CS 43: Computer Networks

11: CDNs and Transport Layer

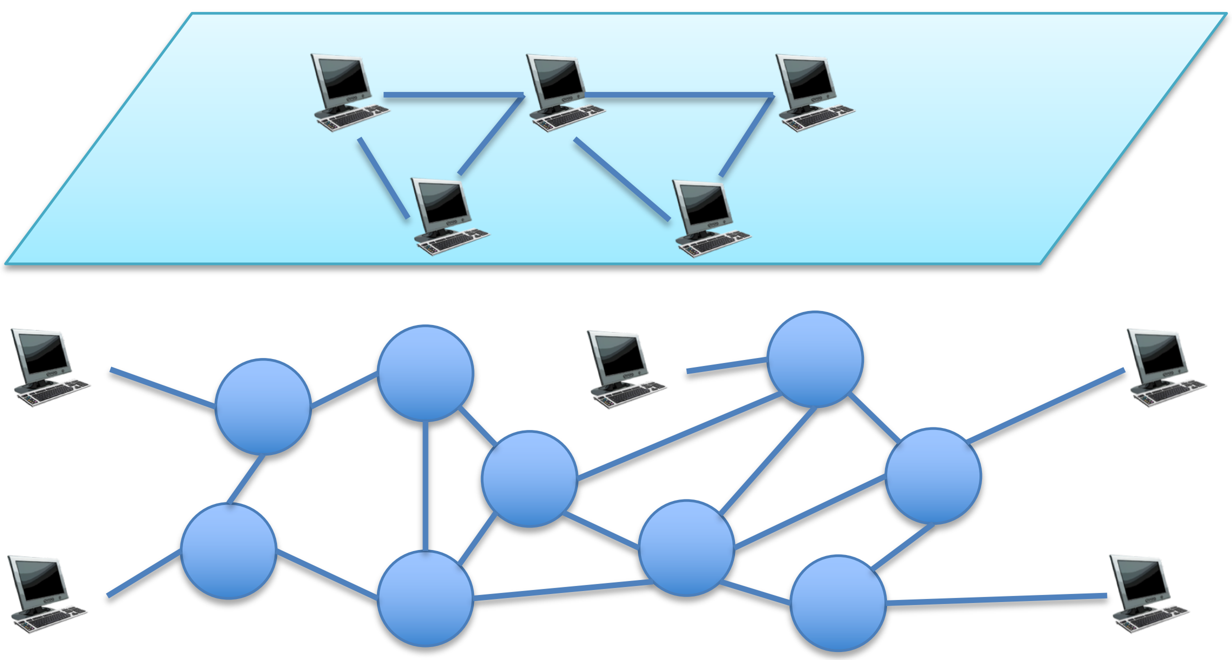
October 8, 2019

SWARTHMORE COLLEGE

Reading Quiz

Overlay Network

- A network made up of “virtual” or logical links
- Virtual links map to one or more physical links



High-Performance Content Distribution

- Problem:
  You have a service that supplies lots of data. You want good performance for all users!

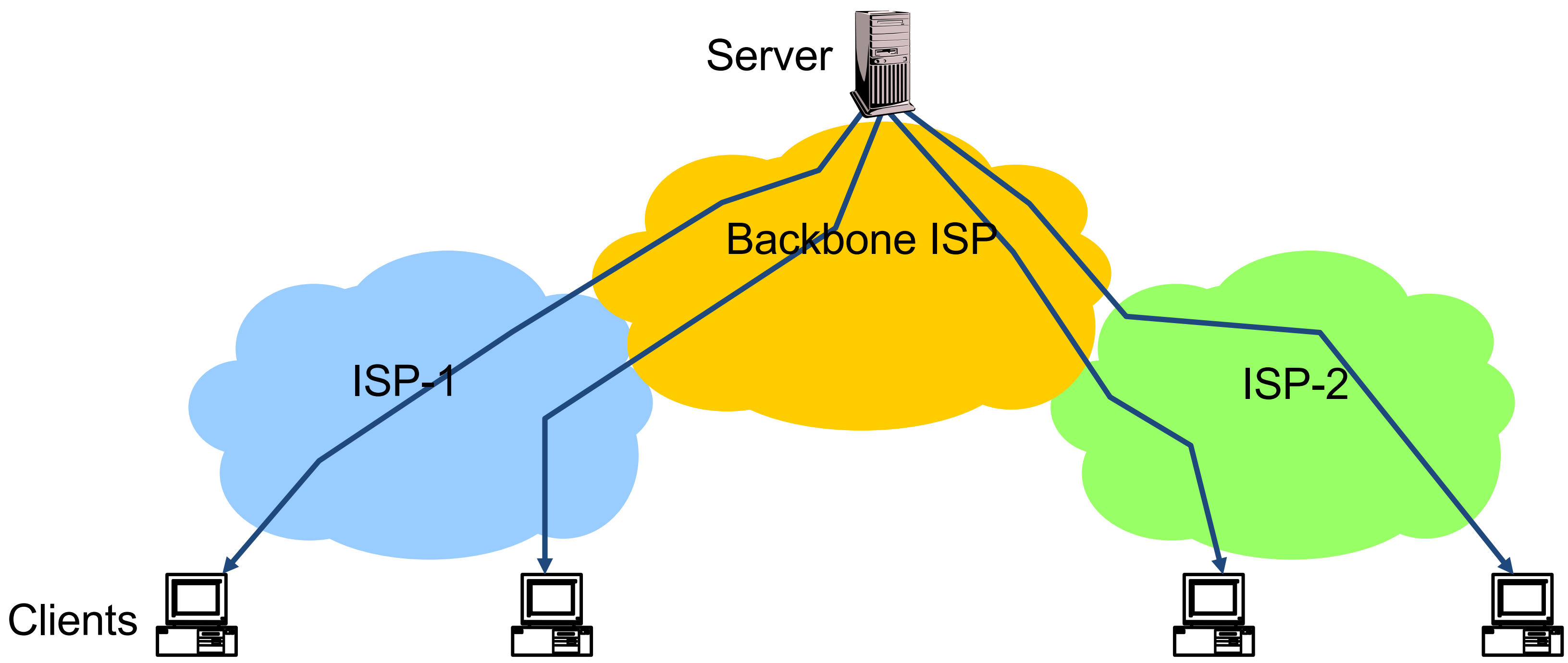
(often "lots of data" means media files)

Key Components of a CDN

- Distributed servers
  - Usually located inside of other ISPs
  - Often located in IXPs (coming up next)
- High-speed network connecting them
- Clients (eyeballs)
  - Can be located anywhere in the world
  - They want fast web performance
- Glue
  - Something that binds clients to “nearby” replica servers

CDN Challenges

– How do we direct the user to a nearby replica instead of the centralized source?
– How do we determine which replica is the best to send them to?
– Ensure that replicas are always available?

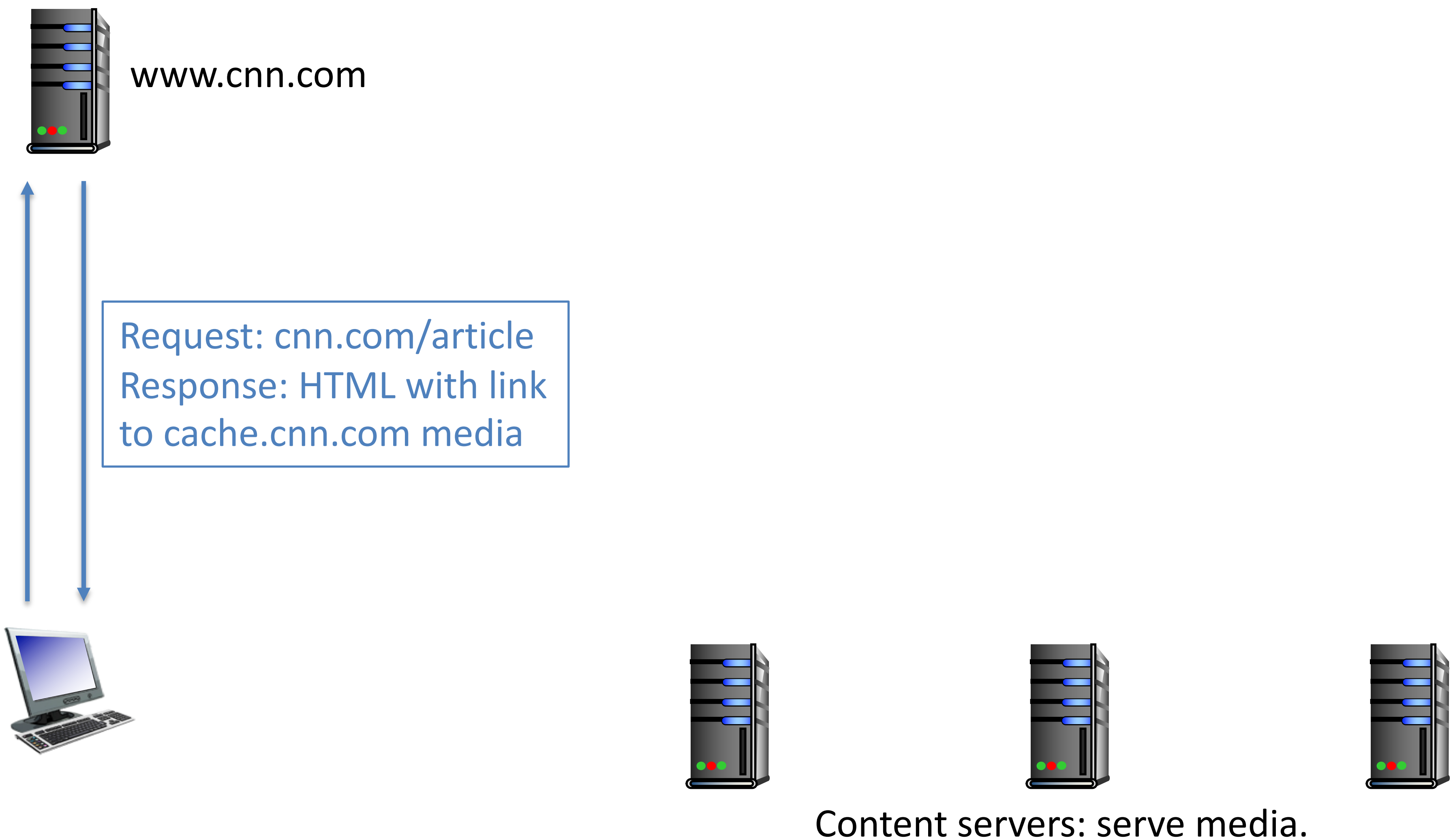


Challenge: Finding the CDN

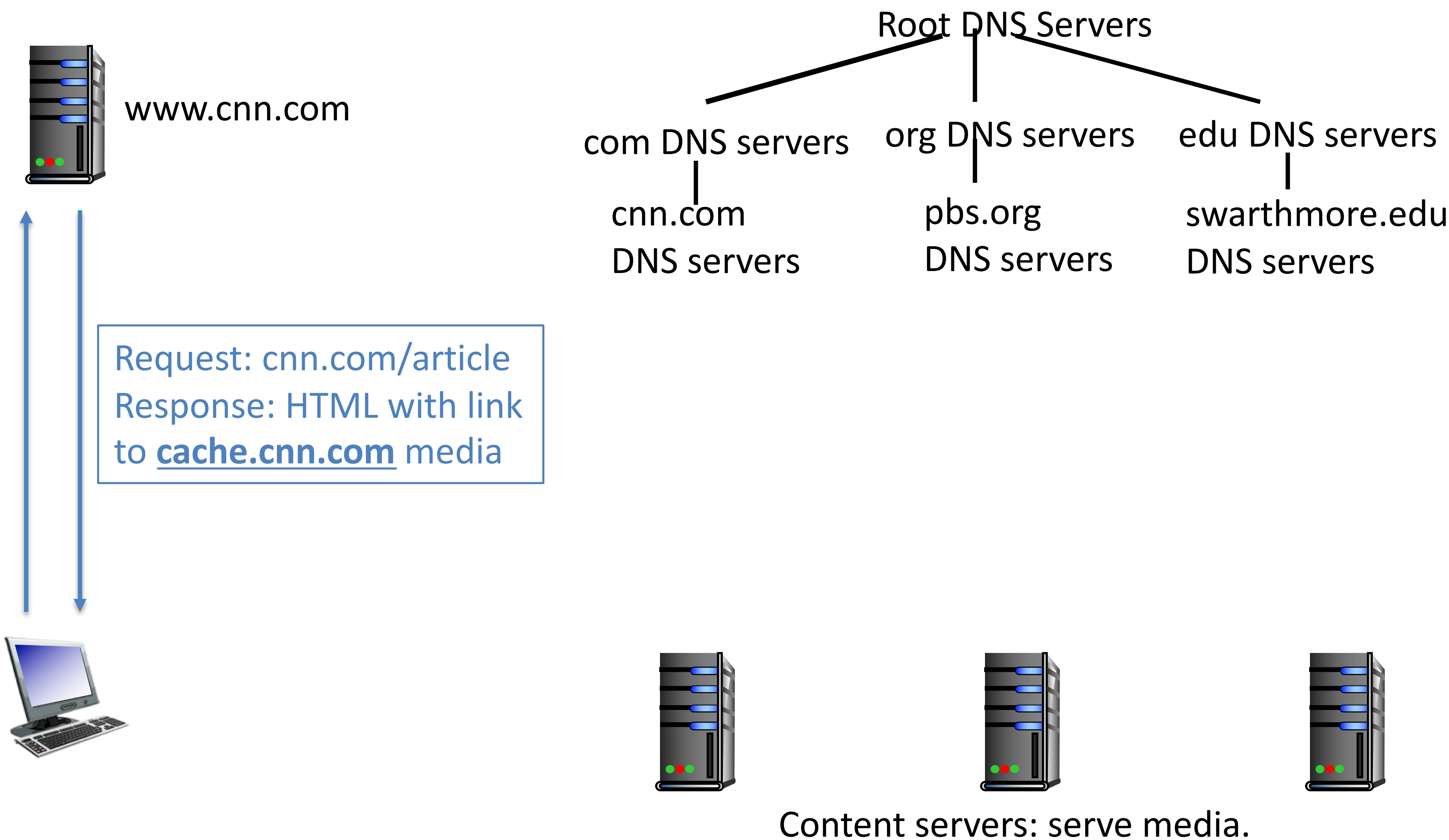
- Three main options:
  - Application redirect (e.g., HTTP)
  - “Anycast” routing
  - DNS resolution (most popular in practice)

- Example: CNN + Akamai

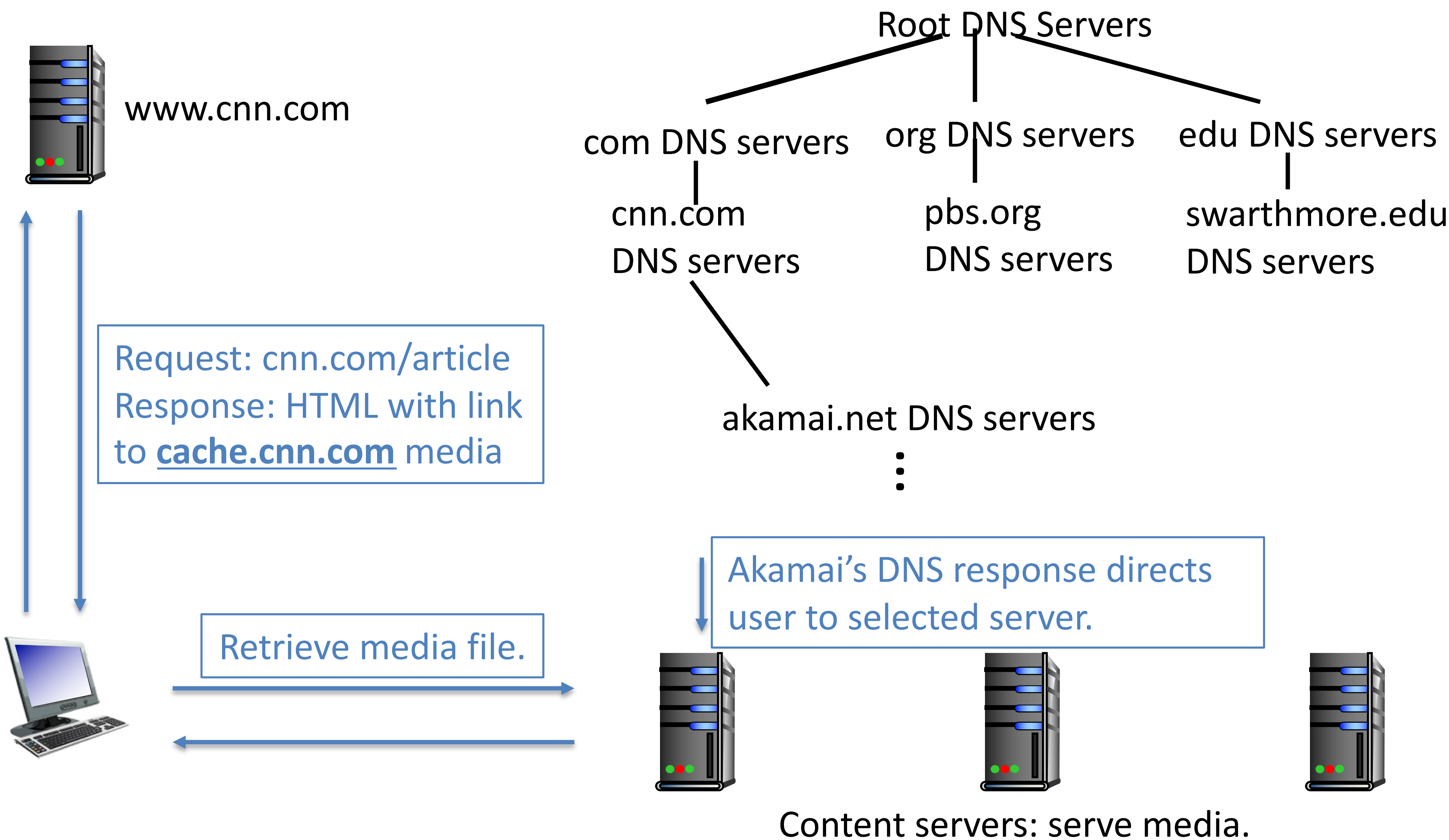
CNN + Akamai


www.cnn.com
Request: cnn.com/article
Response: HTML with link to cache.cnn.com media
Content servers: serve media.

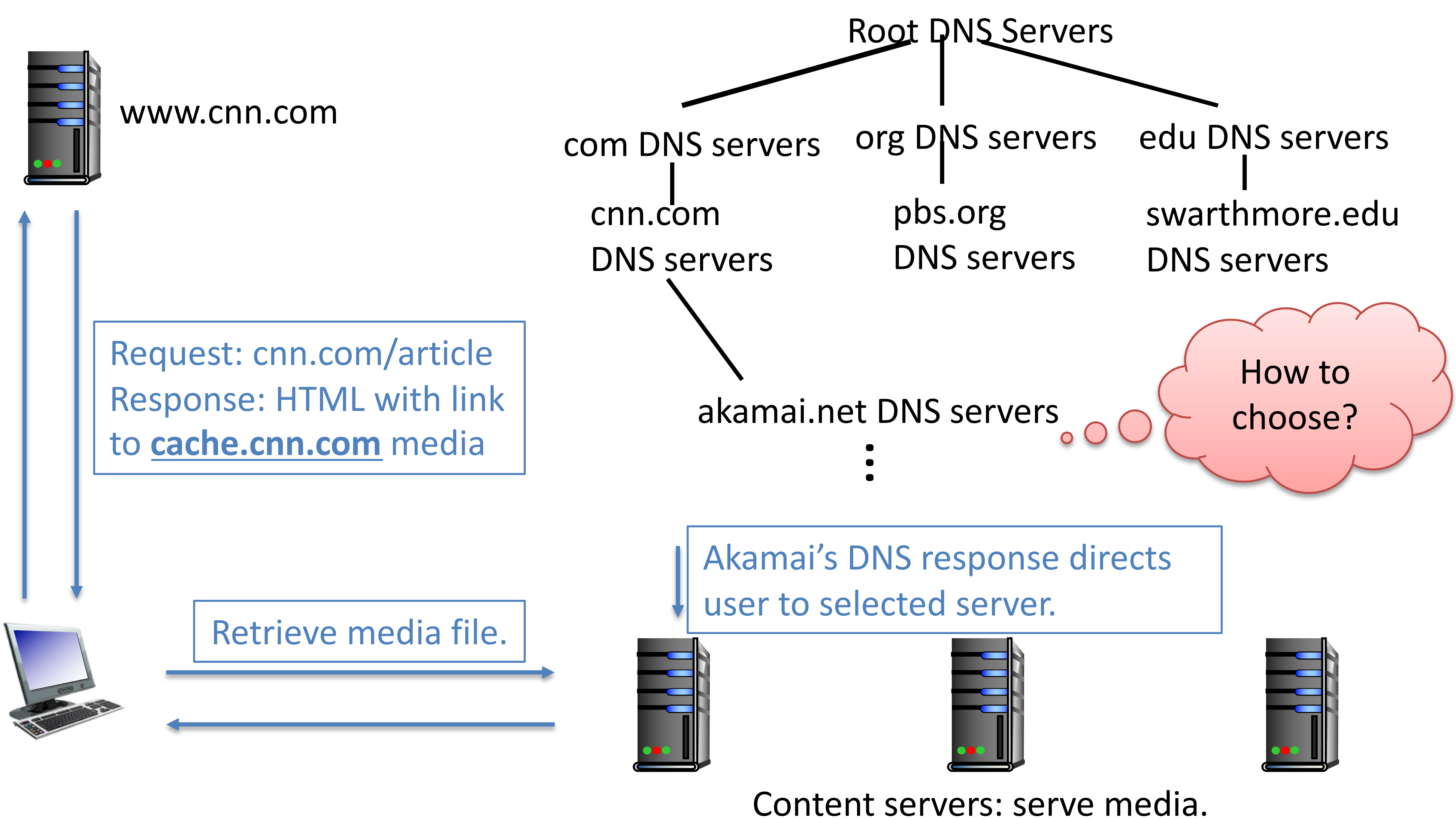
CNN + Akamai


Root DNS Servers
com DNS servers
org DNS servers
edu DNS servers
cnn.com DNS servers
pbs.org DNS servers
swarthmore.edu DNS servers
www.cnn.com
Request: cnn.com/article
Response: HTML with link to **cache.cnn.com** media
Content servers: serve media.

CNN + Akamai


Root DNS Servers
www.cnn.com
com DNS servers
org DNS servers
edu DNS servers
cnn.com DNS servers
pbs.org DNS servers
swarthmore.edu DNS servers
Request: cnn.com/article
Response: HTML with link to cache.cnn.com media
akamai.net DNS servers
Akamai's DNS response directs user to selected server.
Retrieve media file.
Content servers: serve media.

CNN + Akamai


Root DNS Servers
www.cnn.com
com DNS servers
org DNS servers
edu DNS servers
cnn.com DNS servers
pbs.org DNS servers
swarthmore.edu DNS servers
Request: cnn.com/article
Response: HTML with link to cache.cnn.com media
akamai.net DNS servers
How to choose?
Akamai's DNS response directs user to selected server.
Retrieve media file.
Content servers: serve media.

Which metric is most important when choosing a server? (CDN or otherwise)

A. RTT latency

B. Data transfer rate / throughput

C. Hardware ownership

D. Geographic location

E. Some other metic(s)   (such as?)

This is the CDN operator's secret sauce!

Content in today's Internet

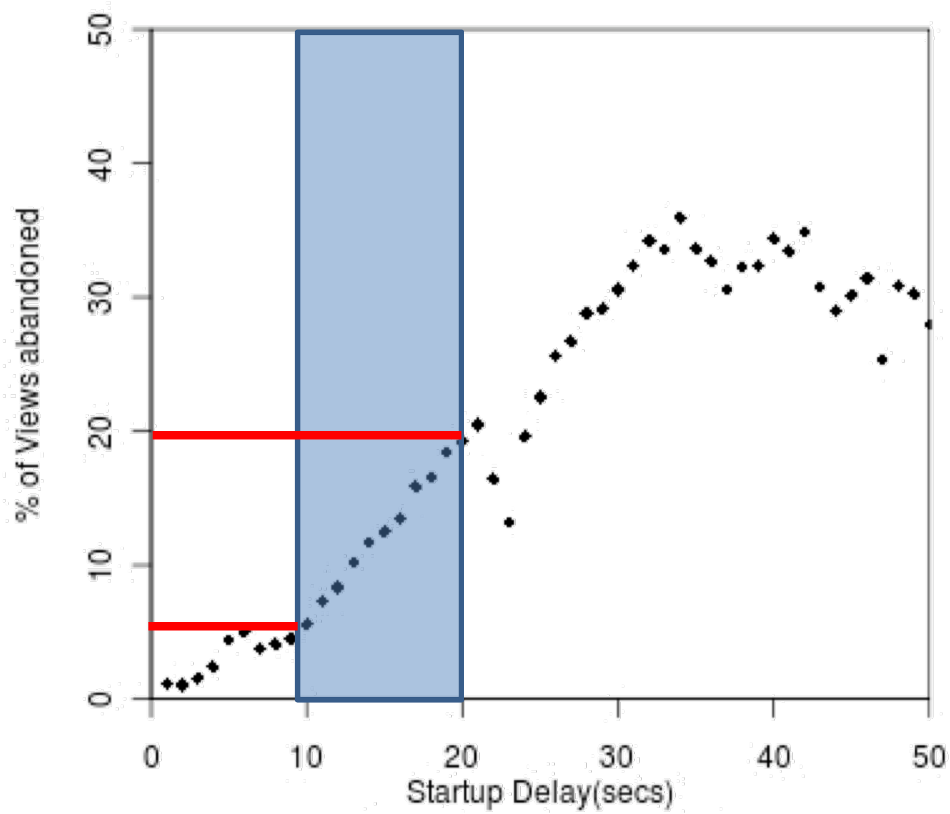
- Most flows are HTTP
  - Web is at least 52% of traffic
  - Median object size is 2.7K, average is 85K (as of 2007)

- Is the Internet designed for this common case?
  - Why?

Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay

- 4x increase in abandonment with 10s increase in delay



Streaming Media

- Straightforward approach: simple GET
- Challenges:
  - Dynamic network characteristics
  - Varying user device capabilities
  - User mobility

Dynamic Adaptive Streaming over HTTP (DASH)

- Encode several versions of the same media file
  - low / medium / high / ultra quality
- Break each file into chunks
- Create a “manifest” to map file versions to chunks / video time offset

Dynamic Adaptive Streaming over HTTP (DASH)

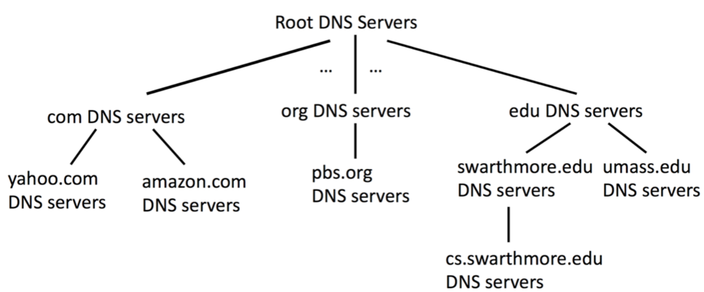
- Client requests manifest file, chooses version
- Requests new chunks as it plays existing ones
- Can switch between versions at any time!

Summary

- Peer-to-peer architectures for:
  - High performance: BitTorrent
  - Decentralized lookup: DHTs
- CDNs: locating “good” replica for media server
- DASH: streaming despite dynamic conditions

Application Layer

Does whatever an application does!



DNS

Chrome

Thunderbird

Skype

Transport Layer

Moving down a layer!

Application Layer

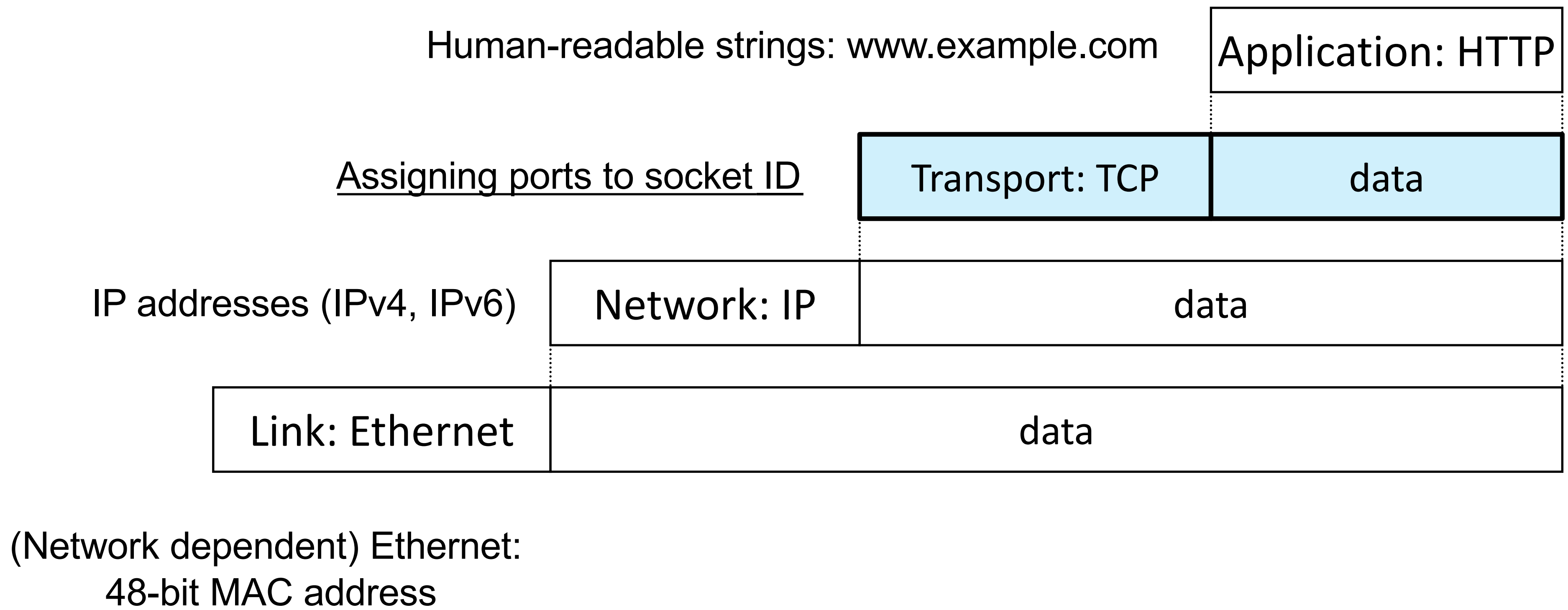
Transport: end-to-end connections, reliability

Network: routing

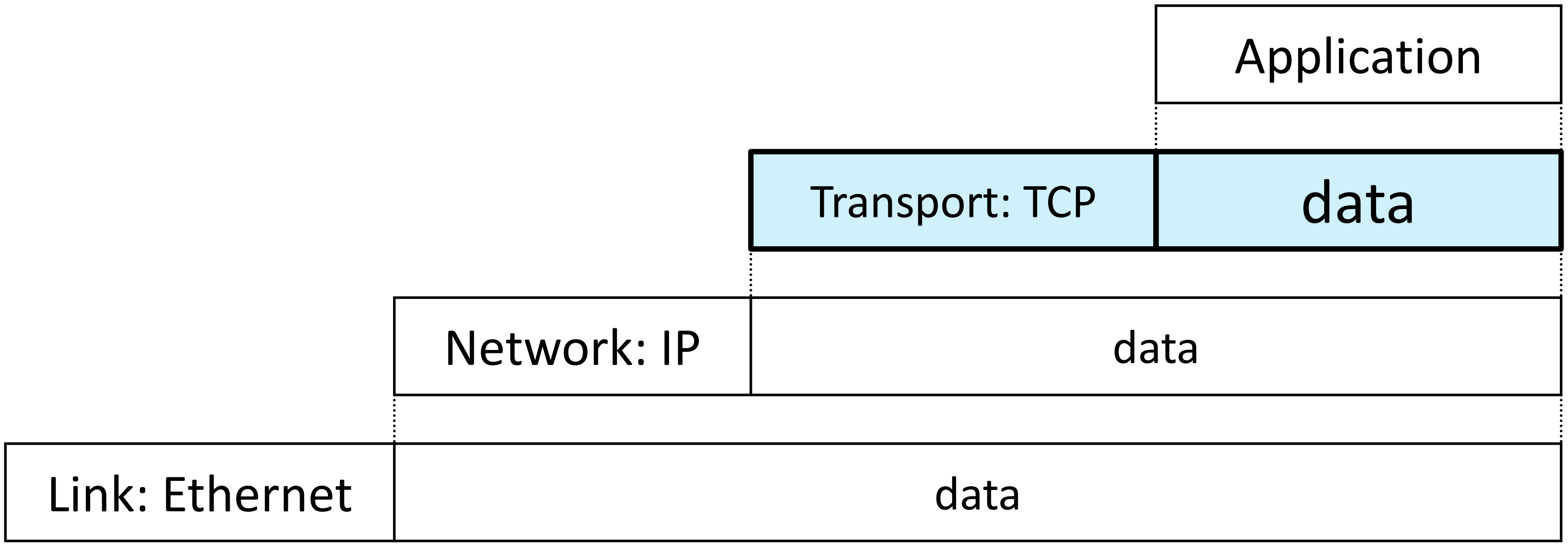
Link (data-link): framing, error detection

Physical: 1’s and 0’s/bits across a medium
(copper, the air, fiber)

Recall: Addressing and Encapsulation

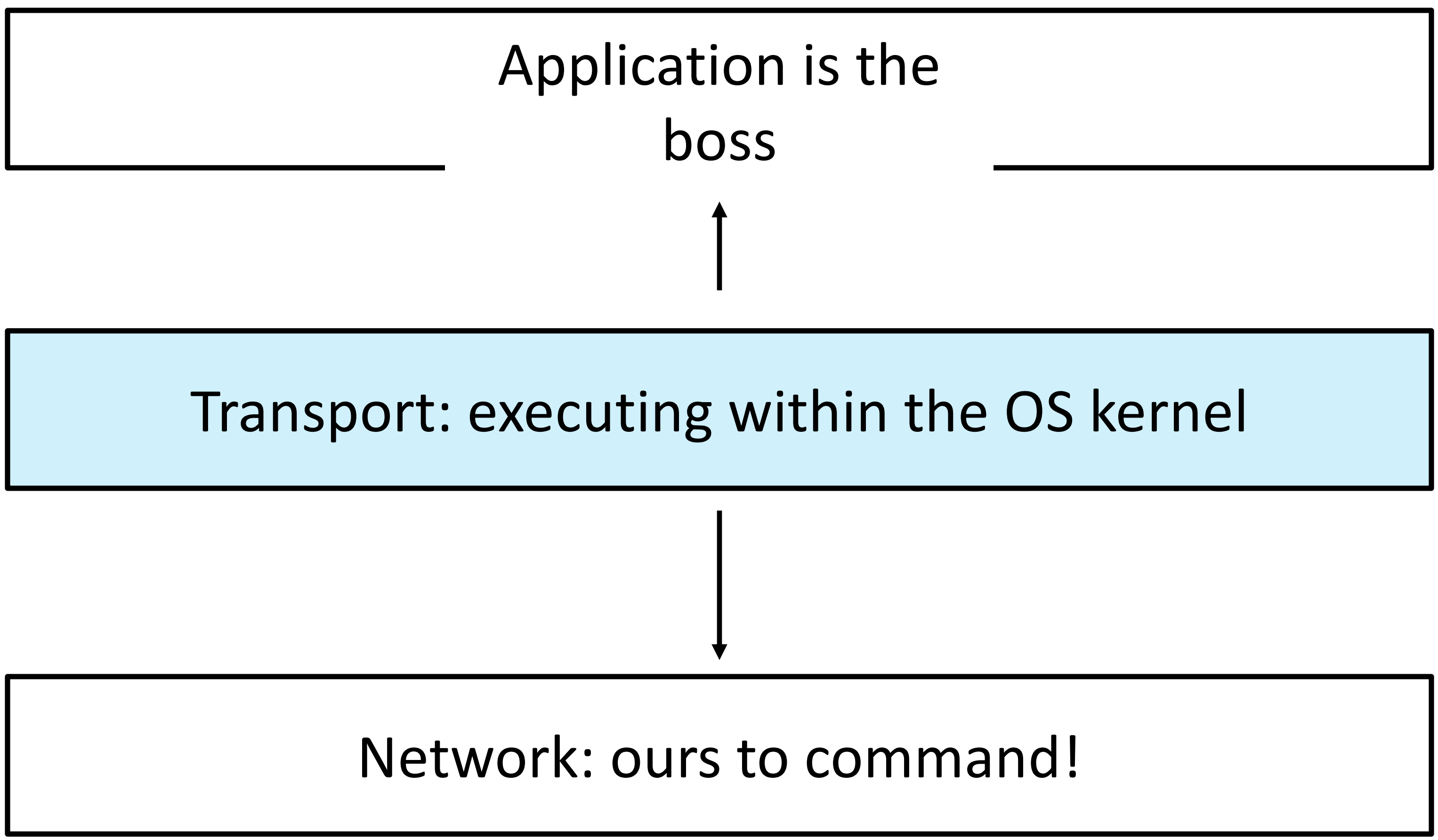


Message Encapsulation



- Higher layer within lower layer
- Each layer has different concerns, provides abstract services to those above

Transport Layer perspective



Transport Layer perspective

Looked at
Application Layer

Transport: executing within the OS kernel

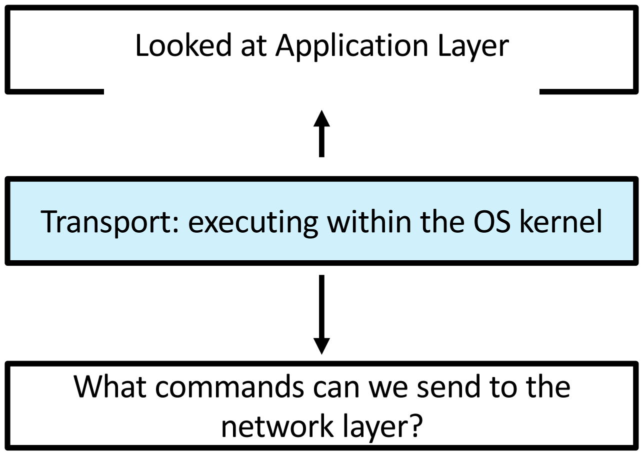
What commands can we send to the network layer?

What services does the network layer provide to the transport layer?

A. Find paths through the network

B. Reliable transfer, with guaranteed delivery rates

C. Best-effort delivery



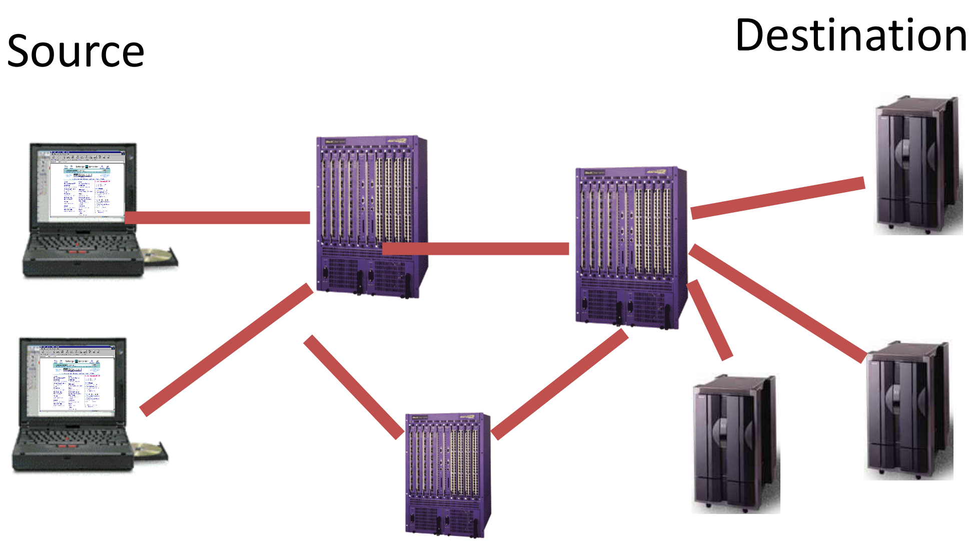
Network Layer API

send_to_host (data, host) : logical communication between end-hosts

- ✓ Find paths through the network
- ✓ Best-effort delivery!

reliable data transfer

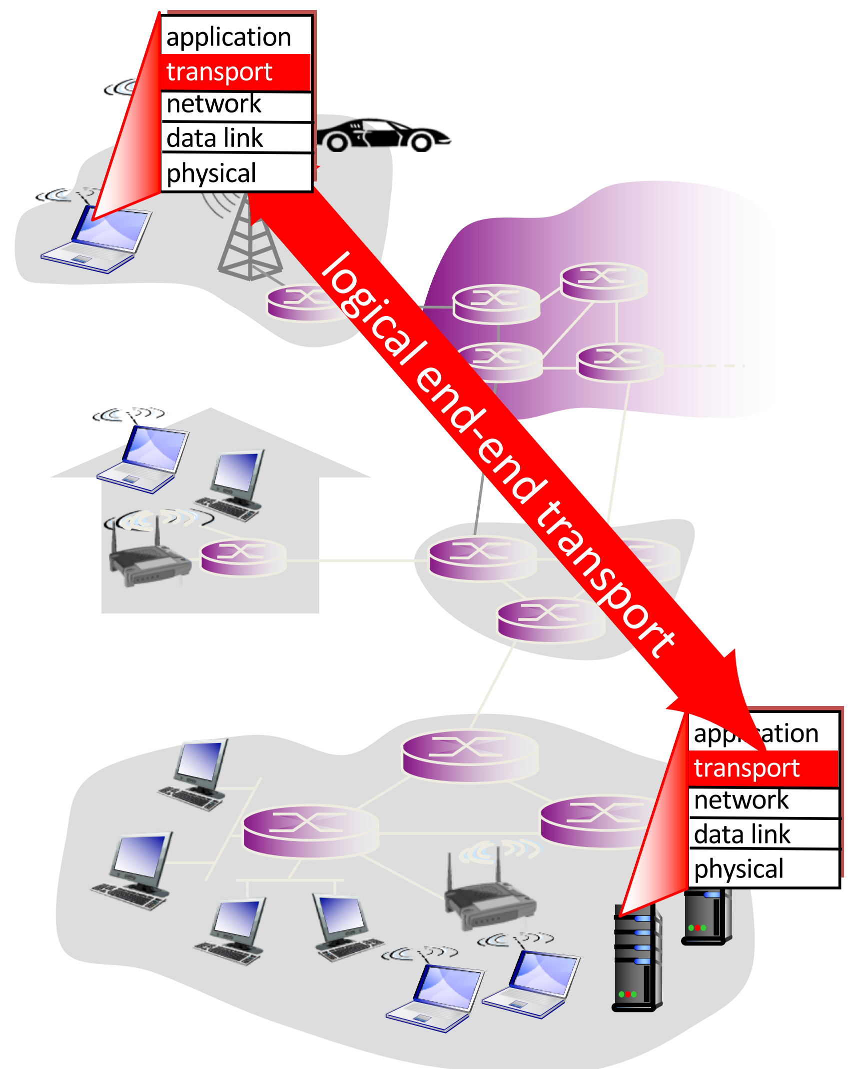
guaranteed delivery (or rate!)



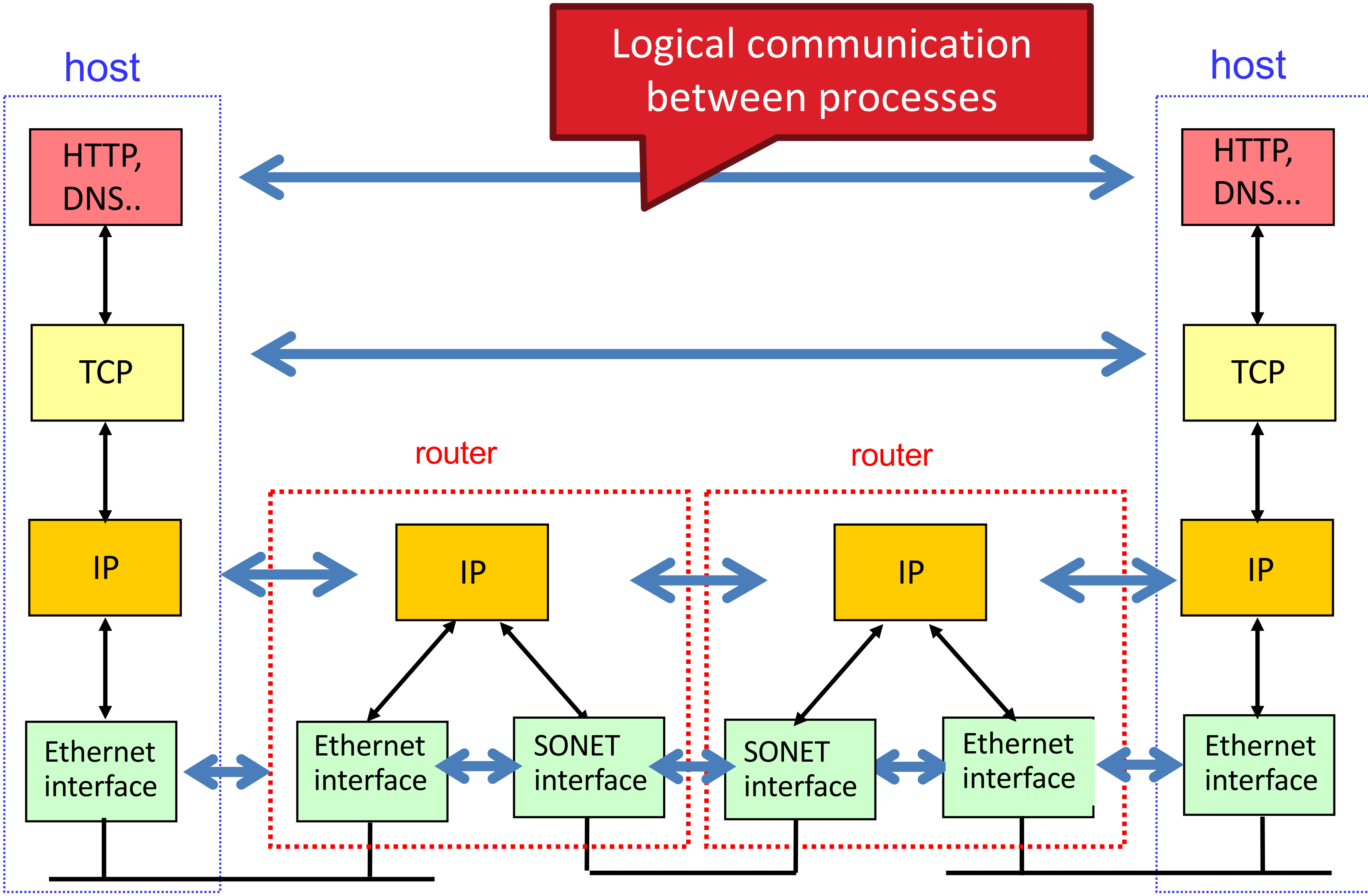
Transport Layer API

Provides logical communication between processes.

send_data_to_application (data, port, socket)



Transport Layer: Runs on end systems



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.

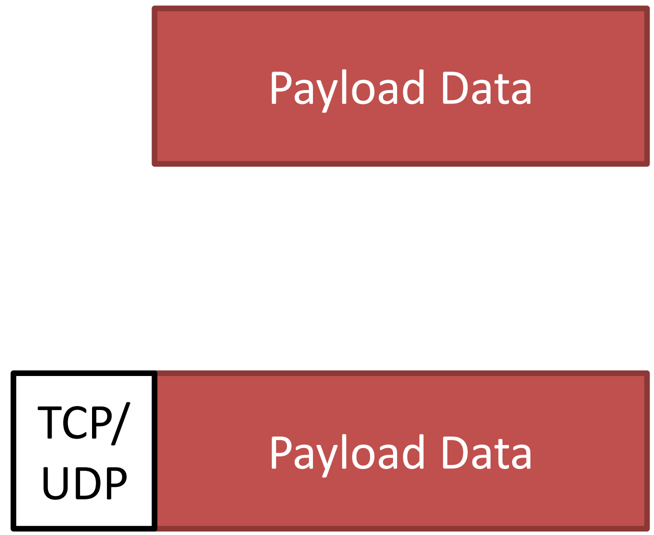
B. Sometimes all we need is error detection.

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

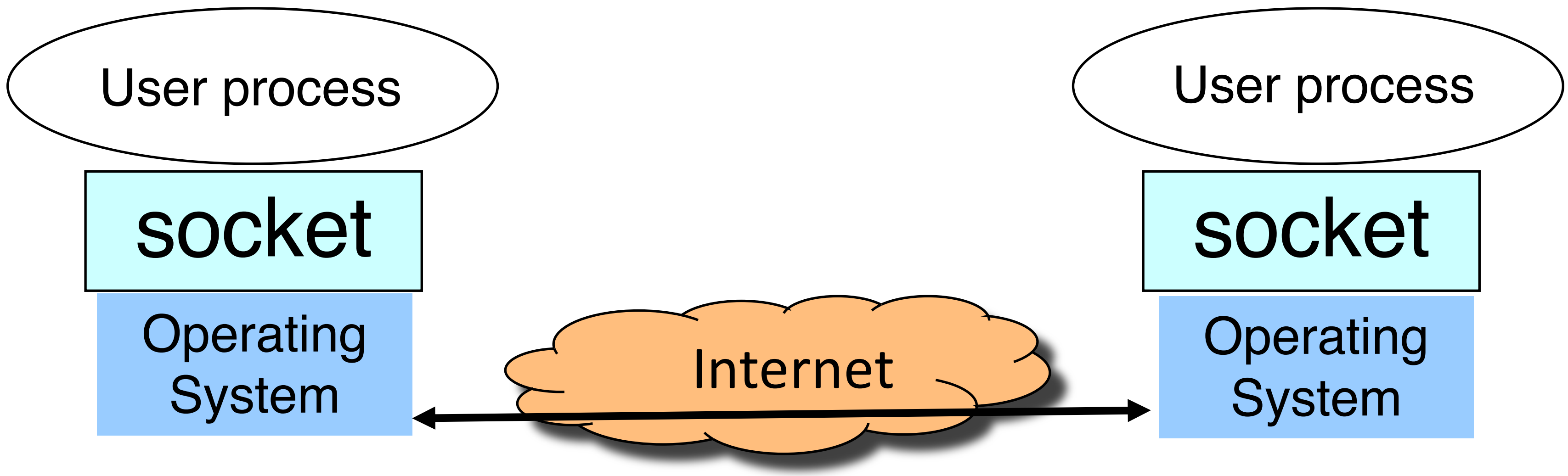
- Establishing state (making a connection)
  - Recall HTTP 1.0 vs. HTTP 1.1
  - Extra communication round trip

- Delays due to loss / reordering.

- Playing fair might cost you!

Socket Abstraction

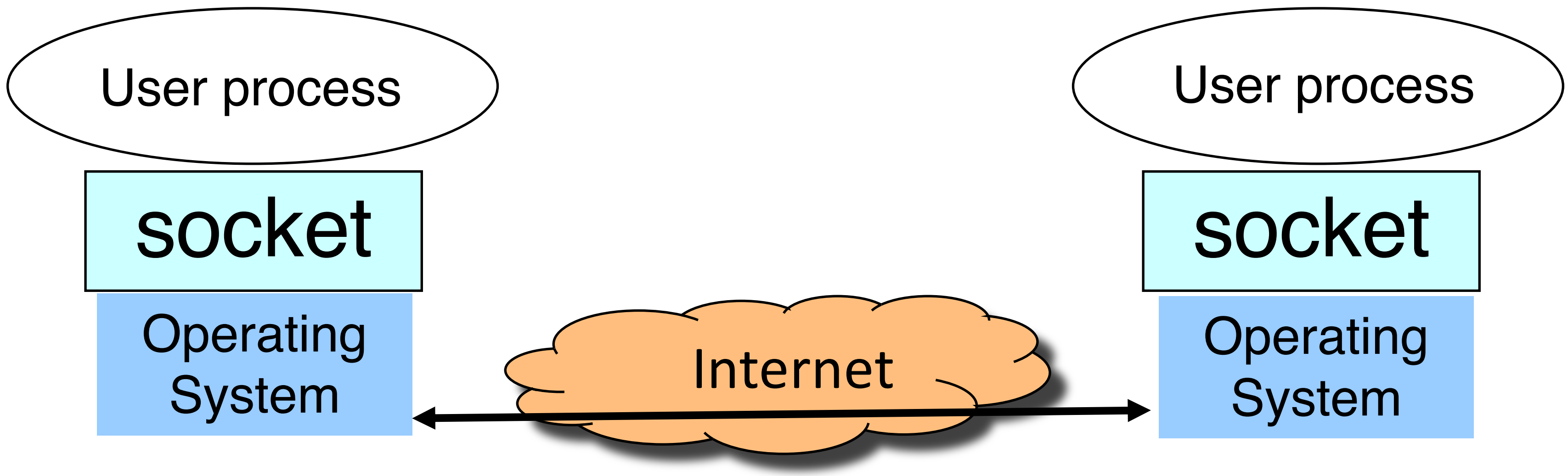
Socket Abstraction



Applications communicate using “sockets”

- Stream socket: reliable stream of bytes
- Message socket: unreliable message delivery

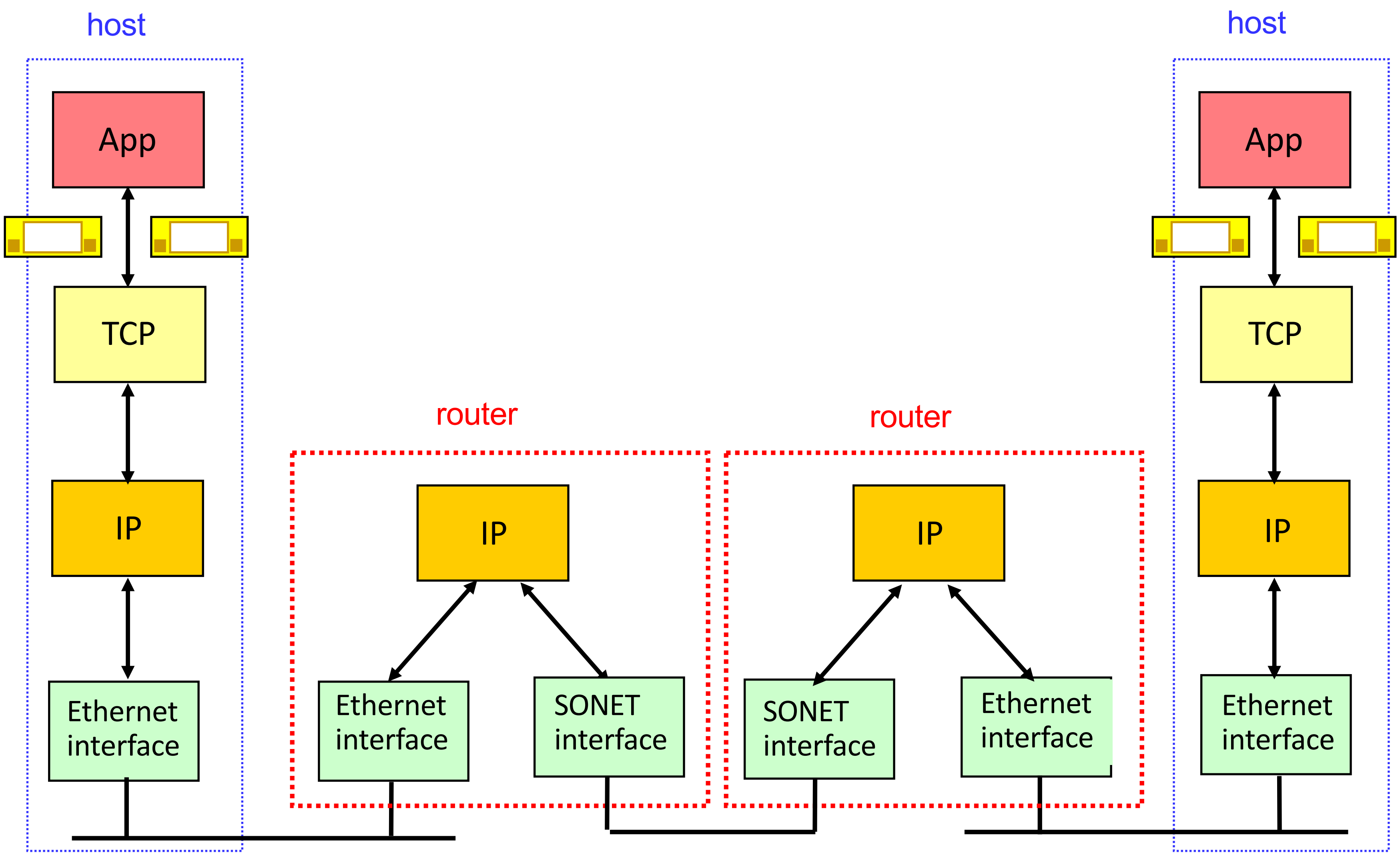
Socket Abstraction



Applications communicate using "sockets"/mailboxes

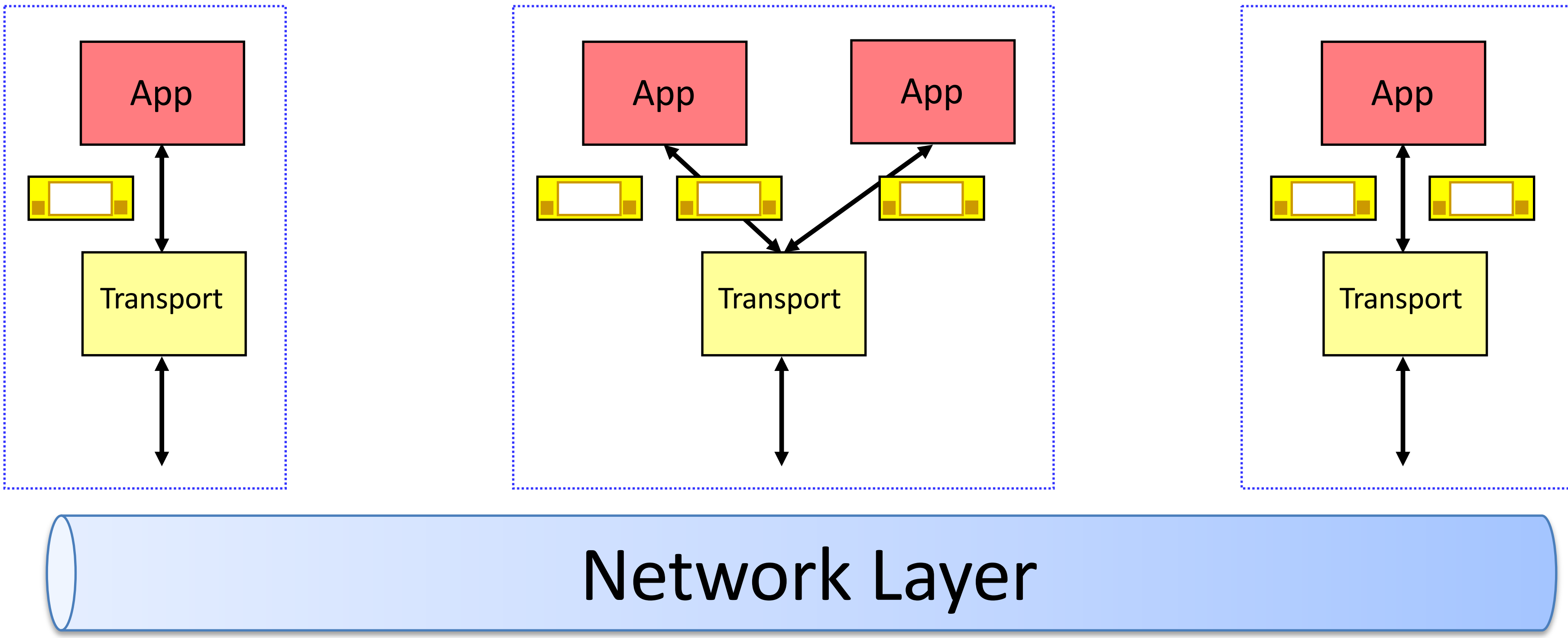
- Different mail-delivery service choices:
- TCP, UDP, ICMP, SCTP

Addressing Applications using Ports



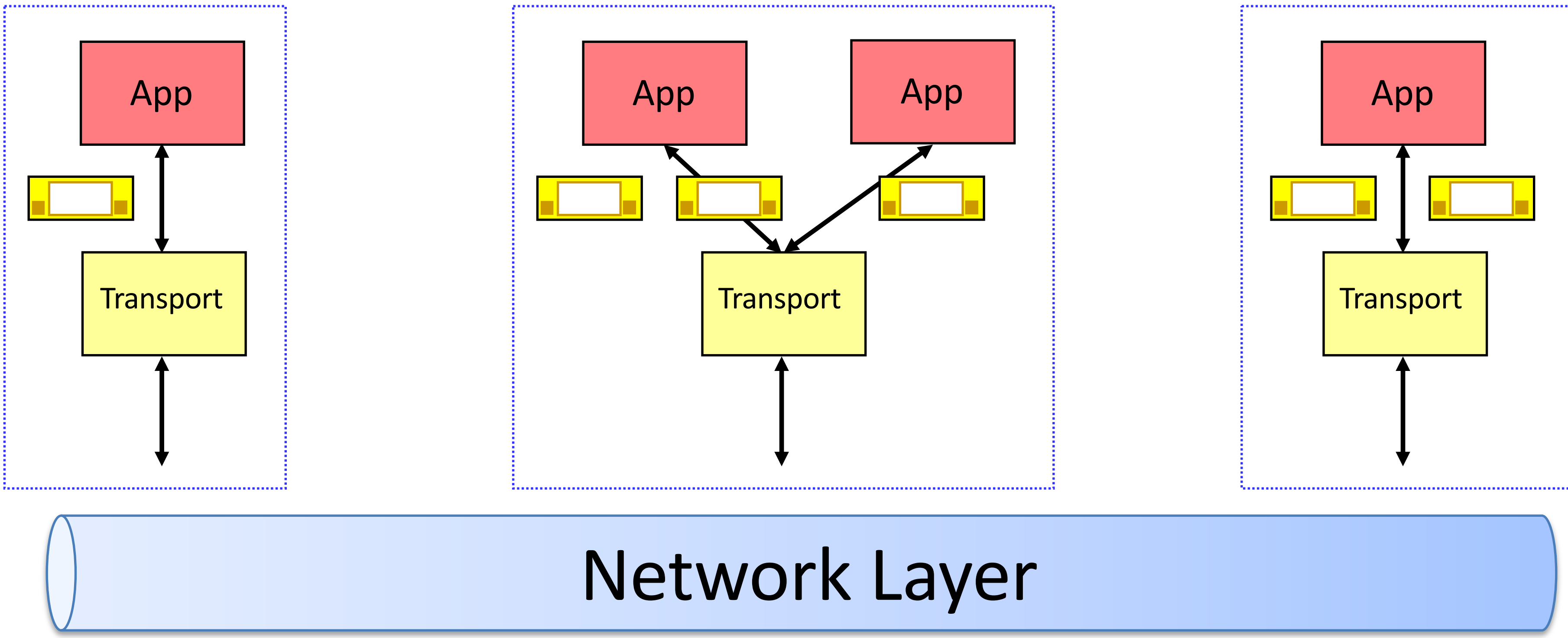
Multiplexing

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



Multiplexing

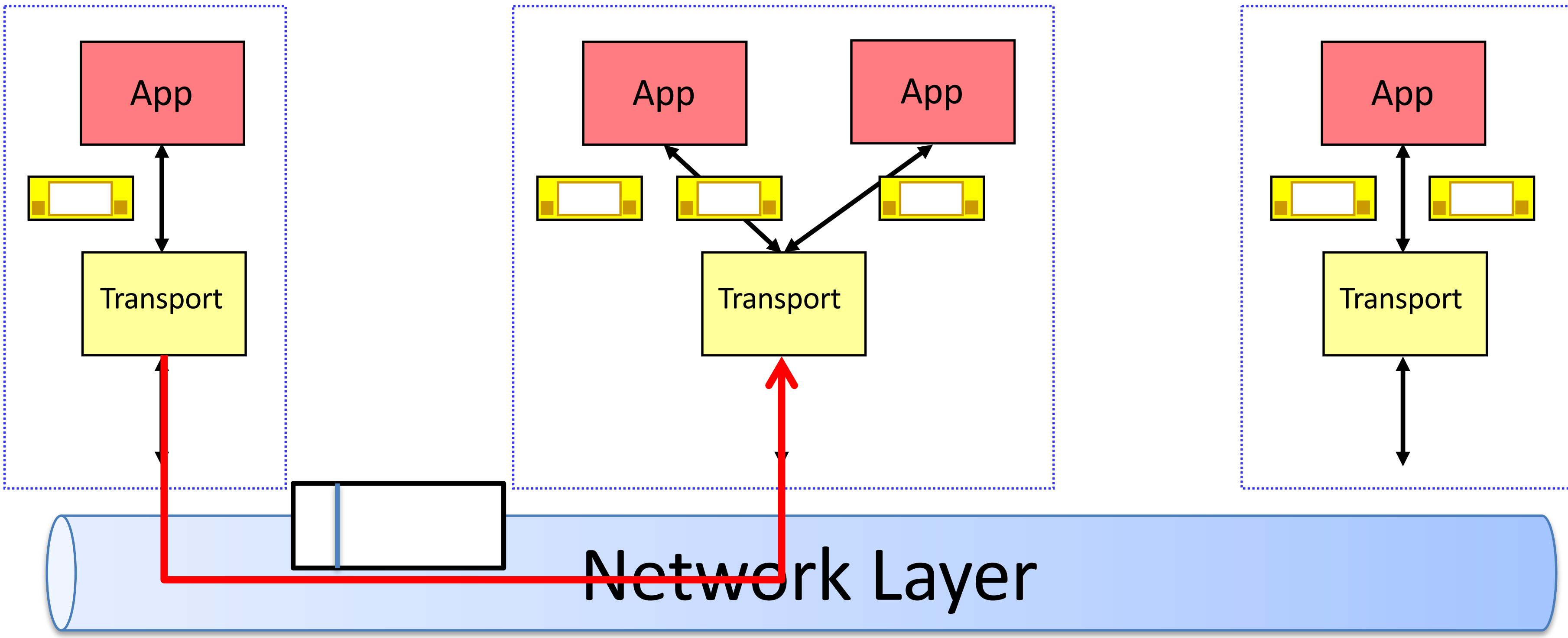
(Simultaneous transmission of two or more signals/messages over a single channel.)



- 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.)

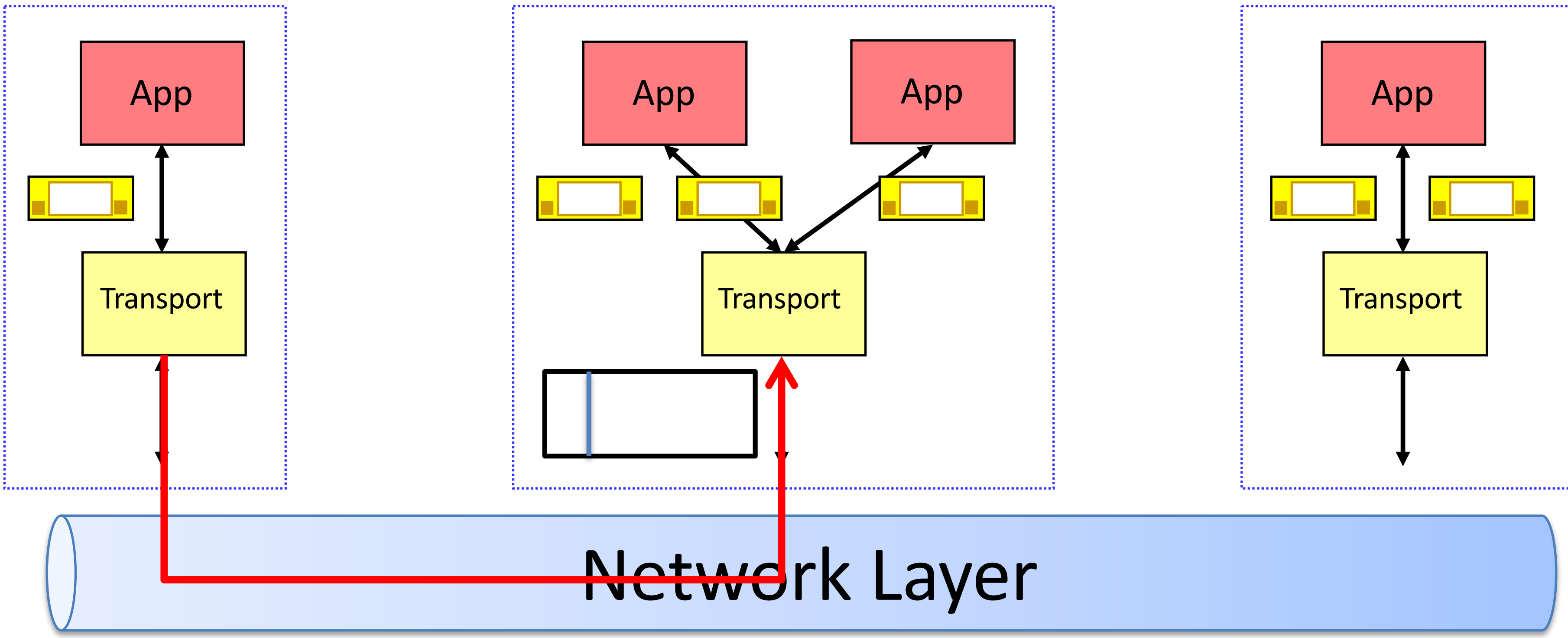


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

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

De-multiplexing

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

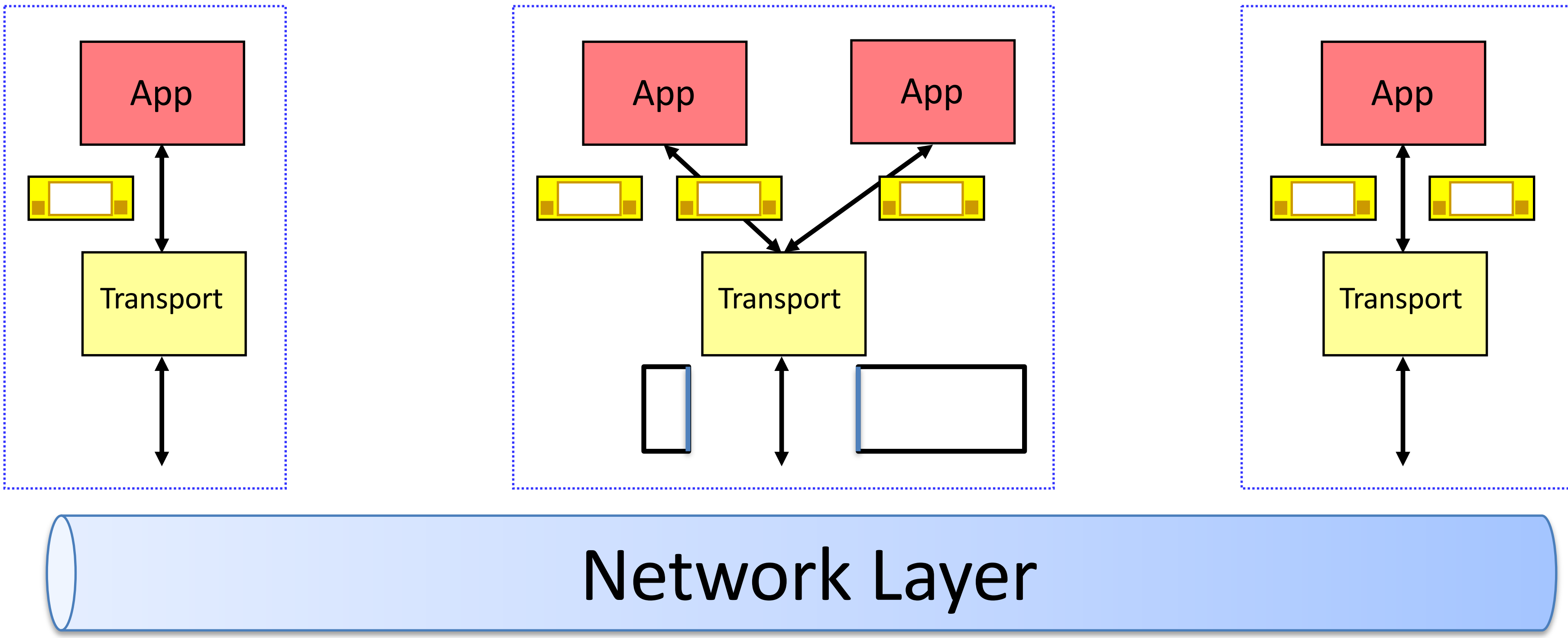


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

De-multiplexing

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

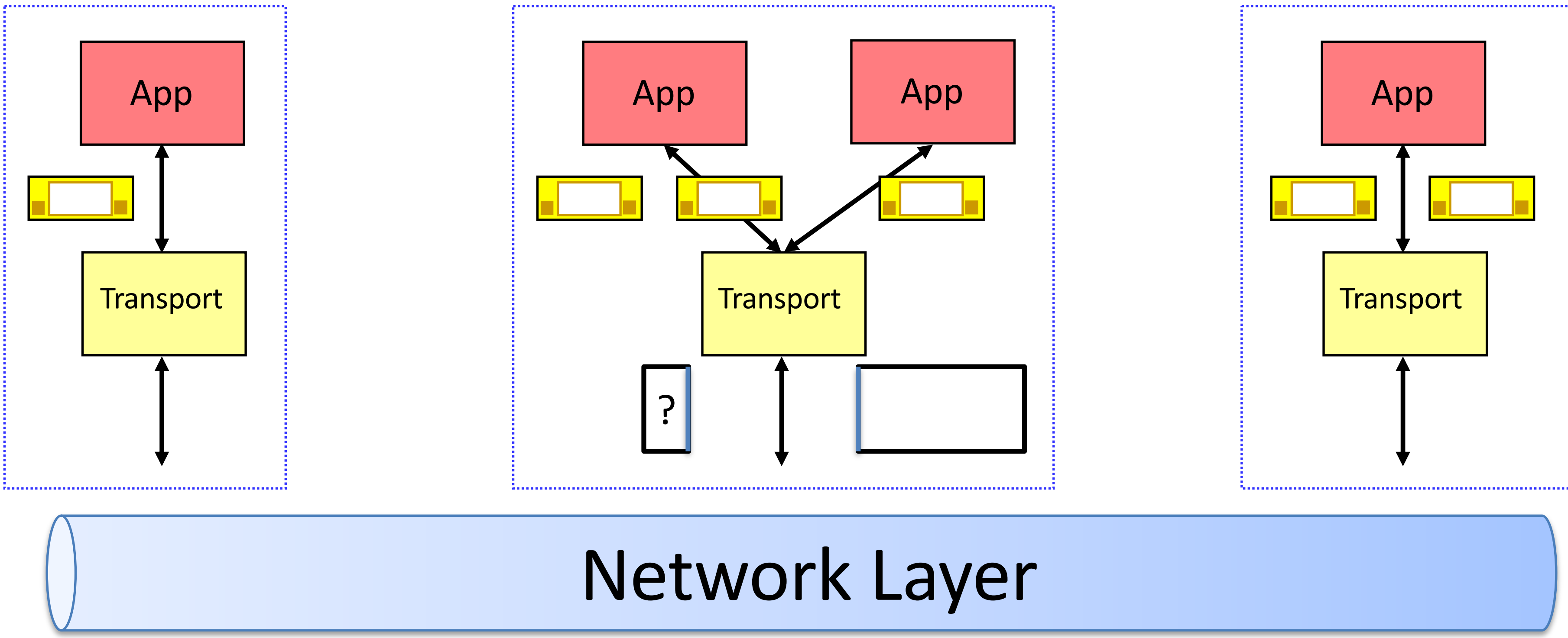


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- Receivers check header, deliver data to correct socket.

De-multiplexing

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

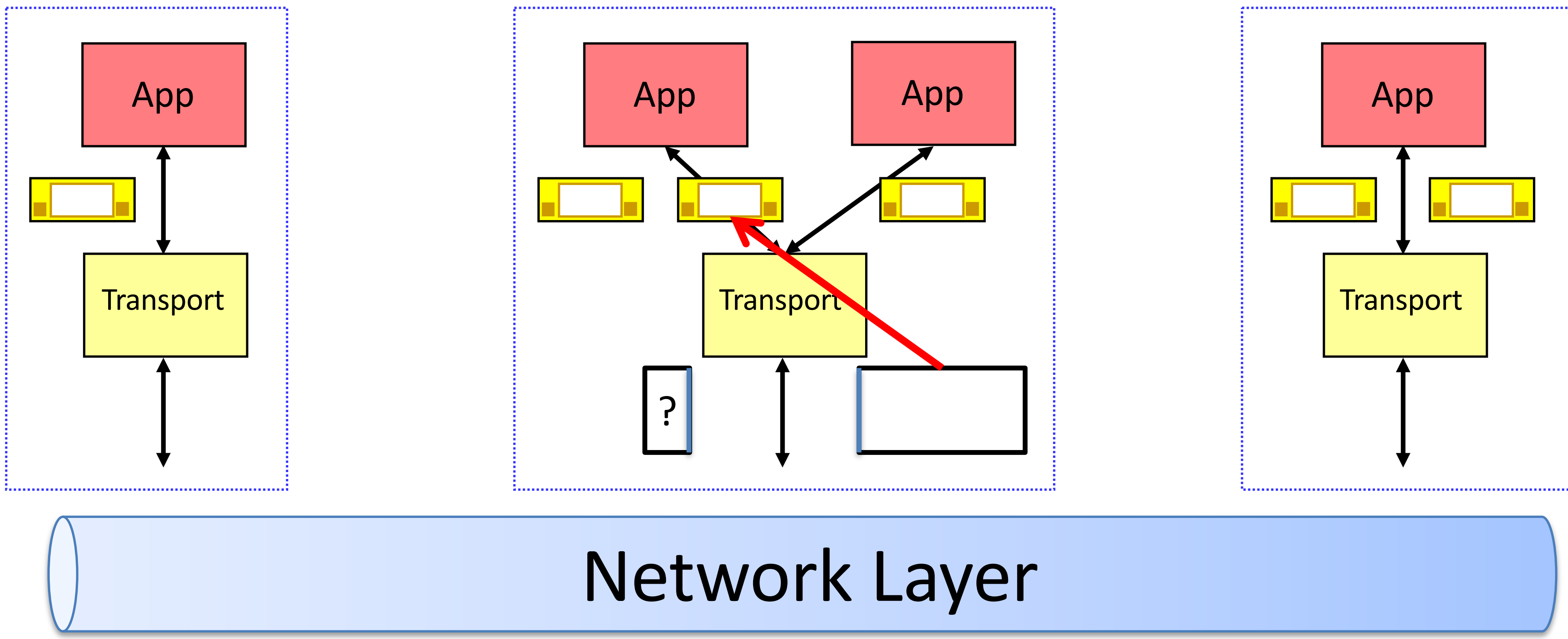


- The network is a shared resource.
  - It does NOT care about your applications, sockets, etc.

- 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.

Two Main Transport Layers

- User Datagram Protocol (UDP)
  - Unreliable, unordered delivery
- Transmission Control Protocol (TCP)
  - Reliable in-order delivery

UDP – User Datagram Protocol

- Unreliable, unordered service
- Adds:
  - multiplexing,
  - checksum (error detection)

UDP: User Datagram Protocol [RFC 768]

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

UDP: User Datagram Protocol [RFC 768]

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

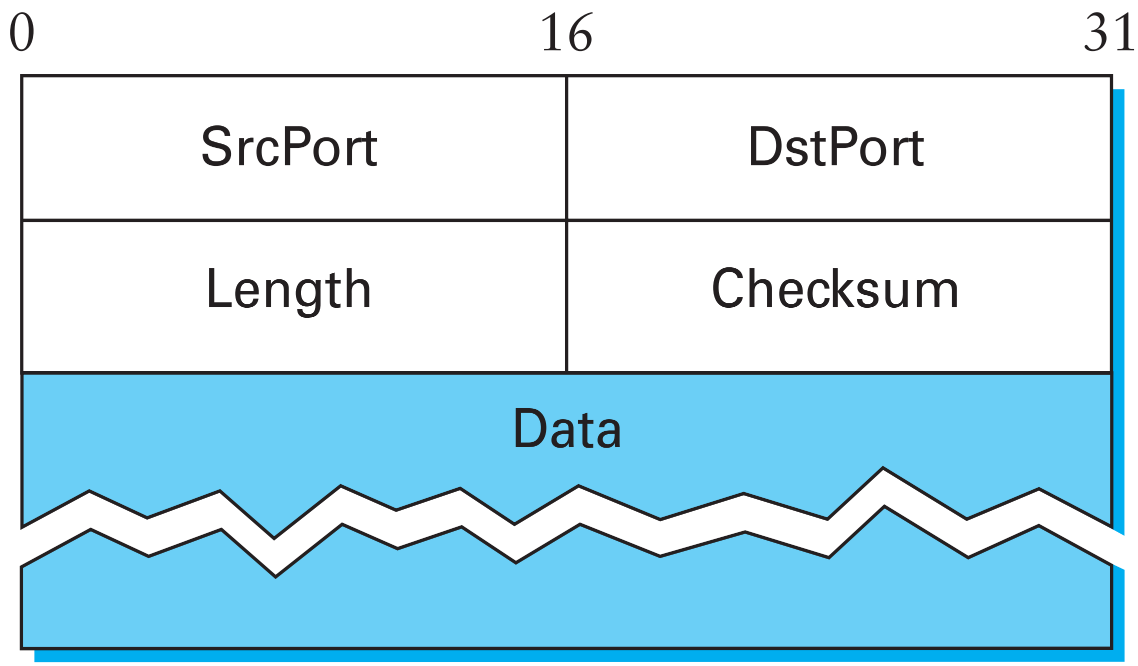
¯_(ツ)_/¯

How many of the following steps does UDP implement? (which ones?)

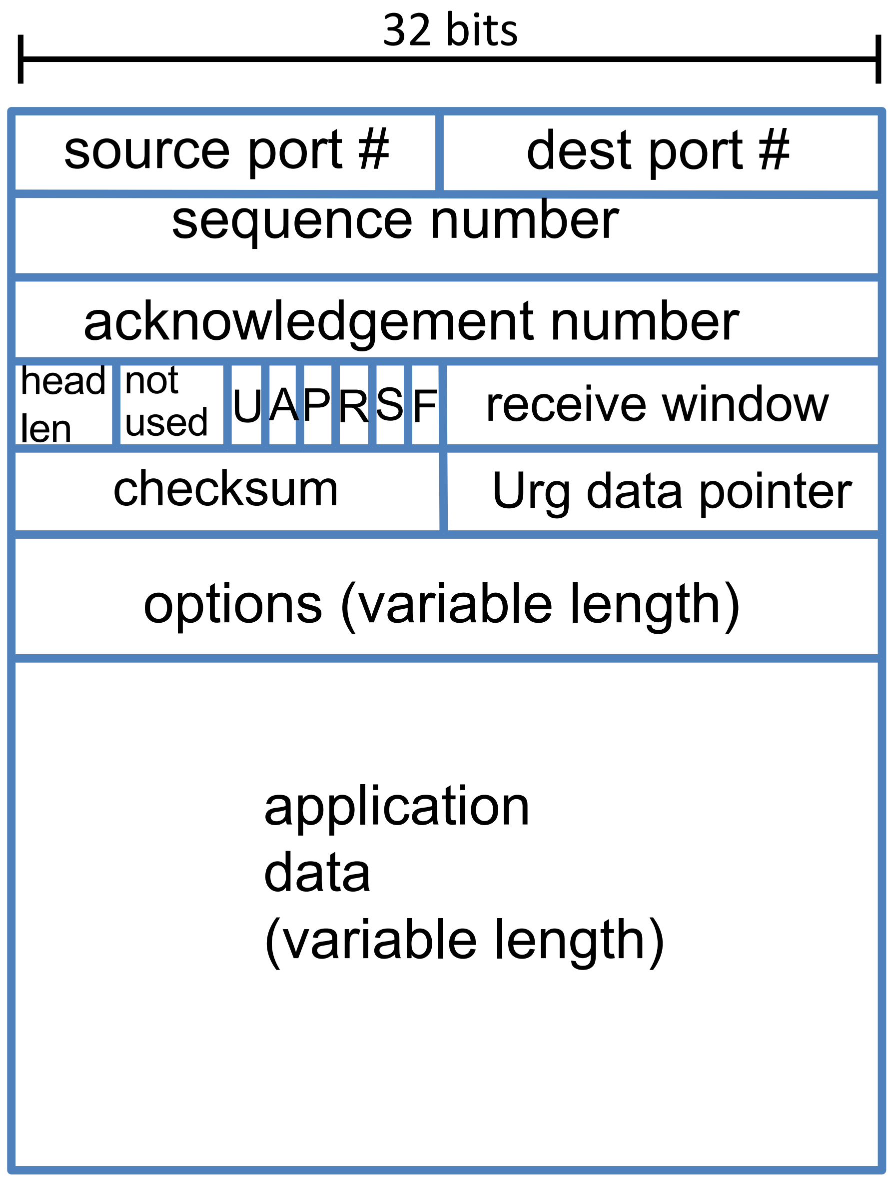
1. exchange an initiate handshake (connection setup)
2. break up packet into segments at the source and number them
3. place segments in order at the destination
4. error-checking with checksum

A. 1
B. 2
C. 3
D. 4

UDP Segment



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 Checksum

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

UDP Checksum

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 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

One's Compliment

- 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.

Recall: TCP send() blocking

With TCP, send() blocks if buffer full.

UDP sendto() blocking

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