Network Communication

Processes communicating

Process: program running within a host.

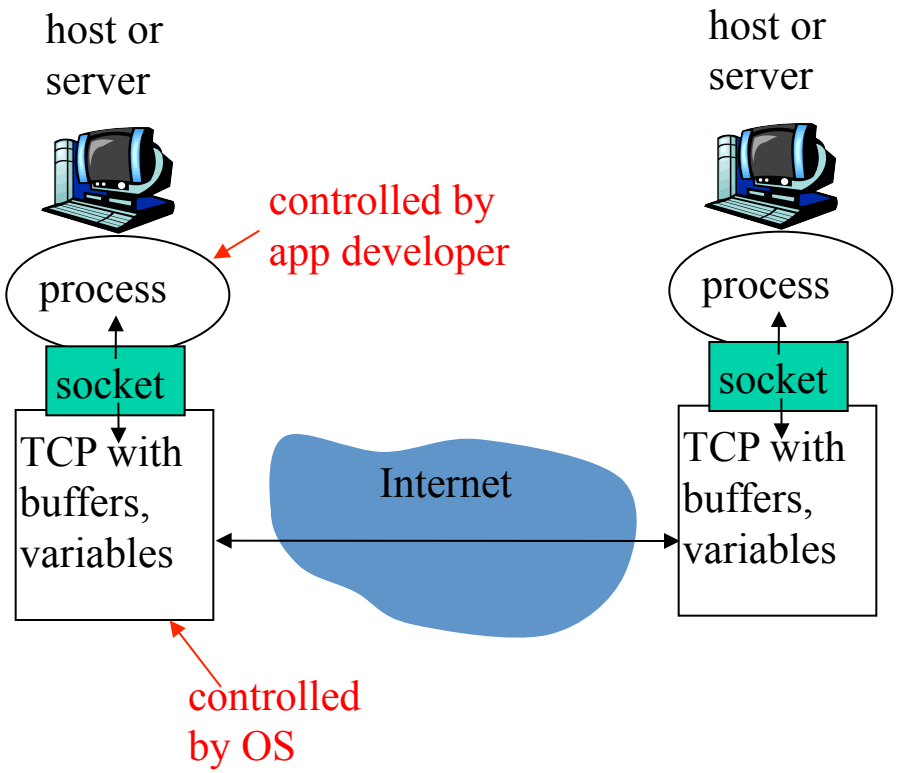
- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

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

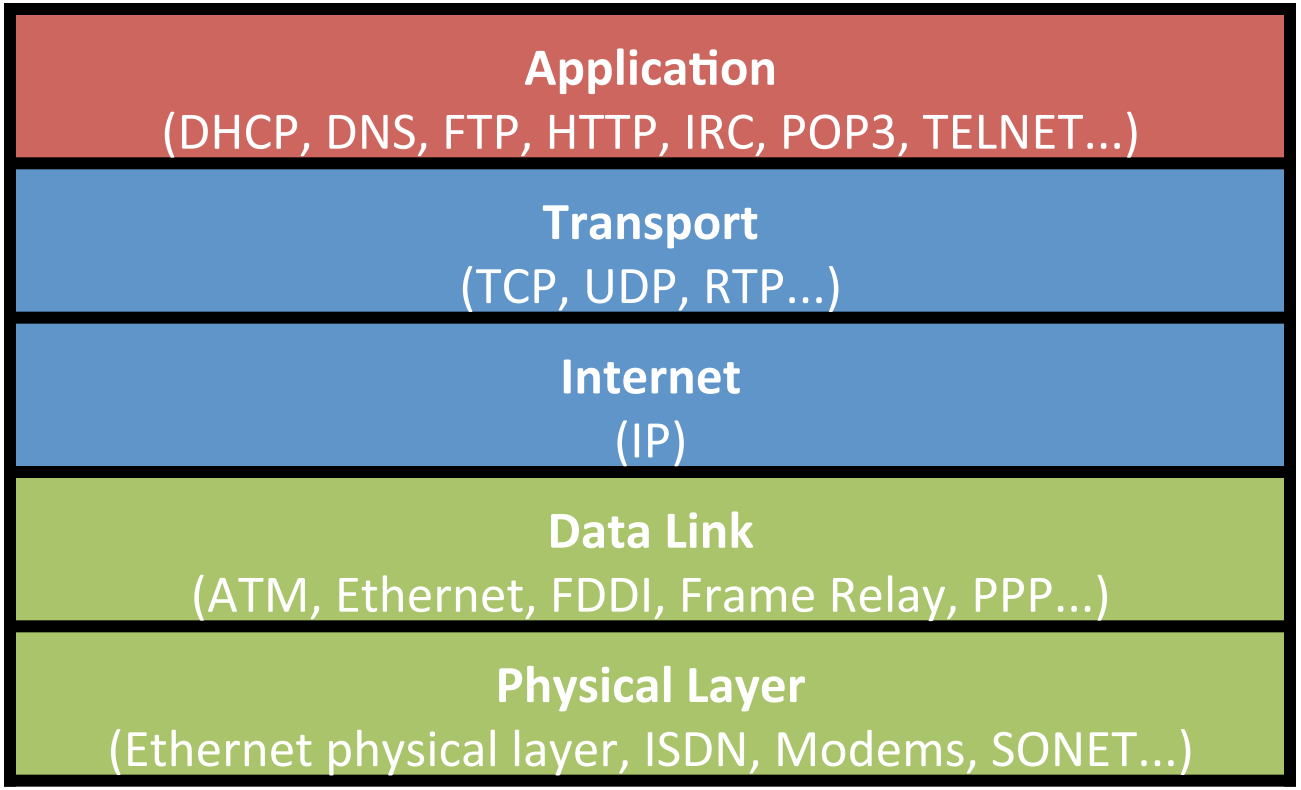
- For a process to receive messages, it must have an identifier
- A host has a unique 32-bit IP address

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

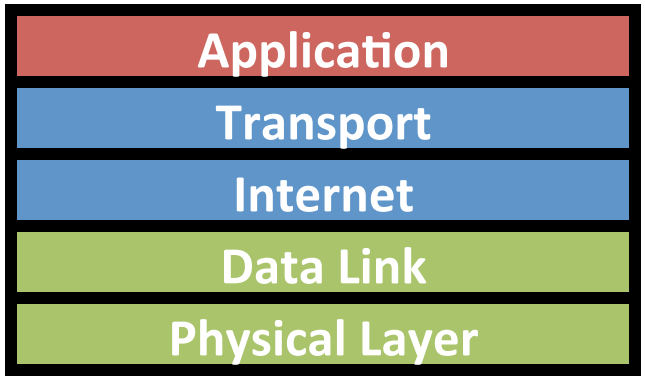
- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- More on this later

Internet Layer Model



Packets


Ethernet Header
IP Header
Protocol Header
Application Header
Data


Application
Transport
Internet
Data Link
Physical Layer

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

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

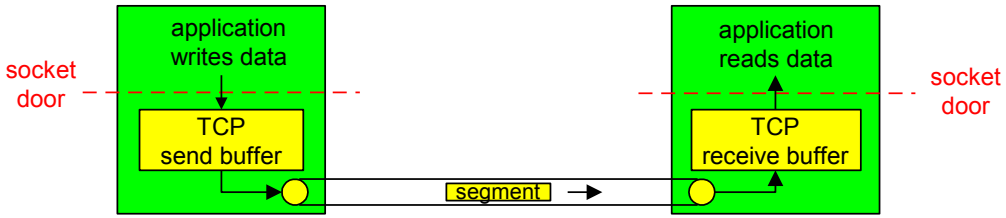
Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired
- Higher level functionality can be added by applications

TCP: Overview

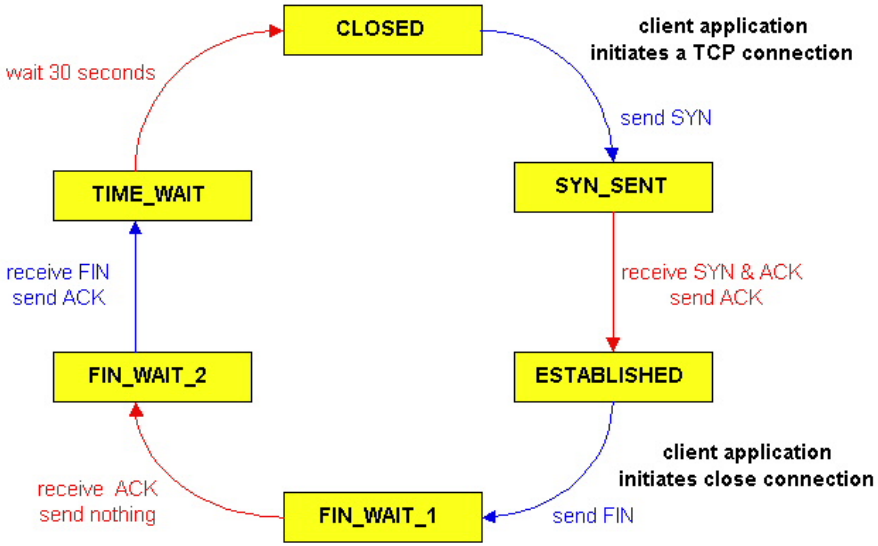
RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



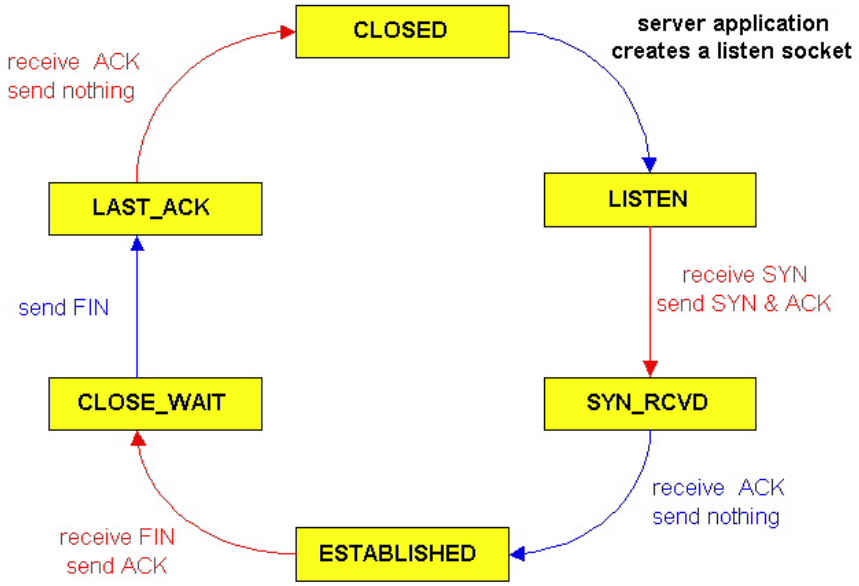
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP Connection Management



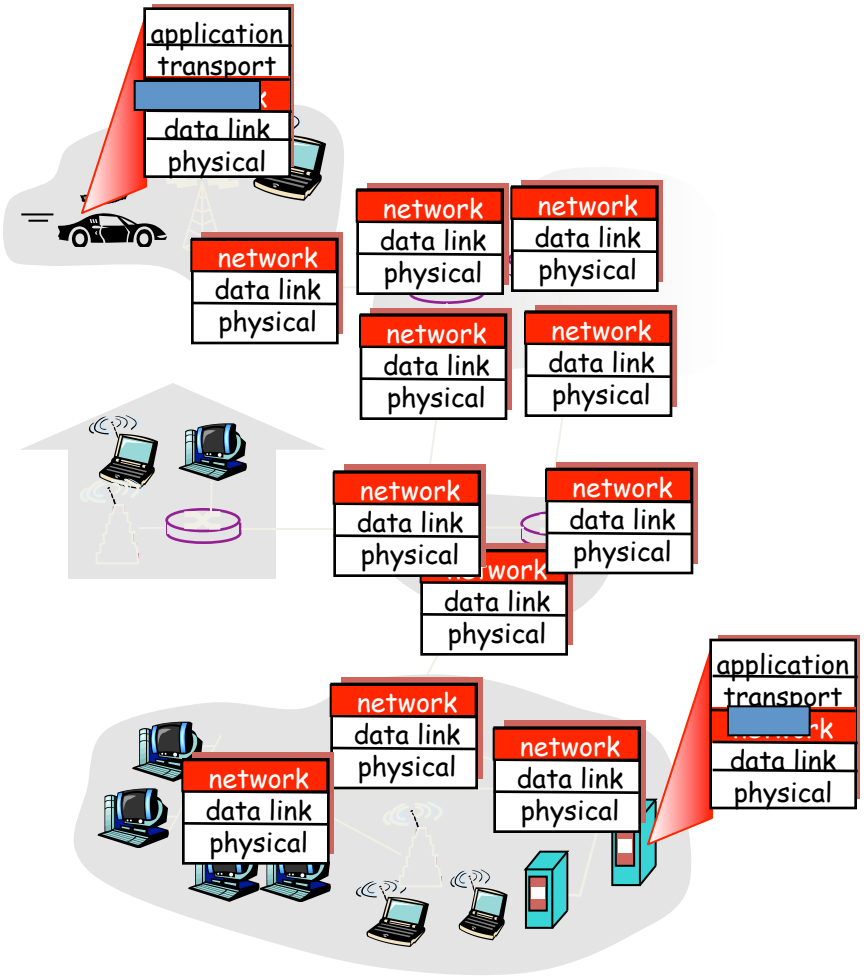
TCP client lifecycle

TCP server lifecycle

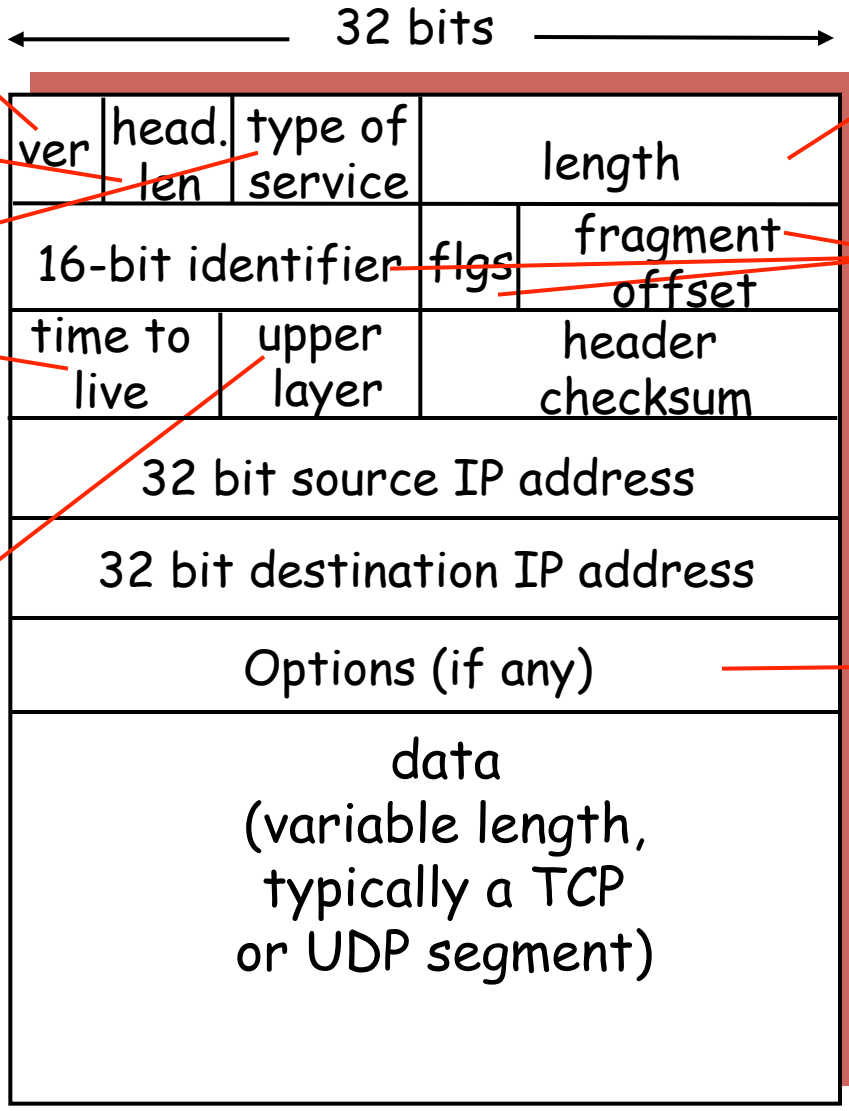


IP Network layer

- Transport segment from sending to receiving host
- On sending side encapsulates segments into datagrams
- On recving side, delivers segments to transport layer
- Network layer protocols in *every* host, router
- Router examines header fields in all IP datagrams passing through it



IP datagram format



IP protocol version number

header length (bytes)

"type" of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to

total datagram length (bytes)

for fragmentation/ reassembly

E.g. timestamp, record route taken, specify list of routers to visit.

how much overhead with TCP?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

SERVER STUFF

socket()

- Creates a Socket Descriptor

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

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

Parameter	Values
domain	•PF_INET for IPv4 •PF_INET6 for IPv6
type	•SOCK_STREAM •SOCK_DGRAM •SOCK_SEQPACKET •SOCK_RAW
protocol	IPPROTO_IP (defined as 0)

bind()

- Attach a socket to a port

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

```
memset(&my_addr, 0, sizeof(struct sockaddr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(PORT);
my_addr.sin_addr.s_addr =net_addr(“127.0.0.1");
```

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)

accept()

- Block and wait for connection to occur

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

- 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);

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

CLIENT STUFF

socket()

- Creates a Socket Descriptor

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

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

Parameter	Values
domain	•PF_INET for IPv4 •PF_INET6 for IPv6
type	•SOCK_STREAM •SOCK_DGRAM •SOCK_SEQPACKET •SOCK_RAW
protocol	IPPROTO_IP (defined as 0)

connect()

- Connect to a server located at some address and port

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

```
memset(&my_addr, 0, sizeof(struct sockaddr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(PORT);
my_addr.sin_addr.s_addr = inet_addr(“127.0.0.1");
```

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 */
```