Lecture 6

Network Programming - Sockets & RMI
What is a socket

• A layer of code that runs on top of a lower level network protocol like IP
• TCP, UDP and Multicast Sockets
• A connection between two parties via a layer on top of IP
Internet/WWW Protocol Layers

Client & Server Applications communicate via the protocols

SNMP  DNS  NFS  FTP  TelNET  POP  SMTP  BEEP  HTTP  LDAP

UDP (Unreliable)

TCP (Reliable)

IP

Ethernet 802  SLIP/PPP

Cat 1-5  FDDI

Frame Relay  ATM  ISDN  ADSL  X.25
IP

• Internetworking Protocol – 1982?
• Runs on top of Ethernet (IEEE802) protocol and others (ATM, FDDI, X.25)
• Provides sending packets between addresses
• “Fire and forget” protocol
• IPv4 addresses are 4 numbers separated by dots
  – 10.255.0.128 (all numbers are between 0-255 or 0x00 - 0xFF hexadecimal or a 32 bit binary number. It takes 4 bytes to hold the address.)
  – Means $2^{32} = 4,294,967,296 - 1 \approx 4.29 \times 10^9$ possible addresses!
  – Like “phone numbers” for internet hosts
IP

• Connectionless (like sending a letter not like a phone call)
• Addressing and route selection
• Packet assembly and disassembly
• Header
  – Version
  – Source Address – 4 bytes
  – Destination Address – 4 bytes
  – Checksum
  – TTL (Time To Live) – number of router hops allowed
  – Total Length
  – Payload – between 1 – 64K bytes long
IP Addresses

• Static – Never changes unless you change it in your OS setup. Assigned to the MAC address (Unique ID embedded in the NIC) of your NIC (Network Interface Card) on a machine.

• Dynamic – assigned by a piece of software on the LAN that maintains a pool of addresses and hands them out as needed. They expire and you must get a new one (automatic). Using DHCP (Dynamic Host Configuration Protocol) your machine gets new IP Addresses. Efficient and flexible on a network with machines moving around or attaching and unattaching.
TCP

• Transmission Control Protocol
• Connection oriented
• Guaranteed packet delivery – error free, in sequence sent, no dup,
• Runs on top of IP
• Slower than UDP
• Each IP Address also can have up to 65536 TCP Ports
  – Like phone extensions on a phone number.
TCP Socket

- IP Address + TCP Port Numbers of Client and Server
- Always PTP – Point to Point – like a phone call
- Handshaking between Client TCP Port and Server machine. Establishes a socket connection.
- Packets sent get acknowledged by the receiver.
- TCP handles congestion control
- Sockets system dependant implementation
- BSD Sockets machine independent
TCP Packet

- Source TCP Port – 2 bytes
  – Address is in the IP Header
- Target TCP Port - 2 bytes
  – Address is in the IP Header
- Sequence Number – 4 bytes = 0 – (2^{32} - 1)
- Acknowledgement Number
- Window – size of the sender's buffer
- Checksum – 2 bytes
- Control Bits
- Packet Size is in the IP header
- Payload
Well Known TCP Ports (0-1024)

- 80 HTTP
- 443 SSL
- 20 & 21 FTP (20 control, 21 data)
- 25 SMTP
- 110 POP3
- 23 TelNet
- 161 SNMP
- 7 Echo
- 53 DNS
- others
Socket communication Using TCP

Initiator Socket App

Initiator Socket App

Acceptor Socket App

TCP (Reliable)

IP

Ethernet 802

Cat 1-5
If you don’t have a network card you can have a permanent IP address on your machine.
adding a loopback adapter

Add/Remove Hardware Wizard

New Hardware Detection
The wizard automatically locates new Plug and Play hardware.

Windows is searching for new Plug and Play hardware to install.
Searching...
add a new device
select ‘no’
select other devices
microsoft & loopback adapter
click next
finished installing the loopback adapter
you should have a new entry in your networking and dialup
properties
properties for TCP/IP
set it to a convenient address like 100.100.100.100
Getting the IP address of a machine

- on windows create a DOS/command prompt
- run ipconfig.exe or winipconfig.exe
- should be on you machine somewhere you may nee to search for it
What is localhost?

• Is a short cut that says using this machine’s default IP address
• You could use 127.0.0.1 which is the loopback address for any machine also but it means you can’t talk to any other boxes
TCP demo

- TCPClientGUI
- TCPEchoTextServerDemo
TCP Client Application

Server URL: 10.0.1.300
Port: 8081

Connect

Text To Send

<<An Exception was thrown on the client>> : java.net.UnknownHostException: 10.0.1.300

SendText

Text Returned From Server
<<An Exception was thrown on the client>> : java.net.ConnectException: Connection refused: no further information
run the **TCPEchoTextServerDemo**

- Output:
  
  <<socket factory created>>
  <<server host address = 0.0.0.0>>
  <<server host port = 8081>>
  <<server SO timeout = 0>>
  <<socket factory returned a new connection :
    connection number 1>>
Sending a string to the server

- Server output:
  [1] String Received = This is a string send by my TCPClientGUI application

  ECHOED BY SERVER: This is a string send by my TCPClientGUI application
TCPClientGUI

- Creates a stream (TCP) **Socket**
- The constructor creates it and attempts to connect with the server on the specified port and IP address.
- Gets a **PrintWriter** to send text strings instead of just stream of bytes
- Gets a **BufferedReader** to get back strings instead of just bytes.
TCPEchoTextServerDemo

- Creates a `ServerSocket` (a factory that returns a new socket for every client that connects)
- create a thread to run the processing for that socket on
- start the thread
- listen for more client connections
TCPEchoTextServerDemo

- Gets a PrintWriter to send text strings instead of just stream of bytes
- Gets a BufferedReader to get back strings instead of just bytes.
- Sends a message to the client saying “Hello to my client”
- Loops waiting for a string from the reader and then just appends some text and returns it through the writer.
Three clients are talking to a single multithreaded server application
output from the server

<<socket factory created>>
<<server host address = 0.0.0.0>>
<<server host port = 8081>>
<<server SO timeout = 0>>
<<socket factory returned a new connection : connection number 1>>
<<socket factory returned a new connection : connection number 2>>
<<socket factory returned a new connection : connection number 3>>

[3] String Received = This is text from client three
[2] String Received = This is from client number Two the best client, I'm told
Multithreaded servers

- Efficient to a certain extent but …
- If the clients aren’t talking much it wastes resources
- So, maybe you should limit the number of threads you create and just reuse them
- That’s thread pooling
- And maybe your clients should connect, do their work, disconnect, then later reconnect, so some work, … that’s HTTP’s strategy for scaling.
TCP sockets are streams of bytes

• You can send almost any type of object not just bytes
• Primitives int, long, ...
• String, and other java objects
• HTTP uses TCP sockets
TCP parameters

- **SO_TIMEOUT** – amount of time to block waiting, if exceeded throws a `InterruptedException`
- **TCP_NODELAY** – sends packets immediately no matter how small
- **SO_LINGER** – should it wait after a close to try and send unsent data?
- **TCP_KEEPALIVE** – detects lost clients by sending a packet every two hours if no packets sent by client. Not part of TCP.
- `shutdownOutput()` - signals the other end that no more data will be sent
- `shutdownInput()` - signals the other end its not accepting any new data
Initiator & Acceptor

• The code that creates a socket is the initiator of the socket
• The code that waits an accept() ‘s the socket is the acceptor
• Often incorrectly termed client and server
• Both side of the socket connection can send packets without the other sending one first, unlike the client-server paradigm
mySocket = new Socket(serverURLText.getText(),
    Integer.parseInt(serverTCPPortText.getText()));
myInputStream = mySocket.getInputStream();
myOutputStream = mySocket.getOutputStream();
myBufferedReader = new BufferedReader(new
    InputStreamReader(myInputStream));
myPrintWriter = new PrintWriter(myOutputStream,true);

Thread.currentThread().sleep(1000); //pause to let the server
    //socket get running
if (myBufferedReader.ready()) //see if any message from the
    //server to begin with
    textReturnedTextarea.setText(myBufferedReader.readLine());
public static void main(String[] args)
{
    ServerSocket mySocketFactory = null;
    int numberOfActiveConnections = 0;

    try
    {
        if (args.length == 0) // no commandline args
        {
            mySocketFactory = new ServerSocket(8081);
        }
        else
        {
            mySocketFactory = new ServerSocket(Integer.parseInt(args[0]));
        }
    }
}
Acceptor code (2)

TCPEchoTextServerDemo CurrentTCPEchoTextServerDemoInstance;
Socket socketReturnedFromFactory = mySocketFactory.accept();
numberOfActiveConnections++;

System.out.println("<<socket factory returned a new connection :
    connection number " +
    Integer.toString(numberOfActiveConnections) + ">>");

currentTCPEchoTextServerDemoInstance =
    new TCPEchoTextServerDemo( socketReturnedFromFactory,
                                numberOfActiveConnections );

    //this gets a socket to listen on
    //now start a new thread for that socket's processing
Acceptor code (3)

public TCPEchoTextServerDemo(Socket socketToRunOnThread,
                              int connectionID)
        throws Exception
{
    myConnectionID=connectionID;
    mySocket = socketToRunOnThread;
    myThreadIRunOn = new Thread( this );
    myThreadIRunOn.start();
}

• Constructor creates a new thread to run itself on
• Gets a socket from the instantiator ( the main() )
UDP

• User Datagram Protocol
• Not connection oriented
• “Fire and forget” IP
• No guarantee – lost packets, dups, and out of sequence ok
• Faster than TCP – lower overhead
UDP Packet

• Inside the payload of an IP packet
• Source & Destination Port
  – Addresses are in the IP Header
• Checksum – 2 bytes
• Message Length is in IP Header
Socket communication Using UDP

Sender Socket App

Listener Socket App

SenderListener Socket App

UDP (Fire and Forget)

IP

Ethernet 802

Cat 1-5
Sending and listening on the same addresses and port

- This is a single GUI talking to itself
two instances talking on the same machine (two ports)
A simple UDP chat application
Using same port but different IP addresses

- Sent from IP 100.100.100.101, this is two ips on the same port.

- Sent back from 169.254.25.129 to socket on IP 100.100.100.101 port 8082.

- Sent from IP 100.100.100.101, this is two ips on the same port.
Datagram Socket

• No initiator and acceptor
• You get a socket and can send a datagram packet to an IP and Port
• If nobody’s listening its ok! It gets thrown away.
• You can just listen and not send, too!
Sending without listening
RMI

• Remote Method Invocation
• The ability to invoke a method on an object that may be in a different process, JVM or machine as though it was a local object
Using RMI

- Create a interface that implements the methods to be remoted and implement the `Remote` interface
  - `public interface DemoRMIServerRemote extends Remote`

- Create class to be remotely invoked that implements the interface
  - `public class DemoRMIServerImpl extends java.rmi.server.UnicastRemoteObject implements DemoRMIServerRemote`

- Run the `rmic` compiler on the class it generates stubs and skeletons
  - `DemoRMIServerImpl_Stub.class`
  - `DemoRMIServerImpl_Skel.class`
Using RMI (2)

- Run `rmiregistry`
- In the class to invoke remotely register the instance with the `rmiregistry` executable
- The client looks up the remote class in the `rmiregistry`
- The client invokes the stub, the stub marshalled the parameters and sends them to the skeleton
- The skeleton invokes the method on the running remote server gets the return values, marshalls them and returns them to the stub
- The stub unmarshalls the values and returns them to the client
**rmic (rmi compiler)**

- creates stubs and skeletons for your class
- open a command prompt
- cd to Lectures
- Make sure to SET YOUR CLASSPATH to point to those files directly! or rmic won’t work
- **type** `rmic Lecture6.DemoRMIServerImpl`
- creates files for you in the Lecture6 sub directory
rmic and generated class files
running the RMI registry

• Starts a process that listens on port 1099 on your machine
A client making a local method call

Box 1 – JVM 1
A client making a remote method invocation

**Box 1 – JVM 1**

**Box 2 – JVM 2**

TCP

**Client**

**Remote Server**

rmiregistry

rmic generates stub and skeleton
Summary

• TCP (Stream) Socket is connection between the initiator and acceptor
• TCP acceptor uses a socket factory to support multiple initiators and a single acceptor
• UDP (Datagram) socket is connectionless (fire and forget)
• RMI allow invoking methods on an object remotely (over TCP)
• Requires creating “extra” interface and generating stubs and skeletons via `rmic`
• Look up a remote object via the `rmiregistry` and invoke it through the stub returned by the `rmiregistry`.
• RMI can send object that contain other objects, etc. A complete graph of objects can be marshalled (serialized) and unmarshalled (deserialized).
Resources

• ..\Java Platform Documentation\docs\guide\rmi\index.html