Network and Security: Introduction

Seungwon Shin
KAIST

Some slides are from Dr. Srinivasan Seshan
Some slides are from Dr. Nick Mckeown

Network Overview

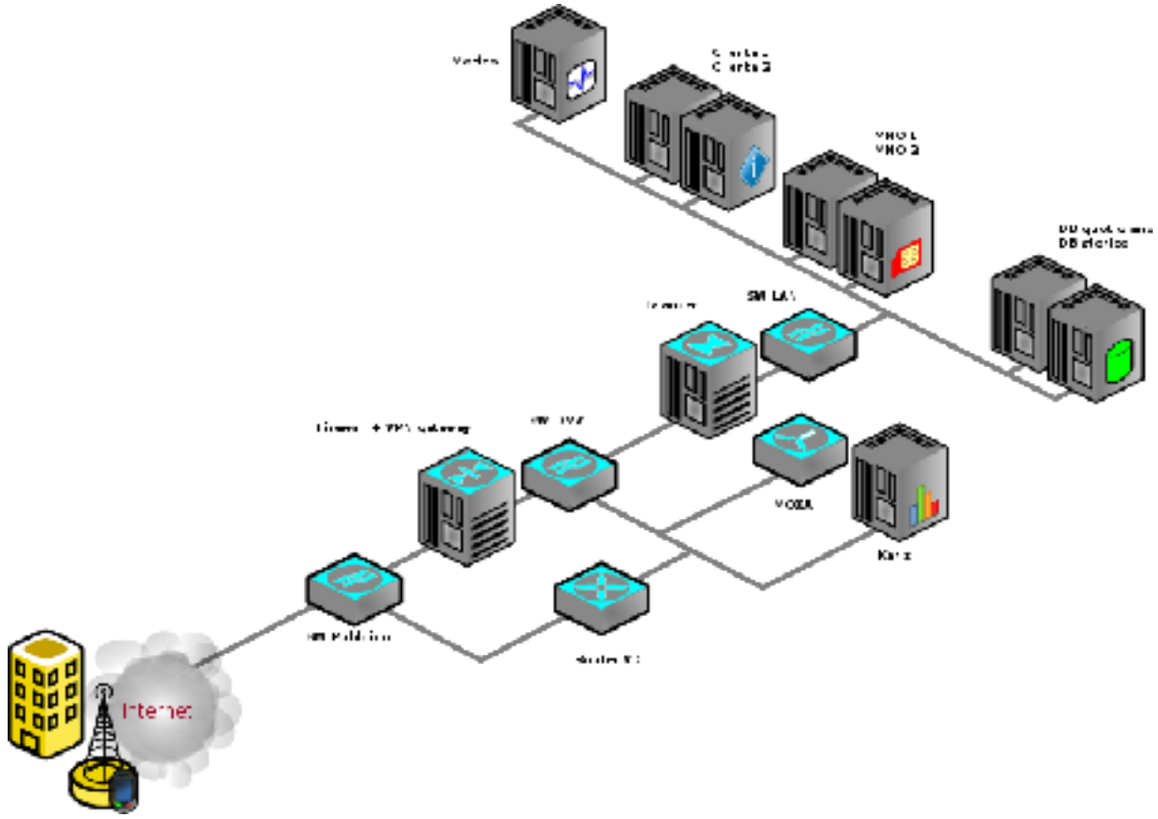
Computer Network

- Definition

  A computer network or data network is a telecommunications network that allows computers to exchange data. In computer networks, networked computing devices pass data to each other along data connections. - ***from Wikipedia***

Computer **Exchange** **Data**

Computer Network



Why is it important?

- *from norman-networok.net*

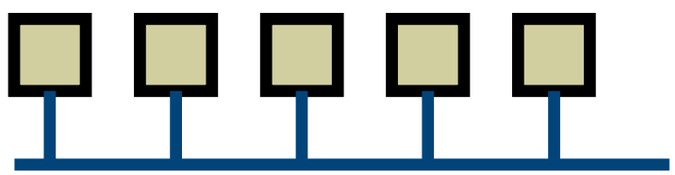
Everything is connected

Network Diagram

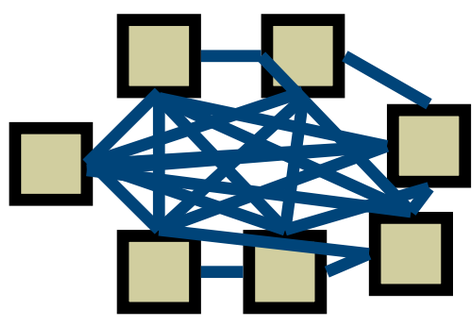
- Drawing something

direct connection

multiple hosts, multiple links



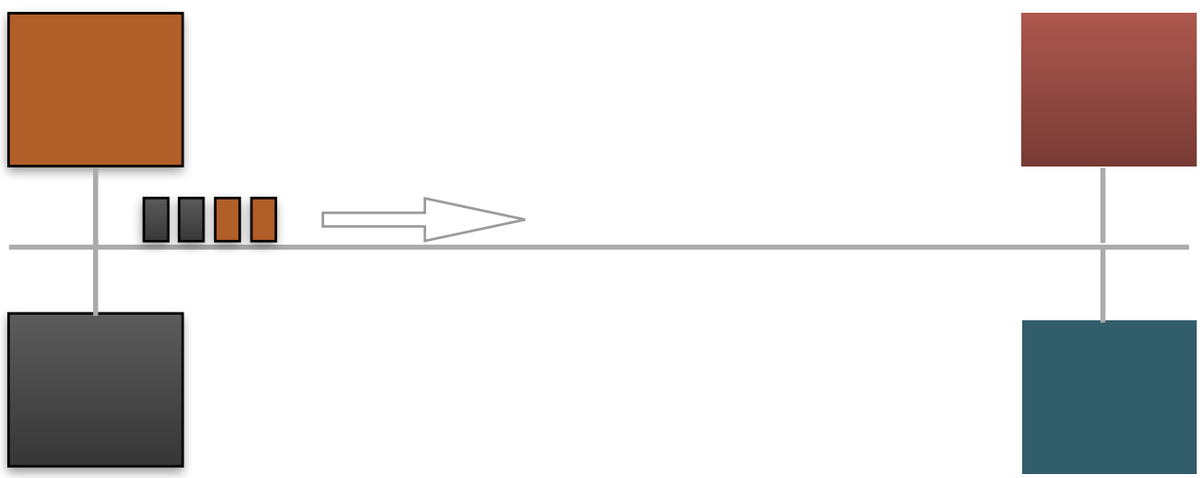
One wire



Wires for everybody!

Network Multiplexing

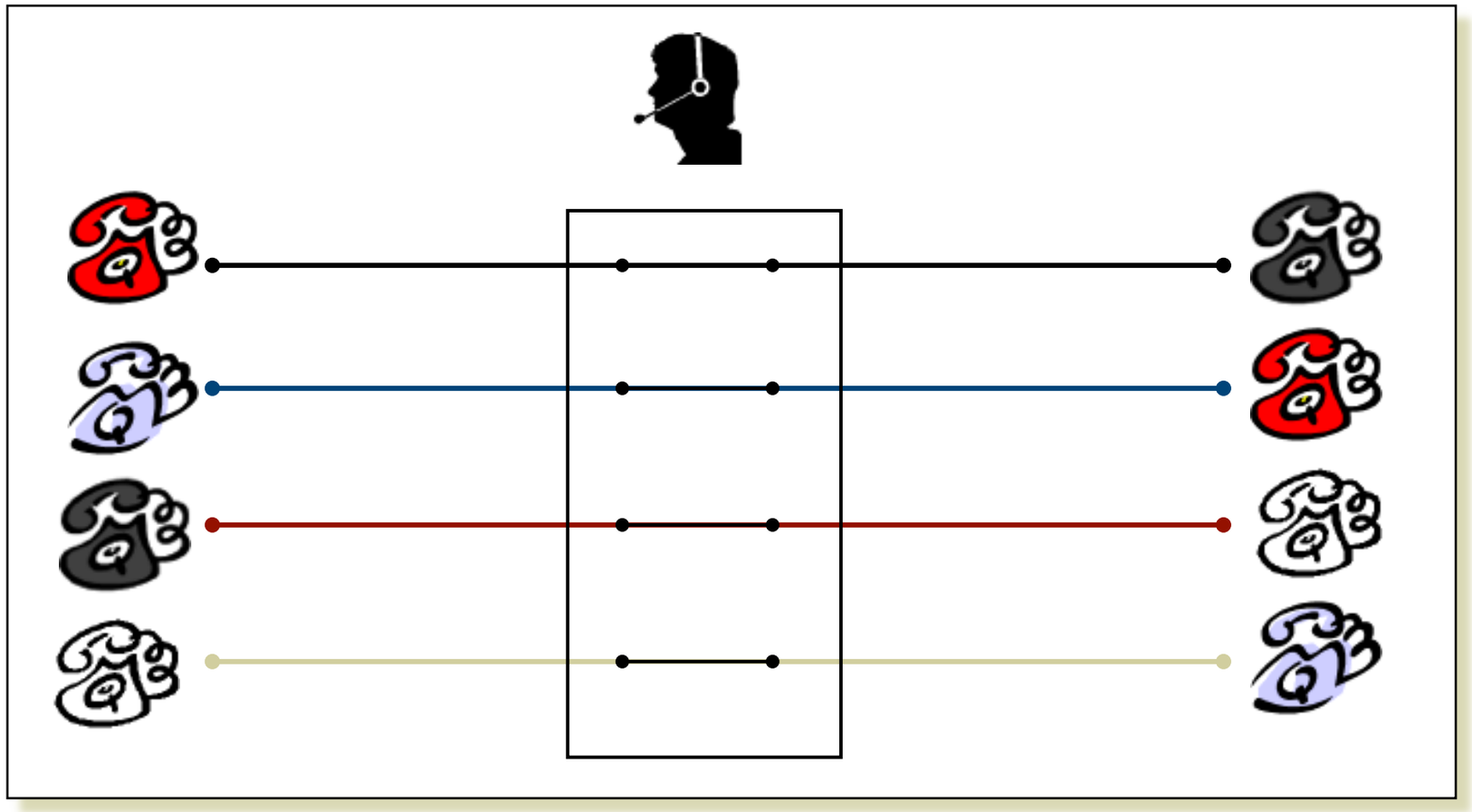
- Multiple hosts
  - How to share a network link
    - switched network
      - resource sharing
        - orange sends two packets to red - others are waiting
        - black sends two packets to green - others are waiting



Switching Approaches

- Circuit switching
  - Source first establishes a connection (circuit) to the destination
    - Each switch along the way stores info about connection (and possibly allocates resources)
  - Source sends the data over the circuit
    - No need to include the destination address with the data since the switches know the path
  - The connection is explicitly torn down
  - Example: telephone network (analog)

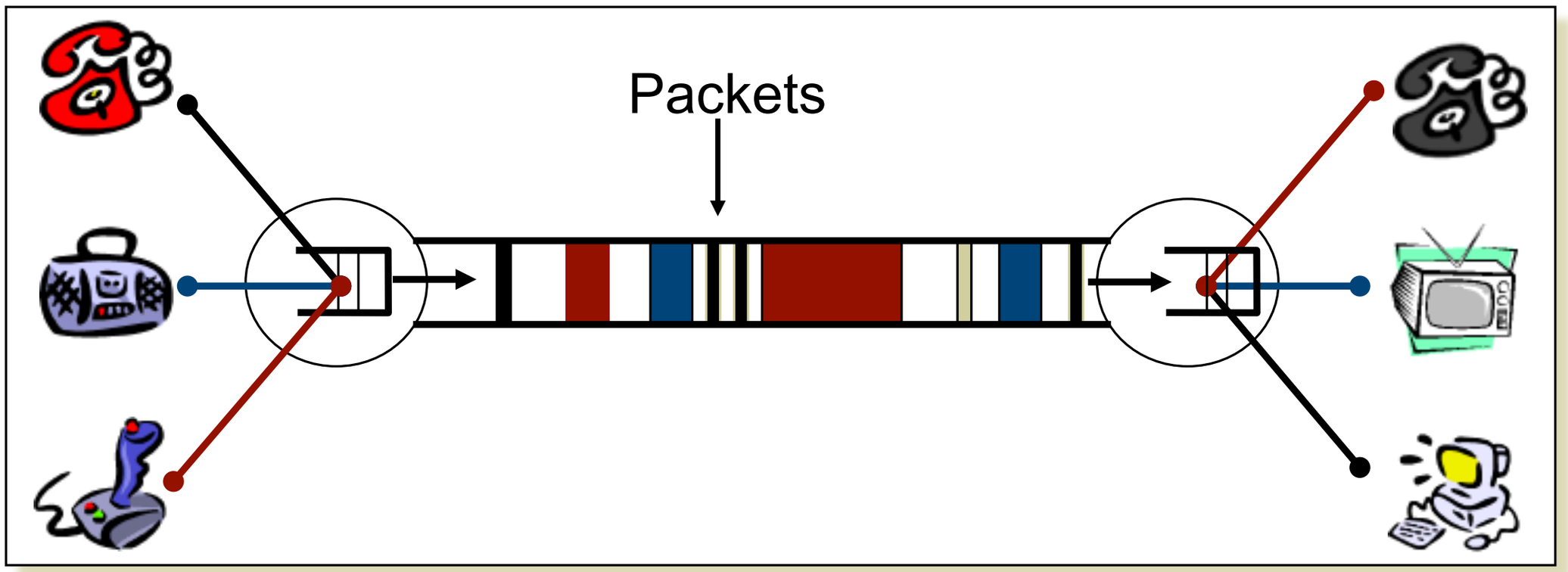
Circuit Switching



Switching Approaches

- Packet switching
  - Source sends information as self-contained packets that have an address.
    - Source may have to break up single message in multiple
  - Each packet travels independently to the destination host.
    - Switches use the address in the packet to determine how to forward the packets
    - Store and forward
  - Analogy: a letter in surface mail.

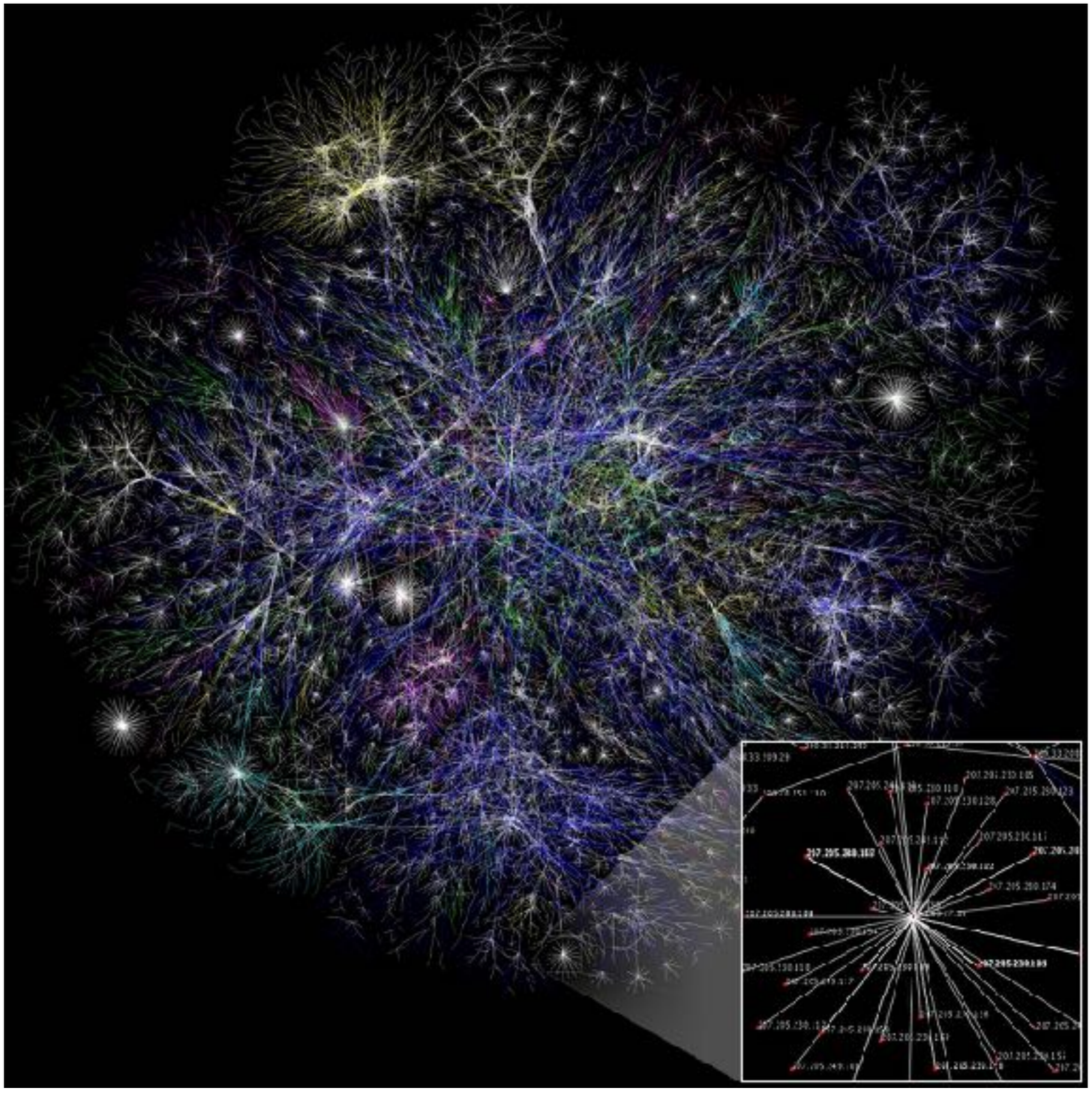
Packet Switching



Internet

- An inter-net: a network of networks.
- Networks are connected using routers that support communication in a hierarchical fashion
- Often need other special devices at the boundaries for security, accounting, ..
- The Internet: the interconnected set of networks of the Internet Service Providers (ISPs)
- About 17,000 different networks make up the Internet

Internet



How to find Nodes?

- Naming

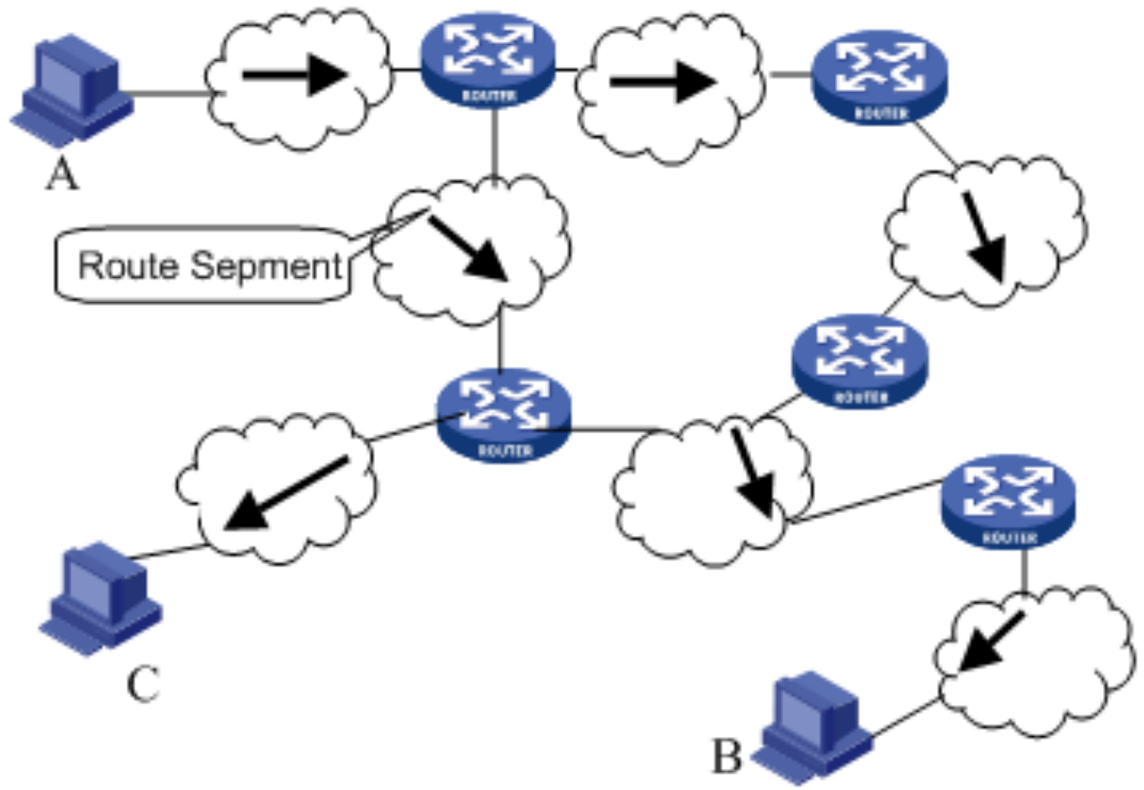
what is the IP address of nss.kaist.ac.kr

it is 143.248.111.111

DNS

Routing

- What is it?
  - an approach of sending packets to a destination
    - e.g.,
      - OSPF, BGP, ISIS, and more
  - routing vs. switching?



Protocol and Layer

Protocol

- Definition

A communication protocol is a system of digital rules for data exchange within or between computers. - ***from Wikipedia***

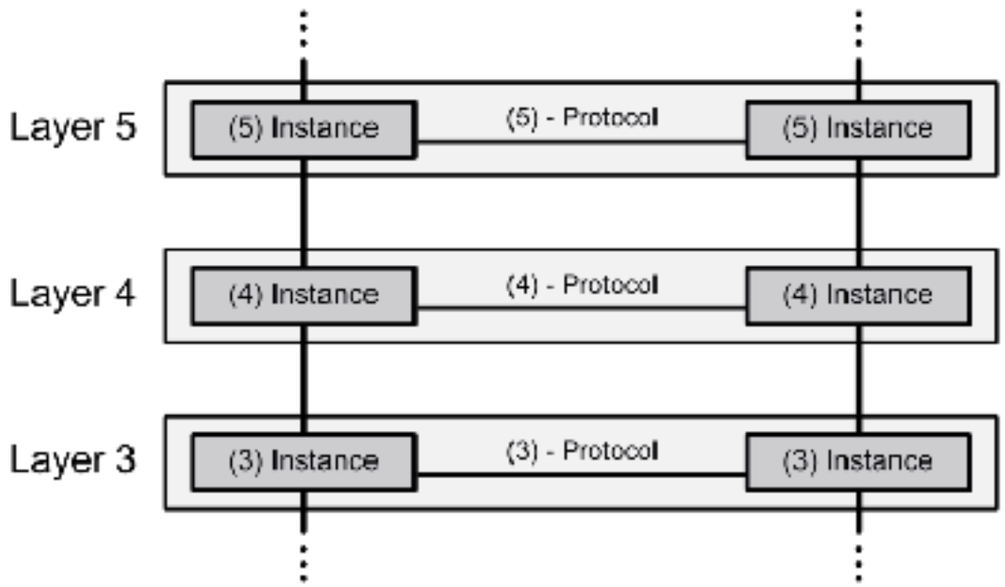
RULE for Communication

OSI Model

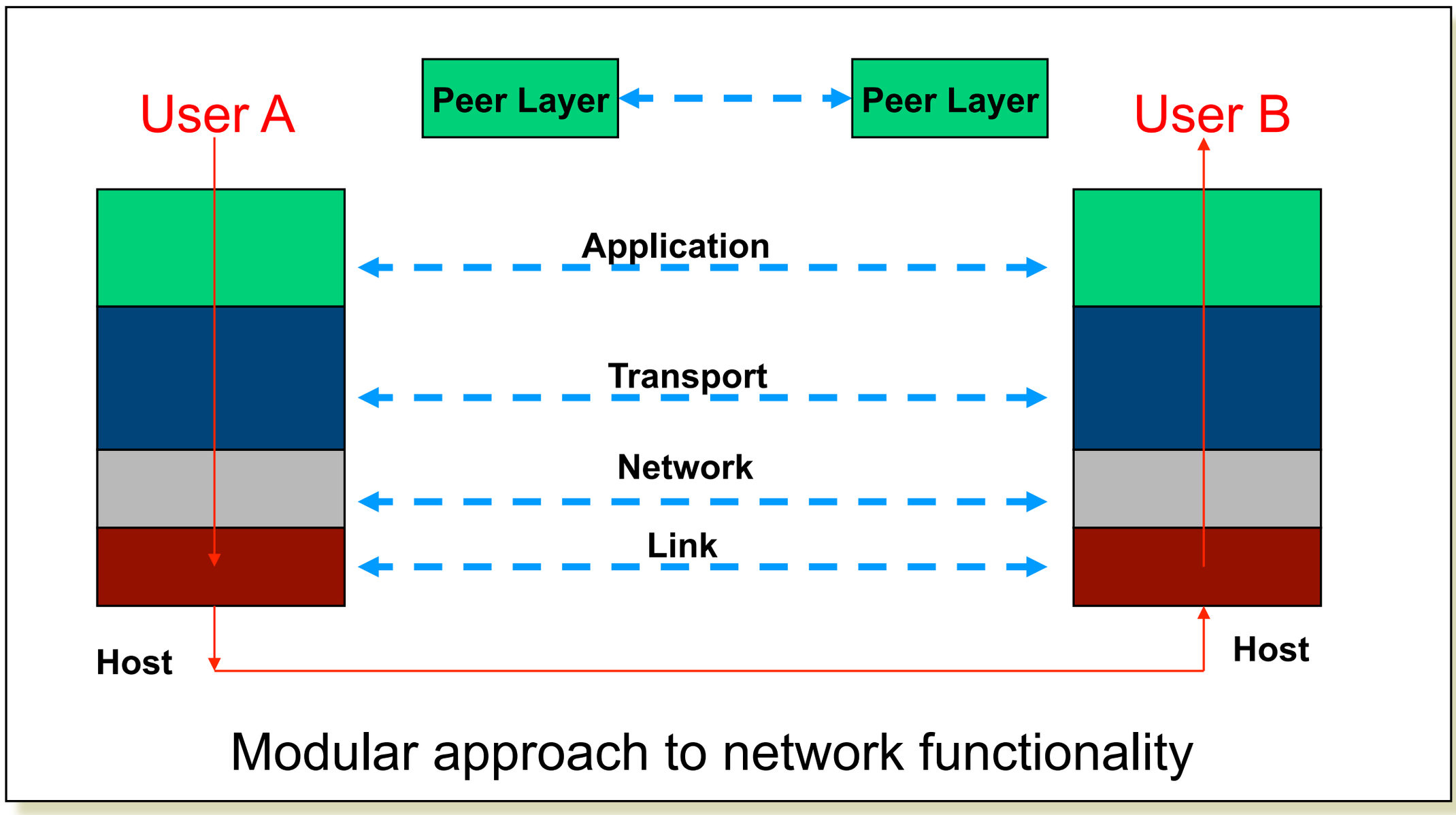
- OSI
  - Open System Interconnection model

- Definition

A conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers. - ***from Wikipedia***

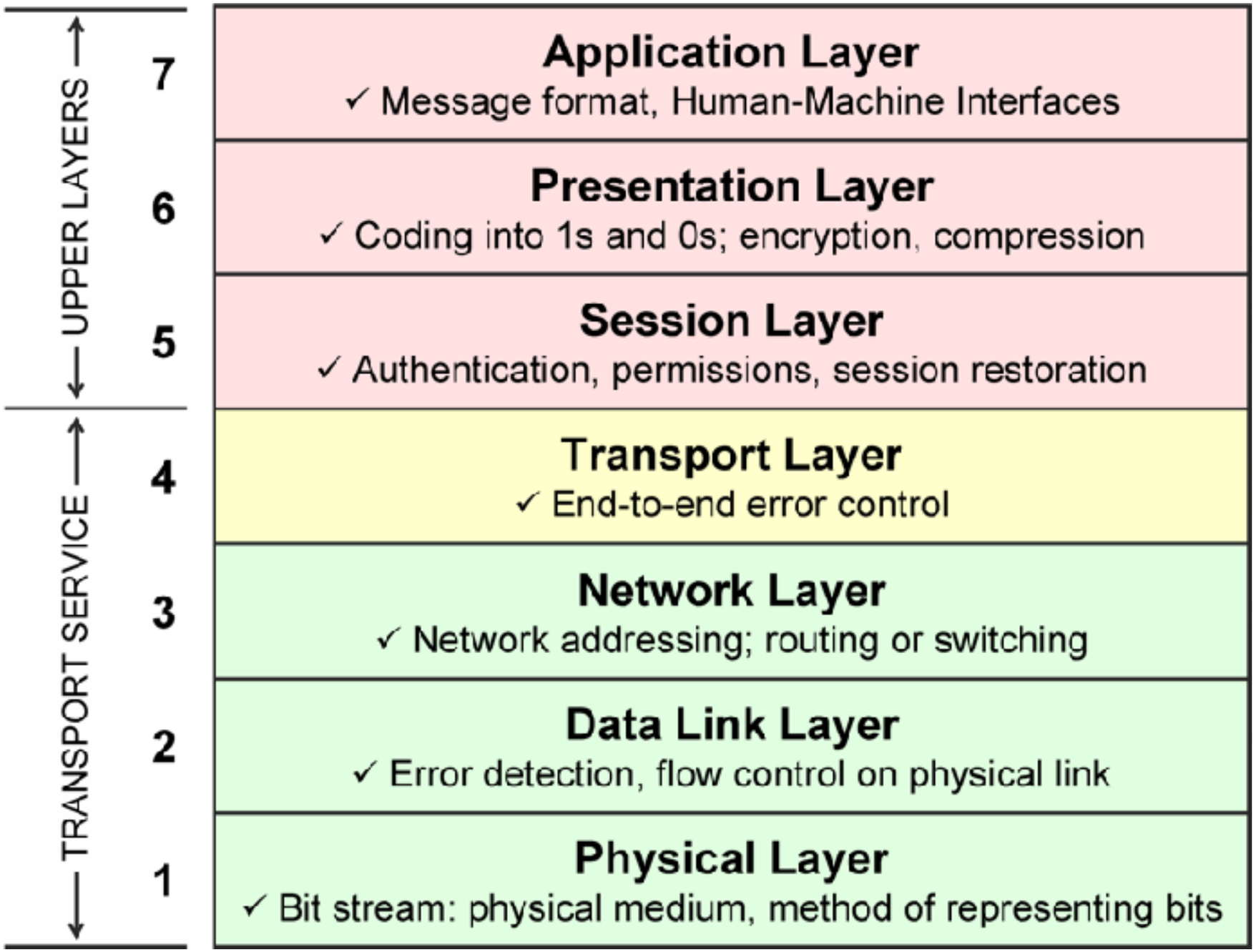


Layering



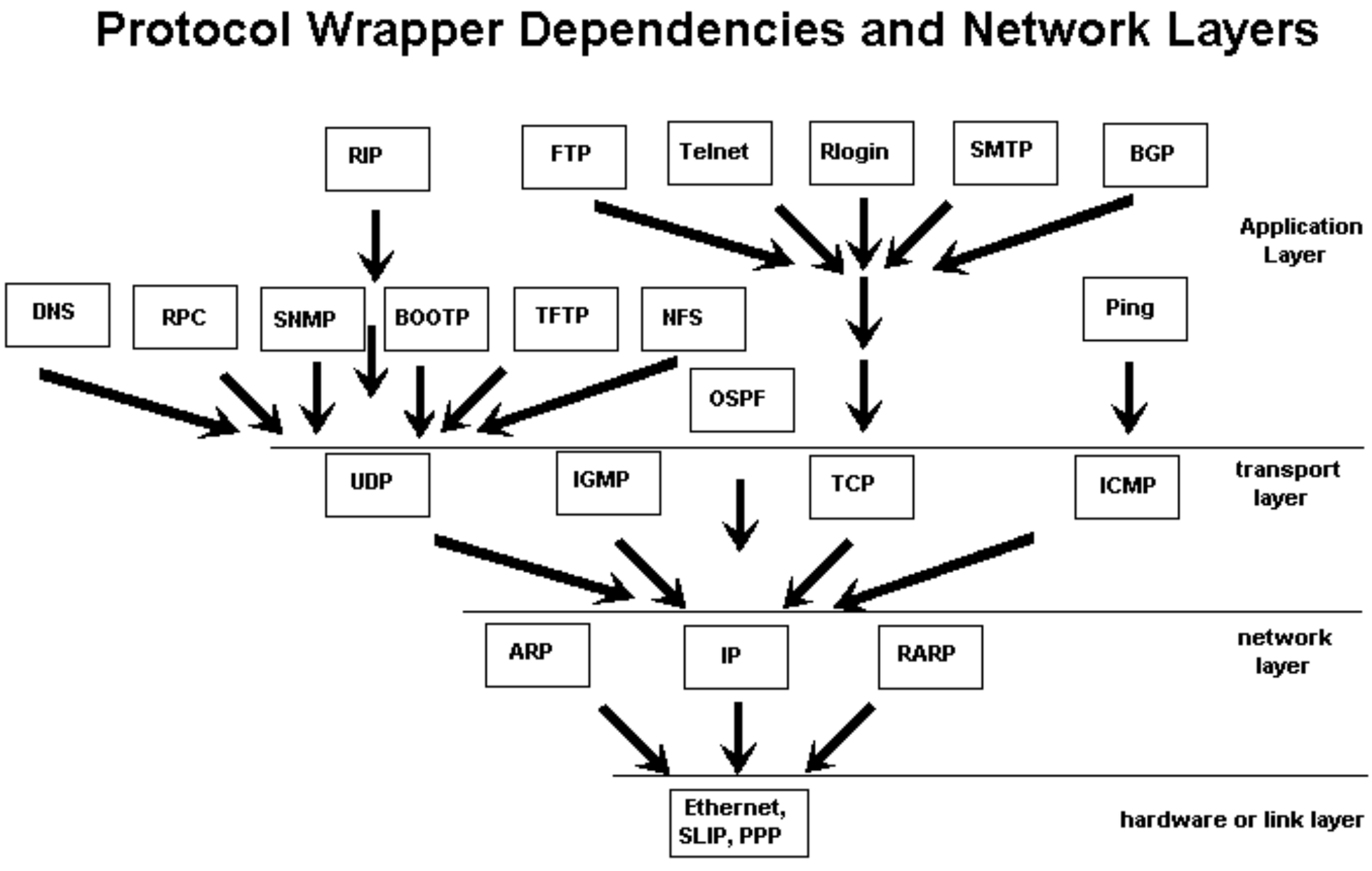
Modular approach to network functionality

OSI-7 Layer

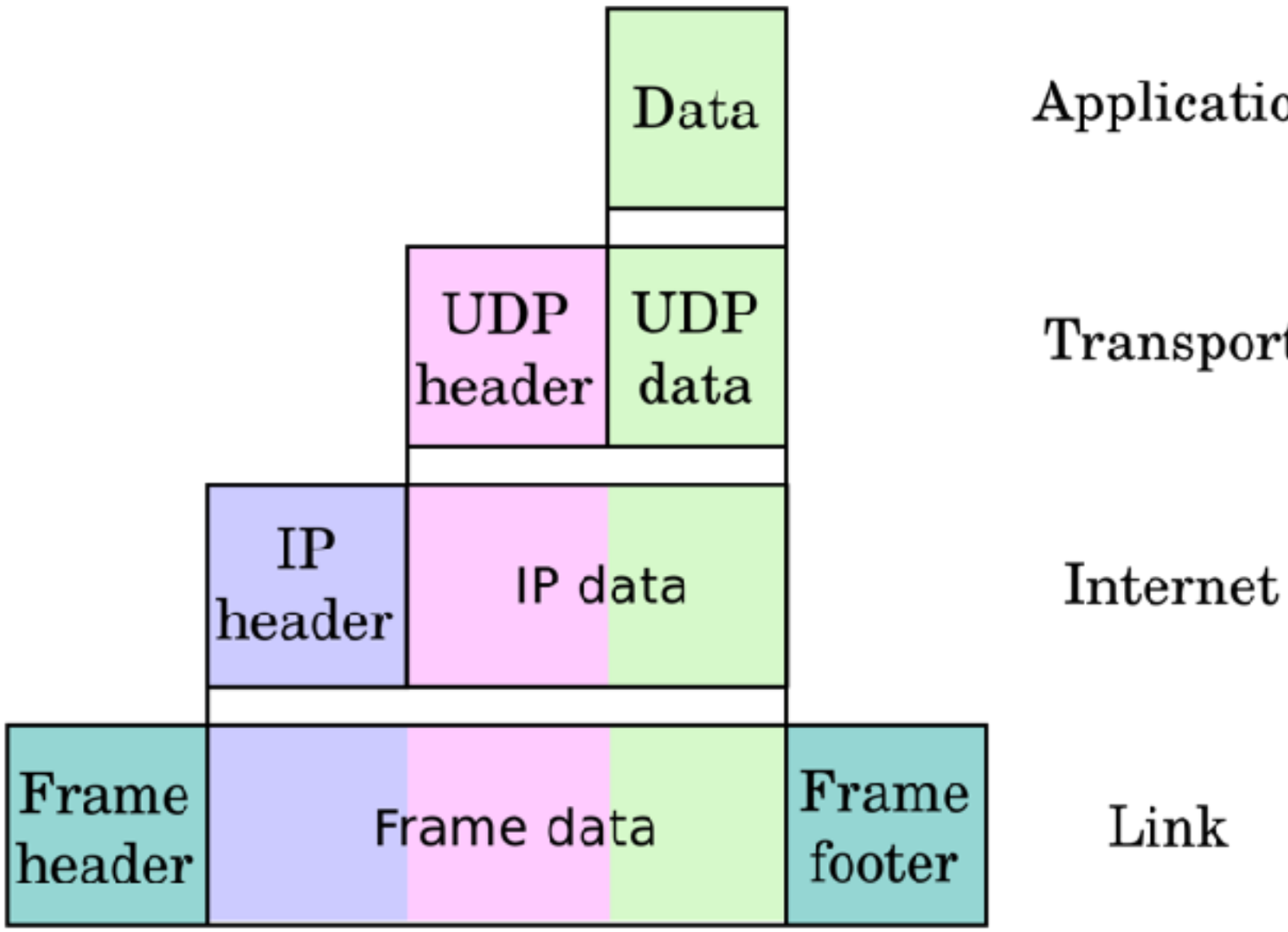


· *from cisco.com*

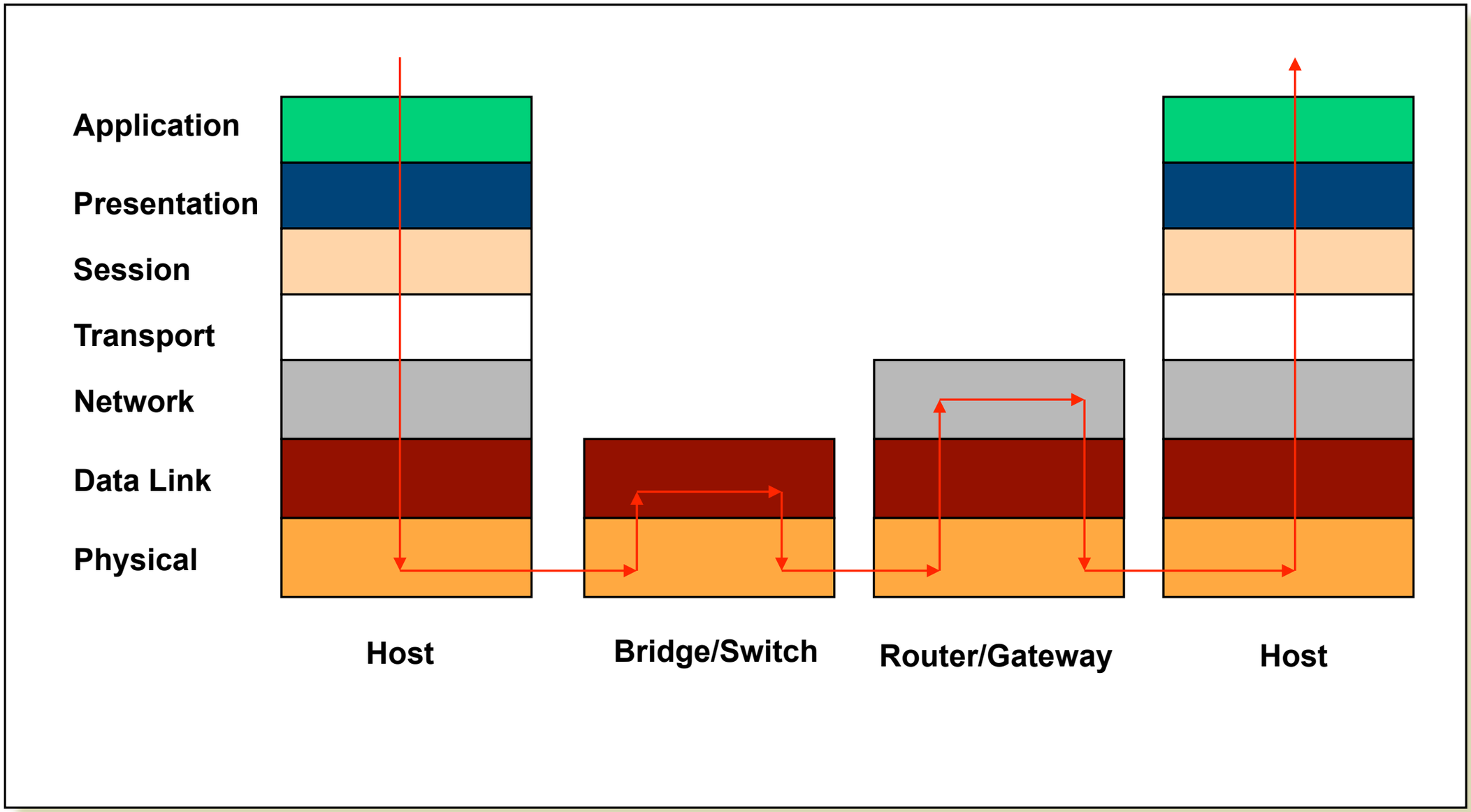
Protocols in OSI-7 layer



Headers


Data
UDP header
UDP data
IP header
IP data
Frame header
Frame data
Frame footer
Application
Transport
Internet
Link

Layering again

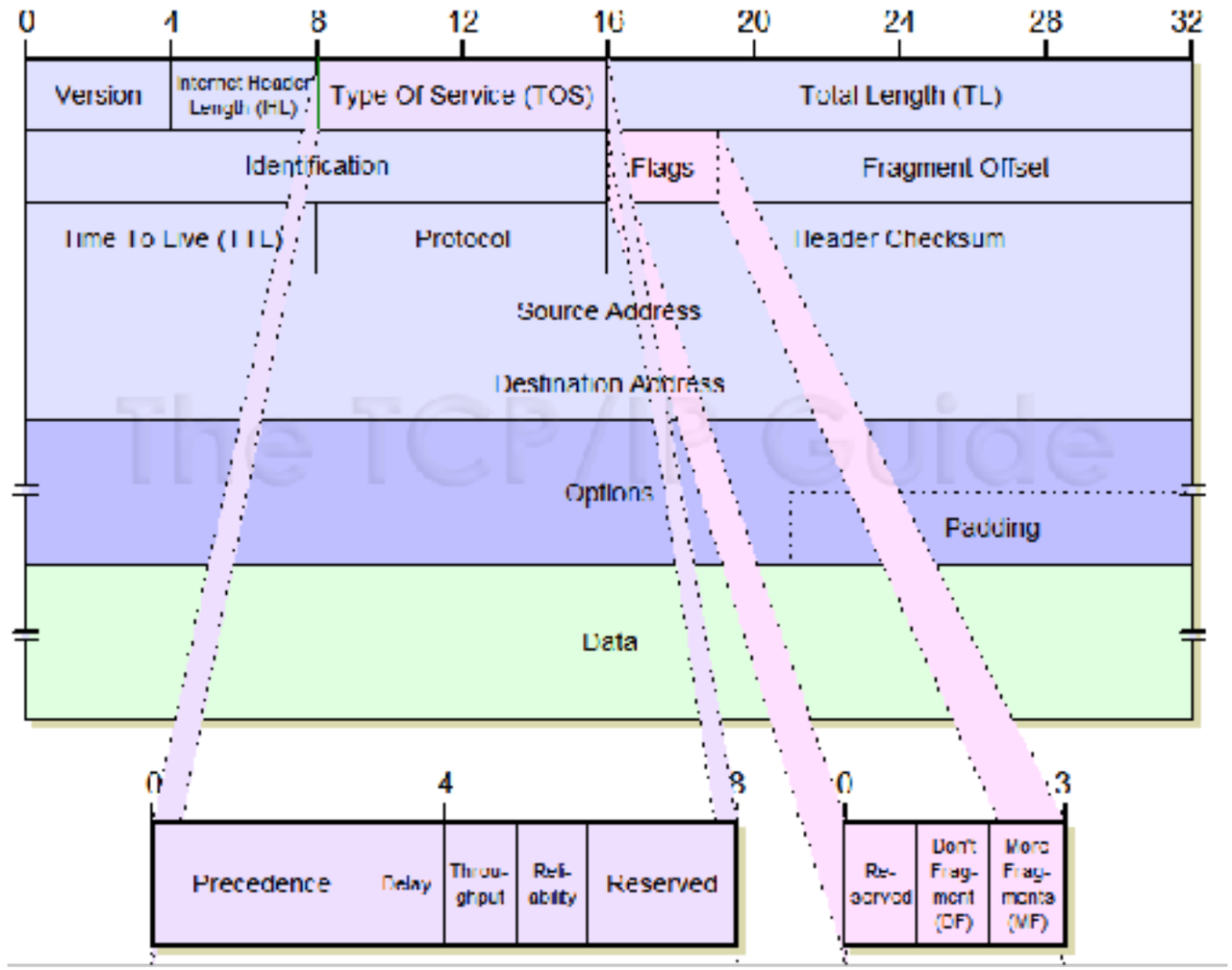


In Detail

IP

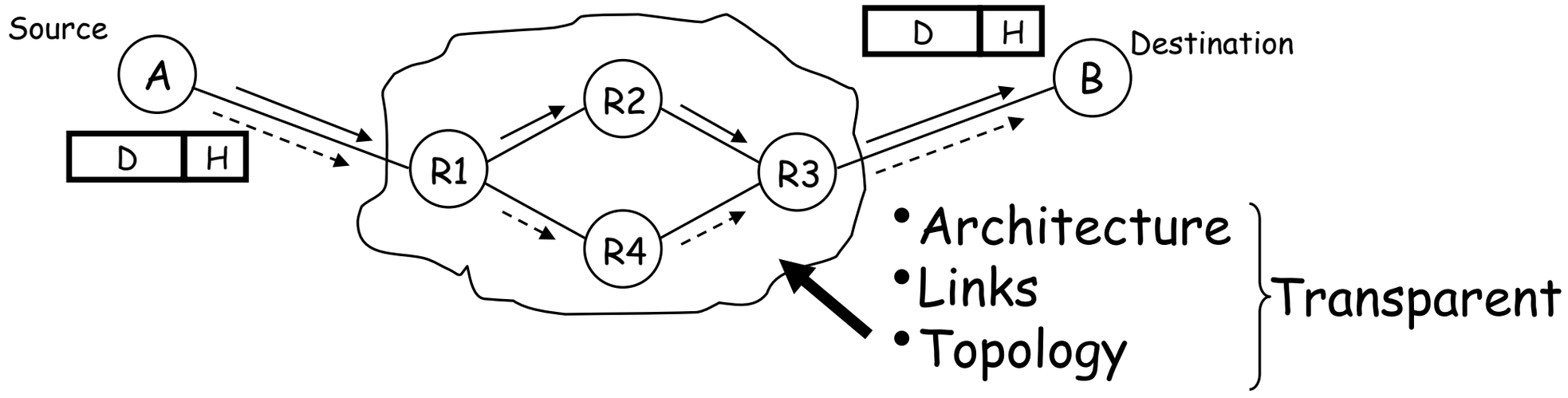
- Definition

  The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. - ***from Wikipedia***

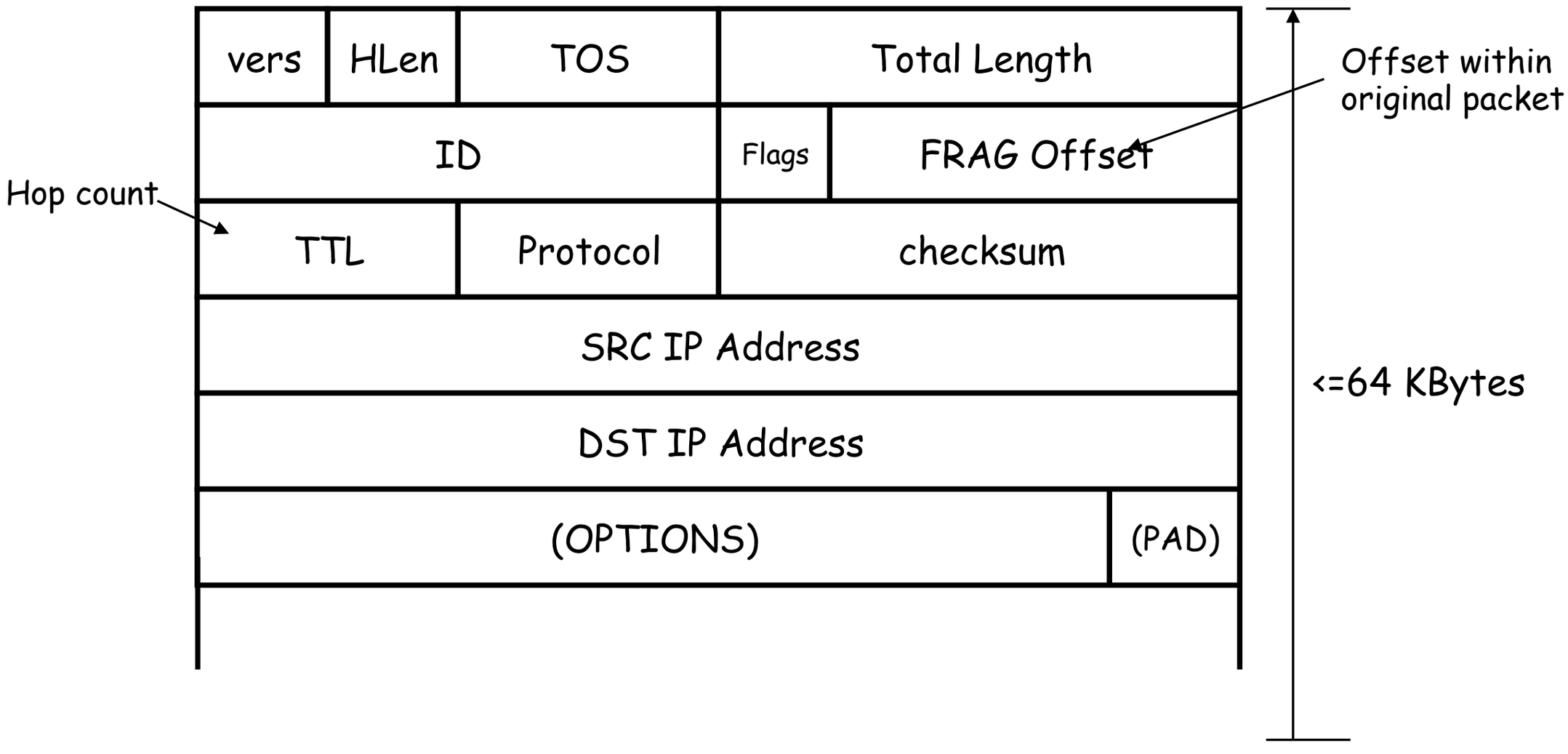


IP

- Characteristics of IP
  - CONNECTIONLESS: mis-sequencing
  - UNRELIABLE: may drop packets…
  - BEST EFFORT: … but only if necessary
  - DATAGRAM: individually routed

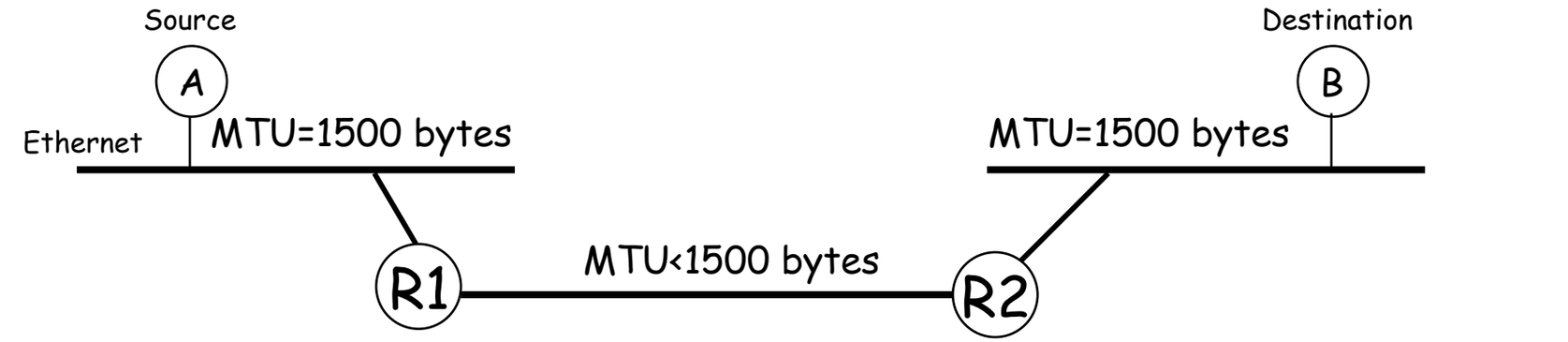


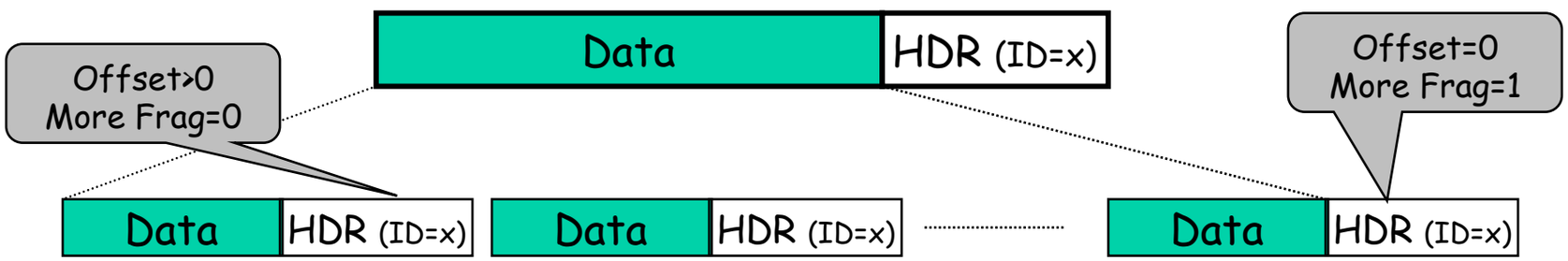
IP Datagram


vers
HLen
TOS
Total Length
ID
Flags
FRAG Offset
Hop count
TTL
Protocol
checksum
SRC IP Address
DST IP Address
(OPTIONS)
(PAD)
Offset within original packet
<=64 KBytes

IP Fragmentation

- A router may receive a packet larger than the maximum transmission unit (MTU) of the outgoing link





IP Fragmentation

- Fragments are re-assembled by the destination host; not by intermediate routers.
- To avoid fragmentation, hosts commonly use path MTU discovery to find the smallest MTU along the path.
- Path MTU discovery involves sending various size datagrams until they do not require fragmentation along the path.
- Most links use MTU>=1500bytes today.

ICMP

- Internet Control Message Protocol:
  - Used by a router/end-host to report some types of error:
  - E.g. Destination Unreachable: packet can’t be forwarded to/towards its destination.
  - E.g. Time Exceeded: TTL reached zero, or fragment didn’t arrive in time. Traceroute uses this error to its advantage.
  - An ICMP message is an IP datagram, and is sent back to the source of the packet that caused the error.

TCP and UDP

- TCP and UDP
  - TCP
    - Transmission Control Protocol
  - UDP
    - User Datagram Protocol
- core protocols of the Internet

Key difference between them?

TCP

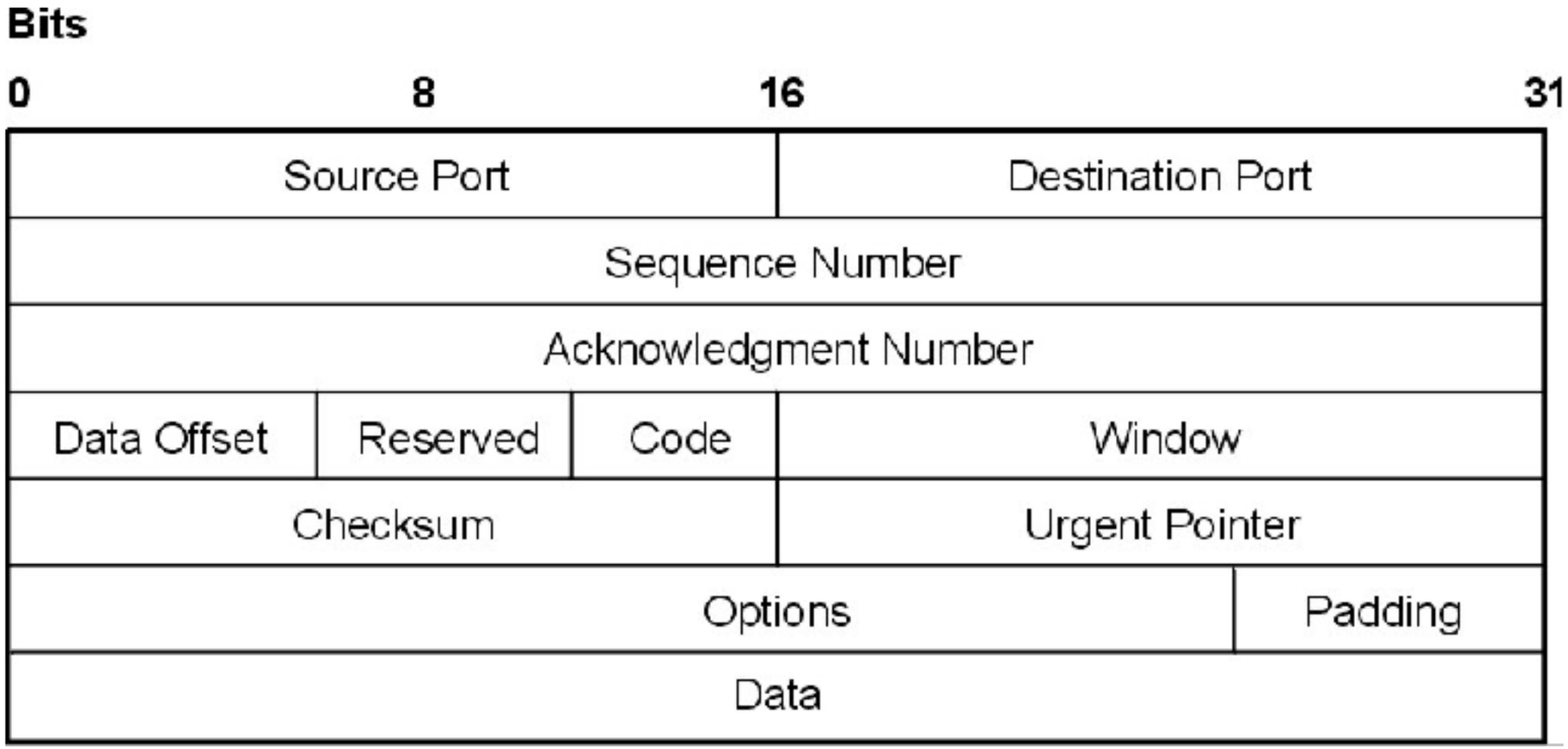
- Key features
  - connection oriented
  - reliable - how?
  - ordered - how?
  - traffic control - how?

Retransmission - ACK

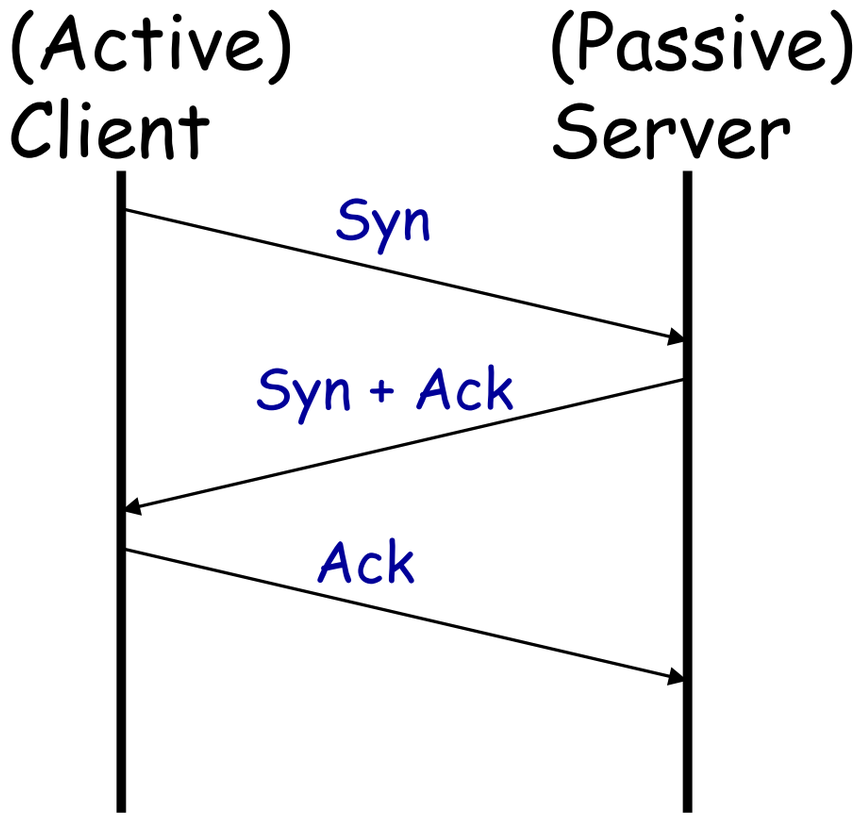
Sequence number - SEQ

Flow Control - Window size

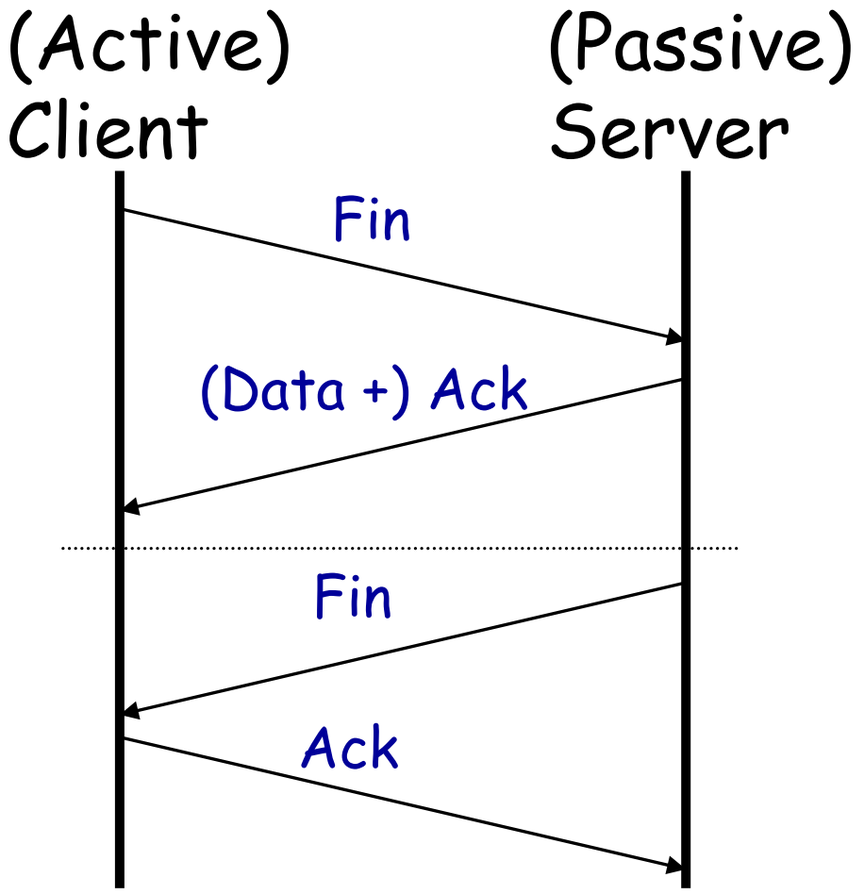
TCP Header


Bits
0
8
16
31
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
Code
Window
Checksum
Urgent Pointer
Options
Padding
Data

TCP: 3-way handshake



Connection Setup
3-way handshake



Connection Close/Teardown
2 x 2-way handshake

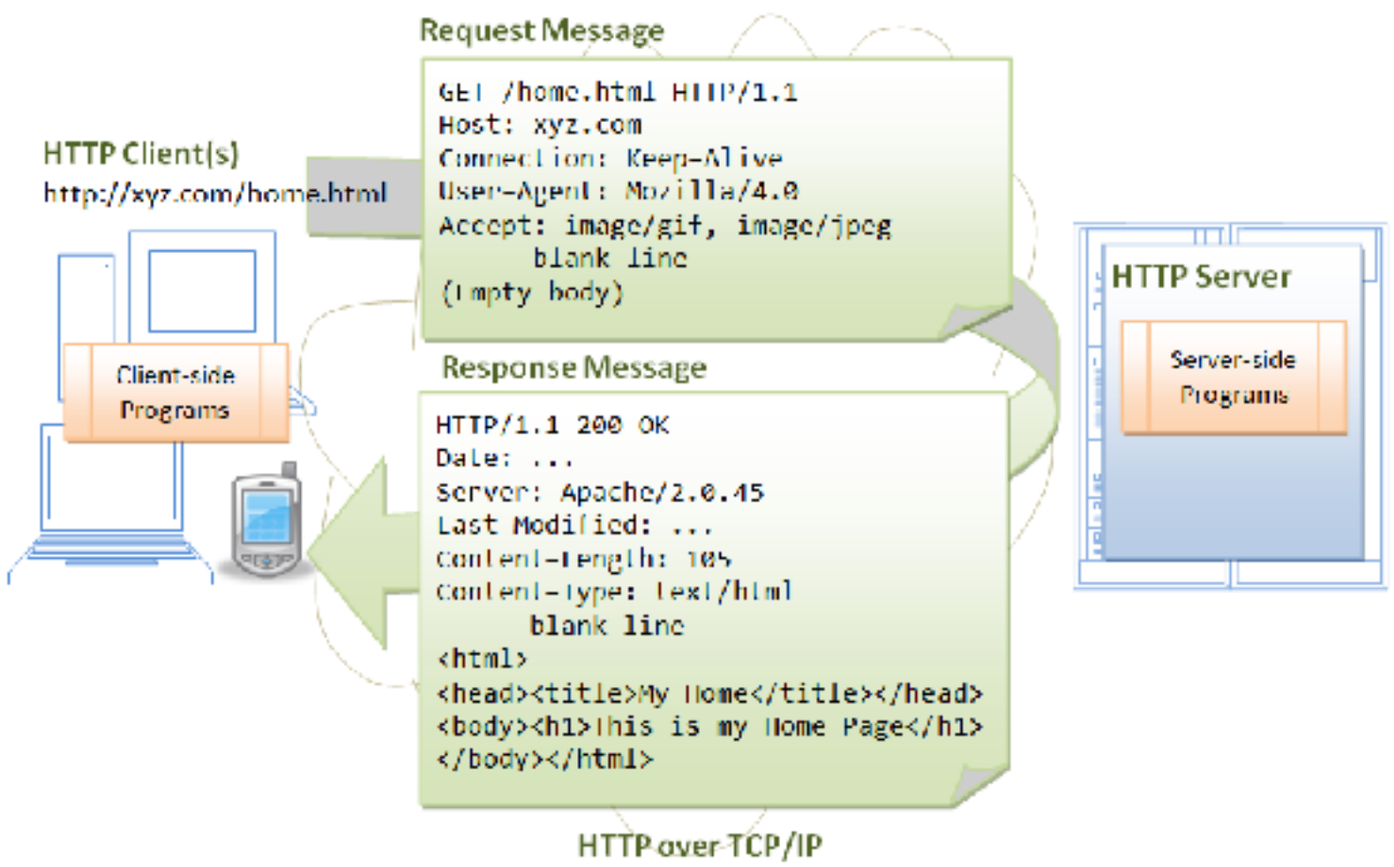
UDP

- UDP is a connectionless datagram service.
  - There is no connection establishment: packets may show up at any time.
- UDP is unreliable:
  - No acknowledgements to indicate delivery of data.
  - Checksums cover the header, and only optionally cover the data.
  - Contains no mechanism to detect missing or mis-sequenced packets.
  - No mechanism for automatic retransmission.
  - No mechanism for flow control, and so can over-run the receiver.

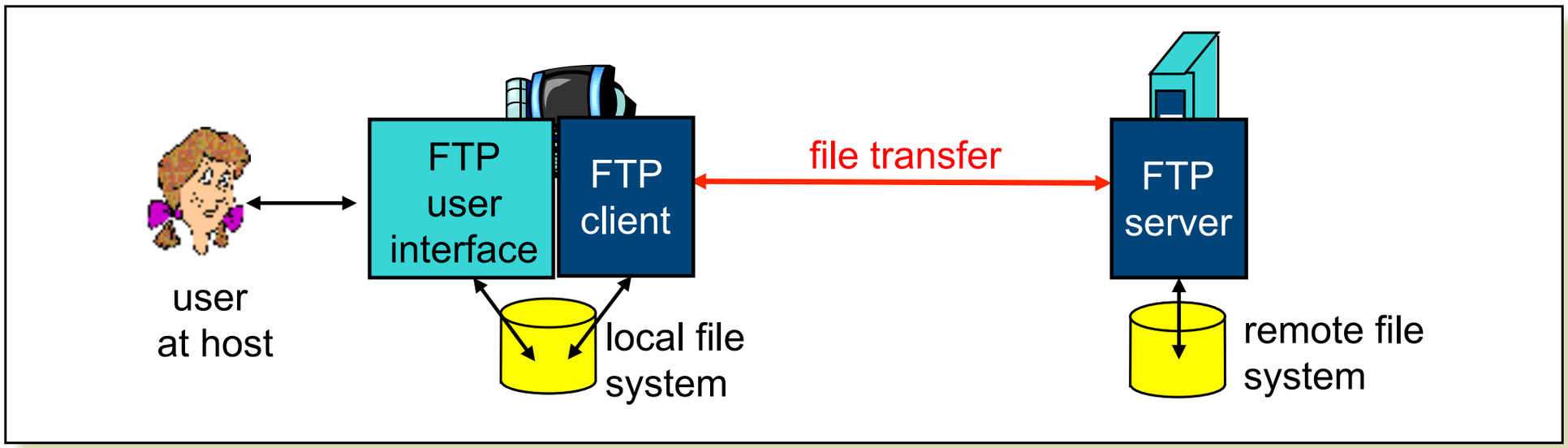
HTTP

- Definition

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. - ***from Wikipedia***



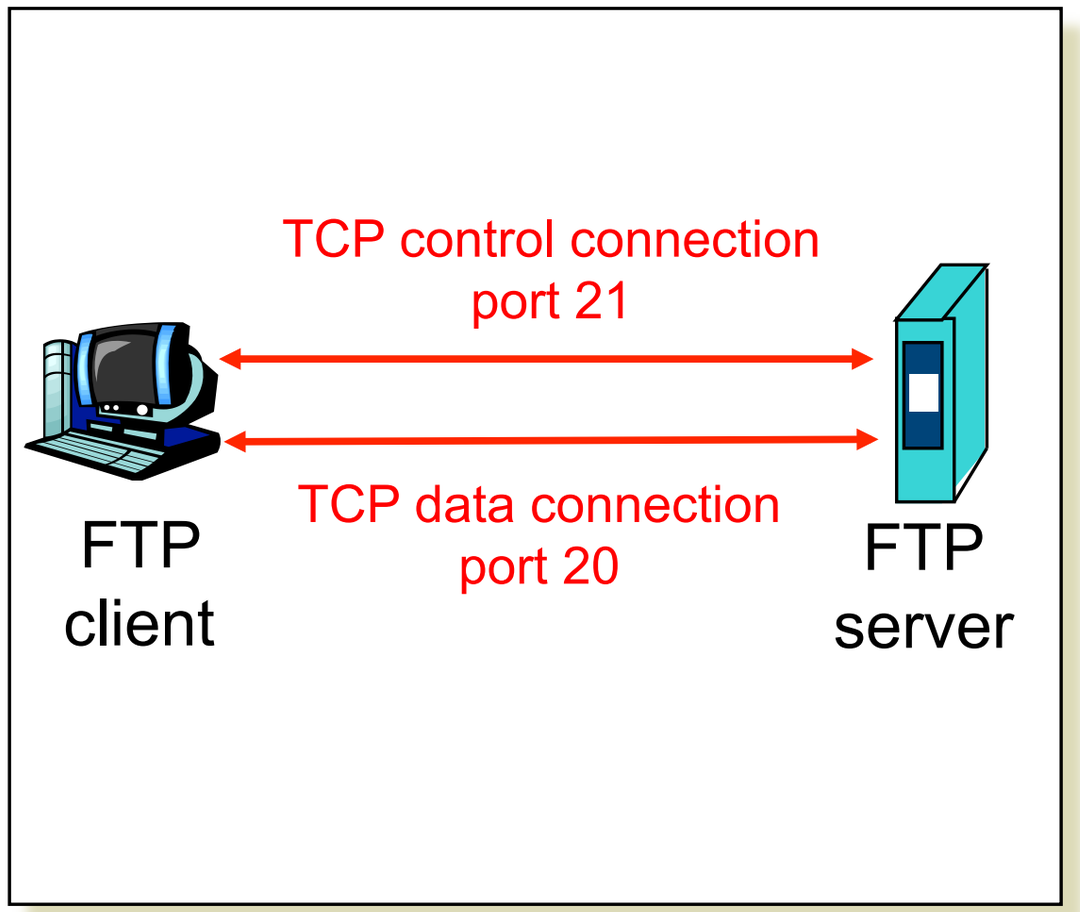
FTP



- Transfer file to/from remote host
- Client/server model
  - *Client:* side that initiates transfer (either to/from remote)
  - *Server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP

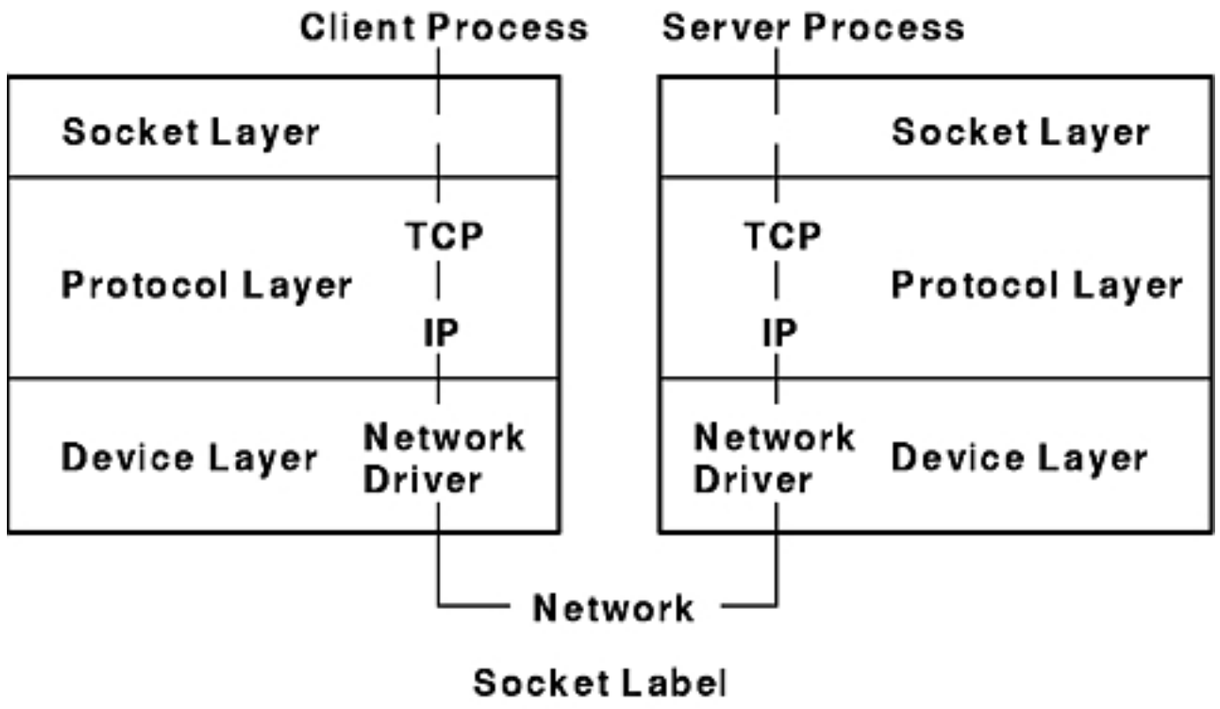
- Ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- Two parallel TCP connections opened:
  - Control: exchange commands, responses between client, server.
    "out of band control"
  - Data: file data to/from server
- Ftp server maintains "state": current directory, earlier authentication



SOCKET

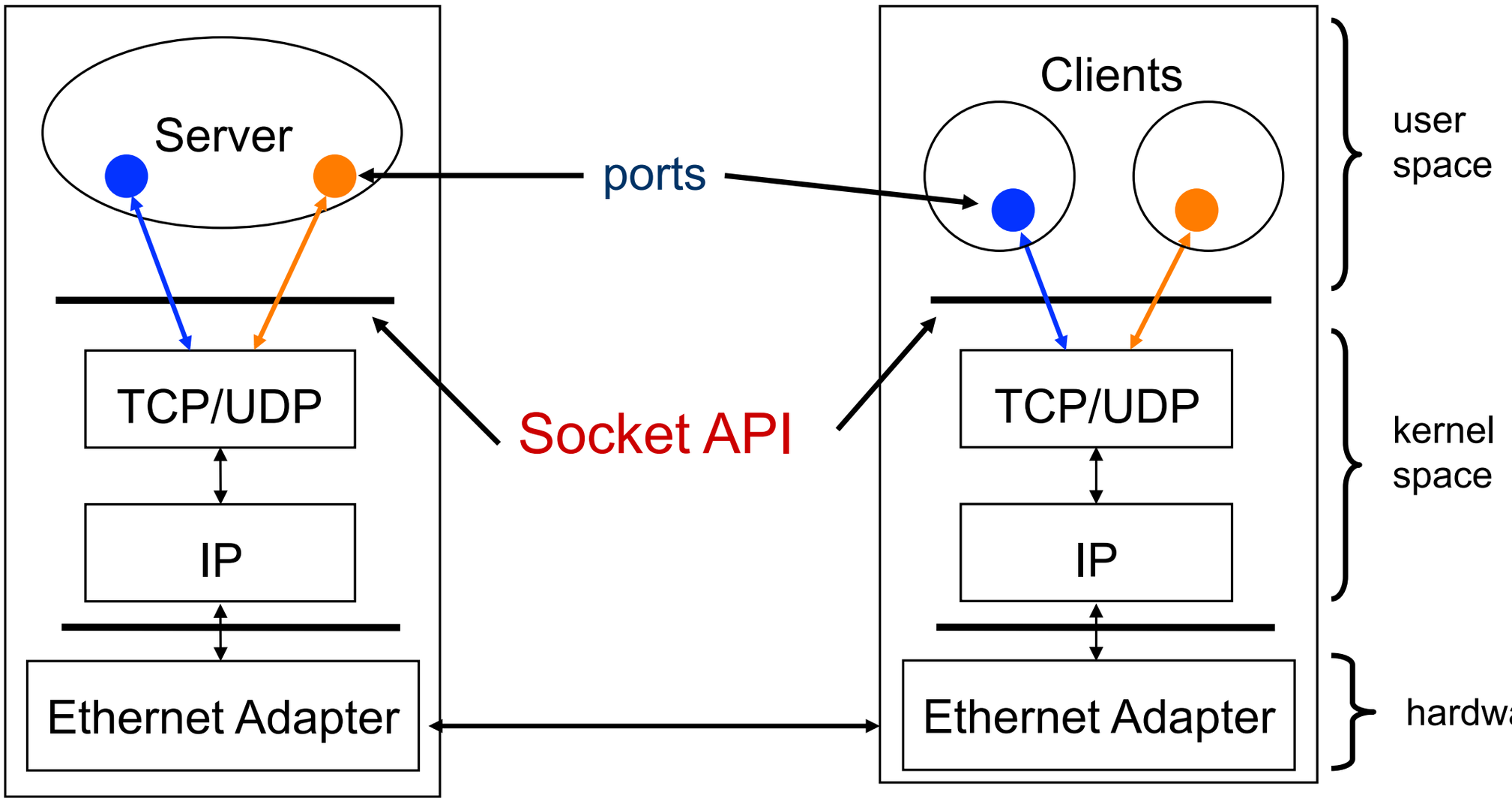
- Definition

A network socket is an endpoint of an inter-process communication flow across a computer network… from ***wikipedia***



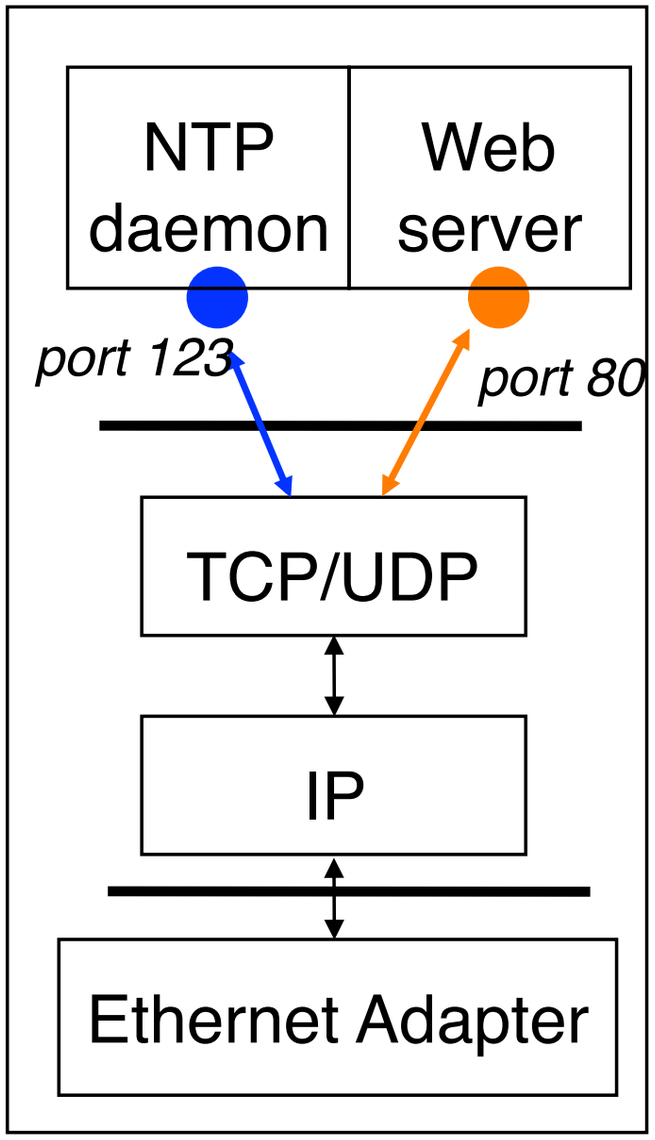
Client - Server

Server and Client exchange messages over the network through a common Socket API



Network Port

- Port numbers are used to identify "entities" on a host
- Port numbers can be
  - Well-known (port 0-1023)
  - Dynamic or private (port 1024-65535)
- Servers/daemons usually use well-known ports
  - Any client can identify the server/service
  - HTTP = 80, FTP = 21, Telnet = 23, ...
  - /etc/service defines well-known ports
- Clients usually use dynamic ports
  - Assigned by the kernel at run time



Code: in_addr, sockaddr_in

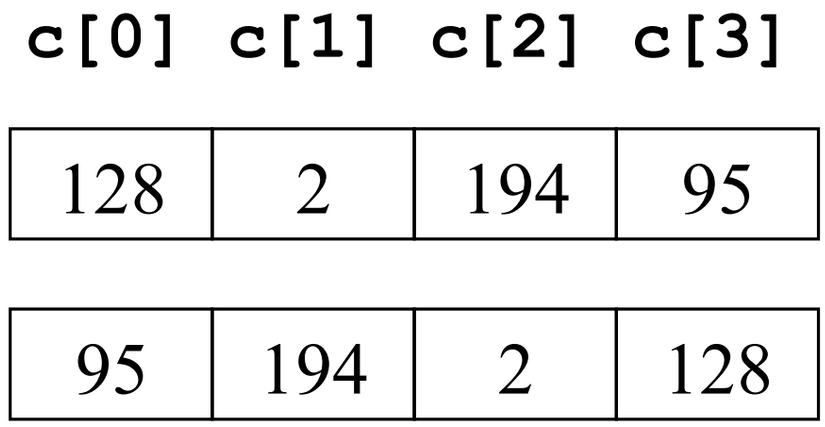
```
#include <netinet/in.h>

/* Internet address structure */
struct in_addr {
        u_long s_addr;          /* 32-bit IPv4 address */
};                              /* network byte ordered */

/* Socket address, Internet style. */
struct sockaddr_in {
      u_char  sin_family;       /* Address Family */
      u_short sin_port;         /* UDP or TCP Port# */
                                /* network byte ordered */
      struct in_addr sin_addr;        /* Internet Address */
      char    sin_zero[8];      /* unused */
};
```

- sin_family = AF_INET selects Internet address family

Byte Ordering

```
union {
   u_int32_t addr;  /* 4 bytes address */
   char c[4];
} un;
/* 128.2.194.95 */
un.addr = 0x8002c25f;
/* c[0] = ? */
```

- Big Endian
  - Sun Solaris, PowerPC, ...
- Little Endian
  - i386, alpha, ...
- Network byte order = Big Endian



How to Convert

- Converts between **host byte order** and **network byte order**
  - ‘h’ = host byte order
  - ‘n’ = network byte order
  - ‘l’ = long (4 bytes), converts IP addresses
  - ‘s’ = short (2 bytes), converts port numbers

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int
hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int
netshort);
```

Socket Example

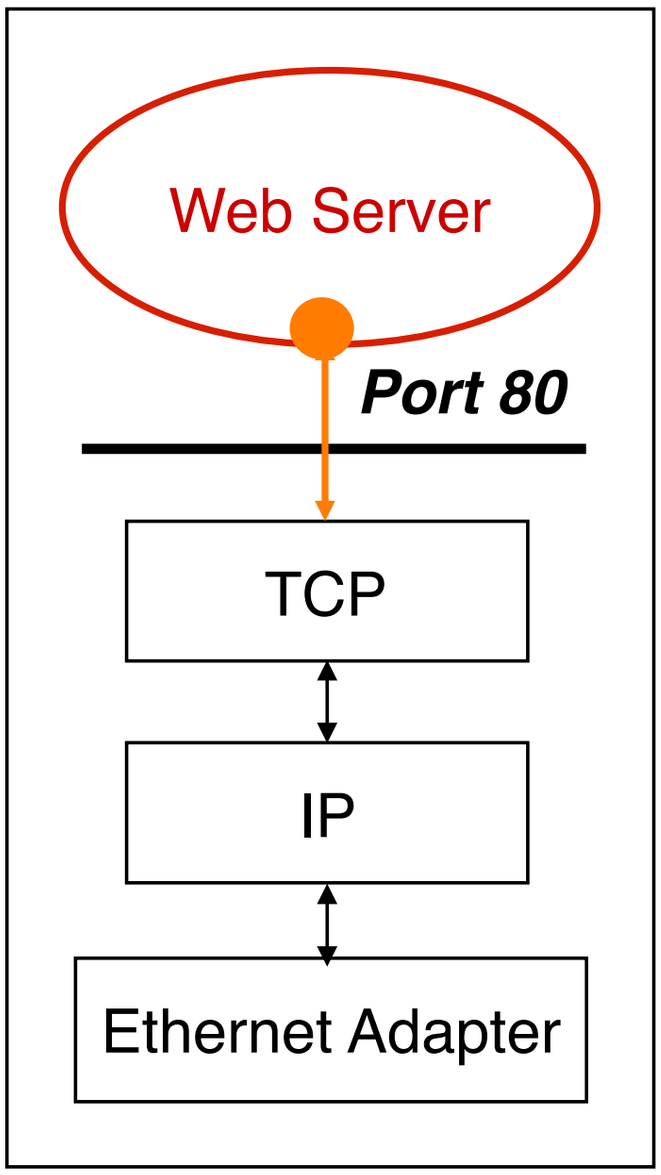
- A socket is a file descriptor that lets an application read/write data from/to the network

```
int fd;          /* socket descriptor */
if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(1);
}
```

- ***socket*** returns an integer (socket descriptor)
  - fd < 0 indicates that an error occurred
  - socket descriptors are similar to file descriptors

- AF_INET: associates a socket with the Internet protocol family
- SOCK_STREAM: selects the TCP protocol
- SOCK_DGRAM: selects the UDP protocol

TCP server example



- For example: web server
- **What does a *web server* need to do so that a *web client* can connect to it?**

socket( )

- Since web traffic uses TCP, the web server must create a socket of type SOCK_STREAM

```
int fd;             /* socket descriptor */

if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("socket");
      exit(1);
}
```

- ***socket*** returns an integer (**socket descriptor**)
  - ***fd*** < 0 indicates that an error occurred
- **AF_INET** associates a socket with the Internet protocol family
- **SOCK_STREAM selects the TCP protocol**

bind( )

- A ***socket*** can be bound to a ***port***

```
int fd;                                  /* socket descriptor */
struct sockaddr_in srv;     /* used by bind() */

/* create the socket */

srv.sin_family = AF_INET; /* use the Internet addr family */

srv.sin_port = htons(80); /* bind socket ‘fd’ to port 80*/

/* bind: a client may connect to any of my addresses */
srv.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(fd, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
        perror("bind"); exit(1);
}
```

- **Still not quite ready to communicate with a client...**

listen( )

- ***listen*** indicates that the server will accept a connection

```
int fd;                          /* socket descriptor */
struct sockaddr_in srv;          /* used by bind() */

/* 1) create the socket */
/* 2) bind the socket to a port */

if(listen(fd, 5) < 0) {
        perror("listen");
        exit(1);
}
```

- **Still not quite ready to communicate with a client...**

accept( )

- ***accept*** blocks waiting for a connection

```
int fd;                                  /* socket descriptor */
struct sockaddr_in srv;                 /* used by bind() */
struct sockaddr_in cli;                 /* used by accept() */
int newfd;                              /* returned by accept() */
int cli_len = sizeof(cli); /* used by accept() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */

newfd = accept(fd, (struct sockaddr*) &cli, &cli_len);
if(newfd < 0) {
        perror("accept");    exit(1);
}
```

- ***accept*** returns a new socket (***newfd***) with the same properties as the original socket (***fd***)
  - ***newfd*** < 0 indicates that an error occurred

accept( ) more

```
struct sockaddr_in cli;          /* used by accept() */
int newfd;                       /* returned by accept() */
int cli_len = sizeof(cli);       /* used by accept() */

newfd = accept(fd, (struct sockaddr*) &cli, &cli_len);
if(newfd < 0) {
      perror("accept");
      exit(1);
}
```

- How does the server know which client it is?
  - **cli.sin_addr.s_addr** contains the client's ***IP address***
  - **cli.sin_port** contains the client's ***port number***
- Now the server can exchange data with the client by using ***read*** and ***write*** on the descriptor ***newfd***.
- Why does ***accept*** need to return a new descriptor?

read( )

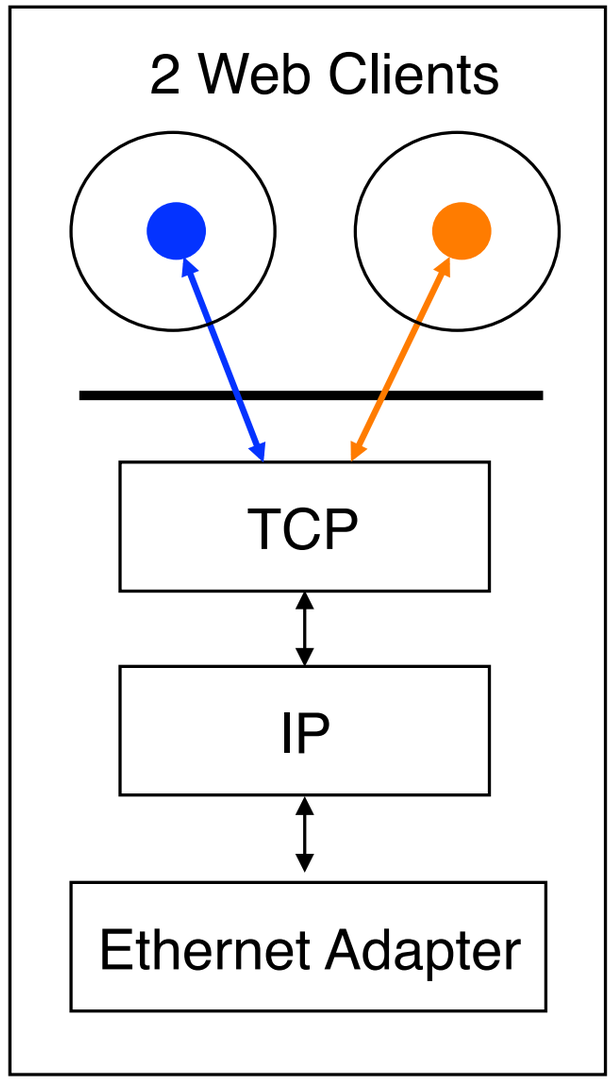
- ***read*** can be used with a socket
- ***read*** **blocks waiting for data from the client but does not guarantee that sizeof(buf) is read**

```
int fd;                                  /* socket descriptor */
char buf[512];                           /* used by read() */
int nbytes;                             /* used by read() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */
/* 4) accept the incoming connection */

if((nbytes = read(newfd, buf, sizeof(buf))) < 0) {
        perror("read"); exit(1);
}
```

TCP client example

- For example: web client

- **How does a *web client* connect to a *web server*?**



How to find a server

- IP Addresses are commonly written as strings (“128.2.35.50”), but programs deal with IP addresses as integers.

Converting strings to numerical address:

```
struct sockaddr_in srv;

srv.sin_addr.s_addr = inet_addr(“128.2.35.50”);
if(srv.sin_addr.s_addr == (in_addr_t) -1) {
      fprintf(stderr, "inet_addr failed!\n"); exit(1);
}
```

Converting a numerical address to a string:

```
struct sockaddr_in srv;
char *t = inet_ntoa(srv.sin_addr);
if(t == 0) {
      fprintf(stderr, “inet_ntoa failed!\n”); exit(1);
}
```

connect( )

- ***connect*** allows a client to connect to a server...

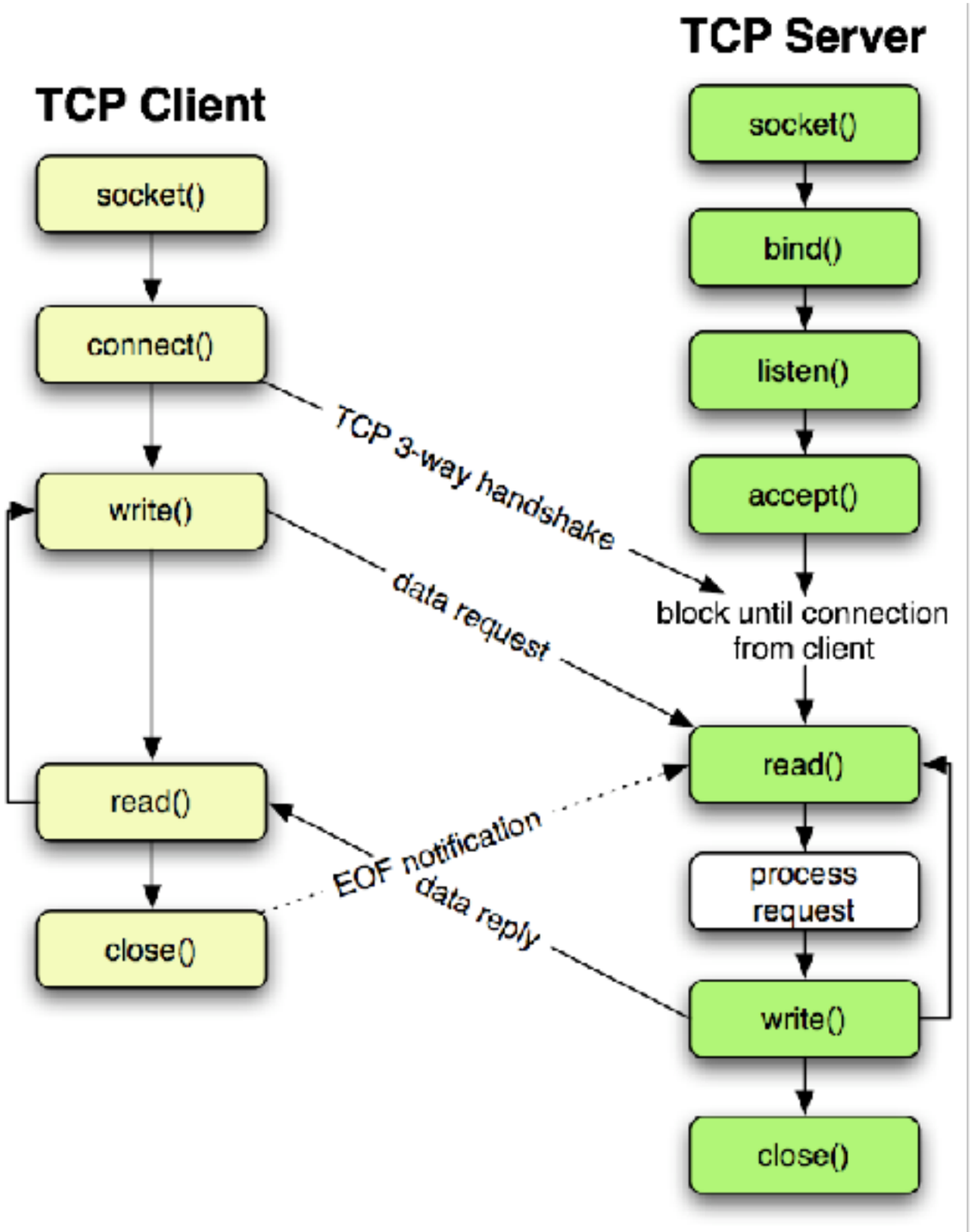
```
int fd;                                 /* socket descriptor */
struct sockaddr_in srv;                 /* used by connect() */

/* create the socket */

/* connect: use the Internet address family */
srv.sin_family = AF_INET;

/* connect: socket 'fd' to port 80 */
srv.sin_port = htons(80);

/* connect: connect to IP Address "128.2.35.50" */
srv.sin_addr.s_addr = inet_addr("128.2.35.50");

if(connect(fd, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
        perror("connect"); exit(1);
}
```

write( )

- ***write*** can be used with a socket

```
int fd;                                 /* socket descriptor */
struct sockaddr_in srv;                 /* used by connect() */
char buf[512];                          /* used by write() */
int nbytes;                             /* used by write() */

/* 1) create the socket */
/* 2) connect() to the server */

/* Example: A client could “write” a request to a server */
if((nbytes = write(fd, buf, sizeof(buf))) < 0) {
        perror(“write”);
        exit(1);
}
```

Network Program with SOCKET (TCP case)



Class Summary

- Network
  - some basic things