

CSCD 330

Network Programming

Winter 2016

Lecture 7

Application Layer – Socket Programming in Java

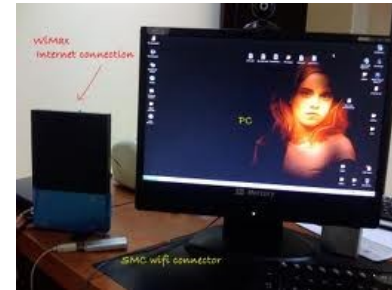
Reading: Chapter 2, Java links Relevant Links
page

Some Material in these slides from J.F Kurose and K.W. Ross
All material copyright 1996-2007



Review Client/Server Programming

Firefox
HTTP 80



- So far,
 - Host has IP Address
146.187.134.22
 - Network Layer identifier
 - Every network “device” has this identifier
 - Phone, Toaster, etc.
 - Processes running on Hosts
 - Assigned a port number
 - Port numbers identifiers for processes
 - Some port numbers reserved 1 - 1023
 - Other port numbers reserved for widely recognized processes

Review Client/Server Programming

- **Communication Between Client/Server**
 - Uses object, “Socket”
 - Socket is the API between a program and the TCP/IP stack of the OS
 - It has an input stream and an output stream built in to it
 - Both the client and server define different ends of this socket
 - Link to the Java .net Package API

<http://download.java.net/jdk7/archive/b123/docs/api/java/net/package-summary.html>

TCP/IP Client/Server



- How this works in Java
 - Server Socket
 1. Binds socket to specific port number
 2. Listens for incoming connections on that port
 3. When connection attempted, it accepts connection, creates a regular socket for communication to client
 4. Port number is different and selected by stack software

TCP/IP Client/Server

- Java Code for Server

```
ss = new ServerSocket (port);
```

```
// Loop forever
```

```
While (true) {
```

```
    // Get a connection
```

```
    Socket newSocket = ss.accept ();
```

```
    // Deal with the connection
```

```
    // ....
```

```
}
```

Server socket listens on a port

Inside loop waits for connection

Creates a new socket object representing new connection

What is not obvious is that the new connection is through a different port number

TCP/IP Client/Server

- **Java code for Client, Send/Receive Data**

```
// Create a socket for communicating with server
```

```
Socket clientSocket = new Socket ("hostname", 6789);
```

↑
Create a client TCP socket, with host and port

```
// Create data streams for communicating through the socket
```

```
BufferedReader in = new BufferedReader
```

```
(new InputStreamReader (clientSocket.getInputStream ());
```

```
PrintWriter out = new PrintWriter (clientSocket.getOutputStream ());
```

```
System.out.println (in.readLine ()); // Print output to screen
```

↑
Then, create streams to send input and get it back from server



UDP

UDP Socket Programming

UDP no real “connection” between client and server

- No handshaking
- Sender attaches IP address and destination port to each packet
- Server must extract IP address and port of sender from received packet, so answer can be sent back!

UDP, transmitted data may be

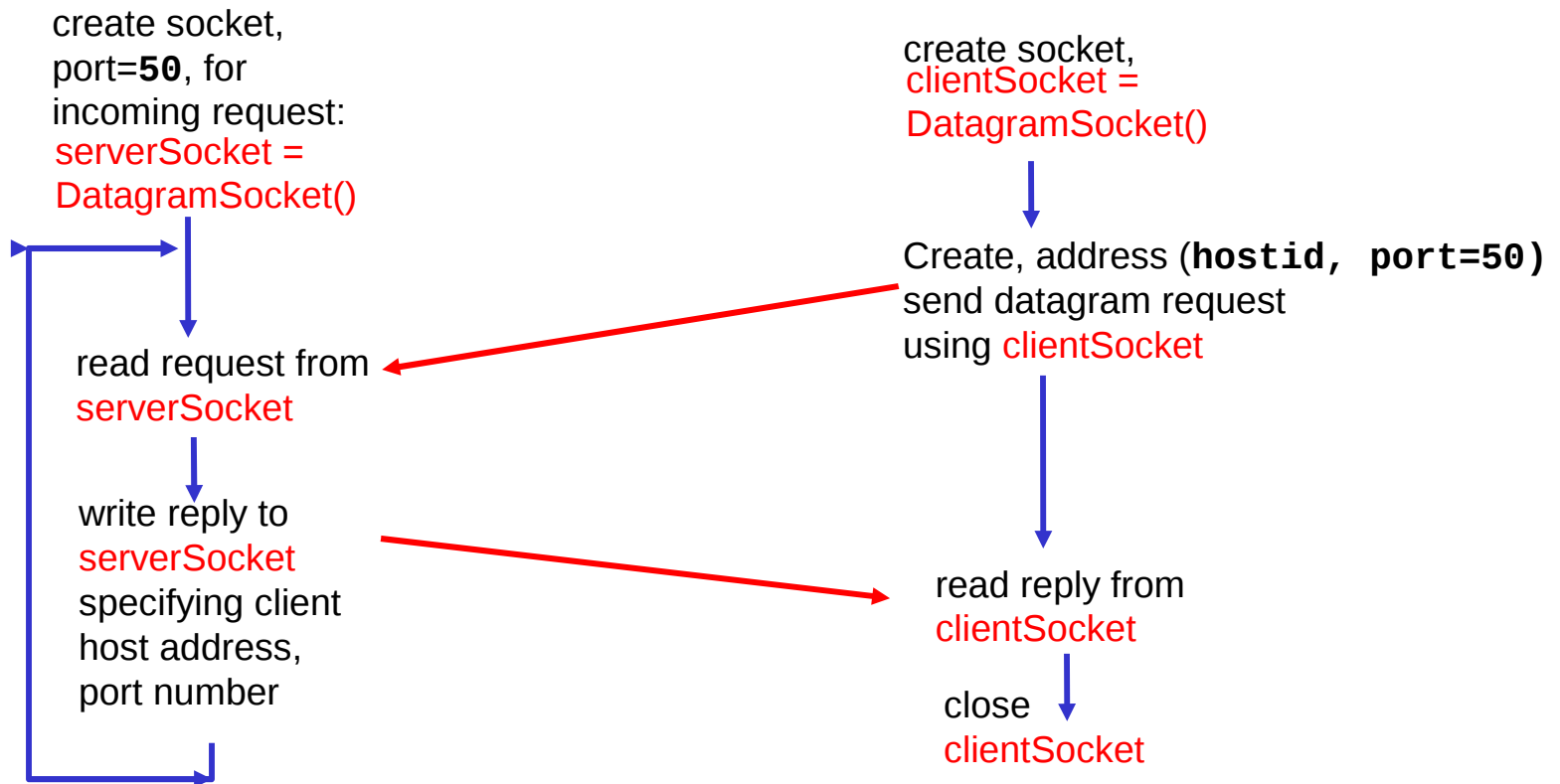
- Received out of order, or
- Lost



Client/server socket interaction: UDP

Server (running on `hostid`)

Client



UDP Summary

Client



Server

- No connection setup – no “pipe”
- Main Differences, TCP:
 1. Each batch of bytes sent with attached address information
 2. No special ServerSocket class in java

UDP Summary



- **Create a Packet**
 - Push it out into the network through a socket
 - Server accepts the packet addressed to him
- **Mail is a lot like UDP**
 - Each letter needs the address of the destination
 - Independent letters sent to the same address

Java Classes for UDP

- Datagrams for connectionless protocol
- Two classes implement datagrams in Java:
 - `java.net.DatagramPacket`
 - `java.net.DatagramSocket`
- `DatagramPacket` is **actual packet of information**, an array of bytes, that is transmitted over the network.
- `DatagramSocket` is **socket** that sends and receives `DatagramPackets` across the network.
- Think of `DatagramPacket` as a **letter** and `DatagramSocket` as the **mailbox** that the mail carrier uses to pick up and drop off your letters
- Need both classes for UDP sockets



Java Classes for UDP

- `DatagramPacket` class provides programmer with two constructors.

- First is for `DatagramPackets`
that receive data

Constructor needs

Array to store the data

Amount of data to receive

- Second is for `DatagramPackets`
that send data

Constructor needs

Array to store the data

Amount of data to send

Plus **destination address** and **port number**



Java Classes for UDP



- **DatagramSocket** represents connectionless socket
It provides two constructors,
 1. Programmer can specify a port OR
 2. Allow system to randomly use a port
- **Methods**
 - Two most important methods, **send()** and **receive()**
 - Each takes an argument of a constructed **DatagramPacket**
- **send()** method
 - Data in packet is sent to specified host and port
- **receive()** method
 - Will block execution until a packet is received by underlying socket, then data copied into packet provided

Sentence Capitalizer (Again)

Example: Java client (UDP)

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

```
class UDPClient {
    public static void main(String args[]) throws Exception
    {
```

Create input stream

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

Keyboard

Create client socket

```
        DatagramSocket clientSocket = new DatagramSocket();
```

Translate hostname to IP address using DNS

```
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
```

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

Read from keyboard

Example: Java client (UDP), cont.

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

Send datagram
to server

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

```
clientSocket.send(sendPacket);
```

Read datagram
from server

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

```
clientSocket.receive(receivePacket);
```

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

```
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}
```

```
}
```


Sentence Capitalizer (Again)

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);
```

Create datagram socket at port 9876 →

```
        byte[] receiveData = new byte[1024];  
        byte[] sendData = new byte[1024];
```

Create space for received datagram] →

```
        while(true)  
        {
```

```
            DatagramPacket receivePacket =  
                new DatagramPacket(receiveData, receiveData.length);  
            serverSocket.receive(receivePacket);
```

Receive datagram] →

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());
```

Get IP addr
port #, of
sender

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

Create datagram
to send to client

```
sendData = capitalizedSentence.getBytes();
```

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

Write out
datagram
to socket

```
serverSocket.send(sendPacket);
```

```
} ← End of while loop,  
loop back and wait for  
another datagram
```

```
}  
}
```

Summary

- Brief coverage of Java sockets - TCP/UDP
- Should be enough to get started
 - Examples available as links on the main class page
- Will cover threads next
- Also, practice client-server in the lab
- Read references in RelatedLinks for more information

*Understanding & Mastering
Threaded Programming*

2nd Edition
Covers Java 7



Java
Threads

O'REILLY*

Scott Oaks & Henry Wong

