Network Applications: UDP and TCP Socket Programming

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

9/17/2013

Outline

- Recap
  - Network application programming: intro to sockets
    - UDP
    - TCP

Recap: DNS

- Function
  - map between (domain name, service) to value, e.g.,
    - (www.cs.yale.edu, Addr) -> 128.36.229.30
    - (cs.yale.edu, Email) -> netra.cs.yale.edu

Recap: DatagramSocket (Java)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatagramSocket()</td>
<td>constructs a datagram socket and binds it to any available port on the local host</td>
</tr>
<tr>
<td>DatagramSocket(int port)</td>
<td>constructs a datagram socket and binds it to the specified port on the local host machine.</td>
</tr>
<tr>
<td>DatagramSocket(InetAddress addr)</td>
<td>creates a datagram socket and binds to the specified local port and address.</td>
</tr>
<tr>
<td>DatagramSocket(InetAddress bindaddr)</td>
<td>creates a datagram socket and binds to the specified local socket address.</td>
</tr>
<tr>
<td>DatagramPacket(byte[] buf, int length)</td>
<td>constructs a DatagramPacket for receiving packets of length length.</td>
</tr>
<tr>
<td>DatagramPacket(byte[] buf, int length, InetAddress addr, int port)</td>
<td>constructs a datagram packet for sending packets of length length to the specified port and address.</td>
</tr>
<tr>
<td>receive(DatagramPacket p)</td>
<td>receives a datagram packet from this socket.</td>
</tr>
<tr>
<td>send(DatagramPacket p)</td>
<td>sends a datagram packet from this socket.</td>
</tr>
<tr>
<td>close()</td>
<td>closes this datagram socket.</td>
</tr>
</tbody>
</table>

Connectionless UDP: Big Picture (Java version)

Server (running on serv)

1. create socket:
   - port, for incoming request:
     - serverSocket = new DatagramSocket();
2. read request from serverSocket:
3. generate reply, create datagram using client host address, port number:
4. send reply to serverSocket:
5. close clientSocket

Client

1. Create socket with port number:
   - DatagramSocket clientSocket = new DatagramSocket(8876);
2. If no port number is specified, the OS will pick one.
**Example: UDPServer.java**

A simple UDP server which changes any received sentence to upper case.

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

class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        // Create datagram socket
        // Bind at port 9876
        // Check socket state:
        // % netstat –p udp –n
        // System State after the Call
        // server UDP socket space
        // address:  {*:9876}
        // snd/recv buf:
        // 128.36.59.2
        // 32.5.128.36.
        // 230.2
        // address:  {128.36.2
        // 32.5:53}
        // snd/recv buf:
        // num addresses shown as *
        // local port
        // only shown at *
        // % ifconfig -a
        // local address
        // why shown as *
        // % ifconfig
        // Binding to IP Addresses
        InetAddress sIP1 = InetAddress.getByName("localhost");
        DatagramSocket ssock1 = new DatagramSocket(9876, sIP1);
        InetAddress sIP2 = InetAddress.getByName("128.36.59.2");
        DatagramSocket ssock2 = new DatagramSocket(9876, sIP2);
        DatagramSocket serverSocket = new DatagramSocket(6789);
        // Binding to IP Addresses
        // server UDP socket space
        // Public address: 128.36.59.2
        // Local address: 127.0.0.1
        // address:  {127.0.0.1:
        // 9876}
        // snd/recv buf:
        // address:  {128.36.59.2
        // 9876}
        // snd/recv buf:
        // address:  {*:6789}
        // snd/recv buf:
        // UDP demultiplexing
        // is based on matching (dst address, dst port)
        // P1 client on server
        // SP: x DP: 9876 S-IP: A D-IP: 127.0.0.1
    }
}
```
Per Socket State

- Each socket has a set of states:
  - local address
  - receive buffer size
  - send buffer size
  - timeout

See DatagramSocket API to display socket state.

Example: socket state after clients sent msgs to the server.

Java Server (UDP): Receiving

```java
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 = null;
        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), receivePacket.getPort());
            serverSocket.send(sendPacket);
        }
    }
}
```

Java Server (UDP): Processing

```java
// process data
String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();
```

Java Server (UDP): Response

```java
interface IPInetAddress {
    String get InetAddress();
    int getPort();
}
```

Example: UDPClIENT.java

A simple UDP client which reads input from keyboard, sends the input to server, and reads the reply back from the server.
Example: Java client (UDP)

```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));
        String sentence = inFromUser.readLine();
        byte[] sendData = sentence.getBytes();
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress sIPAddress = InetAddress.getByName("servname");
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, sIPAddress, 9876);
        clientSocket.send(sendPacket);
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:");
        clientSocket.close();
    }
}
```

Discussion on Example Code

- A simple upper-case echo service is among the simplest network service.
- Are there any problems with the program?

Data Encoding/Decoding

- Pay attention to encoding/decoding of data: transport layer handles only a sequence of bytes

Example: Endianness of Numbers

- int var = 0x0A0B0C0D

Example: String and Chars

- Will we always get back the same string?

Depends on default local platform charset (why?): java.nio.charset.Charset.defaultCharset()
**Example: Charset Troubles**

- Try
  - java EncodingDecoding US-ASCII UTF-8
  - java EncodingDecoding ISO-8859-1 UTF-8

**Encoding/Decoding as a Common Source of Errors**

- Please read chapter 4 of Java Network Programming for more details
- Common mistake even in many (textbook) examples:

**Discussion: UDP/DNS Server Pseudocode**

- Modify the example UDP server code to implement a DNS server.
- `flags`:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

**UDP/DNS Implementation**

- Standard UDP demultiplexing (find out return address by src.addr/src.port of UDP packet) does not always work
- DNS solution: identification remember the mapping

**Outline**

- Recap
- Network application programming
  - UDP
  - TCP

**TCP Socket Design: Starting w/ UDP**

Issue: TCP is designed to provide a pipe abstraction: server reads an ordered sequence of bytes from each connected client

```java
sock.nextByte(client1);
```

Issue 2: How is the server notified that a new client is connected?

```java
newClient = sock.getNewClient();
```
Q: How to decide where to put a new packet?
A: Packet demultiplexing is based on four tuples:
(dst addr, dst port, src addr, src port)

TCP connection-oriented demux

TCP socket big picture

Connection-oriented TCP: Big Picture (C version)
Client/server socket interaction: TCP

Server (running on hostid)

- create socket, port=x, for incoming request:
  - ServerSocket(x)
- wait for incoming connection request
  - connectionSocket = welcomeSocket.accept()
- create socket, port=x, for incoming request:
  - welcomeSocket = ServerSocket(x)
- create socket, connect to hostid, port=x
  - clientSocket = Socket()
- close connectionSocket
- read reply from clientSocket
- close clientSocket

Client

- send request using clientSocket
- read request from connectionSocket
- write reply to connectionSocket

TCP connection setup

Server Flow

- Welcome socket: the waiting room
- connSocket: the operation room

ServerSocket

- ServerSocket()
  - creates an unbound server socket.
- ServerSocket(int port)
  - creates a server socket, bound to the specified port.
- ServerSocket(int port, int backlog)
  - creates a server socket and binds it to the specified local port number, with the specified backlog.
- ServerSocket(int port, int backlog, InetAddress bindAddr)
  - creates a server with the specified port, listen backlog, and local IP address to bind to.
- bind(SocketAddress endpoint)
  - binds the ServerSocket to a specific address (IP address and port number).
- bind(SocketAddress endpoint, int backlog)
  - binds the ServerSocket to a specific address (IP address and port number).
- Socket accept()
  - listens for a connection to be made to this socket and accepts it.
- close()
  - closes this socket.

(Client)Socket

- Socket(InetAddress address, int port)
  - creates a stream socket and connects it to the specified port number at the specified IP address.
- Socket(InetAddress address, int port, InetAddress localAddr, int localPort)
  - creates a socket and connects it to the specified remote address on the specified remote host.
- Socket(String host, int port)
  - creates a stream socket and connects it to the specified port number on the named host.
- bind(SocketAddress bindpoint)
  - binds the socket to a local address.
- connect(SocketAddress endpoint)
  - connects this socket to the server.
- connect(SocketAddress endpoint, int timeout)
  - connects this socket to the server with a specified timeout value.
- InputStream getInputStream()
  - returns an input stream for this socket.
- OutputStream getOutputStream()
  - returns an output stream for this socket.
- close()
  - closes this socket.

OutputStream

- public abstract class OutputStream
  - public abstract void write(int b) throws IOException
  - public void write(byte[] data) throws IOException
  - public void write(byte[] data, int offset, int length) throws IOException
  - public void flush() throws IOException
  - public void close() throws IOException

InputStream

- public abstract class InputStream
  - public abstract int read() throws IOException
  - public void read(byte[] input) throws IOException
  - public void read(byte[] input, int offset, int length) throws IOException
  - public long skip(long n) throws IOException
  - public int available() throws IOException
  - public void close() throws IOException
TCP Example

Example client-server app:
1) client reads line from standard input (inFromUser stream), sends to server via socket (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 (inFromServer stream)

Example: Java client (TCP)
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));
        sentence = inFromUser.readLine();
        Socket clientSocket = new Socket("server.name", 6789);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
        outToServer.writeBytes(sentence + \n);
        BufferedReader inFromServer =
            new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}

Example: Java server (TCP)
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 = inFromClient.readLine();
            String modifiedSentence = clientSentence.toUpperCase();
            System.out.println("FROM CLIENT: " + modifiedSentence);
            DataOutputStream outToClient =
                new DataOutputStream(connectionSocket.getOutputStream());
            outToClient.writeBytes(modifiedSentence + \n);
            BufferedReader inFromServer =
                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
            String serverSentence = inFromServer.readLine();
            System.out.println("FROM SERVER: " + serverSentence);
        }
    }
}

Under the Hood: TCP Multiplexing

Example: Client Initiates Connection

%netstat -p tcp -n -a
**Example: TCP Handshake Done**

- **Server**
  - 128.36.232.2, 198.69.10.10
  - State: Listening
  - Port: 6789

- **Client**
  - 128.36.232.2
  - Port: 6789

**Completed connection queue:**

- 128.36.232.2
- 198.69.10.10

**Sendbuf:**

- 128.36.232.2
- 198.69.10.10

**Recvbuf:**

- 128.36.232.2
- 198.69.10.10

**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()));
            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + '
';
            DataOutputStream  outToClient = new DataOutputStream(connectionSocket.getOutputStream());
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

**Example: Java server (TCP): Processing**

- Create input stream, attached to socket
- Read in line from socket
- Packet demultiplexing is based on (dst addr, dst port, src addr, src port)
- Packet sent to the socket with the best match!

**Example: Java server (TCP): Output**

- Create output stream, attached to socket
- Write out line to socket
- End of while loop, loop back and wait for another client connection

**Analysis**

- Assume that client requests arrive at a rate of lambda/second
- Assume that each request takes 1/mu seconds
- Some basic questions
  - How long is the queue at the welcome socket?
  - What is the response time of a request?
Analysis

- Is there any interop issue in the sample program?
  - `DataOutputStream.writeBytes(String)` truncates
    - [http://docs.oracle.com/javase/1.4.2/docs/api/java/io/DataOutputStream.html#writeBytes(java.lang.String)]