Data Communication & Networks  
G22.2262-001  
Session 11 - Main Theme  
Java Sockets  
Dr. Jean-Claude Franchitti  

New York University  
Computer Science Department  
Courant Institute of Mathematical Sciences  

Agenda  

- Internet Transport-Layer Protocols  
- Multiplexing / Demultiplexing  
- Socket Programming
Part I

*Internet Transport-Layer Protocols*

- Reliable, in-order delivery TCP
  - congestion control
  - flow control
  - connection setup
- Unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- Services not available:
  - delay guarantees
  - bandwidth guarantees
Part II

Multiplexing / Demultiplexing

Multiplexing/Demultiplexing

Demultiplexing at rcv host:
delivering received segments to correct socket

Multiplexing at send host:
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

= socket  = process

```
application P3
  transport
  network
  link
  physical

P1
  transport
  network
  link
  physical

P2
  transport
  network
  link
  physical

P4
  transport
  network
  link
  physical
```

host 1  host 2  host 3
How Demultiplexing Works

- Host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number (recall: well-known port numbers for specific applications)
- Host uses IP addresses & port numbers to direct segment to appropriate socket

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>other header fields</td>
<td></td>
</tr>
<tr>
<td>application data (message)</td>
<td></td>
</tr>
</tbody>
</table>

TCP/UDP segment format

Connectionless Demultiplexing

- Create sockets with port numbers:
  ```java
  DatagramSocket mySocket1 = new DatagramSocket(99111);
  DatagramSocket mySocket2 = new DatagramSocket(99222);
  ```
- UDP socket identified by two-tuple:
  ```java
  (dest IP address, dest port number)
  ```
- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket
Connectionless Demux (cont.)

DatagramSocket serverSocket = new DatagramSocket(6428);

Connection-Oriented Demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request
Connection-Oriented Demux (cont.)

Part III

Socket Programming
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 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)

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)
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)

Client/Server Socket Interaction: TCP

\begin{itemize}
    \item \textbf{Server} (running on hostid)
        \begin{itemize}
            \item create socket, port=x, for incoming request:
            \item welcomeSocket = ServerSocket()
            \item wait for incoming connection request
            \item connectionSocket = welcomeSocket.accept()
            \item read request from connectionSocket
            \item write reply to connectionSocket
            \item close connectionSocket
        \end{itemize}

    \item \textbf{Client}
        \begin{itemize}
            \item create socket, connect to hostid, port=x
            \item clientSocket = Socket()
            \item send request using clientSocket
            \item read reply from clientSocket
            \item close clientSocket
        \end{itemize}
\end{itemize}
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 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();
    }
}

Example: Java Client (TCP), cont.

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();
}
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()));

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

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

            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Example: Java Server (TCP), cont.

```java
Create welcoming socket at port 6789
Wait, on welcoming socket for contact by client
Create input stream, attached to socket

End of while loop, loop back and wait for another client connection
```
Socket Programming With UDP

UDP: no “connection” between client and server
• no handshaking
• sender explicitly attaches IP address and port of destination to each packet
• server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

Client/Server Socket Interaction: UDP

Server (running on hostid)
- create socket, port=x, for incoming request:
  serverSocket = DatagramSocket()
- read request from serverSocket
- write reply to serverSocket specifying client host address, port number

Client
- create socket, clientSocket = DatagramSocket()
- Create, address (hostid, port=x), send datagram request using clientSocket
- read reply from clientSocket
- close clientSocket
Example: Java Client (UDP)

**Diagram:**

- **Client process**
  - Input: receives packet (TCP received "byte stream")
  - Output: sends packet (TCP sent "byte stream")

**Java Code:**

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

class UDPClient {
    public static void main(String args[])
        throws Exception {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        sendData = sentence.getBytes();

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
    }
}
```

**Process:**

1. **Create input stream**
2. **Create client socket**
3. **Translate hostname to IP address using DNS**
4. **Send packet**
5. **Receive packet**
Example: Java Client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

Send datagram to server

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

Read datagram from server

clientSocket.receive(receivePacket);

String modifiedSentence = new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);

clientSocket.close();

Example: Java Server (UDP)

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

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
        }
    }
}
Example: Java Server (UDP), cont.

```java
String sentence = new String(receivePacket.getData());
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();
DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, port);
serverSocket.send(sendPacket);
```

Get IP addr
port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

Part IV

Conclusion
Assignment & Readings

- Assignment #4 (due 04/24/08)
  - Assigned at the completion of Session 9
- Readings
  - Java.Net Package Documentation on Sun’s Java Web site
  - http://java.sun.com/docs/books/tutorial/networking/sockets/

Next Session:
IP Multicast