ECGR 3123: Data Communications & Networking

Tao Han, Ph.D

Assistant Professor
Department of Electrical and Computer Engineering
The William States Lee College of Engineering
University of North Carolina-Charlotte

http://webpages.uncc.edu/than3

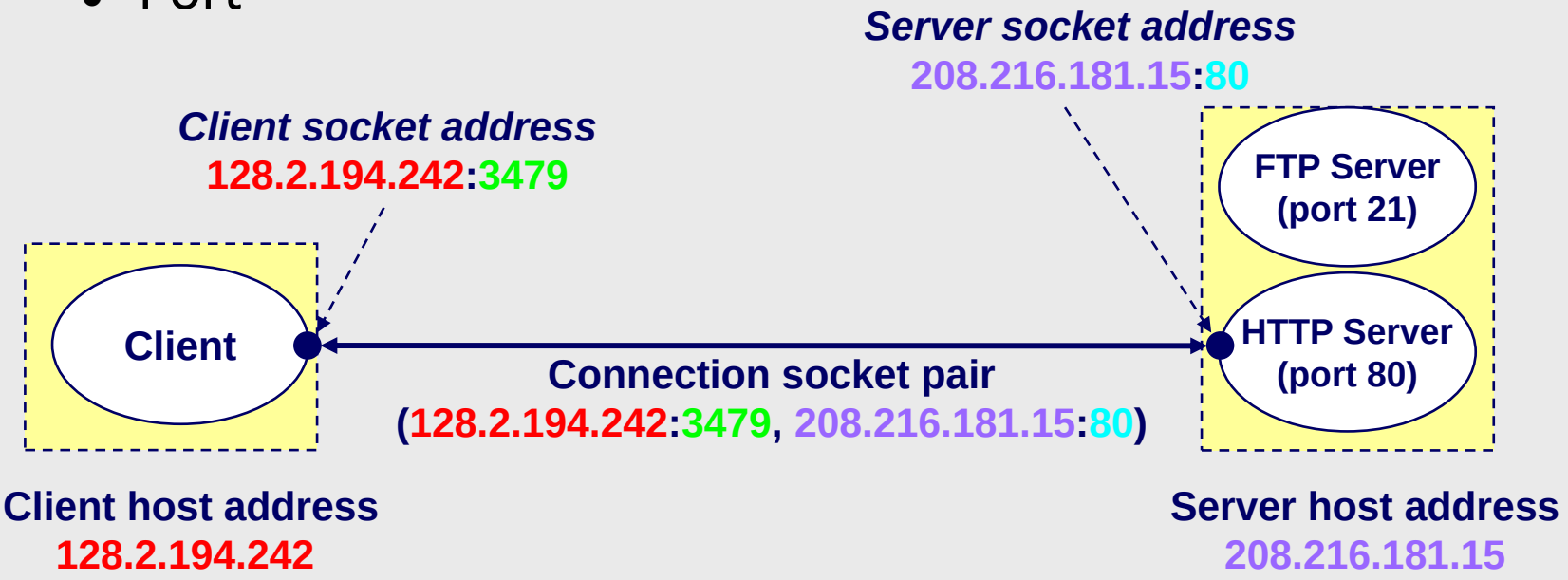
SOCKET PROGRAMMING

Sockets

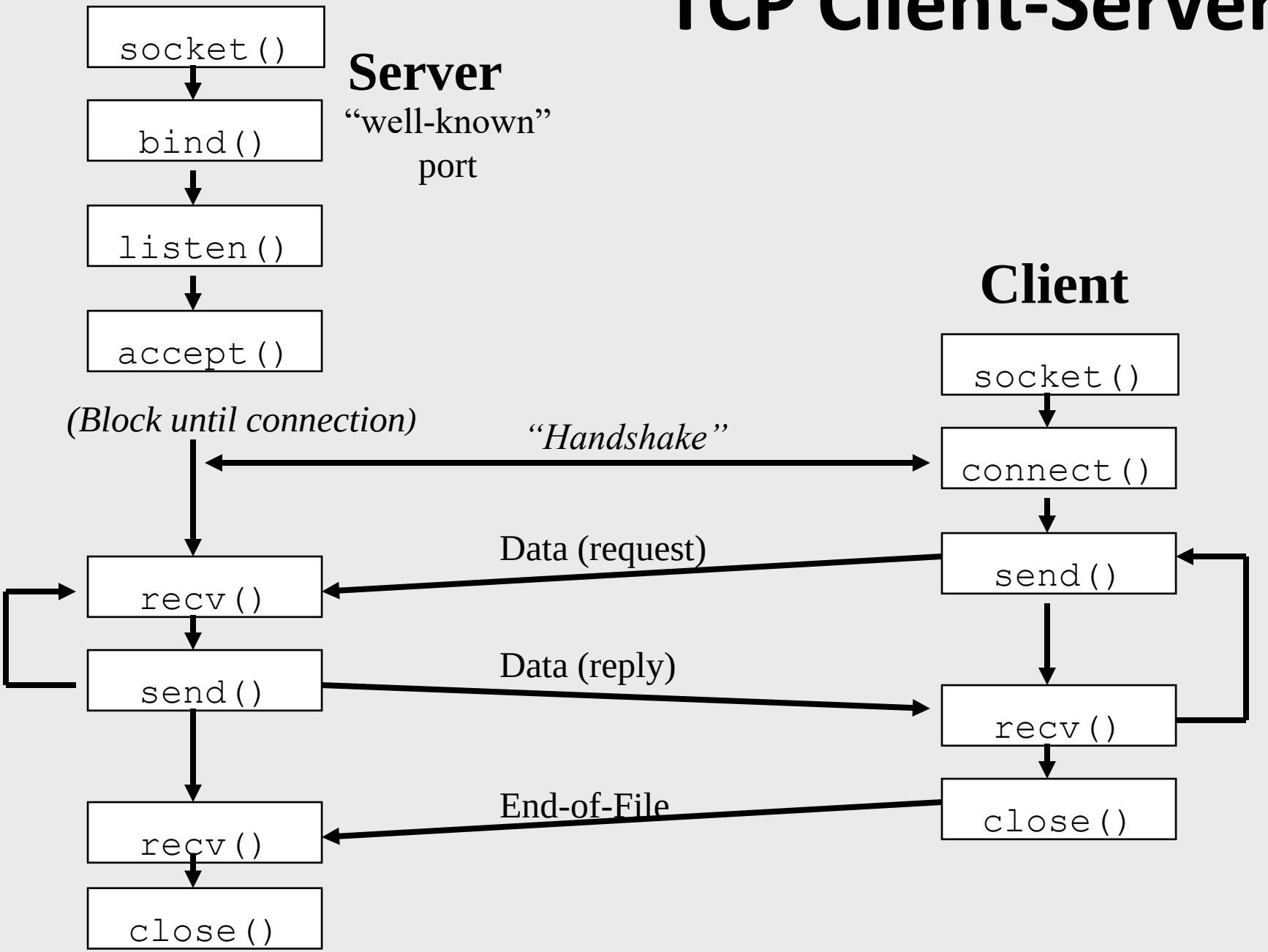
- What is a Socket
  - Socket is the API to access transport layer functions
- How to use sockets
  - Setup socket
    - Where is the remote machine (IP address, hostname)
    - What service gets the data (port)
  - Send and Receive
    - Designed just like any other I/O in unix
    - send -- write
    - recv -- read
  - Close the socket

Identify the Destination

- Addressing
  - IP address
  - Hostname (resolve to IP address via DNS)
- Multiplexing
  - Port



TCP Client-Server



Step 1 – Setup a socket()

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

Create a socket/file descriptor, giving access to transport layer service.

- *family* is one of
  - AF_INET (IPv4), AF_INET6 (IPv6), AF_LOCAL (local Unix),
  - AF_ROUTE (access to routing tables), AF_KEY (new, for encryption)
- *type* is one of
  - SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
  - SOCK_RAW (for special IP packets, PING, etc. Must be root)
- *protocol* is 0 (used for some raw socket options)
- upon success returns socket descriptor
  - Integer, like file descriptor
  - Return -1 if failure

Step 2: bind()

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

Assign a local protocol address ("name") to a socket.

- *sockfd* is socket descriptor from `socket()`
- *myaddr* is a pointer to address struct with:
  - *port number* and *IP address*
  - if port is 0, then host will pick ephemeral port
  - IP address = INADDR_ANY (unless multiple nics)
- *addrlen* is length of structure
- returns 0 if ok, -1 on error
  - EADDRINUSE ("Address already in use")

Socket Address Structure

```
struct in_addr {
  in_addr_t s_addr;          /* 32-bit IPv4 addresses */
};
struct sockaddr_in {
  unit8_t       sin_len;     /* length of structure */
  sa_family_t   sin_family;  /* AF_INET */
  in_port_t     sin_port;    /* TCP/UDP Port num */
  struct in_addr sin_addr;   /* IPv4 address (above) */
  char sin_zero[8];          /* unused */
}
```

- Are also “generic” and “IPv6” socket structures

Convert Byte order

- **Little Endian** – In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1).
- **Big Endian** – In this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).

Convert Byte order

‘h’ : host byte order

‘n’ : network byte order

‘s’ : short (16bit)

‘l’ : long (32bit)

```
uint16_t htons(uint16_t);
uint16_t ntohs(uint_16_t);

uint32_t htonl(uint32_t);
uint32_t ntohl(uint32_t);
```

In_addr values should be stored in network byte order

bind() example

```
int sd;
struct sockaddr_in ma;
sd = socket(AF_INET,SOCK_STREAM,0);

ma.sin_family = AF_INET;
ma.sin_port=htons(5100);
ma.sin_addr.s_addr=htonl(INADDR_ANY);
if(bind(sd, (struct sockaddr *) &ma,
  sizeof(ma))!= -1) { …}
```

Step 3: listen()

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

Change socket state for TCP server.

- *sockfd* is socket descriptor from `socket()`
- *backlog* is maximum number of *incomplete* connections
  - historically 5
  - rarely above 15 on a even moderate Web server!

Step 4: accept()

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

Return next completed connection.

- *sockfd* is socket descriptor from `socket()`
- *cliaddr* and *addrlen* return protocol address from client
- returns brand new descriptor, created by OS

listen/connect() example

```
struct sockaddr_in ca;
//After bind..
listen(sd,5);
calen = sizeof(ca);
cd = accept(sd,(struct sockaddr
  *) ca, &calen);

//read and write using cd
```

Client: connect()

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

Connect to server.

- *sockfd* is socket descriptor from `socket()`
- *servaddr* is a pointer to a structure with:
  - *port number* and *IP address*
  - must be specified (unlike `bind()`) – how?
- *addrlen* is length of structure
- client doesn't need `bind()`
  - OS will pick ephemeral port
- returns socket descriptor if ok, -1 on error

connect() example

```
int sd;
struct sockaddr_in sa;
sd = socket(AF_INET,SOCK_STREAM,0);


sa.sin_family = AF_INET;
sa.sin_port=htons(5100);
sa.sin_addr.s_addr=
  htonl(inet_addr("128.226.123.101"));
if(connect(sd, (struct sockaddr *) &sa,
  sizeof(sa))!= -1) { …}
```

`close()/shutdown`

`int close(int sockfd);`

Close socket for use.

- *sockfd* is socket descriptor from `socket()`
- Returns -1 if error

Sending and Receiving

```
int recv(int sockfd, void
  *buff, size_t mbytes, int
  flags);
int send(int sockfd, void
  *buff, size_t mbytes, int
  flags);
```

Domain Name System (DNS)

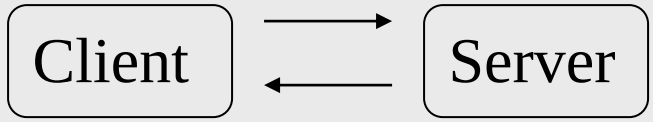
- What if I want to send data to “www.slashdot.org”?
  - DNS: Conceptually, DNS is a database collection of host entries

```
struct hostent {
   char *h_name;       // official hostname
   char **h_aliases;   // vector of alternative hostnames
   int    h_addrtype;   // address type, e.g. AF_INET
   int    h_length;       // length of address in bytes, e.g. 4 for IPv4
   char **h_addr_list; // vector of addresses
   char *h_addr;                  // first host address, synonym for h_addr_list[0]
};
```

- hostname -> IP address
  - *struct hostent *gethostbyname(const char *name);*
- IP address -> hostname
  - *struct hostent *gethostbyaddr(const char *addr, int len, int type);*

Project 1: Client-Server Chat

- Basic requirement (15%):
  - Socket in C++ (Windows, Linux, Mac)
  - Client – Server Communications: Client and server can continuously exchange messages
  - One TCP socket and one UDP socket



- Additional credit (5%)
  - Client – Server – Client Communications

- Additional credit (5%)
  - Group chat: multiple clients can chat with each other in a group