Interaction and Communication between Programs

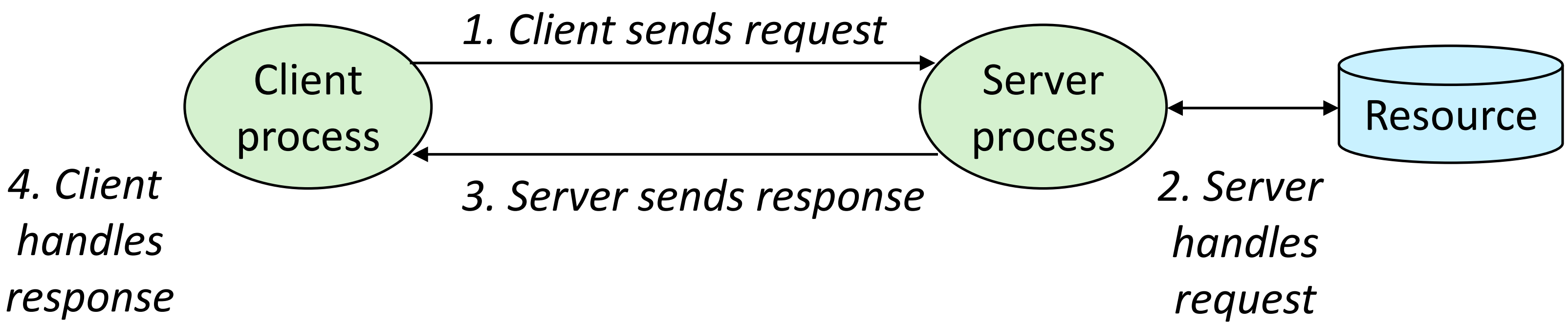
Network Programming

Network Programming

- **Client-Server Model and Computer Networks**
- Global IP Internet
  - naming
  - IPv4, IPv6
  - DNS
  - Internet connection
- Programmer's View
  - The sockets interface
  - Establishing an Internet connection
  - Copy data over an Internet connection

Acknowledgement: slides based on the cs:app2e material

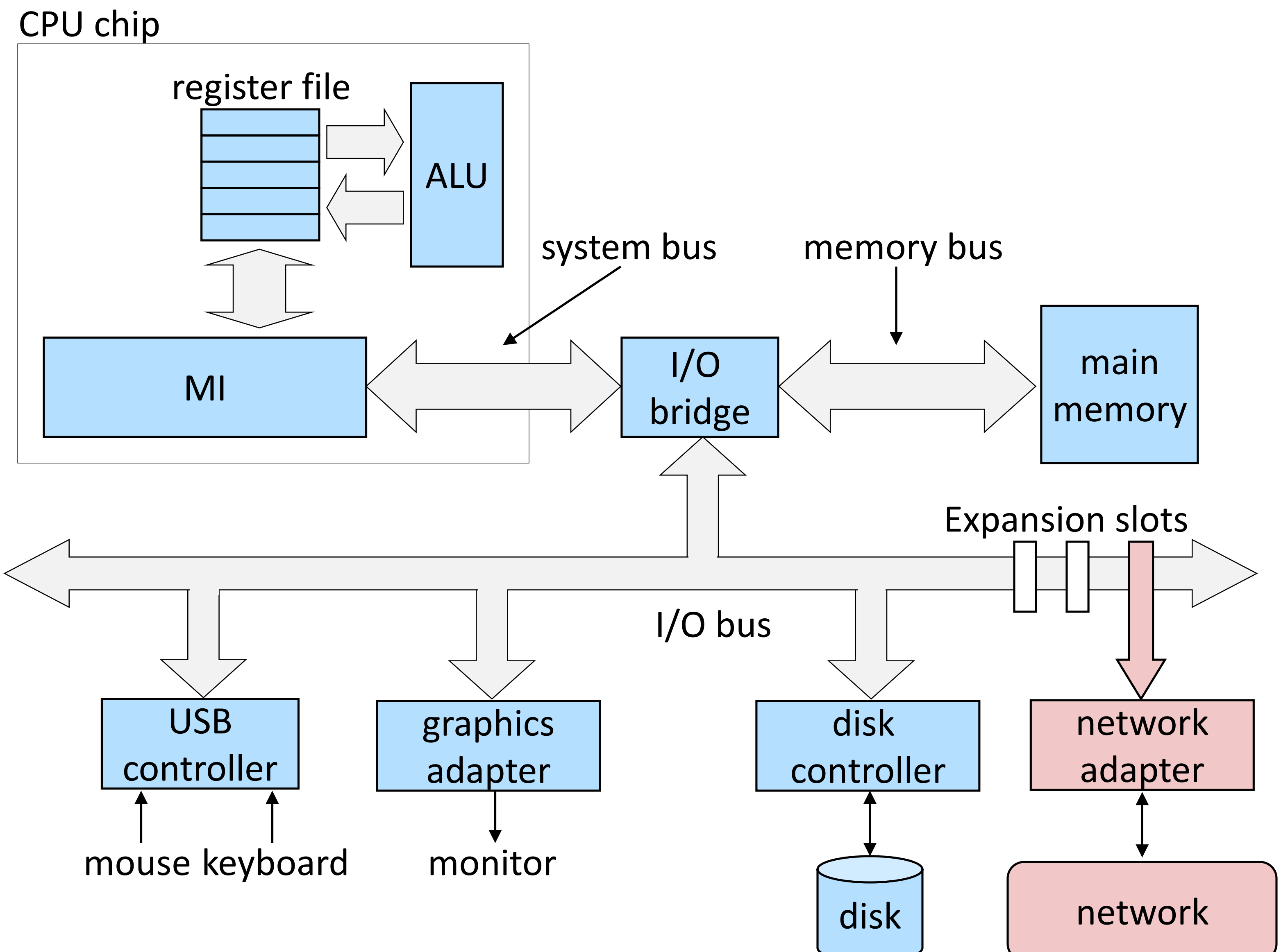
A Client-Server Transaction



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

- Most network applications are based on the client-server model:
  - A server process and one or more client processes
  - Server manages some resource
  - Server provides service by manipulating resource for clients
  - Server activated by request from client (vending machine analogy)

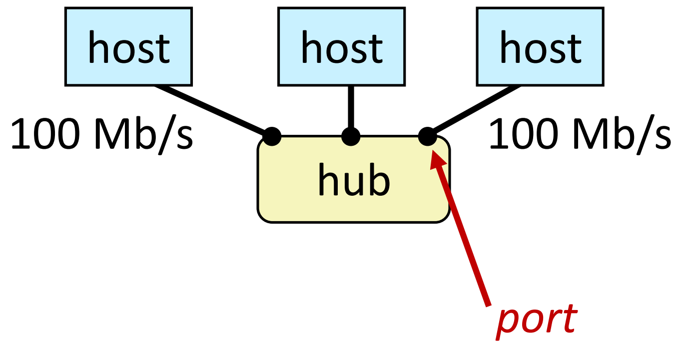
Hardware Organization of a Network Host



Computer Networks

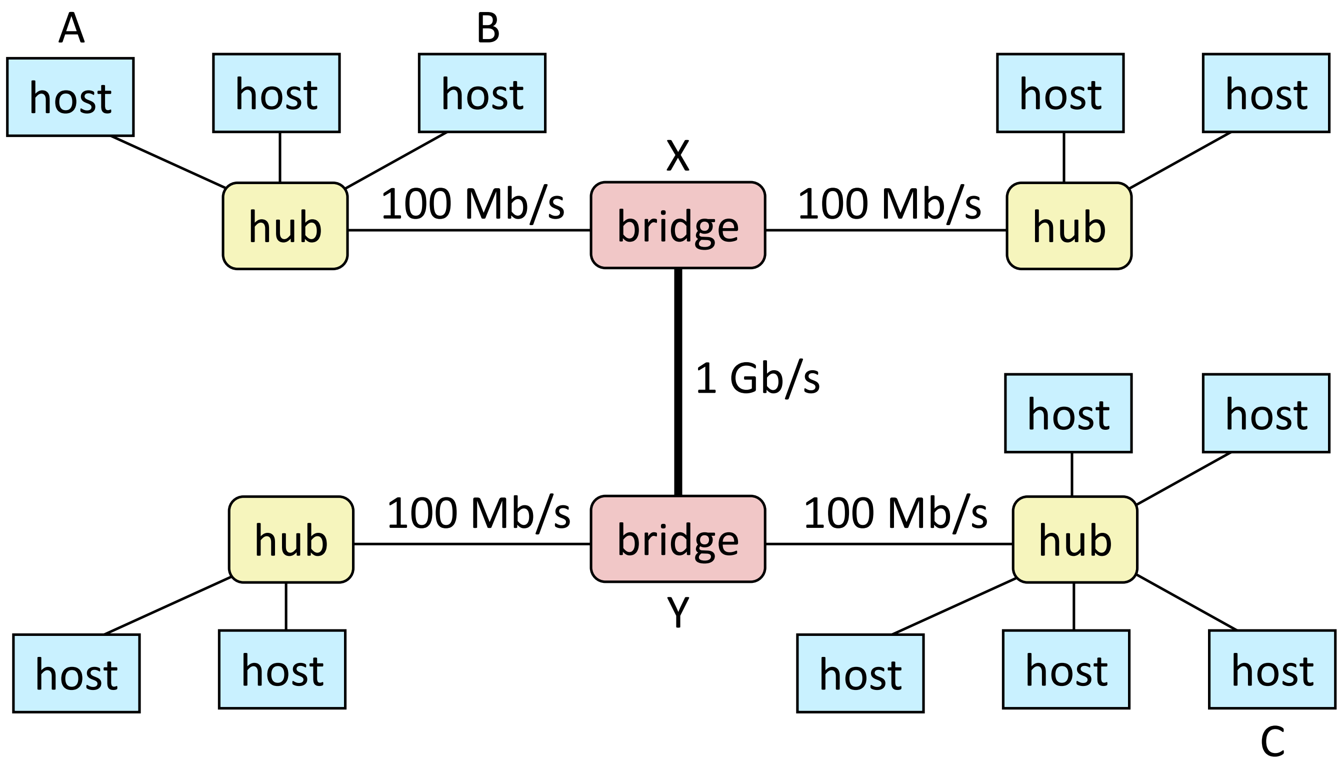
- A network is a hierarchical system of boxes and wires organized by geographical proximity
  - SAN (System Area Network) spans cluster or machine room
    - Switched Ethernet, Quadrics QSW, …
  - LAN (Local Area Network) spans a building or campus
    - Ethernet is most prominent example
  - WAN (Wide Area Network) spans country or world
    - Typically high-speed point-to-point phone lines
- An internetwork (internet) is an interconnected set of networks
  - The Global IP Internet (uppercase "I") is the most famous example of an internet (lowercase "i")
- Let's see how an internet is built from the ground up

Lowest Level: Ethernet Segment



- Ethernet segment consists of a collection of hosts connected by wires (twisted pairs) to a hub
- Spans room or floor in a building
- Operation
  - Each Ethernet adapter has a unique 48-bit address (MAC address)
    - E.g., 00:16:ea:e3:54:e6
  - Hosts send bits to any other host in chunks called frames
  - Hub slavishly copies each bit from each port to every other port
    - Every host sees every bit
    - Note: Hubs are on their way out. Bridges (switches, routers) became cheap enough to replace them (means no more broadcasting)

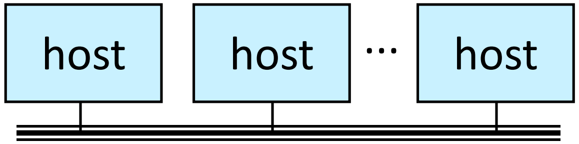
Next Level: Bridged Ethernet Segment



- Spans building or campus
- Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port

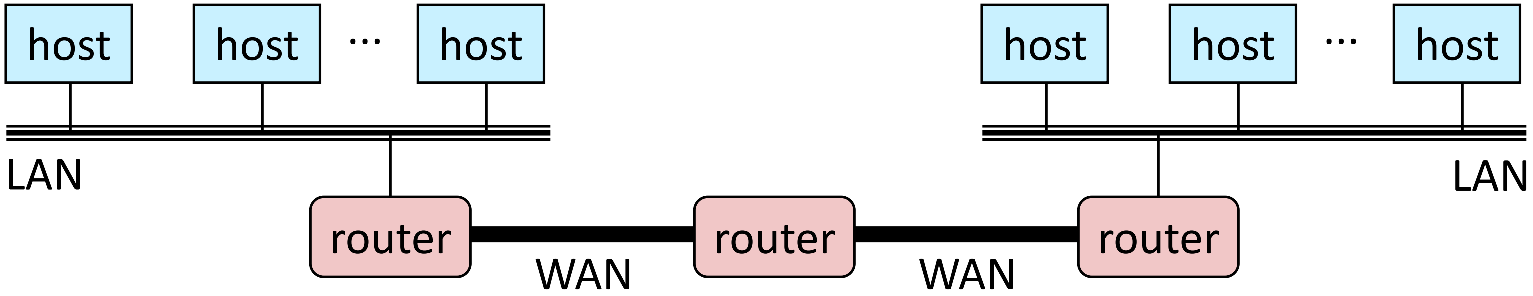
Conceptual View of LANs

- For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:



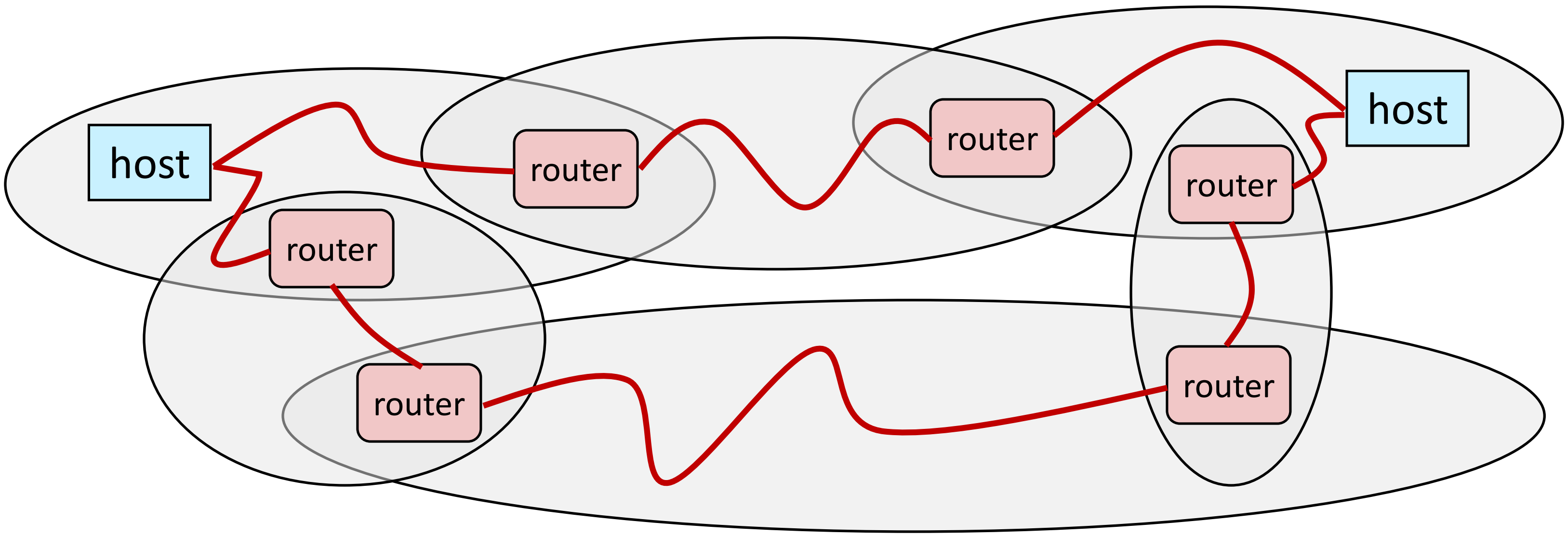
Next Level: internets

- Multiple incompatible LANs can be physically connected by specialized computers called routers
- The connected networks are called an internet



LAN 1 and LAN 2 might be completely different, totally incompatible (e.g., Ethernet and Wifi, 802.11, T1-links, DSL, …)*

Logical Structure of an internet



- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router forms bridge from one network to another
  - Different packets may take different routes

The Notion of an internet Protocol

- How is it possible to send bits across incompatible LANs and WANs?

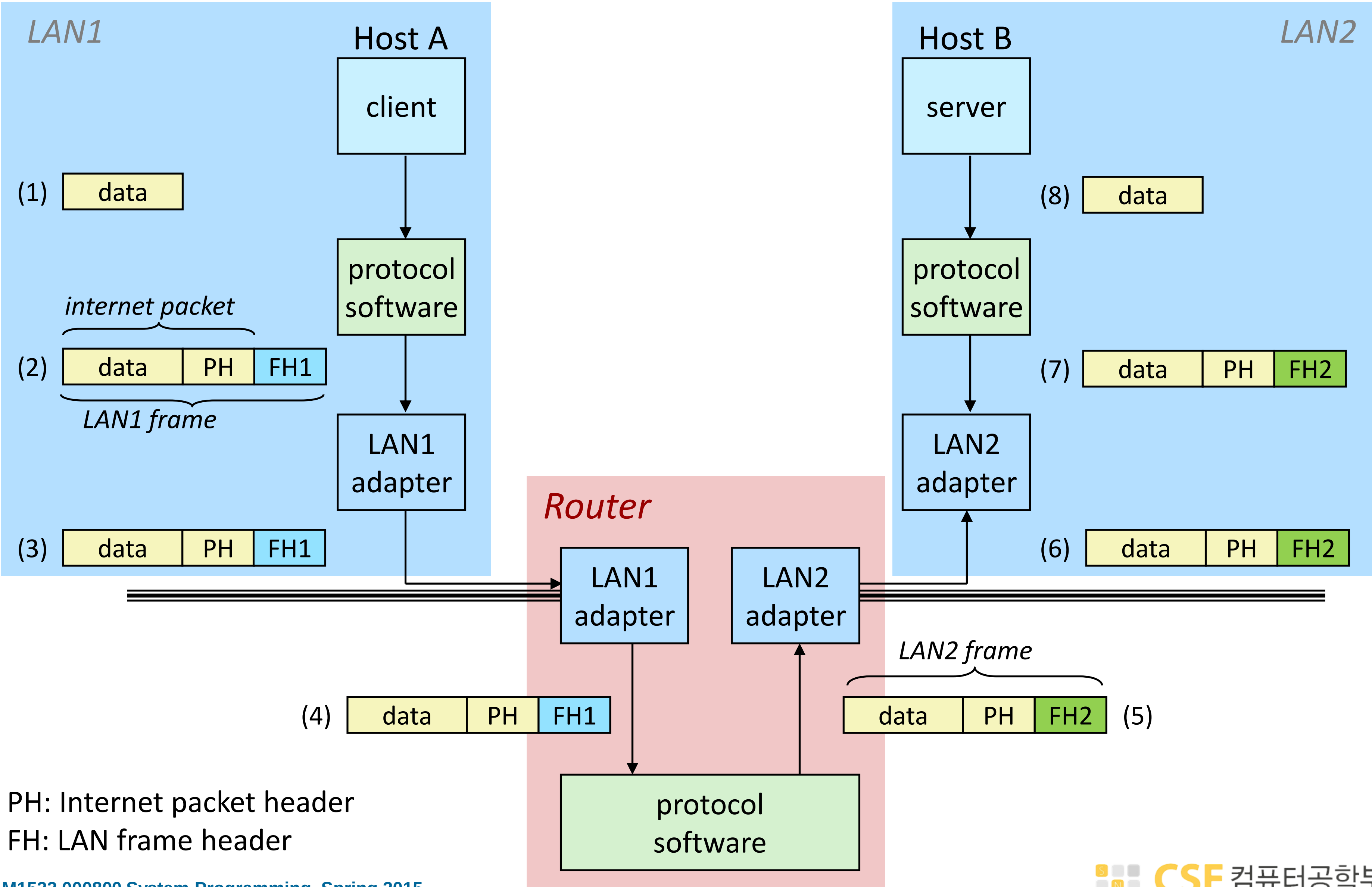
- Solution:
  - protocol software running on each host and router
  - smoothes out the differences between the different networks

- Implements an internet protocol (i.e., set of rules)
  - governs how hosts and routers should cooperate when they transfer data from network to network
  - TCP/IP is the protocol for the global IP Internet

What Does an internet Protocol Do?

- Provides a naming scheme
  - An internet protocol defines a uniform format for host addresses
  - Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it
- Provides a delivery mechanism
  - An internet protocol defines a standard transfer unit (packet)
  - Packet consists of header and payload
    - Header: contains info such as packet size, source and destination addresses
    - Payload: contains data bits sent from source host

Transferring Data Over an internet



PH: Internet packet header
FH: LAN frame header

Other Issues

- We are glossing over a number of important questions:
  - What if different networks have different maximum frame sizes? (segmentation)
  - How do routers know where to forward frames?
  - How are routers informed when the network topology changes?
  - What if packets get lost?

- These (and other) questions are addressed by the area of systems known as computer networking

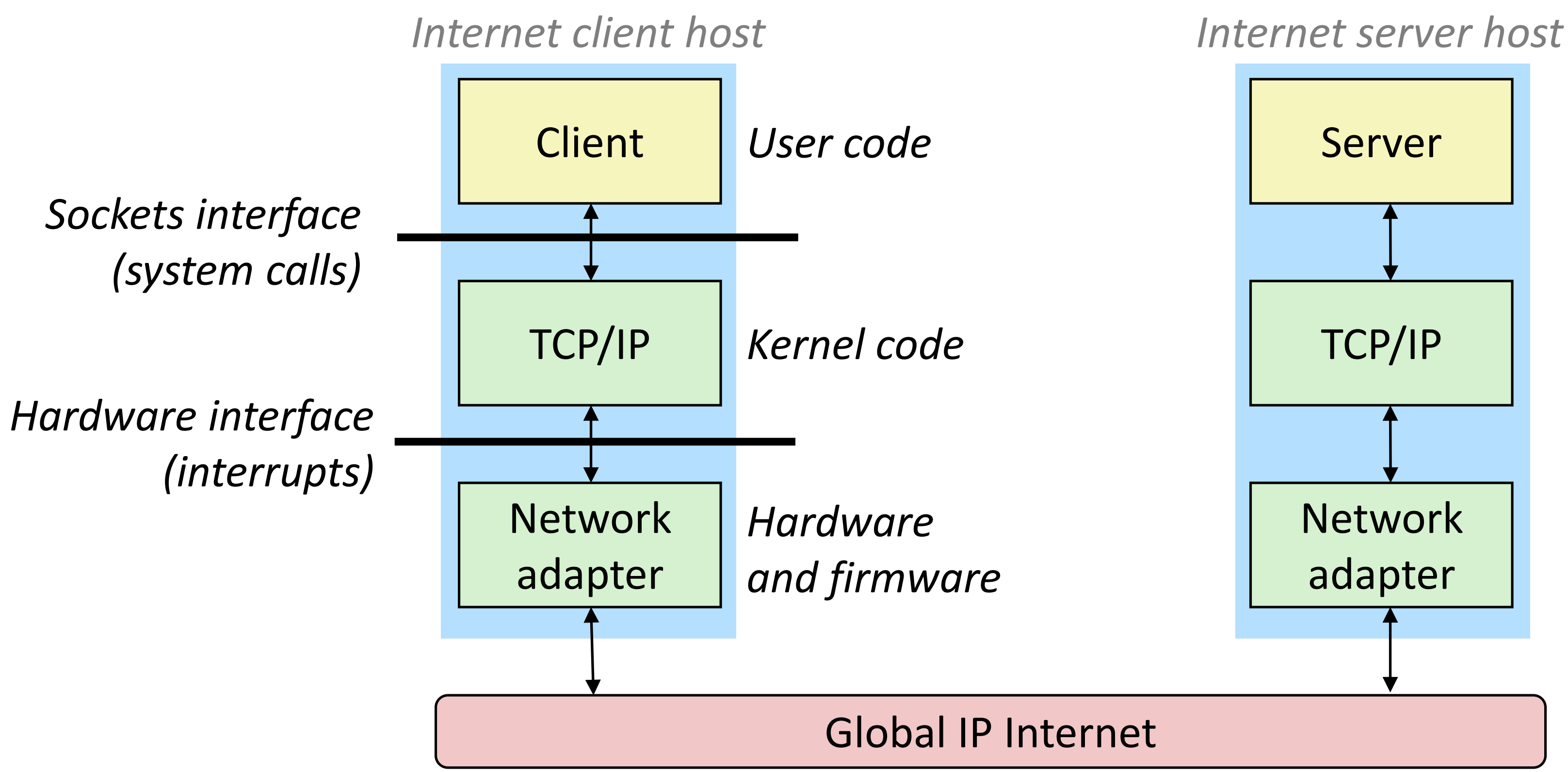
Network Programming

- Client-Server Model and Computer Networks
- **Global IP Internet**
  - **naming**
  - **IPv4, IPv6**
  - **DNS**
  - **Internet connection**
- Programmer's View
  - The sockets interface
  - Establishing an Internet connection
  - Copy data over an Internet connection

Global IP Internet

- Most famous example of an internet
- Based on the TCP/IP protocol family
  - IP (Internet protocol) :
    - Provides basic naming scheme and unreliable delivery capability of packets (datagrams) from host-to-host
  - UDP (Unreliable Datagram Protocol)
    - Uses IP to provide unreliable datagram delivery from process-to-process
  - TCP (Transmission Control Protocol)
    - Uses IP to provide reliable byte streams from process-to-process over connections
- Accessed via a mix of Unix file I/O and functions from the sockets interface

Hardware and Software Organization of an Internet Application



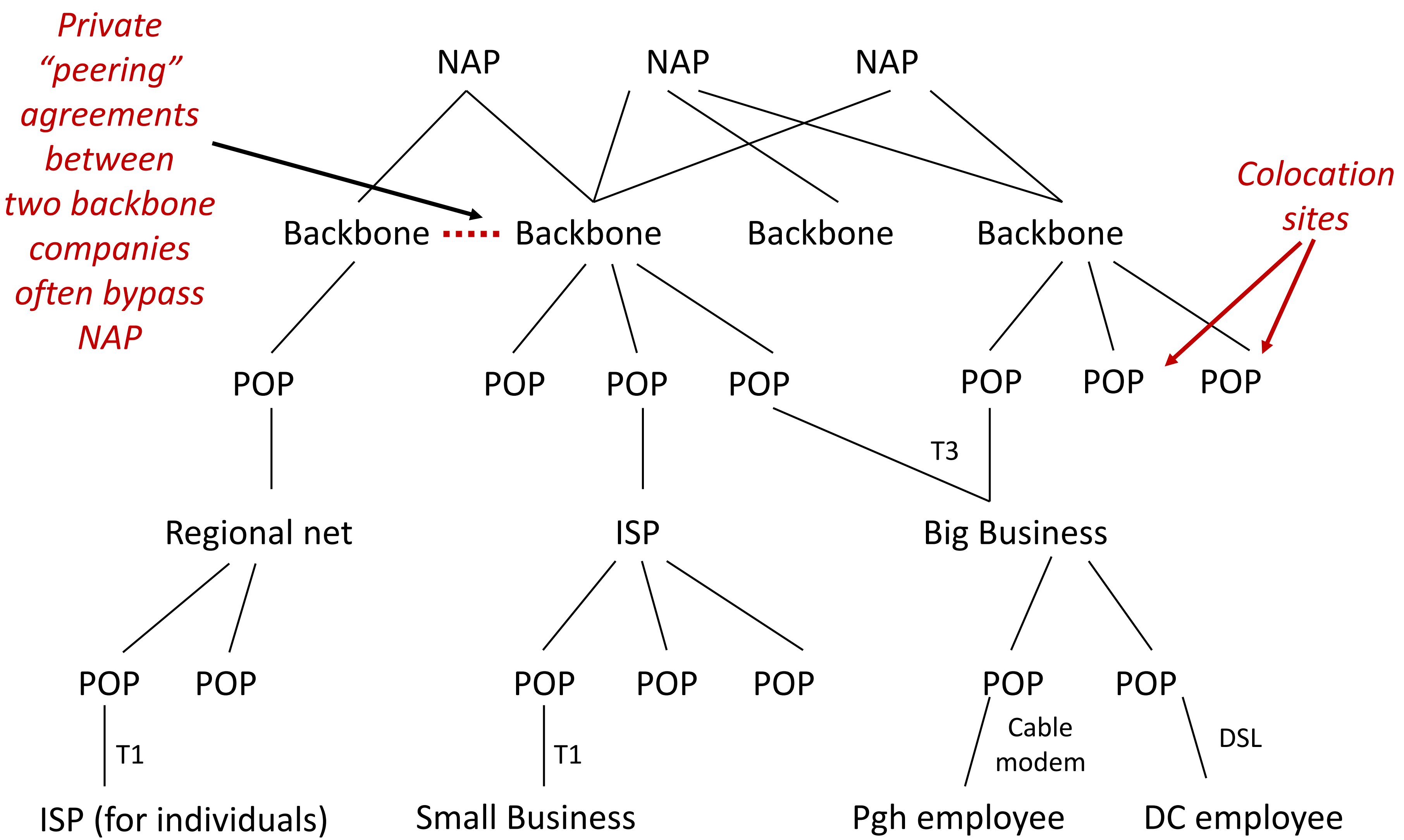
Basic Internet Components

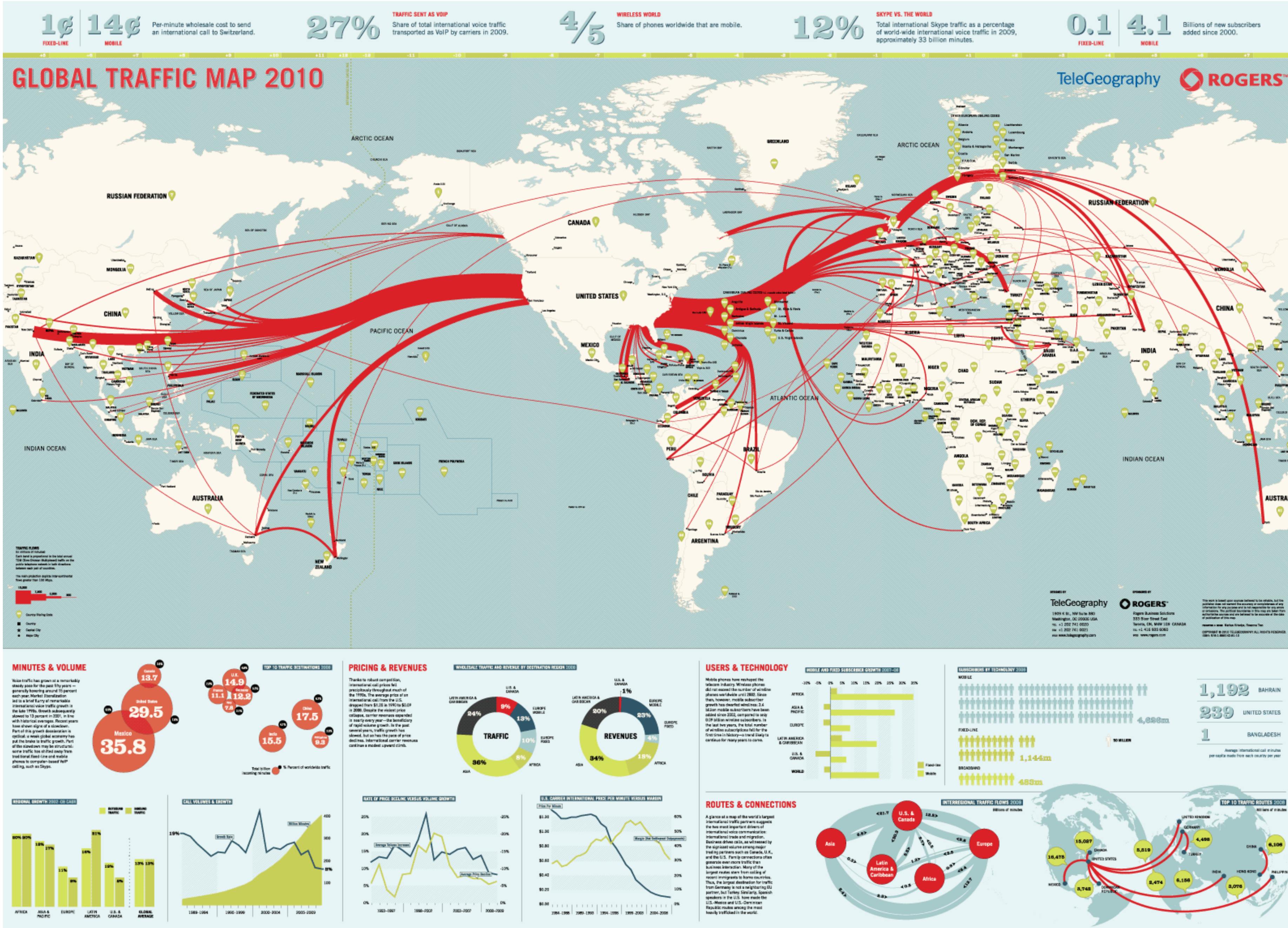
- Internet backbone:
  - collection of routers (nationwide or worldwide) connected by high-speed point-to-point networks
- Network Access Point (NAP):
  - router that connects multiple backbones (often referred to as peers)
- Regional networks:
  - smaller backbones that cover smaller geographical areas (e.g., cities or states)
- Point of presence (POP):
  - machine that is connected to the Internet
- Internet Service Providers (ISPs):
  - provide dial-up or direct access to POPs

NAP-Based Internet Architecture

- NAPs link together commercial backbones provided by companies such as AT&T and Worldcom
- Currently in the US there are about 50 commercial backbones connected by ~12 NAPs (peering points)
- Similar architecture worldwide connects national networks to the Internet

Internet Connection Hierarchy





Source: https://www.telegeography.com/telecom-maps/global-traffic-map.1.html

History and Forecast of Internet Traffic

Year	Global Internet Traffic	4TB HardDisks needed
1992	100 GB / day	1 every 40 days
1997	100 GB / hour	1 every 2 days
2002	100 GB / second	2’160 per day
2007	2000 GB / second	30 per minute
2013	28’875 GB / second	428 per minute
2018	50,000 GB / second	750 per minute

Source: Cisco VNI, 2014

Naming and Communicating on the Internet

- Original Idea
  - Every node on Internet would have unique IP address
    - Everyone would be able to talk directly to everyone
  - No secrecy or authentication
    - Messages visible to routers and hosts on same LAN
    - Possible to forge source field in packet header

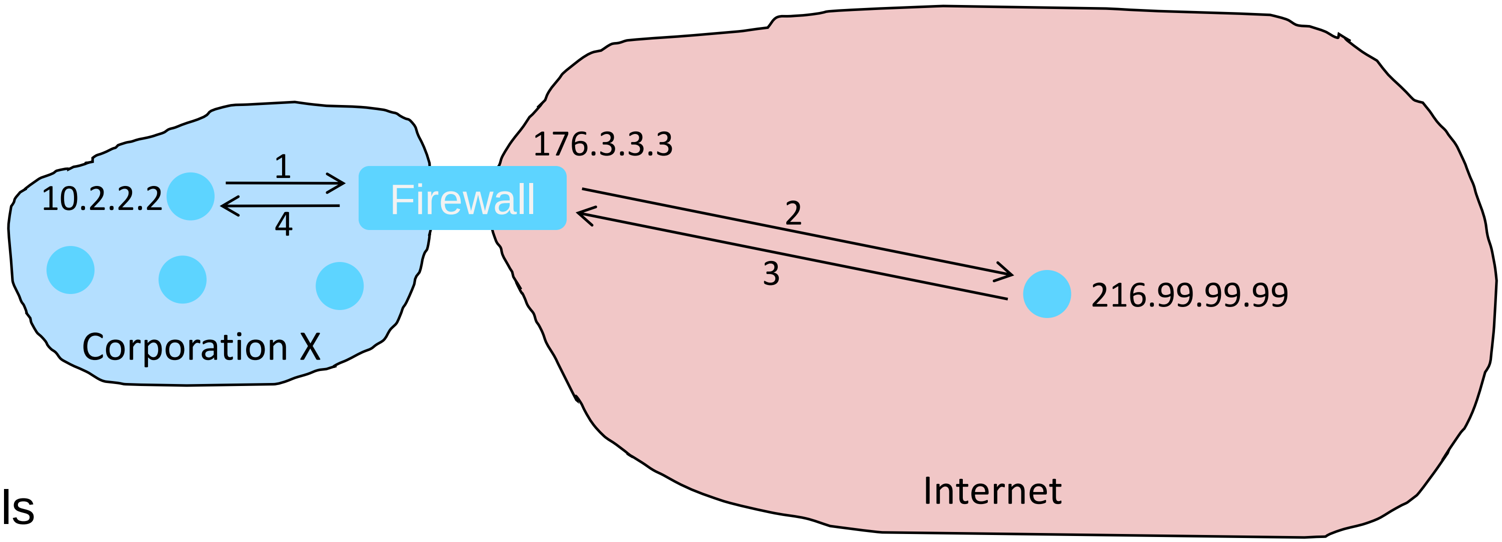
- Shortcomings
  - There aren't enough IP addresses available
  - Don't want everyone to have access or knowledge of all other hosts
  - Security issues mandate secrecy & authentication

Evolution of Internet: Naming

- Dynamic address assignment
  - Most hosts don't need to have known address
    - Only those functioning as servers
  - DHCP (Dynamic Host Configuration Protocol)
    - Local ISP assigns address for temporary use

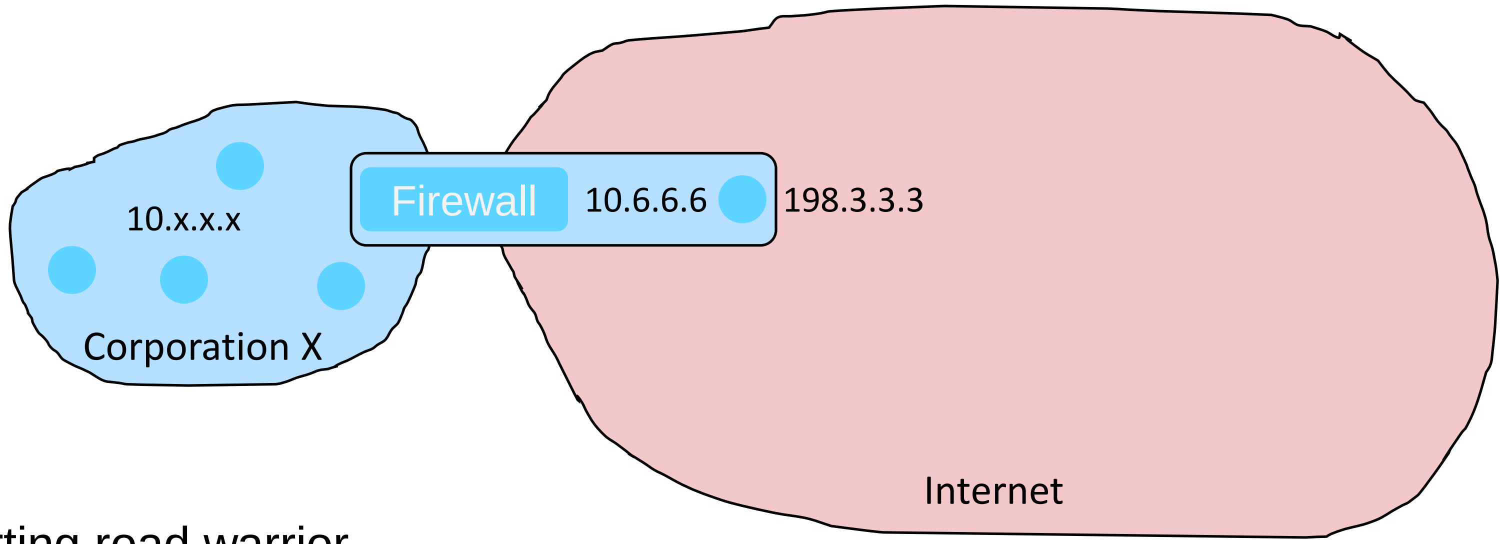
- Example:
  - The CSAP server at SNU (wired connection)
    - IP address 147.46.174.108
    - assigned statically
  - My desktop at work
    - IP address 192.168.1.3
    - assigned dynamically by a router
    - only valid within (“behind”) the office router

Evolution of Internet: Firewalls



- Firewalls
  - Hides organizations nodes from rest of Internet
  - Use local IP addresses within organization
  - For external service, provides proxy service
    - Client request: src=10.2.2.2, dest=216.99.99.99
    - Firewall forwards: src=176.3.3.3, dest=216.99.99.99
    - Server responds: src=216.99.99.99, dest=176.3.3.3
    - Firewall forwards response: src=216.99.99.99, dest=10.2.2.2

Virtual Private Networks



- Supporting road warrior
  - Employee working remotely with assigned IP address 198.3.3.3
  - Wants to appear to rest of corporation as if working internally
    - From address 10.6.6.6
    - Gives access to internal services (e.g., ability to send mail)
- Virtual Private Network (VPN)
  - Overlays private network on top of regular Internet

A Programmer's View of the Internet

- Hosts are mapped to a set of 32-bit IP addresses
  - 147.46.174.108
- The set of IP addresses is mapped to a set of identifiers called Internet domain names
  - 147.46.174.108 is mapped to csap.snu.ac.kr
- A process on one Internet host can communicate with a process on another Internet host over a connection

IP Addresses (IPv4)

- 32-bit IP addresses are stored in an IP address struct
  - IP addresses are always stored in memory in network byte order (big-endian byte order)
  - True in general for any integer transferred in a packet header from one machine to another.
    - E.g., the port number used to identify an Internet connection.

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

Useful network byte-order conversion functions ("l" = 32 bits, "s" = 16 bits)

`htonl`: convert uint32_t from host to network byte order
`htons`: convert uint16_t from host to network byte order
`ntohl`: convert uint32_t from network to host byte order
`ntohs`: convert uint16_t from network to host byte order

Dotted Decimal Notation (IPv4)

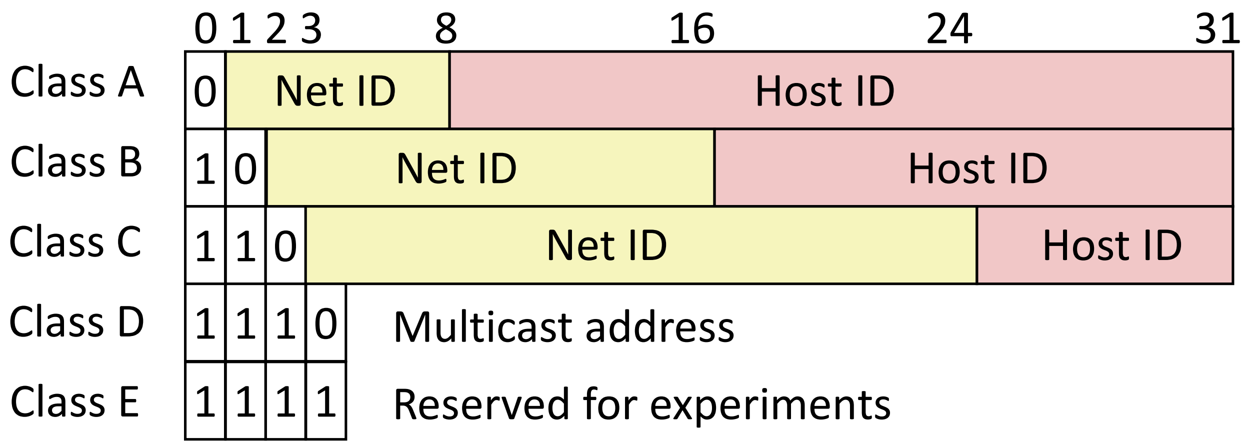
- By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period
  - IP address: 0x932EAE6C = 147.46.174.108

- Functions for converting between binary IP addresses and dotted decimal strings:
  - inet_aton: dotted decimal string → IP address in network byte order
  - inet_ntoa: IP address in network byte order → dotted decimal string

  - “n” denotes network representation
  - “a” denotes application representation

IPv4 Address Structure

- IPv4 Address space divided into classes:

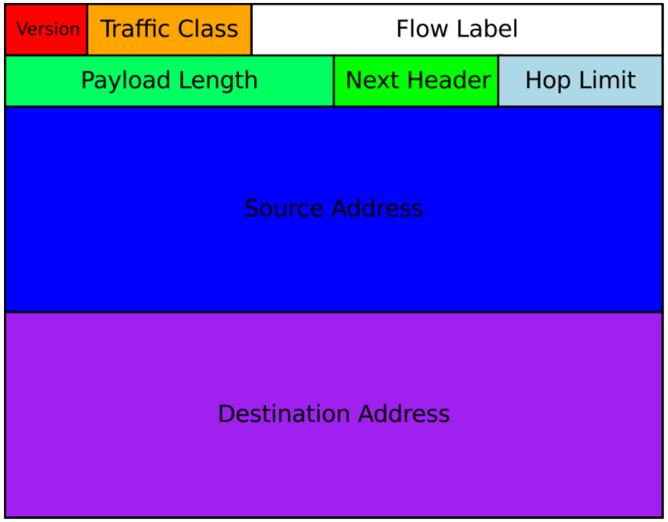


- Network ID Written in form w.x.y.z/n
  - n = number of bits in host address
  - E.g., SNU written as 147.46.0.0/16
    - Class B address
- Unrouted (private) IP addresses:
  - 10.0.0.0/8   172.16.0.0/12   192.168.0.0/16

IPv6

- Internet Protocol version 6
  - successor of IPv4
  - 128-bit addresses
    - 2^{128} = 340’282’366’920’938’463’463’374’607’431’768’211’456 addresses
    - 8 billion people: ~ 42’535’295’865’117’307’932’921’825’928 IPs/person

- (Improved) Support for
  - multicasting
  - stateless address autoconfiguration
  - IPsec
  - simplified routing
  - mobility
  - privacy



source: Wikipedia

IPv6 Dotted Decimal Notation

- By convention, each 16-bit word in a 128-bit IPv6 address is represented by its hexadecimal value and separated by a period
  - rules
    - omit leading zeroes
    - replace one or more groups of consecutive zeroes by a double quotation
  - IPv6 address: 0x20124190020300000000000020120611
  - text representation: 2012:4190:203::2012:0611

- loopback (localhost; 127.0.0.1 in IPv4)
  - IPv6: 0x00000000000000000000000000000001
  - textual representation: ::1

Internet Domain Names

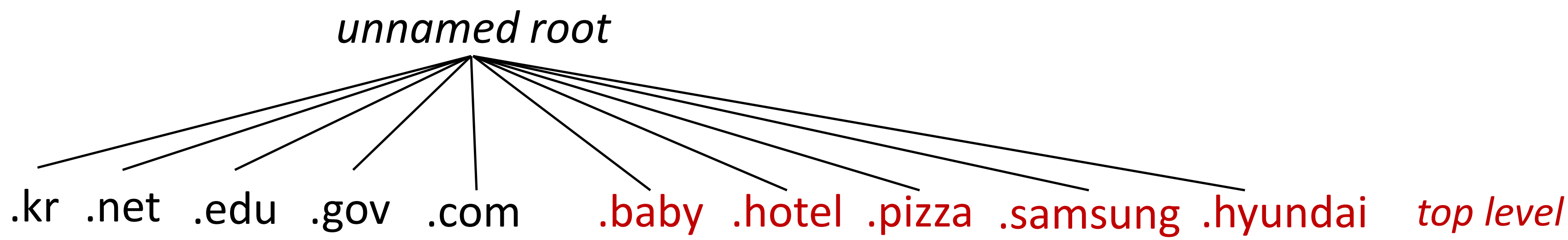
- Domain name: strings ("labels") separated by dots
  - max. length of a label: 63 characters
  - max. length of the full domain name: 253 characters
  - right-most label designates the top-level domain, each label to the left specifies a subdomain (up to max. 127 levels)
  - example: www.snu.ac.kr
- Fully-qualified vs. unqualified domain names
  - a fully qualified domain name includes a dot at the end
    - www.snu.ac.kr.
  - for unqualified domain names, in principle, the domain name resolver automatically append the system's default domain name
    - "www" inside SNU resolves (should resolve) to www.snu.ac.kr.
  - browsers automatically append the missing dot to get a FQDN
    - try http://www.snu.ac.kr./

Top-Level Domains (TLD)

- Originally, only 7 generic top-level domains (gTLD) existed (predate ICANN)
  - .com, .org, .net, .int, .edu, .gov, .mil
  - current gTLDs:
    - .com, .info, .net, .org plus (the restricted) .biz, .name, .pro
    - .edu, .gov, .int, .mil are now considered sponsored TLDs (sTLD)

- Country code top-level domains (ccTLD; predate ICANN)
  - .kr, .ch, .tv, …
  - recently: internationalized TLDs (IDN TLDs)
    - xn--3e0b707e → .한국

Generic Top-Level Domains (gTLD)



- application period closed (May 30, 2012): ~2’000 applications
- went live in November 2013, currently (May 15, 2015) >900 gTLDs have been assigned; over 5.5 mio domains using a gTLD have been registered

- so, can I register .egger and then use http://bernhard.egger for my homepage?
  - application fee to ICANN: $185’000
  - annual fee to ICANN: $25’000
  - plus operational cost
  - → um, no, thanks.

- then, who did register new gTLDs?
  - Google (101), Amazon (76), Microsoft (11)
  - Donuts (domain name company): 307

- updated list of top-level domains:
  http://data.iana.org/TLD/tlds-alpha-by-domain.txt

Domain Naming System (DNS)

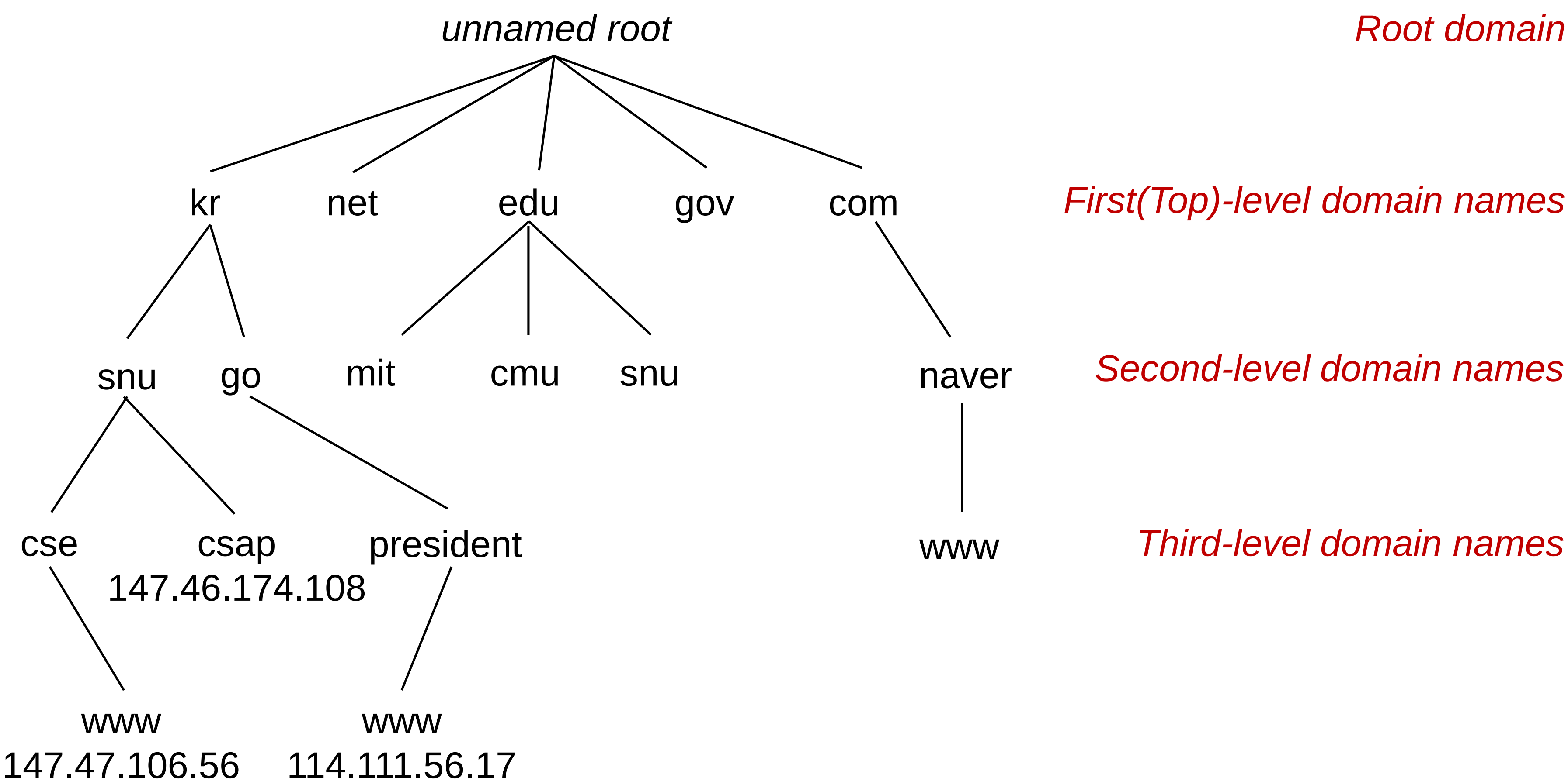
- The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called DNS
  - Conceptually, programmers can view the DNS database as a collection of millions of host entry structures:

```
/* DNS host entry structure */
struct hostent {
   char   *h_name;       /* official domain name of host */
   char   **h_aliases;   /* null-terminated array of domain names */
   int    h_addrtype;    /* host address type (AF_INET) */
   int    h_length;      /* length of an address, in bytes */
   char   **h_addr_list; /* null-terminated array of in_addr structs */
};
```

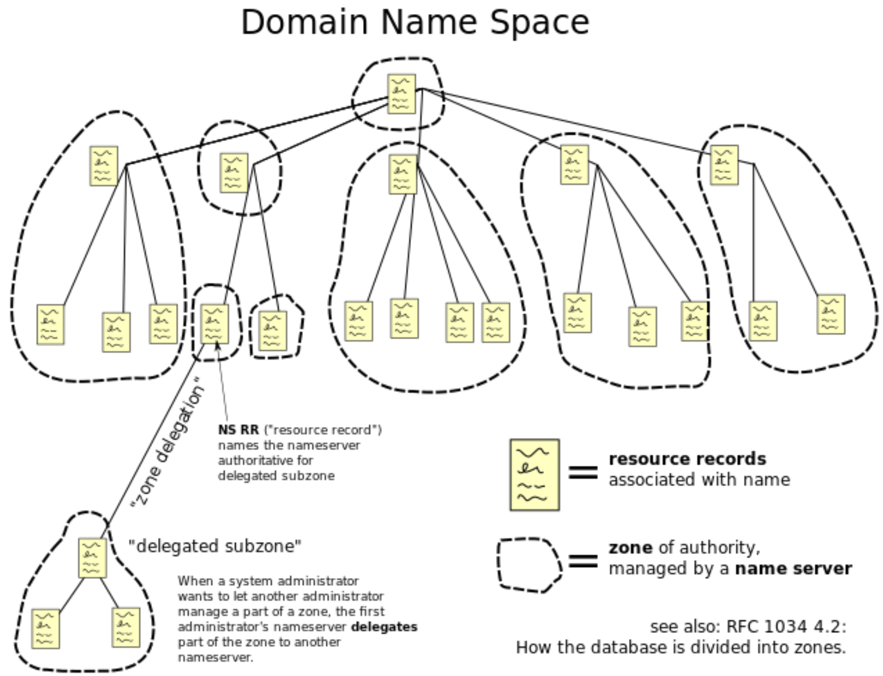
- Functions for retrieving host entries from DNS:
  - gethostbyname: query key is a DNS domain name.
  - gethostbyaddr: query key is an IP address.

DNS Organization

- Hierarchical organization of DNS servers

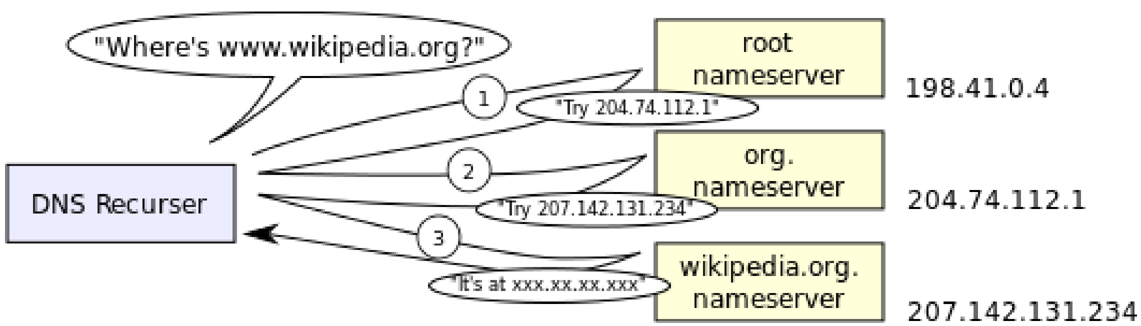


DNS Queries



Source: http://en.wikipedia.org/wiki/Domain_Name_System

- Each domain has at least two designated name servers
- These authoritative name servers contain the original DNS data for a domain
- Address resolution
  - recursively start at one of the 13 root name servers and find the authoritative name server for the given domain
  - to reduce traffic and increase lookup, DNS cache servers save results
    - hence the up to 24-hour period to propagate IP address changes



Source: http://en.wikipedia.org/wiki/Domain_Name_System

Properties of DNS Host Entries

- Each host entry is an equivalence class of domain names and IP addresses
- Each host has a locally defined domain name localhost which always maps to the loopback address 127.0.0.1
- Different kinds of mappings are possible:
  - Simple case: one-to-one mapping between domain name and IP address:
    - csap.snu.ac.kr maps to 147.46.174.108
  - Multiple domain names mapped to the same IP address:
    - eecs.mit.edu and cs.mit.edu both map to 18.62.1.6
  - Multiple domain names mapped to multiple IP addresses:
    - google.com maps to multiple IP addresses
  - Some valid domain names don't map to any IP address:
    - for example: ics.cs.cmu.edu

CSE 컴퓨터공학부 Department of Computer Science & Engineering

A Program That Queries DNS

```
int main(int argc, char **argv) { /* argv[1] is a domain name */
    char **pp;                      /* or dotted decimal IP addr */
    struct in_addr addr;
    struct hostent *hostp;

    if (inet_aton(argv[1], &addr) != 0)
        hostp = Gethostbyaddr((const char *)&addr, sizeof(addr),
                AF_INET);
    else
        hostp = Gethostbyname(argv[1]);
    printf("official hostname: %s\n", hostp->h_name);

    for (pp = hostp->h_aliases; *pp != NULL; pp++)
        printf("alias: %s\n", *pp);

    for (pp = hostp->h_addr_list; *pp != NULL; pp++) {
        addr.s_addr = ((struct in_addr *)*pp)->s_addr;
        printf("address: %s\n", inet_ntoa(addr));
    }
}
```

Using DNS Program

```
linux> ./dns csap.snu.ac.kr
official hostname: csap.snu.ac.kr
address 147.46.174.108

linux> ./dns 147.46.80.1
official hostname: ercc.snu.ac.kr
address: 147.46.80.1

linux> ./dns www.google.com
official hostname: www.l.google.com
alias: www.google.com
address: 74.125.71.105
address: 74.125.71.106
address: 74.125.71.147
address: 74.125.71.99
address: 74.125.71.103
address: 74.125.71.104
```

Querying DIG

- Domain Information Groper (dig) provides a scriptable command line interface to DNS

```
linux> dig +short csap.snu.ac.kr
147.46.174.108
linux> dig +short -x 147.46.80.1
ercc.snu.ac.kr.
linux> dig +short www.google.com
www.l.google.com.
74.125.31.106
74.125.31.147
74.125.31.99
74.125.31.103
74.125.31.104
74.125.31.105
linux> dig +short -x 74.125.71.105
hx-in-f105.1e100.net.
```

More Exotic Features of DIG

- Provides more information than you would ever want about DNS

```
linux> dig www.google.com a +trace

; <<>> DiG 9.8.1 <<>> www.google.com a +trace
;; global options: +cmd
.                    351140     IN      NS      g.root-servers.net.
.                    351140     IN      NS      i.root-servers.net.
.                    351140     IN      NS      l.root-servers.net.
…
.                    351140     IN      NS      b.root-servers.net.
.                    351140     IN      NS      k.root-servers.net.
.                    351140     IN      NS      d.root-servers.net.
;; Received 228 bytes from 147.46.37.10#53(147.46.37.10) in 18 ms

com.                 172800     IN      NS      i.gtld-servers.net.
com.                 172800     IN      NS      e.gtld-servers.net.
com.                 172800     IN      NS      j.gtld-servers.net.
…
com.                 172800     IN      NS      c.gtld-servers.net.
com.                 172800     IN      NS      h.gtld-servers.net.
;; Received 492 bytes from 192.228.79.201#53(192.228.79.201) in 354 ms

google.com.          172800     IN      NS      ns2.google.com.
…
google.com.          172800     IN      NS      ns4.google.com.
;; Received 168 bytes from 192.55.83.30#53(192.55.83.30) in 206 ms

www.google.com.              604800     IN      CNAME   www.l.google.com.
www.l.google.com.    300        IN      A       74.125.71.103
…
www.l.google.com.    300        IN      A       74.125.71.104
;; Received 148 bytes from 216.239.34.10#53(216.239.34.10) in 98 ms
```

Internet Connections

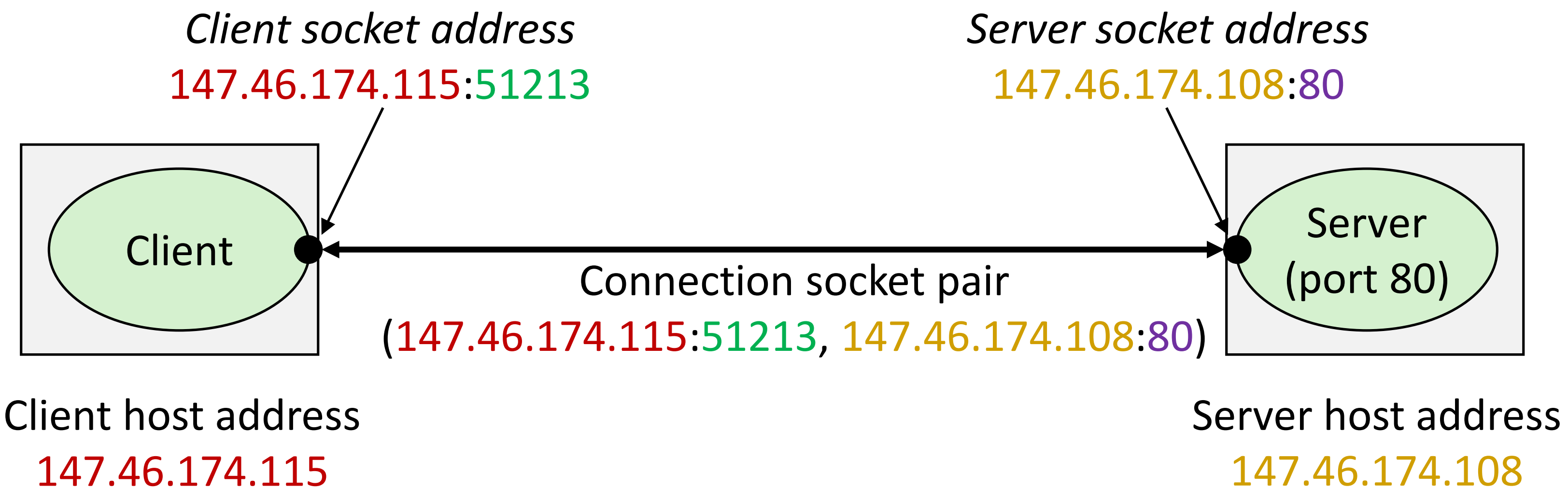
- Clients and servers communicate by sending streams of bytes over connections:
  - Point-to-point, full-duplex (2-way communication), and reliable.

- 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 (sock et pair)
  - (cliaddr:cliport, servaddr:servport)

Putting it all Together: Anatomy of an Internet Connection



Network Programming

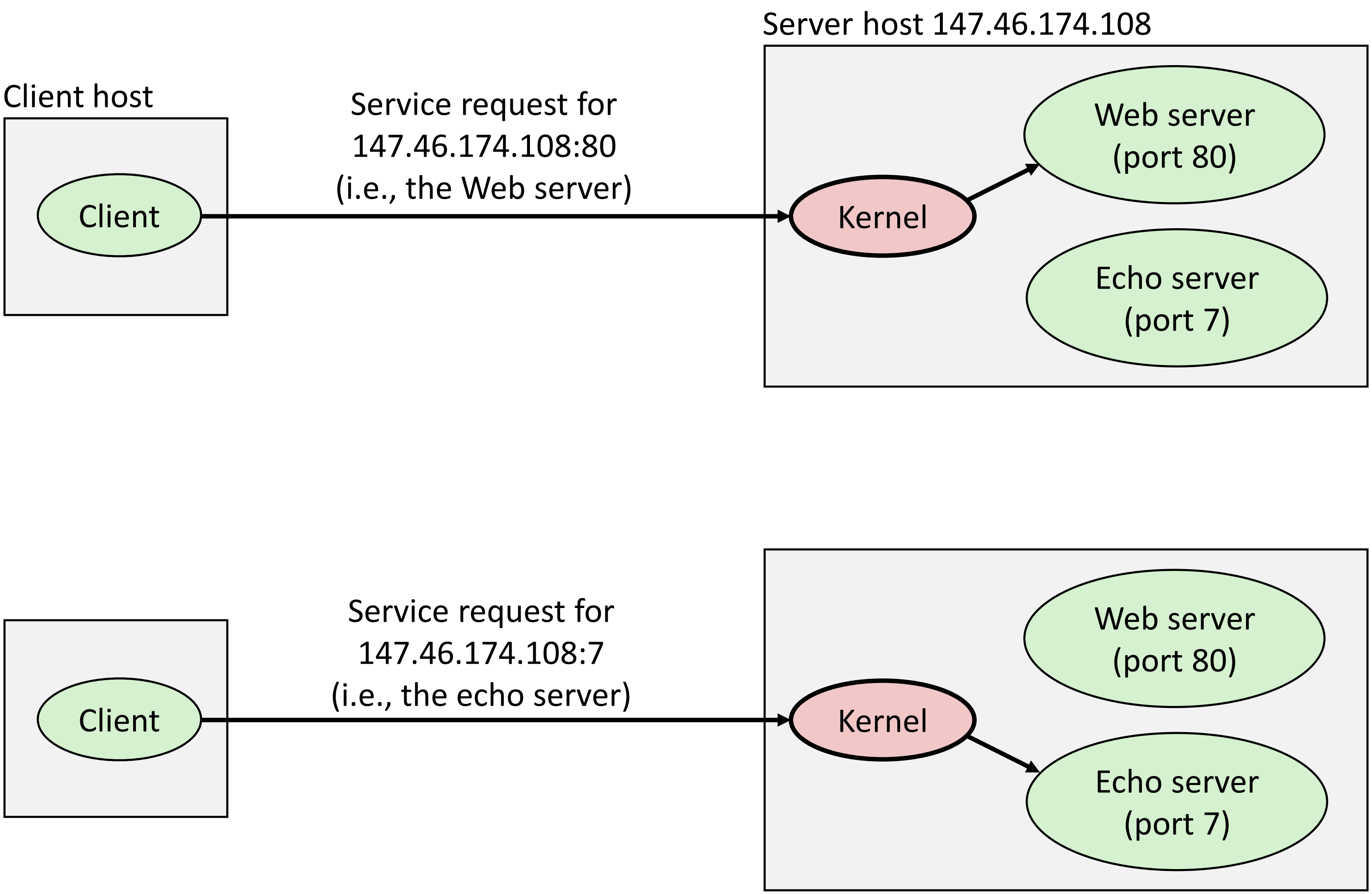
- Client-Server Model and Computer Networks
- Global IP Internet
  - naming
  - IPv4, IPv6
  - DNS
  - Internet connection
- **Programmer's View**
  - **The sockets interface**
  - **Establishing an Internet connection**
  - **Copy data over an Internet connection**

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
- 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

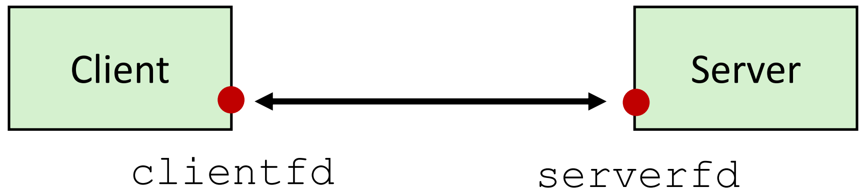
> See `/etc/services` for a comprehensive list of the port mappings on a Linux machine

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

Example: Echo Client and Server

On Client

On Server

```
csap> ./echoserveri 15213
```

```
linux> echoclient csap.snu.ac.kr 15213
```

```
server connected to langnau.snu.ac.kr
(147.47.174.115), port 64690
```

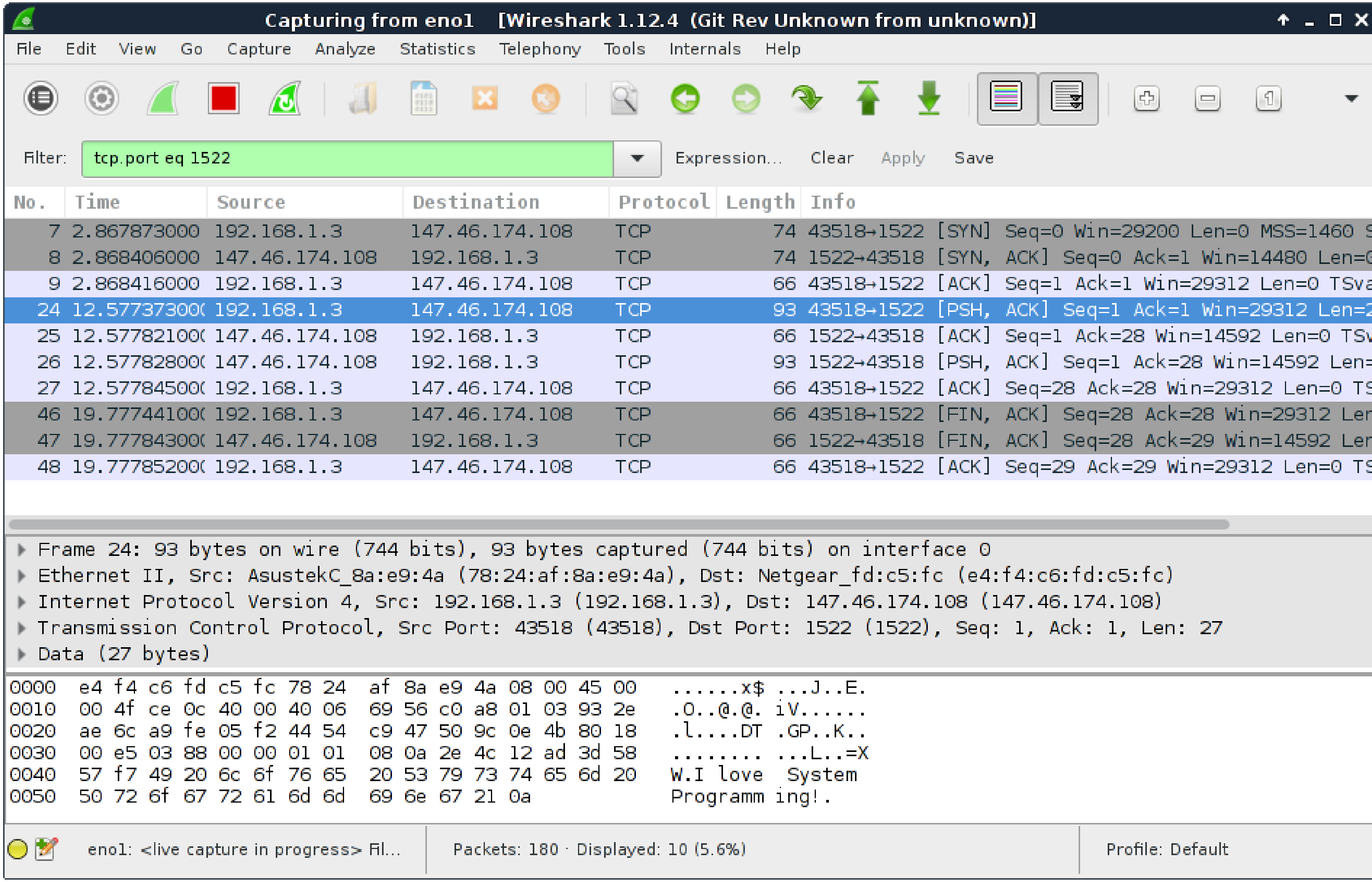
```
type: hello there
```

```
server received 12 bytes
```

```
echo: HELLO THERE
type: ^D
```

```
Connection closed
```

Watching Echo Client / Server



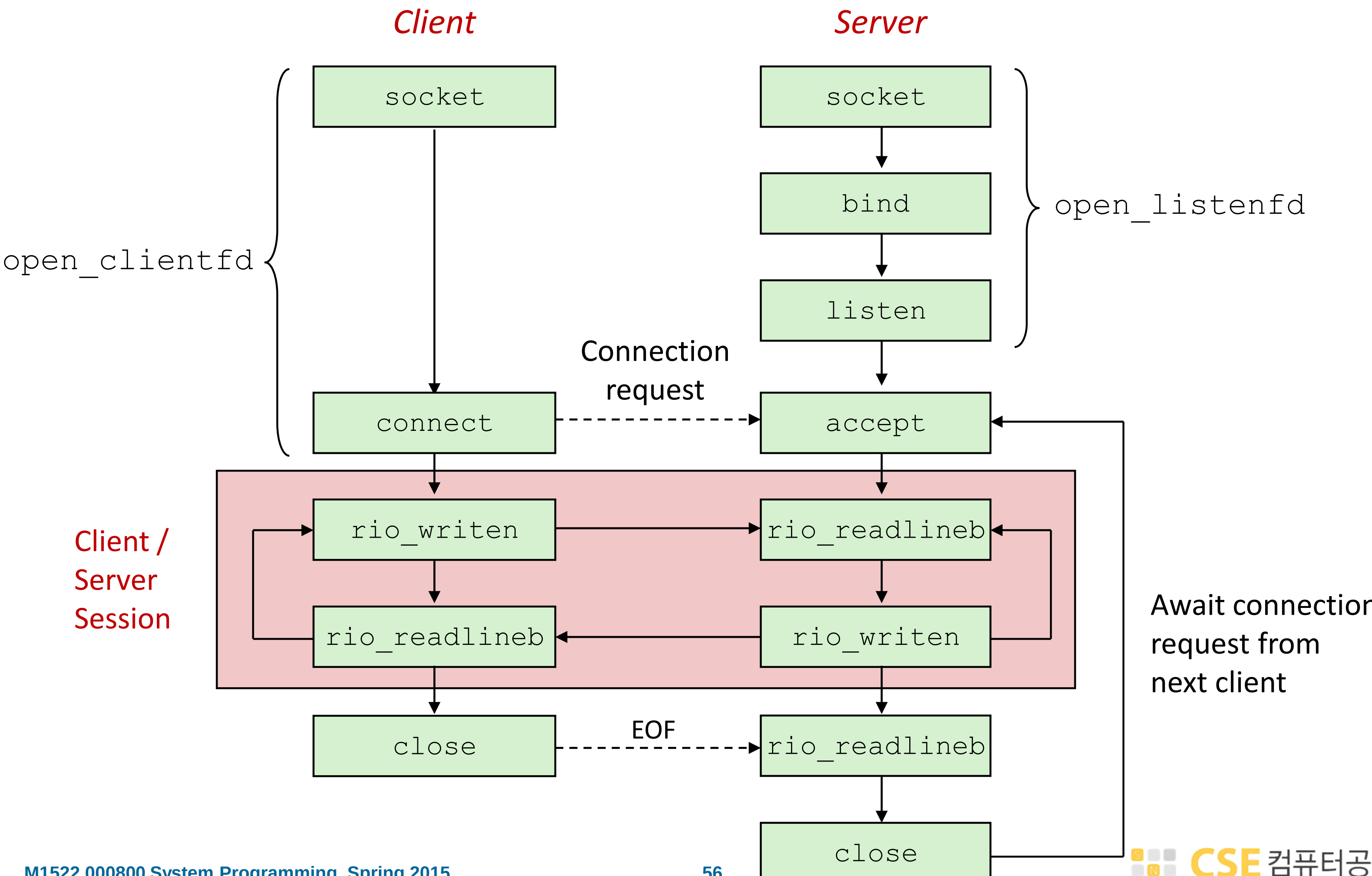
Ethical Issues

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

- Network traffic should be considered private; as a consequence, any packet interception of network information that is not intended for your use is not only an invasion of privacy but may also be a violation of university policy.

- Pledge
  - Instructors, TAs, and students of this class will not run packet sniffers in promiscuous mode inside SNU's network.

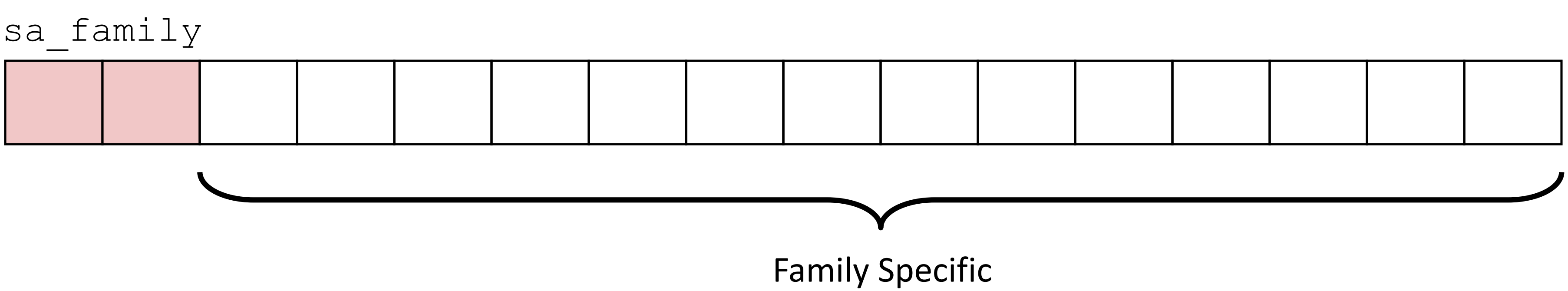
Overview of the Sockets Interface



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

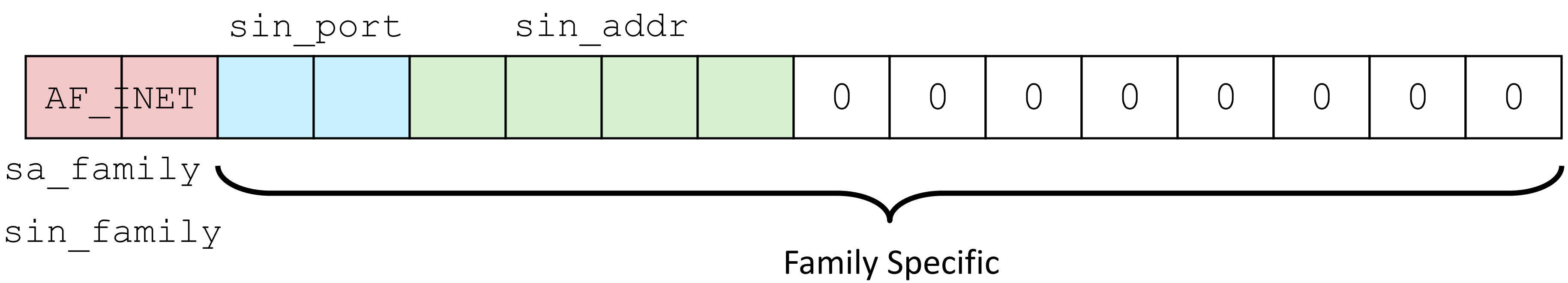
```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.  */
};
```



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) */
};
```



Echo Client Main Routine

```
#include "csapp.h"

/* usage: ./echoclient host port */
int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;
    host = argv[1];  port = atoi(argv[2]);
    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);
    printf("type:"); fflush(stdout);
    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        printf("echo:");
        Fputs(buf, stdout);
        printf("type:"); fflush(stdout);
    }
    Close(clientfd);
    exit(0);
}
```

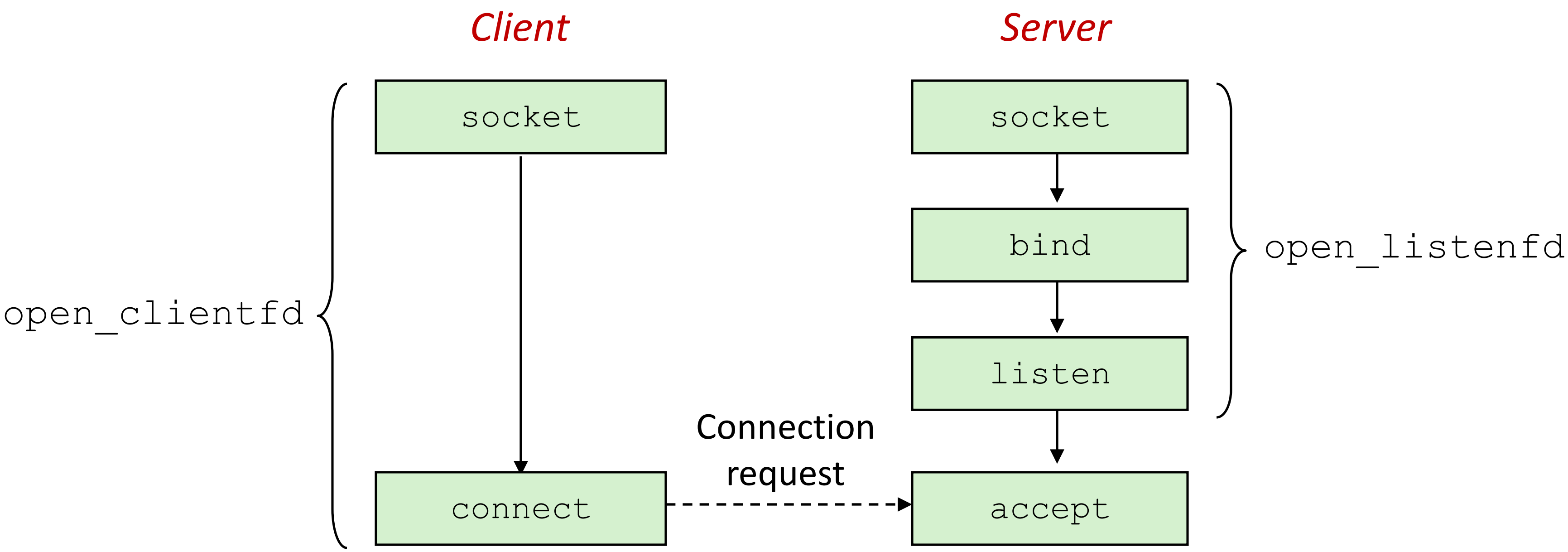
Read input line

Send line to server

Receive line from server

Print server response

Overview of the Sockets Interface



Echo Client: open_clientfd

```
int open_clientfd(char *hostname, int port) {
  int clientfd;
  struct hostent *hp;
  struct sockaddr_in serveraddr;

  if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1; /* check errno for cause of error */

  /* Fill in the server's IP address and port */
  if ((hp = gethostbyname(hostname)) == NULL)
    return -2; /* check h_errno for cause of error */
  bzero((char *) &serveraddr, sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  bcopy((char *)hp->h_addr_list[0],
        (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
  serveraddr.sin_port = htons(port);

  /* Establish a connection with the server */
  if (connect(clientfd, (SA *) &serveraddr,
      sizeof(serveraddr)) < 0)
    return -1;
  return clientfd;
}
```

This function opens a connection from the client to the server at `hostname:port`

Create socket

Create address

Establish connection

Echo Client: open_clientfd (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 */

if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1; /* check errno for cause of error */

... <more>
```

Echo Client: open_clientfd (gethostbyname)

- The client then builds the server's Internet address

```
int clientfd;                 /* socket descriptor */
struct hostent *hp;           /* DNS host entry */
struct sockaddr_in serveraddr; /* server's IP address */

...

/* fill in the server's IP address and port */
if ((hp = gethostbyname(hostname)) == NULL)
    return -2; /* check h_errno for cause of error */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(port);
bcopy((char *)hp->h_addr_list[0],
      (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
```

Check this out!

A Careful Look at bcopy Arguments

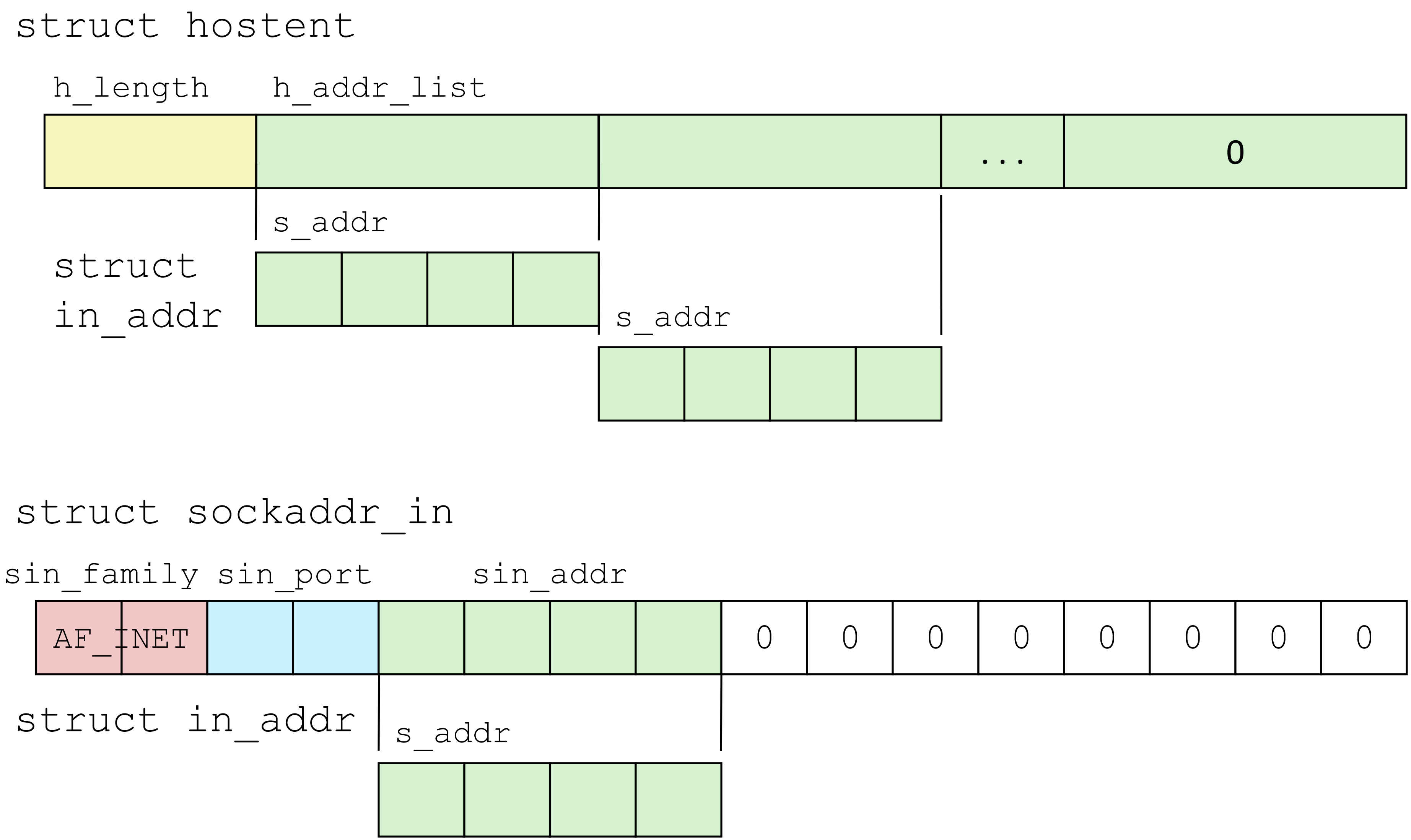
```
/* DNS host entry structure */
struct hostent {
   . . .
   int     h_length;       /* length of an address, in bytes */
   char    **h_addr_list; /* null-terminated array of in_addr structs */
};
```

```
struct sockaddr_in  {
  . . .
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  . . .
};
```

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

```
struct hostent *hp;              /* DNS host entry */
struct sockaddr_in serveraddr; /* server's IP address */
...
bcopy((char *)hp->h_addr_list[0], /* src, dest */
      (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
```

Bcopy Argument Data Structures



Echo Client: open_clientfd (connect)

- 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 */
  typedef struct sockaddr SA;        /* generic sockaddr */
...
  /* Establish a connection with the server */
  if (connect(clientfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
    return -1;
  return clientfd;
}
```

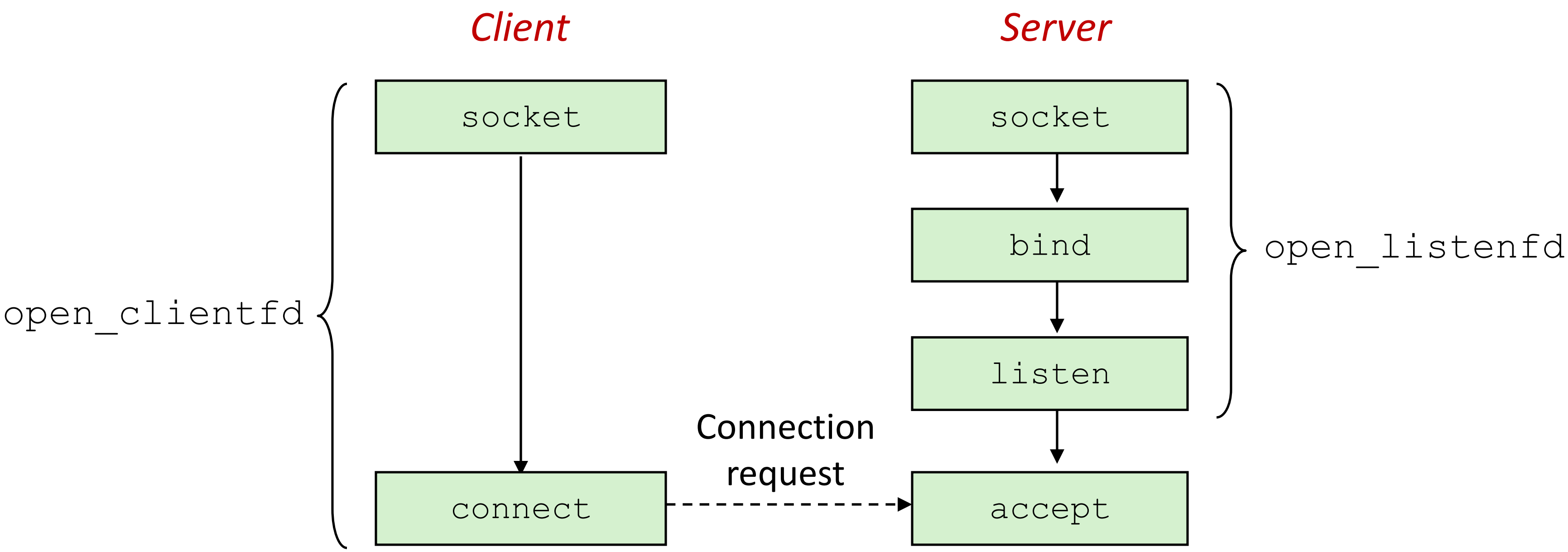
Echo Server: Main Routine

```
int main(int argc, char **argv) {
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    struct hostent *hp;
    char *haddrp;
    unsigned short client_port;

    port = atoi(argv[1]); /* the server listens on a port passed
                             on the command line */
    listenfd = open_listenfd(port);

    while (1) {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
        hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                         sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        haddrp = inet_ntoa(clientaddr.sin_addr);
        client_port = ntohs(clientaddr.sin_port);
        printf("server connected to %s (%s), port %u\n",
                hp->h_name, haddrp, client_port);
        echo(connfd);
        Close(connfd);
    }
}
```

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

Echo Server: open_listenfd

```
int open_listenfd(int port)
{
    int listenfd, optval=1;
    struct sockaddr_in serveraddr;

    /* Create a socket descriptor */
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1;

    /* Eliminates "Address already in use" error from bind. */
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                   (const void *)&optval , sizeof(int)) < 0)
        return -1;

... <more>
```

Echo Server: open_listenfd (cont.)

```
...

  /* Listenfd will be an endpoint for all requests to port
       on any IP address for this host */
    bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons((unsigned short)port);
    if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
        return -1;

    /* Make it a listening socket ready to accept
       connection requests */
    if (listen(listenfd, LISTENQ) < 0)
        return -1;

   return listenfd;
}
```

Echo Server: open_listenfd (socket)

- socket creates a socket descriptor on the server
  - AF_INET: indicates that the socket is associated with Internet protocols
  - SOCK_STREAM: selects a reliable byte stream connection (TCP)

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

/* Create a socket descriptor */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1;
```

Echo Server: open_listenfd (setsockopt)

- The socket can be given some attributes

```
...
/* Eliminates "Address already in use" error from bind(). */
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
               (const void *)&optval , sizeof(int)) < 0)
    return -1;
```

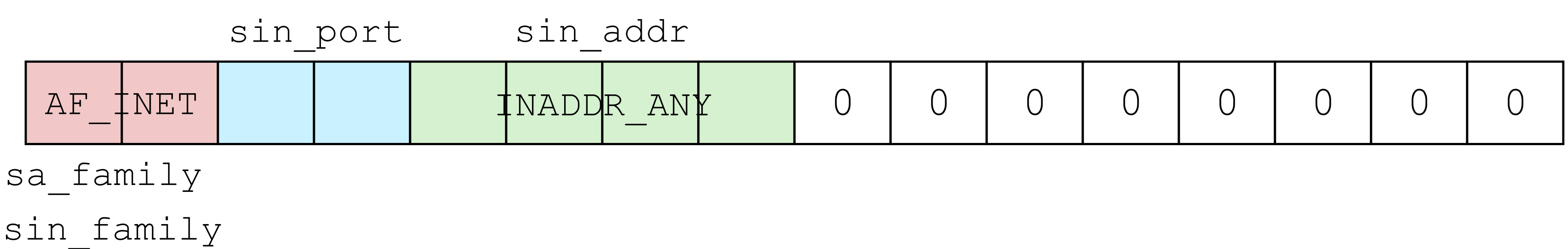
- Handy trick that allows us to rerun the server immediately after we kill it
  - Otherwise we would have to wait about 15 seconds
  - Eliminates "Address already in use" error from bind()
- Strongly suggest you do this for all your servers to simplify debugging

Echo Server: open_listenfd (initialize socket address)

- Initialize socket with server port number
- Accept connection from any IP address

```
  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 */
  bzero((char *) &serveraddr, sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  serveraddr.sin_port = htons((unsigned short)port);
  serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

- IP addr and port stored in network (big-endian) byte order



Echo Server: open_listenfd (bind)

- 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 */
  if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
      return -1;
```

Echo Server: open_listenfd (listen)

- listen indicates that this socket will accept connection (connect) requests from clients
- LISTENQ is constant indicating how many pending requests allowed

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

...
 /* Make it a listening socket ready to accept connection requests */
    if (listen(listenfd, LISTENQ) < 0)
        return -1;
    return listenfd;
}
```

- We're finally ready to enter the main server loop that accepts and processes client connection requests.

Echo Server: Main Loop

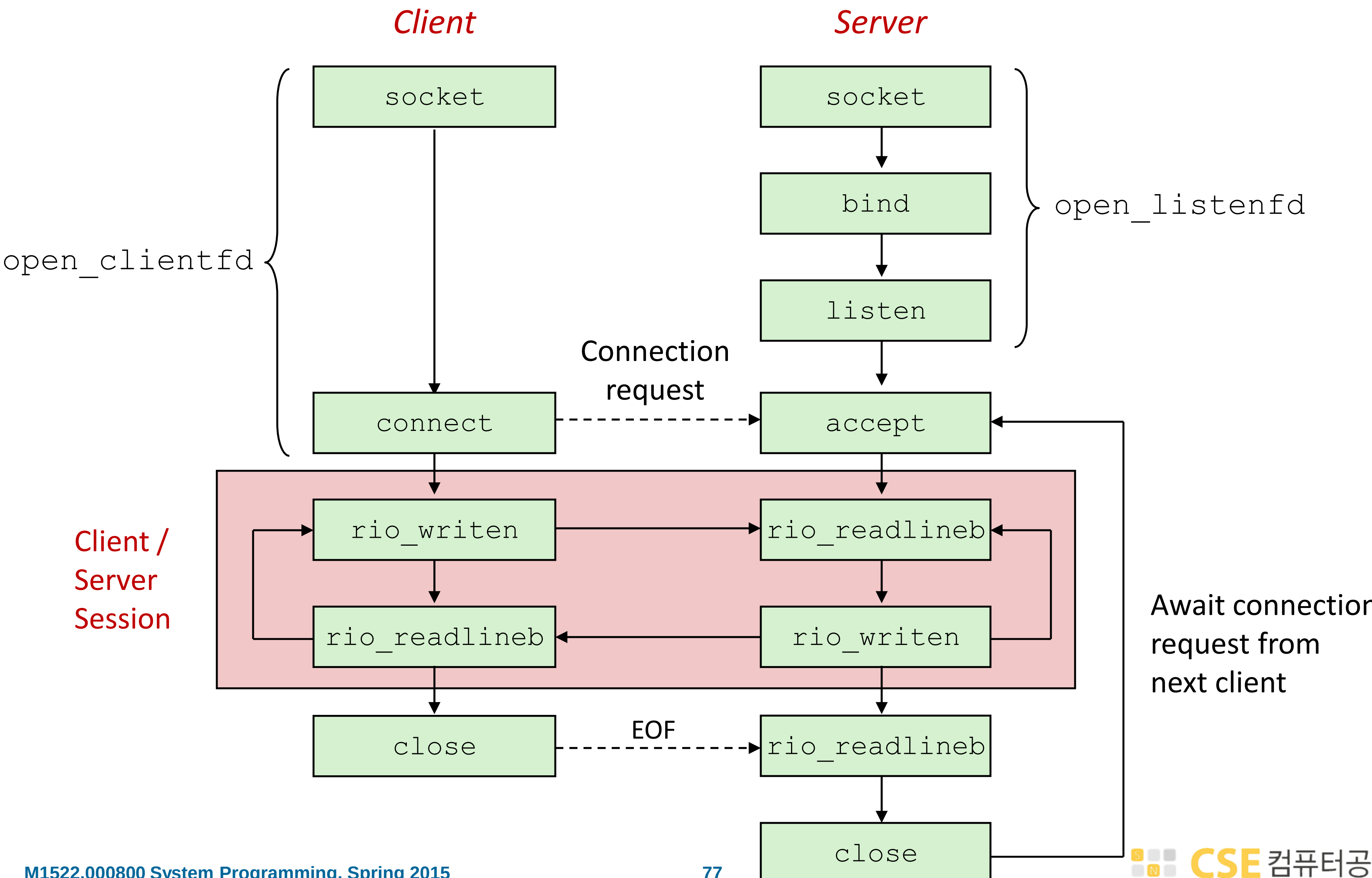
- The server loops endlessly, waiting for connection requests, then reading input from the client, and echoing the input back to the client.

```
main() {

   /* create and configure the listening socket */

   while(1) {
      /* Accept(): wait for a connection request */
      /* echo(): read and echo input lines from client til EOF */
      /* Close(): close the connection */
   }
}
```

Overview of the Sockets Interface



Echo Server: accept

- accept() blocks waiting for a connection request

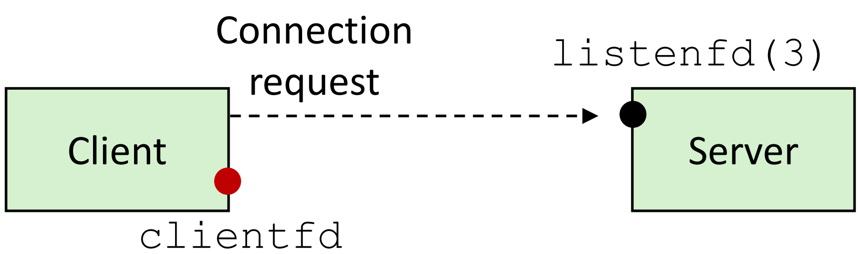
```
int listenfd; /* listening descriptor */
int connfd;   /* connected descriptor */
struct sockaddr_in clientaddr;
int clientlen;

clientlen = sizeof(clientaddr);
connfd = Accept(listenfd, (SA *)&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`

Connected vs. Listening Descriptors

- Listening descriptor
  - End point for client connection requests
  - Created once and exists for lifetime of the server

- Connected descriptor
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client
  - Exists only as long as it takes to service client

- Why the distinction?
  - Allows for concurrent servers that can communicate over many client connections simultaneously
    - E.g., Each time we receive a new request, we fork a child to handle the request

Echo Server: Identifying the Client

- The server can determine the domain name, IP address, and port of the client

```
struct hostent *hp;  /* pointer to DNS host entry */
char *haddrp;        /* pointer to dotted decimal string */
unsigned short client_port;
hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                        sizeof(clientaddr.sin_addr.s_addr), AF_INET);
haddrp = inet_ntoa(clientaddr.sin_addr);
client_port = ntohs(clientaddr.sin_port);
printf("server connected to %s (%s), port %u\n",
       hp->h_name, haddrp, client_port);
```

Echo Server: echo

- The server uses RIO to read and echo text lines until EOF (end-of-file) is encountered.
  - EOF notification caused by client calling close(clientfd)

```
void echo(int connfd)
{
    size_t n;
    char buf[MAXLINE];
    rio_t rio;

    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
        upper_case(buf);
        Rio_writen(connfd, buf, n);
        printf("server received %d bytes\n", n);
    }
}
```

Testing Servers Using telnet

- The telnet program is invaluable for testing servers that transmit ASCII strings over Internet connections
  - Our simple echo server
  - Web servers
  - Mail servers

- Usage:
  - unix> telnet <host> <portnumber>
  - Creates a connection with a server running on <host> and listening on port <portnumber>

Testing the Echo Server With telnet

```
greatwhite> echoserver 15213

linux> telnet csap.snu.ac.kr 15213
Trying 147.46.174.108...
Connected to csap.snu.ac.kr.
Escape character is '^]'.
hi there
HI THERE
```

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