Lecture 5
S2 - Summary of Network Protocols Programming in JSE for Distributed Systems
Section 2

presentation

DAD – Distributed Applications Development
Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro
Cristian Toma – Business Card

Cristian Toma
IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00
Agenda for Lecture 5

1. Networking Recap
   FTP

2. Networking Recap
   HTTP - S2 End

3. Exchange Ideas
Networking Recapitulation

Networking IP, FTP programming, TCP/IP programming issues
1. Networking TCP/IP Stack

HOW TCP/IP Works – recap from previous lecture and from BSc.

[Diagram of TCP/IP stack]

http://buildingautomationmonthly.com/tcpip-an-overview/
http://www.barrgroup.com/Embedded-Systems/How-To/Embedded-TCP-IP
1. Networking TCP/IP Stack

TCP/IP Stack Model – recap from previous lecture and from BSc.

http://talktoanit.com/c/?p=78
# 1. Networking: TCP/IP Stack

## ISO/OSI Model vs. TCP/IP – recap from previous lecture and from BSc.:

<table>
<thead>
<tr>
<th>ISO/OSI Model</th>
<th>TCP/IP DoD Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transport Layer</strong></td>
<td>TCP: protocol 6, UDP: protocol 17</td>
</tr>
<tr>
<td><strong>Internet Layer</strong></td>
<td>IP</td>
</tr>
<tr>
<td><strong>Network Access Layer 1 &amp; 2</strong></td>
<td>Ethernet, PPP, Frame Relay, MAC addresses, ARP</td>
</tr>
<tr>
<td><strong>Network Access Layer 1 &amp; 2 PDU: Bits or Data Stream</strong></td>
<td>Electrons, RF or Light</td>
</tr>
</tbody>
</table>

http://buildingautomationmonthly.com/tcpip-an-overview/
1. TCP/IP Network Programming – UDP Programming – Socket Primitives:

UDP Server:
- socket()
- bind()
- recvfrom()
- sendto()
- close()

UDP Client:
- socket()
- sento()
- recvfrom()
- close()

Network:
- Request
- Response

Recap from previous lecture and from BSc. + Multi-threading included in samples
1. TCP/IP Network Programming – UDP Programming – Socket Primitives:

```java
package eu.ase.net.udp;
import java.io.*;
import java.net.*;
public class UDPServer {
    public static void main(String[] args) {
        // get a datagram socket
        DatagramSocket socket = null;
        byte[] bufResp = null;
        byte[] bufRecv = null;
        try {
            socket = new DatagramSocket(778); // it is correct because this constructor executes "bind"
            while(true) {
                bufRecv = new byte[256];
                // receive request
                DatagramPacket packet = new DatagramPacket(bufRecv, bufRecv.length);
                socket.receive(packet);

                // figure out response
                String respString = new String("OK");
                bufResp = respString.getBytes();

                // send the response to the client at "address" and "port"
                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                packet = new DatagramPacket(bufResp, bufResp.length, address, port);
                socket.send(packet);
            }
        } catch(IOException ioe) {
        }
    }
}
```

Recap from previous lecture and from BSc. + Multi-threading included in samples
Recap from previous lecture and from BSc. + Multi-threading included in samples
1. TCP/IP Network Programming – TCP Programming – Socket Primitives:

TCP Server:
- socket()
- bind()
- listen()
- accept()
- read()/recv()
- write()/send()
- close()

TCP Client:
- socket()
- -
- -
- connect()
- write()/send()
- read()/recv()
- close

Network:
- Establishing the connection / TCP Handshake
- Request
- Response

Recap from previous lecture and from BSc. + Multi-threading included in samples
1. TCP/IP Network Programming – TCP Programming – Socket Primitives:

Recap from previous lecture and from BSc.

```java
Server

ServerSocket serverSocket = null;
Socket clientSocket = null;

boolean listening = true;

OutputStream os = null; PrintWriter out = null;
InputStream is = null; BufferedReader in = null;
String inputLine = null, outputLine = null;

//SEVERSOCKET = SOCKET+BIND+LISTEN
serverSocket = new ServerSocket(4801);
clientSocket = serverSocket.accept(); //ACCEPT

//STABILIREA CONEXIUNII
is = clientSocket.getInputStream();
in = new BufferedReader(new InputStreamReader(is));

os = clientSocket.getOutputStream();
out = new PrintWriter(os, true);

while ((inputLine = in.readLine()) != null) {
    System.out.println(inputLine);
    outputLine = new String("OK");
    out.println(outputLine);
    out.flush();
    if (outputLine.compareTo("La revedere!") == 0) {break;}
}

Client

Socket clientSocket = null;
PrintWriter outC = null;
BufferedReader inC = null;

clientSocket = new Socket(args[0],
Integer.parseInt(args[1])); //SOCKET

//CONNECT = OUT2SERVER + INfromSERVER

//OUT2SERVER
outC = new PrintWriter(clientSocket.getOutputStream(), true);

//INfromSERVER
inC = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

String lin = "";
outC.println("As vrea sa ma conectez."); //SEND
lin = inC.readLine(); //RECV
System.out.println("Server: " + lin);
```
FTP Programming – RFC 959:

for certain informative replies. The following commands specify FTP service requests:

**RETRIEVE (RETR)**

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

**STORE (STOR)**

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site, then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

**STORE UNIQUE (STOU)**
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:

```
\   /   \
A - ASCII | N - Non-print
|-><|    T - Telnet format effectors
E - EBCDIC | C - Carriage Control (ASA)
/   \     \\
I - Image
```

L <byte size> - Local byte Byte size
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

- Server.java
- ServerPI.java
- ServerDTP.java
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

FTP Client
Start TCP packets from Port 1427

FTP Server listens TCP port 21

TCP SYN

TCP SYN ACK

TCP ACK

FTP Control Response TCP PUSH,ACK

FTP Control Command TCP PUSH,ACK

FTP Control Response TCP PUSH,ACK

220 Microsoft FTP Service

331 Password required for Cristi.

230 User Cristi logged in.

200 PORT command successful.

150 Opening ASCII mode data connection for b.txt.

Data Transfer between 5001 client TCP port and 20 server TCP port
FTP Programming – RFC 959:

In order to use **active mode**, the client sends a PORT command, with the IP and port as argument.

The format for the IP and port is "h1,h2,h3,h4,p1,p2".

Each field is a decimal representation of 8 bits of the host IP, followed by the chosen data port.

For example, the client with an IP of 192.168.0.5, listening on port 5001 for the data connection will send the command "PORT 192,168,0,5,19,137".

The port fields should be interpreted as $p1\times256 + p2 = \text{port}$, or, in this example, $19\times256 + 137 = 5001$

The FTP client will be TCP server in port 5001 and the FTP server became TCP client from port 20
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

Frame 20 (79 bytes on wire, 79 bytes captured)
Internet Protocol, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.1 (192.168.0.1)
File Transfer Protocol (FTP)

<table>
<thead>
<tr>
<th>PORT 192,168,0,5,19,137</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request command: PORT</td>
</tr>
<tr>
<td>Request arg: 192,168,0,5,19,137</td>
</tr>
<tr>
<td>Active IP address: 192.168.0.5 (192.168.0.5)</td>
</tr>
<tr>
<td>Active port: 5001</td>
</tr>
</tbody>
</table>

```
```

```
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

FTP Client
After PORT negotiation listens the
5001 TCP port

FTP Server listens TCP port 20
Now in Active Mode

TCP SYN
TCP SYN ACK
TCP ACK

FTP Data Command TCP PUSH, ACK
TCP FIN ACK
TCP ACK
TCP FIN ACK
TCP ACK

0000 00 14 85 4a 6c 47 00 13 e8 cd 71 7f 08 00 45 00
0010 00 4e 1b 8c 40 00 80 06 5d c7 c0 a8 00 05 c0 a8
0020 00 01 13 89 00 14 cf c4 cd 7e cc 9c ab 85 50 18
0030 b3 a6 57 ec 00 00 54 68 69 73 20 69 73 20 61 20
0040 74 65 73 74 0b 0a 66 69 6c 65 20 73 65 6e 74 20
0050 6f 76 65 72 0d 0a 46 54 50 2e 0d 0a

...JIG...q...E.
.N...@....]

...........~...P.
This is a test file sent over...FT P...
1. TCP/IP Network Programming – FTP

FTP Programming – RFC 959:

Frame 27 (92 bytes on wire, 92 bytes captured)
Internet Protocol, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 5001 (5001), Dst Port: ftp-data (20), Seq: 1, Ack: 1, Len: 38

FTP Data

FTP Data: This is a test\r\nfile sent over\r\nFTP.\r\n
FTP Programming – RFC 959:

FTP Client
Start TCP packets from Port 1427

FTP Server listens TCP port 21

Data Transfer between 5001 client TCP port and 20 server TCP port

FTP Control Response TCP PUSH, ACK

FTP Control Command TCP PUSH, ACK

FTP Control Response TCP PUSH, ACK

TCP RST ACK

QUIT

226 Transfer complete.

221
FTP Programming – RFC 959:

In **passive mode**, the FTP server opens a dynamic port (49152–65535), sends the FTP client the server's IP address to connect to and the port on which it is listening (a 16 bit value broken into a high and low byte, like explained before) over the control stream and waits for a connection from the FTP client.

In this case the FTP client binds the source port of the connection to a dynamic port between 49152 and 65535.

To use passive mode, the client sends the PASV command to which the server would reply with something similar to "227 Entering Passive Mode (127,0,0,1,192,52)". The syntax of the IP address and port are the same as for the argument to the PORT command. HERE server port is 49204.

In extended passive mode, the FTP server operates exactly the same as passive mode, however it only transmits the port number (not broken into high and low bytes) and the client is to assume that it connects to the same IP address that was originally connected to. Extended passive mode was added by RFC 2428 in September 1998.
Section Conclusion

Fact: **DAD needs Network Programming**

In few **samples** it is simple to remember: UDP and TCP programming is useful for HTC ... didactical samples: SNMP over UDP for monitoring, SMTP/IMAP4/POP3 over TCP for e-mail notification, FTP over TCP for file and data transfer, HTTP over TCP for web & web services, and in general TCP for RPC/RMI, CORBA, JMS, SOAP, P2P-JXTA – for distributed computing and systems.
2. TCP/IP Network Programming

HTTP Programming – RFC 2616:

Request = Request-Line ; Section 5.1
* (( general-header ; Section 4.5
| request-header ; Section 5.3
| entity-header ) CRLF) ; Section 7.1
CRLF
[ message-body ] ; Section 4.3

5.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF
2. TCP/IP Network Programming

HTTP Programming – RFC 2616:

5.1.1 Method

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

Method = "OPTIONS" ; Section 9.2
| "GET" ; Section 9.3
| "HEAD" ; Section 9.4
| "POST" ; Section 9.5
| "PUT" ; Section 9.6
| "DELETE" ; Section 9.7
| "TRACE" ; Section 9.8
| "CONNECT" ; Section 9.9
| extension-method

extension-method = token
2. TCP/IP Network Programming

HTTP Programming – RFC 2616:

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>10.10.10.66</td>
<td>72.14.221.104</td>
<td>TCP</td>
<td>ttyinfo &gt; http</td>
</tr>
<tr>
<td>2</td>
<td>0.046480</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>http &gt; ttyinfo</td>
</tr>
<tr>
<td>3</td>
<td>0.049535</td>
<td>10.10.10.66</td>
<td>72.14.221.104</td>
<td>TCP</td>
<td>ttyinfo &gt; http</td>
</tr>
<tr>
<td>4</td>
<td>0.109161</td>
<td>10.10.10.66</td>
<td>72.14.221.104</td>
<td>HTTP</td>
<td>GET / HTTP/1.1</td>
</tr>
<tr>
<td>5</td>
<td>0.148781</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>http &gt; ttyinfo</td>
</tr>
<tr>
<td>6</td>
<td>0.156888</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>[TCP segment of a]</td>
</tr>
<tr>
<td>7</td>
<td>0.157715</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>[TCP segment of a]</td>
</tr>
<tr>
<td>8</td>
<td>0.157759</td>
<td>10.10.10.66</td>
<td>72.14.221.104</td>
<td>TCP</td>
<td>ttyinfo &gt; http</td>
</tr>
<tr>
<td>9</td>
<td>0.185421</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>[TCP segment of a]</td>
</tr>
<tr>
<td>10</td>
<td>0.201322</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>[TCP segment of a]</td>
</tr>
<tr>
<td>11</td>
<td>0.201368</td>
<td>10.10.10.66</td>
<td>72.14.221.104</td>
<td>TCP</td>
<td>ttyinfo &gt; http</td>
</tr>
<tr>
<td>12</td>
<td>0.201518</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>TCP</td>
<td>[TCP segment of a]</td>
</tr>
<tr>
<td>13</td>
<td>0.201733</td>
<td>72.14.221.104</td>
<td>10.10.10.66</td>
<td>HTTP</td>
<td>HTTP/1.1 300 ok</td>
</tr>
</tbody>
</table>

- Frame 4 (255 bytes on wire, 255 bytes captured)
- Transmission Control Protocol, Src Port: ttyinfo (2012), Dst Port: http (80), Seq: 1, Ack 2

```
GET / HTTP/1.1
Host: www.google.com
```

```
<?xml version=...customer;b
```

```
Keep-alive: cont
```

```
ent-type: application/x-www-form-urlencoded
```

- b...Bpu...E.
- ...ABO... ...B.
- .h...P.F...chnp.
- GET / HTTP
- /1.1...Us er-Agent
- e.ro...Ac cept: te
- if, imag e/jpeg,
- x; q=0.2, w/*; q=
- .Conn ection:
- keep-all ve...cont
- -urlenco ded....

2. TCP/IP Network Programming

Q&A - Distributed Applications Development – Section 2
Summary of Network Protocols Programming in JSE for Distributed Systems

SNMP – www.snmp4j.org
* Zabbix, Cacti, Nagios, Ganglia?

SMTP – java.mail.*;
POP3
IMAP4

- Items necessary for section 2 learning – Network Protocols Programming:
  - JAVA Source Code – Lectures + Labs/Seminars – http://acs.ase.ro,
  - Network Standards & Protocols – RFCs + Lectures,
  - Network Traffic Analyzer – Wire-Shark

- Do we need SNMP for programming and development of distributed applications? Are useful SMTP/POP3/IMAP4 distributed applications development? Do we need FTP & HTTP?

- What are the advantages and disadvantages for “reinventing the weel” when are in the market standard software libraries and frameworks?
2. TCP/IP Network Programming

Q&A - Distributed Applications Development – Section 2
Summary of Network Protocols Programming in JSE for Distributed Systems

Issues discussed during the network protocols programming:

- Client/Server Model
- Request/Response Message Passing Model – the client asks info or resources and the server responds
- Connection oriented socket versus datagram socket – reliable versus non-reliable data transfer – TCP vs. UDP
- The messages format – headers & info – overload vs. payload
- Protocols and servers statefull vs. stateless
- Protocols with messages exchange in synchronous or asynchronous mode – network call-back functions
- The security of the communications between the clients and servers is an important issue, but it is not included in this lecture because the curriculum and time resources.
Fact: **DAD needs Network Programming**

In few **samples** it is simple to remember: UDP and TCP programming is useful for HTC ... didactical samples: SNMP over UDP for monitoring, SMTP/IMAP4/POP3 over TCP for e-mail notification, FTP over TCP for file and data transfer, HTTP over TCP for web & web services, and in general TCP for RPC/RMI, CORBA, JMS, SOAP, P2P-JXTA – for distributed computing and systems.
Java Sockets Network Programming & Java Servlet Technology Intro

Communicate & Exchange Ideas
Questions & Answers!

But wait...
There’s More!
Thanks!

DAD – Distributed Application Development
End of Lecture 5 – Section 2