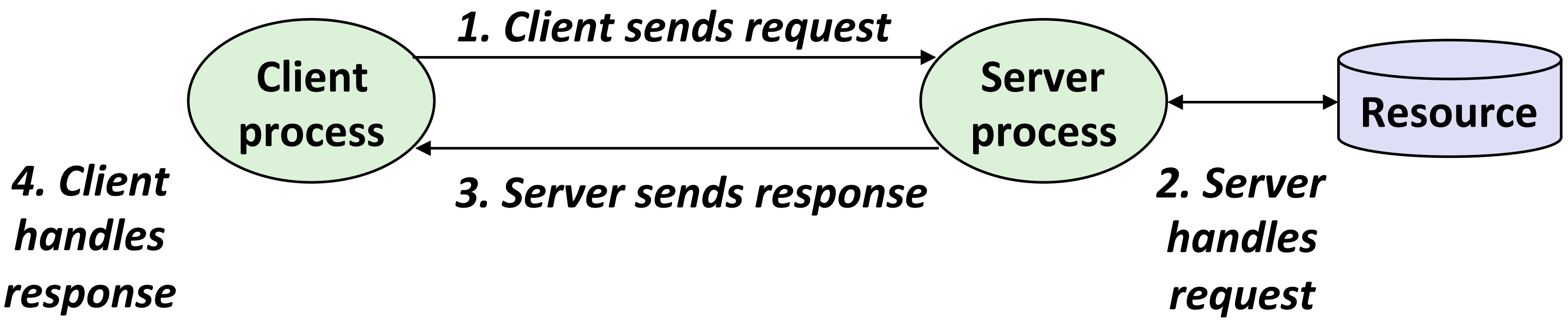
CSSE132
Introduction to Computer Systems

29 : Network Programming

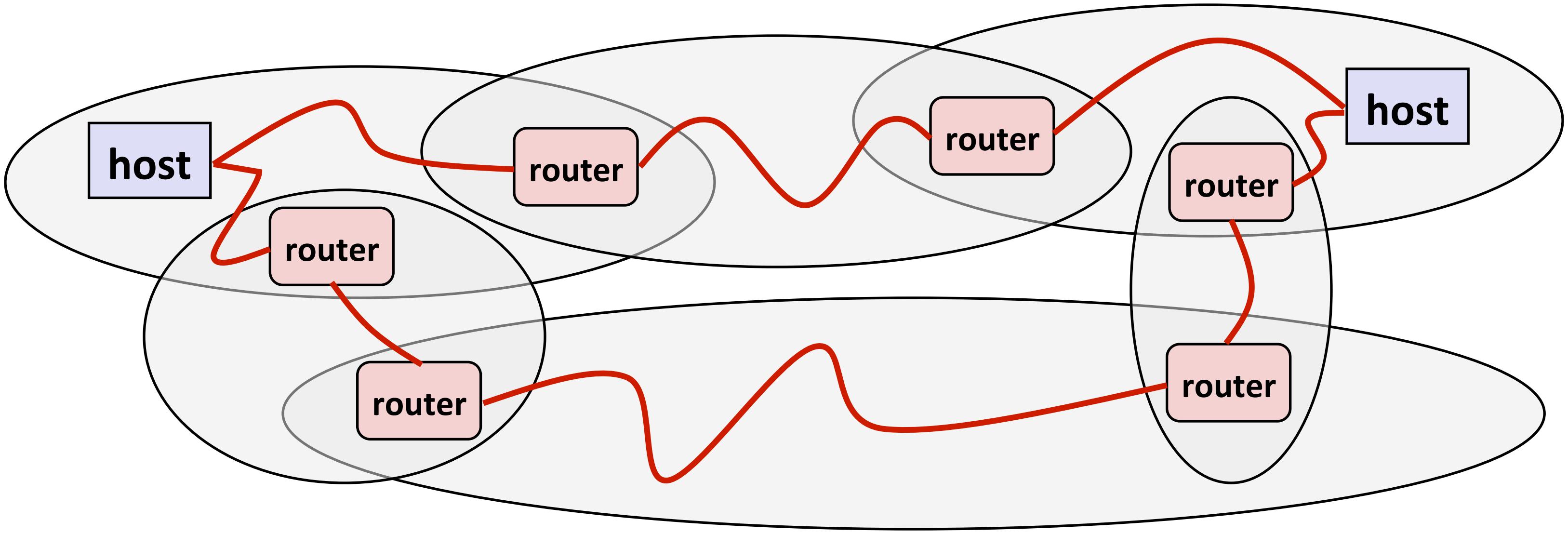
April 29, 2013

Last Time: Client-Server Transaction



Note: clients and servers are processes running on hosts (can be the same or different hosts)

Last Time: Logical Structure of an internet



Internet Connections

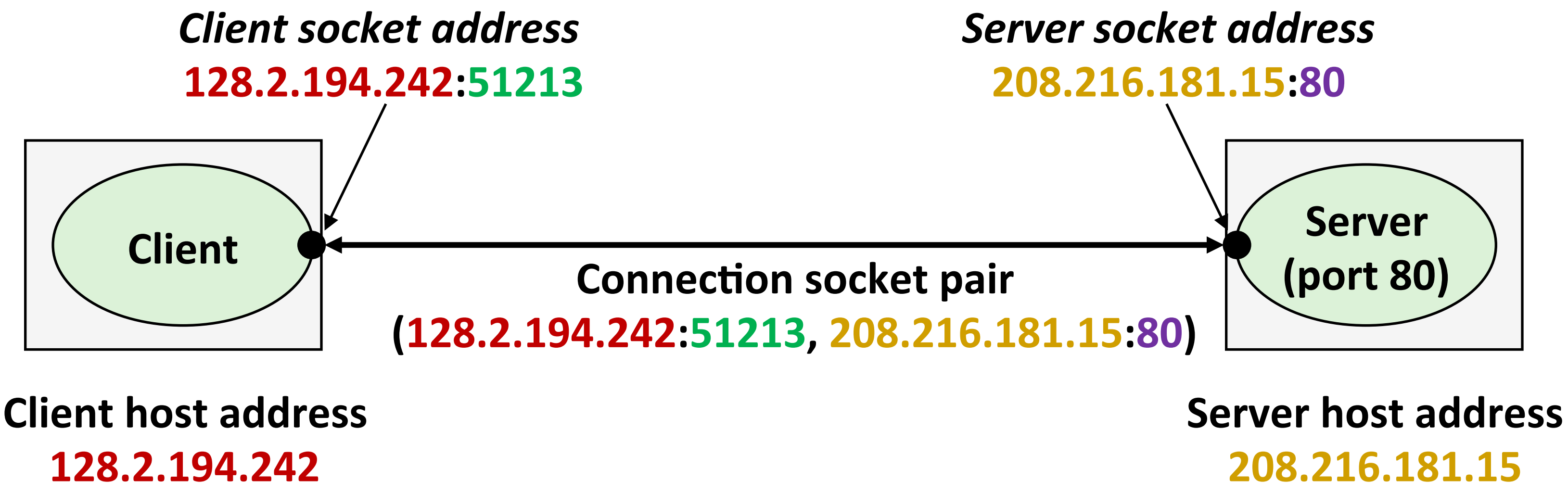
- **Clients and servers communicate by sending streams of bytes over *connections*:**
  - Point-to-point, full-duplex (2-way communication), and reliable.

- ***A socket* is an endpoint of a connection**
  - Socket address is an **`IPaddress:port`** pair

- **A *port* is a 16-bit integer that identifies a process:**
  - ***Ephemeral port*:** Assigned automatically on client when client makes a connection request
  - ***Well-known port:*** Associated with some service provided by a server (e.g., port 80 is associated with Web servers)

- **A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)**
  - **`(cliaddr:cliport, servaddr:servport)`**

Putting it all Together:
Anatomy of an Internet Connection



Client socket address
128.2.194.242:51213

Server socket address
208.216.181.15:80

Client

Server
(port 80)

Connection socket pair
(128.2.194.242:51213, 208.216.181.15:80)

Client host address
128.2.194.242

Server host address
208.216.181.15

51213 is an ephemeral port
allocated by the kernel

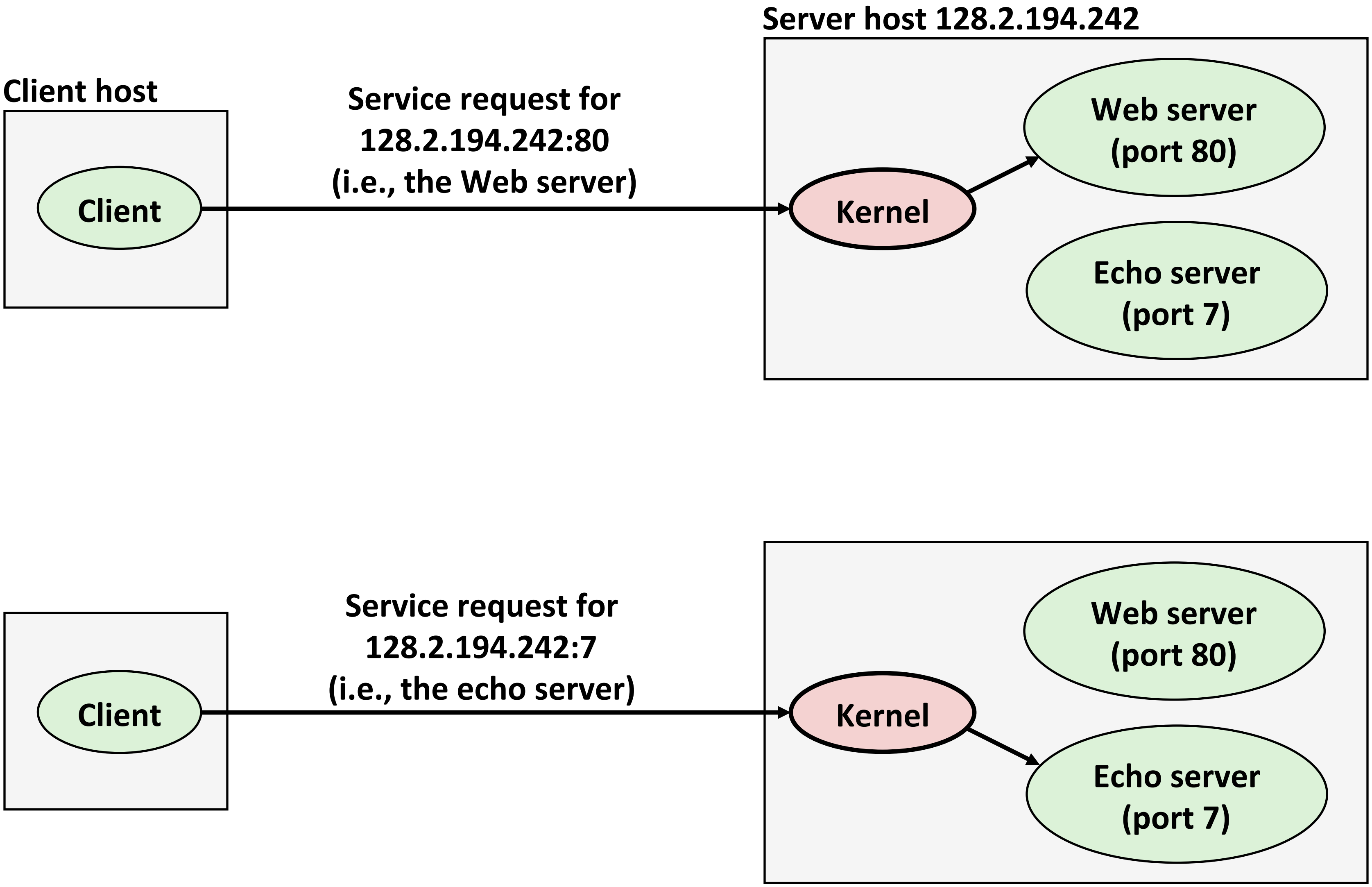
80 is a well-known port
associated with Web servers

Clients

- **Examples of client programs**
  - Web browsers, **`ftp`**, **`telnet`**, **`ssh`**

- **How does a client find the server?**
  - The IP address in the server socket address identifies the host (more precisely, an adapter on the host)
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well know ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

Using Ports to Identify Services



Servers

- **Servers are long-running processes (daemons)**
  - Created at boot-time (typically) by the init process (process 1)
  - Run continuously until the machine is turned off

- **Each server waits for requests to arrive on a well-known port associated with a particular service**
  - Port 7: echo server
  - Port 23: telnet server
  - Port 25: mail server
  - Port 80: HTTP server

- **A machine that runs a server process is also often referred to as a "server"**

Server Examples

- **Web server (port 80)**
  - Resource: files/compute cycles (CGI programs)
  - Service: retrieves files and runs CGI programs on behalf of the client

- **FTP server (20, 21)**
  - Resource: files
  - Service: stores and retrieve files

See `/etc/services` for a comprehensive list of the port mappings on a Linux machine

- **Telnet server (23)**
  - Resource: terminal
  - Service: proxies a terminal on the server machine

- **Mail server (25)**
  - Resource: email “spool” file
  - Service: stores mail messages in spool file

Sockets Interface

- Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols

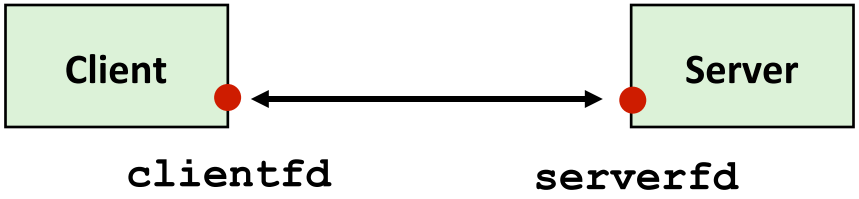
- Provides a user-level interface to the network

- Underlying basis for all Internet applications

- Based on client/server programming model

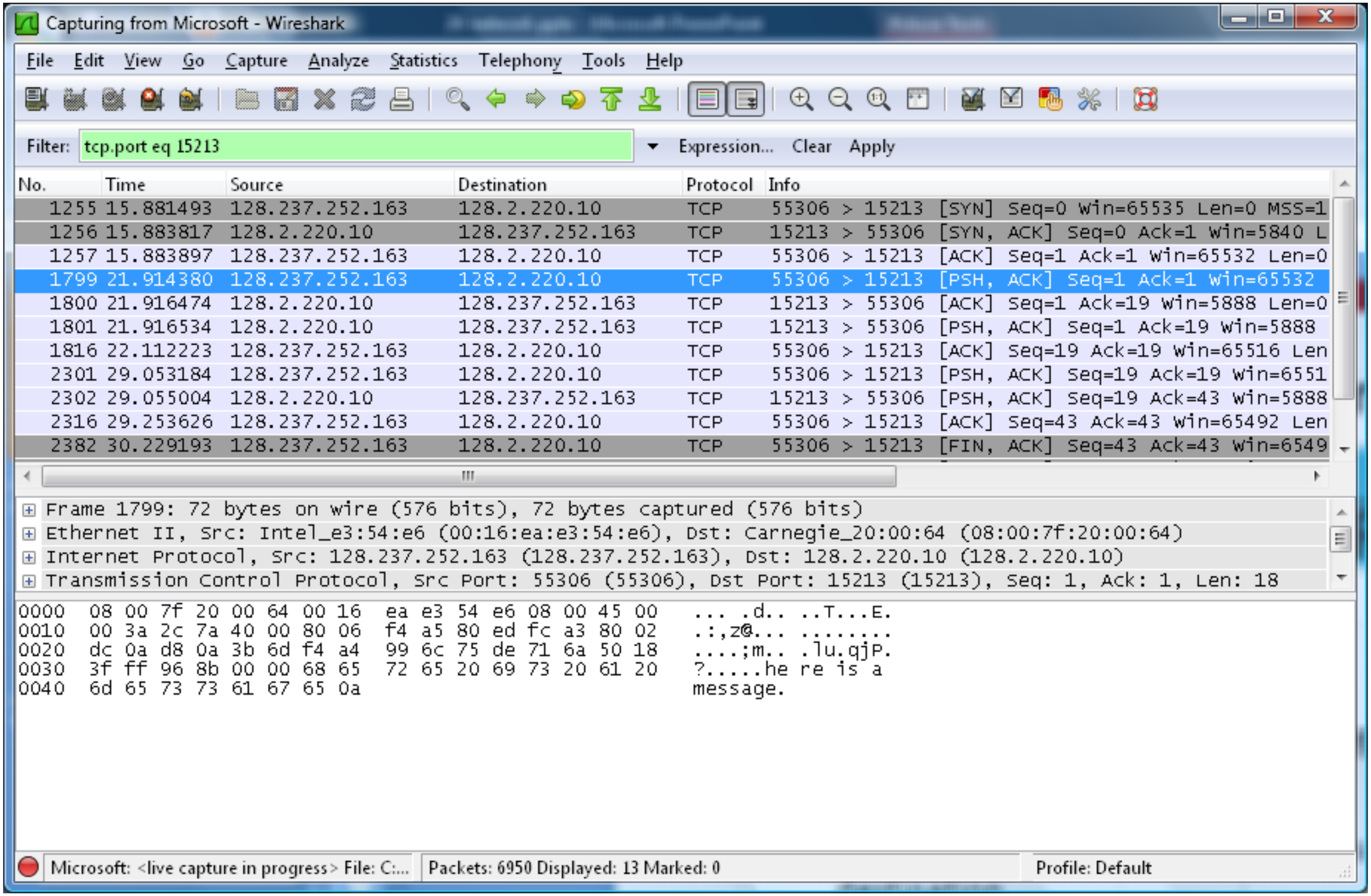
Sockets

- **What is a socket?**
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a **file descriptor** that lets the application read/write from/to the network
    - ***Remember:*** All Unix I/O devices, including networks, are modeled as files
- **Clients and servers communicate with each other by reading from and writing to socket descriptors**



- **The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors**

Watching Echo Client / Server



Filter: tcp.port eq 15213

```
No.    Time      Source          Destination     Protocol Info
  1255 15.881493 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [SYN] Seq=0 Win=65535 Len=0 MSS=1
  1256 15.883817 128.2.220.10    128.237.252.163 TCP      15213 > 55306 [SYN, ACK] Seq=0 Ack=1 Win=5840 L
  1257 15.883897 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [ACK] Seq=1 Ack=1 Win=65532 Len=0
  1799 21.914380 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [PSH, ACK] Seq=1 Ack=1 Win=65532
  1800 21.916474 128.2.220.10    128.237.252.163 TCP      15213 > 55306 [ACK] Seq=1 Ack=19 Win=5888 Len=0
  1801 21.916534 128.2.220.10    128.237.252.163 TCP      15213 > 55306 [PSH, ACK] Seq=1 Ack=19 Win=5888
  1816 22.112223 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [ACK] Seq=19 Ack=19 Win=65516 Len
  2301 29.053184 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [PSH, ACK] Seq=19 Ack=19 Win=6551
  2302 29.055004 128.2.220.10    128.237.252.163 TCP      15213 > 55306 [PSH, ACK] Seq=19 Ack=43 Win=5888
  2316 29.253626 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [ACK] Seq=43 Ack=43 Win=65492 Len
  2382 30.229193 128.237.252.163 128.2.220.10    TCP      55306 > 15213 [FIN, ACK] Seq=43 Ack=43 Win=6549
```

```
⊞ Frame 1799: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)
⊞ Ethernet II, Src: Intel_e3:54:e6 (00:16:ea:e3:54:e6), Dst: Carnegie_20:00:64 (08:00:7f:20:00:64)
⊞ Internet Protocol, Src: 128.237.252.163 (128.237.252.163), Dst: 128.2.220.10 (128.2.220.10)
⊞ Transmission Control Protocol, Src Port: 55306 (55306), Dst Port: 15213 (15213), Seq: 1, Ack: 1, Len: 18
```

```
0000  08 00 7f 20 00 64 00 16  ea e3 54 e6 08 00 45 00   ... .d.. ..T...E.
0010  00 3a 2c 7a 40 00 80 06  f4 a5 80 ed fc a3 80 02   .:,z@... ........
0020  dc 0a d8 0a 3b 6d f4 a4  99 6c 75 de 71 6a 50 18   ....;m.. .lu.qjP.
0030  3f ff 96 8b 00 00 68 65  72 65 20 69 73 20 61 20   ?.....he re is a
0040  6d 65 73 73 61 67 65 0a                            message.
```

Microsoft: <live capture in progress> File: C:...

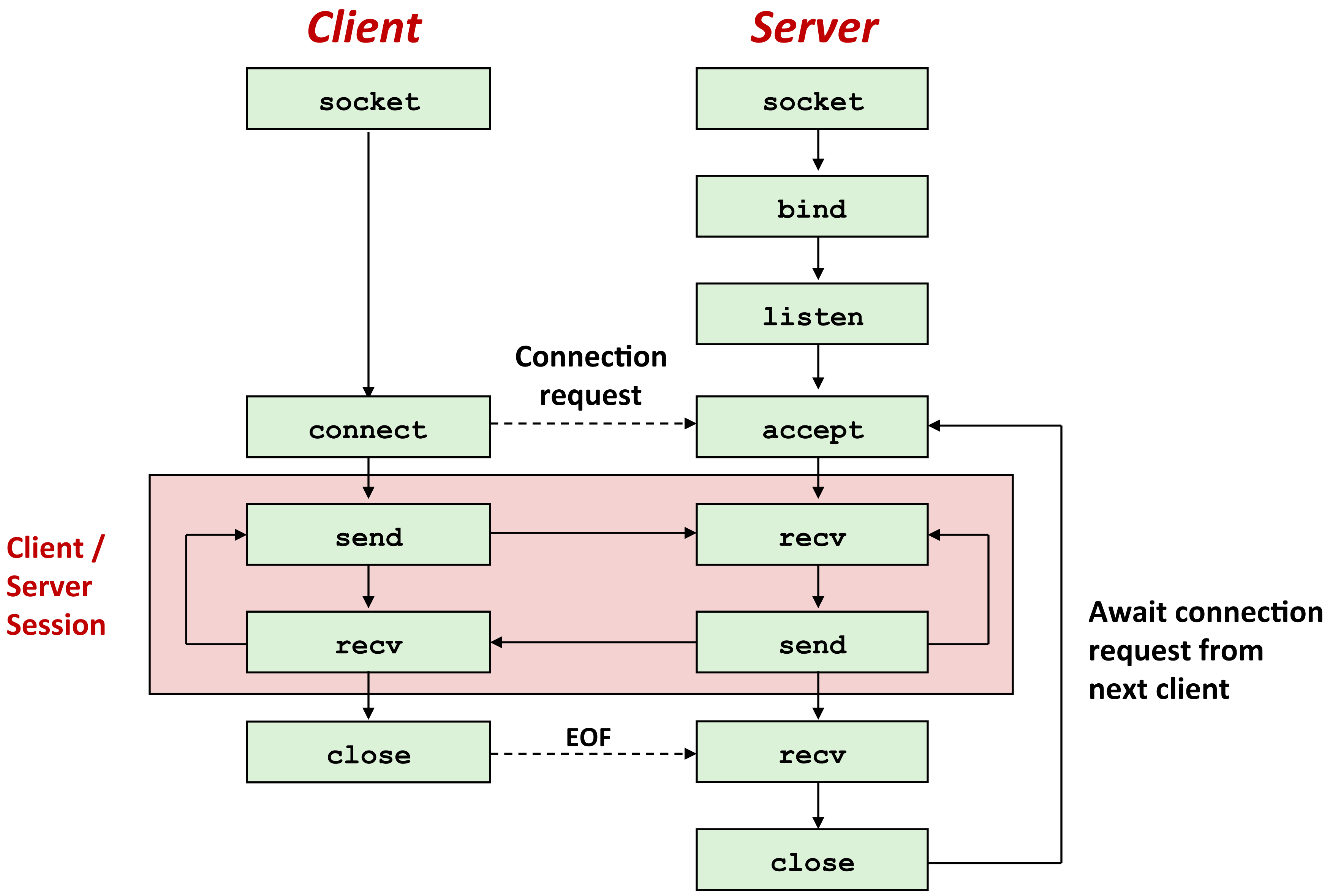
Packets: 6950 Displayed: 13 Marked: 0

Profile: Default

Ethical Issues

- **Packet Sniffer**
  - Program that records network traffic visible at node
  - Promiscuous mode: Record traffic that does not have this host as source or destination

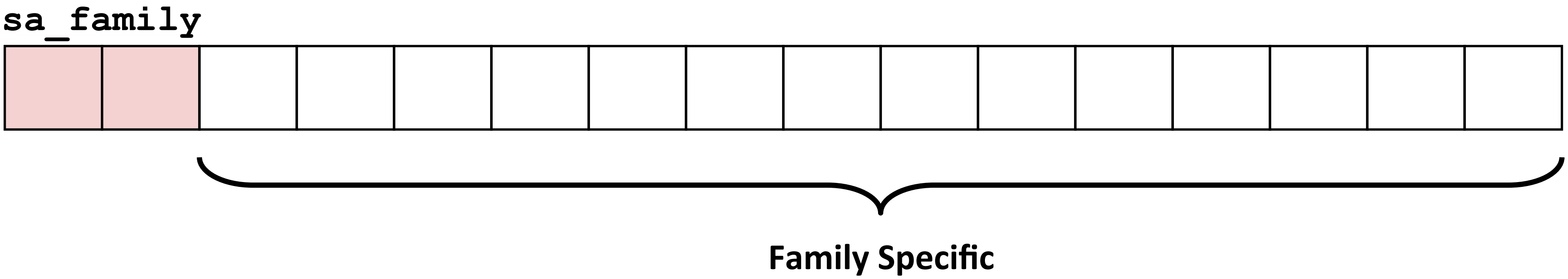
Overview of the Sockets Interface



Socket Address Structures

- **Generic socket address:**
  - For address arguments to **`connect`**, **`bind`**, and **`accept`**
  - Necessary only because C did not have generic (**`void *`**) pointers when the sockets interface was designed

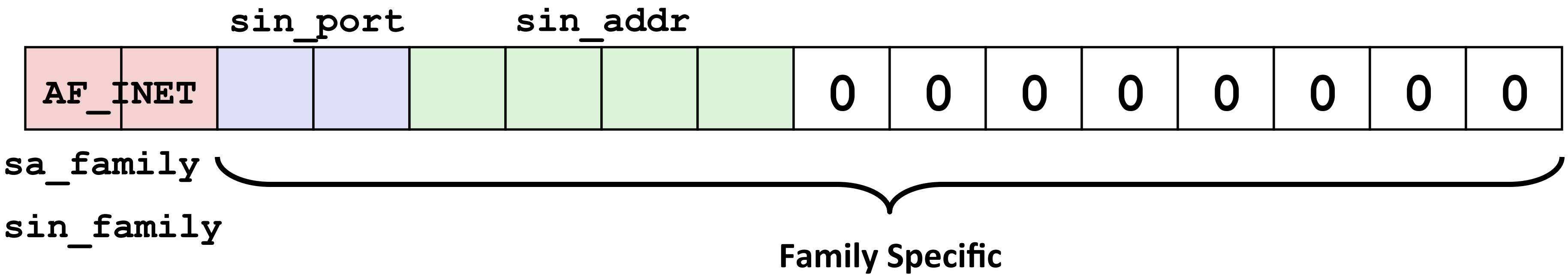
```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.   */
};
```



Socket Address Structures

- **Internet-specific socket address:**
  - Must cast (**`sockaddr_in *`**) to (**`sockaddr *`**) for **`connect`**, **`bind`**, and **`accept`**

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```



Creating a TCP socket

- **`socket` creates a socket descriptor on the client**
  - Just allocates & initializes some internal data structures
  - **`AF_INET`**: indicates that the socket is associated with Internet protocols
  - **`SOCK_STREAM`**: selects a reliable byte stream connection
    - provided by TCP

```
int clientfd;  /* socket descriptor */
clientfd = socket(AF_INET, SOCK_STREAM, 0);
if (clientfd < 0)
    die_with_error(“socket() error”);
    /* check errno for cause of error */

... <more>
```

Closing a TCP socket

```
close(clientfd);
```

Client: Connecting to a TCP socket

- **Finally the client creates a connection with the server**
  - Client process suspends (blocks) until the connection is created
  - After resuming, the client is ready to begin exchanging messages with the server via Unix I/O calls on descriptor **`clientfd`**

```
int clientfd;                     /* socket descriptor */
struct sockaddr_in serveraddr;    /* server address */
...
  /* Establish a connection with the server */
int result;
result = connect(clientfd, (struct sockaddr *)&serveraddr,
  sizeof(serveraddr));
if (result < 0)
    die_with_error("recv() failed");
```

Client: Send string to Server

- Arguments
  - Sock – socket file descriptor
  - Input_string - string to send
  - Length of string
  - Optional flags

```
send(clientfd,(void *)input_string, strlen(input_string), 0);
```

Client: Receive message from server

- Arguments
  - sock – socket file descriptor
  - received_string - string to send
  - Size of the received_string variable
  - Optional flags
- Returns
  - Number of bytes received

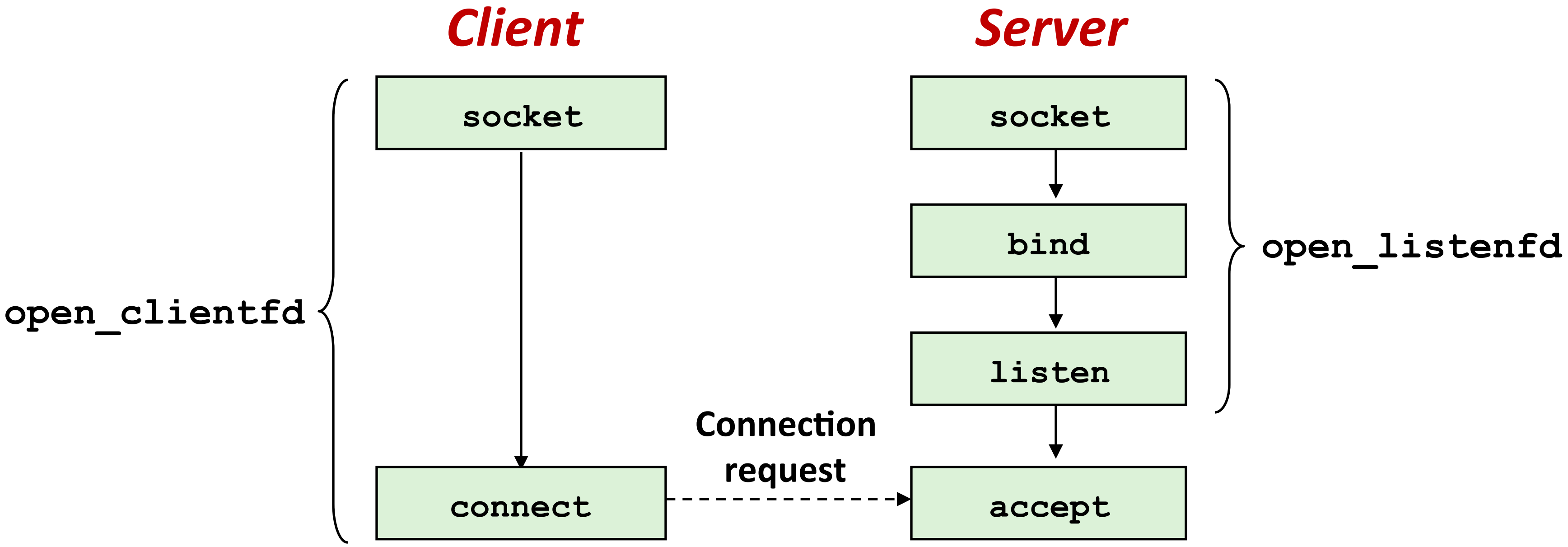
```
received_bytes = recv(clientfd,(void *)received_string,
  sizeof(received_string), 0);

if (received_bytes < 0)
  die_with_error("recv() failed");

received_string[sizeof(received_string)-1] = ‘\0’;
```

Server: Receive message from client

- Arguments
  - sock – socket file descriptor
  - received_string - string to send
  - Size of the received_string variable
  - Optional flags
- Returns
  - Number of bytes received

```
received_bytes = recv(sock,(void *)received_string,
  sizeof(received_string), 0);

if (received_bytes < 0)
  die_with_error("recv() failed");

received_string[sizeof(received_string)-1] = ‘\0’;
```

Overview of the Sockets Interface



- **Office Telephone Analogy for Server**
  - Socket: Buy a phone
  - Bind: Tell the local administrator what number you want to use
  - Listen: Plug the phone in
  - Accept: Answer the phone when it rings

Server: `Binding to an address`

- `bind` associates the socket with the socket address we just created

```
int listenfd;                    /* listening socket */
struct sockaddr_in serveraddr; /* server’s socket addr */

...
/* listenfd will be an endpoint for all requests to port
   on any IP address for this host */
int bind_result =
bind(listenfd, (struct sockaddr *)&serveraddr,sizeof(serveraddr));

if (bind_result < 0)
      die_with_error(“bind() failed”);
```

Server: Listen for connections

- listen indicates that this socket will accept connection (connect) requests from clients

```
int listenfd; /* listening socket */

...
/* Make it a listening socket ready to accept connection requests */
int listen_result = listen(listenfd, 5);
if (listen_result < 0)
  die_with_error(“listen failed”);
```

Echo Server: Main Loop

- The server loops endlessly, waiting for connection requests, then reading input from the client, and echoing the input back to the client.

```
main() {

   /* create and configure the listening socket */

   while(1) {
      /* accept(): wait for a connection request */
      /* Read and echo input lines from client */
      /* close(): close the connection */
   }
}
```

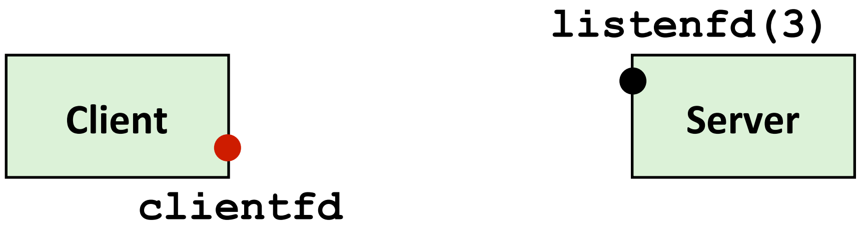
Echo Server: `accept`

- **`accept()` blocks waiting for a connection request**

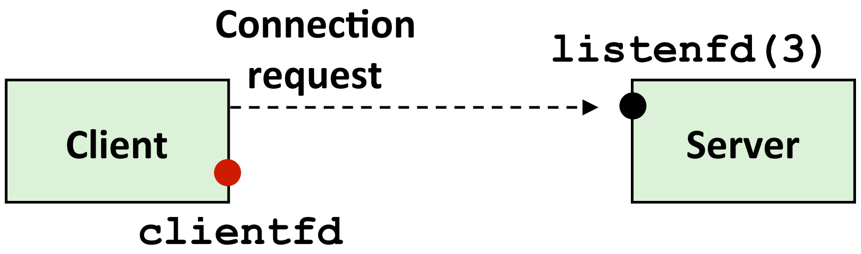
```
int listenfd; /* listening descriptor */
int connfd;   /* connected descriptor */
struct sockaddr_in clientaddr;
int clientlen;

clientlen = sizeof(clientaddr);
connfd = accept(listenfd, (struct sockaddr *)&clientaddr,
&clientlen);
```

- **`accept` returns a *connected descriptor* (`connfd`) with the same properties as the *listening descriptor* (`listenfd`)**
  - Returns when the connection between client and server is created and ready for I/O transfers
  - All I/O with the client will be done via the connected socket
- **`accept` also fills in client's IP address**

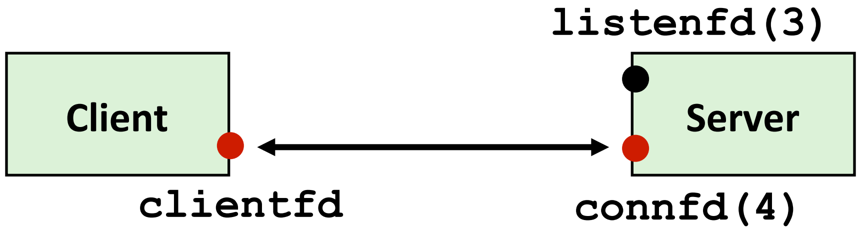
Echo Server: `accept` Illustrated



1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`



2. Client makes connection request by calling and blocking in `connect`



3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`

For More Information

- **W. Richard Stevens, "Unix Network Programming: Networking APIs: Sockets and XTI", Volume 1, Second Edition, Prentice Hall, 1998**
  - THE network programming bible
- **Unix Man Pages**
  - Good for detailed information about specific functions
- **Complete versions of the echo client and server are developed in the text**
  - Updated versions linked to course website
  - Feel free to use this code in your assignments