CS43: Computer Networks
The Transport Layer & UDP

Kevin Webb
Swarthmore College

October 3, 2017

Transport Layer

- Moving “down” a layer.
- Current perspective:
  - Application is the boss…
  - Usually executing within the OS kernel.
  - The network layer is ours to command!

Network Layer (Context)

- What it does: finds paths through network
  – *Routing* from one end host to another

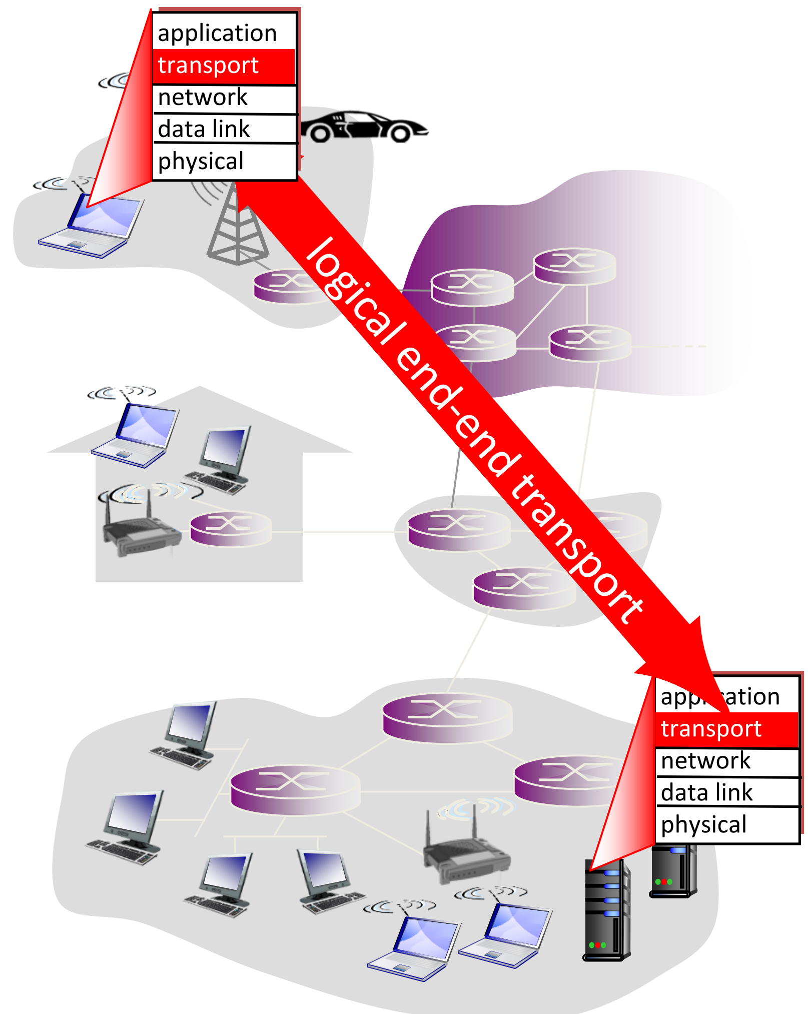
- What it doesn't:
  – Reliable transfer: "best effort delivery"
  – Guarantee paths
  – Arbitrate transfer rates

- For now, think of the network layer as giving us an "API" with one function: sendtohost(data, host). Promise: the data will go there. Usually.

Transport services and protocols

- Provides *logical communication* between processes.

- Runs in end systems.
  - Sender: breaks application messages into *segments*, passes to network layer
  - Receiver: reassembles segments into messages, passes to app layer
  - Exports services to application that network layer does not provide



How many of these services might we provide at the transport layer? Which?

- Reliable transfers
- Error detection
- Error correction
- Bandwidth guarantees
- Latency guarantees
- Encryption
- Message ordering
- Link sharing fairness

A. 4 or fewer
B. 5
C. 6
D. 7
E. All 8

How many of these services might we provide at the transport layer? Which?

- Reliable transfers (T)
- Error detection (U, T)
- Error correction (T)
- Bandwidth guarantees
- Latency guarantees
- Encryption
- Message ordering (T)
- Link sharing fairness (T)

Critical question: Can it be done at the end host?

A. 4 or fewer
B. 5
C. 6
D. 7
E. All 8

TCP sounds great! UDP…meh. Why do we need it?

A. It has good performance characteristics.

B. Sometimes all we need is error detection.

C. We still need to distinguish between sockets.

D. It basically just fills a gap in our layering model.

Adding Features

- Nothing comes for free

- Data given by application

- Apply header
  - Keeps transport state
  - Attached by sender
  - Decoded by receiver

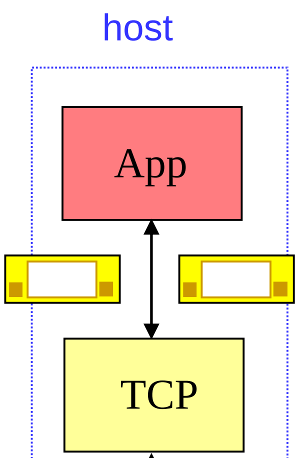
(TCP) Overhead

- Establishing state (making a connection)
  - Recall HTTP 1.0 vs. HTTP 1.1
  - Extra communication round trip

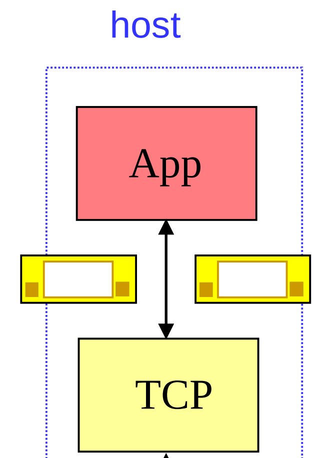
- Delays due to loss / reordering.

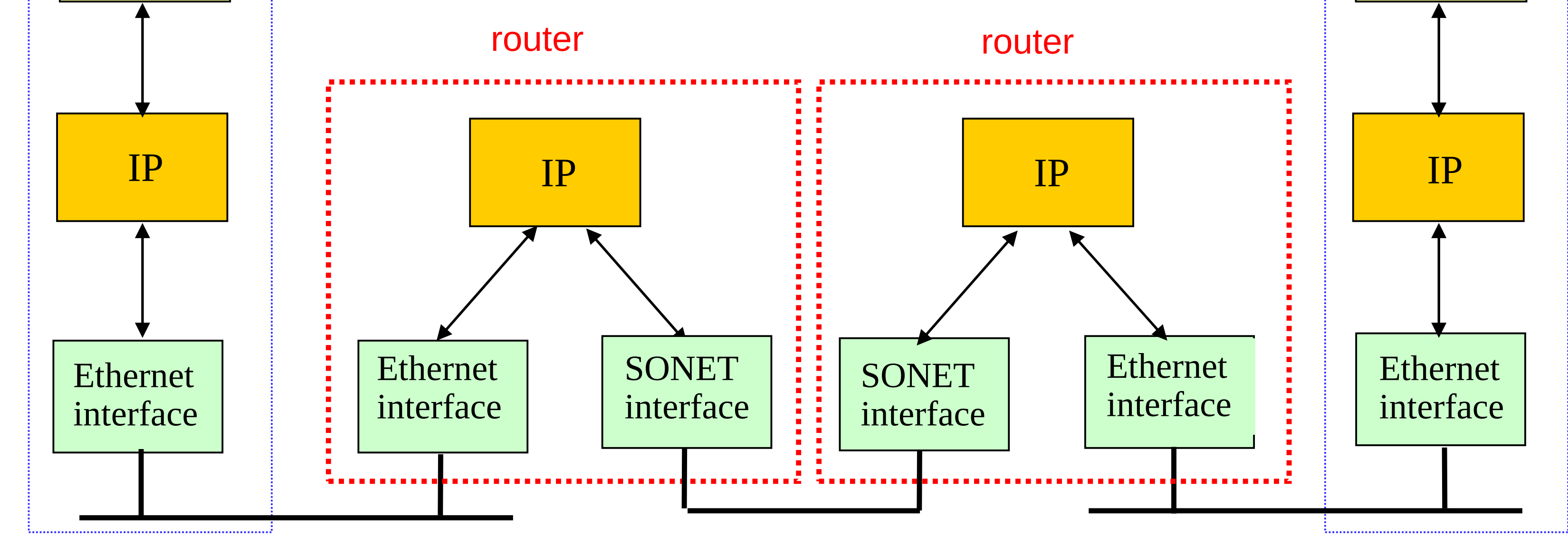
- Playing fair might cost you!

Recall: Addressing Sockets



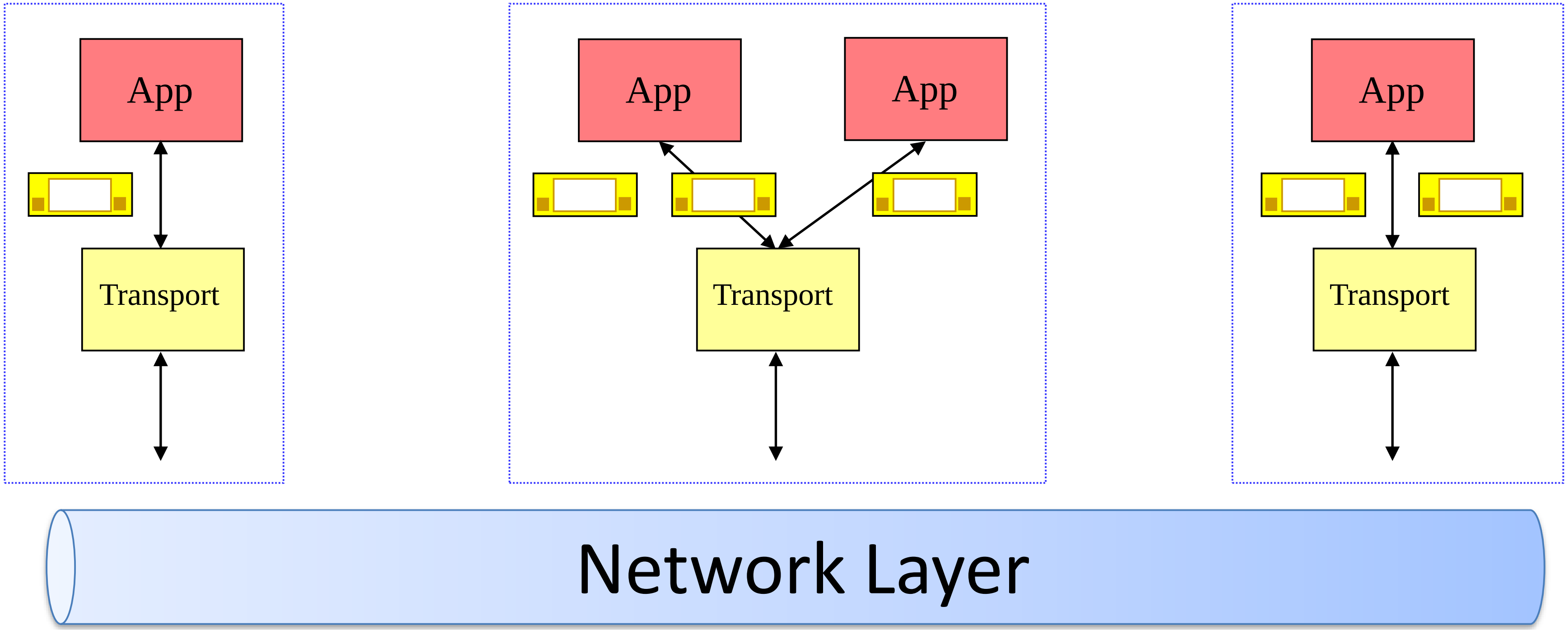
- IP address identifies device interface
- Need another identifier: port
  - 16-bit, unsigned integer value
  - Differentiates sockets





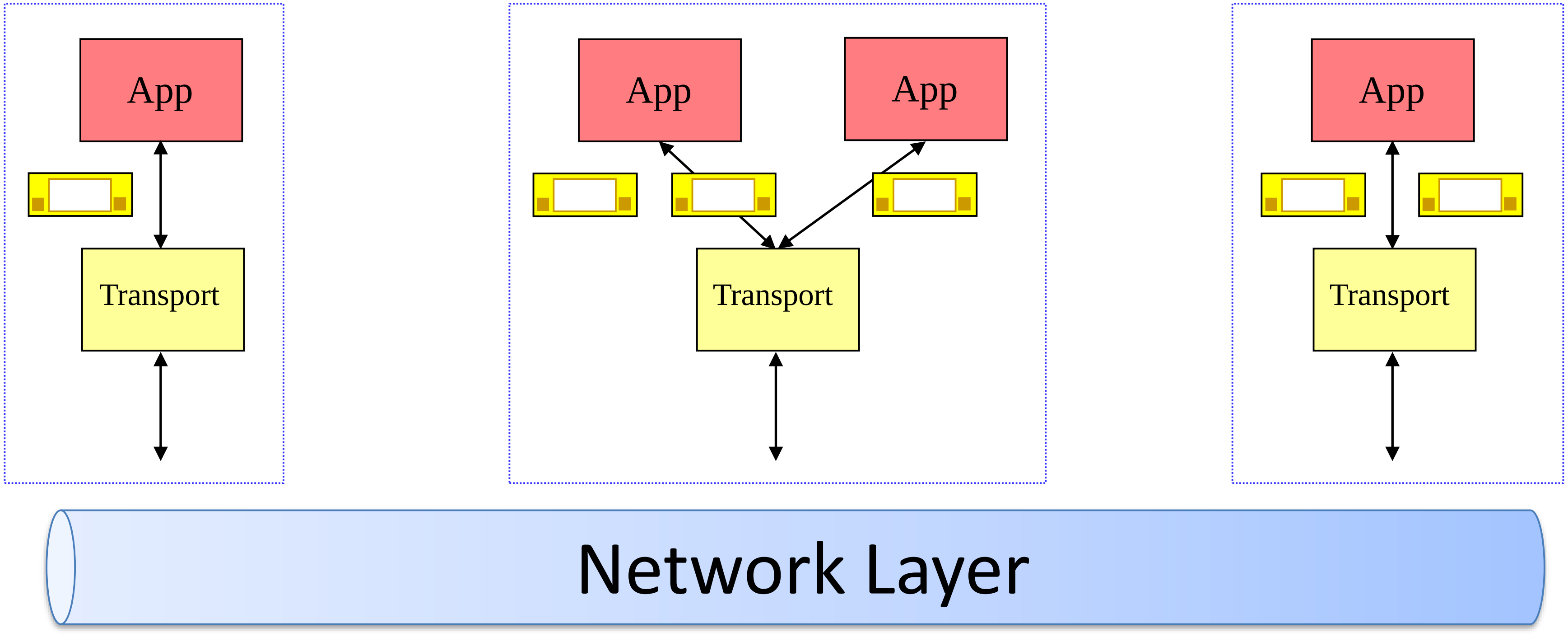
Multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)



Multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)

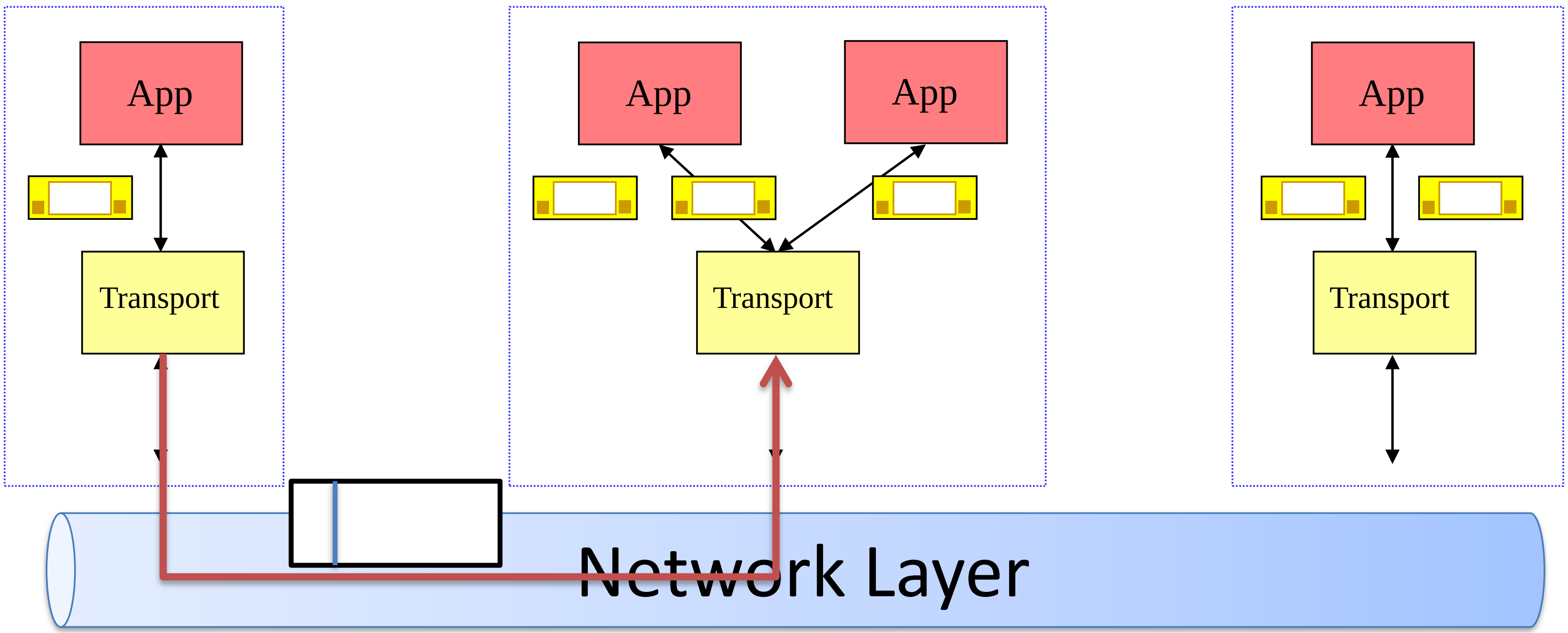


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Senders mark segments, in header, with identifier (port)

Multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)

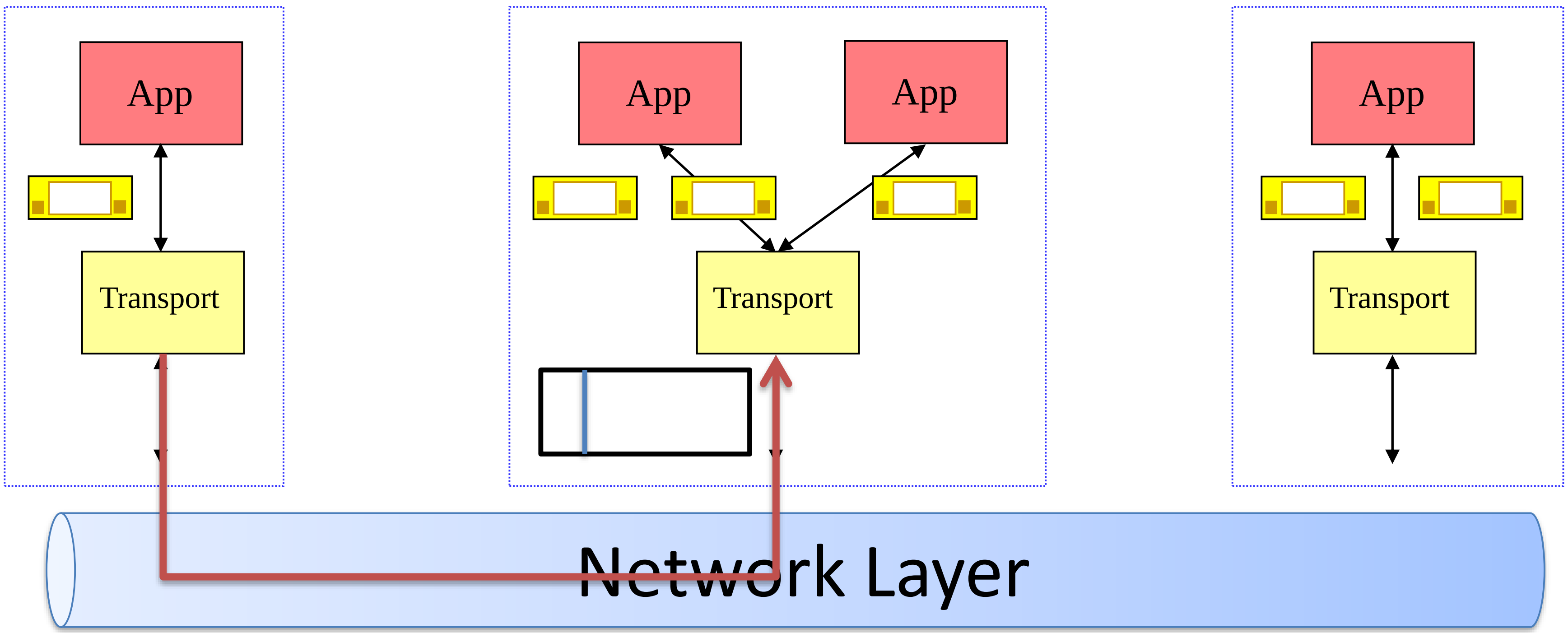


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Senders mark segments, in header, with identifier (port)

De-multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)

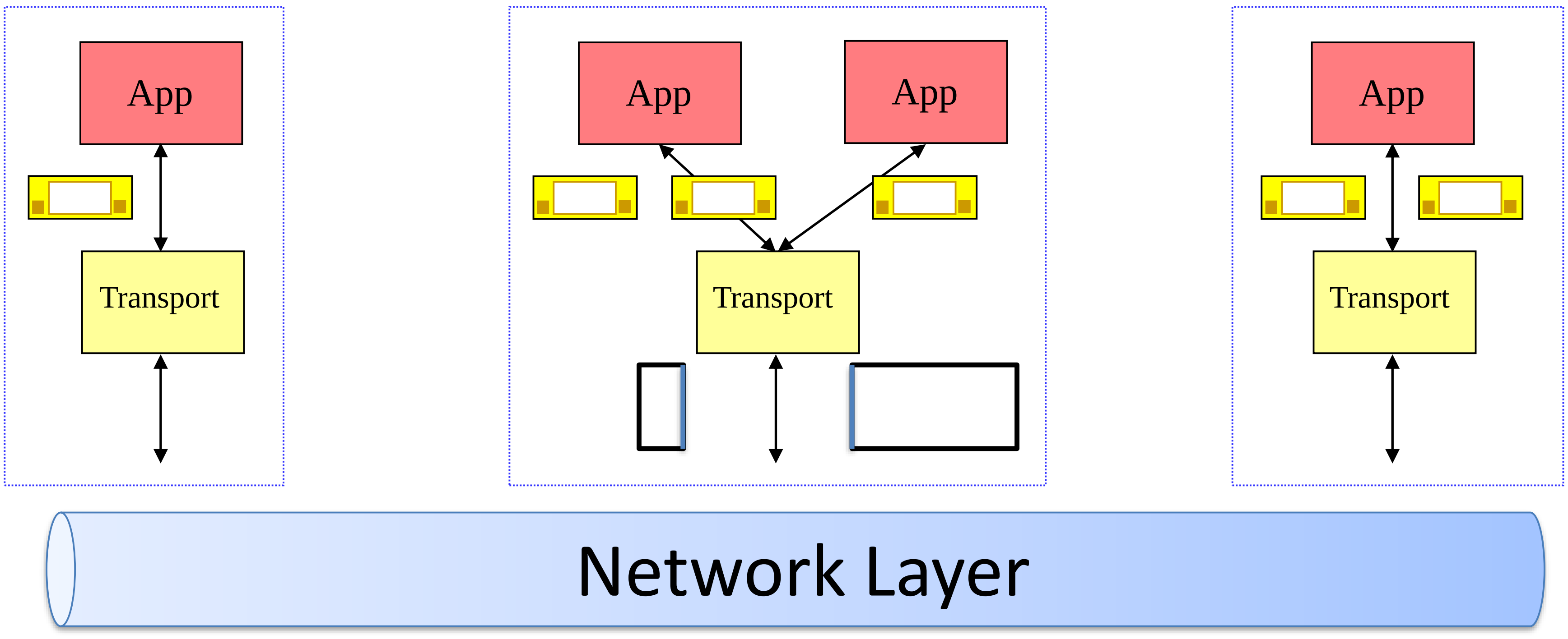


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

De-multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)

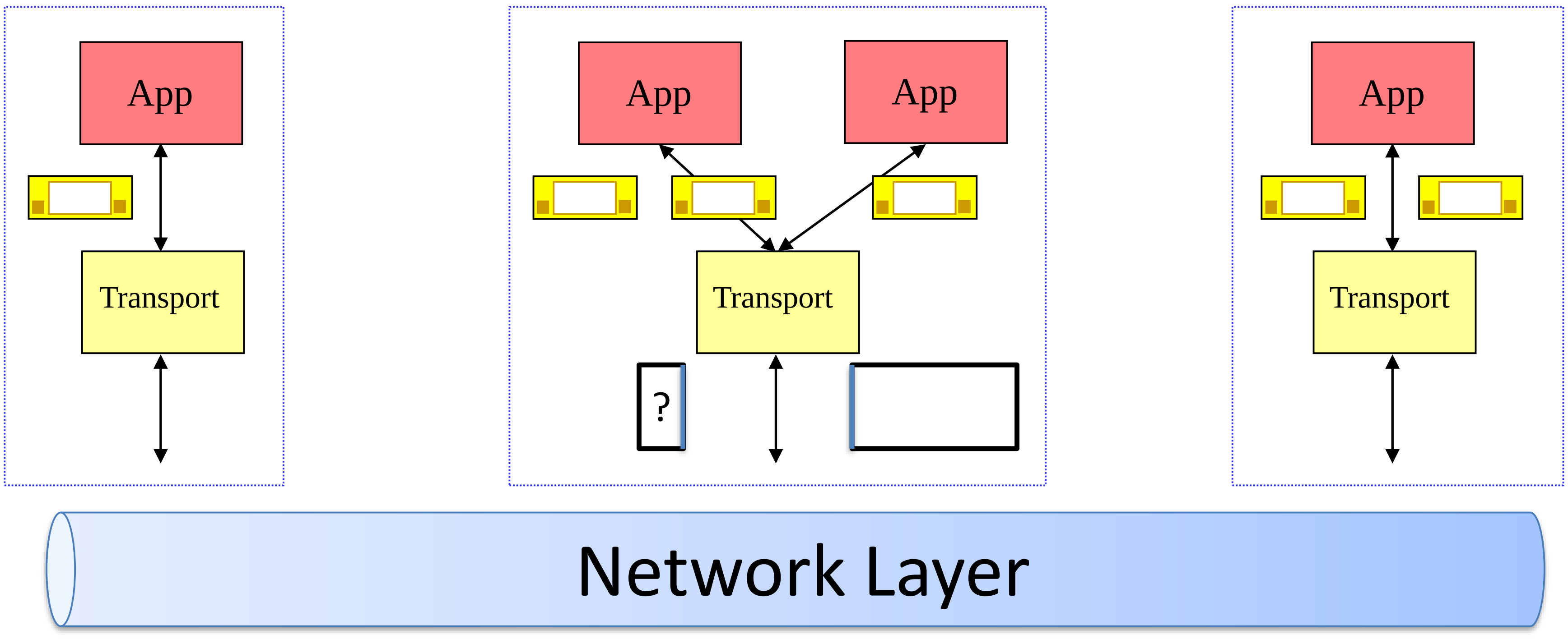


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

De-multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)

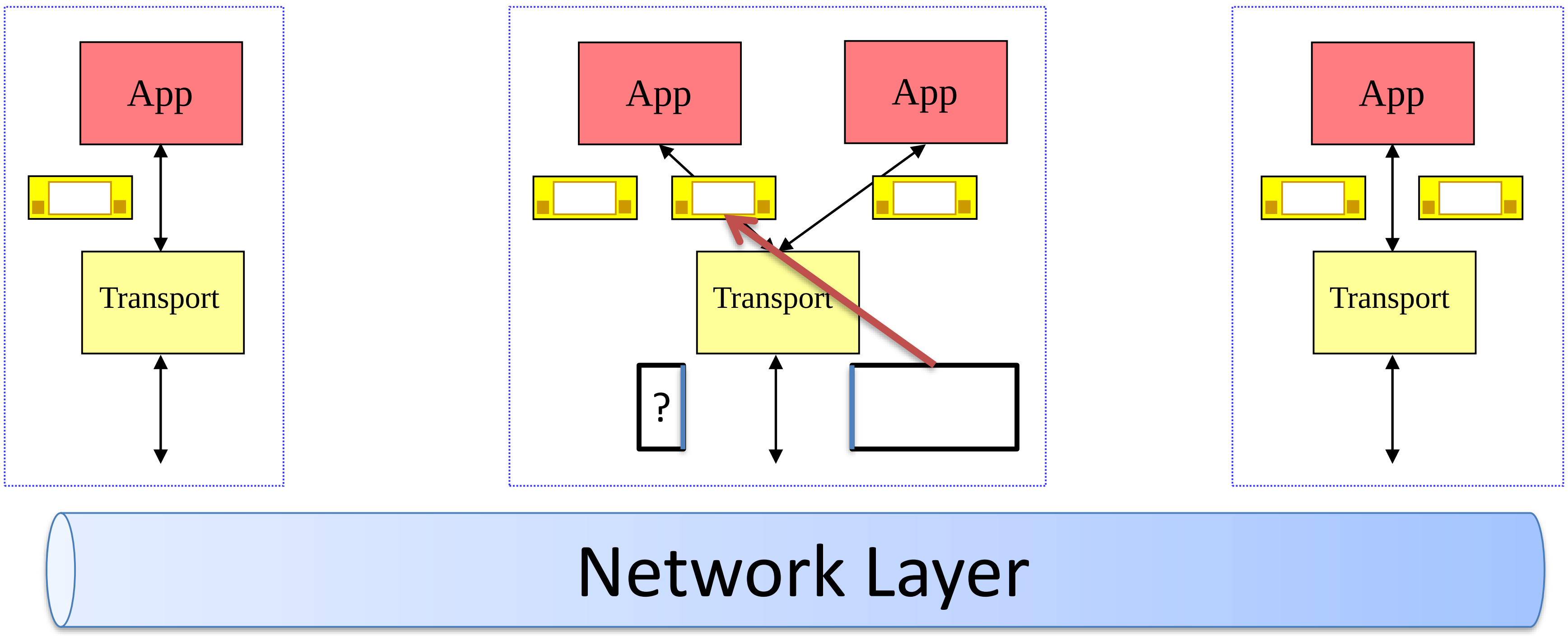


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

De-multiplexing

(Simultaneous transmission of two or more signals/messages over a single channel.)



- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

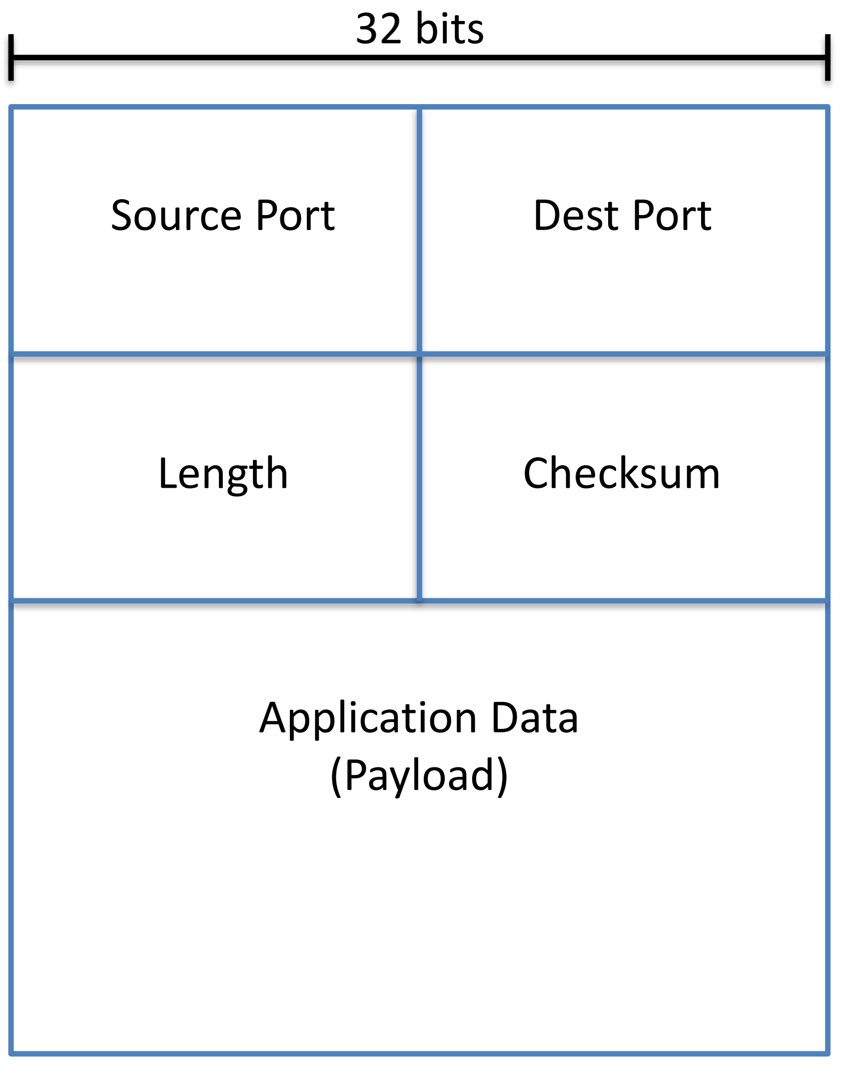
UDP: User Datagram Protocol [RFC 768]

- "No frills," "Bare bones" Internet transport protocol
  - RFC 768 (1980)
  - Length of the document?

- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order
  - (Same as underlying network layer)

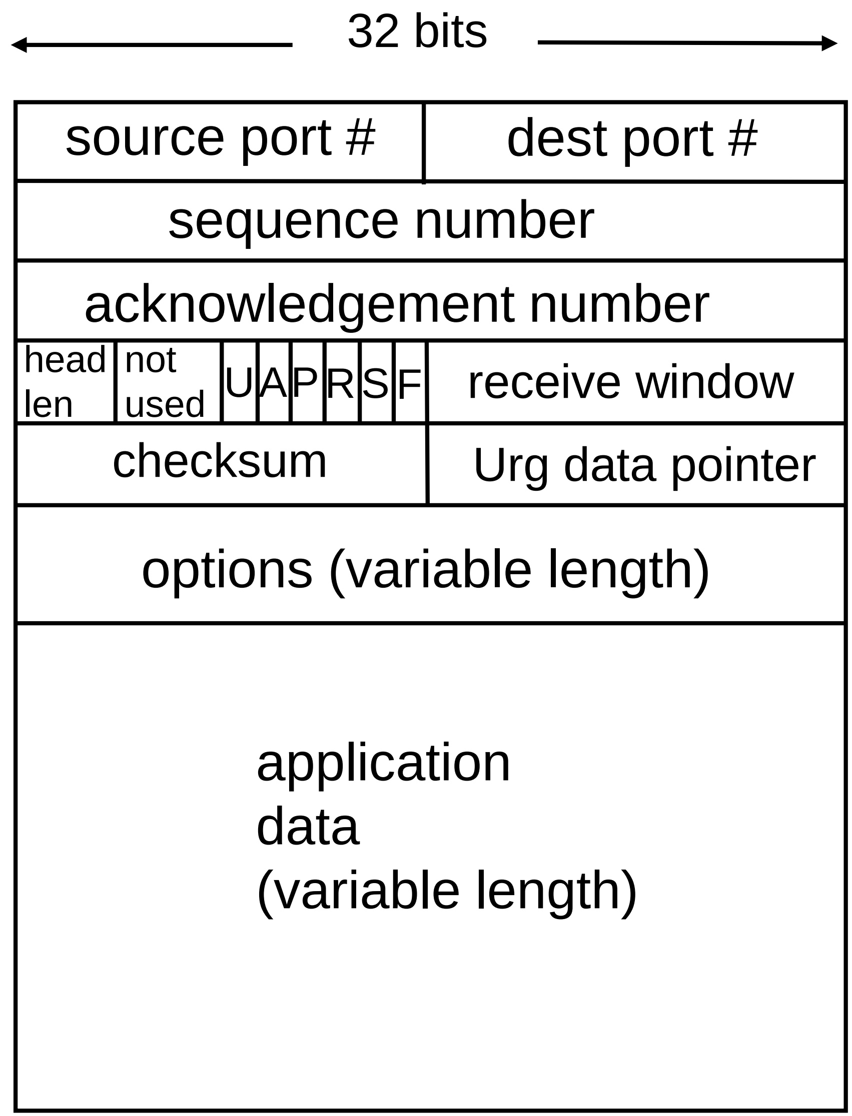
- *Connectionless:*
  - No initial state transferred between parties (no handshake)
  - Each UDP segment is handled independently

UDP Segment

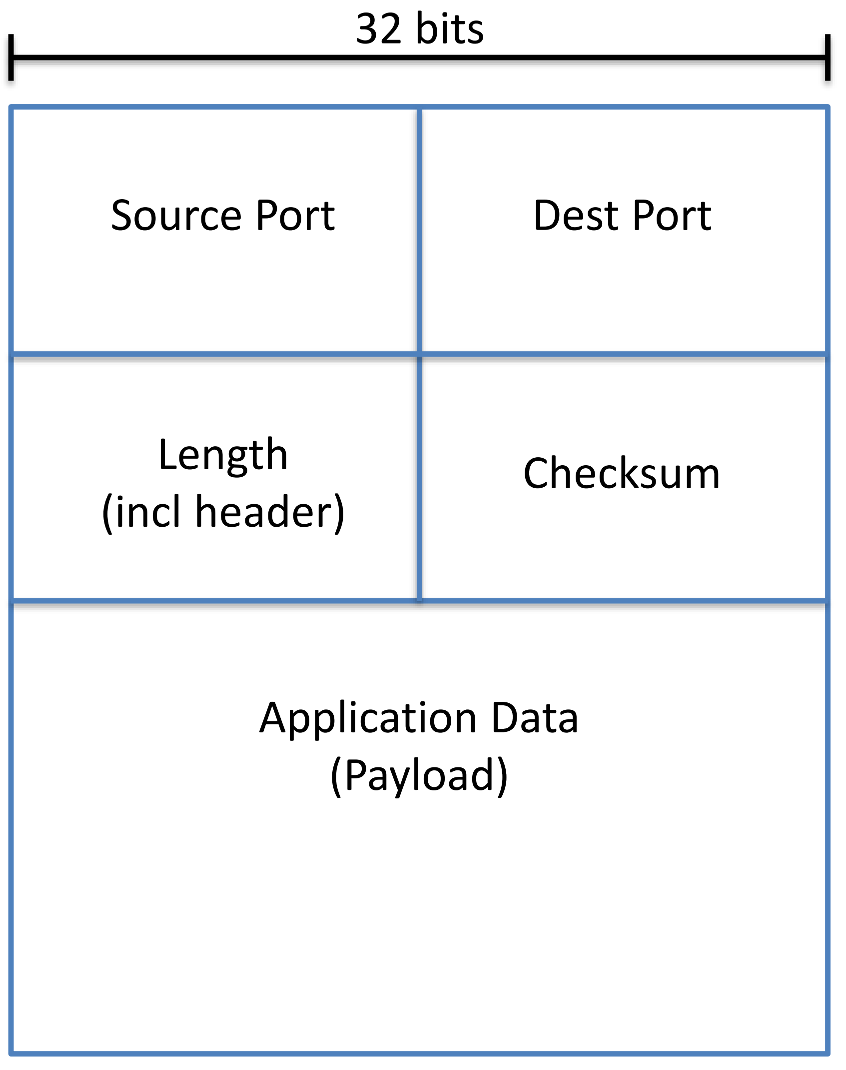


32 bits
Source Port
Dest Port
Length
Checksum
Application Data
(Payload)

TCP Segment


32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)

UDP Segment


32 bits
Source Port
Dest Port
Length
(incl header)
Checksum
Application Data
(Payload)

UDP Checksum

- Goal: Detect transmission errors (e.g. flipped bits)
  - Router memory errors
  - Driver bugs
  - Electromagnetic interference

- RFC: “Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.”

UDP Checksum

- Goal: Detect transmission errors (e.g. flipped bits)
  - Router memory errors
  - Driver bugs
  - Electromagnetic interference

- At the sender:
  - Treat the entire segment as 16-bit integer values
  - Add them all together (sum)
  - Put the 1's complement in the checksum header field

Recall CS31

- In bitwise compliment, all of the bits in a binary number are flipped.

- So 1111000011110000 -> 0000111100001111

Checksum 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

Receiver

- Add all the received data together as 16-bit integers
- Add that to the checksum
- If result is not 1111 1111 1111 1111, there are errors!

If our checksum addition yields all ones, are we guaranteed to be error-free?

A. Yes

B. No

UDP Applications

- Latency sensitive
  - Quick request/response (DNS)
  - Network management (SNMP, TFTP)
  - Voice/video chat
- Error correction unnecessary (periodic msgs)
- Communicating with *lots* of others

What if you want something more reliable than UDP, but faster/not as full featured as TCP?

A. Sorry, you’re out of luck.

B. Write your own transport protocol.

C. Add in the features you want at the application layer.

TCP: send() Blocking

- Recall: With TCP, send() blocks if buffer full.

UDP: sendto() Blocking?

- Recall: With TCP, send() blocks if buffer full.

- Does UDP need to block? Should it?

A. Yes, if buffers are full, it should.
B. It doesn't need to, but it might be useful.
C. No, it does not need to and shouldn't do so.

Summary

- UDP: No frills transport protocol.
- Simple, 8-byte header with ports, len, cksum
- Checksum protects against most bit flips