

Chapter 3: Transport Layer

Our goals:

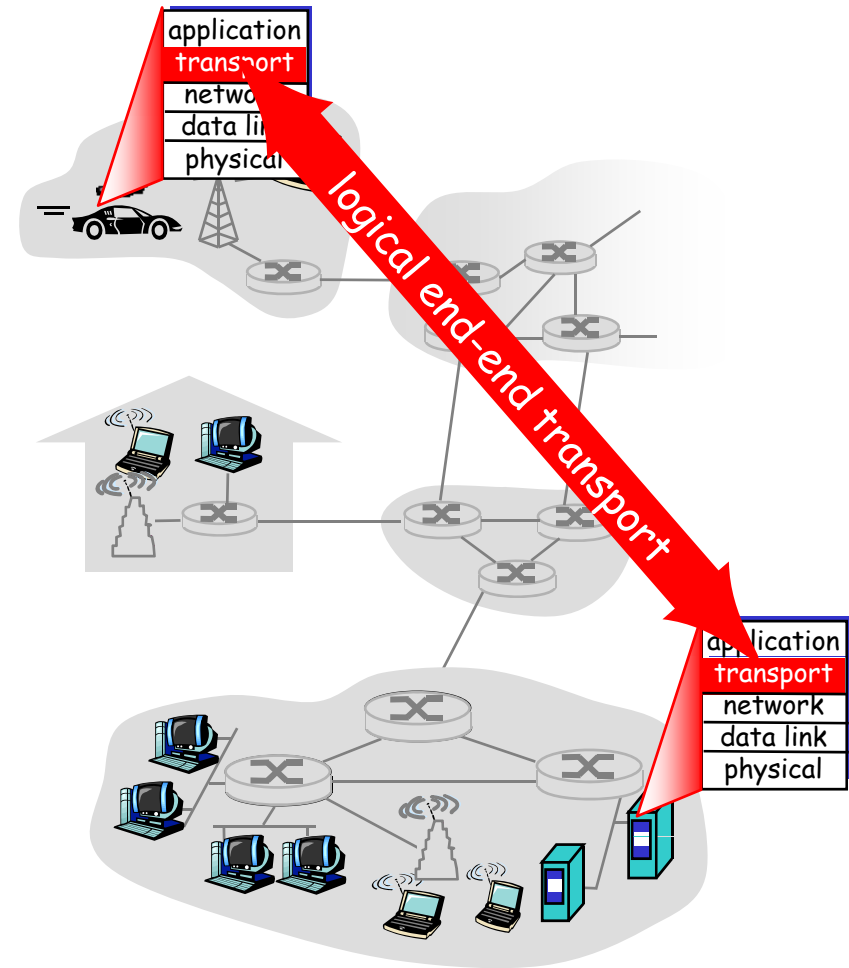
- understand principles behind transport layer services:
 - multiplexing/demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- learn about transport layer protocols in the Internet:
 - UDP: connectionless transport
 - TCP: connection-oriented transport
 - TCP congestion control

Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

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 vs. network layer

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

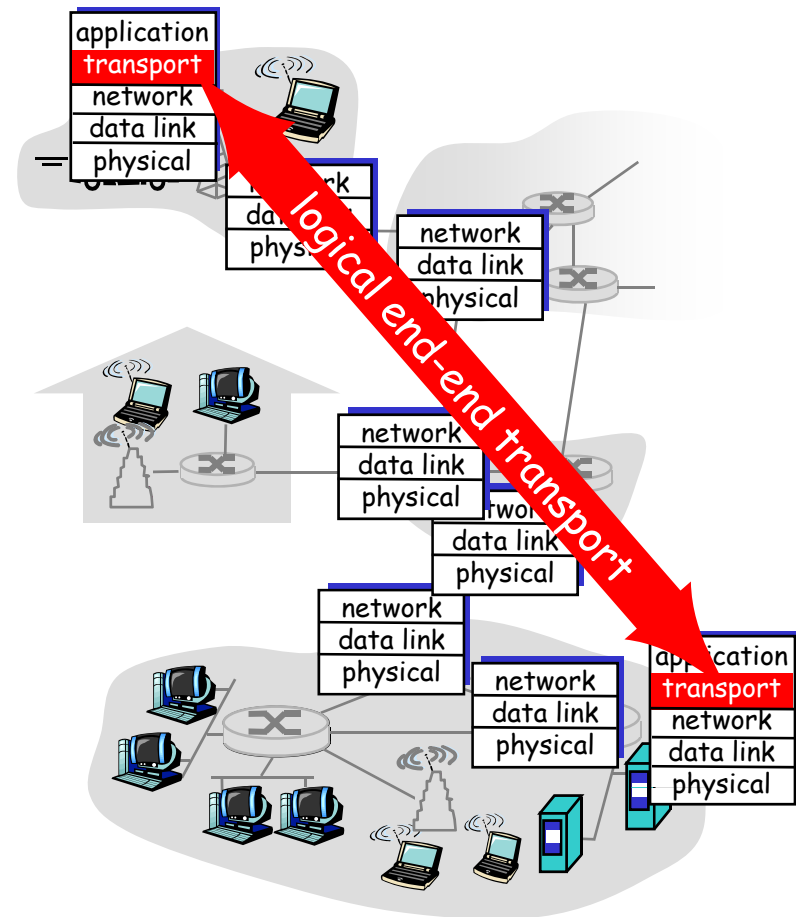
Household analogy:

12 kids sending letters to 12 kids

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

Internet transport-layer protocols

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



Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

Multiplexing/demultiplexing

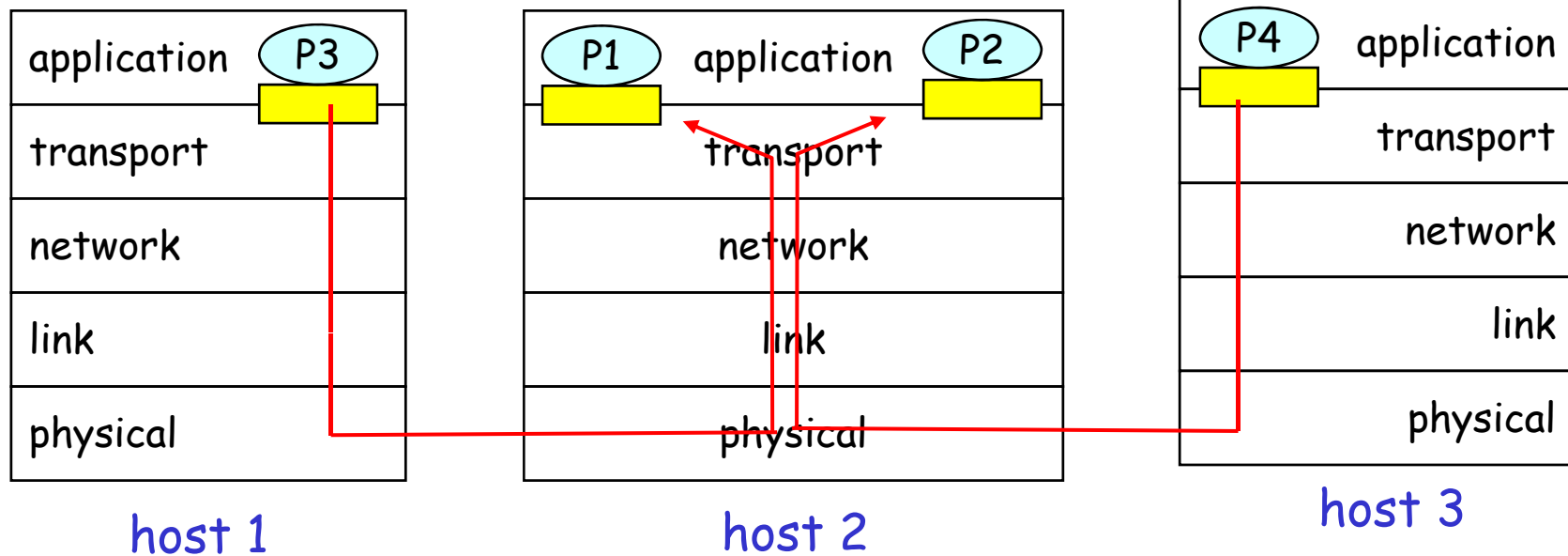
Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

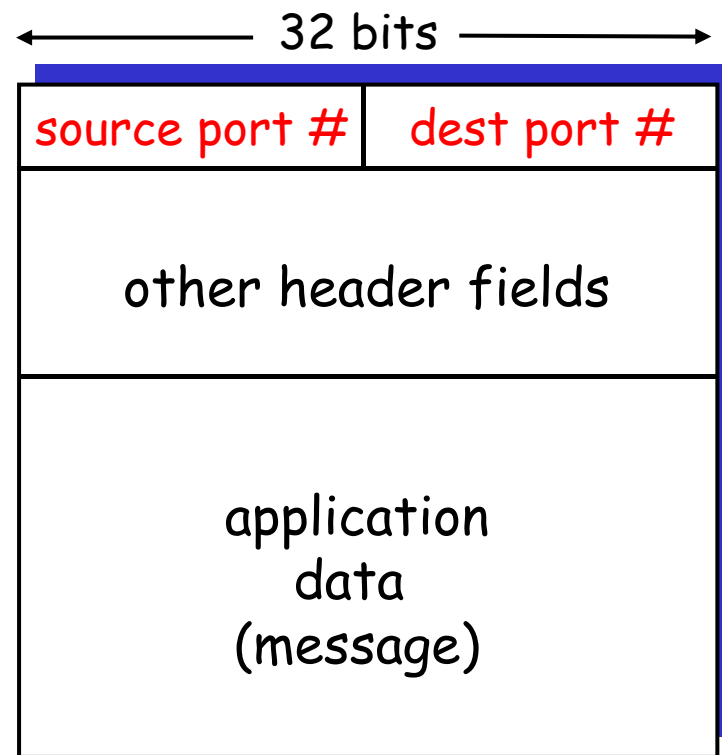
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

■ = socket ○ = process



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(12534);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(12535);
```

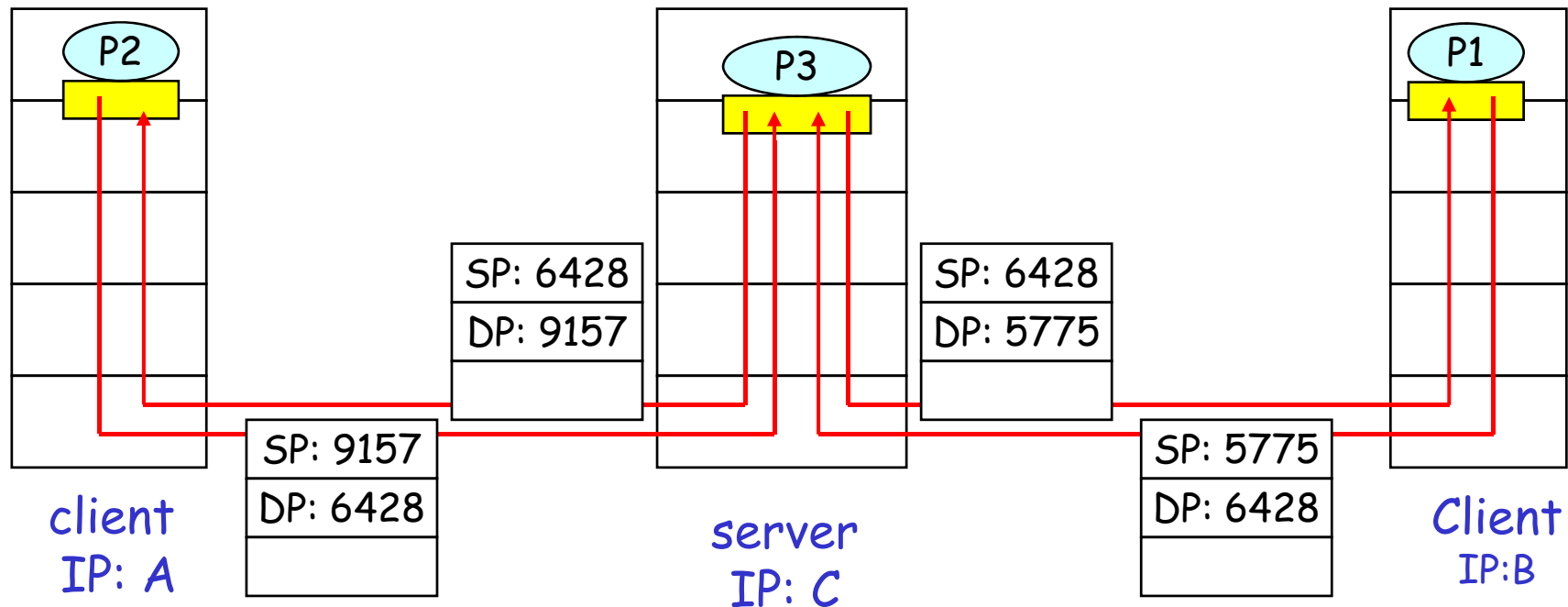
- ❑ UDP socket identified by two-tuple:

(dest IP address, dest port number)

- ❑ 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 directed to same socket

Connectionless demux (cont)

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

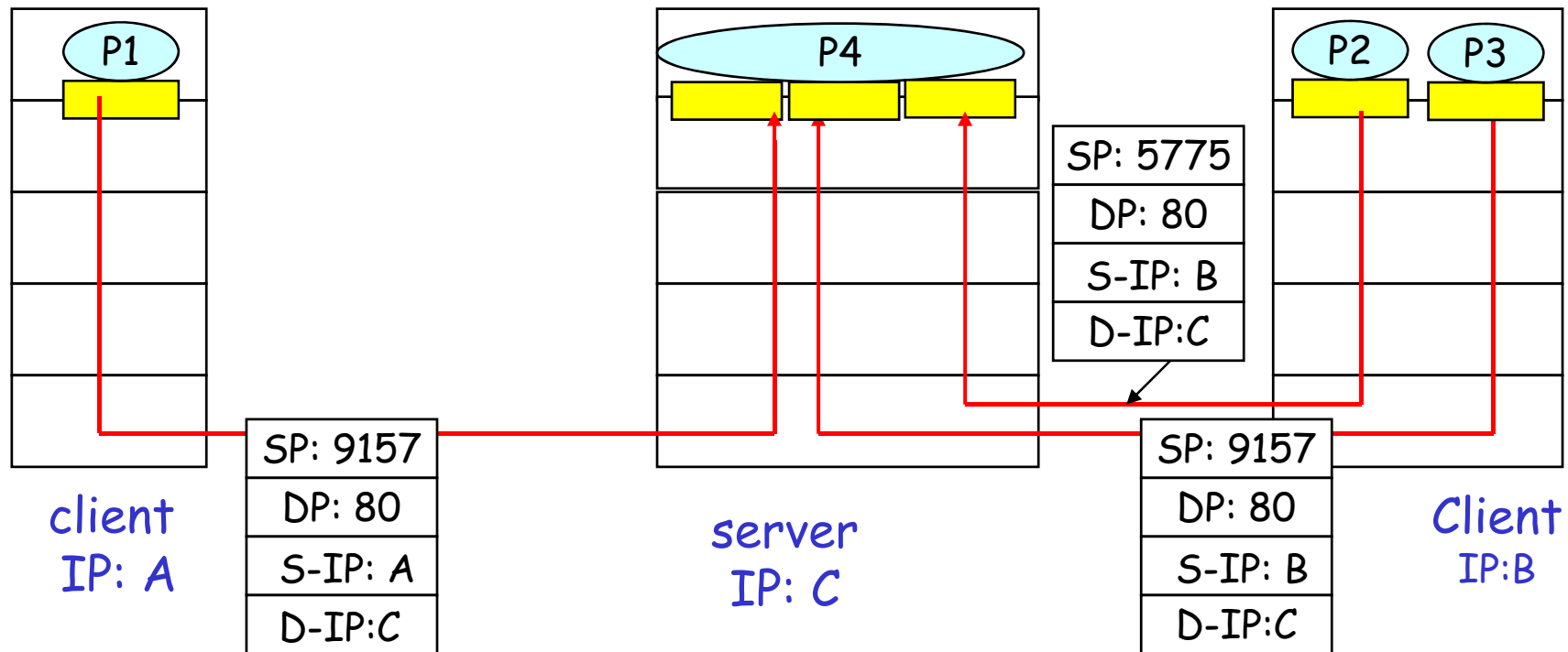


SP provides "return address"

Connection-oriented demux

- ❑ TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❑ receiving 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: Threaded Web Server



Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

UDP: User Datagram Protocol [RFC 768]

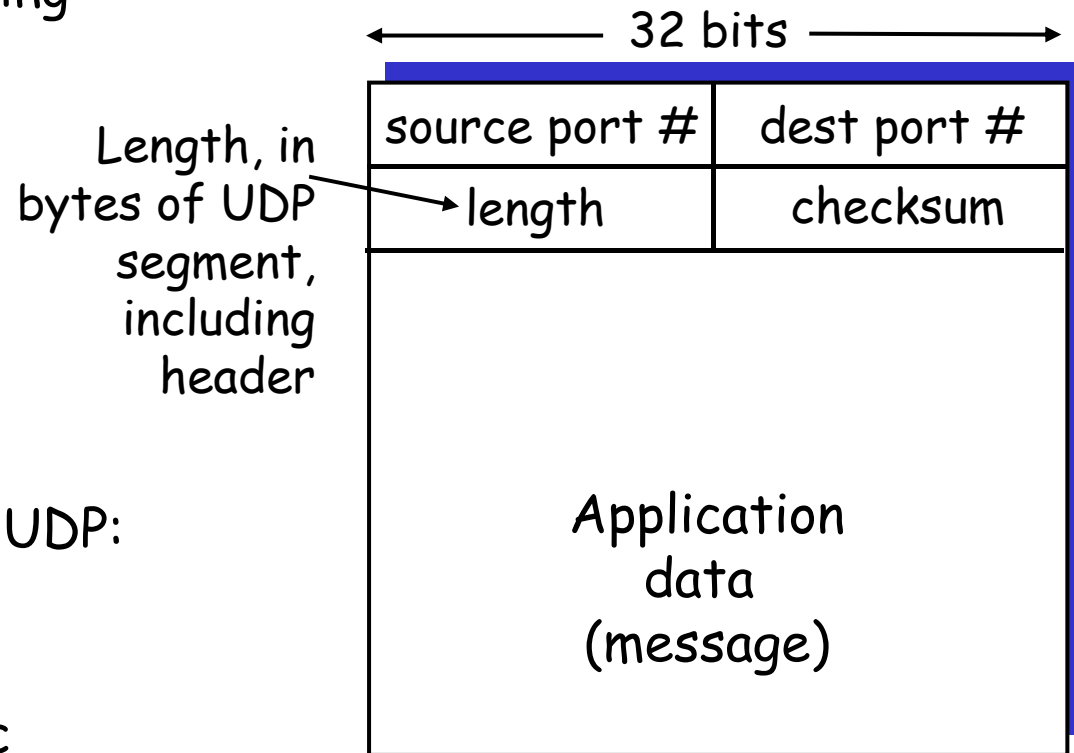
- ❑ “no frills,” “bare bones” Internet transport protocol
- ❑ “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to app
- ❑ *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

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

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 |
| | | <hr/> | | | | | | | | | | | | | | | |
| wraparound | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| | | <hr/> | | | | | | | | | | | | | | | |
| 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 |

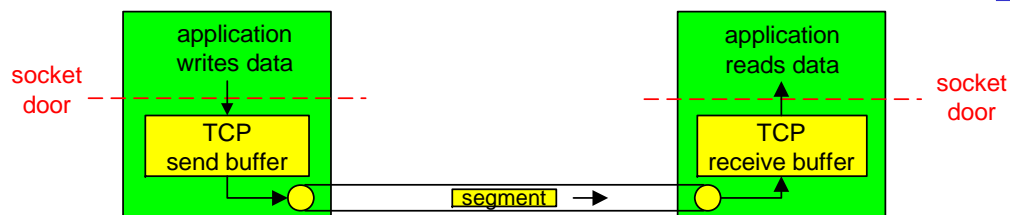
Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

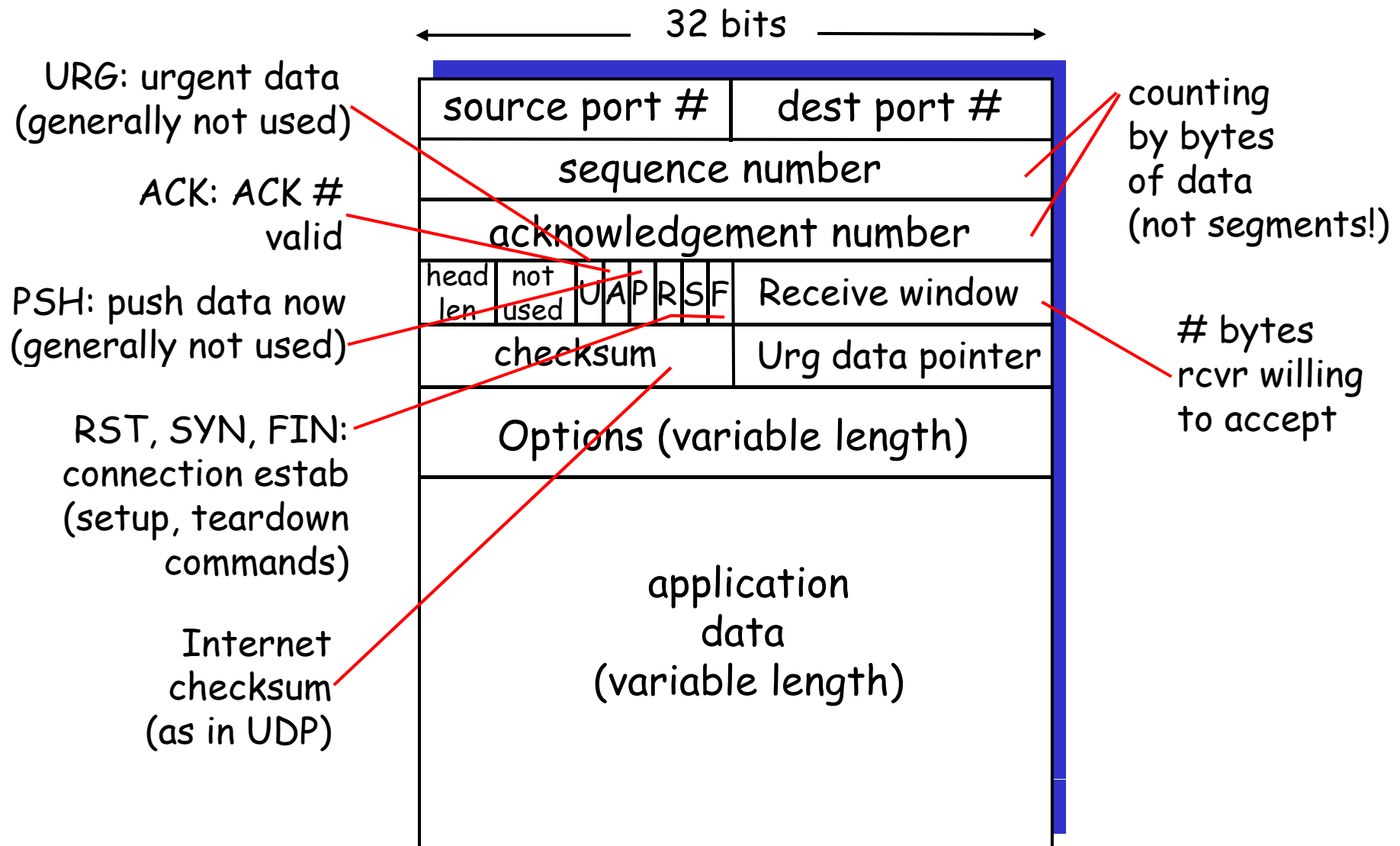
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- ❑ **point-to-point:**
 - one sender, one receiver
 - ❑ **reliable, in-order *byte stream*:**
 - 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 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 implementer

