COMPUTER NETWORKS

Tanzir Ahmed

CSCE 313 Spring 2020

Acknowledgments

- TAMU CSCE 313 Lecture Notes in Networking
  - *Acknowledgment: Profs Gu, Bettati, Tyagi*
- RICE COMP 221 Lecture Notes in Networking
  - *Acknowledgment: Prof. Cox*
- U-Illinois CS241 Lecture Notes in Networking
  - *Acknowledgment: Prof. Angrave*
- Beej's Guide to Network Programming, Ver. 3.0.15
- Socket Programming 101
  - *Acknowledgment: Vivek Ramachandran*
- U-Wisconsin CS-354 Lecture Notes in Networking
  - *Acknowledgment: Prof. Arpaci-Dusseau*

Objectives

- Exposure to the basic underpinnings of the Internet
- Use network socket interfaces effectively

The 2004 A. M. Turing Award

Bob Kahn Vint Cerf

"For pioneering work on internetworking, including the design and implementation of the Internet's basic communications protocols, TCP/IP, and for inspired leadership in networking."

The only Turing Award given to-date to recognize work in computer networking

But at the Same Time…

Daily, e.g. 110 attacks from China; 26 attacks from USA

2/2016 Hackers dumped the records of nearly 30,000 FBI and Department of Homeland Security workers.

1/2017 Hacking/Surveillance company Cellebrite lost 900GB of user data to a hac

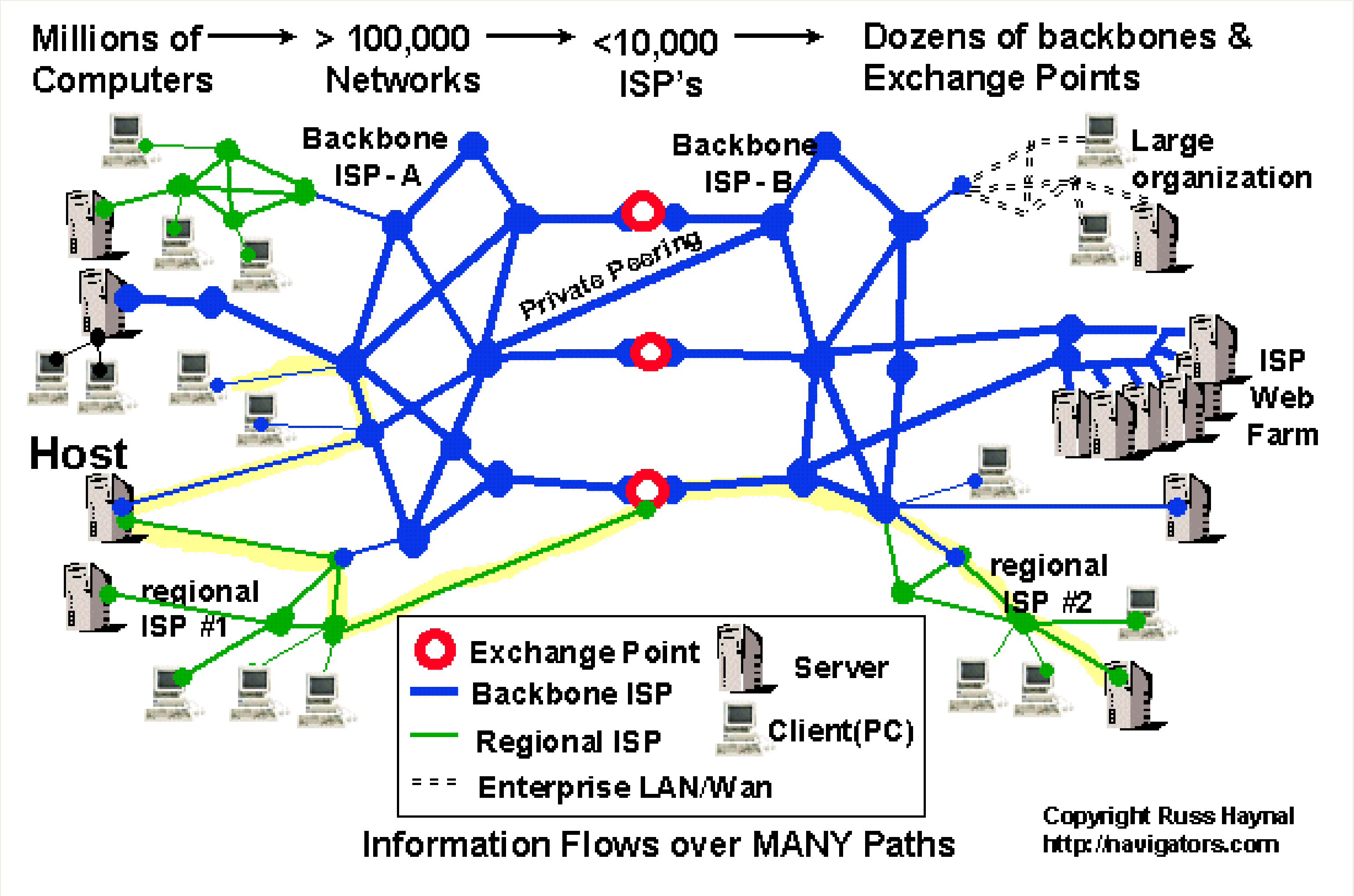
9/2018 Facebook security breach exposes personal data of 50M users

Source: Akamai Technologies, In

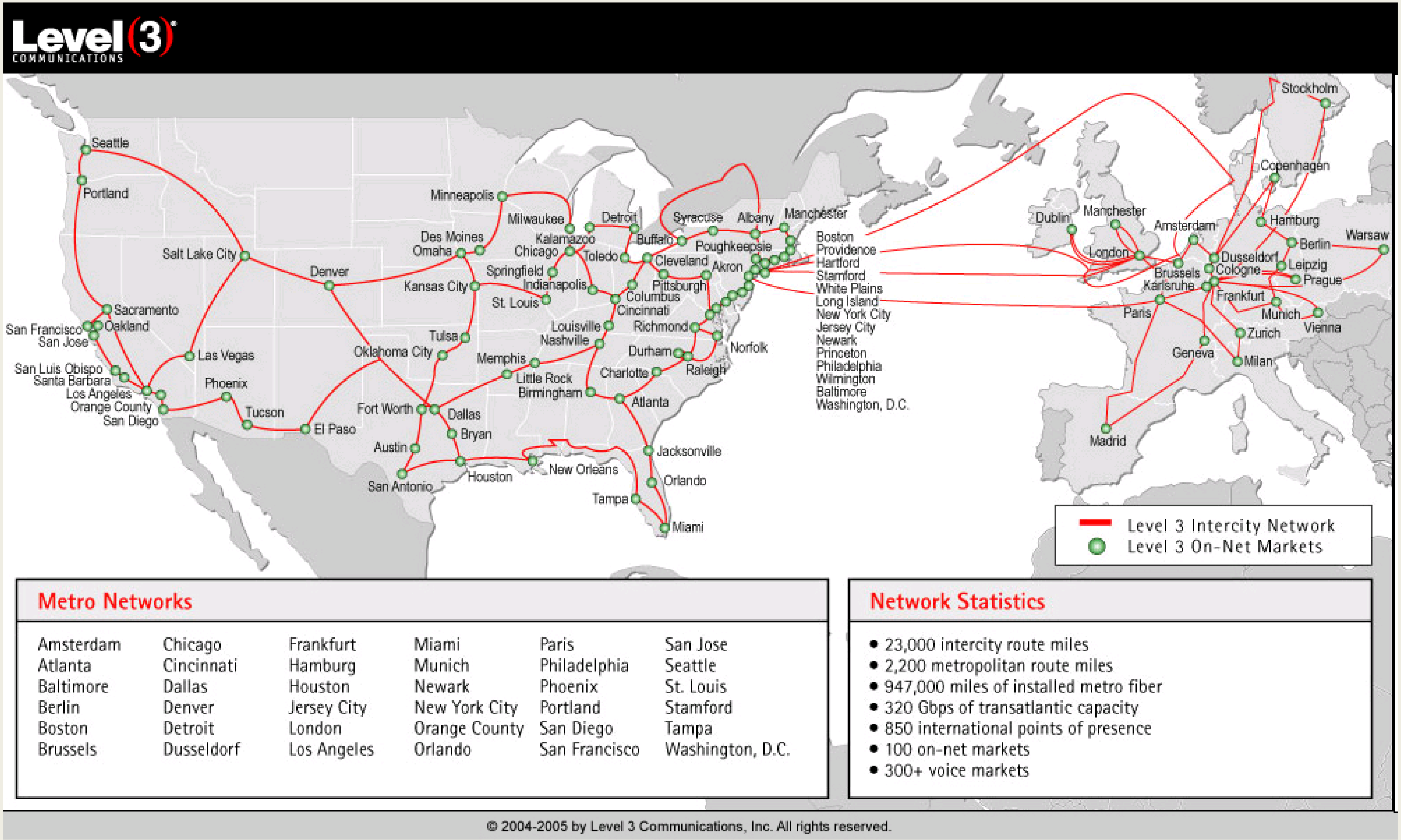
Bottom-Line…..

- Internet is a ubiquitous presence in our lives
- Issues such as Security Lapses present themselves as **opportunities** for making our ways of communication more robust
- Let's now trace back the history from the early days of telephony in the next few slides

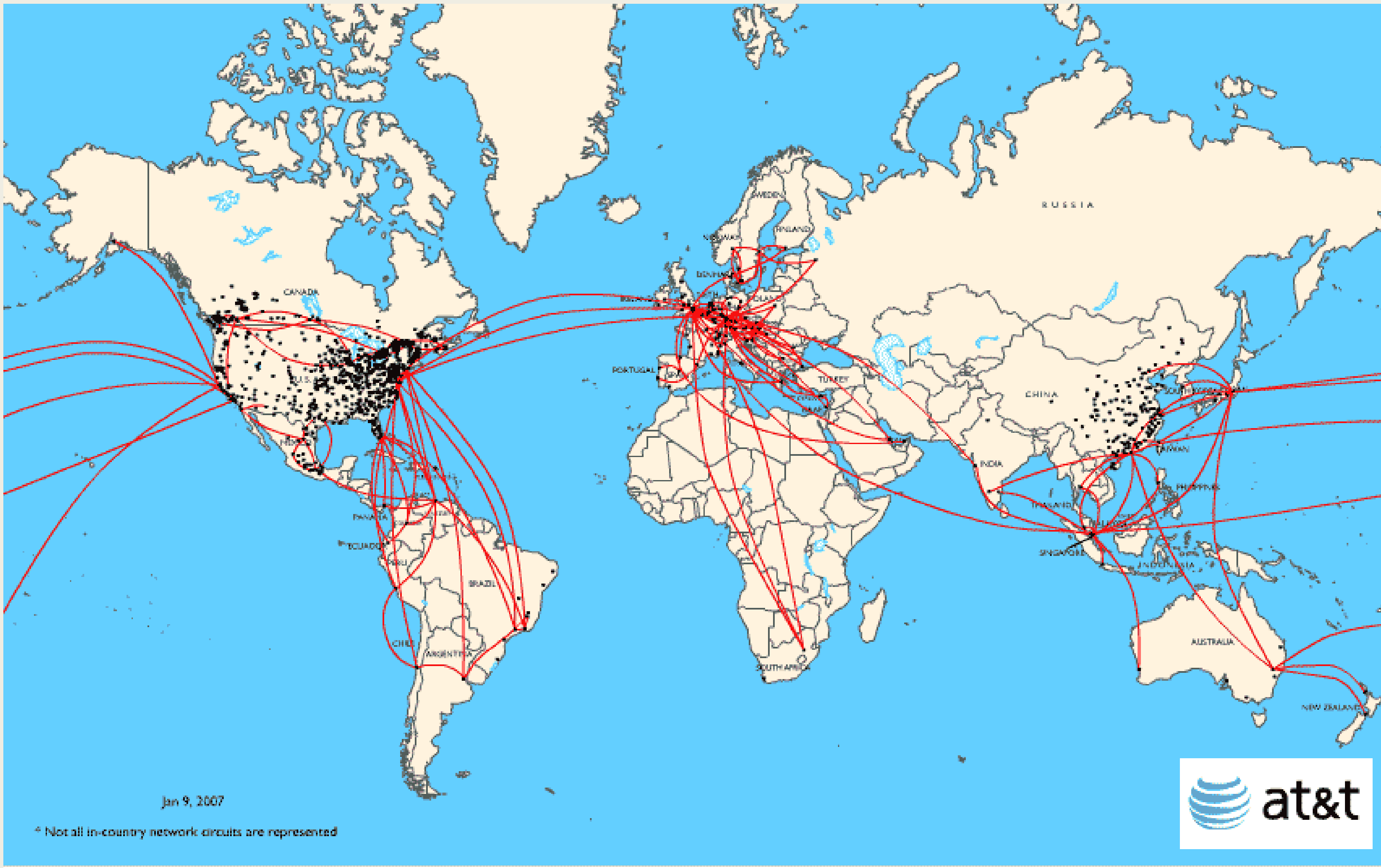
Current Internet Architecture - Conceptual



Level 3 Backbone



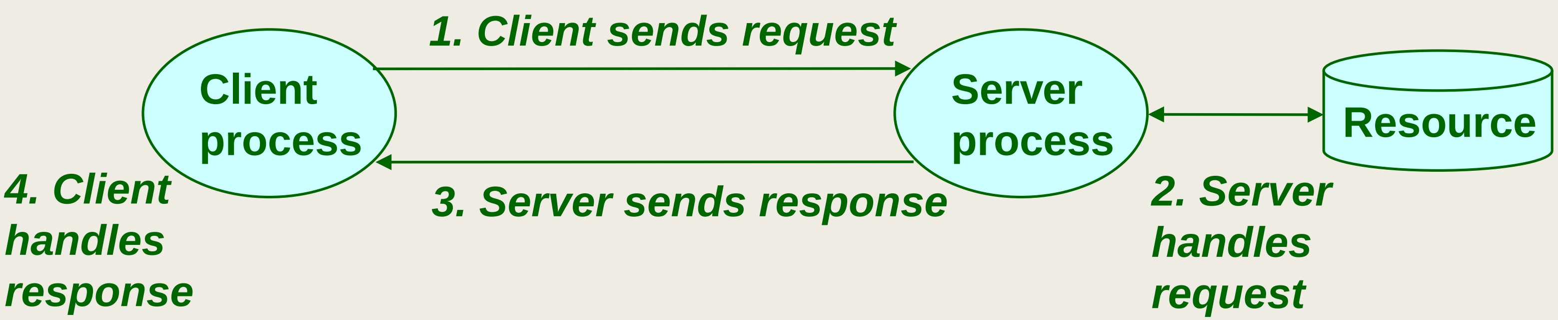
AT&T Backbone



A Client-Server Transaction

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**



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

Computer Networks

A network is a hierarchical system of boxes and wires organized by geographical proximity

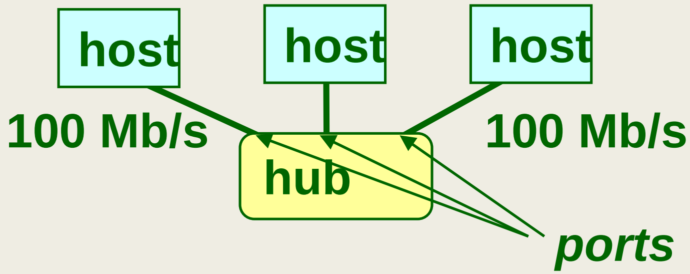
- Cluster network spans cluster or machine room
  - Switched Ethernet, Infiniband, …
- LAN (local area network) spans a building or campus
  - Ethernet is most prominent example
- WAN (wide-area network) spans very long distance
  - A high-speed point-to-point link
  - Leased line or SONET/SDH circuit, or MPLS/ATM circuit

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")

Lowest Level of Connectivity: Ethernet Segment

Ethernet segment consists of a collection of hosts connected by wires (twisted pairs) to a hub



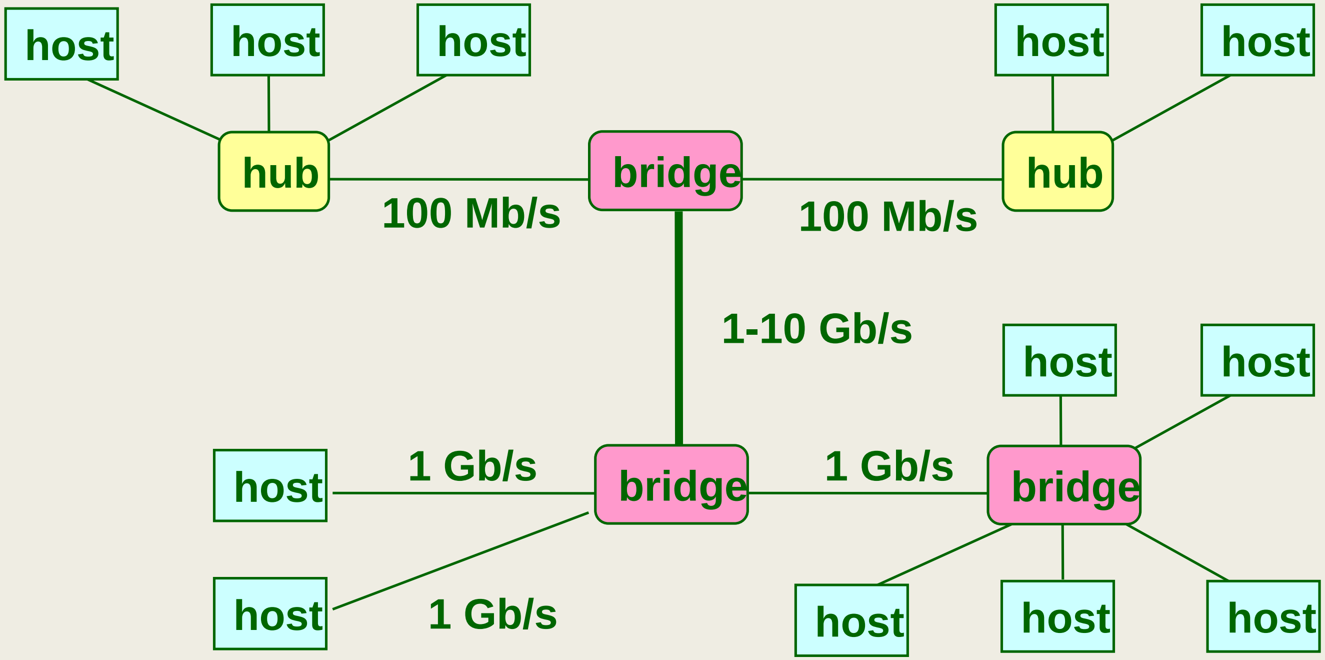
Operation

- Each Ethernet adapter has a unique 48-bit address
- Hosts send bits to any other host in chunks called frames
- Hub copies each bit from each port to every other port
  - Every host sees every bit
- Note: Hubs are largely obsolete
  - Bridges (switches, routers) became cheap enough to replace them (don’t broadcast all traffic)

Next Level: Bridged Ethernet Segment

Spans room, building, or campus

Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port



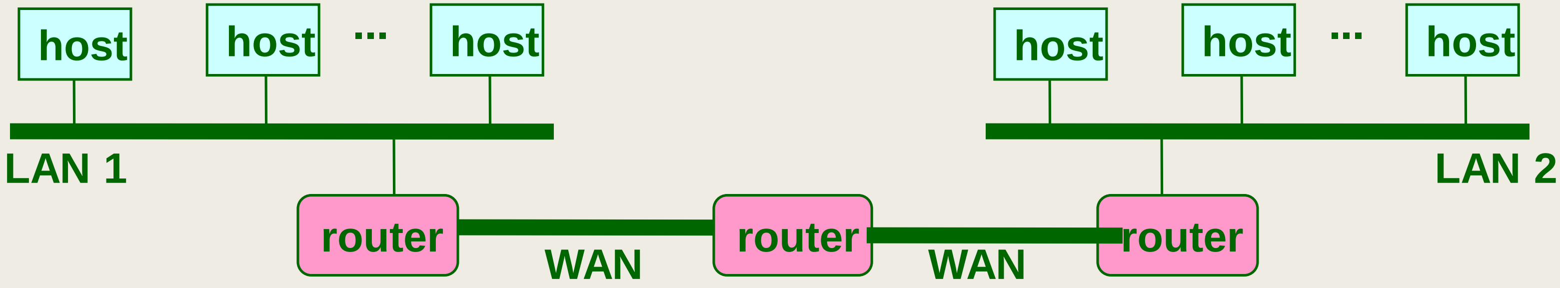
Conceptual View of LANs

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 LANs (e.g., Ethernet and WiFi, 802.11*, T1-links, DSL, …)

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 smoothens out the differences between the different networks

Implements an internet protocol (i.e., set of rules) that 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?

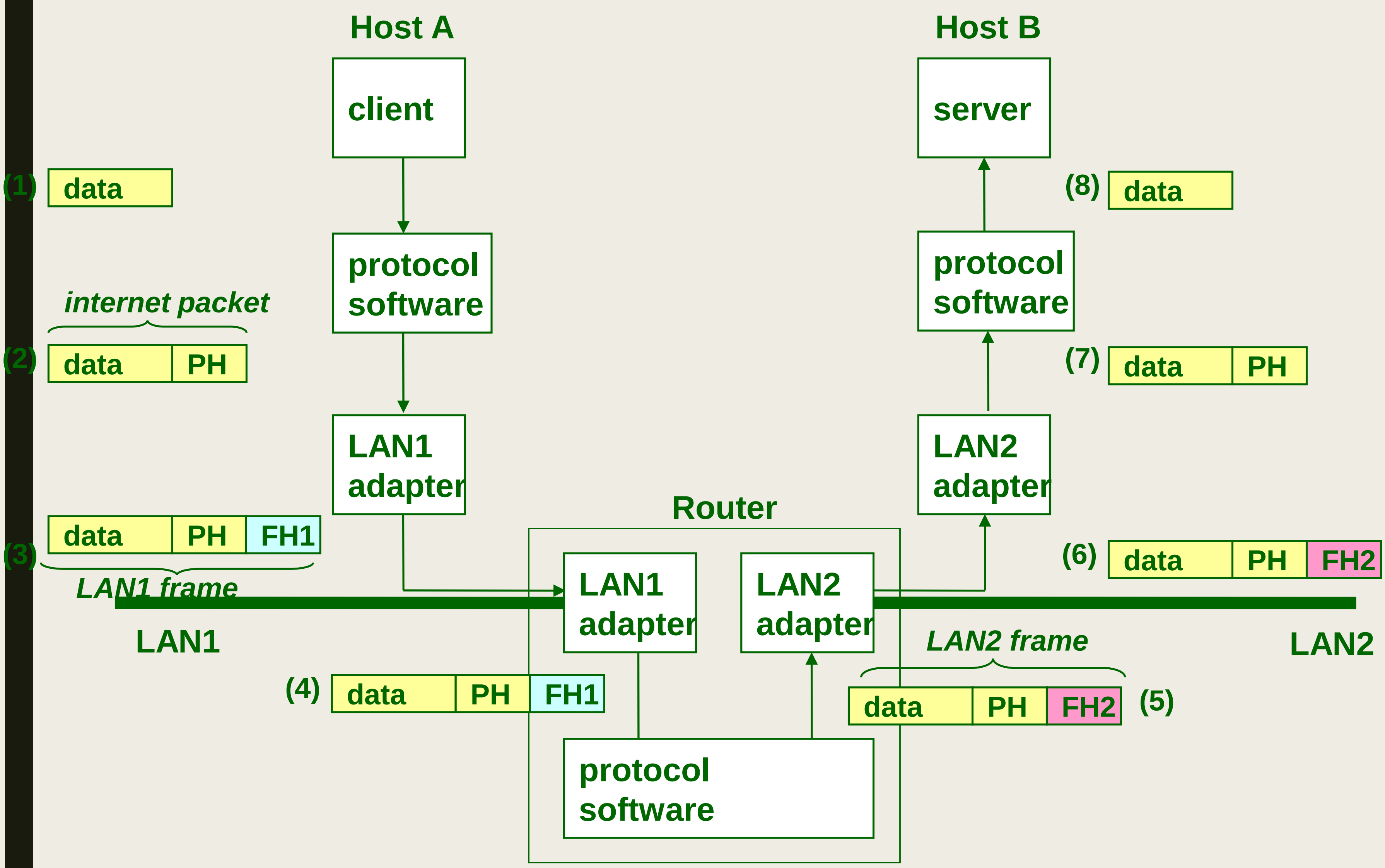
1. Provides a naming scheme

- 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

2. 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



Other Issues

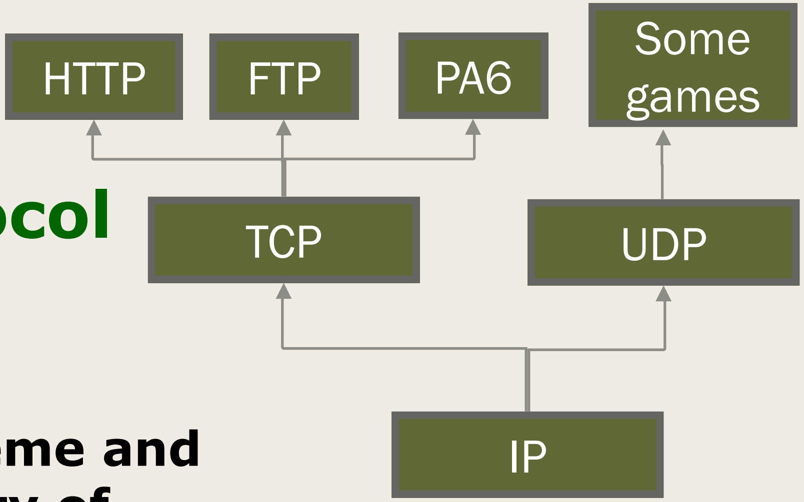
We are glossing over a number of important questions:

- 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?

We'll leave the discussion of these question to computer networking classes

→ CSCE 463

Global IP 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 (User 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

A Programmer’s View of the Internet

Hosts are mapped to a set of 32-bit IP addresses

- **e.g. 128.194.255.88 (4 * 8 bits)**

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

A set of identifiers called Internet domain names are mapped to the set of IP addresses for convenience (Domain Name Server aka DNS)

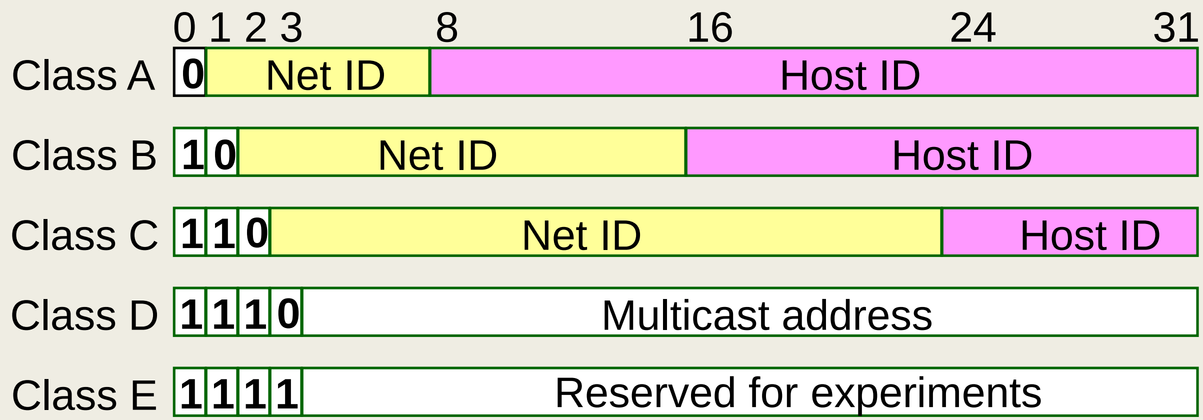
- **linux2.cs.tamu.edu is mapped to 128.194.138.88**
- **A process on one Internet host can communicate with a process on another Internet host over a connection**

Dotted Decimal Notation

- By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period
  - IP address `0x8002C2F2 = 128.2.194.242`
- Functions for converting between binary IP addresses and dotted decimal strings:
  - `inet_pton`: converts a dotted decimal string to an IP address in network byte order
  - `inet_ntop`: converts an IP address in network by order to its corresponding dotted decimal string
  - "`n`" denotes network representation, "`p`" denotes presentation representation

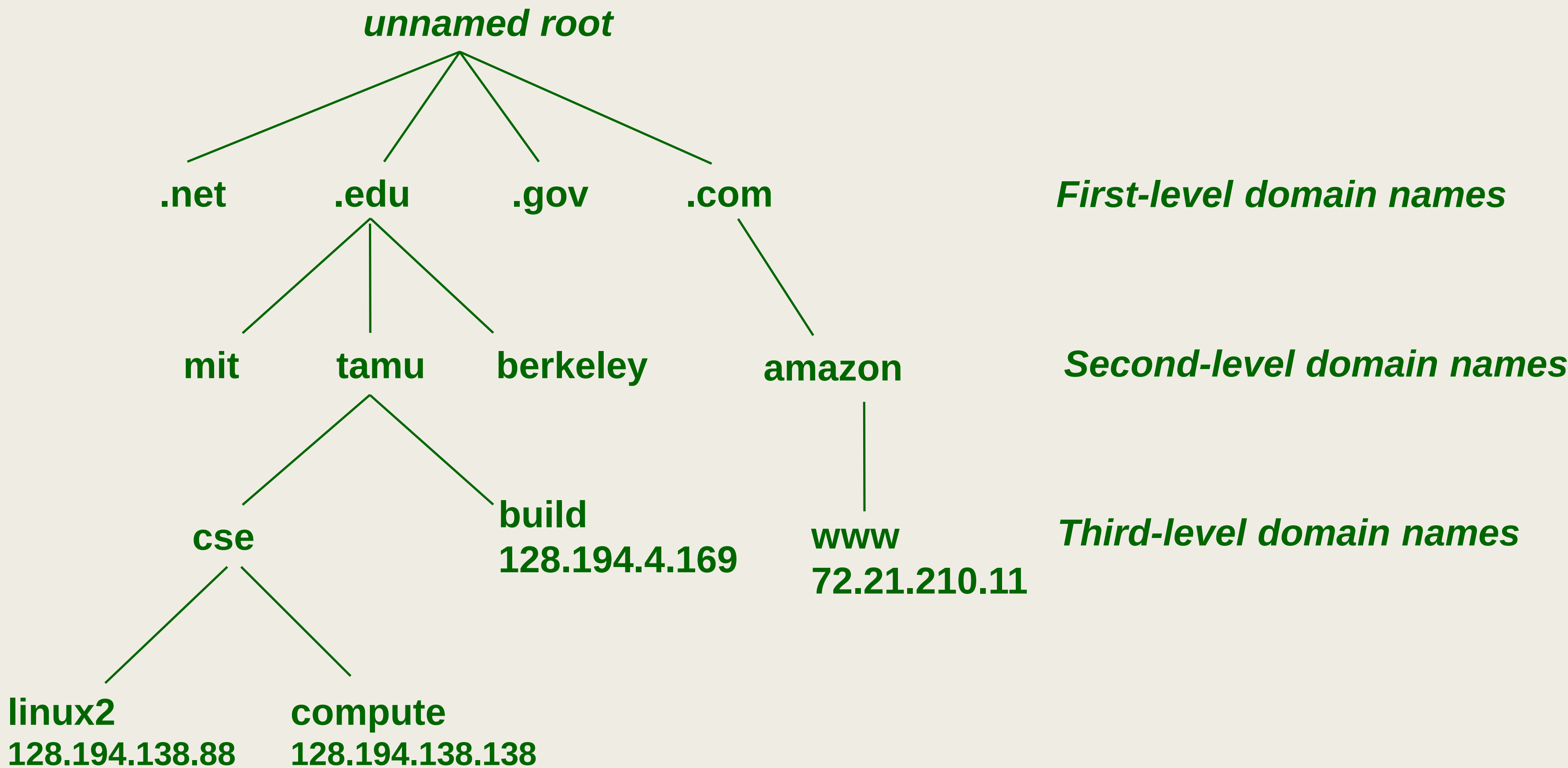
IP Address Structure

- **IP (V4) Address space divided into classes:**



- **Special Addresses for routers and gateways (all 0/1's)**
- **Loop-back address: 127.0.0.1**
- **Un-routed (private) IP addresses:**
  - **10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16**
- **Dynamic IP addresses (DHCP)**

Internet Domain Names



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 addrinfo structures:

DNS has uses beyond just IP address lookup:

- Load balancing for busy servers (e.g., google.com)
- Finding the nearest site for content (e.g., youtube.com)

Functions for retrieving host entries from DNS:

- getaddrinfo: query DNS using domain name or IP

Querying DNS

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

Shows just IP

Reverse lookup

```
unix> dig +short linux2.cse.tamu.edu
128.194.138.88
unix> dig +short -x 128.194.138.85
chevron.cs.tamu.edu.
unix> dig +short www.google.com
216.58.194.36
unix> dig +short www.google.com
172.217.12.68
unix> dig +short build.tamu.edu
compute.cse.tamu.edu.
128.194.138.139
```

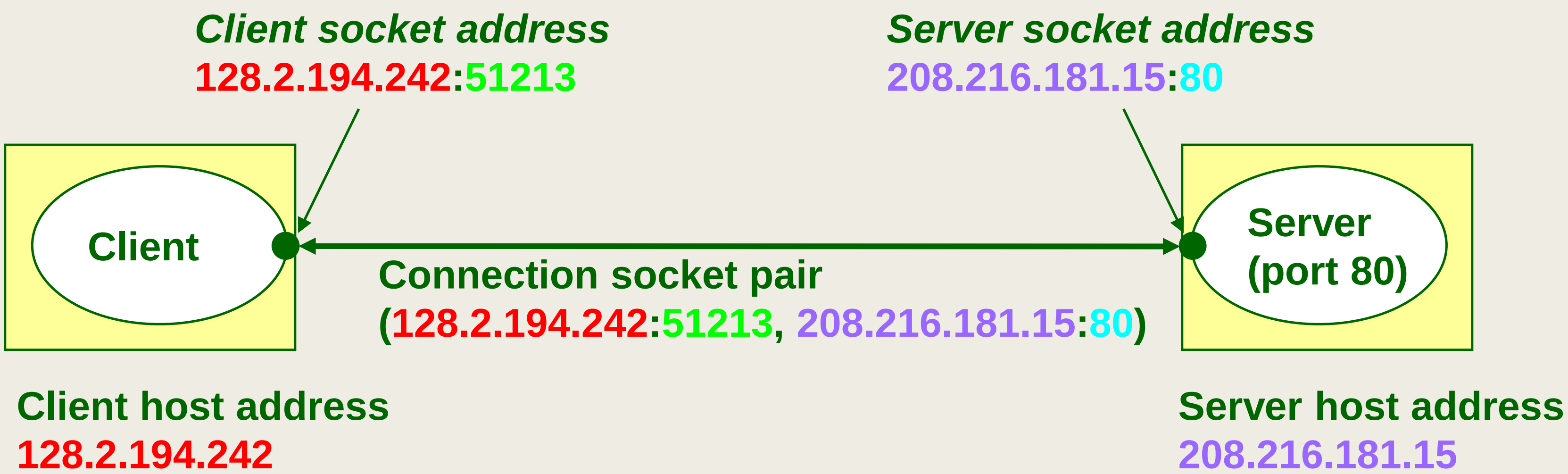
Same server, different IPs

Shows aliases

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 IP address and 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



Prerequisites for an Internet Connection

- There must be a network path between the Client and the Server
  - For instance, there is a path between your cell phone and your computer residing in the same WiFi network
  - A path between your desktop and google.com
- **Firewalls** could artificially **block** network paths
  - Path between off-campus computer and build.tamu.edu is blocked by sys admins
  - Path between compute.cs and linux2.cs is also blocked by admins
  - Amazon EC2 servers do not accept outside connections except for specific ports

Testing for Network Paths

ping command

- Tests if you can reach another host
- Example 1: From off-campus (w/o VPN)

```
osboxes@osboxes:~$ ping www.google.com
PING www.google.com (172.217.1.132) 56(84) bytes of data.
64 bytes from atl14s07-in-f132.1e100.net (172.217.1.132): icmp_seq=1 ttl=55 time=13.8 ms
64 bytes from atl14s07-in-f132.1e100.net (172.217.1.132): icmp_seq=2 ttl=55 time=14.9 ms
64 bytes from atl14s07-in-f132.1e100.net (172.217.1.132): icmp_seq=3 ttl=55 time=13.3 ms
64 bytes from atl14s07-in-f132.1e100.net (172.217.1.132): icmp_seq=4 ttl=55 time=13.5 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/mdev = 13.296/13.863/14.858/0.599 ms
osboxes@osboxes:~$ ping build.tamu.edu
PING compute.cse.tamu.edu (128.194.138.139) 56(84) bytes of data.
^C
--- compute.cse.tamu.edu ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8195ms

osboxes@osboxes:~$
```

- Example 2: Now with VPN connected

```
osboxes@osboxes:~$ ping build.tamu.edu
PING compute.cse.tamu.edu (128.194.138.139) 56(84) bytes of data.
64 bytes from compute.cs.tamu.edu (128.194.138.139): icmp_seq=1 ttl=59 time=23.2 ms
64 bytes from compute.cs.tamu.edu (128.194.138.139): icmp_seq=2 ttl=59 time=23.7 ms
64 bytes from compute.cs.tamu.edu (128.194.138.139): icmp_seq=3 ttl=59 time=23.0 ms
64 bytes from compute.cs.tamu.edu (128.194.138.139): icmp_seq=4 ttl=59 time=24.4 ms
^C
--- compute.cse.tamu.edu ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 22.999/23.560/24.391/0.544 ms
```

TCP Connectivity in the Path

telent command

- A built in TCP client, used to check server existence or connection ability
- Tests if you can make a TCP connection to a server port
- <u>Example 1:</u> Connecting to port 80 (HTTP server) of google

```
osboxes@osboxes:~$ telnet www.google.com 80
Trying 172.217.14.164...
Connected to www.google.com.
Escape character is '^]'.
```

- <u>Example 2:</u> Connecting to port 80 of build.tamu

```
osboxes@osboxes:~$ telnet build.tamu.edu 80
Trying 128.194.138.139...
^C
osboxes@osboxes:~$
```

  - Connection not possible because:
    - No service at port 80
    - Even if it had, firewall does not allow

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

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 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

See `/etc/services` for a comprehensive list of the services available on a UNIX machine

Web server (port 80)

- HTML files/content/other data through HTTP protocol

FTP server (20, 21)

- Service: stores and retrieve files

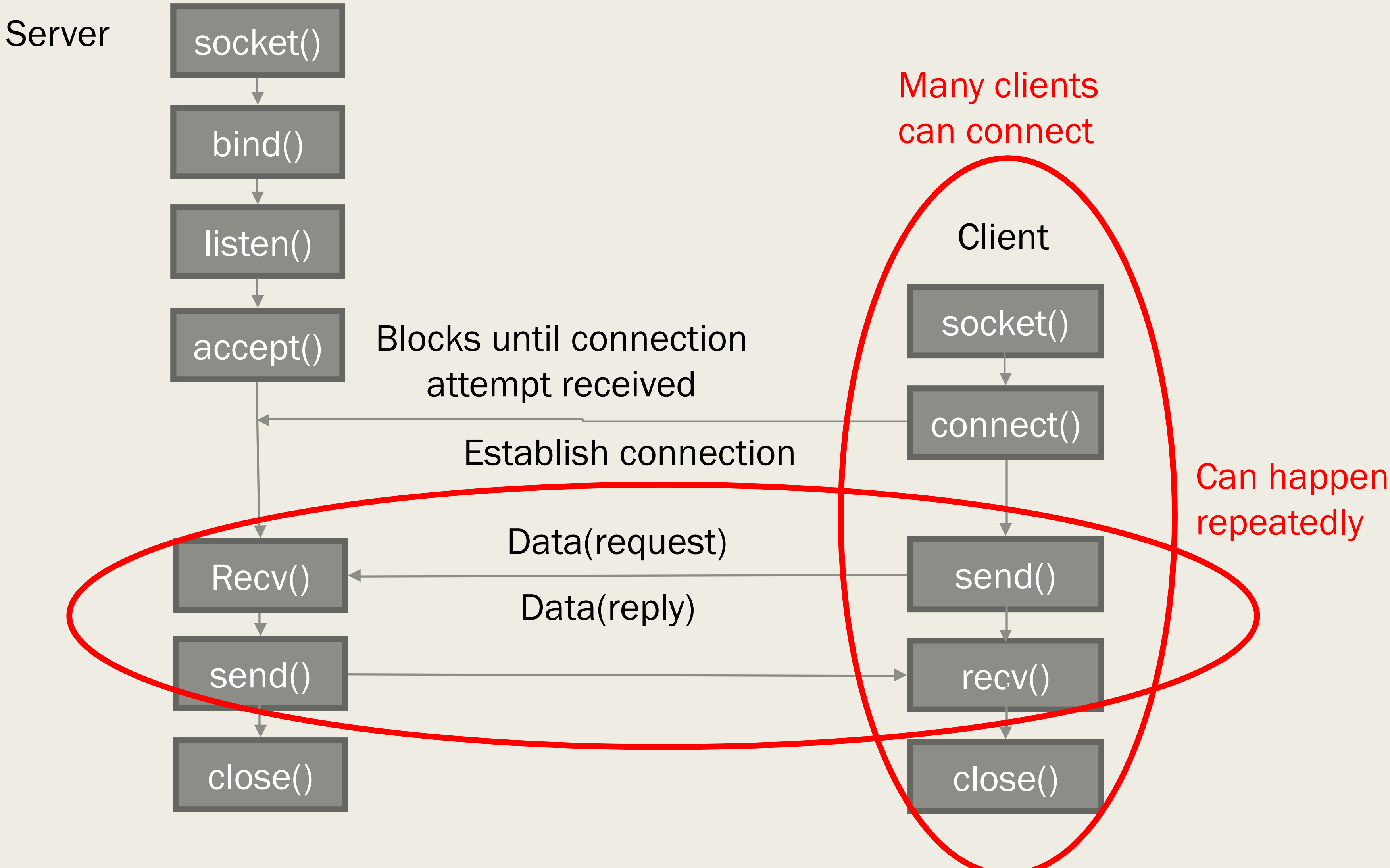
Telnet server (23)

- Resource: terminal
- Service: proxies a terminal on the server machine

Mail server (25)

- Resource: email "spool" file
- Service: stores mail messages in spool file

A Server-Client Interaction in TCP – POSIX Functions



A Closer Look into POSIX Functions

- `getaddrinfo()`
- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `write(), send(), sendto()`
- `read(), recv(), recvfrom()`
- `close()`

Data Structures

```
/* structure for looking up IP address */
struct addrinfo{
    int     ai_flags;
    int     ai_family; // AF_INET=IPv4,AF_INET6= IPv6
    int     ai_socktype;       // TCP or UDP
    int     ai_protocol;
    socklen_t  ai_addrlen;     // length of ai_addr
    struct sockaddr* ai_addr; // contains IP+PORT
    char*   ai_canonname;      // canonical name
    struct addrinfo* ai_next; // next pointer of result
link list
};
/* data structure for IP details (+PORT) */
struct sockaddr_in{
    short      sin_family;  // IPv4 or IPv6
    unsigned short sin_port; // port number
    struct in_addr sin_addr; // 32 bit IP
    char sin_zero [8];
};
/* just a wrapper for the numeric IP */
struct in_addr{
    unsigned long s_addr;
};
```

getaddrinfo() – Looking Up IP address from name

- **The first step to locate a server by the client**
- **Converts easy-to-remember DNS names (e.g., linux.cs.tamu.edu) into machine-usable IP address**
- **Queries DNS servers (a collection of mappings)**

  ```
  int getaddrinfo(char* name, char* port, struct addrinfo* hints, struct addrinfo** result);
  ```

  `name` = name of the host

  `port` = the port where the service (e.g., http, your data server in MP6) is running

  `hints` = provides some initial hint (IPv4/IPv6, TCP/UDP etc.)

  result = linked list of looked up addresses
- **Example:**

  ```
  getaddrinfo("www.example.com", "3490", &hints, &res);
  ```

getaddrinfo() - Detailed

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

//preparing hints data structure
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

// look up the IP address from the name: "www.example.com"
status = getaddrinfo("www.example.net", "3490", &hints, &servinfo);

for(p = res;p != NULL; p = p->ai_next) {
      void *addr;
      char *ipver;
      // get the pointer to the address itself,
      // different fields in IPv4 and IPv6:
      if (p->ai_family == AF_INET) { // IPv4
            struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
            addr = &(ipv4->sin_addr);
            ipver = "IPv4";
      } else { // IPv6
            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
            addr = &(ipv6->sin6_addr);
            ipver = "IPv6";
      }
      // convert the IP to a string and print it:
      inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
      printf("  %s: %s\n", ipver, ipstr);
}
```

socket() – A Connection End Point

- **Creates a communication end-point for a network connection**

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

  domain = PF_INET (IPv4) / PF_INET6 (IPv6)

  type = SOCK_STREAM (TCP) / SOCK_DGRAM (UDP)

  protocol = 0
- **Example:**

```
s = socket (AF_INET, SOCK_STREAM, 0)
```

will create a TCP socket

- **The above call returns 0 on success and -1 on failure**

connect() – Client Attempting Server Connection

- **This is called by the client to attempt a connection with the server**
- **Blocks until the server accepts it**

  `int connect(int sockfd, struct sockaddr* server, socketlen_t server_len)`

`Sockfd: socket variable prepared beforehand`

`server = address of the server (returned by getaddrinfo)`

- **Example:**

```
getaddrinfo("www.example.com", "3490", &hints, &res); // lookup
// make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
// connect to server. Once successful, the socket becomes ready as the endpoint
connect(sockfd, res->ai_addr, res->ai_addrlen);
```

close() – Close a Session

- **Called by both the client and the server**
- **Signals end of a communication**
- **Internally, frees resources associated with a connection**
  - **Important for busy servers, also for PA7**

    `int close(int sock)`
- **Example:**

```
close (sock)
```

bind() – Server Attaching to a Port

- **A server process calls this to associate its socket to a given port**
- **Port number is used by the kernel to forward an incoming packet to a certain process (its socket)**

  `int bind(int sockfd, struct sockaddr* addr, socketlen_t addrlen)`

- **Example:**

```
getaddrinfo(NULL, "3490", &hints, &res); // lookup
// make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
// bind it to the port we passed in to getaddrinfo():
bind(sockfd, res->ai_addr, res->ai_addrlen);
```

listen() – Setting up Server

- **This is a prerequisite before a connection is accepted**
- **Incoming connections wait in a queue before accepted, listen () sets the size of that queue**

  `int listen(int sockfd, int backlog)`

- **Example:**

```
listen (sockfd, 20); // should be replaced by w in your PA6
```

accept() – Server Accepting Client Connection

- **This is called by the server to accept a new client connection**

  int accept(int sockfd, struct sockaddr* client, socketlen_t client_len)

sockfd = socket

client = will hold the client address details

client_len = address length

- **Example:**

```
struct sockaddr client;
accept (sockfd, &client, sizeof (client));
```

send()/recv() – Finally Data

- **Called by both client and server to exchange data**
- **Blocks until the server accepts it**

```
int send(int sock, void* msg, size_t len, int flags)
int recv(int sock, void* msg, size_t len, int flags)
Msg = buffer pointer to send/receive data from/to
Len = sender: length of the message,
      receiver: buffer capacity (to avoid overflow)
```

- **Example:**

```
char *send_msg = "a sample message";
int sent_bytes = send (sockfd, send_msg, strlen (send_msg)+1, 0);

char recv_buffer [1024];
int recv_len = recv (sockfd, recv_buffer, 1024, 0);
```