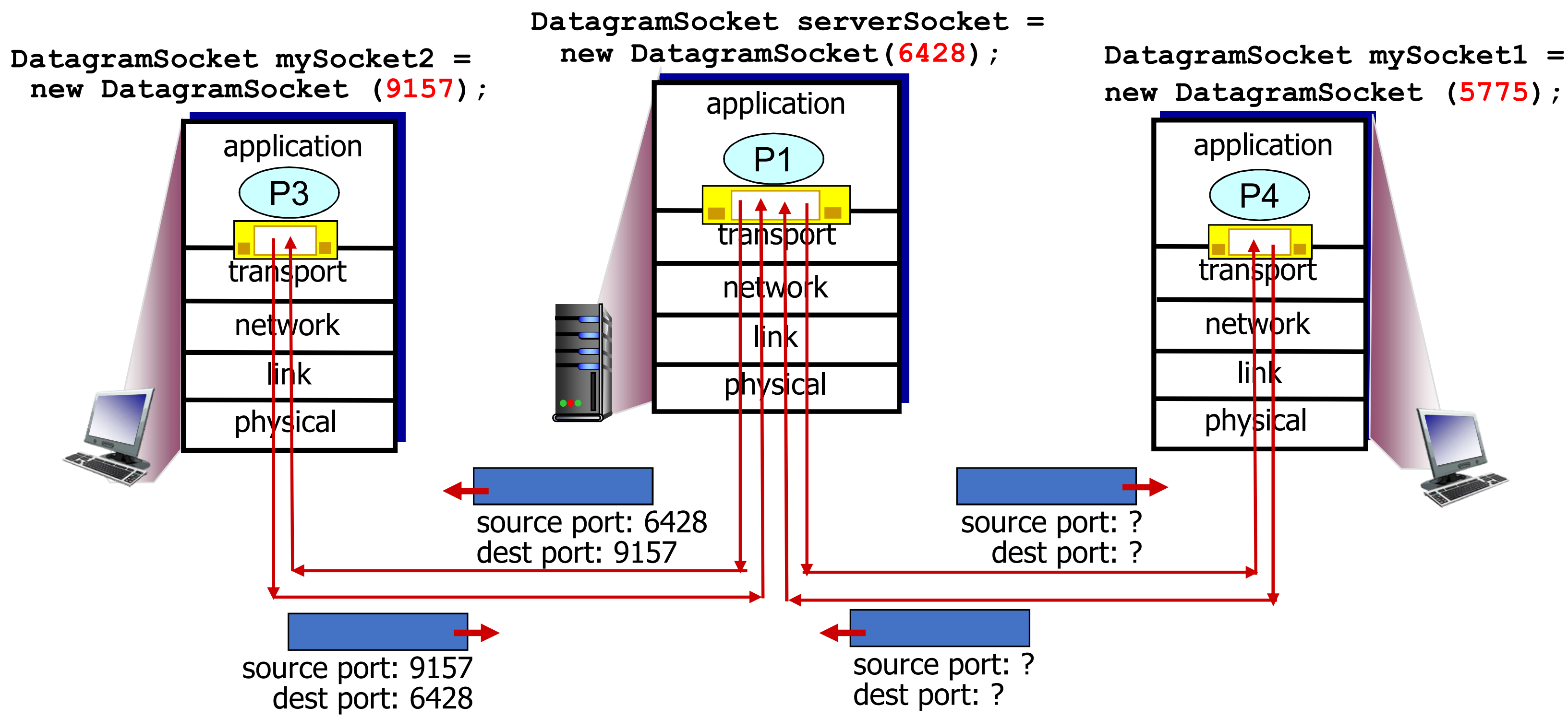


multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket



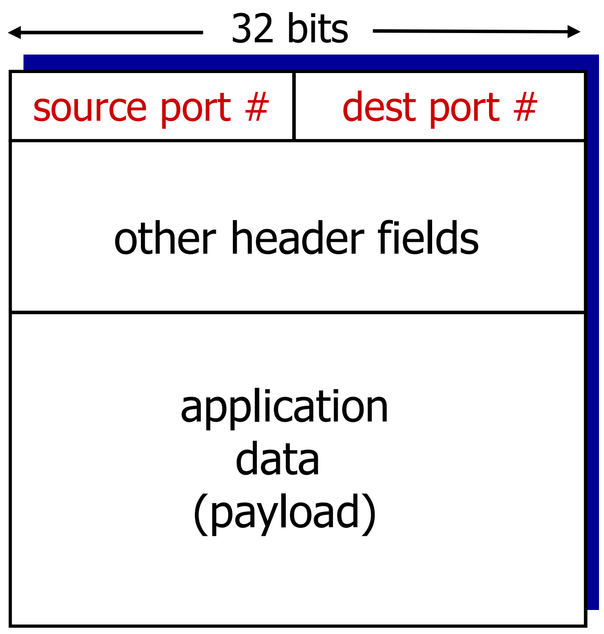
CONNECTIONLESS DEMUX: EXAMPLE



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



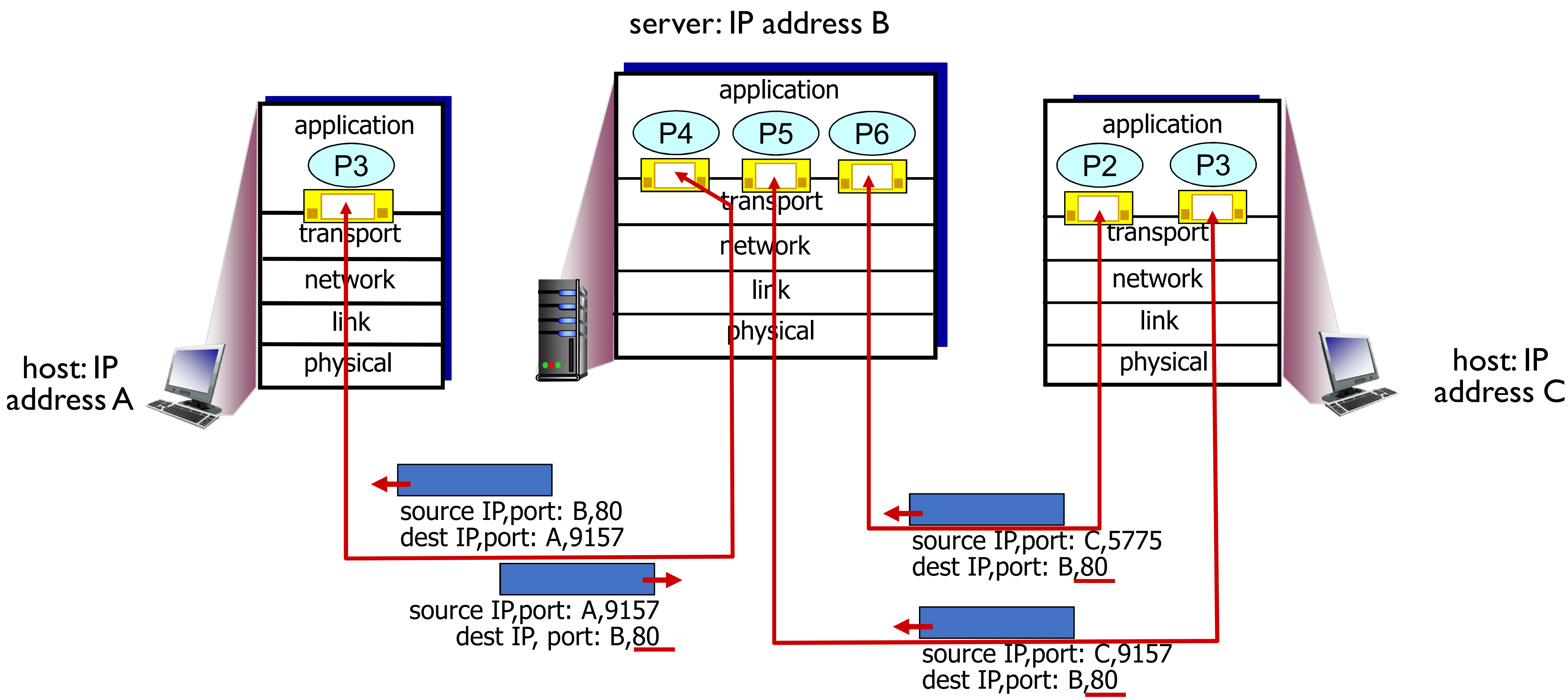
TCP/UDP segment format

CONNECTION-ORIENTED DEMUX

- Transport layer connection is identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket

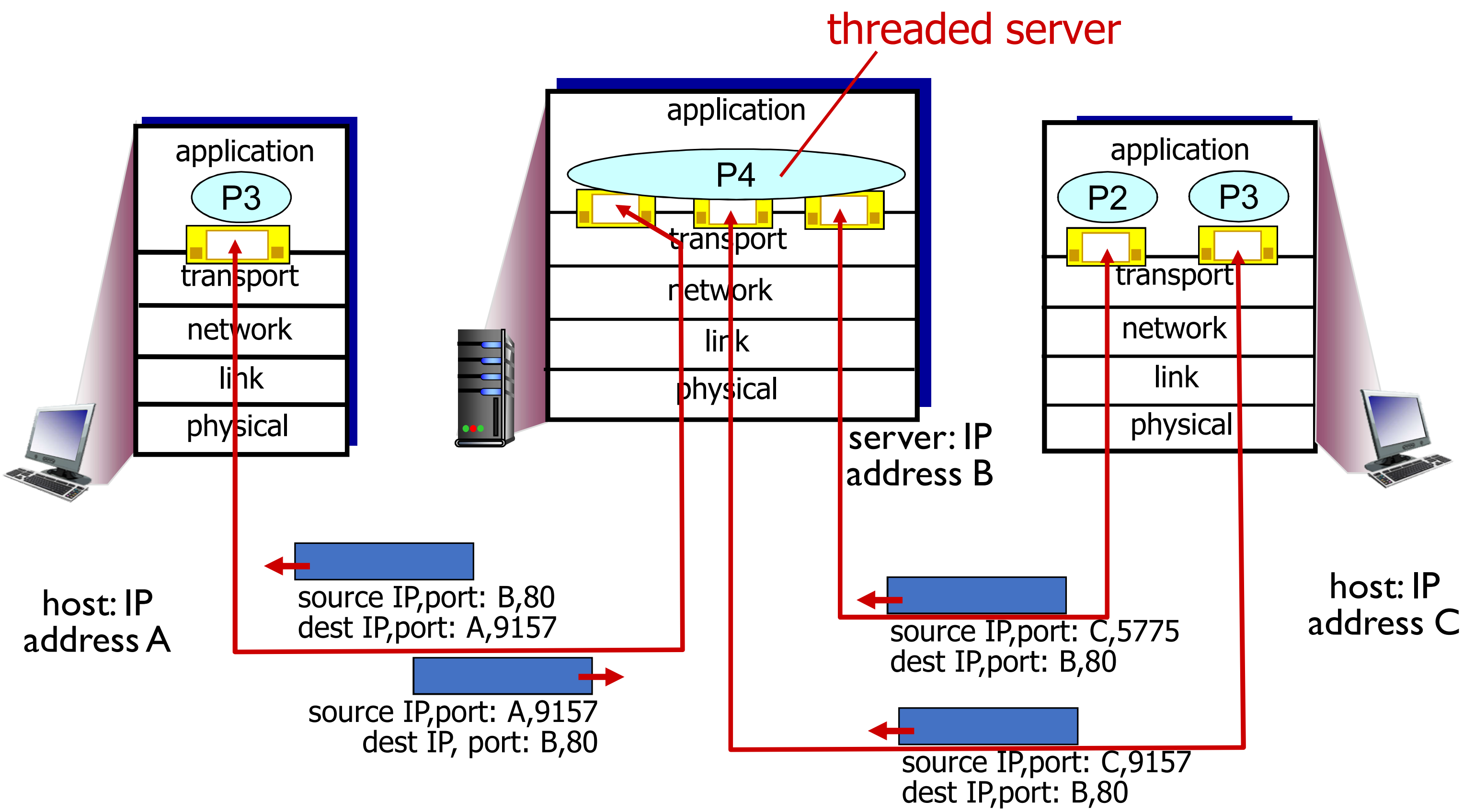
- server host may support many simultaneous sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client.
  - non-persistent HTTP will have different socket for each request

CONNECTION-ORIENTED DEMUX: EXAMPLE

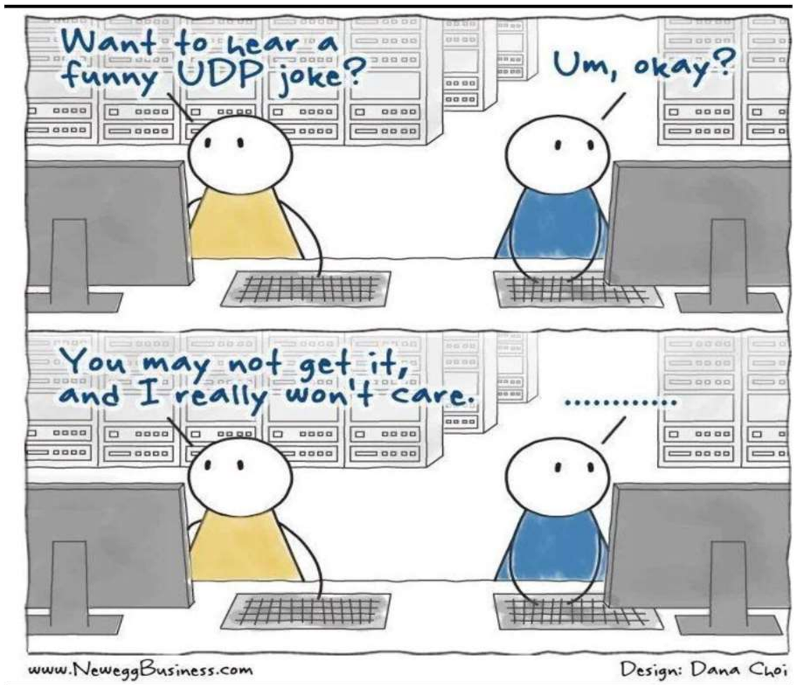


three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

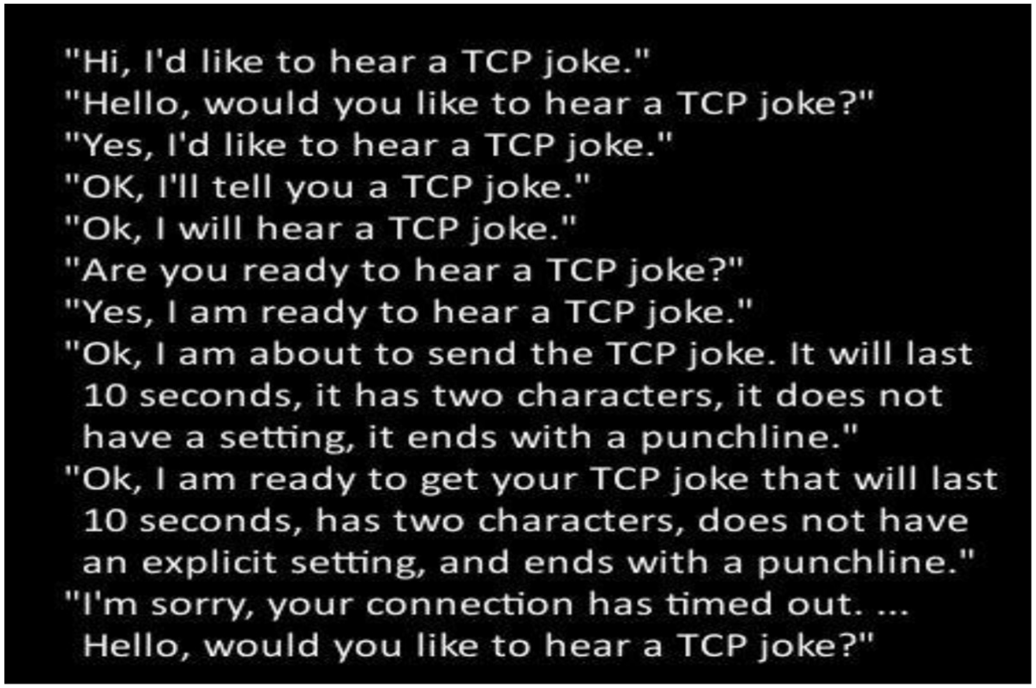
CONNECTION-ORIENTED DEMUX: EXAMPLE



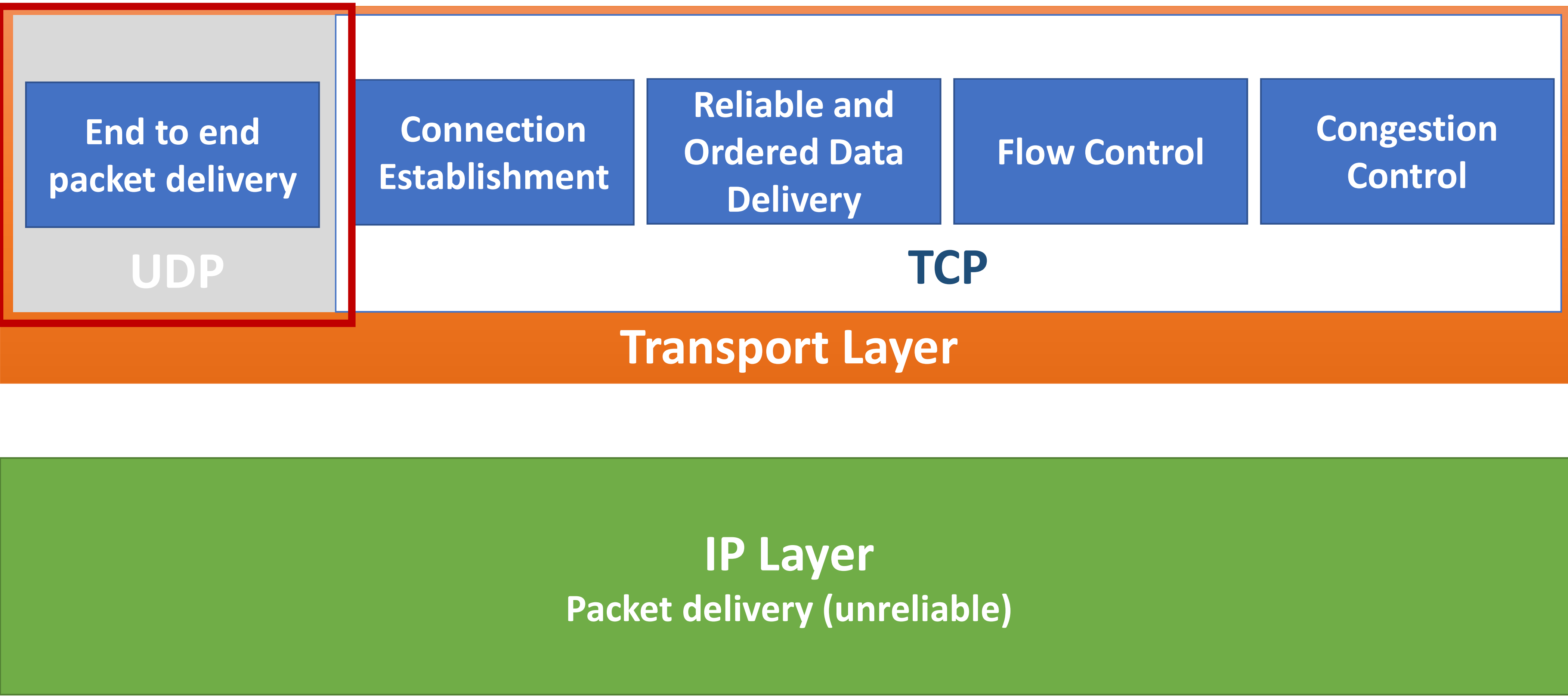
UDP



TCP


"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. ... Hello, would you like to hear a TCP joke?"

Transport Layer Services



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: User Datagram Protocol [RFC 768]

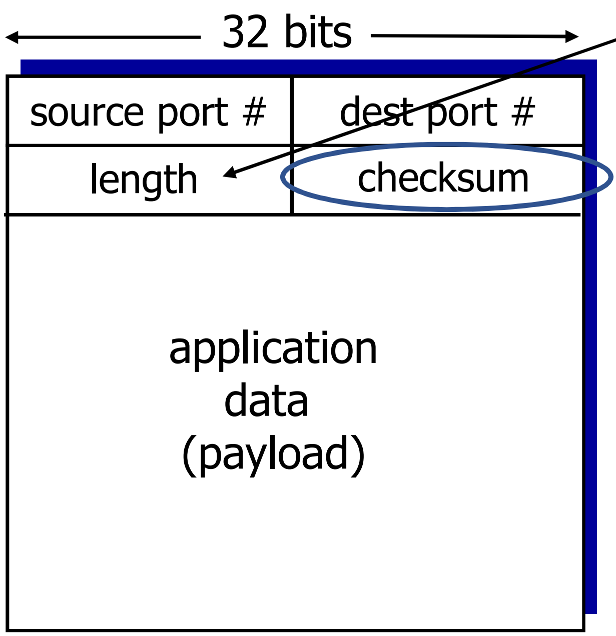
- *"best effort" service*
  - No guarantees on delivery, ….
  - delivered out-of-order
- *connectionless:*
  - *no handshaking* between UDP sender, receiver
- *Message/Datagram:*
  - each UDP segment handled independently of others

- UDP usage:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - DHCP
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

Explore more on UDP-Lite, RUDP, DCCP and SCTP

UDP: SEGMENT HEADER

length, in bytes of UDP segment, including the header



UDP segment format

why is there a UDP?

- simple: no connection state at sender, receiver
- small header size
- no connection establishment (which can add delay)
- no congestion control: UDP can blast away as fast as desired
- No retransmissions or in-order delivery: perceived - better control on latency
- ***Multicast***

UDP Error Detection

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

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

$\underbrace{4 \quad 6}_{\text{receiver-computed checksum}} \neq \underbrace{11}_{\text{sender-computed checksum (as received)}}$

!

```
int16_t checksum = 0;
int16_t comp_checksum (uint32_t count, int8_t* addr)
{
    while(1 < count) {
        sum += *(unt16_t)addr++;
        count -=2;
    }
    if (count)
        sum += *(unt16_t)addr++;

    while(sum >> 16)
        sum = (sum & 0xffff) + (sum >>16);

    return (checksum = ~sum);
}
```

UDP/Internet checksum

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

sender:

- treat contents of UDP segment (including UDP header and *IP header fields*) as sequence of 16-bit integers.
- checksum: addition (one's complement sum) of segment content.
- Update UDP checksum field with the computed checksum value.

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 ….

https://datatracker.ietf.org/doc/html/rfc1071

INTERNET CHECKSUM: AN EXAMPLE

example: add two 16-bit integers

Note: When adding numbers, a carry from the msb needs to be added back to the result.

```
                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   <- Binary sum

sum             1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0   <- Add carry
checksum        0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1   <- 1s complement
```

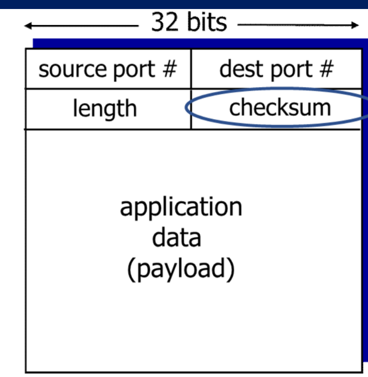
```
                1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0   -> 0 1
                1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1   -> 1 0
             -----------------------------------
wraparound    1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

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
```

What about two or multi-bit errors?

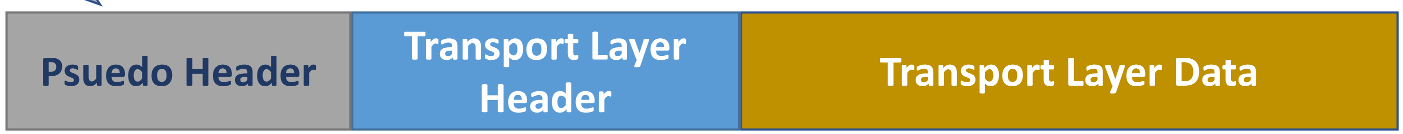
Even though numbers have changed (bit flips), *no* change in checksum!

How is UDP Checksum Computed?



Source IP Address (4B)
Destination IP Address (4B)
Packet length (2B)
Protocol Type (1B)
Reserved Field (1B)

Divide and slice the content into 16 bit pieces (2 Bytes).



At the Sender:

- Transport layer computes the Checksum and adds it to the Checksum field.
- Note: During the calculation of checksum, the checksum field is set with 0's. After the calculation, the computed checksum value will be updated.

At the Receiver:

- Transport layer computes the Checksum and *compares with sent value*.

TCP: Overview RFCs: 793, 1011, 2873, 5961, 6093, 6429, 6528, 6691, 9293

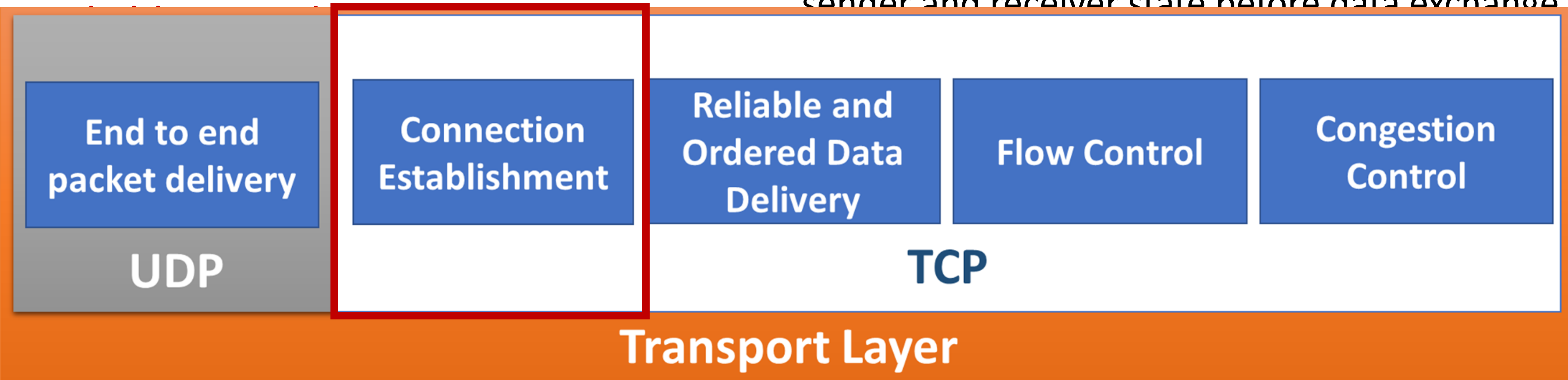
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- pipelined:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender and receiver state before data exchange
- Flow controlled:
  - sender will not overwhelm receiver
- Congestion controlled:
  - sender will not overwhelm the network/ adapt to the network capacity.

Today's Focus:
Understanding TCP Protocol
Connection Setup and Teardown
Sequence and Acknowledgment Numbers
Round Trip time and RTT Estimation

TCP: Overview RFCs: 793, 1011, 2873, 5961, 6093, 6429, 6528, 6691, 9293

- point-to-point:
  - one sender, one receiver

- connection-oriented:
  - handshaking (exchange of control msgs) inits sender and receiver state before data exchange



to the network capacity.

- TCP congestion and flow control set window size

Today's Focus:
Understanding TCP Protocol
Connection Setup and Teardown
Sequence and Acknowledgment Numbers
Round Trip time and RTT Estimation

TCP SEGMENT STRUCTURE

ACK: seq # of next expected byte; A bit: this is an ACK
connection setup and Ordering

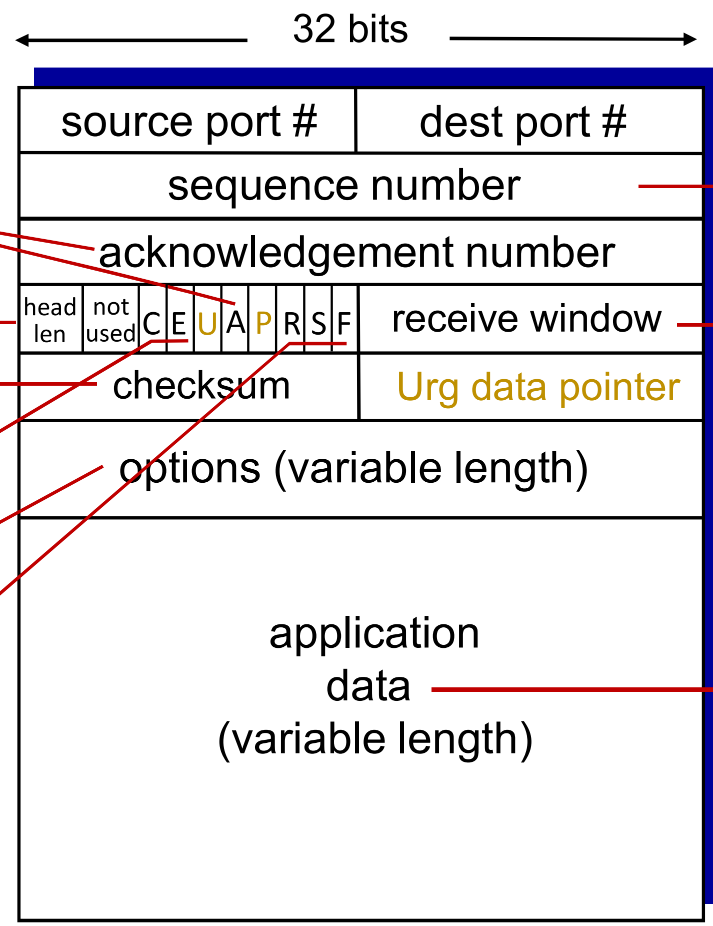
length (of TCP header)

Internet checksum

C, E: congestion notification

TCP options

RST, SYN, FIN: connection management



segment seq #: counting bytes of data into bytestream (not segments!)
Connection setup and ordering

flow control: # bytes receiver willing to accept

data sent by application into TCP socket