Today's Objectives

- Web Server implementation wrap up
- Designing distributed systems
- Introduction to networking
- Networking layers

Review

- What is the general process for a server?
- What is the general process for a client?
- What does Java provide as an abstraction for network communication?
- How does a web server work?

WEB SERVER


```
public static void main(String[] args) {
    try {
        ServerSocket server = new ServerSocket(1999);
        Socket incoming = server.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                        incoming.getInputStream()));
        PrintWriter out = new PrintWriter(
                incoming.getOutputStream(), true);
        out.println("Echo Server. Type BYE to exit");
        String line = null;
        while ((line = in.readLine()) != null) {
            out.println("Echo:" + line.trim());
        }
        incoming.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public static void main(String[] args) {
    try {
        ServerSocket server = new ServerSocket(1999);
        Socket incoming = server.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                        incoming.getInputStream()));
        PrintWriter out = new PrintWriter(
                incoming.getOutputStream(), true);
        out.println("Echo Server. Type BYE to exit");
        String line = null;
        while ((line = in.readLine()) != null) {
            out.println("Echo:" + line.trim());
        }
        incoming.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Issue to handle:
Something that says that client is done?


```
public static void main(String[] args) {
    try {
        ServerSocket server = new ServerSocket(1999);
        Socket incoming = server.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                        incoming.getInputStream()));
        PrintWriter out = new PrintWriter(
                incoming.getOutputStream(), true);
        out.println("Echo Server. Type BYE to exit");
        String line = null;
        while ((line = in.readLine()) != null) {
            if (line.trim().equals("BYE"))
                break;
            else
                out.println("Echo:" + line.trim());
        }
        incoming.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Servers and Multiple Clients

- Servers should handle multiple concurrent clients
- If server only allowed 1 client to connect at any given time, a client can monopolize the service by remaining connected to the server for a long time

What does this sound like a job for?

Clients and Servers

- Question: How do we service other clients?
  - We don't want to consume the server's resources with just one client…
- Three choices
  - Multiple threads
  - Multiple processes
  - Event queue

Multiple Threads Approach

- When server returns from `accept()` with the `Socket`, start a new thread to handle the new connection
- Main server thread can go back and call `accept()` again, waiting for a new client to connect

A Multithreaded Server

```
while (true) {
   Socket incoming = server.accept();
   Thread clientThread = new ThreadedEchoHandler(incoming);
   clientThread.start();
}
```

- User-defined `ThreadedEchoHandler` class derives from `Thread`
- Client communication loop is its `run()` method…

`MultiThreadedServer.java`
`ThreadedEchoHandler.java`

A Multithreaded Server Summary

- Each connection starts a new thread
  - Multiple clients can connect to the server at the same time
- As soon as a client connects, `accept`() returns a `Socket` that encapsulates this new connection
  - `socket` is passed into new thread to handle connection
  - The thread is then started and deals with the connection from then on
- The main thread goes back to waiting for a new connection

Multithreaded Server Issues

- Any problems with having a thread handle each incoming request?

Multithreaded Server Issues

- Any problems with having a thread handle each incoming request?
  - Performance

Multithreaded Server Issues

- For each request, must create a thread
  - Overhead in creating threads
- What happens if receive too many client requests and have to start/fork too many threads?
  - Machine runs out of memory
  - Machine gets bogged down
  - Threads can't make progress

Multi-threaded Server Solutions

Solution: limit # of incoming connections/threads available

- `ServerSocket( int port, int backlog )`
  - Maximum length of queue
  - After `backlog` requests, additional requests are refused
- Create a thread pool
  - Create available threads at startup
  - Get one of these threads when to handle requests
  - See `java.util.concurrent.Executors`

Socket Timeouts

- Reading from a socket indefinitely is a bad idea
  - Network could go down, causing program to wait on socket forever
- Java supports a timeout value
  - If program has been waiting for socket for specified timeout interval, an `Exception` is generated
  - Call `setSoTimeout()` on the socket, in ms

```
Socket socket = new Socket("host", 1998);
socket.setSoTimeout(10000); // 10 second timeout
```

Summary: Implementing a Server

- How do we create network connections?
  - Sockets!
  - Java uses `ServerSockets` and `Sockets` for clients
  - (C/C++ makes no distinction between client and server connections)
- How does the server support multiple clients at once?
  - Using multiple threads or processes
  - Using an event queue

Helpful Hints

- Check out the links on the assignment page
- Start small and test often
  - Small implementation, small tests
- Use telnet or nc for preliminary testing and experimentation
- Use browser:
  - http://localhost:8888/test.html

APACHE HTTPD CONFIGURATION

Apache Config File

On hydros
/etc/httpd

- Document Root

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this
# directory, but symbolic links and aliases may be used to point
# to other locations.
#
DocumentRoot "/var/www/html"
```

- Index

```
# DirectoryIndex: sets the file that Apache will serve if a
# directory is requested.
#
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

Apache Config File: public_html

```
<IfModule mod_userdir.c>
    #
    # UserDir is disabled by default since it can confirm
    # the presence of a username on the system (depending
    # on home directory permissions).
    #
    # To enable requests to /~user/ to serve the user's
    # public_html directory, remove the "UserDir disabled"
    # line above, and uncomment the following line instead:
    #
    UserDir /home/www/users
</IfModule>
```


DESIGNING DISTRIBUTED SYSTEMS

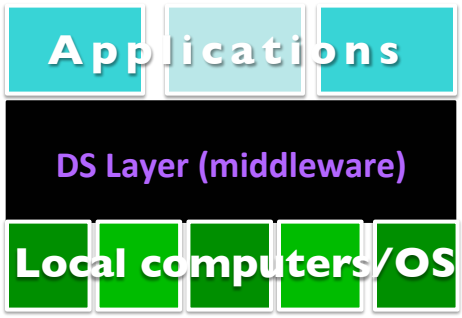
Design

- How can we design a Distributed System to hide heterogeneous computation, communication, etc.?

Review from CSCI209: How do we design for change? How do we design for lots of different "things"?

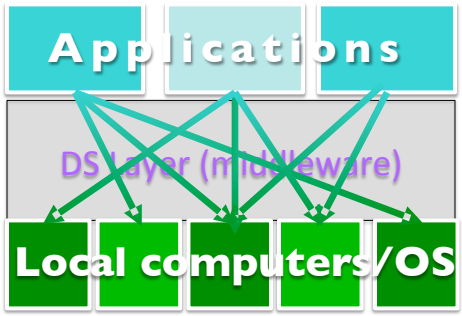
Abstraction!

Abstraction: Common Interface



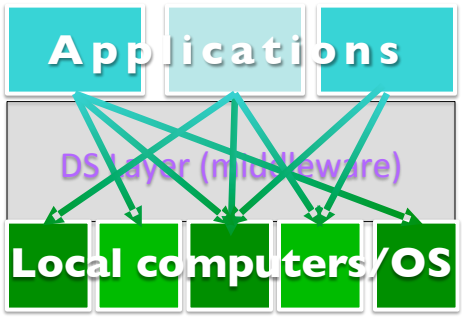
Black box:
Applications do not "see" individual computers

Abstraction: Common Interface



Black box:
Applications do not "see" individual computers

Abstraction: Common Interface

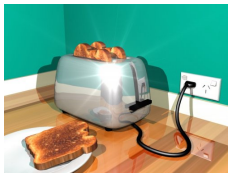


Black box:
Applications do not "see" individual computers

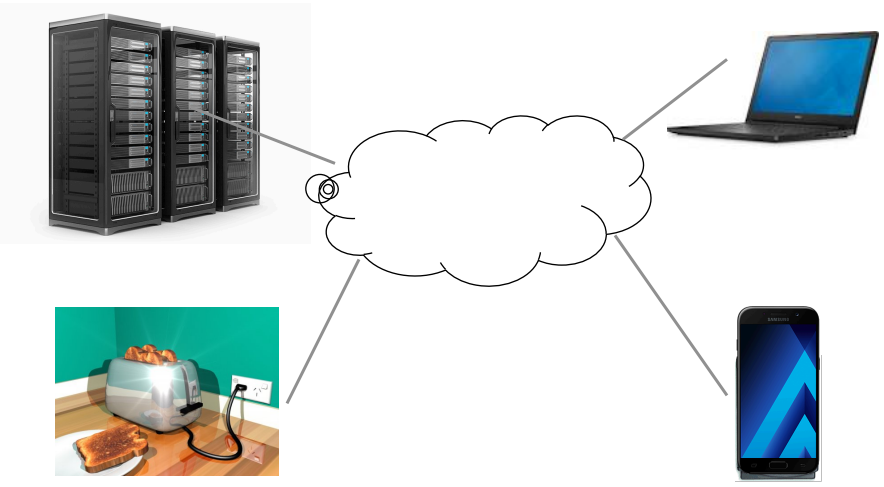
Similarly, how do we get lots of different types of machines to interact?

Networking Goal: Scalable, Arbitrary Communication

Networking Goal: Scalable, Arbitrary Communication

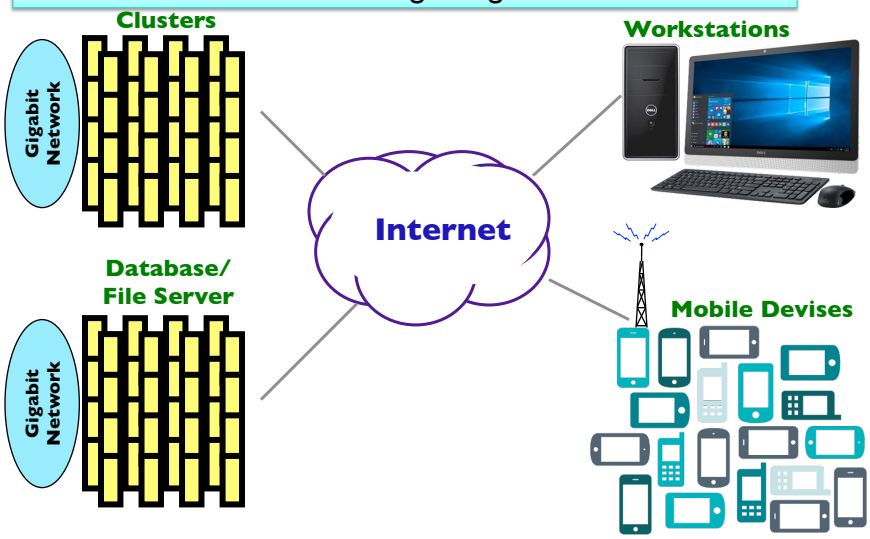


Networking Goal: Scalable, Arbitrary Communication



Distributed Systems Goals

Provide the illusion of a single large "virtual" machine



Layering in Network Design

- Fundamental Question: What services do applications require from the bare hardware?
  - Don't make each app implement the same functionality
  - OSs should implement these services/abstractions
- Network links hand a *frame* to the operating system
- But what abstraction does the application desire?
- Do all applications need the same abstraction?
- What abstractions do intermediate hosts (routers) in the network need?

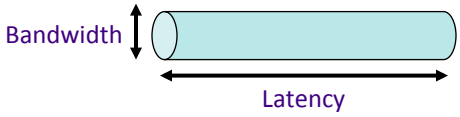
INTRODUCTION TO NETWORKING

Simple Network Model

- A ***network*** is a pipe connecting two computers

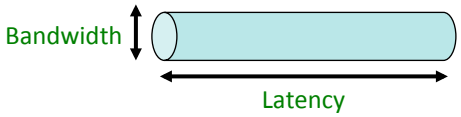
- Performance Metrics
  - Bandwidth, latency, overhead, error rate, message size

Performance Metrics



Metric	Description
Bandwidth	number of bits transmitted per unit of time
Latency	= Propagation + Transmit + Queue • Propagation delay = Distance/SpeedOfLight • Transmit time = Size/Bandwidth • Queue delay = Time packets spend in router queues
Overhead	Time for CPU to put message on wire
Error rate	Probability *p* that message will not arrive intact
Message size	Avg size of packets

Bandwidth vs. Latency → Transmission



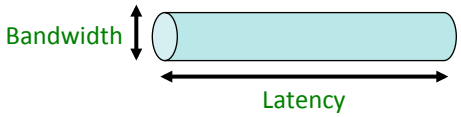
1 Byte Object

	Latency: 1 ms	Latency: 100 ms
Bandwidth: 1 Mbps		
Bandwidth: 100 Mbps		

10 MB Object

	Latency: 1 ms	Latency: 100 ms
Bandwidth: 1 Mbps		
Bandwidth: 100 Mbps		

Bandwidth vs. Latency → Transmission



1 Byte Object

	Latency: 1 ms	Latency: 100 ms
Bandwidth: 1 Mbps	1,008 μs	100,008 μs
Bandwidth: 100 Mbps	1,000 μs	100,000 μs

10 MB Object

	Latency: 1 ms	Latency: 100 ms
Bandwidth: 1 Mbps	80.001 s	80.1 s
Bandwidth: 100 Mbps	.801 s	.9 s

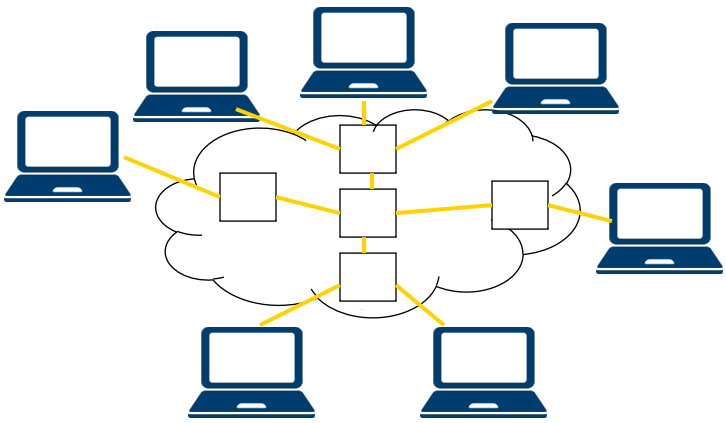
Types of Connections

- Point-to-point

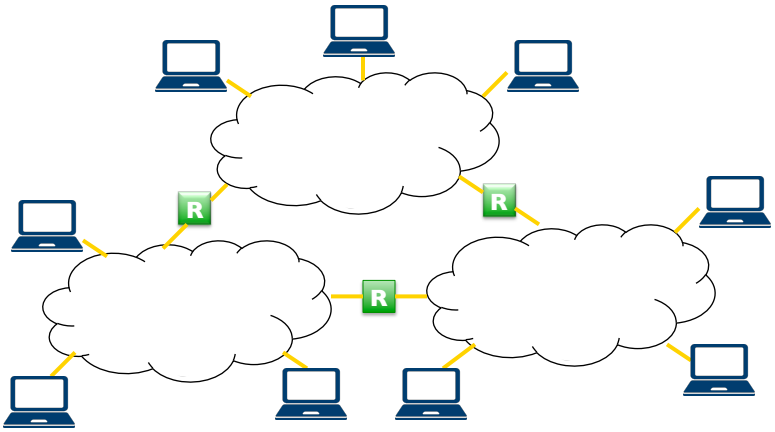
- Multiple access

Yellow line: link
→ May be wired or wireless

Types of Connections: Switched



Types of Connections: Interconnection of Networks



Internet: Network of Networks

- Network delivers packets and locates hosts
- Router (gateway) moves packets between networks
- Software layer: IP interoperability on top of any potential network or link layer
  - Modem, Ethernet, token ring, cell phone, ADSL, cable modem, smoke signals, …
- Minimum possible requirements on underlying networks

Formalizing Definitions

Term	Definition
Host	
Packet	
Link	
Switch	
Router	

Formalizing Definitions

Term	Definition
Host	Computer, mobile device, …
Packet	Unit of transmission across a network
Link	Used to transmit bits; pipe • Wired or wireless • Broadcast or switched (or both)
Switch	Used to move bits between links – Local Area Network (LAN) • Packet switching: stateless, store & forward • Circuit switching: stateful, cut through
Router	Connects networks across wide area (WAN)

TODO

- Read and summarize End to End argument
- Work on web server project in team
  - Figure out what needs to be done
  - Design solution
  - Break into pieces