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: DNS

- Key design features of DNS
  - Hierarchical domain name space allowing delegation
  - Recursive or iterative queries
Recap: DatagramSocket (Java)

- `DatagramSocket()`
  constructs a datagram socket and binds it to any available port on the local host.

- `DatagramSocket(int lport)`
  constructs a datagram socket and binds it to the specified port on the local host machine.

- `DatagramSocket(int lport, InetAddress laddr)`
  creates a datagram socket and binds to the specified local port and laddress.

- `DatagramSocket(SocketAddress bindaddr)`
  creates a datagram socket and binds to the specified local socket address.

- `DatagramPacket(byte[] buf, int length)`
  constructs a DatagramPacket for receiving packets of length length.

- `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
  constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

- `receive(DatagramPacket p)`
  receives a datagram packet from this socket.

- `send(DatagramPacket p)`
  sends a datagram packet from this socket.

- `close()`
  closes this datagram socket.
Connectionless UDP: Big Picture (Java version)

Server (running on serv)

- create socket, port=x, for incoming request:
  serverSocket = DatagramSocket( x )

- read request from serverSocket

- generate reply, create datagram using client host address, port number

- write reply to serverSocket

Client

- create socket, clientSocket = DatagramSocket()

- Create datagram using (serv, x) as (dest addr. port), send request using clientSocket

- read reply from clientSocket

- close clientSocket

- Create socket with port number:
  DatagramSocket sSock = new DatagramSocket(9876);

- 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 Server (UDP): Create Socket

```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 and bind at port 9876
        // Check socket state:
        // %netstat -p udp -n
    }
}
```
System State after the Call

server
128.36.232.5
128.36.230.2

UDP socket space

address: {*:9876}
snd/recv buf:

local port

local address
why shown as "*"?

address: {128.36.232.5:53}
snd/recv buf:

“*” indicates that the socket binds to all IP addresses of the machine:
% ifconfig -a
Binding to IP Addresses

**Server**

Public address: 128.36.59.2
Local address: 127.0.0.1

**UDP socket space**

```
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);
```
**UDP Demultiplexing**

**Server**
- Public address: 128.36.59.2
- Local address: 127.0.0.1

UDP socket space

<table>
<thead>
<tr>
<th>Address</th>
<th>Send/Recv Buf</th>
</tr>
</thead>
<tbody>
<tr>
<td>{127.0.0.1:9876}</td>
<td></td>
</tr>
<tr>
<td>{128.36.59.2:9876}</td>
<td></td>
</tr>
<tr>
<td>{128.36.232.5:53}</td>
<td></td>
</tr>
</tbody>
</table>

**Client on server**

- P1
  - SP: x
  - DP: 9876
  - S-IP: A
  - D-IP: 127.0.0.1

- P2
  - SP: y
  - DP: 9876
  - S-IP: B
  - D-IP: 128.36.59.2

**Client IP:** B

**UDP demultiplexing is based on matching (dst address, dst port)**
UDP Demultiplexing

**Server**
- Public address: 128.36.59.2
- Local address: 127.0.0.1

UDP socket space:
- Address: {127.0.0.1:9876}
- snd/recv buf:

- Address: {128.36.59.2:9876}
- snd/recv buf:

- Address: {*:6789}
- snd/recv buf:

- Address: {128.36.232.5:53}
- snd/recv buf:

**Client on server**
- P1
  - SP: x
  - DP: 9876
  - S-IP: A
  - D-IP: 127.0.0.1

**Client IP: C**
- P3
  - SP: y
  - DP: 6789
  - S-IP: C
  - D-IP: 128.36.59.2

UDP demultiplexing is based on matching (dst address, dst port)
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);
        }
    }
}
```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception {
        // process data
        String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
    }
}

dataGet() returns a pointer to an underlying buffer array; for efficiency, don't assume receive() will reset the rest of the array.

dataGetLength() returns how much data is valid.
Java Server (UDP): Response

- Java DatagramPacket:
  - `getAddress()` / `getPort()` returns the source address/port
Java server (UDP): Reply

Get IP addr
port #, of
sender

InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

Create datagram
to send to client

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
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.
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");
        
        // Create input stream
        // Create client socket
        // Translate hostname to IP address using DNS
    }
}
Example: Java client (UDP), cont.

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

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

clientSocket.send(sendPacket);

Send datagram to server

byte[] receiveData = new byte[1024];
DatagramPacket receivePacket =
new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

Read datagram from server

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

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

if not careful, query sent != query received (how?)
Example: Endianness of Numbers

\[
\text{int } \text{var} = 0\text{xA0B0C0D}
\]

ARM, Power PC, Motorola 68k, IA-64

Intel x86
Example: String and Chars

Will we always get back the same string?

client

String (UTF-16)

String.getBytes()

byte array

server

String (UTF-16)

String(rcvPkt,

0, rcvPkt.getLength());

Depends on default local platform char set (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:
    UseDatagramSockettosendoutandreceiveDatagramPacket.htm
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

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of questions</td>
<td>Number of answer RRs</td>
</tr>
<tr>
<td>Number of authority RRs</td>
<td>Number of additional RRs</td>
</tr>
<tr>
<td>Questions</td>
<td></td>
</tr>
<tr>
<td>(variable number of questions)</td>
<td></td>
</tr>
<tr>
<td>Answers</td>
<td></td>
</tr>
<tr>
<td>(variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>Authority</td>
<td></td>
</tr>
<tr>
<td>(variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>Additional information</td>
<td></td>
</tr>
<tr>
<td>(variable number of resource records)</td>
<td></td>
</tr>
</tbody>
</table>
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 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 can easily support many simultaneous TCP sockets: different connections/sessions are automatically separated into different sockets
Connection-Oriented Demux

client IP: A
- SP: x
- DP: 25
- S-IP: A
- D-IP: S

server IP: S

Client IP:B
- SP: y
- DP: 25
- S-IP: B
- D-IP: S
TCP Socket Big Picture

Client process

Server process

Client socket

Welcoming socket

Connection socket

Three-way handshake

bytes
Summary: 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

TCP provides reliable, in-order transfer of bytes (“pipe”) between client and server
**Connection-oriented TCP: Big Picture (C version)**

**server**
- `welcomeSocket=socket()`: create socket
- `bind(welcomeSocket, ...)`: specify socket address/port
- `listen(welcomeSocket, ...)`: specify that socket `welcomeSocket` is a listening socket
- `connectionSocket=accept(welcomeSocket, ...)`: get a connected connection from the queue for socket `welcomeSocket`; create a new socket identified by `connectionSocket`

**client**
- `clientSocket=socket()`: create socket
- `bind(clientSocket)`: specify socket address
- `connect(clientSocket, serverAddr, serverPort)`: initialize TCP handshake to server; return until TCP handshake is done
- `read()/write()`: do IO on socket `clientSocket`
- `close(clientSocket)`: done

**TCP connection setup**
- `read()/write()`: do IO on socket `connectionSocket`
- `close(connectionSocket)`: done
Client/server socket interaction: TCP

Server (running on hostid)

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

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

- Read request from `connectionSocket`
- Write reply to `connectionSocket`
- Close `connectionSocket`

Client

- Create socket, connect to hostid, port=x:
  - `clientSocket = Socket()`

- Send request using `clientSocket`

- Read reply from `clientSocket`
- Close `clientSocket`
Server Flow

- Create ServerSocket(6789)
- connSocket = accept()
- read request from connSocket
- Serve the request
- close connSocket

-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 port.

- **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 get InputStream()**
  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
public abstract class InputStream

- public abstract int read() throws IOException
- public int read(byte[] input) throws IOException
- public int 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)

```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));
        sentence = inFromUser.readLine();
        Socket clientSocket = new Socket("server.name", 6789);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```
Example: Java client (TCP), cont.

```java
outToServer.writeBytes(sentence + '\n');

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

modifiedSentence = inFromServer.readLine();

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

clientSocket.close();
```
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 capitalizedSentence = clientSentence.toUpperCase();
            System.out.println(capitalizedSentence);
        }
    }
}
Under the Hood: TCP Multiplexing

**server**

128.36.232.5
128.36.230.2

TCP socket space

- state: listening
- address: {*:6789, *:*}
- completed connection queue:
- sendbuf:
- recvbuf:

**client**

198.69.10.10

TCP socket space

- state: starting
- address: {198.69.10.10:1500, *:*}
- sendbuf:
- recvbuf:

%netstat -p tcp -n -a
Example: Client Initiates Connection

**server**
- Address: 128.36.232.5
- Address: 128.36.230.2
- State: listening
- Address: {*:6789, *:*}
- Completed connection queue:
  - Sendbuf:
  - Recvbuf:

**client**
- Address: 198.69.10.10
- State: connecting
- Address: {198.69.10.10:1500, 128.36.232.5:6789}
- Sendbuf:
- Recvbuf:

TCP socket space
Example: TCP Handshake Done

**server**

- 128.36.232.5
- 128.36.230.2

TCP socket space

- state: listening
- address: {*:6789, *:*}
- completed connection queue:
  
  - {128.36.232.5:6789, 198.69.10.10:1500}
- sendbuf:
- recvbuf:

**client**

- 198.69.10.10

TCP socket space

- state: connected
- address: {198.69.10.10:1500, 128.36.232.5:6789}
- sendbuf:
- recvbuf:
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()));
            Wait, on welcoming socket for contact by client
        }
    }
}
```
Example: Server accept()

**server**
128.36.232.5
128.36.230.2

TCP socket space

- state: listening
  - address: {.*,6789, *:*}
  - completed connection queue:
    - sendbuf:
    - recvbuf:

- state: established
  - address: {128.36.232.5:6789, 198.69.10.10.1500}
  - sendbuf:
  - recvbuf:

**client**
198.69.10.10

TCP socket space

- state: connected
  - address: {198.69.10.10.1500, 128.36.232.5:6789}
  - sendbuf:
  - recvbuf:

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): Processing

Create input stream, attached to socket

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

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
Example: Java server (TCP): Output

Create output stream, attached to socket

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

Write out line to socket

outToClient.writeBytes(capitalizedSentence);

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)](http://docs.oracle.com/javase/1.4.2/docs/api/java/io/DataOutputStream.html#writeBytes(java.lang.String))