8 Networking

If you have a wireless laptop, download the following programs:

```
http://www.cs.princeton.edu/introcs/84network/ChatClient.java
http://www.cs.princeton.edu/introcs/84network/In.java
http://www.cs.princeton.edu/introcs/84network/Out.java
```


Lecture Outline

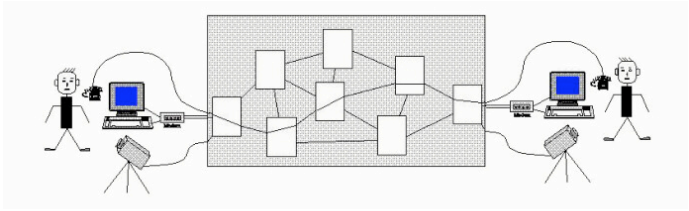
Networking.
- Communicate between different devices.
- Client-server.
- Peer-to-peer.

Networking in Java.
- Mail spoofing.
- Echo.
- Remote control.
- Chatroom.

Detours.
- Graphical user interface.
- Threads.

The Telephone Network

Telephone network.
- Circuit switching: single physical path between sender and receiver is dedicated for duration of communication.
- Has worked for 100 years.
- Advantage: real-time communication.



Internet

Internet.
- Global communication network containing million of computers.
- Computer networks communicate using TCP/IP protocol.
- Provides access to services: email, chat, world wide web, KaZaa.
- Started by military around 1969 as ARPANET: survivability, robustness, efficiency.
- Operating system and hardware independent.

Everybody but you grew up without it!

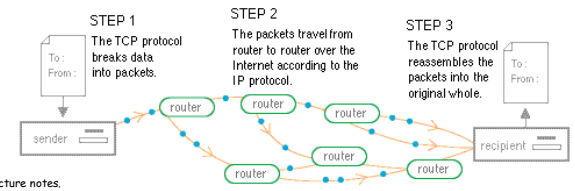
TCP/IP

Internet protocol. (IP)

- Rules for moving bits from A to B.
- Divide data into *packets* (header bits to say where to go, data bits) and transmit each individually, possibly along different paths.
- No guarantee packets arrive in order, or even arrive.

Transmission control protocol. (TCP)

- Rules to provide reliable communication between two computers.
- Packets arrive, and they arrive in order.
  - resend over IP until recipient acknowledges



Reference: COS 111 lecture notes.

Protocols

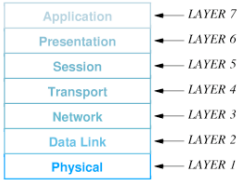
Many higher layer application protocols use TCP/IP.

- Used by http, smtp, telnet, ftp.

Ex 1: HyperText Transfer Protocol (HTTP).

- Set of rules for transferring files (text, graphics, video).
- Server `java.sun.com` waits for connection requests.
- Browser establishes connection with server at `http://java.sun.com`.
- Browser issues `GET` and `POST` commands.
- Server responds with header and body.

tells system to use http protocol to get the specified resource



ISO 7-layer model

SMTP

Ex 2: Simple Mail Transfer Protocol (SMTP).

- Set of rules for sending email.
- Server `smtp.princeton.edu` waits for connection requests on port `25`.
- User specifies sender, recipient, subject, message body, etc.

```
phoenix.Princeton.EDU% telnet smtp.princeton.edu 25
220 smtpserver1.Princeton.EDU ESMTP Sendmail 8.12.9/8.12.9;
Sun, 7 Dec 2003 09:18:47 -0500 (EST)
HELO localhost
250 smtpserver1.Princeton.EDU Hello phoenix.Princeton.EDU
[128.112.128.42], pleased to meet you
MAIL From: AlanTuring@cnn.com          <- Warning: do not spoof without receiver's consent
250 2.1.0 AlanTuring@cnn.com... Sender ok
RCPT TO: wayne@cs.princeton.edu
250 2.1.5 wayne@cs.princeton.edu... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Just wanted to say hi
Hope you're enjoying COS 126.
.
250 2.0.0 hB7EIlgn024881 Message accepted for delivery
quit
```

Open relay. smtp server accessible from anywhere.

Java Client: SMTP

A Java program that uses SMTP.

```
import java.net.Socket;
public class Mail {
   public static void main(String[] args) throws Exception {
      Socket socket = new Socket("smtp.princeton.edu", 25);   open socket
      In in    = new In(socket);                  mail server   port
      Out out = new Out(socket);

      out.println("HELO localhost");                SMTP protocol
      System.out.println(in.readLine());
      out.println("MAIL From: AlanTuring@cnn.com");          <- forged address
      System.out.println(in.readLine());
      out.println("RCPT To: wayne@cs.princeton.edu");        <- read from socket
      System.out.println(in.readLine());
      out.println("DATA");                                   <- write to socket
      out.println("Subject: Just wanted to say hi");
      out.println("Hope you're enjoying COS 126.");
      out.println(".");
      System.out.println(in.readLine());

      out.close();      close socket
      in.close();
      socket.close();
   }
}
```

Client-Server

Stream socket.

- ADT for two-way communication over TCP/IP.
  - read from socket input and write to socket output
- IP address: identifies computer.
- Port number: identifies application.
- Ex: IP address = `128.112.129.71`, Port = `25`.

  `128.112.129.71` → smtp.princeton.edu mail server; `25` → smtp email application
- Purpose of a Socket is to communicate with another Socket.

Client-server model.

- Client = creates socket to communicate with specified server.
- Server = creates one socket to listen for connection requests; creates another socket to communicate with each client.
- Ex: client = web browser, server = web server.

Echo Client

Echo client: connect with server, read text from standard input, send text to server, print whatever server sends back.

```
import java.net.Socket;
public class EchoClient {
   public static void main(String[] args) throws Exception {
      Socket socket = new Socket(args[0], 4444);   ← open socket
      In stdin = new In();
      In in    = new In(socket);   ← server name
      Out out  = new Out(socket);

      String s;
      while ((s = stdin.readLine()) != null) {   ← read from stdin
         out.println(s);   ← send to server
         System.out.println(in.readLine());   ← get from server
      }

      out.close();   ← close socket
      in.close();
      socket.close();
   }
}
```


Echo Server

Echo server: use `ServerSocket` to listen for connection requests; connect with a client; read text that client sends; and send it back.

```
public class EchoServer {
   public static void main(String[] args) throws Exception {
      ServerSocket serverSocket = new ServerSocket(4444);

      while (true) {
         Socket socket = serverSocket.accept();   ← listen and wait for client to request connection
         In   in  = new In (clientSocket);
         Out out = new Out(clientSocket);

         String s;
         while ((s = in.readLine()) != null)   ← read data from client and echo it back
            out.println(s);

         out.close();
         in.close();   ← close input streams and socket
         socket.close();
      }
   }
}
```

(open server socket and listen on port 4444)

Remote Control Server

Remote control server.

- Client sends text commands.
- Server launches: { TiVo recorder, coffee maker, burglar alarm }.
- Server acknowledges task completion.

Trivial modification to Java programs.

- Detail: must be running Java coffee maker on Internet.

Multi-Threaded Chatroom Application

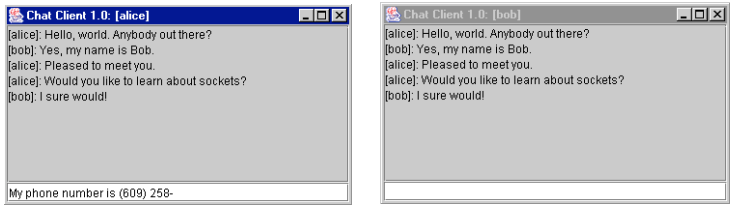
Chatroom client.

- Each clients requests a connection to server.
- Client sends message to server.
- Server broadcasts message to ALL connected clients.

Echo + a few details.

- Graphical user interface (event-based programming).
- Server must process several simultaneous connections (threads).

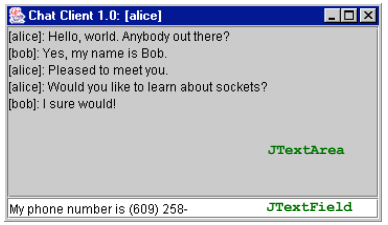
```
% java ChatClient alice bicycle.cs.princeton.edu
```



Graphical User Interface

Graphical user interface.

- Interact with computer using mouse and keyboard.
- Buttons, menus, scrollbars, toolbars, file choosers.
- Ex: Mac, Windows, KDE.



Event-based programming.

mouse click, keyboard entry

- Program remains in infinite loop, waiting for event.
- Upon event, program executes some code.

Callback.

- Programmer registers an event to listen for.
- Programmer writes code to process event.

Graphical User Interface

`enteredText`: Un-editable 10-by-32 scrollable text area.
`typedText`: User types one line of text and hits enter.
`actionPerformed`: Callback when user hits enter.

```
public class ChatClient extends JFrame implements ActionListener {
   ...
   private JTextArea enteredText = new JTextArea(10, 32);
   private JTextField typedText  = new JTextField(32);

   public ChatClient(String screenName, String hostName) {
      ...
      typedText.addActionListener(this);   <-- register event
      Container c = getContentPane();
      c.add(enteredText, BorderLayout.CENTER);
      c.add(typedText,   BorderLayout.SOUTH);
      pack();                                  ^
      show();                                  layout placement of typed text
   }

   public void actionPerformed(ActionEvent e) {        callback
      out.println(typedText.getText());
      typedText.setText("");
   }
```

listen for user entering input

Multi-Threaded Server

Threads of control.

- "Illusion" that several things are happening at once in a program.
  - similar idea used by OS to execute several programs at once
  - parallel computers make illusion a reality
- Timesharing: CPU processes each thread one for a fraction of a second before moving on to the next one.

Multi-threaded server.

- Server listens for connection requests in one thread.
- Server handles communication with each client in a separate thread.
- Makes server *scalable* – can accept requests, independent of speed in handling them.

Java has built-in support for threads.

Java Multi-Threaded Server: ChatServer

Chat server.

- The `listener` thread broadcasts messages to clients.
- Each `client` thread communicates with one client.

```
import java.net.Socket;
import java.net.ServerSocket;
import java.util.ArrayList;

public class ChatServer {
   public static void main(String[] args) throws Exception {
      ArrayList clients        = new ArrayList();
      ServerSocket serverSocket = new ServerSocket(4444);
      ClientListener listener   = new ClientListener(clients);
      listener.start(); ← start thread to broadcast message

      while (true) {
         Socket socket = serverSocket.accept(); ← wait for client
         Client client = new Client(socket);      connection request
         clients.add(client);
         client.start();
      }
   }
}
```

start separate thread to communicate with each client

Listener Thread: Broadcast Messages to Clients

Server constantly monitors messages sent from clients.

```
public class ClientListener extends Thread {
   private ArrayList clients;
   ...

   public void run() {   ← method invoked when thread is started
      while (true) {
         for (int i = 0; i < connections.size(); i++) {
            Client ith = (Client) clients.get(i);
            if (!ith.isAlive()) clients.remove(i);
            String message = ith.getMessage();     check client i

            if (message != null) {
               for (int j = 0; j < clients.size(); j++) {
                  Client jth = (Client) clients.get(j);
                  jth.println(message);
               }
            }                                      broadcast to everyone
         }
         sleep(100);
      }
   }
}
```

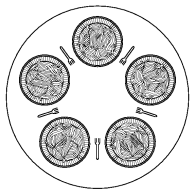

Synchronization

Gotcha: two threads simultaneously accessing the same object.

- i^{th} `connection` thread puts next message into client i's `buffer`.
- `listener` thread gets next message from client i's `buffer`.

Dining philosophers problem.

- To eat, philosopher needs both adjacent chopsticks.
- If each philosopher grabs chopstick to right, then waits for chopstick to left, then everyone starves.
- Deadlocking = bane of programming with threads.



Java has built-in support for synchronization.

- Indicate blocks of code that can't be simultaneously accessed.
- No need unless you are using more than one thread.

Synchronization in Java

Server maintains a separate thread for each client connection.

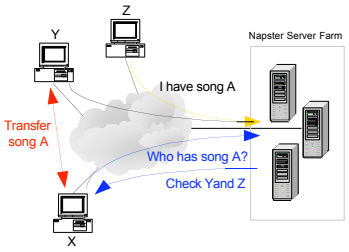
```
public class Client extends Thread {
   private String buffer;   ← message received from client
   ...
   public Client(Socket socket) { ... }

   public void run() {    only this thread gets held up if no incoming message
      String s;
      while ((s = in.readLine()) != null) { setMessage(s); }
   }

   public synchronized String getMessage() {
      if (buffer == null) return null;
      String temp = buffer;
      buffer = null;
      notifyAll();   ← tell setMessage to stop waiting
      return temp;
   }                                              executes in one thread

   public synchronized void setMessage(String s) {
      if (buffer != null)
         wait();
      buffer = s;
   }                                              executes in another thread
}
```

if current message not yet broadcast, then wait to receive next message

Peer-to-Peer Networking

Peer-to-peer networking.
- All clients communicate and share resources as equals.
- Ex: Napster, Kazaa, ICQ, Gnutella, Morpheus, Limewire, Gnucleus, Skype, Jabber, eMule, BitTorrent, SoulSeek, . . .

Centralized (e.g., original Napster)
- Central server keeps index of all files and where they are.
- User queries server to find file, the connects directly with host.
- No central bottleneck.
- Scalable.

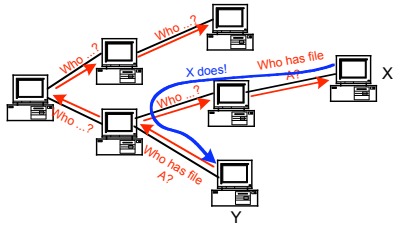


Reference: http://digital5.ece.tntech.edu/hexb/690f03/lectures.htm

Peer-to-Peer Networking

Peer-to-peer networking.
- All clients communicate and share resources as equals.
- Ex: Napster, Kazaa, ICQ, Gnutella, Morpheus, Limewire, Gnucleus, Skype, Jabber, eMule, BitTorrent, SoulSeek, . . .

Decentralized (e.g., Gnutella).
- User queries neighbors to find file, neighbors query their neighbors.
- No central point of control or failure.
- Massively scalable.



Reference: http://digital5.ece.tntech.edu/hexb/690f03/lectures.htm

Summary

Circuit switching vs. packet switching.
- Circuit switching for real-time communication.
- Packet switching is cheaper per bit.
- Never underestimate the bandwidth of a 747 filled with DVDs!

Client-server vs peer-to-peer.
- Client-server for browsing web.
- Peer-to-peer for file-sharing.

Where to learn more? COS 318, COS 461.