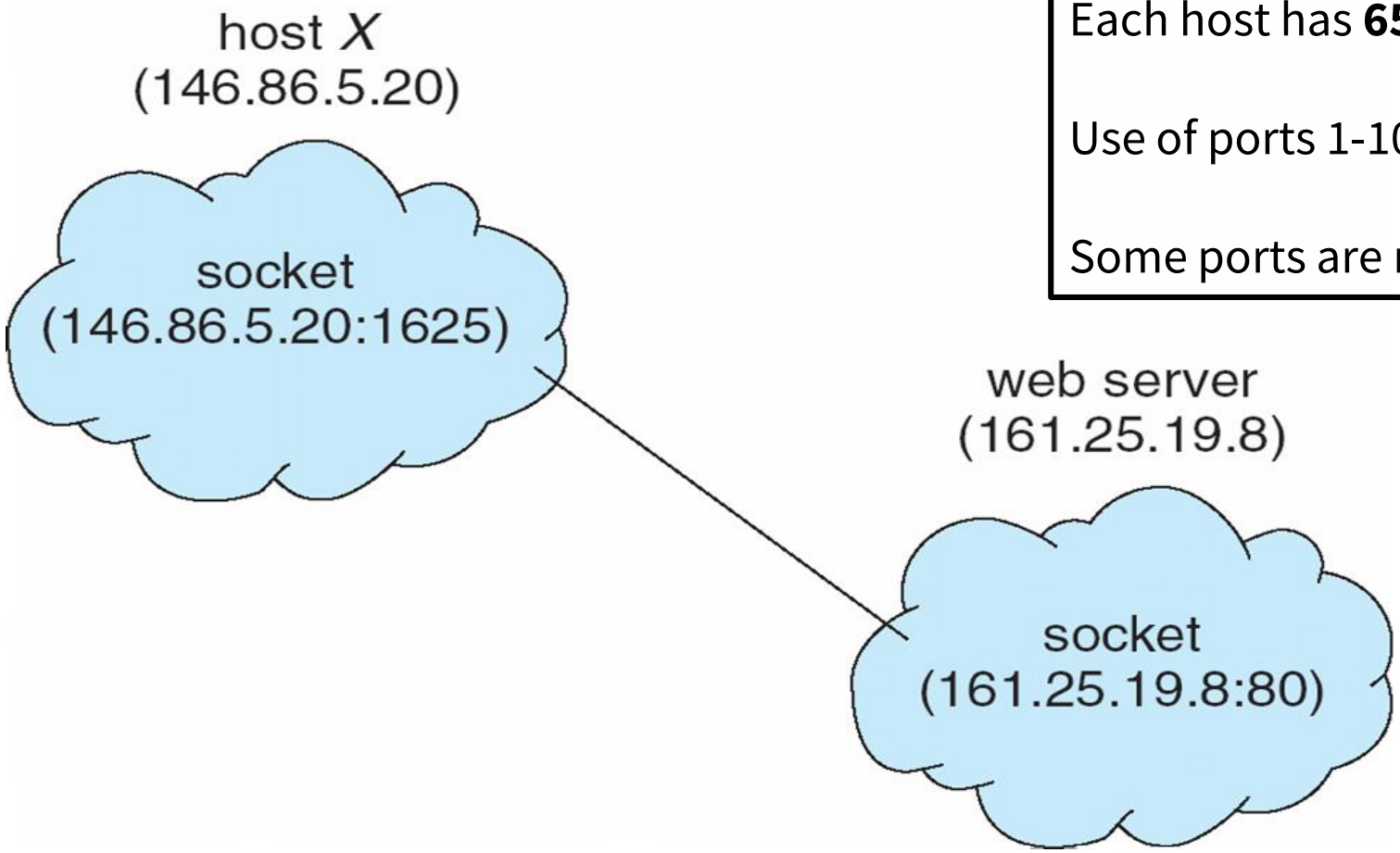
NWEN 241
Systems Programming

Week 7 Tutorial

Recall: Socket communication



Each host has **65,536** ports

Use of ports 1-1023 requires privileges

Some ports are reserved for specific apps

Recall: Socket types

- SOCK_STREAM
  - a.k.a. **TCP**
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional
- SOCK_DGRAM
  - a.k.a. **UDP**
  - unreliable delivery
  - no order guarantees
  - no notion of “connection” – app indicates dest. for each packet
  - can send or receive

We will focus on SOCK_STREAM or TCP socket type

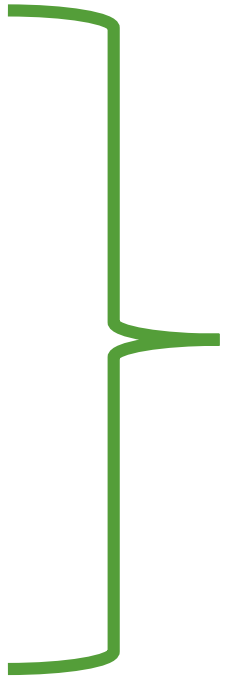
TCP Vs UDP

Feature	TCP	UDP
Connection status	Requires an established connection to transmit data (connection should be closed once transmission is complete)	Connectionless protocol with no requirements for opening, maintaining, or terminating a connection
Guaranteed delivery	Can guarantee delivery of data to the destination router	Cannot guarantee delivery of data to the destination
Retransmission of data	Retransmission of lost packets is possible	No retransmission of lost packets
Method of transfer	Data is read as a byte stream; messages are transmitted to segment boundaries	UDP packets with defined boundaries; sent individually and checked for integrity on arrival
Speed	Slower than UDP (due to overheads involved for maintaining accuracy)	Faster than TCP
Optimal use	Where accuracy is more important than speed. Used by HTTPS, FTP, etc.	Where speed is more important than accuracy Video conferencing, streaming, DNS, VoIP, etc.

Note: TCP establishes a virtual connection – packets may or may not follow the same path (depends if the Network layer protocol are connection oriented.) . IP – is connection-less

Recall: System calls

- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `send() / sendto()`
- `recv() / recvfrom()`



Include

`sys/types.h`

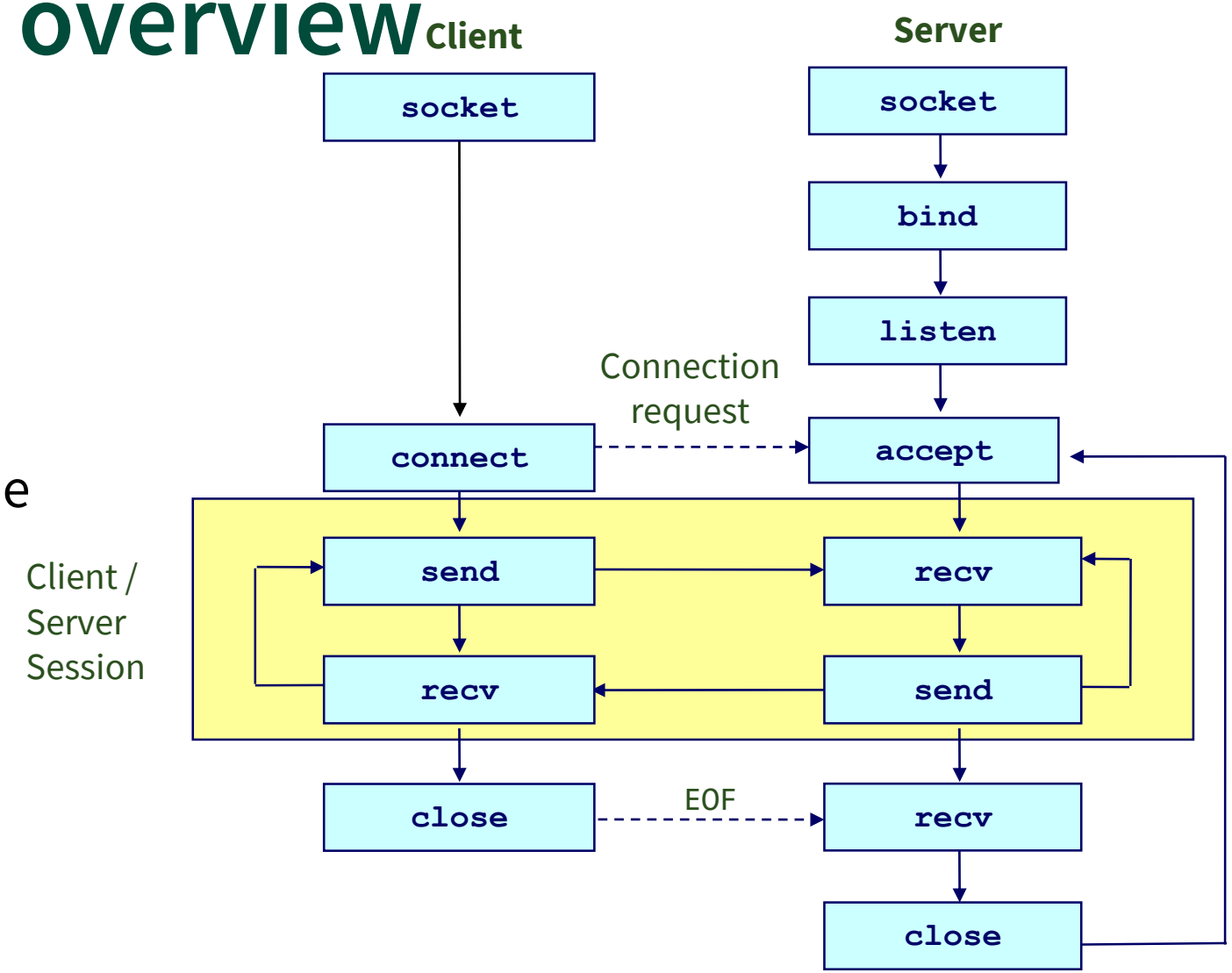
`sys/socket.h`

Recall: TCP Client overview

1) Create a socket with the `socket()` system call

2) Connect the socket to the address of the server using the `connect()` system call

3) Send and receive data



Recall: TCP Server overview

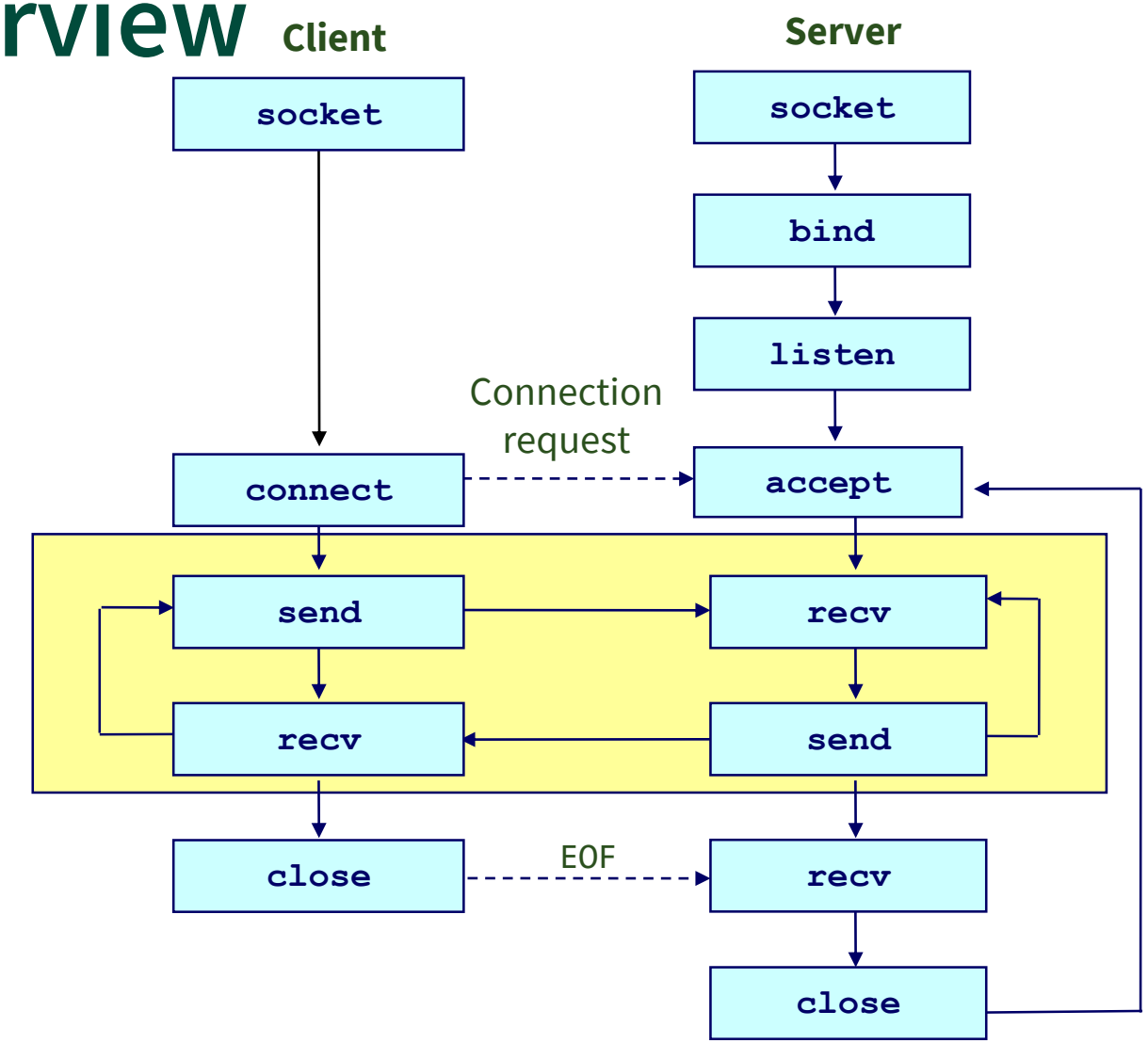
1) Create a socket with the `socket()` system call

2) Bind the socket to an address using the `bind()` system call

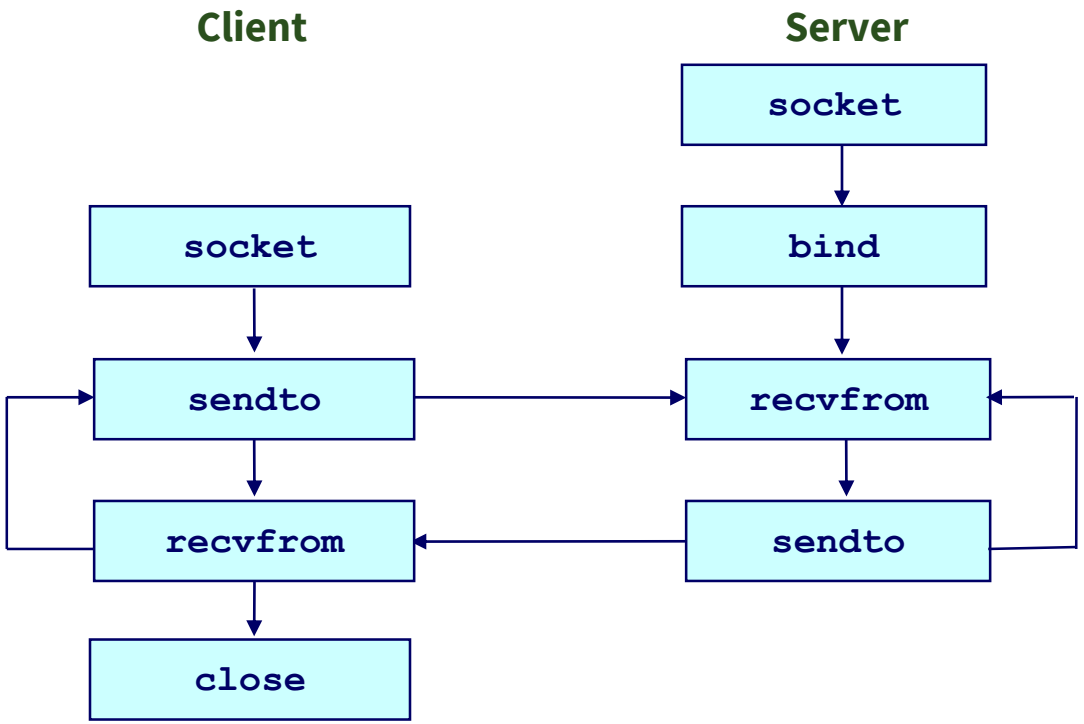
3) Listen for connections with the `listen()` system call

4) Accept a connection with the `accept()` system call

5) Send and receive data



Recall: Client-server communication overview - UDP



Creating a socket (server and client)

- Request a socket from the system. Returns a valid socket file descriptor or -1 if the allocation fails.

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

  - *domain* – communication domain (protocol family) such as AF_INET (IPv4) or AF_INET6 (IPv6)
  - *type* – communication semantics such as SOCK_STREAM (TCP) or SOCK_DGRAM (UDP)
  - *protocol* specifies the protocol, usually 0.

Binding a socket (server only)

- Bind the socket to a local address using the `bind()` system call

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

  - *sockfd* is the socket file descriptor from the result of socket() function
  - *sockaddr* is a custom structure defining IP address and Port
  - *addr* pointer to the structure variable of type *struct sockaddr* which contains the host IP address and port number to bind to (`struct sockaddr_in` in IPv4)
    - *sin_family:* same as domain above
    - *sin_addr.s_addr:* automatically set with INADDR_ANY or manually set with inet_aton()
    - *sin_port:* use *htons(PORT)* to convert to correct endian format
  - *addrlen* is the length of what addr points to

Listen for connection request (Server Only)

- Listen for clients to request connections with the `listen()` system call

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

  - *`sockfd`* is the socket file descriptor (returned by `socket()`)
  - *`backlog`* is the maximum number of pending connections to allow for this socket. `SOMAXCONN` is defined as the number of maximum pending connections allowed by the operating system

Accept connection requests (Server Only)

- Accept a connection with the `accept()` system call and create a new socket file descriptor for this client

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

  - *`sockfd`* is the socket file descriptor (returned by `socket()`)
  - *`addr`* pointer to the structure variable of type *`struct sockaddr`* which contains the host IP address and port of the client
  - *`addrlen`* is a pointer to the length of what `addr` points to

  - If successful, returns non-negative **socket file descriptor**, otherwise, returns -1

Connect to Server (Client Only)

- Establish connection from Client to the Server. Returns
  - This step is only required for connection-oriented sockets (TCP)

```
int connect(int sockfd, const struct sockaddr *addr,
socklen_t addrlen);
```

  - *sockfd* is the socket file descriptor (returned by socket())
  - *addr* is a pointer to the structure `struct sockaddr` which will contain the details of the server socket
  - *addrlen* is a pointer to the length of what addr points to

Send and Receive (Server and Client)

- **Send** and receive data

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

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

  - *sockfd* is the socket file descriptor (returned by accept() or socket())
  - *buf* is a pointer to buffer to be sent
  - *len* is the length of buffer to be sent
  - *flags* is bitwise OR of zero or more options (Optional, rarely used, typically 0)

- `send(sockfd, buf, len, 0);` is equivalent to `write(sockfd, buf, len);`
- `recv(sockfd, buf, len, 0);` is equivalent to `read(sockfd, buf, len);`

Closing a socket

- Socket must be closed after its use

```
int shutdown(int sockfd, int how);
```

```
int close(int sockfd);
```

  - *sockfd* is the socket file descriptor (returned by socket())
  - *how* can either be SHUT_RD (further receptions disallowed), SHUT_WR (further transmissions disallowed), or SHUT_RDWR (further receptions and transmissions disallowed)
  - Shutdown blocks communication without destroying the socket, close blocks the communication and destroys the socket.
  - If successful, returns 0, otherwise, returns -1