Networks and Database Systems

Professor Tom Ellman
Lecture 2
Java TCP Socket Programming

• Host names, IP addresses and DNS.
• Examples: **IPNameFinder**, **MyLocalIPAddress**.
• Ports: Reserved and Unreserved.
• Example: **PortScanner**.
• Client and Server Socket Life Cycles.
• Examples:
  – **TCPEchoClient**, **TCPEchoServer**.
  – **DaytimeServer**, **GetRemoteTime**.
• Network Interfaces: Example: **InterfaceLister**.
• Socket Options: Example: **SocketInfo**.
IPNameFinder

• Given a host name, it returns an IP address.
• Given an IP address, it returns a host name.
• Uses Java InetAddress class.
• Relies on Domain Name System (DNS).
NetBeans

- Making a Java application project.
- Passing parameters to a Java application.
- Running a Java application.
InetAddress Methods

String host = "vnc.cs.vassar.edu";
InetAddress address = InetAddress.getByName(host);

String host = "143.229.6.33";
InetAddress address = InetAddress.getByName(host);

PrintStream sout = System.out;
sout.println("Host name:" + address.getHostName());
sout.println("IP address:" + address.getHostAddress());

Static InetAddress method **getByName** takes a string representing either a host name or an IP address and returns an InetAddress instance. Instance methods **getHostName** and **getHostAddress** return name and IP address respectively.
Enter host name or address: vnc.cs.vassar.edu
Host name: vnc.cs.vassar.edu
IP address: 143.229.6.34

Enter host name or address: 143.229.6.34
Host name: mr34.cs.vassar.edu
IP address: 143.229.6.34

Enter host name or address: mr34.cs.vassar.edu
Host name: mr34.cs.vassar.edu
IP address: 143.229.6.34

A host may have multiple names. One of these is the *canonical* name.
A host may have multiple IP addresses. This feature is used by sites that run multiple servers to handle large traffic volume.
Addresses in IP Version 4

• An IP address is composed of 4 bytes (32 bits).
• Each byte encodes an integer: 0-255.
• Normally written in dot notation:
  • E.g.: 143.229.6.34
• Can encode $2^{32} = 4,294,967,296$ addresses.
• Almost one for every living person.
• Not enough!
Addresses in IP Version 6

- An IP address is composed of 16 bytes (128 bits).
- Each byte encodes an integer: 0-255.
- Normally written in hexadecimal notation:
  - E.g.: \texttt{2001:0db8:85a3:0000:0000:8a2e:0370:7334}
- Can encode $2^{128} \approx 3.4028e+38$ addresses.
- Should be enough, for a while.
DNS: Domain Name System

- A distributed algorithm run on a collection of Domain Name Servers.
- Maintained by ICAN: Internet corporation for assigned names.
- Applet simulation of DNS algorithm:
Finding the Local Machine’s IP Address

InetAddress address = InetAddress.getLocalHost();
System.out.println(address);

Linguini/192.168.10.121
Finding the Local Machine’s IP Address on a Windows Machine

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Tom Ellman>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

  Connection-specific DNS Suffix . :
  IP Address. . . . . . . . . . . . . . . . . . . . . . . : 192.168.10.121
  Subnet Mask . . . . . . . . . . . . . . . . . . . . : 255.255.255.0
  Default Gateway . . . . . . . . . . . . . . . . : 192.168.10.1
Finding the Local Machine’s IP Address on a Linux Machine:

```
ellman@mote33:~$ ifconfig

eth0 Link encap:Ethernet  HWaddr 00:14:5e:19:97:86
   inet addr:143.229.6.33
   ...
Local Loopback Address

• Some IP addresses are reserved for special purposes.
• In IP version 4 a special address is reserved as the local loopback address: \textbf{127.0.0.1}
• IP version 4 will treat this as a reference to the local machine.
Port

• Logical (not physical) concept.
• Number in the range: $0 \ldots 2^{16} = 65535$.
• Means of identifying one of many possible servers running on a given host.
• Port ranges:
  0 ... 1023: Reserved (Well Known Ports).
  1024 ... 49151: Reserved for Registration w/ IANA).
  49152 ... 65535: Dynamic, Private Ports.
Some Well-Known Ports

- Echo 7
- Daytime 13
- FTP (Data) 20
- FTP 21
- Telnet 23
- SMTP 25
- HTTP 80
- NNTP 119
PortScanner

- User enters a host name.
- Program finds the host’s IP address.
- Program successively tries to open connections to ports 0 … 24.
- Program reports successful connections.
- (Not considered a friendly program.)
Socket and ServerSocket

• Java classes.
• Logical (not physical) concept.
• Implements a connection between two hosts.
• Provides input and output streams to both client and server.
• Provides reliable transmission of byte data.
• Relies on TCP / IP to implement the streams.
ServerSocket Life Cycle

1. Create a ServerSocket associated with a port.

2. Repeat:
   a. Wait for a connection request from a client.
   b. Obtain a Socket implementing the connection.
   c. Get InputStream and OutputStream from the Socket.
   d. Use the streams to communicate with the client.
   e. Close the connection.
Client Socket Life Cycle

1. Create a **Socket** associated with a host and a port.
2. Get and **InputStream** and an **OutputStream** from the **Socket**.
3. Use the streams to communicate with the server.
4. Close the connection.
Socket Provides Streams Connecting Client and Server

Client Socket

Input Stream

Output Stream

Server’s Socket

Input Stream

Output Stream
TCPEchoClient & TCPEchoServer

- Client sends messages to server, one line at a time.
- Server echoes message back to client, with a little extra text.
- Illustrates client and server socket life cycles.
Experiments with TCPEcho Programs

• Try running client without running server.

• Get a client and server exchanging messages, and then start a second client.

• What can we learn from these experiments?
Problem with TCPEchoServer

- Server thread blocks when waiting for a connection.
- The server thread cannot do any other work while it is blocked.
- While communicating with a client, the server is not processing new connections.
- Only one client can be connected at a time.

Solutions:
- Multi-threaded server.
- Non-blocking IO.
DaytimeServer & GetRemoteTime

- Server program waits for connection on well-known daytime port.
- Client program inputs host name, finds IP address and opens a socket to server.
- Server sends a time/date object to client.
- Client displays time and date.
SocketInfo

• Demonstrates **Socket** methods that provide information about the connection.

• Remote Host Information:
  – get InetAddress
  – get Port

• Local Host Information:
  – get Local Address
  – get Local Port
• Some machines have multiple physical interfaces to a network.
• E.g., ethernet and wireless.
• Each interface can be represented by a **NetworkInterface** object.
• **NetworkInterface** class has static method to enumerate all network interfaces.
• **Socket** and **ServerSocket** constructors accept additional parameters to indicate desired network interface.
Chapter 2
Application Layer

A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2009
J.F Kurose and K.W. Ross, All Rights Reserved 2: Application Layer

Jim Kurose, Keith Ross
Addison-Wesley, April 2009.
DNS: Domain Name System

People: many identifiers:
   – SSN, name, passport #

Internet hosts, routers:
   – IP address (32 bit) - used for addressing datagrams
   – “name”, e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:
• *distributed database* implemented in hierarchy of many *name servers*
• *application-layer protocol* host, routers, name servers to communicate to resolve names (address/name translation)
  – note: core Internet function, implemented as application-layer protocol
  – complexity at network’s “edge”
DNS

DNS services

- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn’t scale!
Distributed, Hierarchical Database

Client wants IP for www.amazon.com; 1st approx:

- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com
DNS: Root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name servers worldwide
TLD and Authoritative Servers

• Top-level domain (TLD) servers:
  – responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  – Network Solutions maintains servers for com TLD
  – Educause for edu TLD

• Authoritative DNS servers:
  – organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web, mail).
  – can be maintained by organization or service provider
Local Name Server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy
DNS name resolution example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

**iterated query:**
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
DNS name resolution example

recursive query:
- puts burden of name resolution on contacted name server
- heavy load?

```
2: Application Layer
```
DNS: caching and updating records

• once (any) name server learns mapping, it *caches* mapping
  – cache entries timeout (disappear) after some time
  – TLD servers typically cached in local name servers
    • Thus root name servers not often visited

• update/notify mechanisms under design by IETF
  – RFC 2136
Socket programming

**Goal:** learn how to build client/server application that communicate using sockets

**Socket API**

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

a *host-local, application-created, OS-controlled* interface (a “door”) into which application process can both send and receive messages to/from another application process
Socket-programming using TCP

**Socket**: a door between application process and end-end-transport protocol (UCP or TCP)

**TCP service**: reliable transfer of **bytes** from one process to another
Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client’s contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When **client creates socket:** client TCP establishes connection to server TCP

- When contacted by client, **server TCP creates new socket** for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

**application viewpoint**

*TCP provides reliable, in-order transfer of bytes (“pipe”) between client and server*
Client/server socket interaction: TCP

**Server** (running on `hostid`)

- create socket, port=`x`, for incoming request:
  
  ```java
  welcomeSocket = ServerSocket()
  ```

- wait for incoming connection request:
  
  ```java
  connectionSocket = welcomeSocket.accept()
  ```

- read request from `connectionSocket`

- write reply to `connectionSocket`

- close `connectionSocket`

**Client**

- create socket, connect to `hostid`, port=`x`
  
  ```java
  clientSocket = Socket()
  ```

- send request using `clientSocket`

- read reply from `clientSocket`

- close `clientSocket`
Stream jargon

- A **stream** is a sequence of characters that flow into or out of a process.
- An **input stream** is attached to some input source for the process, e.g., keyboard or socket.
- An **output stream** is attached to an output source, e.g., monitor or socket.

2: Application Layer
Socket programming with TCP

Example client-server app:

1) client reads line from standard input (\texttt{inFromUser} stream),
   sends to server via socket (\texttt{outToServer} stream)

2) server reads line from socket

3) server converts line to uppercase,
   sends back to client

4) client reads, prints modified line from socket (\texttt{inFromServer} stream)
Example: Java client (TCP)

```java
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        String sentence;
        String modifiedSentence;
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```
Example: Java client (TCP), cont.

```java
BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
```

Create input stream attached to socket

Send line to server

Read line from server
Example: Java server (TCP)

```java
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));

            String clientSentence;  // Create welcoming socket at port 6789
            String capitalizedSentence;  // Wait, on welcoming socket for contact by client

            ServerSocket welcomeSocket = new ServerSocket(6789);  // Create input stream, attached to socket

            System.out.println(inFromClient.readLine());

            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));

            // Read the client's message
            String clientSentence = inFromClient.readLine();

            // Capitalize the message
            String capitalizedSentence = clientSentence.toUpperCase()

            // Print the capitalized message
            System.out.println(capitalizedSentence);
        }
    }
}
```
Example: Java server (TCP), cont

```java
DataOutputStream outToClient =
    new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();
capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);
```