

CS241 System Programming Socket Communication (III)

Klara Nahrstedt

Lecture 36

4/19/2006

Content

- Socket Implementation
- Introduction to Connection-less Communication
 - UDP Protocol

Administrative

- Read R&R Section 18.7
- Read R&R Section 20.1 and 20.2

Sockets

- Communication endpoint
- (IP address, Port number)
- Client-server – server listens to a port
- Telnet Port 23, ftp port 21, web server port 80

Ports

- Ports < 1024, standard
- Ports > 1024, user created
- All connections unique

Socket Communication

Client socket
(146.86.5.2/1625)

Web server socket
(161.25.19.9/80)

```
graph LR; A("Client socket  
(146.86.5.2/1625)") --- B("Web server socket  
(161.25.19.9/80)");
```

The diagram illustrates a network connection between two endpoints. On the left, a light blue oval contains the text 'Client socket (146.86.5.2/1625)'. On the right, another light blue oval contains the text 'Web server socket (161.25.19.9/80)'. A dark blue line connects the two ovals, representing the communication channel between the client and the server.

UICI built on top of 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 Function

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

- Socket function

- Creates a communication endpoint and returns a file pointer (s)
- **domain** is AF_UNIX or AF_INET (indicate IPv4)
- **type** is SOCK_STREAM for connection-oriented TCP or SOCK_DGRAM for connectionless UDP
- **protocol** is usually 0 (since each type protocol has only one protocol available)

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

Socket Structure (1)

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;  
struct in_addr sin_addr;
```

Socket Structure (2)

```
sa_family_t sin_family;  
in_port_t sin_port;  
struct in_addr sin_addr;
```

- The `sin_family` should be `AF_INET`.
- The `sin_port` should be the port number in `network byte order`. You can convert a port number to network byte order using:
`uint16_t htons(uint16_t port);`
- For a `server`, the `sin_addr` can be set to `INADDR_ANY` to accept connections on any interface card.
- For a `client`, it must be filled in with the `IP address` of the remote machine in `network byte order`.

Socket Structure (3)

```
struct sockaddr_in server;  
int sock;
```

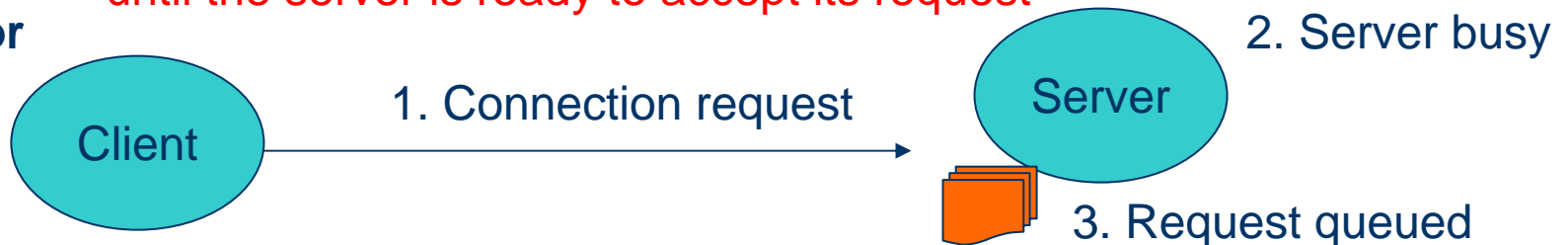
```
server.sin_family = AF_INET;  
server.sin_addr.s_addr = htonl(INADDR_ANY);  
server.sin_port = htons((short)8652);  
if (bind(sock, (struct sockaddr *)&server, sizeof(server)) == -1)  
    perror("Failed to bind the socket to port");
```

Listen function

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

- Socket function
 - creates a communication point
- Bind function
 - associates this end-point with a particular network address
- Listen function
 - 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

Reason for Listen



Accept Function

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

- Accept call allows server to handle incoming client connections
- The accept call 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 Function

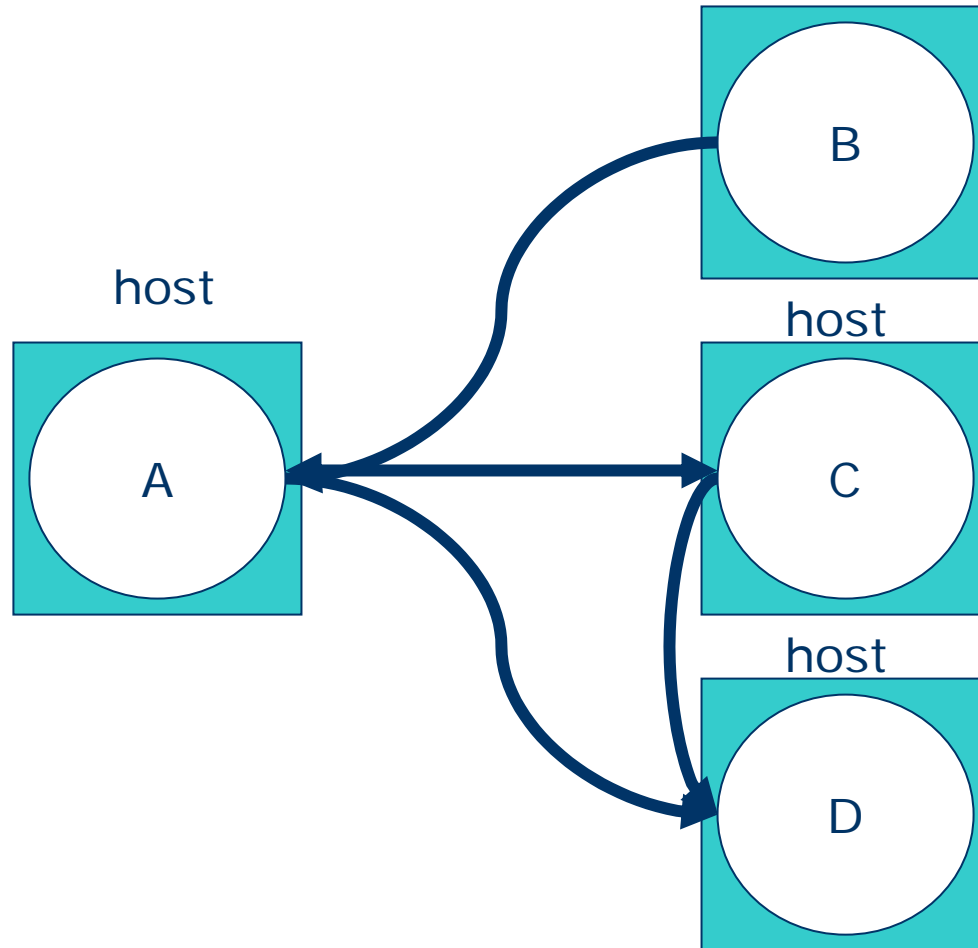
```
int connect(int socket, const struct sockaddr *address,  
            socklen_t address_len);
```

- Client calls **socket** to setup the communication end point, and then calls **connect** to establish a link to the well-known port of the remote server
- You specify the **address and port number** of the remote server in the second parameter.
- The third parameter contains the **length of the buffer** that you are using for the second parameter.

Connectionless Communication

- Connectionless communication
 - Abstraction based on transmission of a single message (called datagram) between sender and receiver
 - Makes an association between two end points
- Datagram
 - Unit of data transmitted from one end point to another

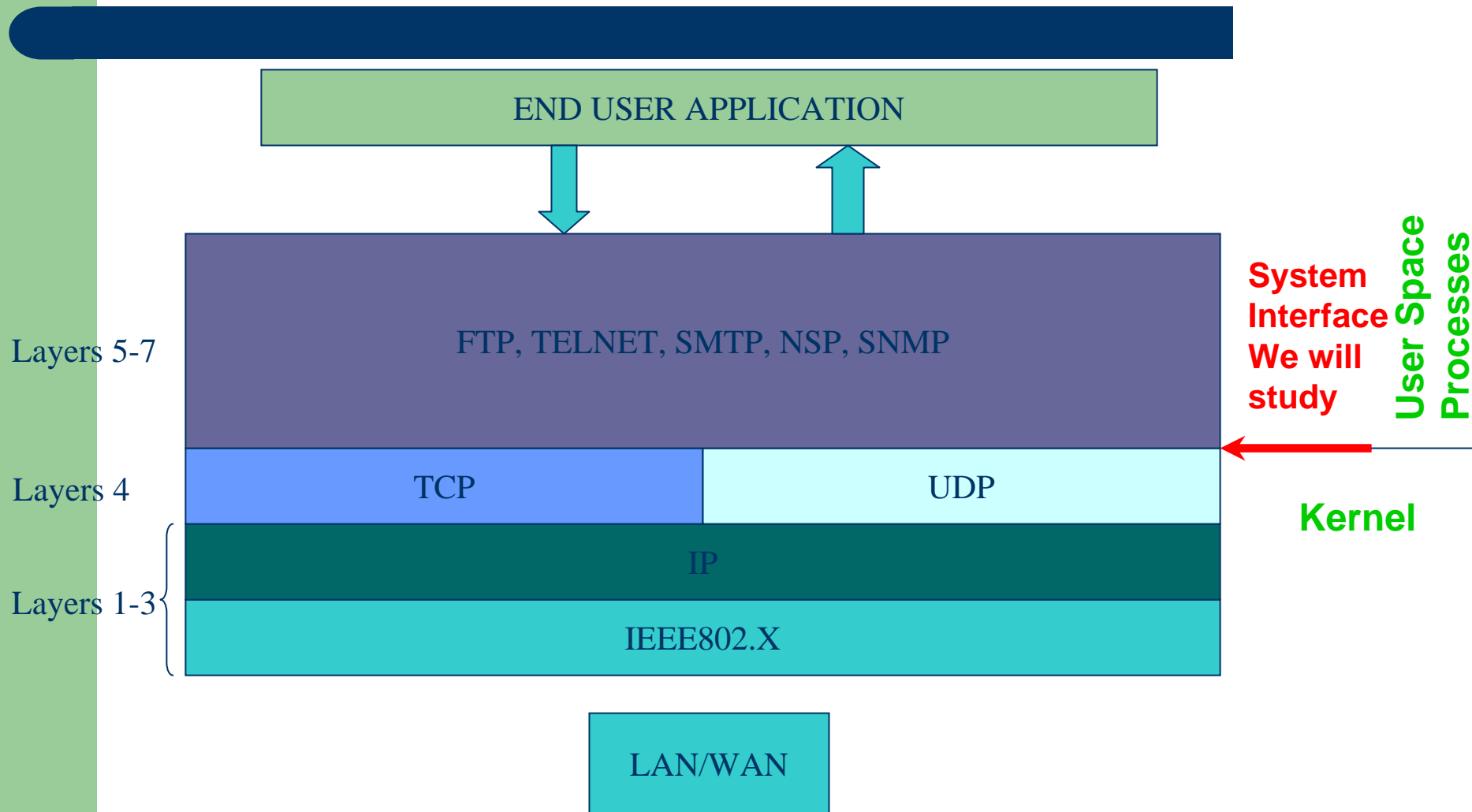
Four Processes with Connectionless Communication Endpoints (Example)



User Datagram Protocol (UDP)

- UDP is a **common Internet protocol**
 - Used very commonly in multimedia streaming applications
 - Used in various network services such as Domain Name Service (DNS), Network File System (NFS), Network Time Protocol (NTP), Simple Network Management Protocol (SNMP)
- Characteristics of UDP
 - Unreliable
 - **What does unreliable protocol mean?**
 - No congestion control
 - **What does it mean for the network if no congestion control exists?**
 - No flow control
 - **What does it mean for the network if no flow control exists?**

UDP/IP Protocol Layers



UDP Communication Endpoints

- Host IP and Port Number
- UDP is unreliable
- UDP Packets sometimes have checksums
 - What is checksum?
 - Why are checksums important?
 - Where are checksums stored?
- Not authenticated
- UDP datagrams arrive atomically
 - What does it mean that the datagrams arrive atomically?
 - How is it achieved?

UICI UDP Simplified Interface for Connectionless Communication

<code>int u_opendup(u_port_t port)</code>	Creates a UDP socket and if port > 0, binds socket to port returns the socket file descriptor
<code>ssize_t u_recvfrom(int fd, void *buf, size_t nbytes, u_buf_t *ubufp)</code>	Waits for up to nbytes from socket fd, returns number of bytes received on return buf has received bytes and ubufp points to sender address
<code>ssize_t u_recvfromtimed(int fd, void *buf, size_t nbytes, u_buf_t *ubufp, double time)</code>	Waits at most time seconds for up to nbytes from socket fd, returns number of bytes received on return buf has received bytes and ubufp points to sender address

UICI UDP Simplified Interface for Connectionless Communication

<pre>ssize_t u_sendto(int fd, void *buf, size_t nbytes, u_buf_t *ubufp)</pre>	<p>sends nbytes of buf on socket fd, to the receiver specified by ubufp returns number of bytes actually sent</p>
<pre>ssize_t u_sendtohost(int fd, void *buf, size_t nbytes, Char *hostn, u_port_t port)</pre>	<p>sends nbytes of buf on socket fd, to the receiver specified by hostn and port returns number of bytes actually sent</p>

UICI UDP Simplified Interface for Connectionless Communication

<code>void u_gethostname(u_buf_t *ubufp, char *hostn, int hostnsize)</code>	Copies host name specified by ubufp into buffer hostn of size hostnsize
<code>void u_gethostinfo(u_buf_t *ubufp, char *info, int infosize)</code>	Copies printable string containing host name and port specified by ubufp into user supplied buffer info of size infosize
<code>int u_comparehost(u_buf_t *ubufp, char *hostn, u_port_t port)</code>	Returns 1 if host and port specified by ubufp match given host name and port number , or else returns 0

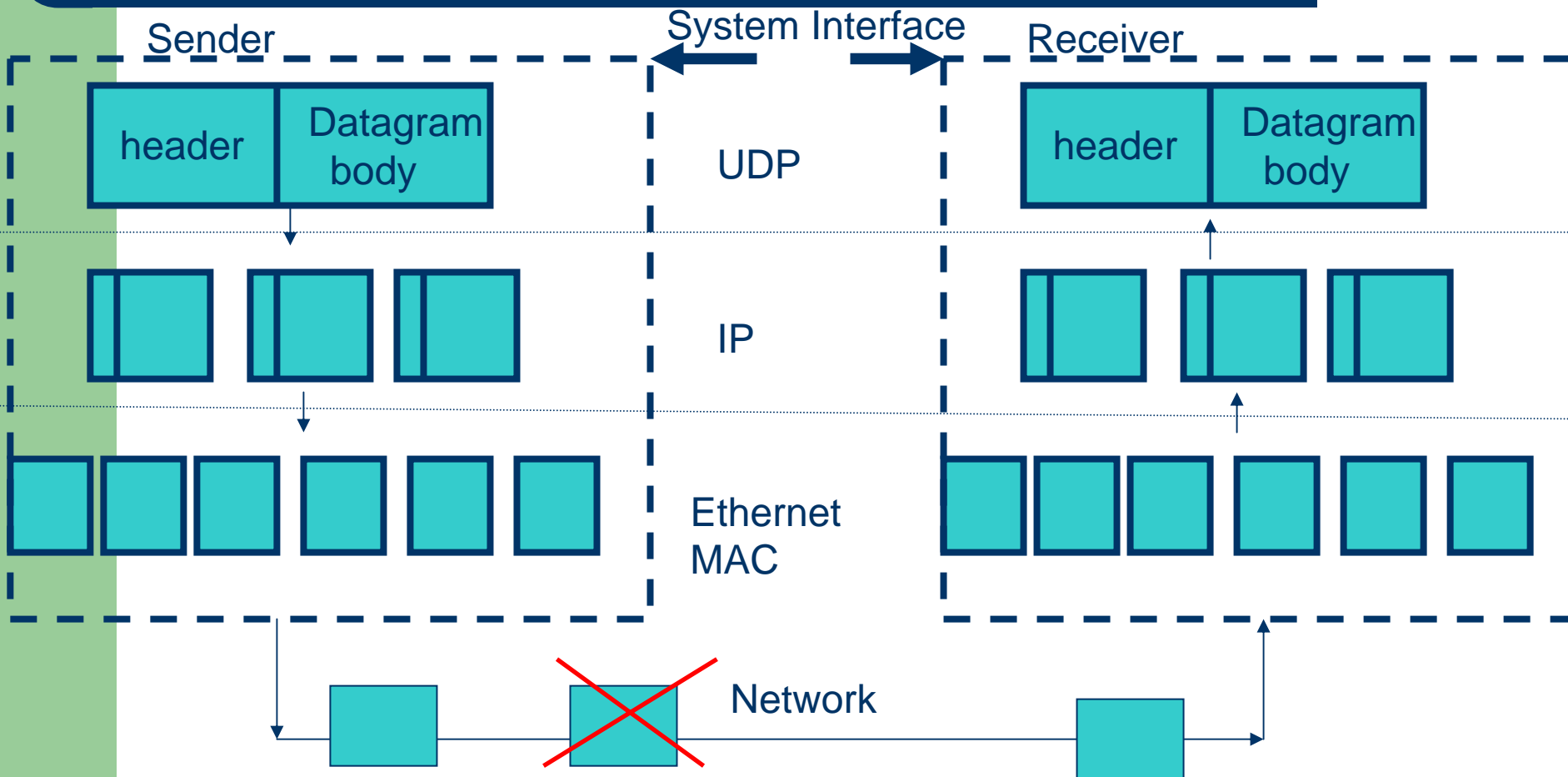
Details

- **u_openudp** – If port is 0, doesn't bind to port. Typically, servers bind to port, clients do not.
- **u_recvfromtimed** - Robust receivers call this function to avoid blocking indefinitely
- **u_sendtohost** – Clients use this to initiate communication with server on well-known port

Buffer Sizes

- UDP messages are sent **atomically**
- UDP datagram has a maximum message size
- Sending a message that is too large returns **-1 and errno set to EMSGSIZE**
- U-recvfrom reads exactly one message;
 - If message is larger than nbytes, buf is filled and truncate the message. **No error is generated !!!!**
 - If message is smaller than nbytes, and buf includes entire message
- UDP Datagrams are ≤ 8192 .

Fragmentation and Reassembly



Summary

- Socket interfaces are the central entities of the UICI interface
- Sockets are part of the UNIX standard and in practice you will program with sockets (not with UICI interfaces)
- UICI is a good educational tool
- UDP is an unreliable protocol that is begin used in many applications due to its simplicity and real-time behavior, but if we need a reliable information transfer, we should use TCP or augment UDP with reliability, flow control and congestion control.
- Many protocols are being built on top of UDP if new algorithms want to be tried out.