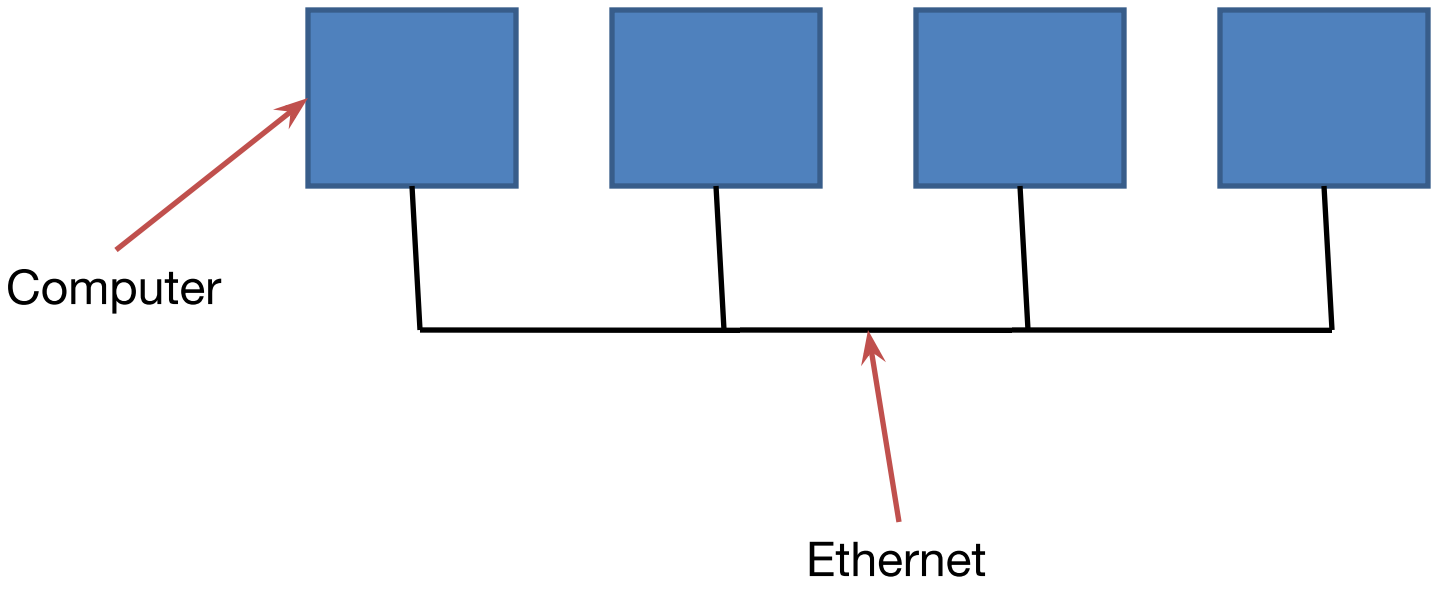
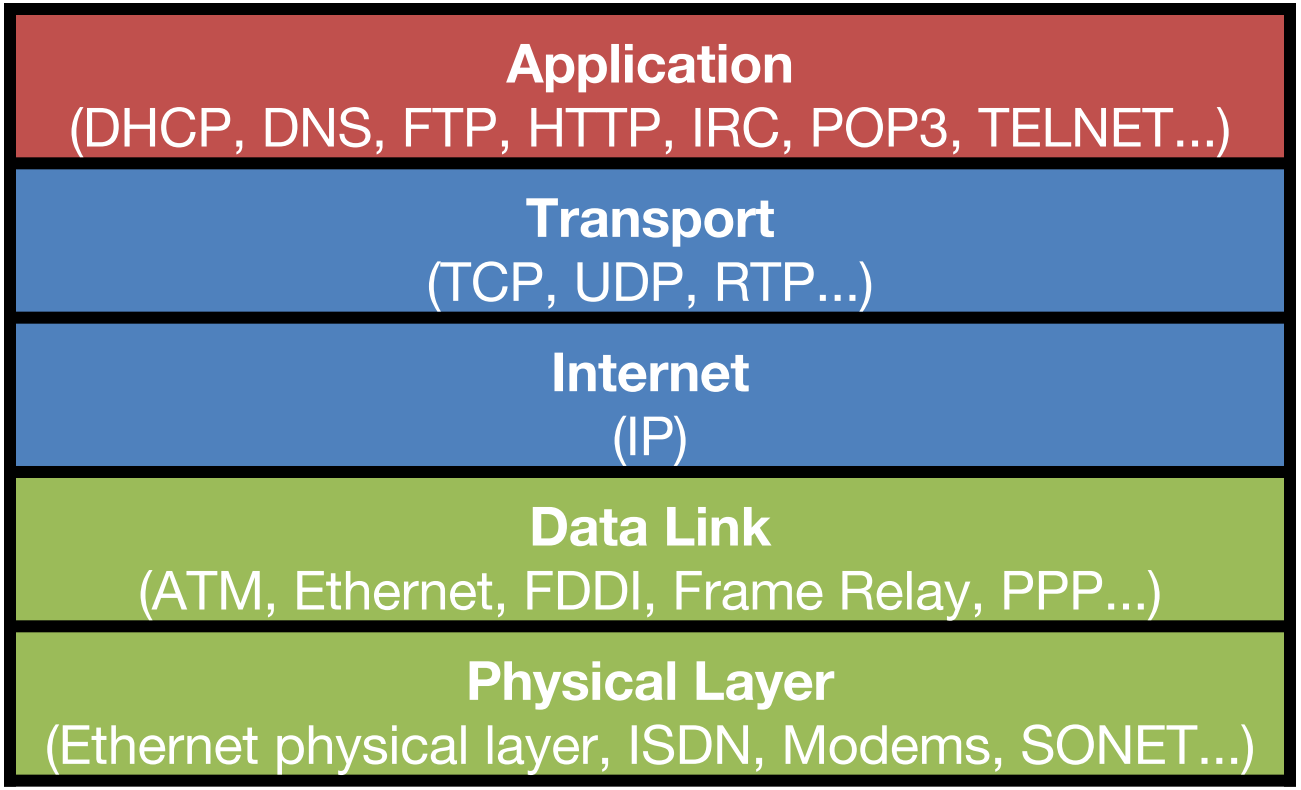
Network Communication

CS449 Spring 2016

Network



Internet Layer Model



Internet Protocol (IP)

- **Protocol** – A standard procedure for regulating data transmission between computers.
- **IP Addresses** – 32-bit (v4) or 128 (v6) number denoting an destination or source
- **Port** – a number representing a particular listener on a machine

Transmission Control Protocol (TCP)

- **Connection-oriented** – Make a circuit with a remote machine

- Guarantees data arrives, and in-order

User Datagram Protocol (UDP)

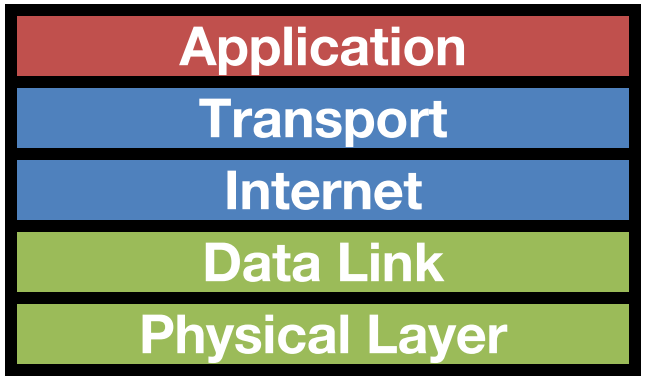
- **Datagram** – (play on telegram) A message with no acknowledgement
- **Connectionless** – Send and forget
- No guarantee data arrives or is in the order sent

IP Packet

	Bits 0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Type of Service	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options				
192-	Data				

Packets


Ethernet Header
IP Header
Protocol Header
Application Header
Data


Application
Transport
Internet
Data Link
Physical Layer

NETWORK PROGRAMMING

Processes communicating

Process: program running within a host.

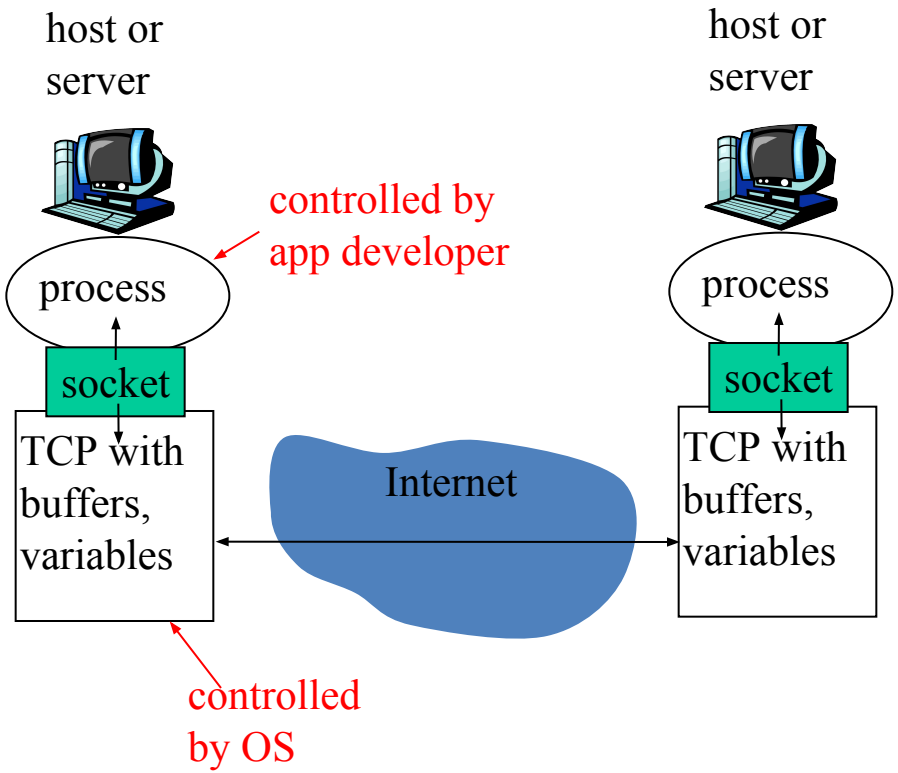
- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that initiates communication
Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from a socket
- socket is a software communication channel
  - sending process sends into socket
  - sending process relies on transport infrastructure on other side of socket to deliver message to socket at receiving process



API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Addressing processes

- For a process to receive messages, it must have an identifier
- A host has a unique 32-bit IP address

Q: does the IP address of the host on which the process runs suffice for identifying the process?

Answer: No, many processes can be running on same host

- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

App-layer protocol defines

- Types of messages exchanged,
  - eg, request & response
- Message Syntax
  - What fields in messages & how fields are delineated
- Message Semantics
  - Meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- eg, Skype

BERKELEY SOCKETS

Socket

- UNIX treats everything as a file
  - File Descriptor
  - read()/ write()
- Treat network as a file called a socket
- Berkeley sockets are de facto standard API

socket()

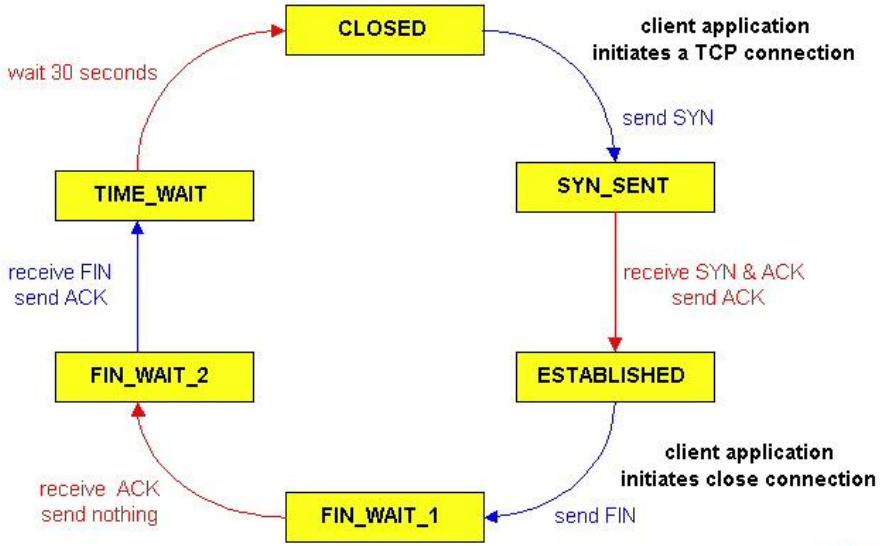
- Creates a Socket Descriptor

```
#include <sys/types.h>
#include <sys/socket.h>

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

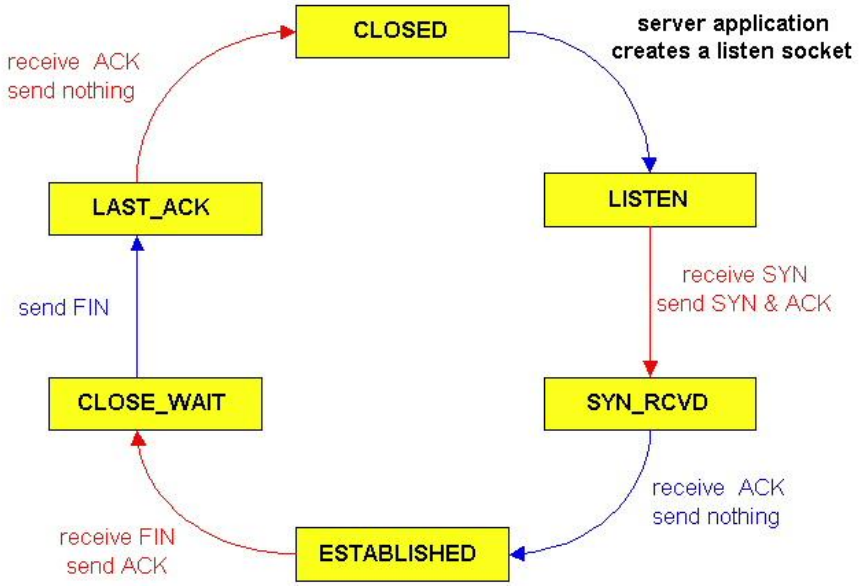
Parameter	Values
domain	•AF_INET for IPv4 •AF_INET6 for IPv6
type	•SOCK_STREAM •SOCK_DGRAM •SOCK_SEQPACKET •SOCK_RAW
protocol	IPPROTO_IP (defined as 0)

TCP Connection Management



TCP client lifecycle

TCP server lifecycle



SERVER STUFF

bind()

- Attach a socket to a port

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


struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
bind(fd, (struct sockaddr *)addr, sizeof(addr));
```

- Returns 0 on success, -1 on error

listen()

- Set up a listening socket

```
int listen(int sockfd, int backlog);
```

- Backlog – how many pending connections are allowed to wait (OS max usually around 20, set to lower, ~10)
- Returns 0 on success, -1 on error

accept()

- Block and wait for connection to occur

```
int accept(int sockfd, struct sockaddr
  *cliaddr, socklen_t *addrlen);
```

- Returns file descriptor for the new socket, or -1 on error
- Will return information about the client through the structure (can be NULL)

send() and recv()

```
int send(int sockfd, const void *msg, int
  len, int flags);
```

- By default blocks until `len` bytes are sent
- Returns number of bytes sent, or -1 on error

```
int recv(int sockfd, void *buf, int len,
  unsigned int flags);
```

- Returns any amount of data available up to `len` bytes
- Returns number of bytes received, or 0 if peer closed connection, or -1 on error

errno and perror()

- Whenever operations on descriptors fail, the `errno` global variable is set with error type
- Especially important to check for error for socket operations because it involves remote machine
- `void perror(const char *s)`
  - prints descriptive message to stdout for `errno`, prefixed by `s` (if not NULL)

```
if(bind(socket, &addr, sizeof(addr)) < 0
  {
  perror(NULL);
}
```

CLIENT STUFF

connect()

- Connect to a server located at some address and port

```
int connect(int sockfd, struct sockaddr *serv_addr,
   int addrlen);


struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr =net_addr("127.0.0.1");
connect(fd, (struct sockaddr *)&addr, sizeof(addr));
```

send() and recv()

```
int send(int sockfd, const void *msg, int
  len, int flags);

int recv(int sockfd, void *buf, int len,
  unsigned int flags);
```

CONNECTIONLESS COMMUNICATION

Datagram Send and Receive

```
int sendto(int sockfd, const void *msg, int
  len, unsigned int flags, const struct
  sockaddr *to, socklen_t tolen);

int recvfrom(int sockfd, void *buf, int len,
  unsigned int flags, struct sockaddr *from,
  int *fromlen);
```

DNS

- Domain Name Server
- Resolve a name to an IP address:

http://www.cs.pitt.edu -> 130.49.220.23

DNS

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);


struct hostent {
      char    *h_name;         /* official name of host */
      char    **h_aliases;     /* alias list */
      int     h_addrtype;      /* host address type */
      int     h_length;        /* length of address */
      char    **h_addr_list;   /* list of addresses */
}
#define h_addr  h_addr_list[0]  /* for backward
   compatibility */
```