



---

# TCP Programming and TCP/IP Protocol Issues

Lecture 36

Klara Nahrstedt

# CS241 Administrative

- Read Stallings Chapter 13, R&R 18.1-18.3, 18.7, 18.8
- LMP3 (Part I) Due April 24 – extension until 9am Tuesday
- LMP3 (Part II) Due April 30
- Last Regular Quiz will be on Friday, April 27 on Networking
- Homework 2 posted today – Due Wednesday, May 2, 4pm
- Monday, April 30
  - We will have LMP3 Quiz
  - We will have guest lecture – speaker from Mathematica talking about challenges of Mathematica Development on Different OS Platforms (UNIX, MAC, Linux, Windows).
- Wednesday, May 2 – In Class Review Session for Final Exam

# Socket-Based APIs

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

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

```
int bind (int s, const struct sockaddr *address,  
size_t address_len);
```

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

```
int accept (int s, struct sockaddr *restrict address,  
int *restrict address_len);
```

# Socket Creation in C: socket

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

s: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain

e.g., AF\_INET typically used (Internet domain)

type: communication type

SOCK\_STREAM: reliable, 2-way, connection-based service

SOCK\_DGRAM: unreliable, connectionless,

other values: need root permission, rarely used, or obsolete

protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0 (IP)

**NOTE:** socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

# Socket Structure

The format of the address `struct sockaddr` is determined by the address family (domain).

For `AF_INET` it is a `struct sockaddr_in`

Socket structure is defined in `netinet/in.h`

Socket structure has at least the following members in network byte order:

```
sa_family_t sin_family;  
in_port_t sin_port; /* Host Port Number */  
struct in_addr sin_addr; /* IP Address*/
```

# Bind Function

```
int bind(int s, const struct sockaddr *address, size_t  
address_len);
```

## Bind function

Associates the handle for a socket communication endpoint with a specific logical network connection.

Note: Internet domain protocols specify logical connection by a port number

**s** is the file descriptor returned by `socket( )`

**address** contains info about the family, port and machine

**address\_len** is the size of the structure used for the address

# Connection Setup (SOCK\_STREAM) - TCP

Recall: no connection setup for SOCK\_DGRAM

A connection occurs between two kinds of participants

- passive: waits for an active participant to request connection

- active: initiates connection request to passive side

Once connection is established, passive and active participants are “similar”

- both can send & receive data

- either can terminate the connection

Analogy?

- telephone

# Connection setup cont'd

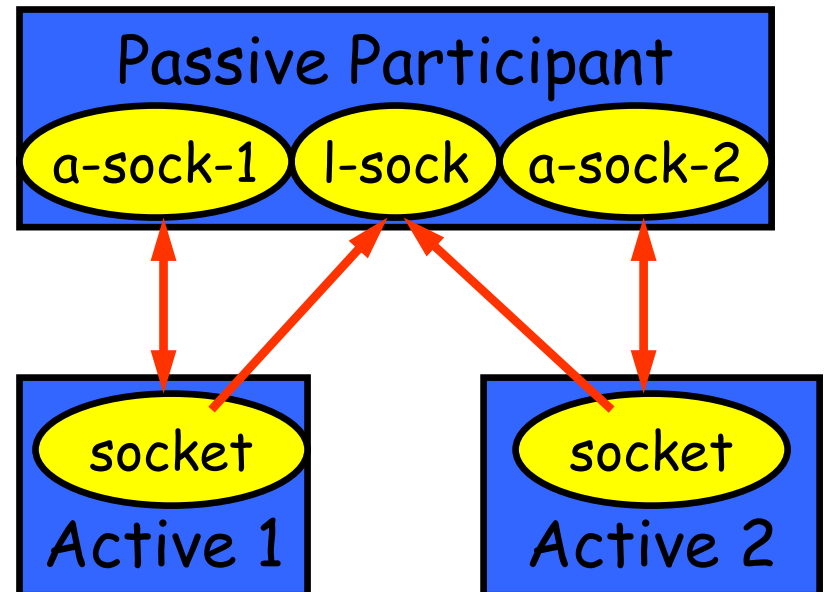
- Passive participant

- step 1: **listen** (for incoming requests)
- step 3: **accept** (a request)
- step 4: data transfer

- Active participant

- step 2: request & establish **connection**

- **The accepted connection is on a new socket**
- The old socket continues to listen for other active participants



# Listen function

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

0 if listening, -1 if error

**sock**: integer, socket descriptor

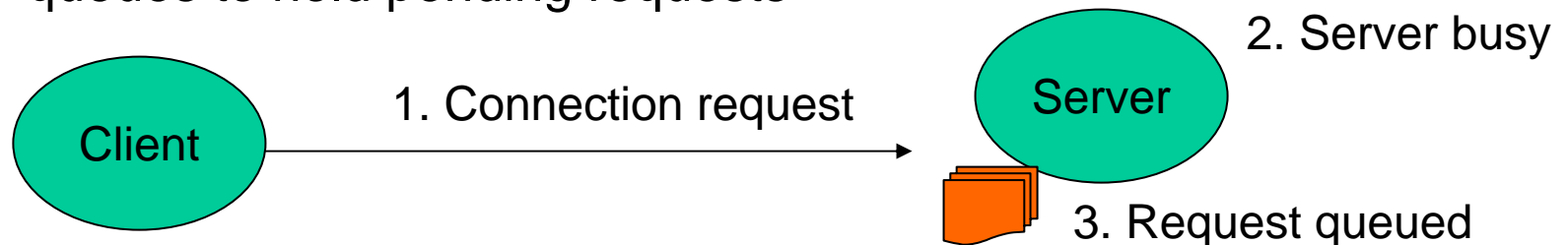
**backlog**: integer, # of active participants that can “wait” for a connection

Socket(): creates a communication point

Bind(): associates this end-point with a particular address

Listen function (non-blocking)

Causes the underlying system network infrastructure to allocate queues to hold pending requests



Reason: if client requests connection, server may be busy, so the host network subsystem must queue the client connection request until the server is ready to accept its request

# Accept Function

```
#include <sys/socket.h>  
int accept (int socket, struct sockaddr *restrict address,  
socklen_t *restrict address_len);
```

## A **blocking** call

Extracts the first connection on the queue of pending connections  
creates a new socket with the same socket type protocol and address  
family as the specified socket  
allocates a new file descriptor for that socket

Returns **communication file descriptor** if successful or -1 otherwise.

Server fills the **second parameter with information about the client**

You fill in the size of the buffer used for the second parameter and on  
return the **third parameter contains the actual size** needed to  
contain this information.

connect call `/* Client side */`

```
int status = connect(sock, &name, namelen);
```

`status`: 0 if successful connect, -1 otherwise

`sock`: integer, socket to be used in connection

`name`: struct sockaddr: address of passive participant

`namelen`: integer, sizeof(name)

connect is **blocking**

# Sending / Receiving Data

With a connection (SOCK\_STREAM):

```
int count = send(sock, &buf, len, flags);
```

`count`: # bytes transmitted (-1 if error)

`buf`: char[], buffer to be transmitted

`len`: integer, length of buffer (in bytes) to transmit

`flags`: integer, special options, usually just 0

```
int count = recv(sock, &buf, len, flags);
```

`count`: # bytes received (-1 if error)

`buf`: void[], stores received bytes

`len`: # bytes received

`flags`: integer, special options, usually just 0

Calls are **blocking** [returns only after data is sent (to socket buf) / received]

# close

When finished using a socket, the socket should be closed:

```
status = close(s);
```

status: 0 if successful, -1 if error

s: the file descriptor (socket being closed)

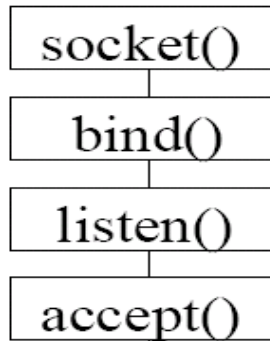
## Closing a socket

closes a connection (for SOCK\_STREAM)

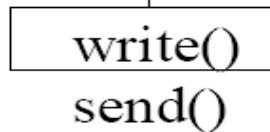
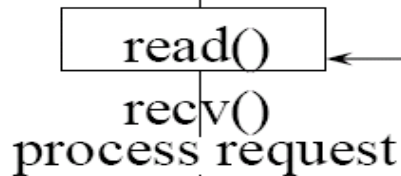
frees up the port used by the socket

# Typical TCP Server-Client

**Server**  
(connection-oriented protocol)

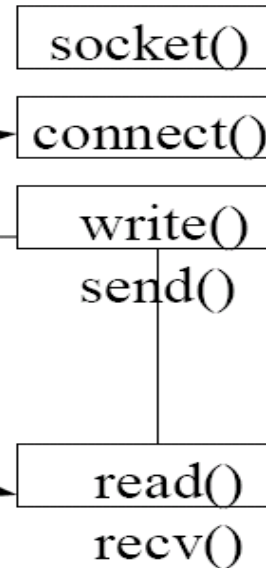


blocks until connection  
from client



**Socket system calls for  
connection-oriented  
protocol**

**Client**



connection establishment

data (request)

data (reply)

# Internet Protocol

## Physical Layer

Physical medium, optical cables, wireless medium to move data over

## Network Access Layer (MAC – Medium Access Control)

Exchange of data and Coordination among computers over local area network

using circuit switching or packet switching paradigms

hub – a network element directly connecting two computers on LAN (Local Area Network)

bridge – a network element connecting two LANs

## Internet Layer

Routing of information

router – a network element connecting two networks

## Host-to-Host /Transport Layer

End-to-end transmission – TCP, UDP

## Application Layer

Logic for supporting user applications

# TCP/IP Protocol Architecture

Reliable connection for transfer of data between applications

Connection – temporary logical association between two entities in different systems

## TCP Header

32 bit length

Source Port		Destination Port	
Sequence Number			
Acknowledgement Number			
Header Length	Reserved	Flags	Window
Checksum		Urgent Pointer	
Options + Padding			

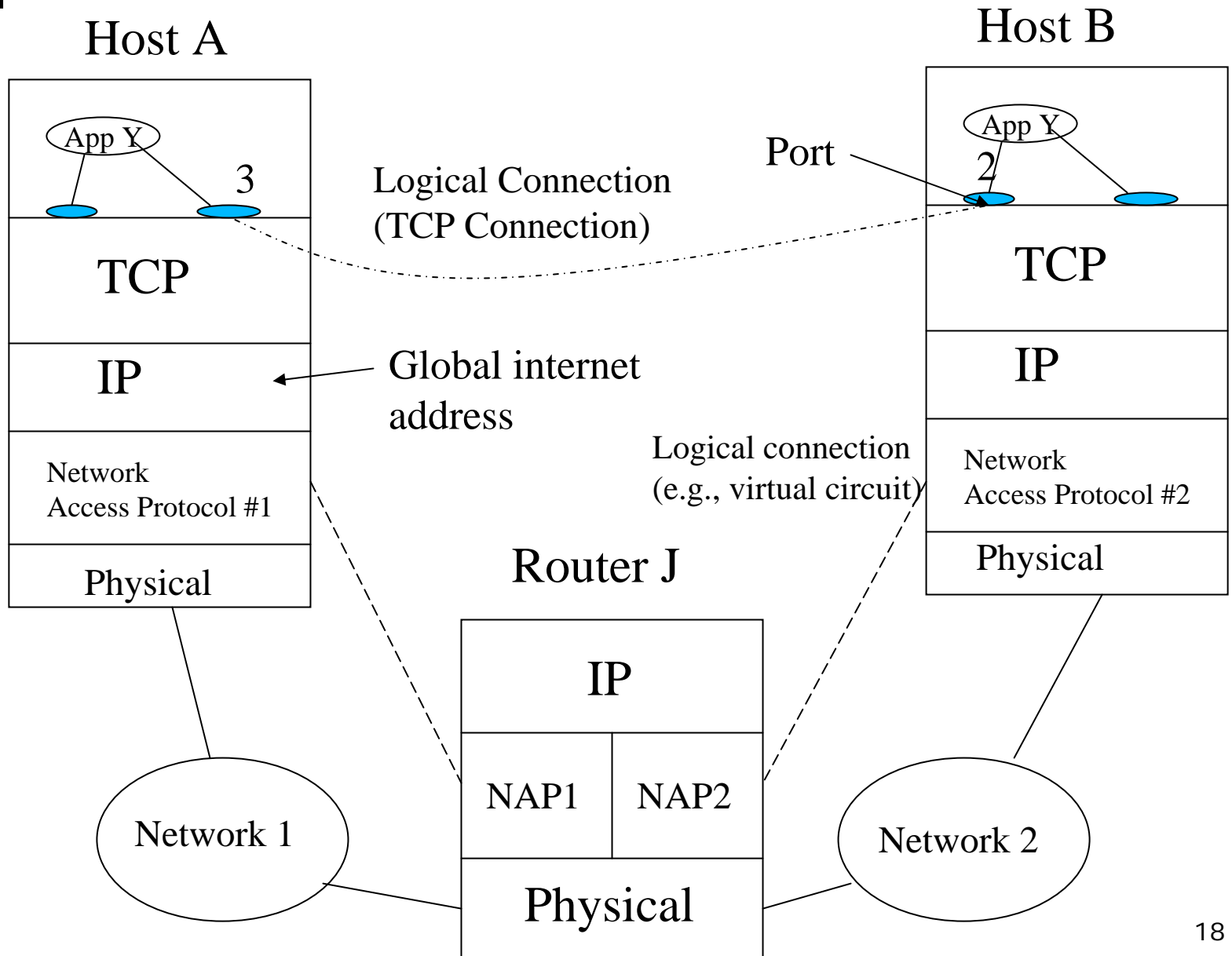
# IP and IPv6

## Interconnectivity Protocol – Internet Protocol (IP)

### IP Header

Version	IHL	DS	ECN	Total Length	
Identification				Flags	Fragment Offset
Time to Live	Protocol		↓	Header Checksum	
Source Address					
Destination Address					
Options + Padding					

# Operations of TCP/IP



# Send Protocol

Sender Application (e.g., App Y on Host A) sends message to TCP over the port 3

TCP hands the message down to IP with instructions to send it to host B, port 2

IP hands message the message down to MAC layer (Network Access Protocol – e.g., Ethernet) with instructions to send it to router J

TCP sends smaller pieces – TCP segments

IP sends IP Datagrams

# Receive Protocol

Router examines IP header and based on destination address

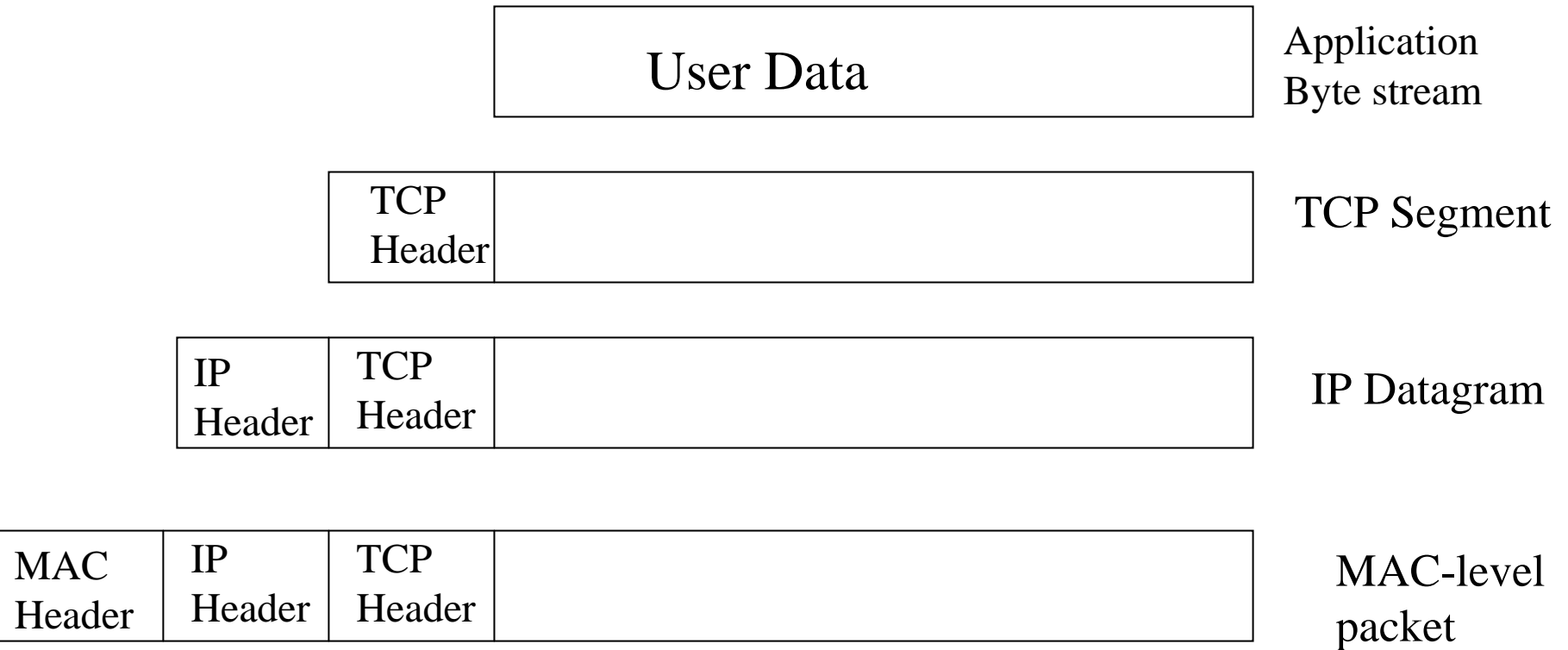
Router directs the IP datagram out across network 2 to B – router augments the datagram with a network access header

Network access protocol on B removes the NAP header and passes the message to higher layer (IP)

IP removes the IP header and passes the message to TCP

TCP does processing and then removes the TCP header and passes the message to application.

# Protocol Data Units (PDUs) in the TCP/IP Architecture



# TCP Functions

**Reliability** – no packets are lost

Achieved via feedback and acknowledgement mechanism

Each packet must be acknowledgement (positive acknowledgement)

If packet is lost and TCP does not get an acknowledgement packet within certain time, TCP retransmits the packet

**Flow Control** – TCP reacts to congestion in the network and slows down if congestion is detected

Achieved via controlling the sending rate

TCP starts to send packets slowly (additive increase in sending packets)

If TCP detects congestion (e.g., packet was lost/not acknowledged), it decreases its rate to send very fast (multiplicative decrease in sending packets)

**Ordering** – packets are given to application in sending order, even if they are received out of order

Achieved via sequence numbering

# Host Names and IP Addresses

Internet Users use Host Names

csil-linux1.cs.uiuc.edu

csil-linux3.cs.uiuc.edu

Host names must be mapped to numeric network addresses for most of the network library calls

System admins define mechanisms by which names are translated into network addresses

**Domain Name Service (DNS) is the glue that integrates naming on the Internet**

Host names are stored in ASCII strings

# IP Address

IP addresses are specified

- **binary** in network byte order in **s\_addr** field of **struct in\_addr**

- **human readable form** – dotted-decimal notation or Internet address dot notation

129.115.30.129 (IP address of usp.cs.utsa.edu)

IPv4 address – 32 bits (4Bytes long)

IPv6 address – 128 bits (16Bytes long)

**in\_addr\_t inet\_addr(const char \*cp)**

- Function converts a dotted-decimal notation address to binary in network byte order

**char \*inet\_ntoa(const struct in\_addr\_in)**

- Function converts binary form into dotted-decimal notation

# Conversion of Host Name to IP Address

Traditional way of converting host name to a binary address is

```
#include <netdb.h>  
struct hostent *gethostbyname(const char *name)  
  
struct hostent {  
    char *h_name; /*canonical 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 */  
};
```

# Example

```
char *hostn = "usp.cs.utsa.edu";
struct hostent *hp;
struct sockaddr_in server;

if ((hp = gethostbyname(hostn)) == NULL
    fprintf(stderr, "Failed to resolve host name \n");
else
    memcpy((char *)&server.sin_addr.s_addr,
           hp->h_addr_list[0],
           hp->h_length);
```

# Conversion from Address to name

```
#include <netdb.h>
```

```
struct hostent *gethostbyaddr (const void *addr,  
                                socklen_t len, int type);
```

For IPv4 – type = AF\_INET

# Example

```
struct hostent *hp;
```

```
struct sockaddr_in net;
```

```
int sock;
```

```
if ((hp=gethostbyaddr(&net.sin_addr, 4,AF_INET))  
    printf("Host name is %s\n", hp->h_name);
```

# Another approach to Conversion

Use new POSIX standard 2001

1. `getnameinfo` (instead of `gethostbyname`)
2. `getaddrinfo` (instead of `gethostbyaddr`)

These routines do not use static data

These functions are safe to use in a threaded environment

Problem: not available on many systems

# Summary

Implementation of TCP

TCP/IP Architecture

Host and IP Addressing