UIC Department of Computer Science
UNIVERSITY OF ILLINOIS AT CHICAGO
COLLEGE OF ENGINEERING

CS450 – Introduction to Networking
Lecture 7 – Client/Server Model & Assignment 2

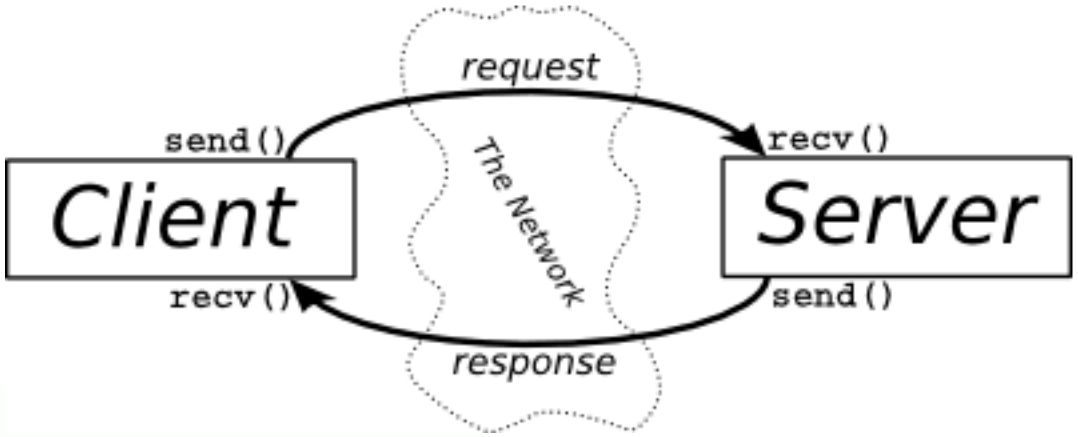
Phu Phung

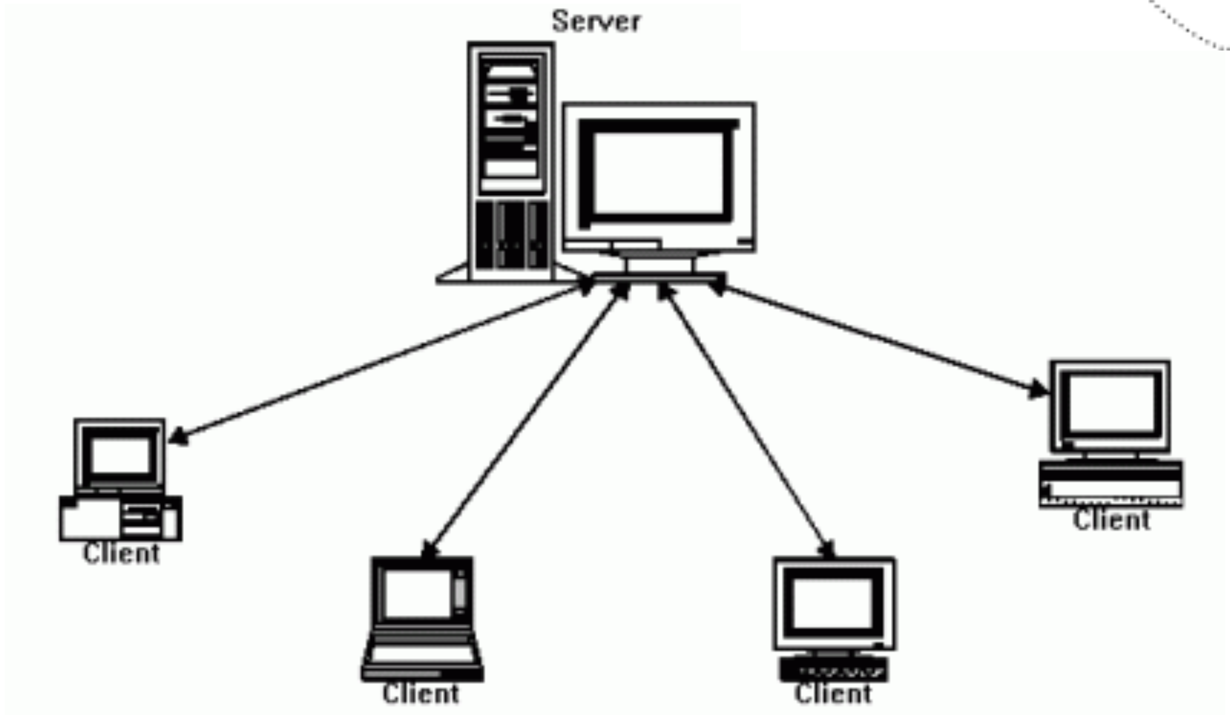
January 28, 2015

Office hours (changed)

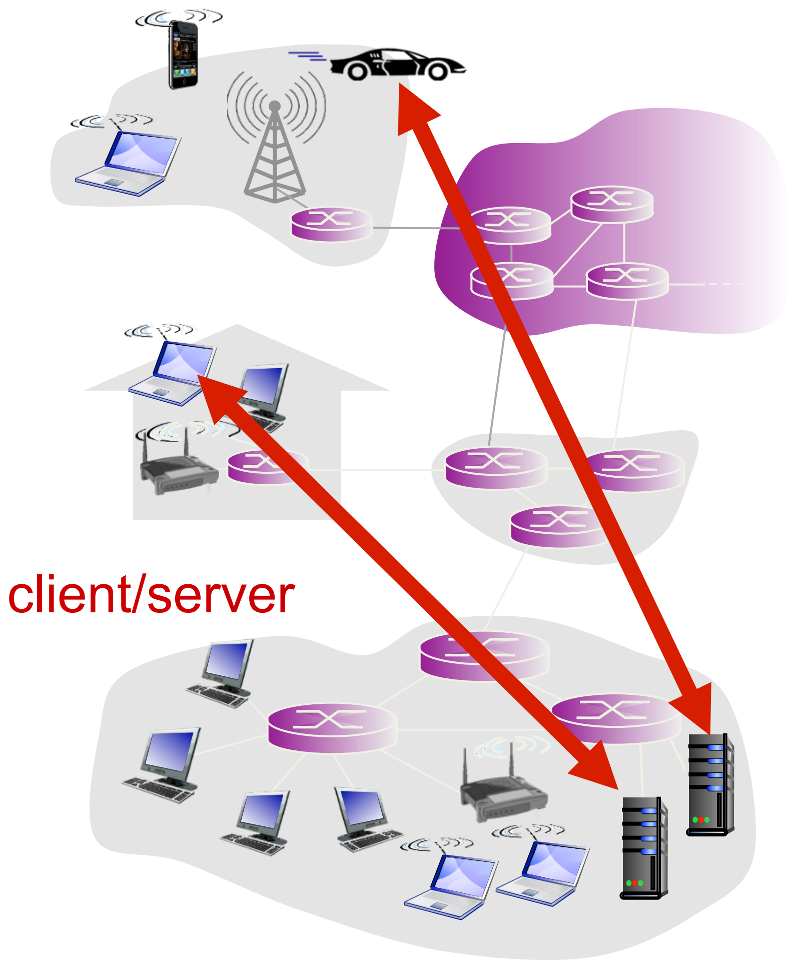
- Phu Phung (instructor)
  - Mondays 11AM-1PM
  - Office: SEO 1216
- Xiang Huo (TA)
  - Fridays 11AM-1PM
  - Office: SEO 1306

Simple client-server model





Client-server architecture



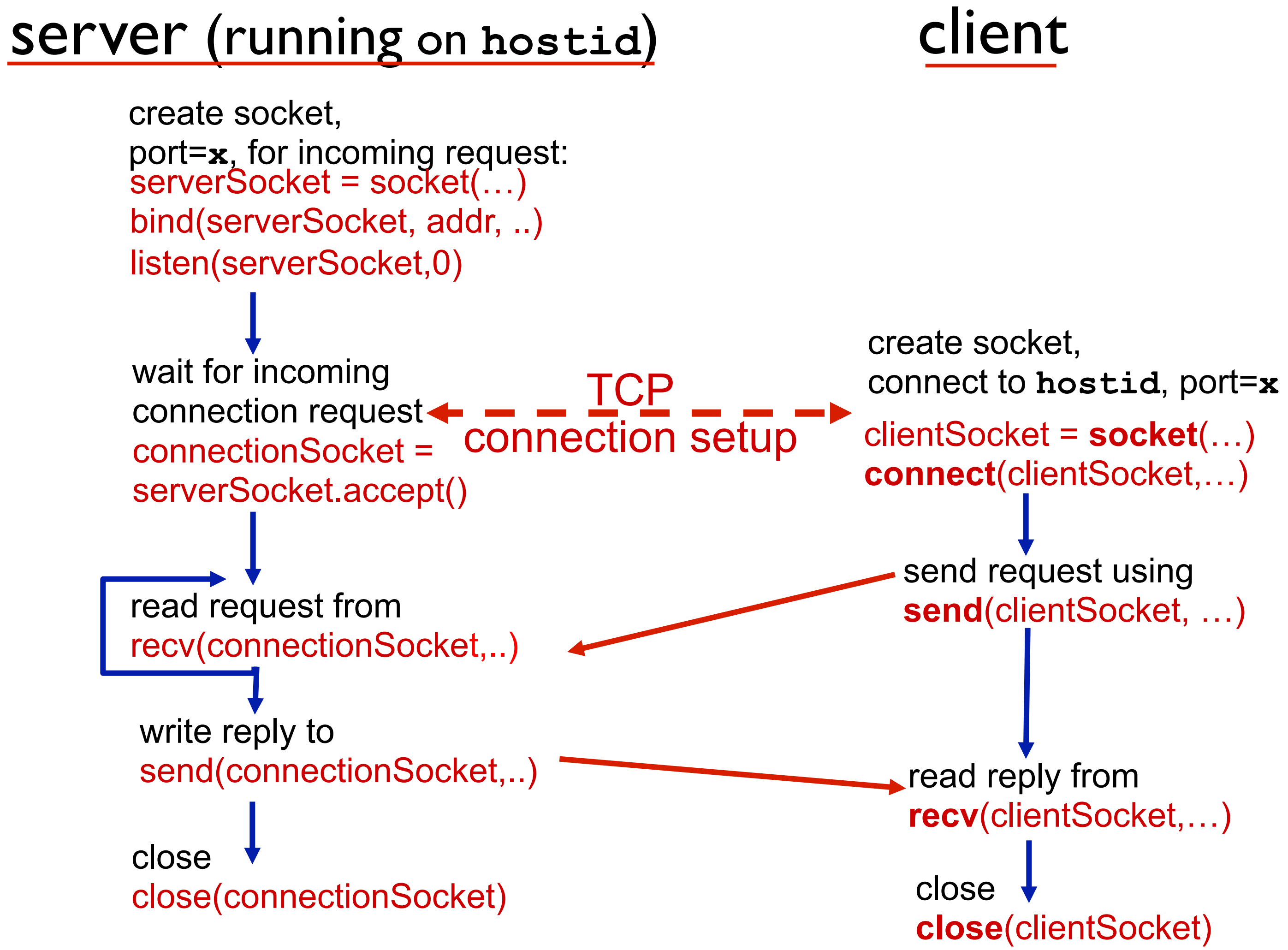
server:

- always-on host/service
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Client/server socket interaction: TCP



Beej's Guide to Network Programming

http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html

Server programming

- Prepare address (with port)
- Create socket
- Bind the socket to the address
- Listen on the port
- Accept a new connection
  - Create a new socket to communicate with client

Prepare server address

```
int status;
struct addrinfo hints;
struct addrinfo *servinfo;  // will point to the results

memset(&hints, 0, sizeof hints); // make sure the struct is empty
hints.ai_family = AF_UNSPEC;     // don't care IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
hints.ai_flags = AI_PASSIVE;     // fill in my IP for me

if ((status = getaddrinfo(NULL, "server_port", &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
    exit(1);
}
```

Replace with user input

Create socket and bind to port

```
// loop through all the results and bind to the first we can
for(p = servinfo; p != NULL; p = p->ai_next) {
  if ((server_socket=socket(p->ai_family, p->ai_socktype, p->ai_protocol))<0) {
    perror("server: socket");
    continue;}
  int yes=1;
  // lose the pesky "Address already in use" error message
  if (setsockopt(server_socket,SOL_SOCKET,SO_REUSEADDR, &yes,sizeof(int)) == -1){
            perror("setsockopt");
            exit(1); }
  if (bind(server_socket, p->ai_addr, p->ai_addrlen) == -1) {
            close(server_socket);
            perror("server: cannot bind");
            continue;}
        break;
}
```

Listen and wait for connections

```
#define BACKLOG 10  //how many pending connections queue will hold
//server_socket was created previously
if (listen(server_socket, BACKLOG)<0){
    perror("Error listening for connection");
    exit(1);
}
```

Accept and handle a client request

```
struct sockaddr_in remote_addr;
unsigned int socklen = sizeof(remote_addr);
while(1) {
  int new_socket;
  new_socket= accept(server_socket, (struct sockaddr*)&remote_addr,
                          &socklen);
  if(new_socket < 0) {
            perror("Error accepting connection");
            exit(1)
  }
  pthread_t client; //#include <pthread.h>
  pthread_create(&client,0,handle_client,(void*)new_socket);
}
//void* handle_client(void *sock);
```

Assignment 2 – Multi-threaded web server

- Handle HTTP request and construct HTTP response
- Input: <port_number> <directory>

  ```
  ./hw2 8080 WWW
  ```
- Deadline Sunday Feb 15, 11:59 PM
  - 1 bonus point for submission 36 hours before deadline
  - Firm deadline -> Start early

Assignment 2 – Cases to handle

- non-existing file: return error 404 with a readable error message, just like a webserver would
- html, text, jpeg, gif and png files should all display properly. Return the proper HTTP Content-type header
- if the requested path is a directory, you should handle request as if it was for the file index.html inside that directory
- use multithreading with pthreads to handle concurrent incoming connections
- the webserver should response with a list of saved files when the user requests a directory that does not contain an index.html file

HTTP response revisit

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html\r\n
\r\n
data data data data data ...
```

Next lecture

- DNS
  - Readings 2.5