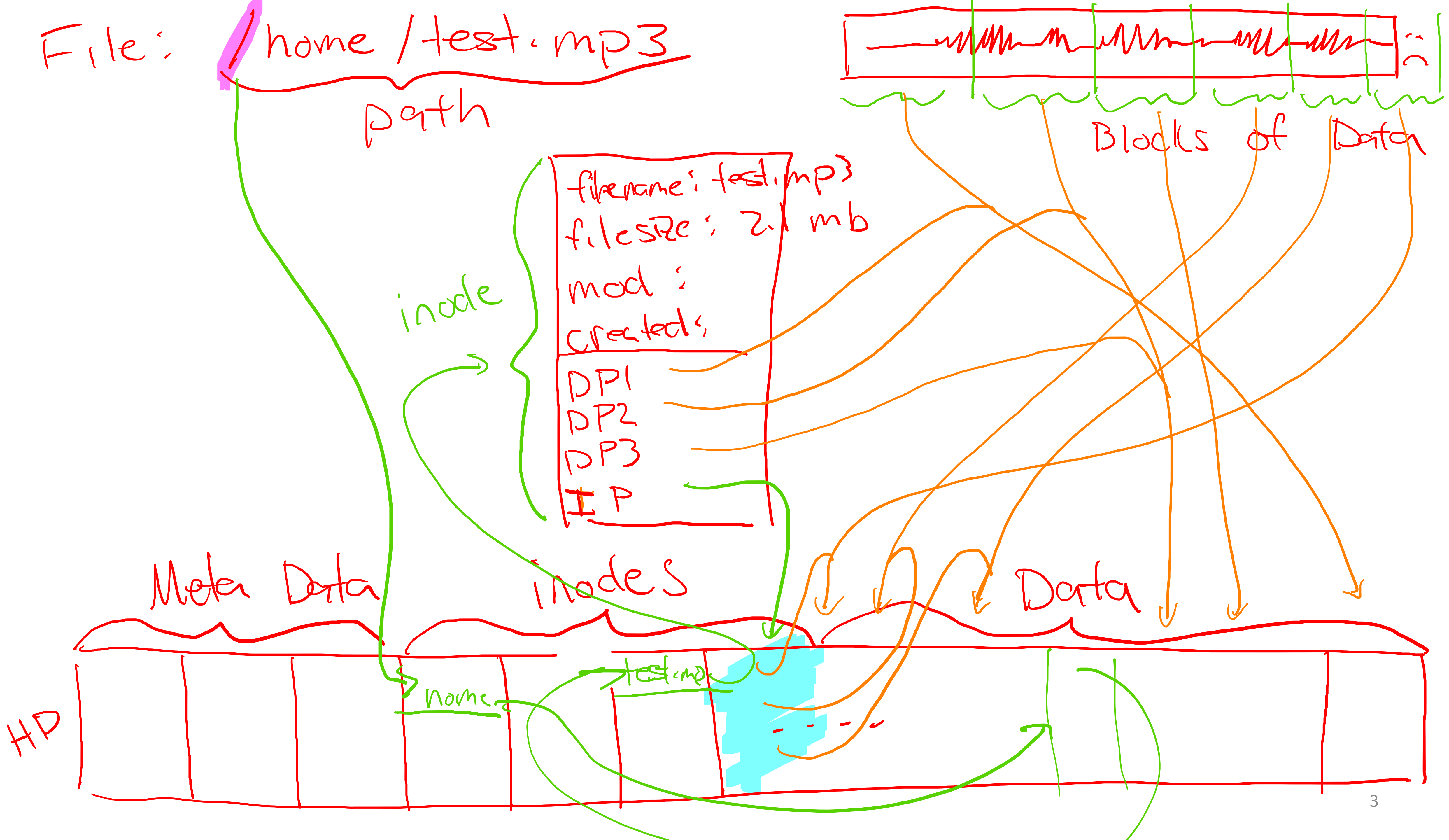
Networks

Drawing: File Systems

- Take three minutes to draw “file systems”

- Some reminders
    - Blocks
    - inodes
    - Direct and indirect pointers
    - Fast file system (FFS)


File: /home/test.mp3
path
Blocks of Data
inode
filename: test.mp3
filesize: 2.1 mb
mod:
created:
DP1
DP2
DP3
IP
Meta Data
inodes
Data
HD
home
test.mp3

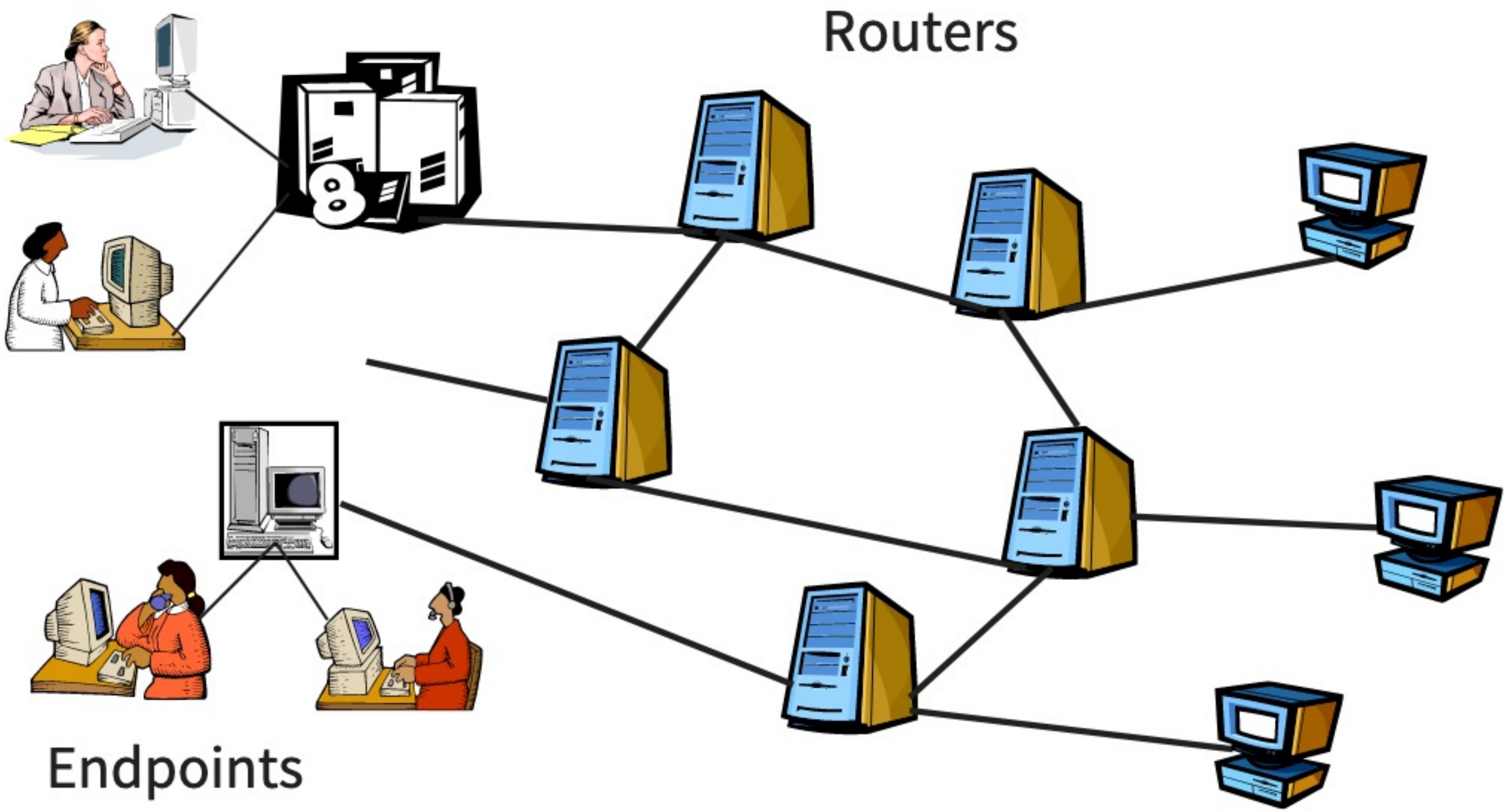
All About Protocols

- How would you send a message from your computer to the server?

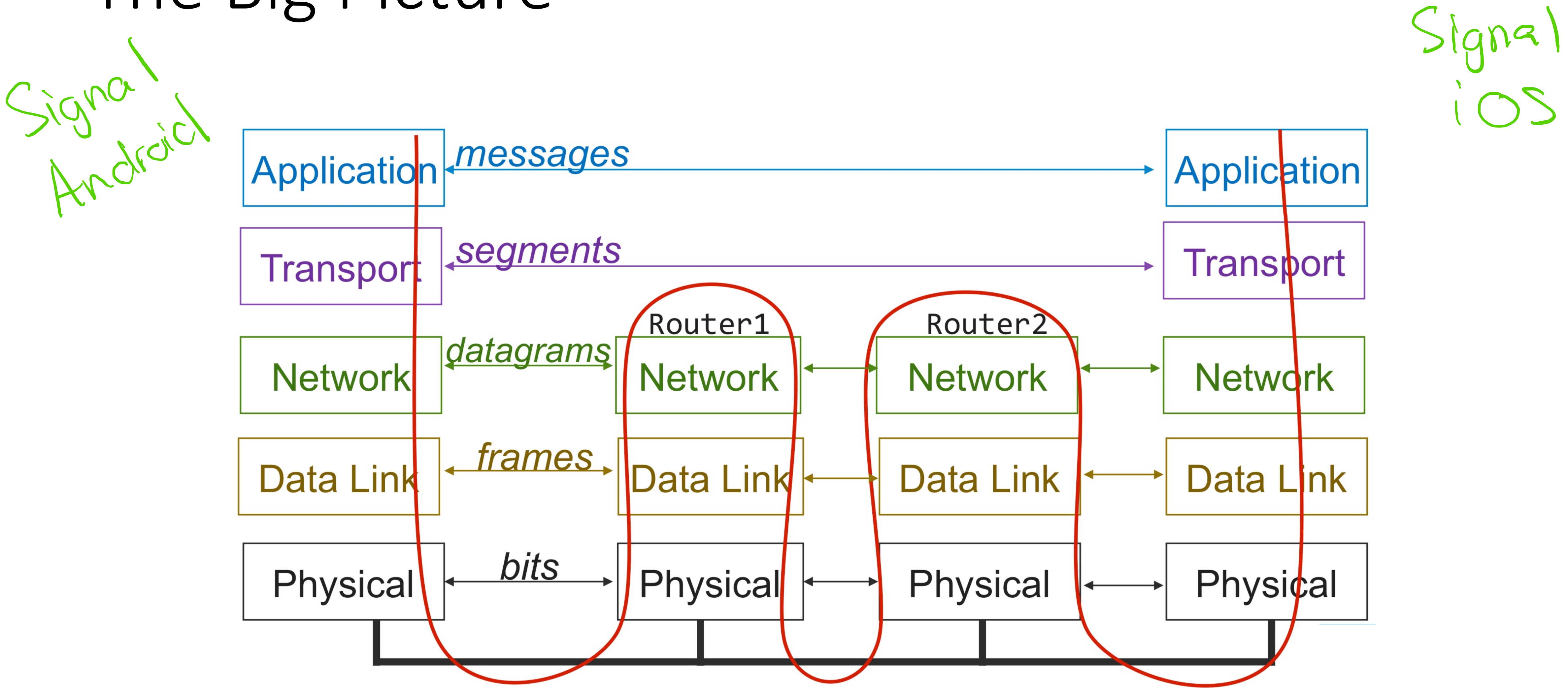
- What is involved in the process?
  - What hardware?
  - What software?

- How would you ensure privacy? Error handling?

The Big Picture



The Big Picture



The Big Picture

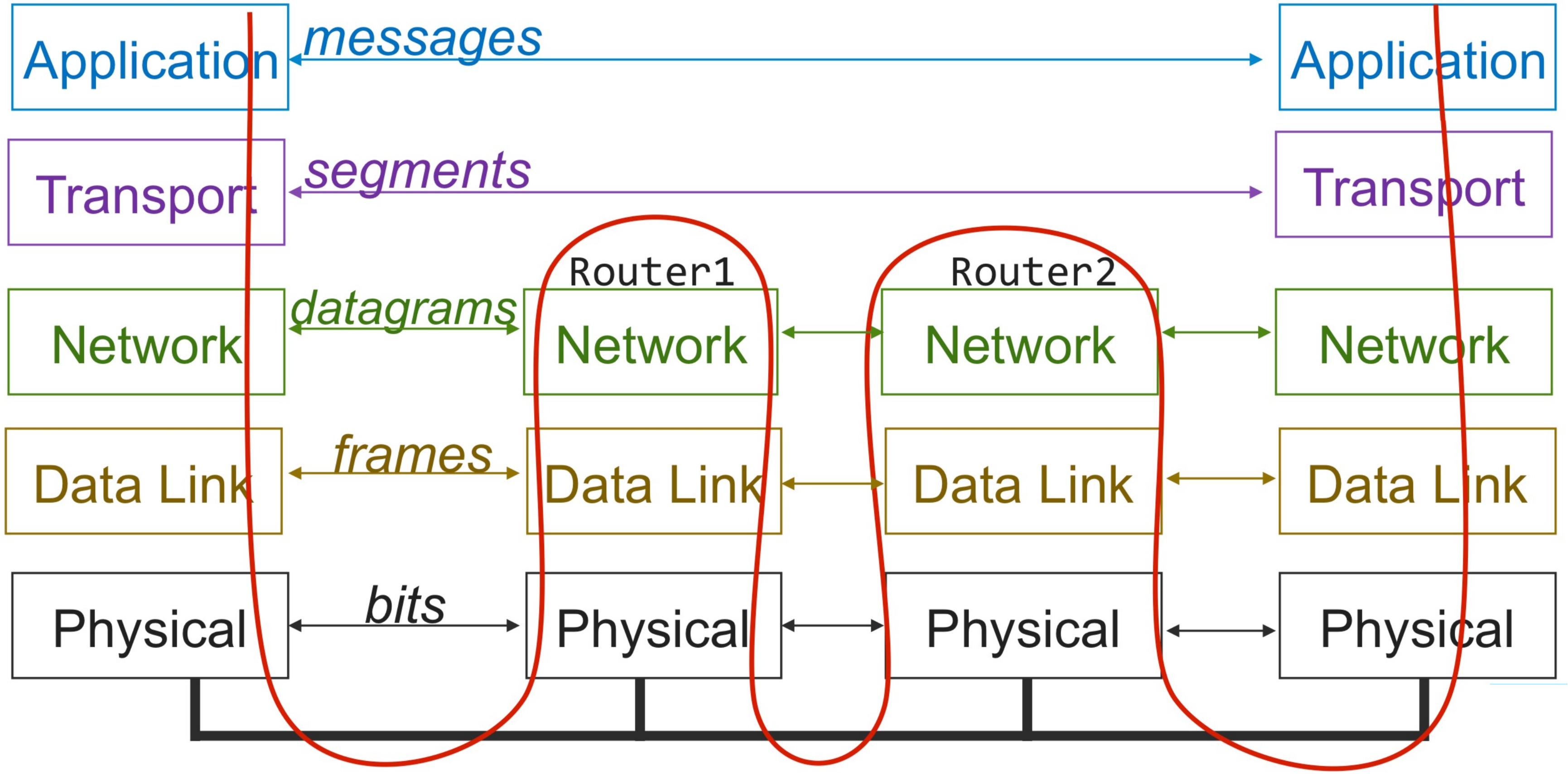
Firefox (e.g., HTTPS)

Operating System Library (e.g., TCP)

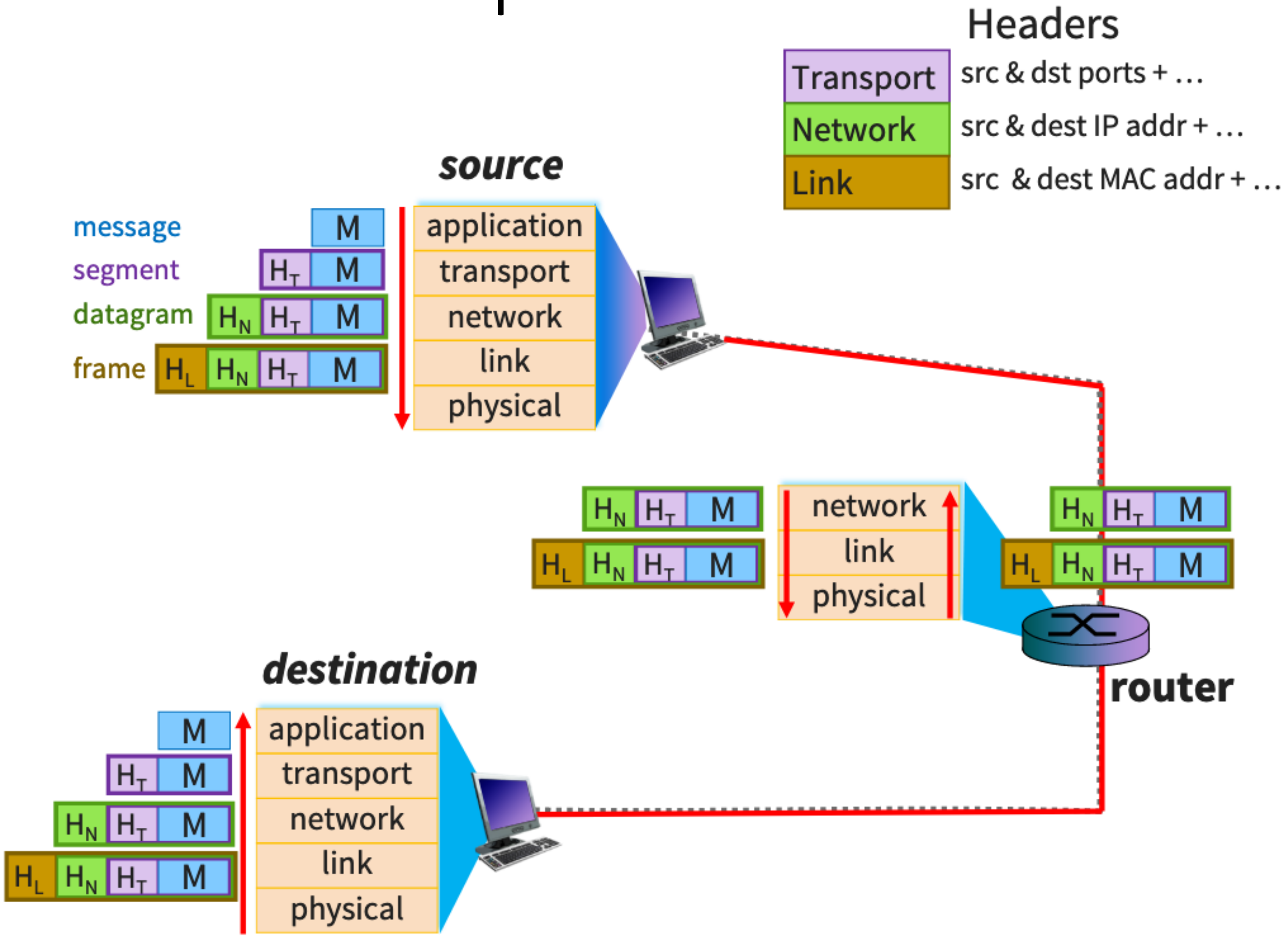
OS and Software Drivers (e.g., IP)

Hardware Interface (e.g., Wi-Fi)

Physical Transmission (e.g., Fiber)



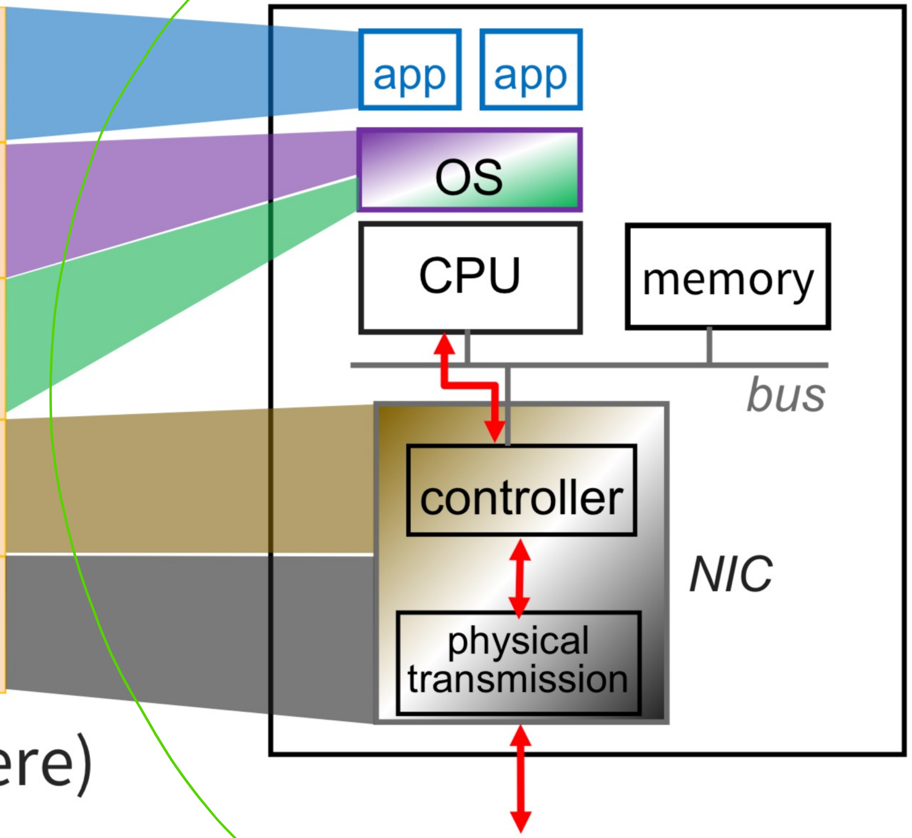
Protocols and Encapsulation

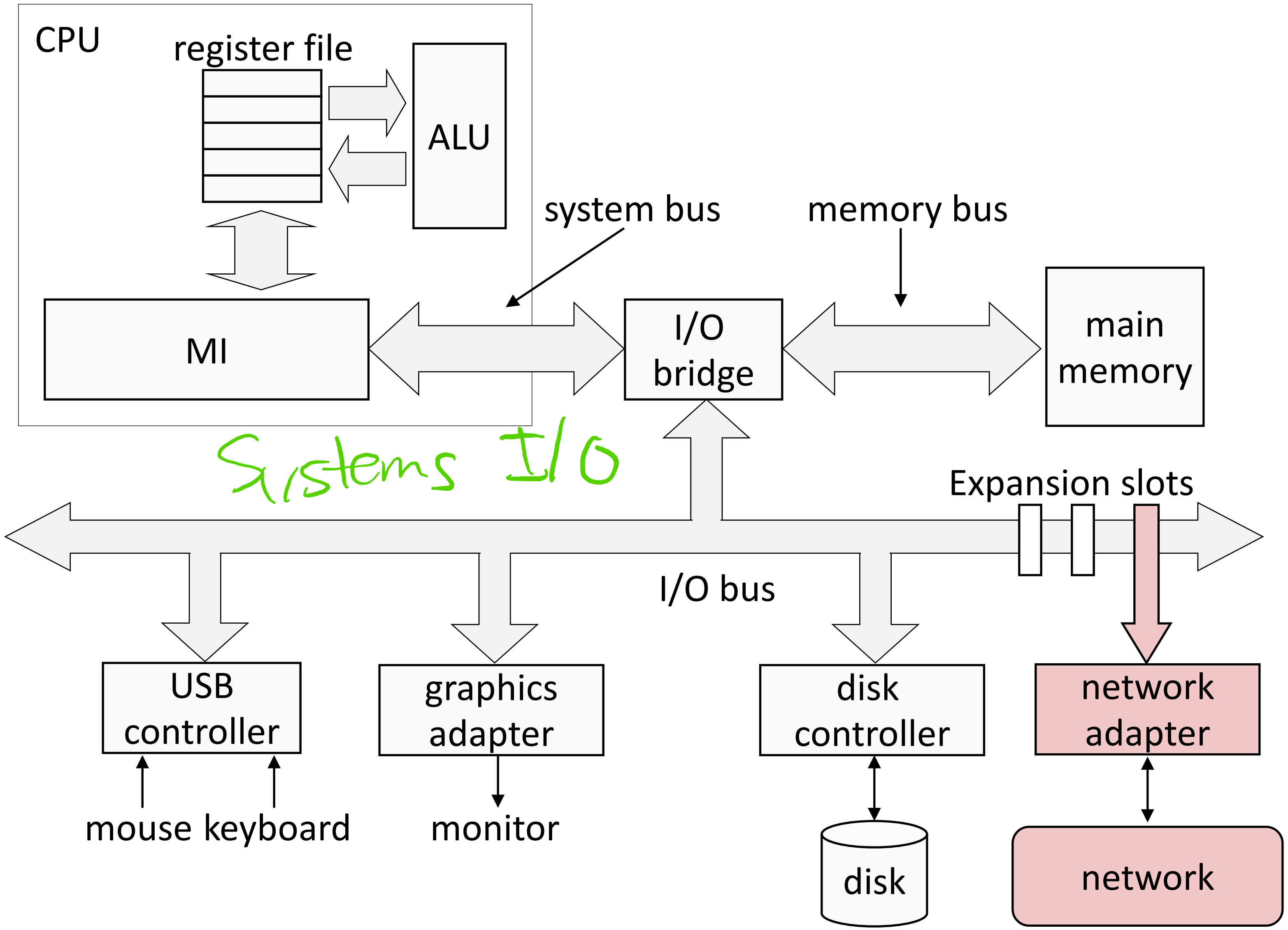


Hardware and Software Interfaces

Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding

(Hard to draw firm lines here)




CPU
register file
ALU
MI
system bus
memory bus
I/O bridge
main memory
Systems I/O
Expansion slots
I/O bus
USB controller
graphics adapter
disk controller
network adapter
mouse keyboard
monitor
disk
network

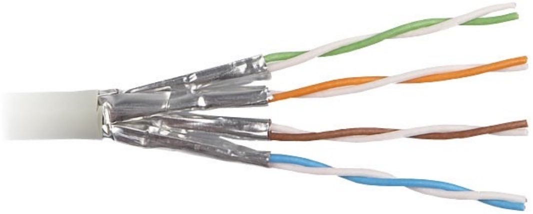
First Up → Physical

Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding

(Hard to draw firm lines here)

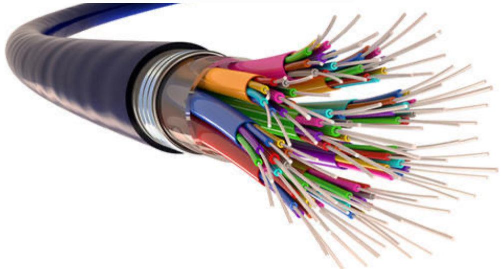
Physical Layer

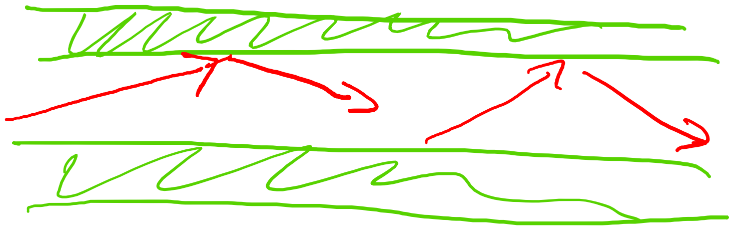
- Twisted Pair



- Coaxial

- Fiber





- Radio

Next Up → Link

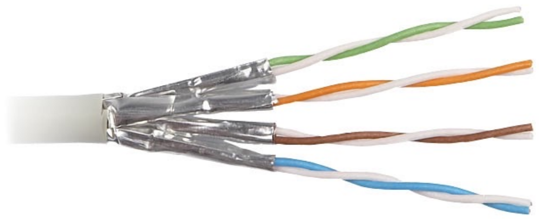
Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding

(Hard to draw firm lines here)

Data Link Layer

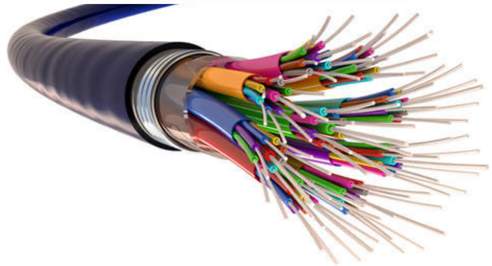
- DSL
- Ethernet
- WiFi (802.11)
- 3G
- LTE
- 5G

Twisted Pair



Coaxial

Fiber



Radio

Data Link Layer

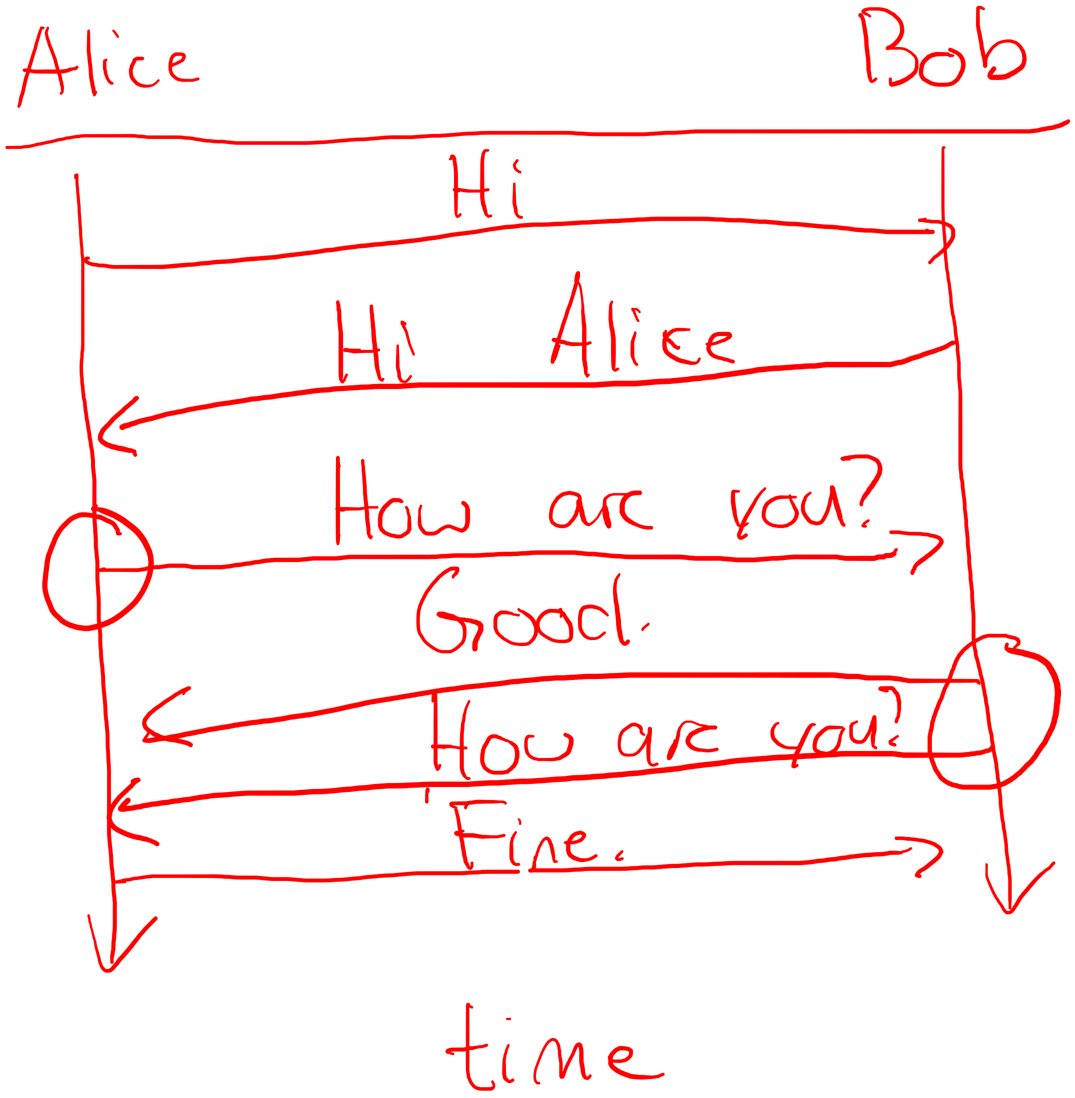
Client + Server

- Each host has one or more network adapter
  - Network interface controller (or card), aka NIC
  - Handles physical layer and protocol

- Each network adapter has a media access control (MAC) address
  - Unique to that network instance

- Messages are organized as packets

Communication Protocol

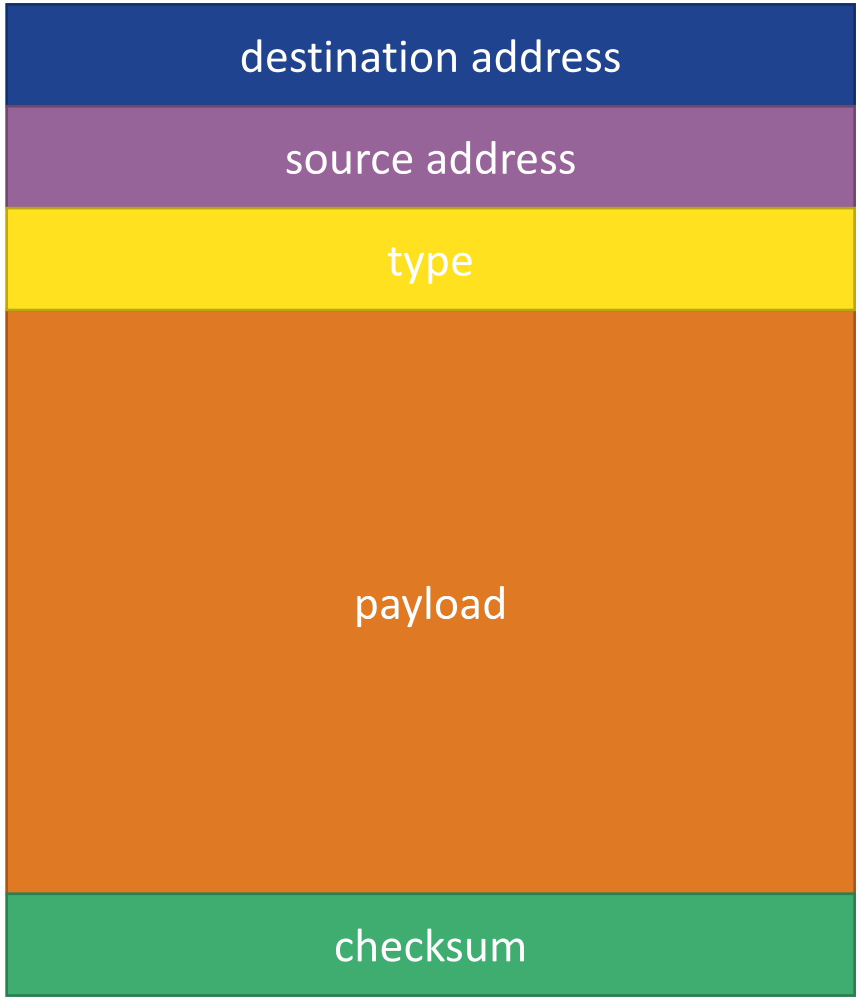


<u>Anomalies</u>

- Protocol Deviation
- Data Corruption
- Lost Connection
- Network Latency
- Out-of-Order Delivery

Ethernet

- Developed 1976 at Xerox
- Simple, scales well
- Very successful, still in widespread use
- Example address: `b8:e3:56:15:6a:72`
- **Carrier sense:** listen before you speak
- **Multiple access:** multiple hosts on network
- **Collision detection:** detect and respond to cases where two messages collide



Example: Ethernet

- Carrier sense: broadcast if wire is available
- In case of collision: stop, sleep, retry
  - sleep time is determined by collision number
  - abort after 16 attempts
- Problems handled up the chain

Example: Ethernet

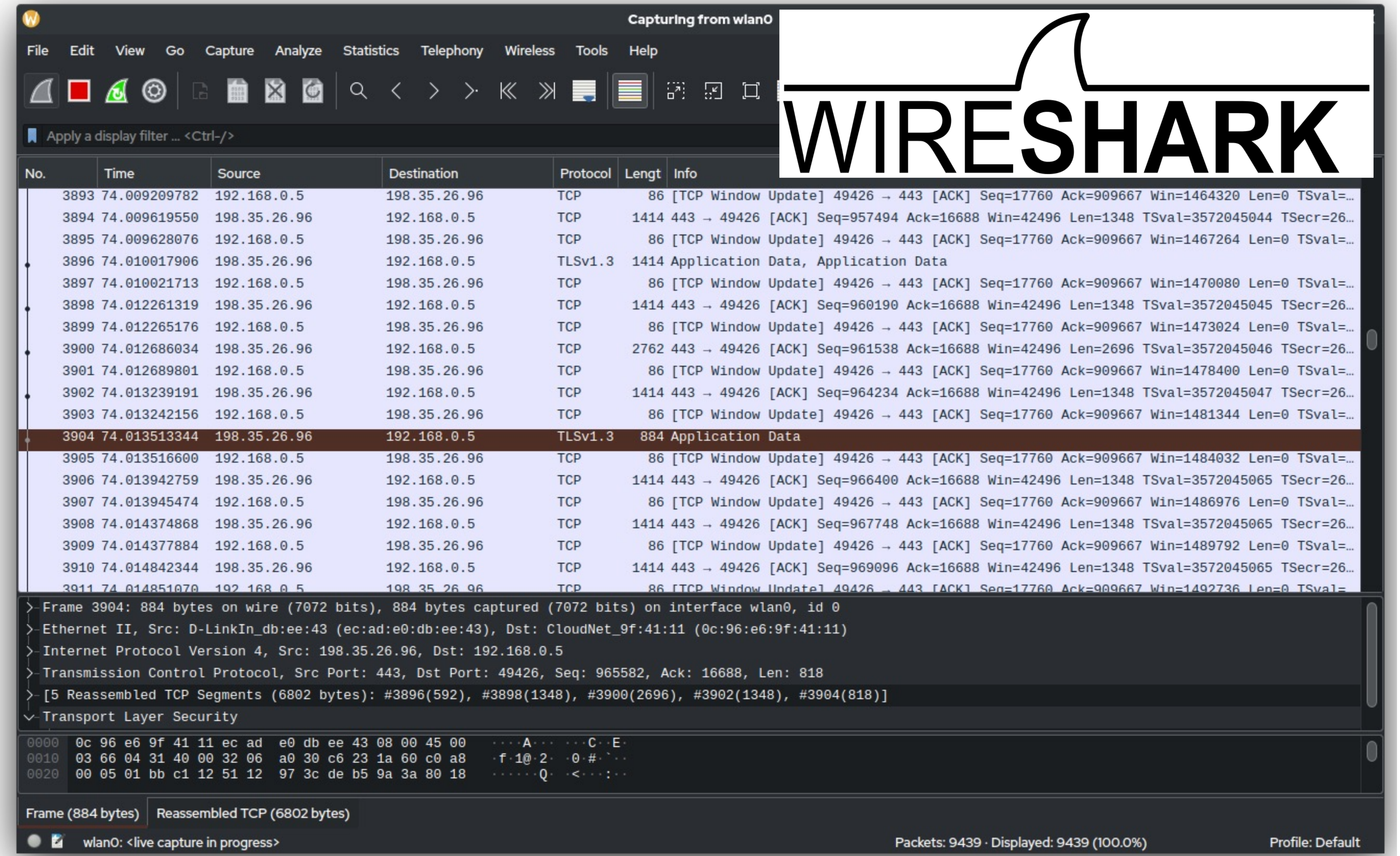
Advantages

- Completely decentralized
- Inexpensive
  - No state in the network
  - No arbiter
  - Cheap physical links

Disadvantages

- Endpoints must be trusted
- Data is available for all to see
  - Can place ethernet card in promiscuous mode and listen to all messages

WIRESHARK



Capturing from wlan0
WIRESHARK
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Lengt Info
3893 74.009209782 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1464320 Len=0 TSval=…
3894 74.009619550 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=957494 Ack=16688 Win=42496 Len=1348 TSval=3572045044 TSecr=26…
3895 74.009628076 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1467264 Len=0 TSval=…
3896 74.010017906 198.35.26.96 192.168.0.5 TLSv1.3 1414 Application Data, Application Data
3897 74.010021713 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1470080 Len=0 TSval=…
3898 74.012261319 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=960190 Ack=16688 Win=42496 Len=1348 TSval=3572045045 TSecr=26…
3899 74.012265176 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1473024 Len=0 TSval=…
3900 74.012686034 198.35.26.96 192.168.0.5 TCP 2762 443 → 49426 [ACK] Seq=961538 Ack=16688 Win=42496 Len=2696 TSval=3572045046 TSecr=26…
3901 74.012689801 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1478400 Len=0 TSval=…
3902 74.013239191 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=964234 Ack=16688 Win=42496 Len=1348 TSval=3572045047 TSecr=26…
3903 74.013242156 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1481344 Len=0 TSval=…
3904 74.013513344 198.35.26.96 192.168.0.5 TLSv1.3 884 Application Data
3905 74.013516600 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1484032 Len=0 TSval=…
3906 74.013942759 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=966400 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
3907 74.013945474 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1486976 Len=0 TSval=…
3908 74.014374868 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=967748 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
3909 74.014377884 192.168.0.5 198.35.26.96 TCP 86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1489792 Len=0 TSval=…
3910 74.014842344 198.35.26.96 192.168.0.5 TCP 1414 443 → 49426 [ACK] Seq=969096 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
Frame 3904: 884 bytes on wire (7072 bits), 884 bytes captured (7072 bits) on interface wlan0, id 0
Ethernet II, Src: D-LinkIn_db:ee:43 (ec:ad:e0:db:ee:43), Dst: CloudNet_9f:41:11 (0c:96:e6:9f:41:11)
Internet Protocol Version 4, Src: 198.35.26.96, Dst: 192.168.0.5
Transmission Control Protocol, Src Port: 443, Dst Port: 49426, Seq: 965582, Ack: 16688, Len: 818
[5 Reassembled TCP Segments (6802 bytes): #3896(592), #3898(1348), #3900(2696), #3902(1348), #3904(818)]
Transport Layer Security
0000 0c 96 e6 9f 41 11 ec ad e0 db ee 43 08 00 45 00
0010 03 66 04 31 40 00 32 06 a0 30 c6 23 1a 60 c0 a8
0020 00 05 01 bb c1 12 51 12 97 3c de b5 9a 3a 80 18
Frame (884 bytes) Reassembled TCP (6802 bytes)
wlan0: <live capture in progress>
Packets: 9439 · Displayed: 9439 (100.0%)
Profile: Default

Practice with Data Link Layer

- Which of the following are examples of data link layer protocols?

  a. 4G LTE

  b. Ethernet

  c. Fiber

  d. WiFi (802.11)

  e. IP

Practice with Data Link Layer

- Which of the following are examples of data link layer protocols?

  a. 4G LTE

  b. Ethernet

  c. Fiber

  d. WiFi (802.11)

  e. IP

Next Up → Network

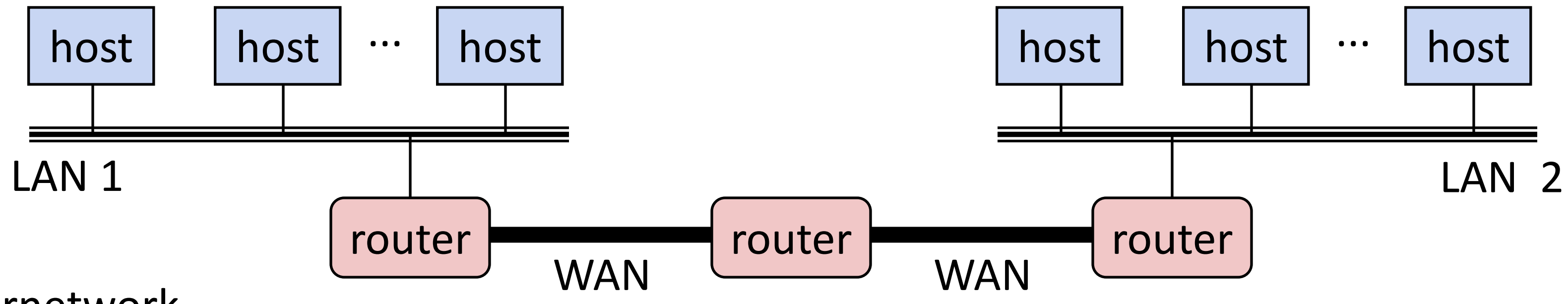
Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding

(Hard to draw firm lines here)

Network Layer

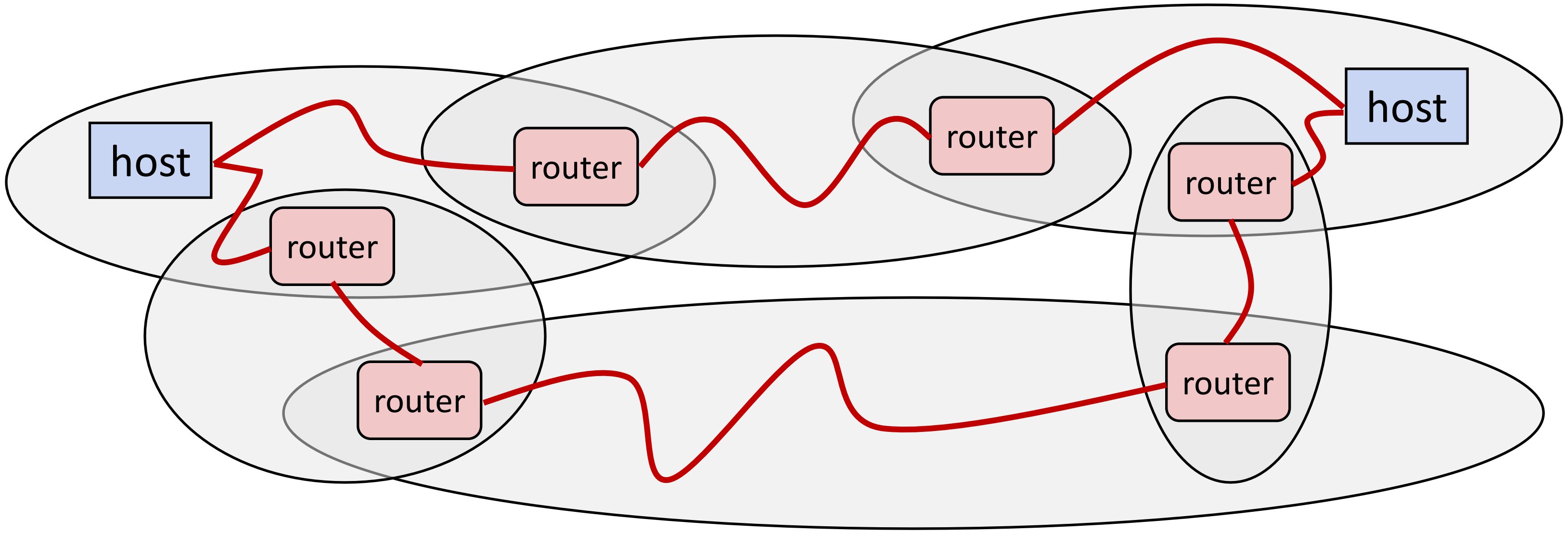
- There are lots of lots of local area networks (LANs)
  - Each determines its own protocols, address format, packet format

- What if we wanted to connect them together?
  - Physically connected by specialized computers called routers
  - Routers with multiple network adapters can translate
  - Standardize address and packet formats



- This is an internetwork
  - Aka wide-area network (WAN)
  - Aka internet

Logical Structure of an internet



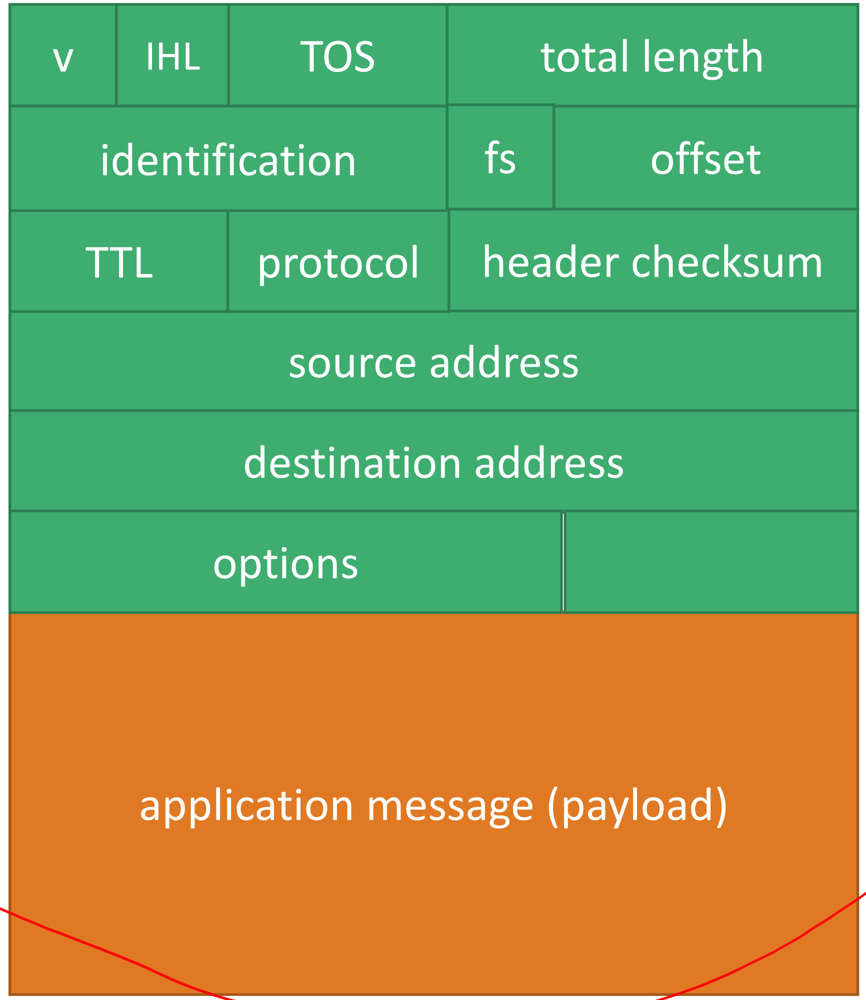
- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router forms bridge from one network to another
  - Different packets may take different routes

Internet Protocol (IP)

- Initiated by the DoD in 60s-70s
- Currently transitioning (very slowly) from IPv4 to IPv6

IPv4

- Example address: `128.84.12.43`
- Interoperable
- Network dynamically routes packets from source to destination



Aside: IPv4 and IPv6

- The original Internet Protocol, with its 32-bit addresses, is known as Internet Protocol Version 4 (IPv4)

- 1996: Internet Engineering Task Force (IETF) introduced Internet Protocol Version 6 (IPv6) with 128-bit addresses
  - Intended as the successor to IPv4

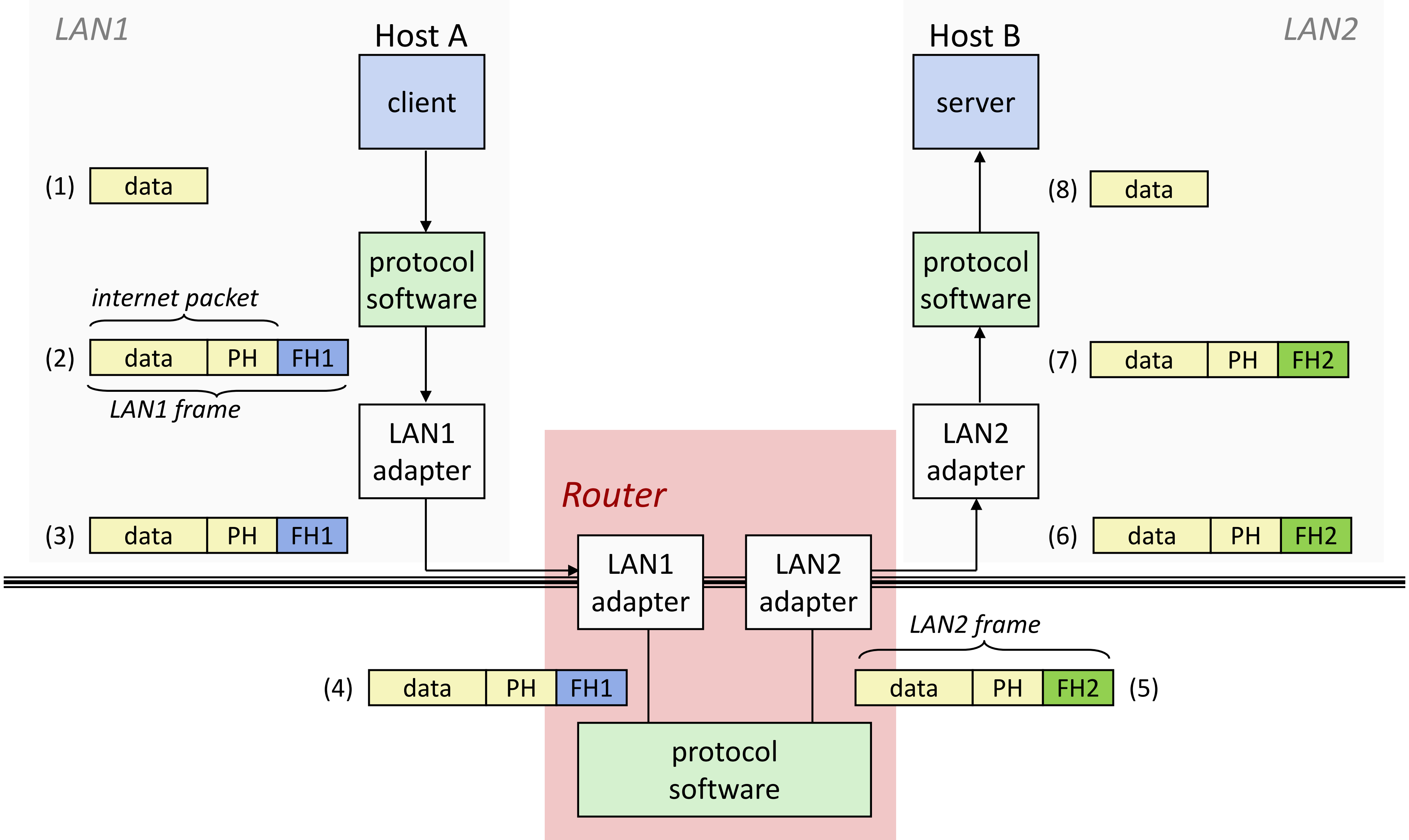
- As of November 2022, majority of Internet traffic still carried by IPv4
  - 40% of users access Google services using IPv6.
  - Up from about 30% in Nov 2020

- We will focus on IPv4 but we'll see how to write networking code that is protocol-independent.

PH: Internet packet header
FH: LAN frame header

Transferring internet Data Via Encapsulation

Application can scramble the data for privacy

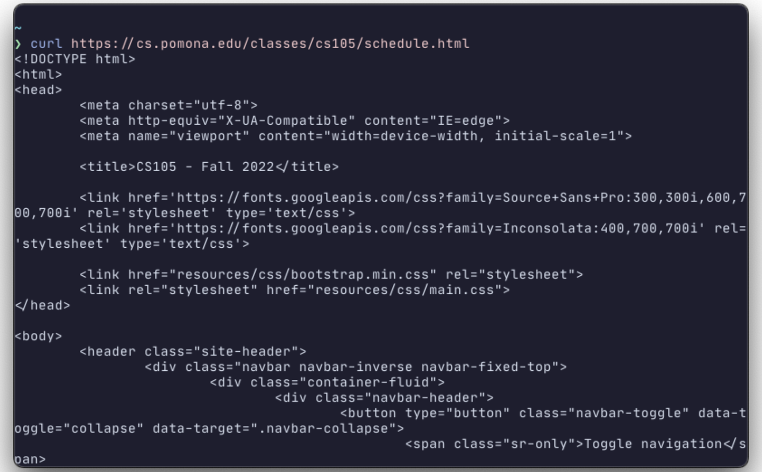


Exercise 2: IP addresses

What is the current IP address assigned to your computer?

Try these:

```
curl https://cs.pomona.edu/classes/cs105/schedule.html
curl ipinfo.io
```



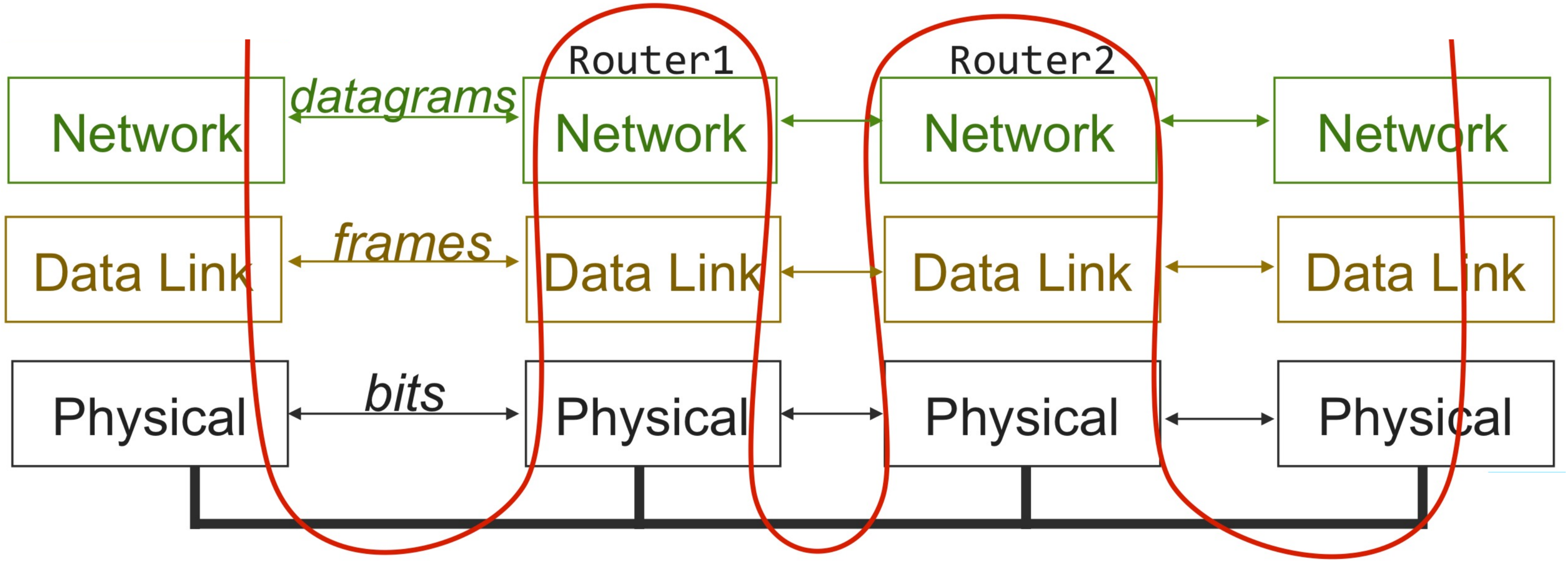
```
~
❯ curl ipinfo.io
{
  "ip": "134.173.84.6",
  "city": "Claremont",
  "region": "California",
  "country": "US",
  "loc": "34.0967,-117.7198",
  "org": "AS3659 Claremont University Consortium",
  "postal": "91711",
  "timezone": "America/Los_Angeles",
  "readme": "https://ipinfo.io/missingauth"
}⏎

~
❯
```

Routing

What if a packet doesn't reach its destination?

That depends on the layer above the network.

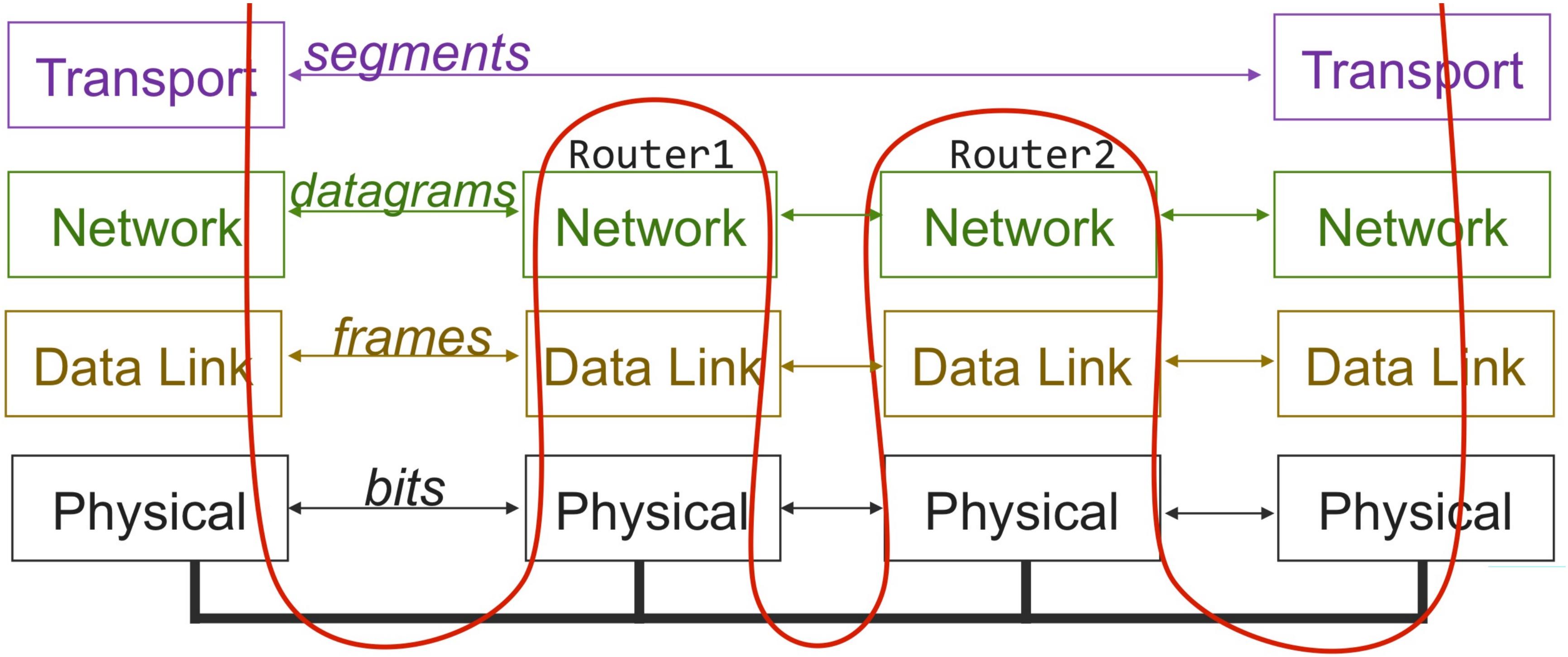


Next Up → Transport

Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding

(Hard to draw firm lines here)

Transport Layer



Transport Layer

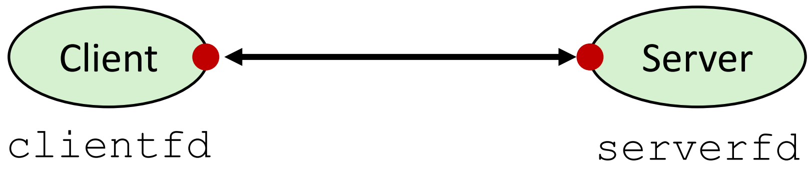
- Clients and servers communicate by sending streams of bytes over a **connection**.
- A transport layer endpoint is identified by an **IP address** and a **port**, a 16-bit integer that identifies a process
  - Ephemeral port: Assigned automatically by client kernel when client makes a connection request.
  - Well-known port: Associated with some service provided by a server (e.g., port 80 is associated with Web servers)

(Unix) Socket Programming

What is an endpoint?

- A socket
- IP address + port
- To the OS kernel, a socket is an endpoint of communication
- To an application, a socket is a file descriptor that lets the application read/write from/to the network (***Note:*** All Unix I/O devices, including networks, are modeled as files)

Hosts communicate with each other by reading from and writing to socket descriptors

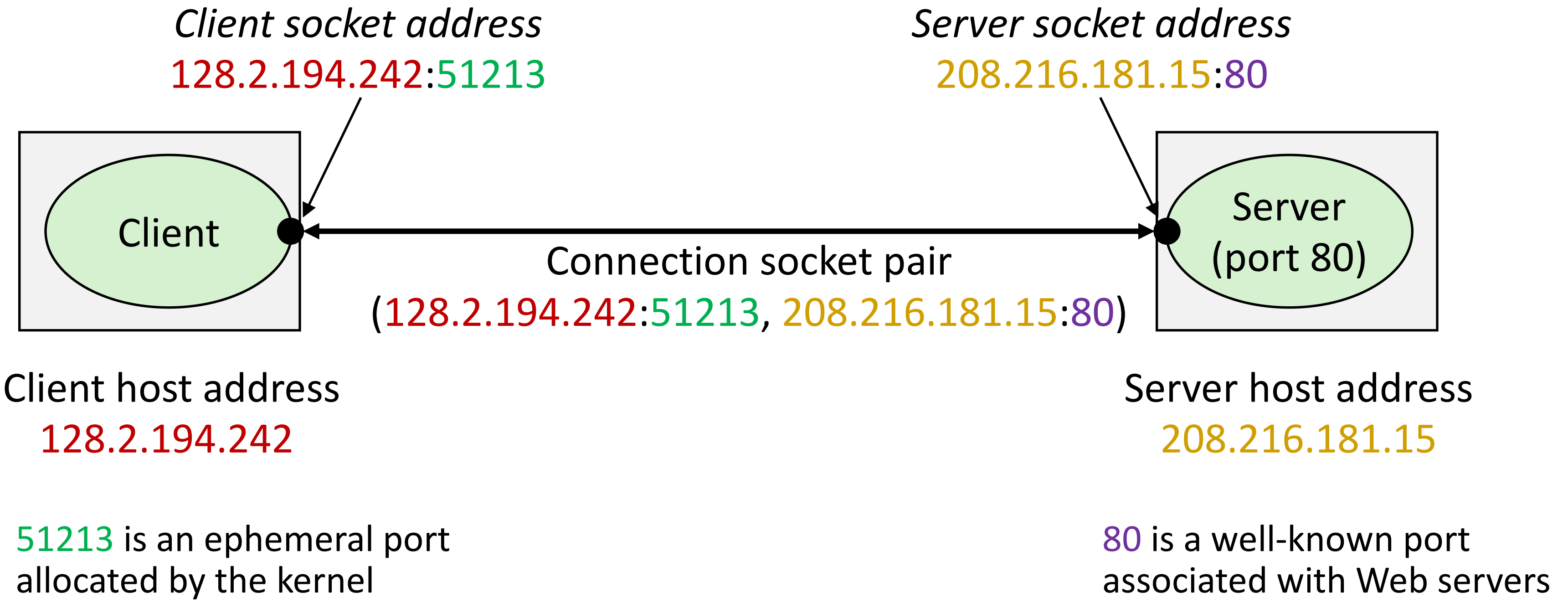


The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors

Anatomy of a Connection

A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)

`(cliaddr:cliport, servaddr:servport)`



Well-known Ports and Service Names

- Popular services have permanently assigned **well-known ports** *and* corresponding ***well-known service names***:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - Web servers: 80/http

- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.

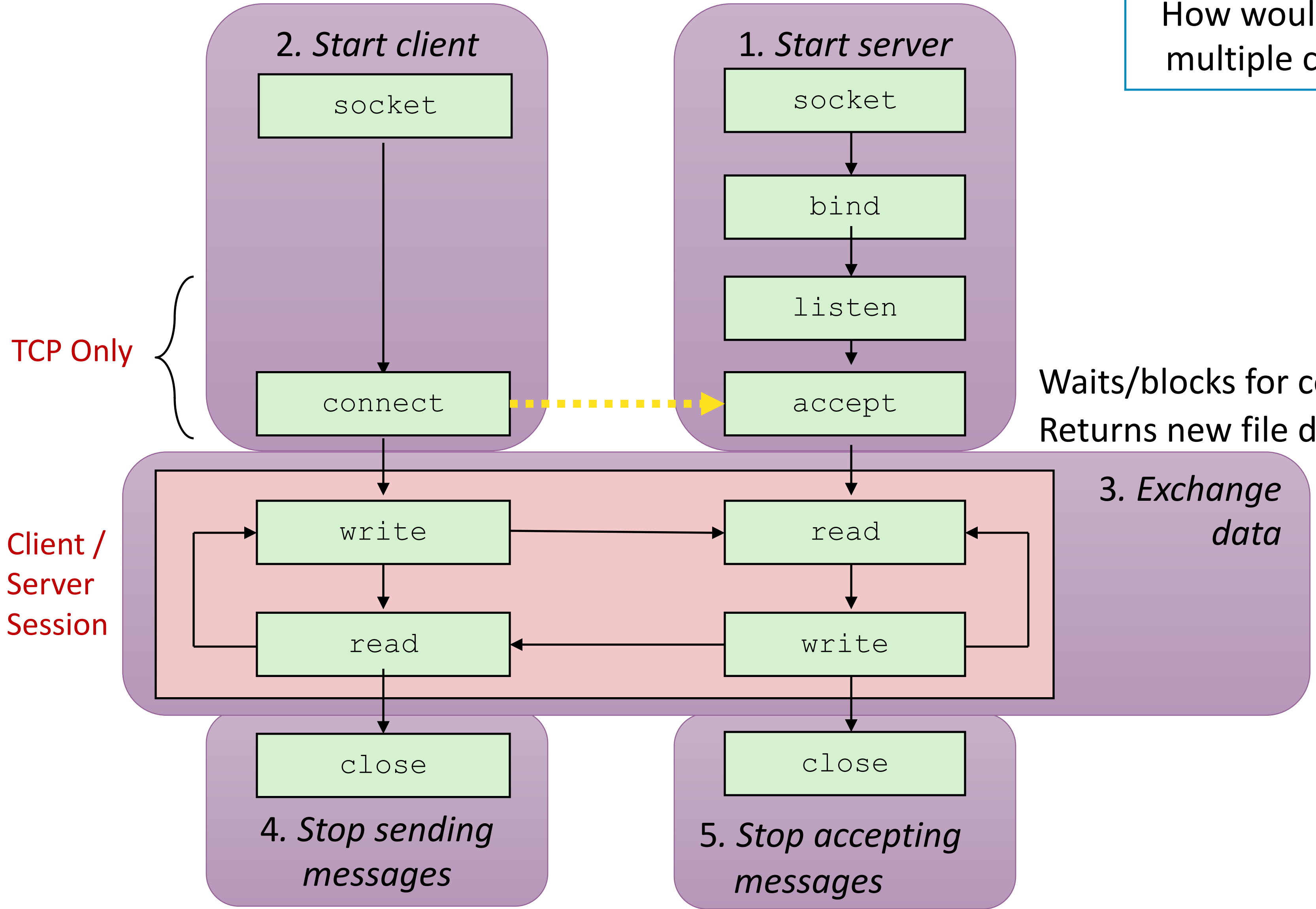
File: **/etc/services**

```
# /etc/services:
# $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2016-07-08
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#       http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]

tcpmux          1/tcp                           # TCP port service multiplexer
tcpmux          1/udp                           # TCP port service multiplexer
rje             5/tcp                           # Remote Job Entry
rje             5/udp                           # Remote Job Entry
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
systat          11/udp          users
daytime         13/tcp
daytime         13/udp
qotd            17/tcp          quote
qotd            17/udp          quote
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp-data        20/udp
# 21 is registered to ftp, but also used by fsp
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                          # The Secure Shell (SSH) Protocol
ssh             22/udp                          # The Secure Shell (SSH) Protocol
telnet          23/tcp
telnet          23/udp
# 24 - private mail system
lmtp            24/tcp                          # LMTP Mail Delivery
lmtp            24/udp                          # LMTP Mail Delivery
smtp            25/tcp          mail
smtp            25/udp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/tcp          resource        # resource location
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
:
```

Sockets Interface

How would you accept multiple connections?



Waits/blocks for connection
Returns new file descriptor for read/write

```
while(1) {  // main accept() loop
    sin_size = sizeof their_addr;
    new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
    if (new_fd == -1) {
        perror("accept");
        continue;
    }

    inet_ntop(their_addr.ss_family,
        get_in_addr((struct sockaddr *)&their_addr),
        s, sizeof s);
    printf("server: got connection from %s\n", s);

    if (!fork()) { // this is the child process
        close(sockfd); // child doesn't need the listener
        if (send(new_fd, "Hello, world!", 13, 0) == -1)
            perror("send");
        close(new_fd);
        exit(0);
    }
    close(new_fd);  // parent doesn't need this
}
```

Sockets Interface: `socket`

- Clients and servers use the `socket` function to create a *socket descriptor*:

```
int socket(int domain, int type, int protocol)
```

- Example:

```
int clientfd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
int clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

Indicates that we are using 32-bit IPV4 addresses (AF_INET)

Indicates transport protocol (SOCK_STREAM)

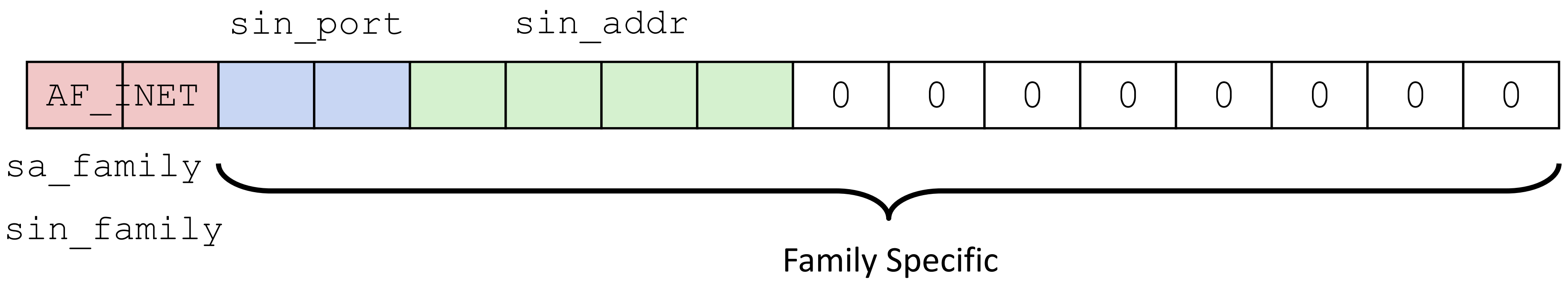
Protocol specific! Best practice is to use `getaddrinfo` to generate the parameters automatically, so that code is protocol independent.

```
getaddrinfo("www.example.com", "http", &hints, &res);
int s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

Socket Address Structures

- Internet-specific socket address:
  - Must cast (`struct sockaddr_in *`) to (`struct sockaddr *`) for functions that take socket address arguments.

```
struct sockaddr_in  {
  uint16_t        sin_family;  /* Protocol family (always AF_INET) */
  uint16_t        sin_port;    /* Port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```



Sockets Interface: `bind`

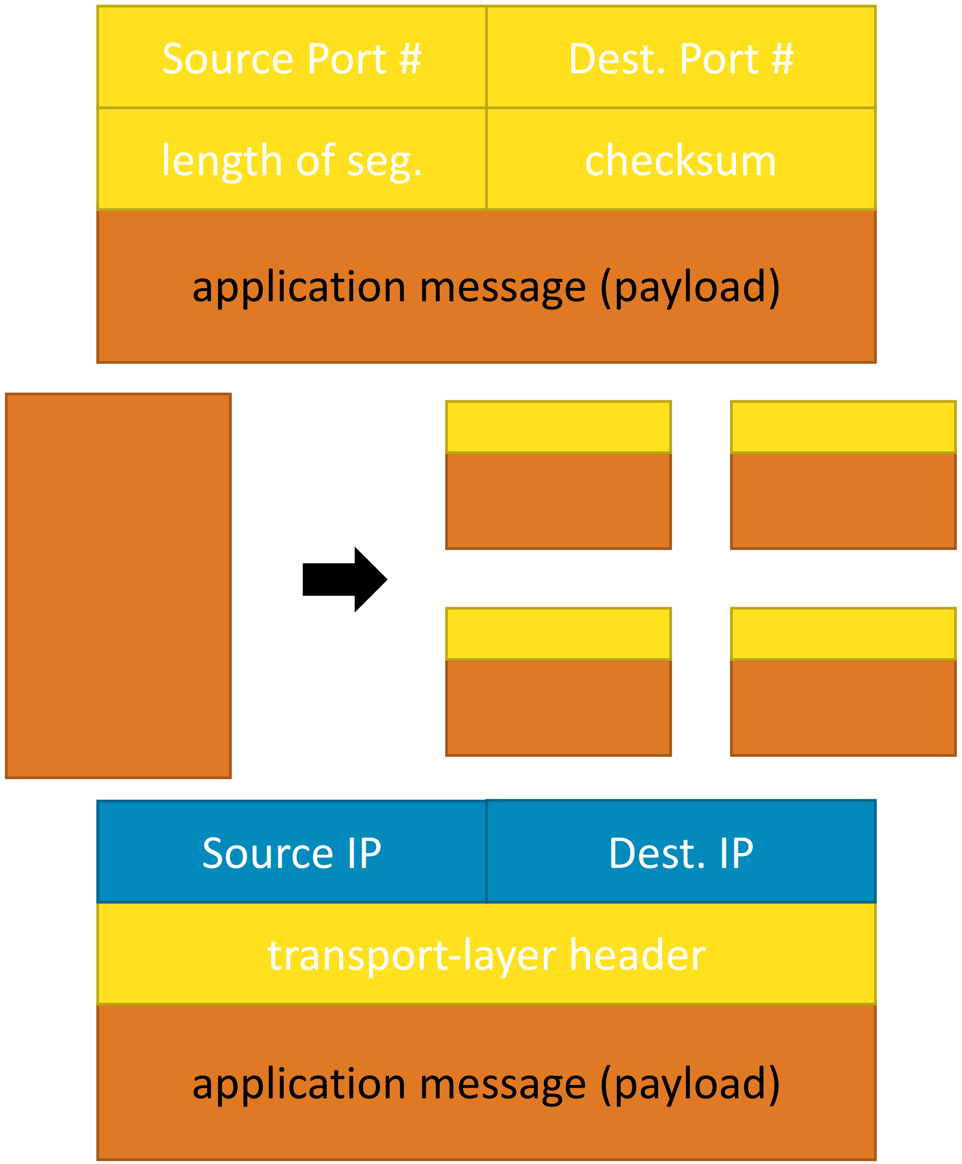
- A server uses `bind` to ask the kernel to associate the server's socket address with a socket descriptor:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- The process can read bytes that arrive on the connection whose endpoint is `addr` by reading from descriptor `sockfd`.
- Similarly, writes to `sockfd` are transferred along connection whose endpoint is `addr`.
- Protocol specific! Best practice is to use `getaddrinfo` to generate the parameters automatically, so that code is protocol independent.

Transport Layer Segments

- Sending application:
  - Specifies IP address and port
  - Uses socket bound to source port
- Transport layer:
  - Breaks application message into smaller chunks
  - Adds transport-layer header to each message to form a segment
- Network layer (IP):
  - Adds network-layer header to each datagram



Should the transport layer guarantee packet delivery?

No. It might not be necessary for all applications.

Transport Layer Protocols

User Datagram Protocol (UDP)

- Unreliable, unordered delivery
- Connectionless
- Best-effort, segments might be lost, delivered out-of-order, duplicated
- Reliability (if required) is the responsibility of the app

Transmission Control Protocol (TCP)

- Reliable, inorder delivery
- Connection setup
- Flow control
- Congestion control

Note: neither guarantees latency or bandwidth

UDP: tradeoffs

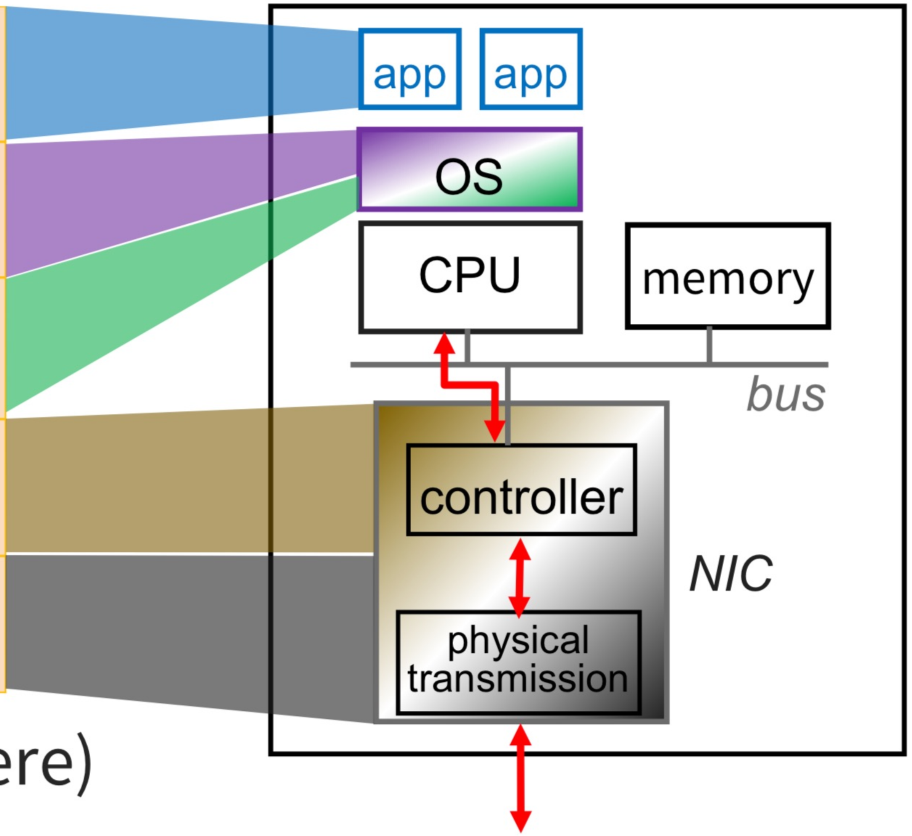
- Fast:
  - No connection setup
  - No rate-limiting
- Simple:
  - No connection state
  - Small header (8 bytes)
- (Possibly) extra work for applications
  - Reordering
  - Duplicate suppression
  - Handle missing packets

Transport Protocols by Application

Application	Application-Level Protocol	Transport Protocol
Name Translation	DNS	Typically UDP
Routing Protocol	RIP	Typically UDP
Network Management	SNMP	Typically UDP
Remote File Server	NFS	Typically UDP
Streaming multimedia	(proprietary)	UDP or TCP
Internet telephony	(proprietary)	UDP or TCP
Remote terminal access	Telnet	TCP
File Transfer	(S)FTP	TCP
Email	SMTP	TCP
Web	HTTP(S)	TCP

Hardware and Software Interfaces

Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding



(Hard to draw firm lines here)

The Big Picture