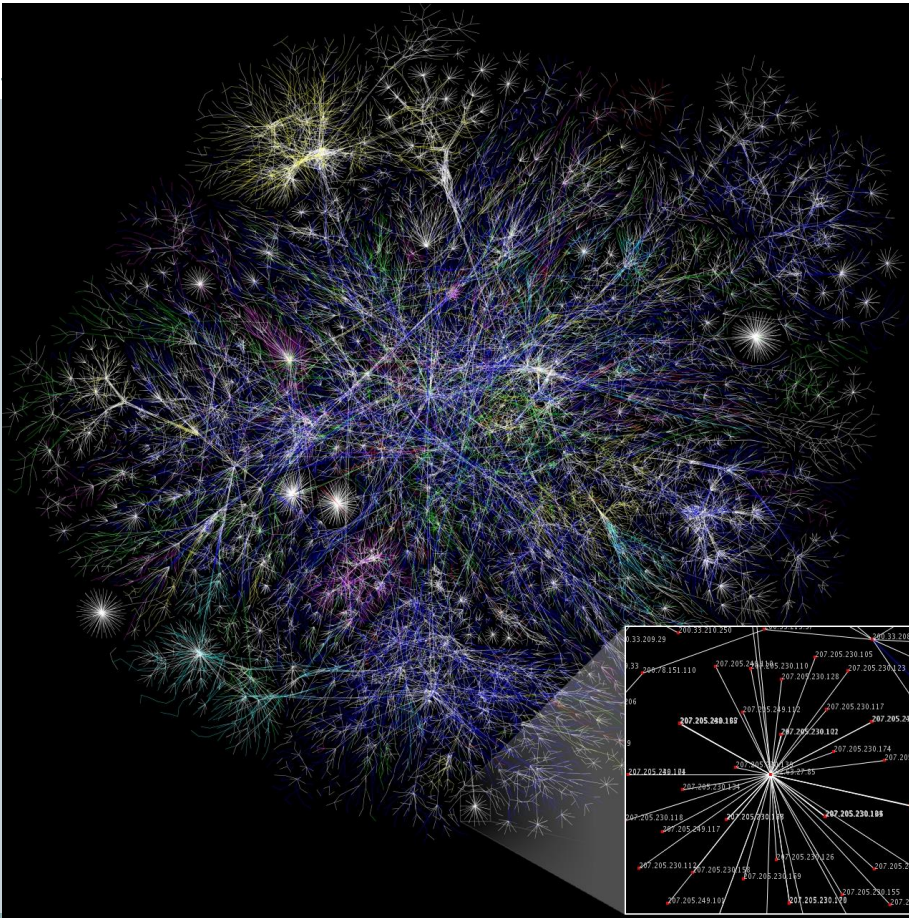


# Networking and Socket Communication



# Outline

- Networking basics
  - Difference between: **clients** and **servers**
  - **Addressing**
    - ✦ IP addresses, hostnames, DNS
    - ✦ Private addresses, localhost
  - **Port numbers**
- Socket communication
  - **Byte-level communication** between two hosts
  - Java client: reading/writing text
  - Java server: accepting clients, reading/writing text
- Single threaded examples
  - Magic-8 ball
  - Magic-8 ball persistent
- Multi-threaded servers
  - Magic-8 ball multi-threaded server
  - Shared key/value server

## Clients and Servers

- Client program

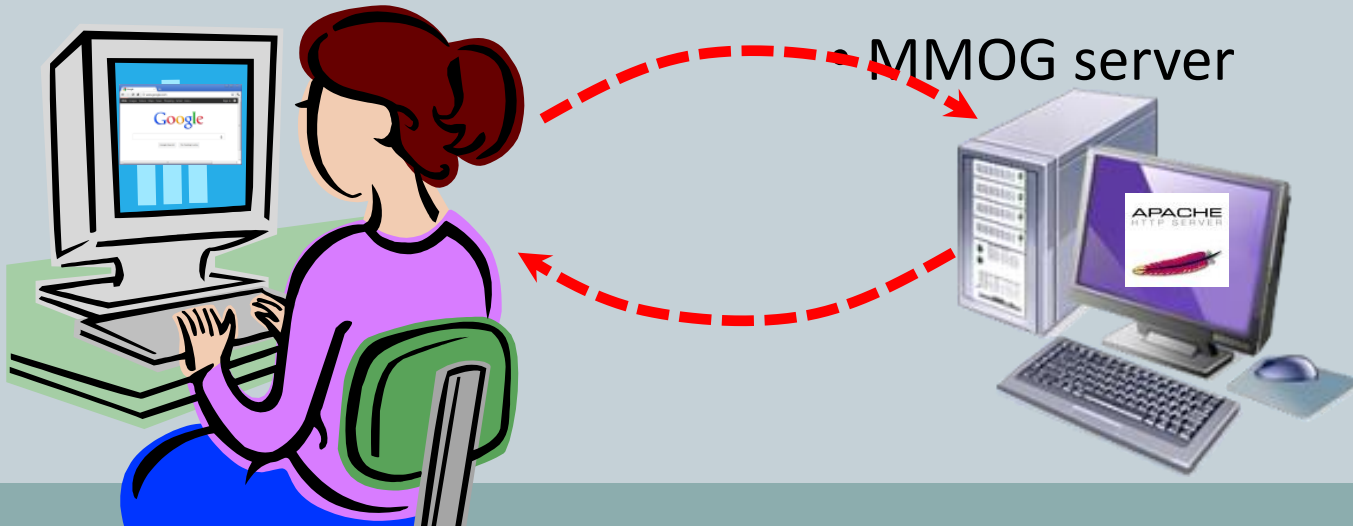
- Requests a service

- ✦ Web browser
    - ✦ Streaming audio player
    - ✦ Twitter client
    - ✦ MMOG client

- Server program

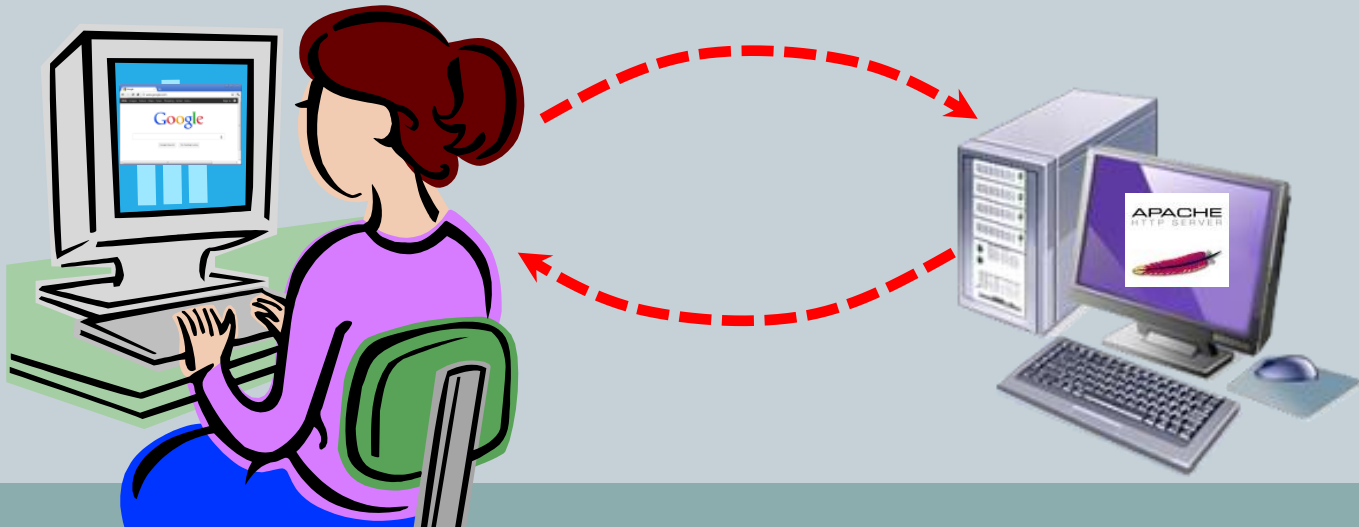
- Provides a service

- Web server
    - Streaming audio from radio station
    - Server at Twitter
    - MMOG server



# Clients and Servers

- Client program
  - "sometimes on"
  - Doesn't talk to other clients
  - Needs to know server's address
- Server program
  - "always on"
  - Handles requests from many clients
  - Needs fixed address



# Communication Components

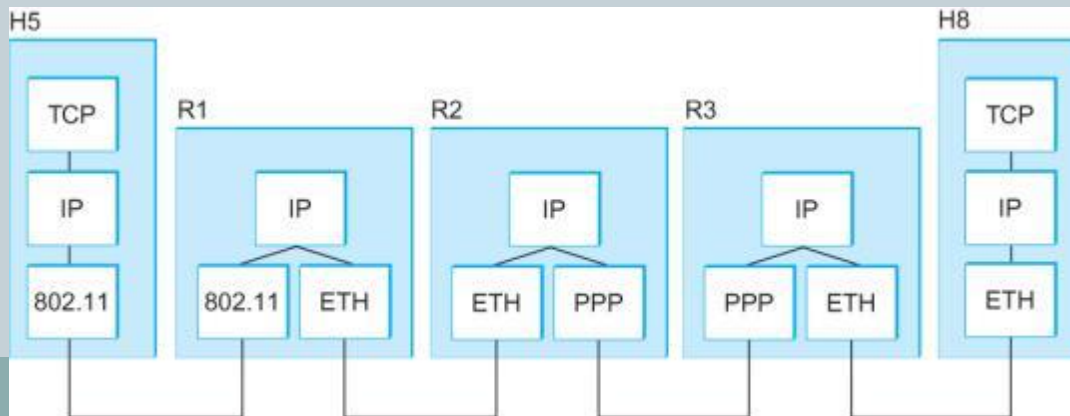
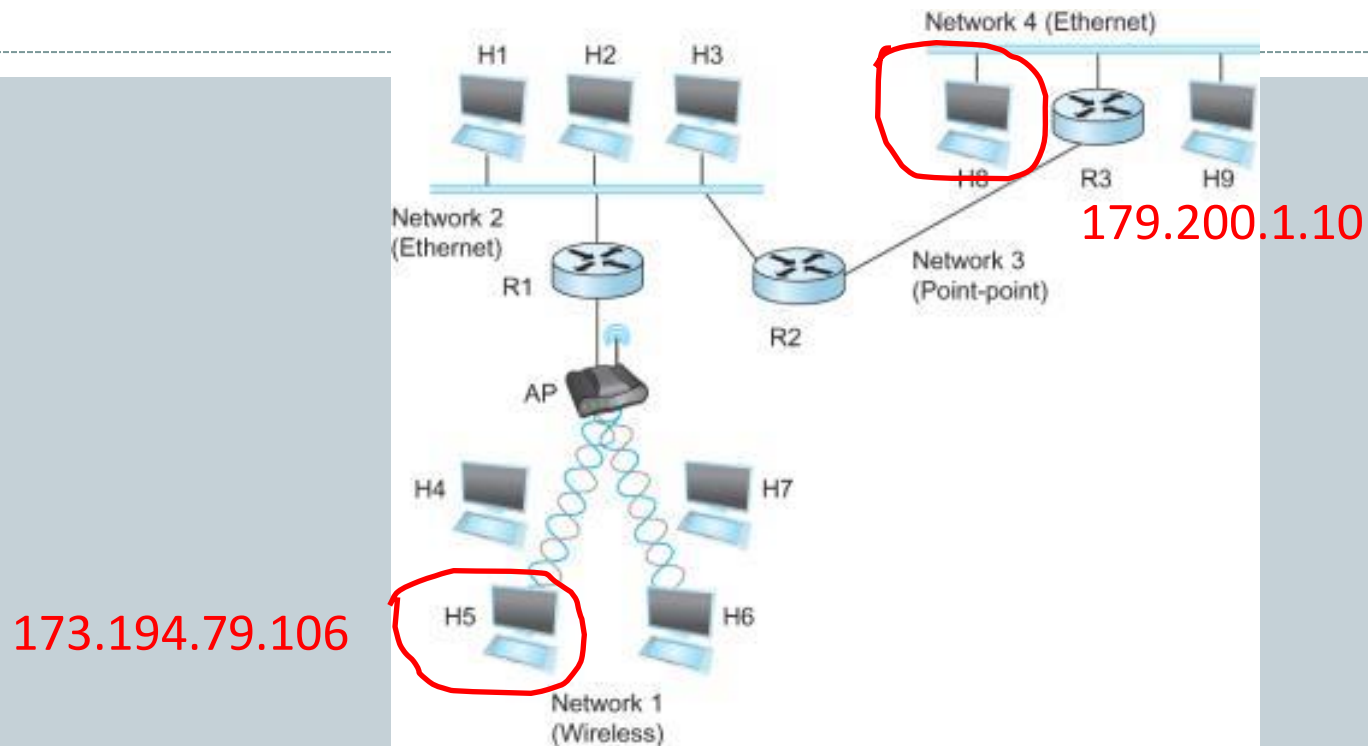
- Network
  - Transports data from source to destination host
  - Uses destination IP address
- Operating system
  - Data is forwarded to a "silo" based on port #
    - ✦ e.g. Port 80 requests routed to the web server program
- Application
  - Actually reads and writes to socket
  - Implements application-specific "magic"
    - ✦ e.g. Implementing a mail reading/writing protocol
    - ✦ e.g. Implementing a file retrieval (FTP) protocol
    - ✦ e.g. Implementing a particular online game

# Naming Computers

- **Goal: Establish communication between A and B**
  - How do computer A and B refer to each other?
  - The network needs an addressing system
- **IP (Internet Protocol) address**
  - IPv4 address
    - ✦ 32 bits ~ 4 billion hosts
    - ✦ Usually expressed as four numbers 0-255 (8 bits)
    - ✦ e.g. 173.194.79.106
  - IP address uniquely identifies a network endpoint
  - Devices inside network (e.g. switches, routers) use a packet's IP address to get it to its destination



## Communication from H5 to H8



# DNS – Domain Name System

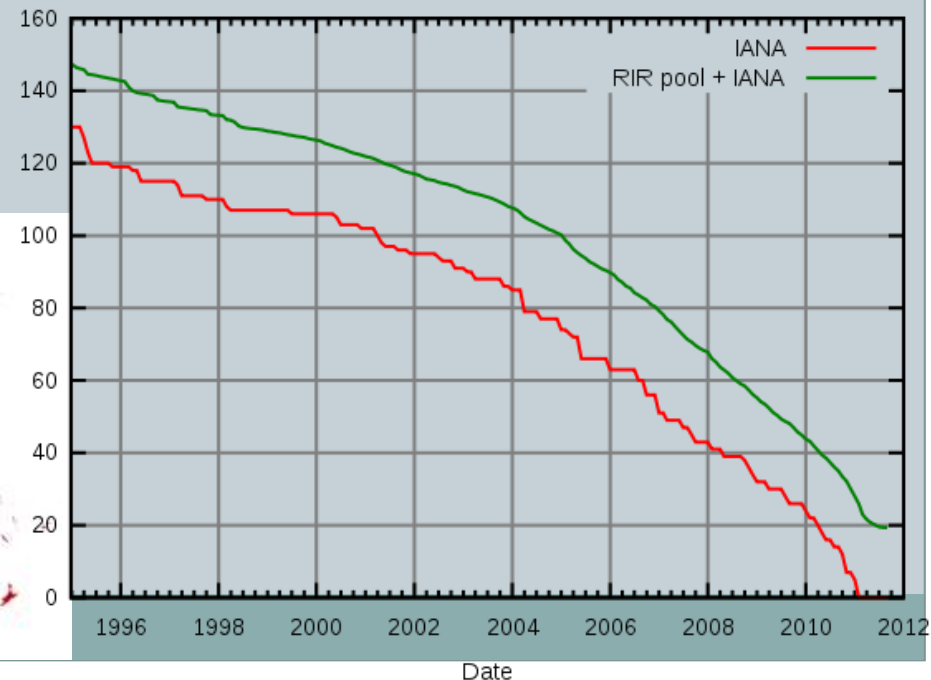
- Problem 1: Humans can't remember all the numbers in an IP address
- Domain Name System (DNS)
  - Converts readable name to numeric IP address
    - ✦ e.g. `www.google.com` -> `173.194.79.106`





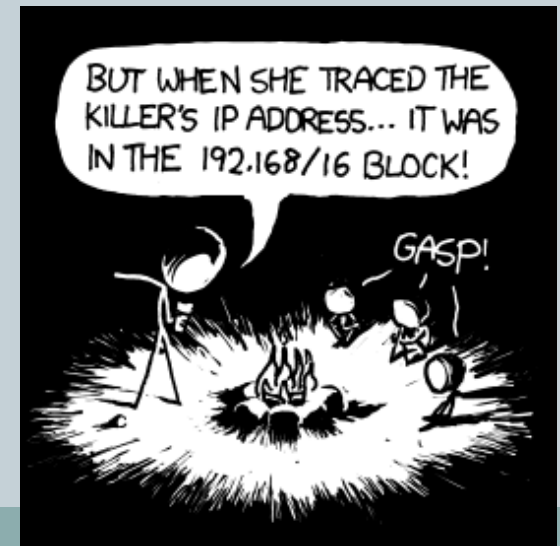
## IPv4 exhaustion

- **Problem 2: IPv4 only has 4 billion addresses**
  - 7 billion people, all want a laptop, Xbox & iPhone
- **Jan. 31, 2011**
  - Last unreserved IANA /8 blocks allocated
  - 5 remaining blocks allocated to Regional Internet registries (RIR)
  - IPv6 went live in 2012



# Private IP addresses

- Private IP addresses
  - Allow construction of a private network
    - ✦ Route data between endpoints on the private network
    - ✦ Addresses aren't valid outside network
    - ✦ 192.168.x.x, 10.x.x.x, 172.16/31.x.x
  - Typically what you'll have:
    - ✦ On home network
    - ✦ On campus network (wired/wireless)
  - 127.0.0.1 (localhost)



# Port Numbers

- **Problem 3: Many apps on same computer want to talk at same time**
  - Chrome process:
    - ✦ Browser tab 1 wants: `http://google.com`
    - ✦ Browser tab 2 wants: `http://google.com/gmail`
    - ✦ Browser tab 3 wants: `http://facebook.com`
  - Thunderbird process:
    - ✦ Email client wants IMAP4 to `techmail.mtech.edu`
- **Solution: Use IP address + port number**
  - A 16-bit number: 0 - 65535
    - ✦ Port number determines app message is routed to
    - ✦ Just a "virtual" port, only exists in the OS

## Port Numbers

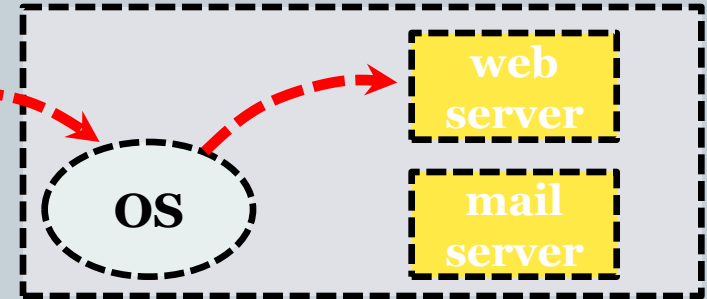
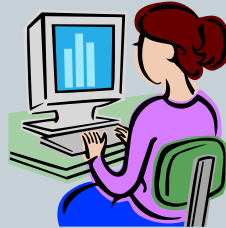
- Popular applications have known ports
  - Ports 0 - 1023: reserved for well-known services
    - ✦ Only administrators can start servers on these ports
  - Ports 1024 - 65535: available to any user-level application

Port	Service
21	File transfer protocol (FTP)
22	Secure shell (SSH)
23	Telnet
25	Simple mail transfer protocol (SMTP)
53	Domain name system (DNS)
80	Hypertext transfer protocol (HTTP)
110	Post office protocol (POP)
143	Internet message access protocol (IMAP)
443	HTTP secure (HTTPS)

## Use of Port Number

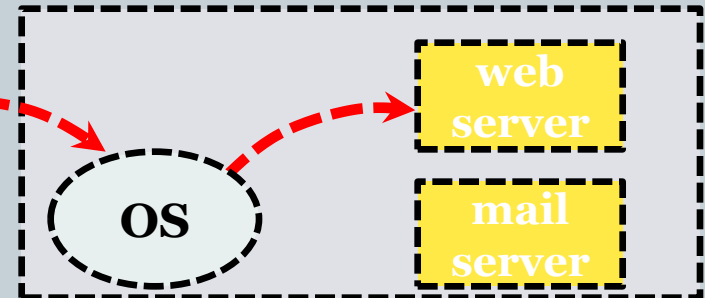
Requesting a non-secure web page

192.168.23.100:80



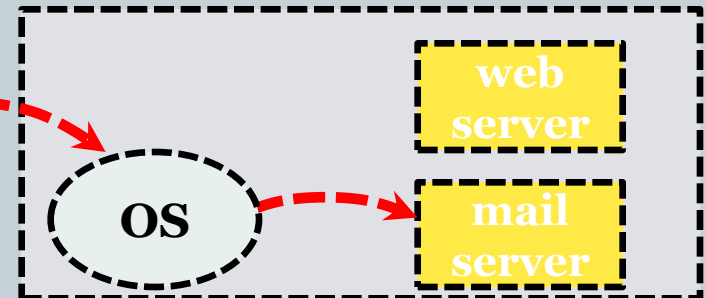
Requesting a secure web page

192.168.23.100:443



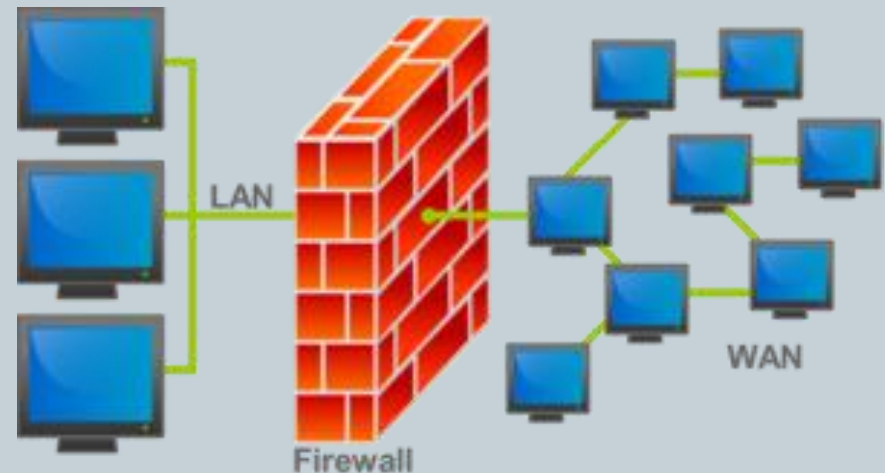
Requesting new email messages

192.168.23.100:143



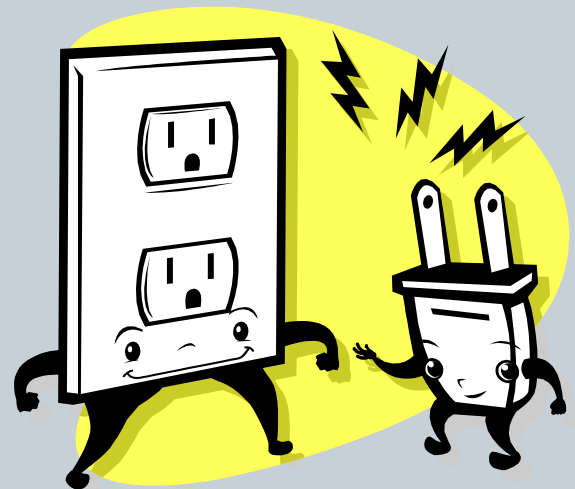
## Firewalls

- **Problem 4: You can't always get there from here:**
  - Communication may be filtered by network
    - ✦ e.g. by a firewall at the border of Tech's network
    - ✦ e.g. by the wireless access point in Main Hall
  - Often by the port number



## Sockets

- **Socket API** (Application Programming Interface)
  - Allows communication over IP (Internet Protocol)
  - Originally in **Berkeley Unix**
    - Thus: Berkeley sockets or BSD sockets
  - **De facto standard** in all operating systems
  - API in most programming languages:
    - C/C++
    - Java
    - C#
    - ...





# Java Client: Reading from a Socket

- **Step 1: Create a new Socket object**

- Needs to know IP address of server + port number

```
Socket socket = new Socket("127.0.0.1", 5000);
```

- **Step 2: Create an InputStreamReader**

- Converts low-level socket data into characters stream

```
InputStreamReader stream = new InputStreamReader(socket.getInputStream());
```

- **Step 3: Create a BufferedReader**

- Provides buffered reading of character stream

```
BufferedReader reader = new BufferedReader(stream);
```

- **Step 4: Read some text**

```
String message = reader.readLine();
```

## BufferedReader

### Method Summary

void	<u>close</u> ()	Close the stream
void	<u>mark</u> (int readAheadLimit)	Mark the present position in the stream.
boolean	<u>markSupported</u> ()	Tell whether this stream supports the mark() operation, which it does.
int	<u>read</u> ()	Read a single character.
int	<u>read</u> (char[] cbuf, int off, int len)	Read characters into a portion of an array.
<u>String</u>	<u>readLine</u> ()	Read a line of text.
boolean	<u>ready</u> ()	Tell whether this stream is ready to be read.
void	<u>reset</u> ()	Reset the stream to the most recent mark.
long	<u>skip</u> (long n)	Skip characters.

# Java Client: Writing to a Socket

- **Step 1: Create a new Socket object**
  - Or use an existing one
  - You can combine reads and writes to same socket

```
Socket socket = new Socket("127.0.0.1", 5000);
```

- **Step 2: Create an PrintWriter**
  - Seen previously when writing to a file

```
PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
```

- **Step 3: Write something**

```
writer.println("Hello over there!");
```

## PrintWriter

void	<u>print</u> (double d) Prints a double-precision floating-point number.
void	<u>print</u> (float f) Prints a floating-point number.
void	<u>print</u> (int i) Prints an integer.
void	<u>print</u> (long l) Prints a long integer.
void	<u>print</u> (Object obj) Prints an object.
void	<u>print</u> (String s) Prints a string.
<u>PrintWriter</u>	<u>printf</u> (Locale l, String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
<u>PrintWriter</u>	<u>printf</u> (String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
void	<u>println</u> () Terminates the current line by writing the line separator string.
void	<u>println</u> (boolean x) Prints a boolean value and then terminates the line.
void	<u>println</u> (char x) Prints a character and then terminates the line.
void	<u>println</u> (char[] x) Prints an array of characters and then terminates the line.

Just some of  
the methods in  
PrintWriter

# Java Socket Server

- Client needs somebody to talk to!
- Server slightly different than client:
  - Must be **running before client** connects
  - Server **decides port number** to listen on
    - ✦ But doesn't specify IP address
    - ✦ Doesn't know who is going to connect
  - Blocks, waiting to ***accept*** an incoming client
  - Then reading/writing just as in client

# Java Socket Server

- **Step 1: Create a ServerSocket object**
  - Declares what port you are listening on
  - Nobody else on the computer better be using it!

```
ServerSocket serverSock = new ServerSocket(5000);
```

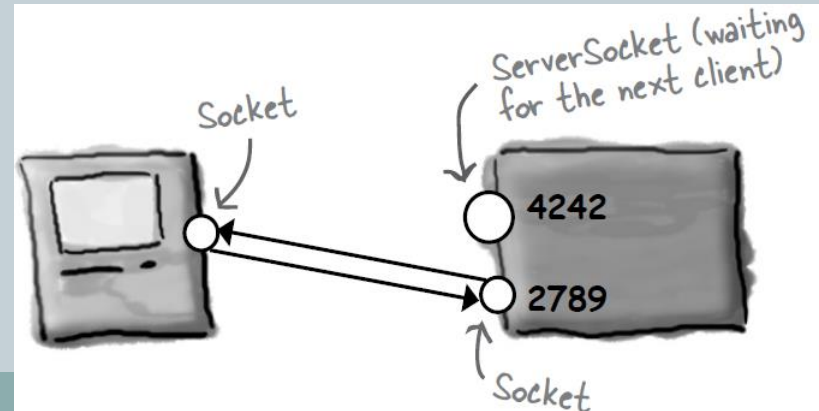
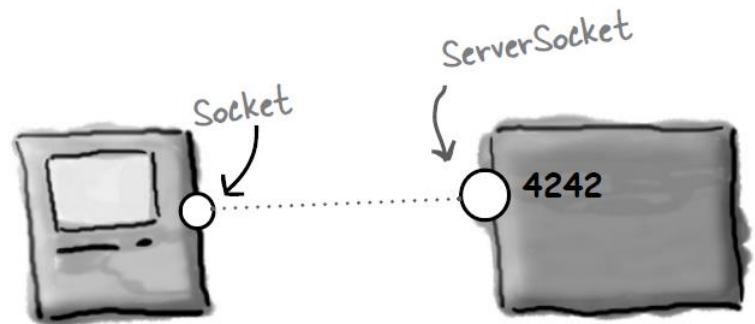
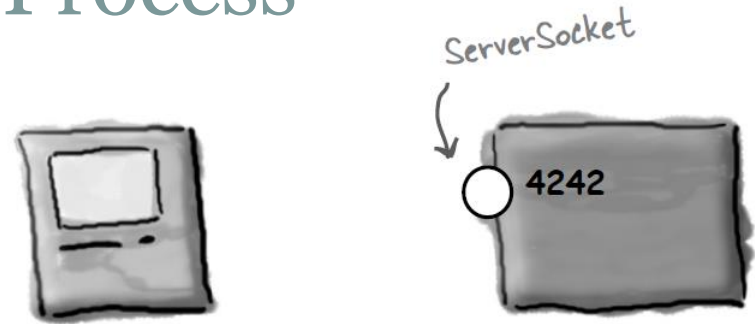
- **Step 2: Wait for a client to connect**
  - `accept()` method blocks until client arrives
  - Returns a new `Socket` object for talking to client

```
Socket sock = serverSock.accept();
```

- **Step 3: Read/write same way as a client**
  - Create `BufferedReader` for reading strings
  - Create `PrintWriter` for writing strings

## Connection Process

1. Server program starts up.
2. Starts listening on port 4242.
3. OS sends all inbound connection requests to 4242 to the server program.
4. Client program starts up
5. Requests connection to server IP address on port 4242.
6. Server establishes a socket connection to client, using outgoing port number 2789
7. Server can listen for new clients on the 4242 port number.





# Magic 8 ball: Internet Edition

- **Server:**

- katie.mtech.edu – dept. server
- Public IP address
- Running on port 5000
- Delivers 1 of 20 messages at random

- **Client:**

- My laptop on the wireless network
- Your desktop on the wired network
  - ✦ Both have a private IP address
- Displays message from the server



**Client1 @ 192.168.1.100**



```
% java Magic8Client  
150.131.202.152  
5000
```

**Server @ 150.131.202.152**  
**katie.mtech.edu**



```
% java Magic8Server  
5000
```

**Client2 @ 192.168.1.110**



```
% java ValueClient  
katie.mtech.edu 6000
```

**Client3 @ 192.168.1.120**



```
% java ValueClient  
150.131.202.152 6000
```

```
% java ValueServer 6000
```

Thread 1

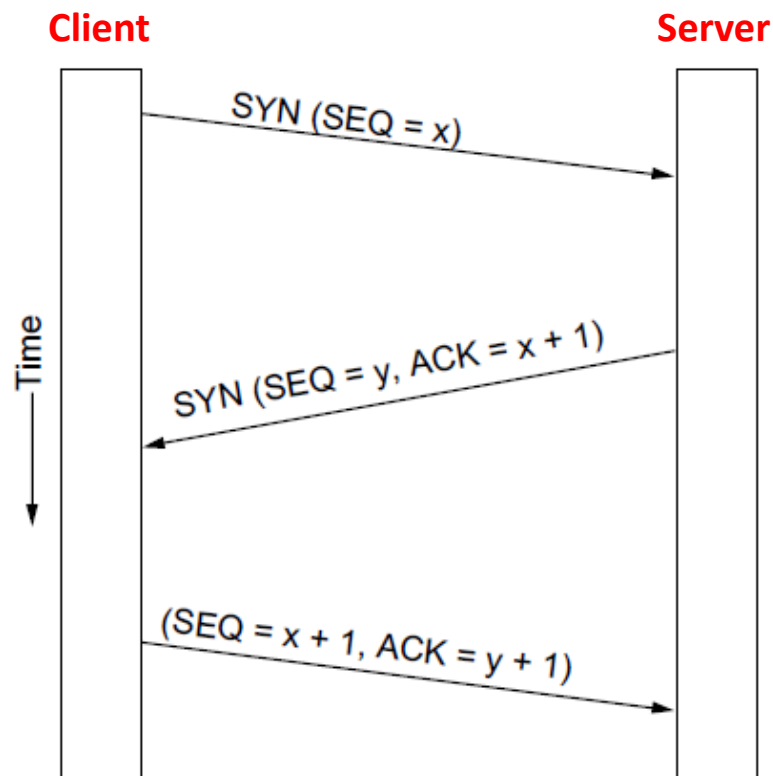
Thread  
2

# Communication Reliability

- **Socket communication protocol:**
  - We'll use **TCP** (Transmission Control Protocol)
  - **TCP/IP** = TCP over IP (Internet Protocol)
  - **IP protocol:**
    - ✦ De facto standard for Internet communication
    - ✦ But: only provides "best effort" delivery
      - Messages may or may not get there
      - Messages may get reordered in transit
  - Luckily: **TCP provides reliable in-order** delivery
    - ✦ You can be sure what you read/write will get there (unless something really bad happens)

# Establishing a Connection

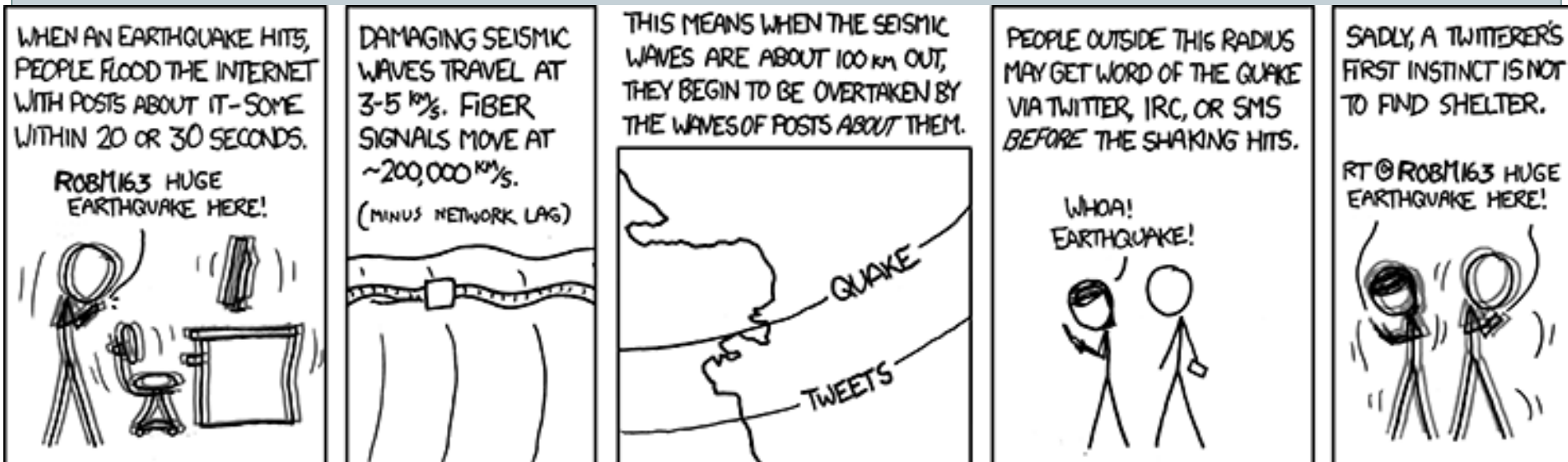
- Starting a socket connection:
  - 3-way handshake
  - Connection takes a bit to startup
  - Keep around if you have an ongoing conversation



## Latency

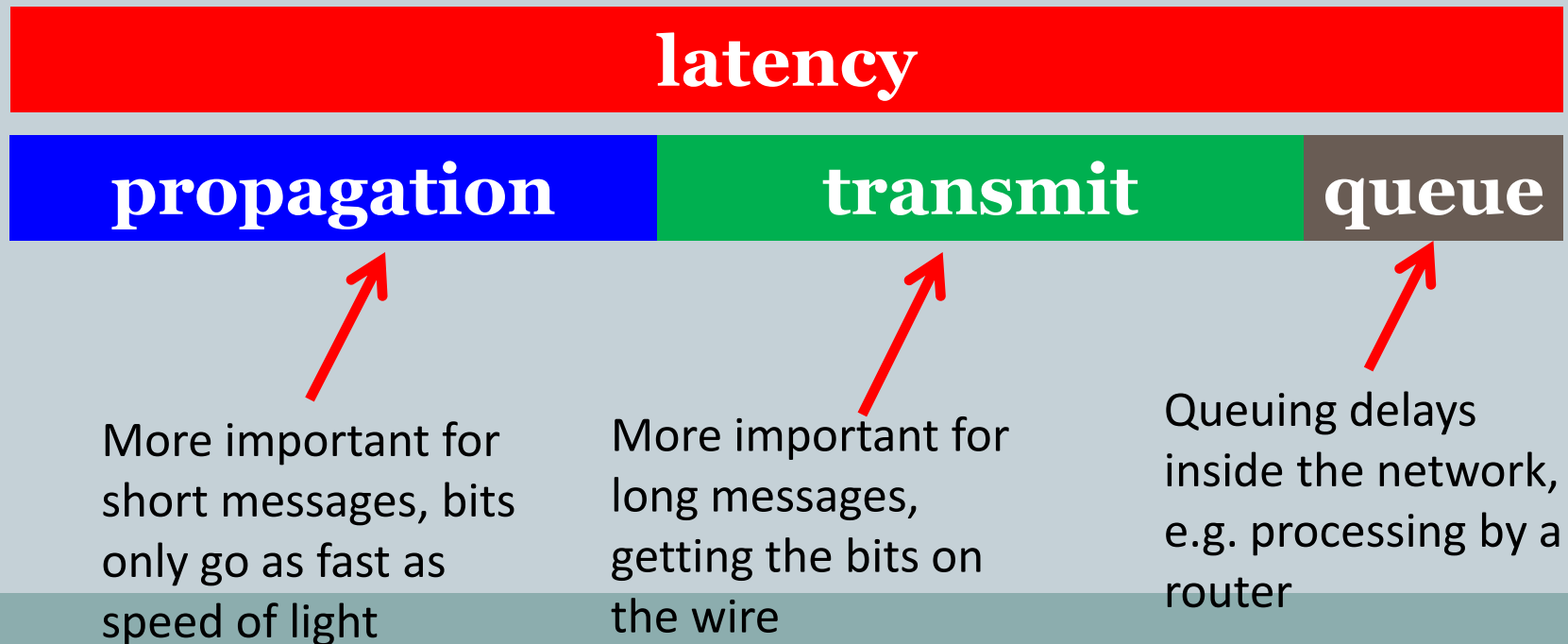
- Signals can only go so fast:

Medium	Speed of light
Vacuum	$3.0 \times 10^8$ m/s
Copper cable	$2.3 \times 10^8$ m/s
Optical fiber	$2.0 \times 10^8$ m/s



# Latency

- latency = propagation + transmit + queue
- propagation = distance / speed of light
- transmit = size / bandwidth



## Firewalls

- Network hardware/software may interfere
  - e.g. Hosts on the same network but traffic blocked for certain port numbers

Keith's laptop  
10.1.20.100



katie.mtech.edu  
150.131.202.152

William's laptop  
10.1.20.101

```
% java Magic8Client  
10.1.20.100 5000
```

```
% java Magic8Server  
5000
```

This failed, wireless access point (AP) blocked attempt to connect to port 5000 on 10.1.20.100



# Handy network utilities

- ping <hostname or IP address>
  - Test if you can reach the destination
    - ✦ Time for a tiny message to go there and come back
      - Round Trip Time (RTT)
  - ✦ Note: some hosts may disable responding to pings

```
% ping keithv.com
```

```
Ping % ping katie.mtech.edu
```

```
Rep % ping bbc.co.uk
```

```
Rep Pinging bbc.co.uk [212.58.241.131] with 32 bytes of data:  
Rep Request time=162ms TTL=229  
Rep Request time=160ms TTL=229  
Ping Request time=162ms TTL=229  
Request time=163ms TTL=229  
Appr Ping statistics for 212.58.241.131:
```

```
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 160ms, Maximum = 163ms, Average = 161ms
```

## Handy network utilities

- **ipconfig (Windows), ifconfig (Mac/unix)**
  - Find out your wired/wireless IP address

```
c:\ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```
Connection-specific DNS Suffix  . : passcall
Link-local IPv6 Address . . . . . : fe80::615f:559:cfb6:8d35%10
IPv4 Address. . . . . : 192.168.1.6
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
```

# Magic 8 ball: Internet Edition

- **Server:**
  - `katie.mtech.edu`, **public IP address**
  - Running on **port 5000**
  - Delivering 1 of 20 messages
  - Services a **single client at a time**
- **Client(s):**
  - My laptop on the wireless network
  - Your laptop on the wireless network
  - **Private IP address**
  - Displays message from the server



# Magic 8 ball: Persistent Connections

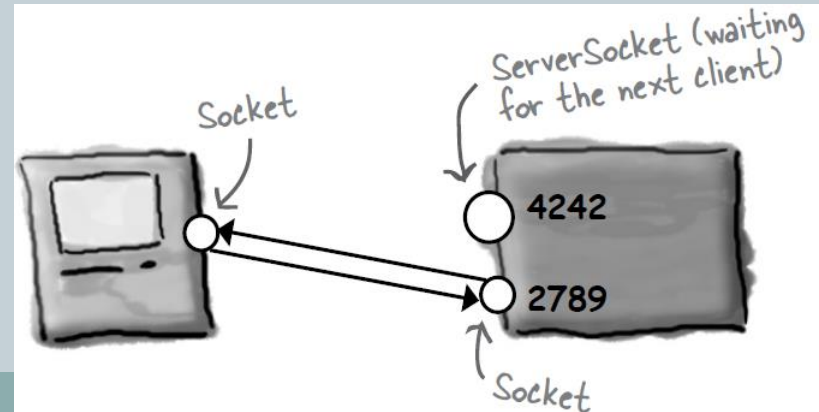
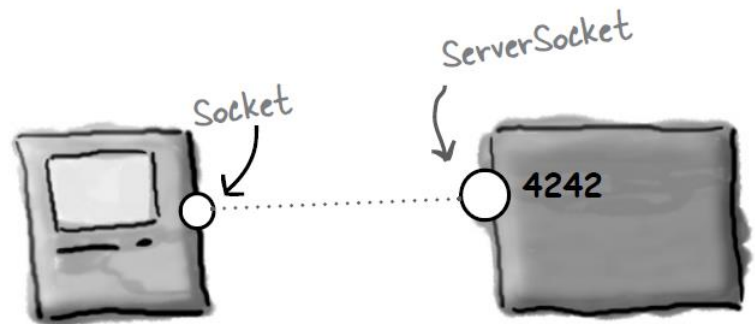
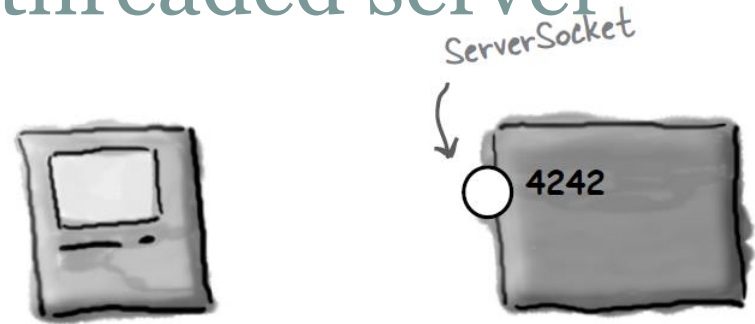
- Original version: One prediction per connection
- Persistent version:
  - A *protocol* between client and server

Client	Server
	Wait for client
Make connection to server	
Send name of user	
	Send first fortune
Receive first fortune	
Send "MORE"	
	Receive command "MORE"
	Send second fortune
Receive second fortune	
Send "QUIT"	
Close socket	Receive command "QUIT"
	Close socket

## MULTI-THREADED EXAMPLE

# Magic 8 ball: Multi-threaded server

- Problem with persistent version:
  - One client can hog the 8-ball for a long time
- Multi-threaded server:
  - Spawn a **thread** to handle **each client**
  - Server's main thread can then **wait for a new client**



# Summary

- **Networking basics**
  - Difference between: **clients** and **servers**
  - **Addressing**
    - ✦ IP addresses, hostnames, DNS
    - ✦ Private addresses, localhost
  - **Port numbers**
- **Socket communication**
  - **Byte-level communication** between two hosts
  - Java client: reading/writing text
  - Java server: accepting clients, reading/writing text
- **Single threaded examples**
  - Magic-8 ball
  - Magic-8 ball persistent
- **Multi-threaded servers**
  - Magic-8 ball multi-threaded server
  - Shared key/value server

