Lecture 10

Java SE Network Programming

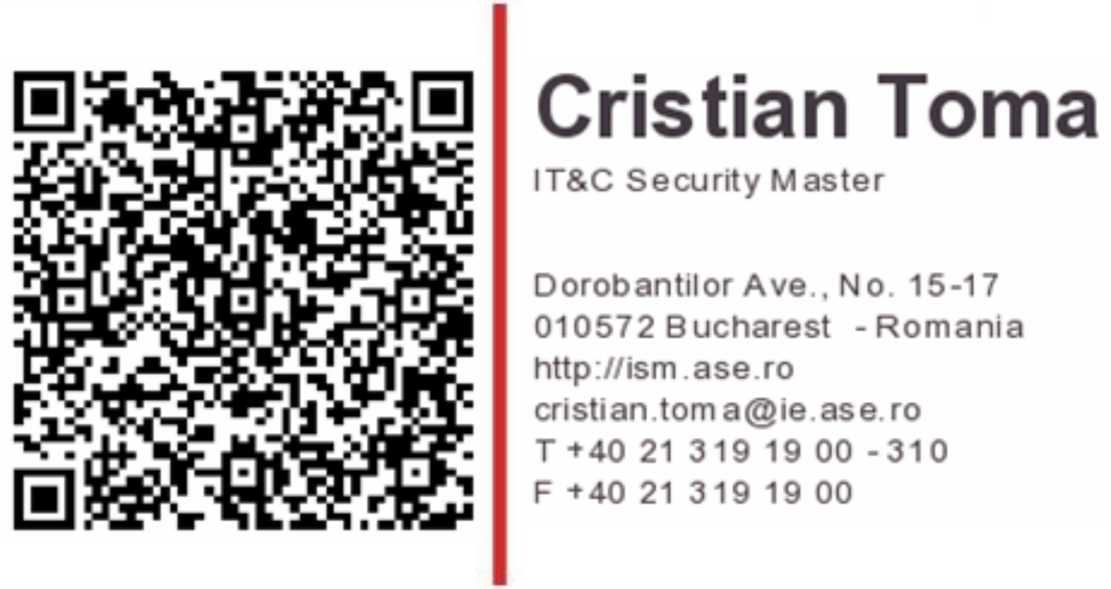
presentation

Java Programming – Software App Development

Cristian Toma

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

Cristian Toma – Business Card



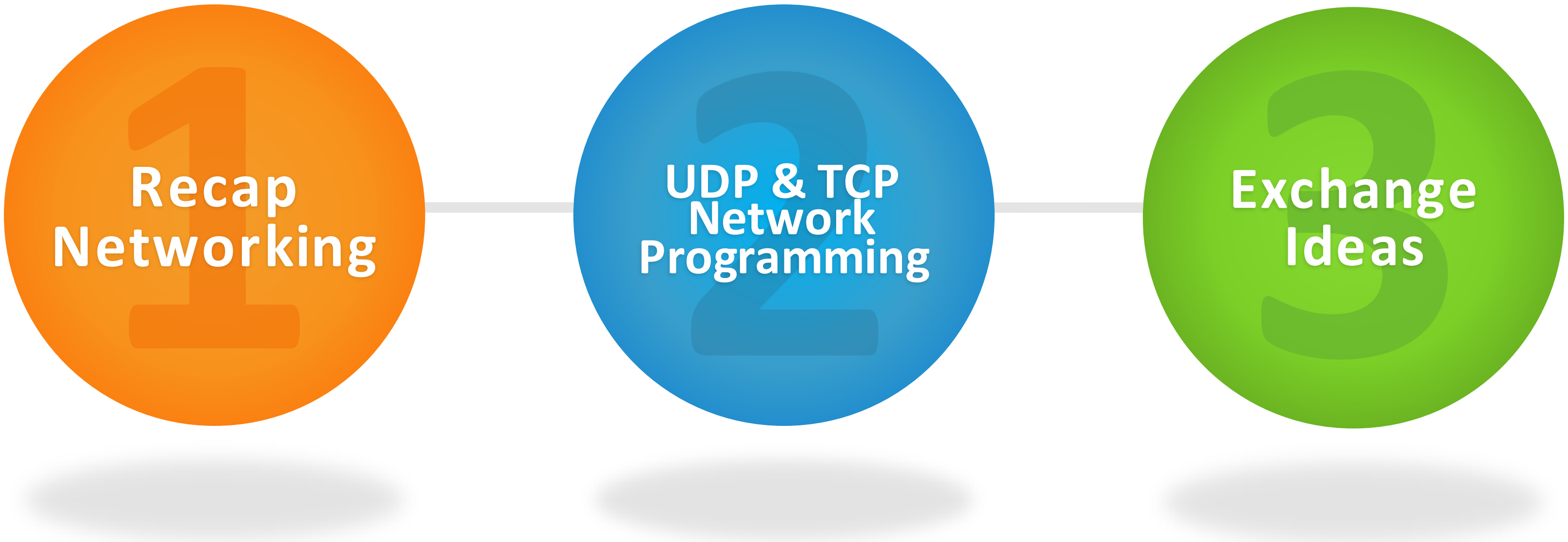
Cristian Toma

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

IT&C SECURITY MASTER
ISM
2005
ism.ase.ro
Scientia potestas est

Agenda for Lecture 10



1 Recap Networking

2 UDP & TCP Network Programming

3 Exchange Ideas

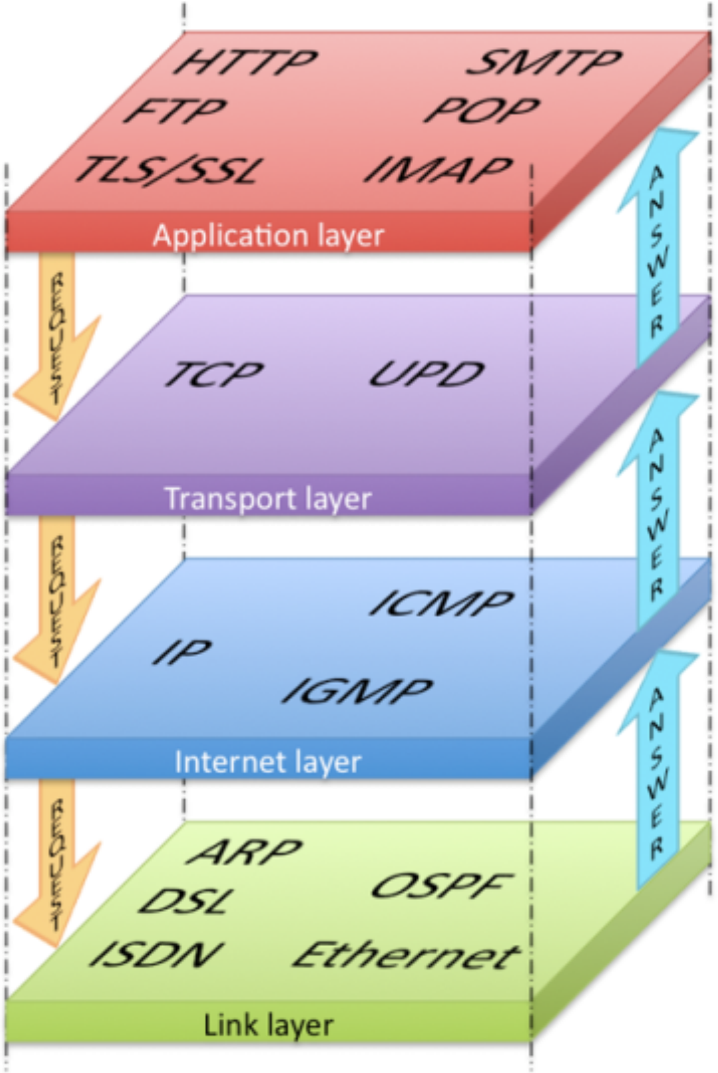
1

Networking IP, UDP and TCP programming, TCP/IP state machine

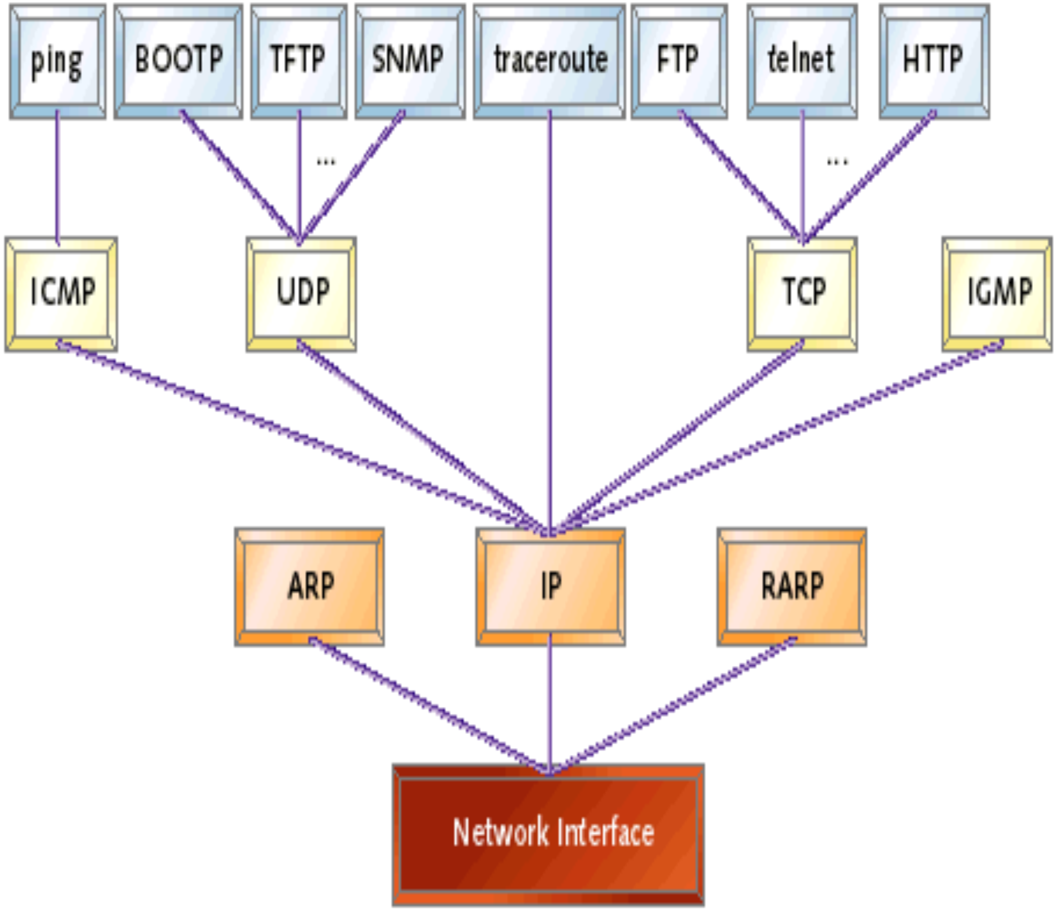
Networking Recapitulation

1. **Networking** TCP/IP Stack

HOW TCP/IP Works:



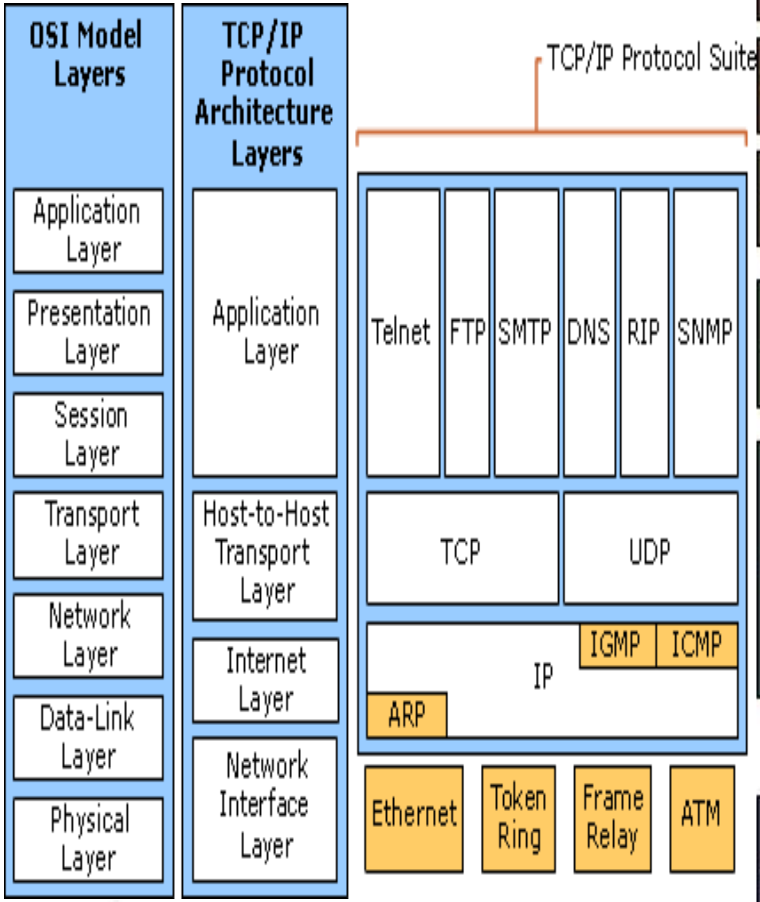
http://buildingautomationmonthly.com/tcpip-an-overview/

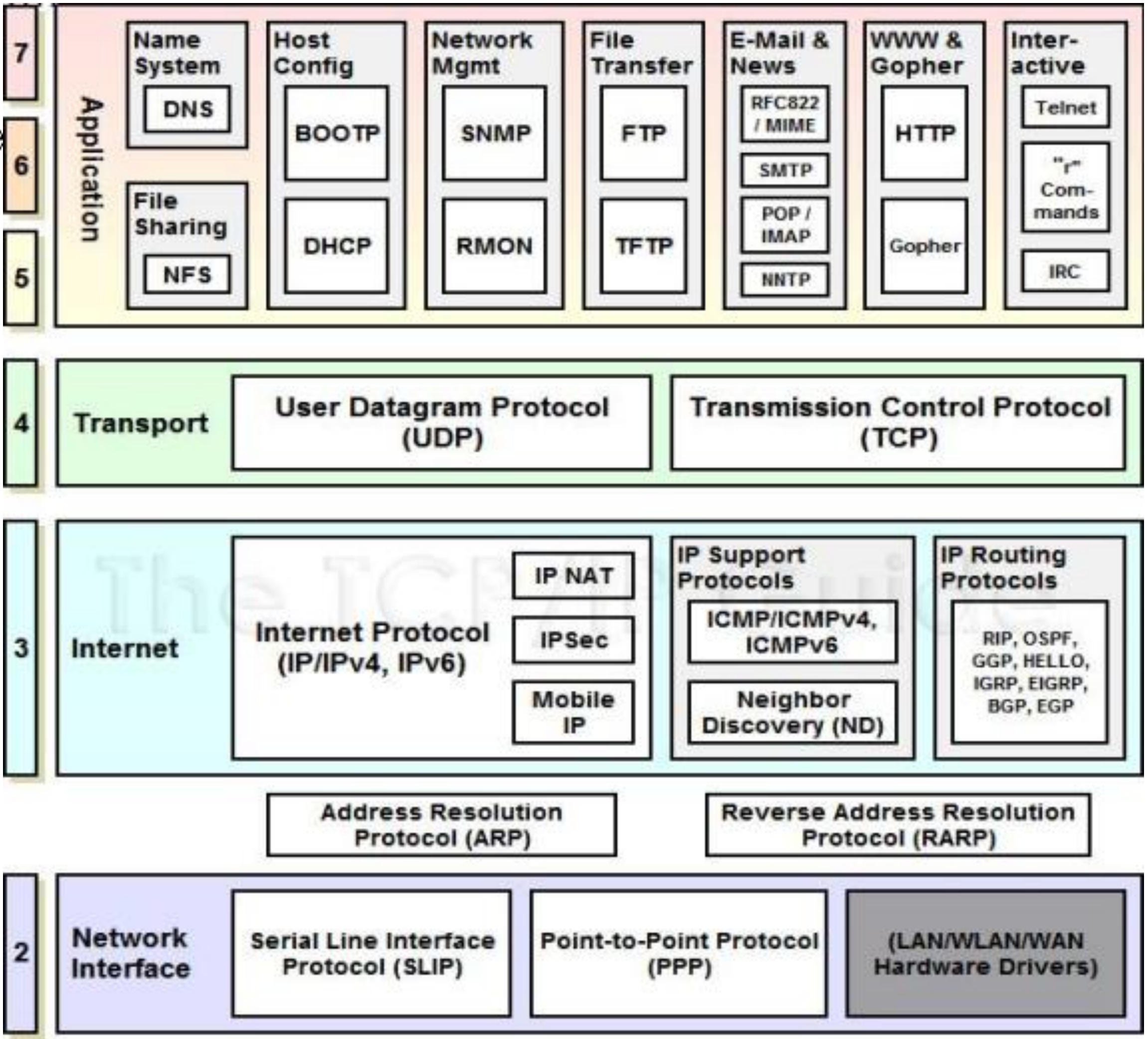


http://www.barrgroup.com/Embedded-Systems/How-To/Embedded-TCP-IP

1. **Networking** TCP/IP Stack

TCP/IP Stack Model:



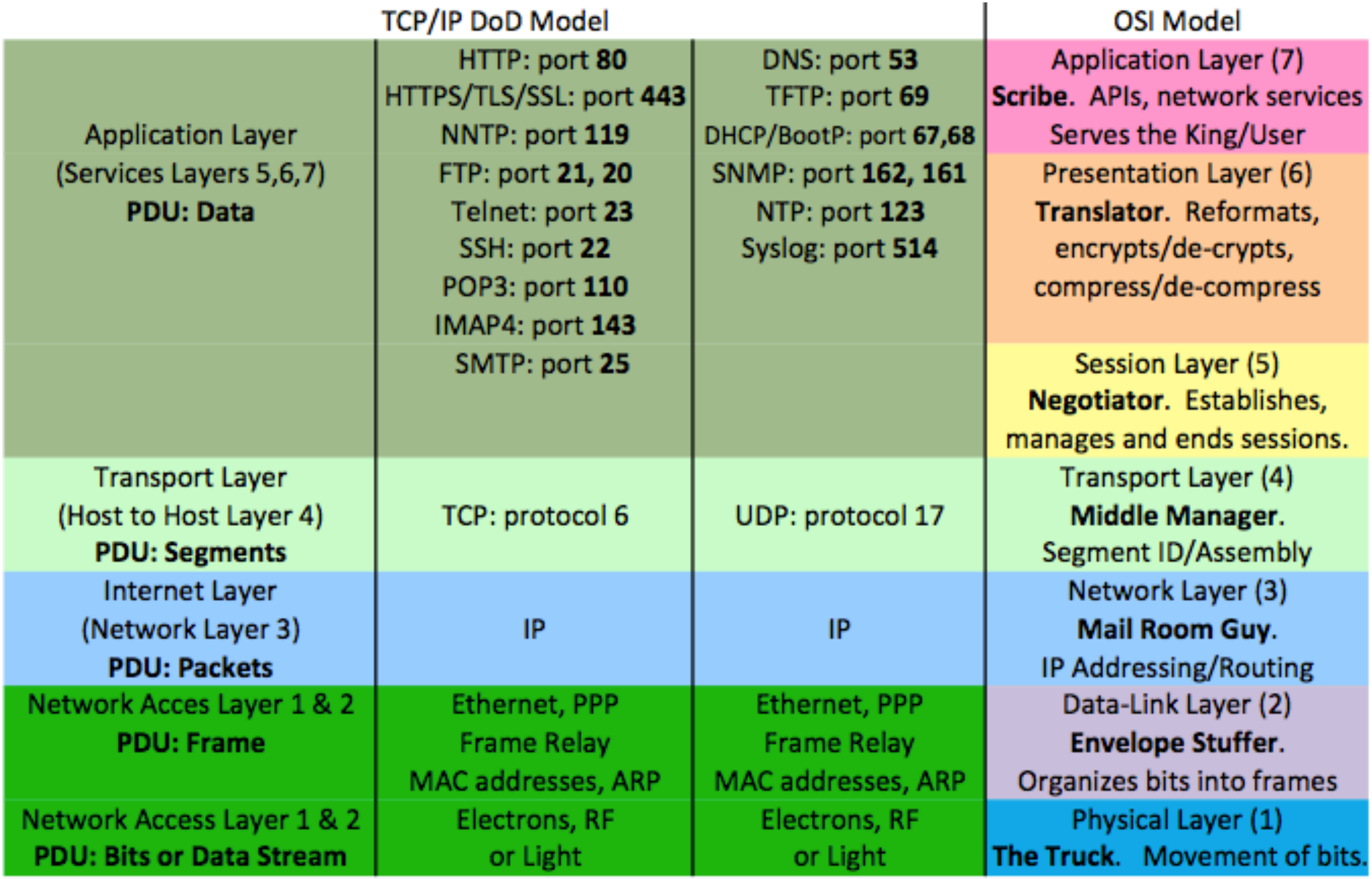


http://technet.microsoft.com/en-us/library/cc786128(v=ws.10).aspx

http://talktoanit.com/c/?p=78

1. **Networking** TCP/IP Stack

ISO/OSI Model vs. TCP/IP:

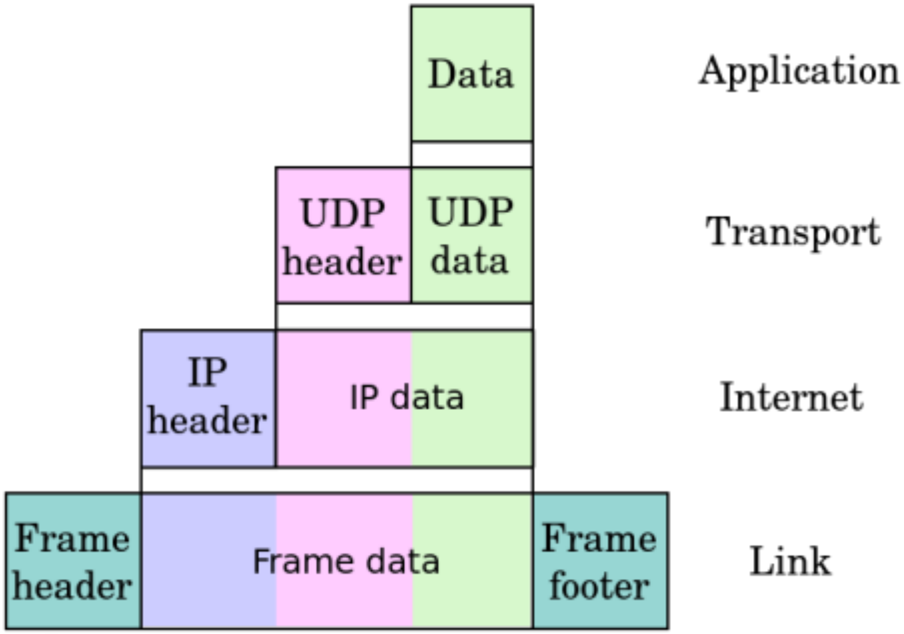


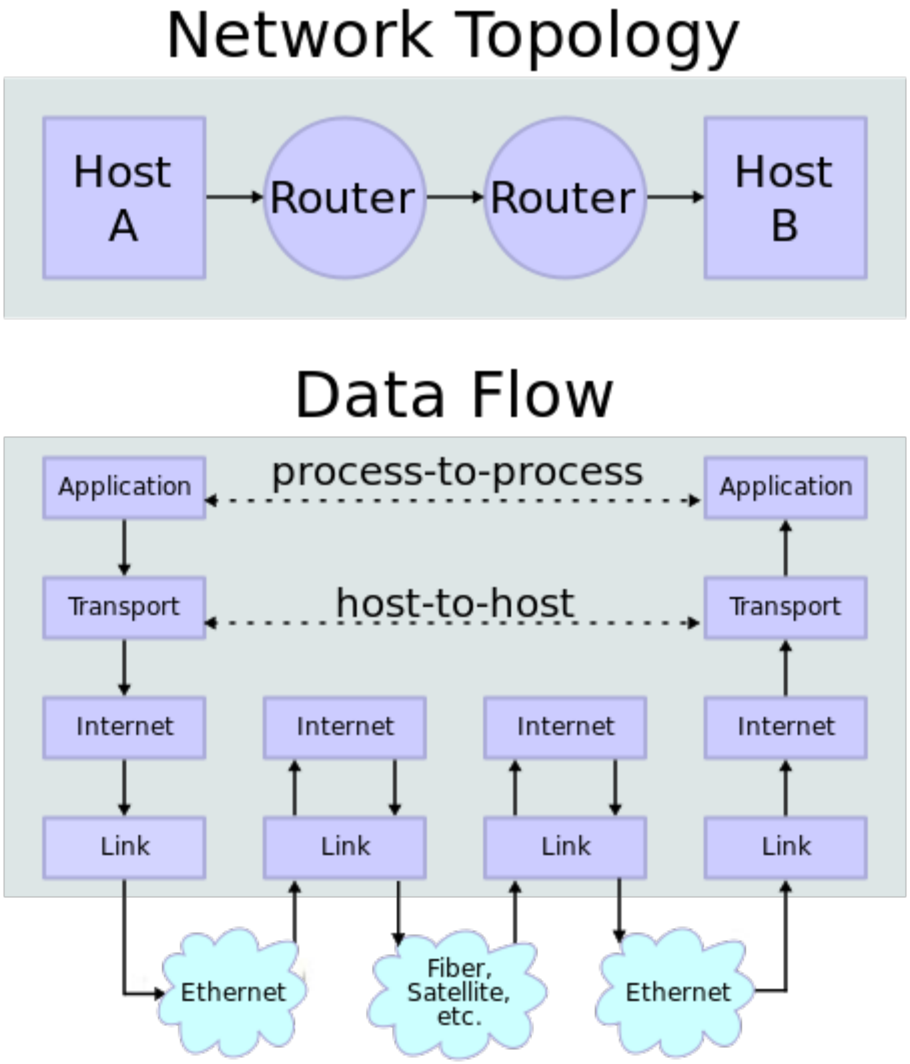
TCP/IP DoD Model			OSI Model
Application Layer (Services Layers 5,6,7) **PDU: Data**	HTTP: port **80** HTTPS/TLS/SSL: port **443** NNTP: port **119**	DNS: port **53** TFTP: port **69** DHCP/BootP: port **67,68**	Application Layer (7) **Scribe.** APIs, network services Serves the King/User
	FTP: port **21, 20** Telnet: port **23** SSH: port **22** POP3: port **110** IMAP4: port **143**	SNMP: port **162, 161** NTP: port **123** Syslog: port **514**	Presentation Layer (6) **Translator.** Reformats, encrypts/de-crypts, compress/de-compress
	SMTP: port **25**		Session Layer (5) **Negotiator.** Establishes, manages and ends sessions.
Transport Layer (Host to Host Layer 4) **PDU: Segments**	TCP: protocol 6	UDP: protocol 17	Transport Layer (4) **Middle Manager.** Segment ID/Assembly
Internet Layer (Network Layer 3) **PDU: Packets**	IP	IP	Network Layer (3) **Mail Room Guy.** IP Addressing/Routing
Network Acces Layer 1 & 2 **PDU: Frame**	Ethernet, PPP Frame Relay MAC addresses, ARP	Ethernet, PPP Frame Relay MAC addresses, ARP	Data-Link Layer (2) **Envelope Stuffer.** Organizes bits into frames
Network Access Layer 1 & 2 **PDU: Bits or Data Stream**	Electrons, RF or Light	Electrons, RF or Light	Physical Layer (1) **The Truck.** Movement of bits.

http://buildingautomationmonthly.com/tcpip-an-overview/

1. Networking TCP/IP Stack

TCP/IP Message Encapsulation:

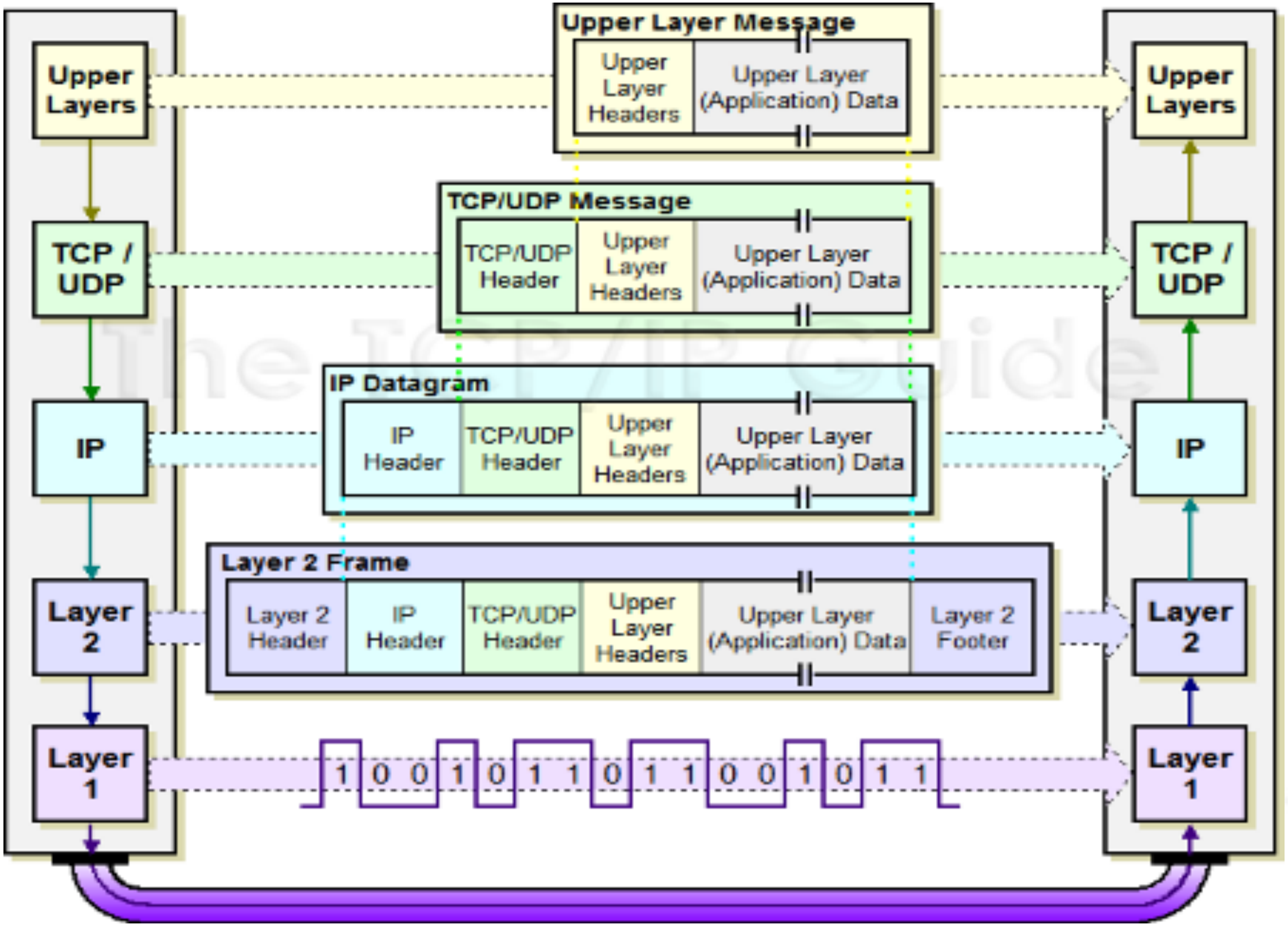




http://en.wikipedia.org/wiki/Internet_protocol_suite

1. **Networking** TCP/IP Stack

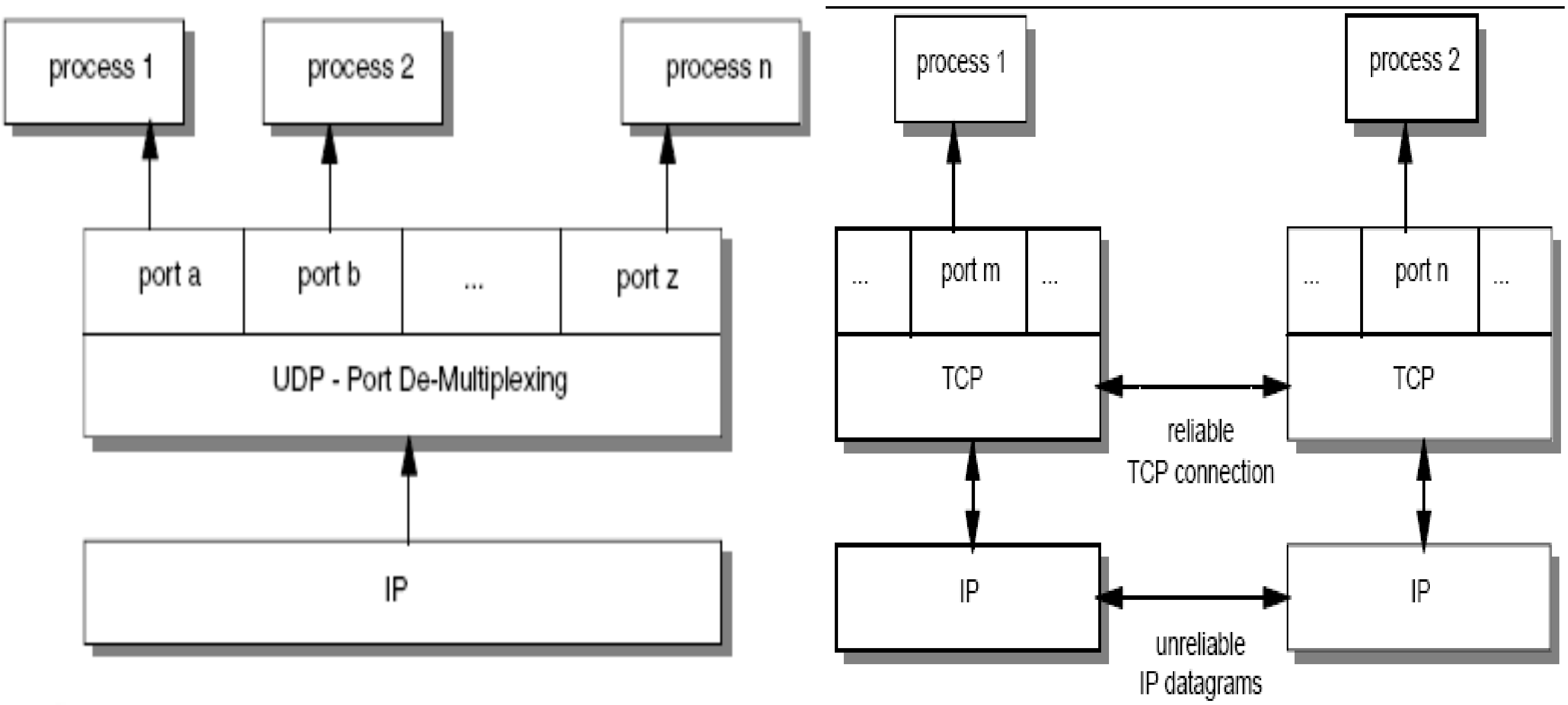
TCP/IP Message Flow:



http://buildingautomationmonthly.com/tcpip-an-overview/

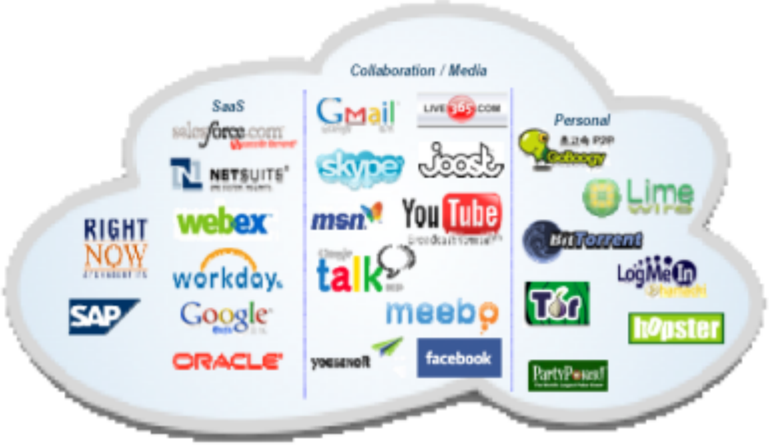
1. **Networking** TCP/IP Stack

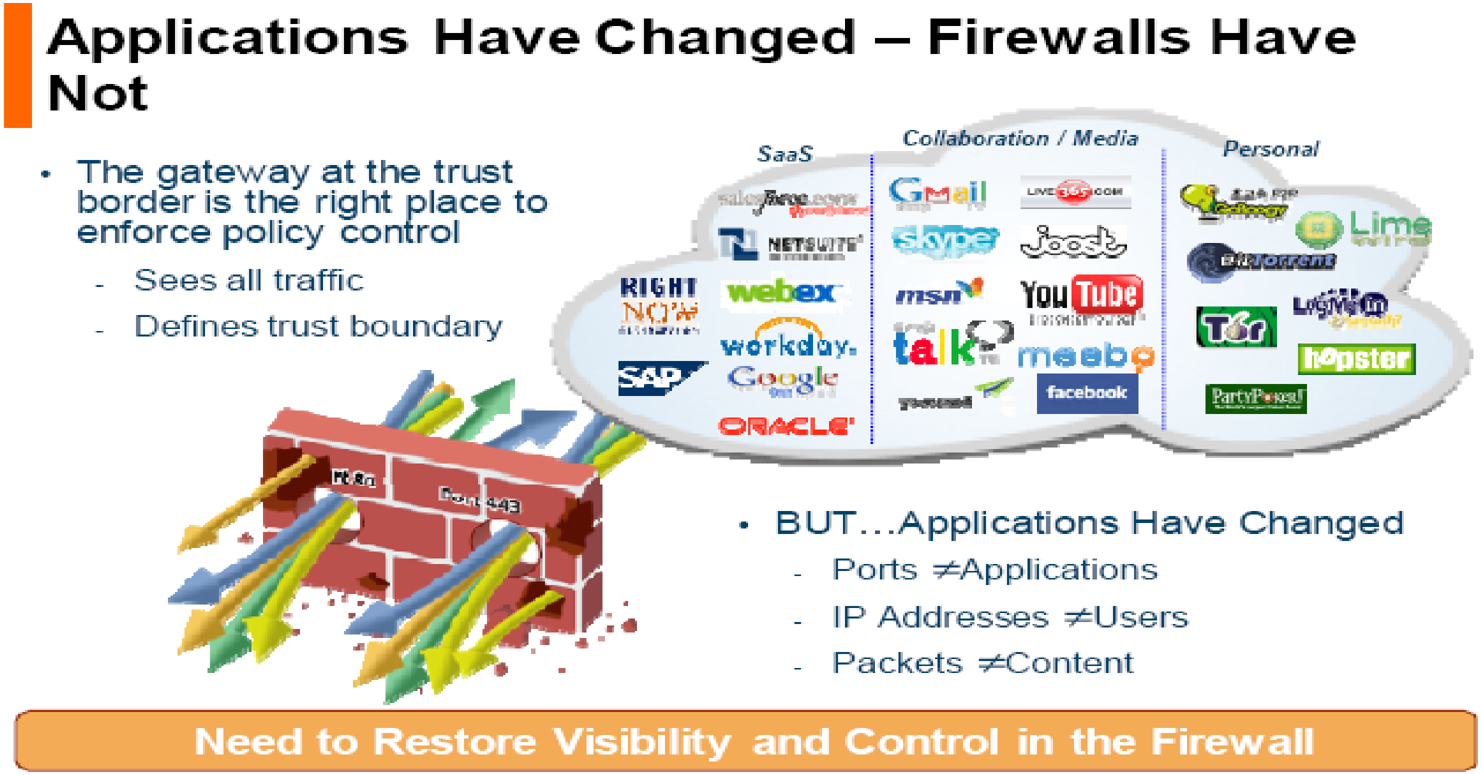
TCP/IP App/Port Multiplexing:



1. **Networking** TCP/IP Stack

What is running on port 80?





Applications Have Changed – Firewalls Have Not

- The gateway at the trust border is the right place to enforce policy control
  - Sees all traffic
  - Defines trust boundary

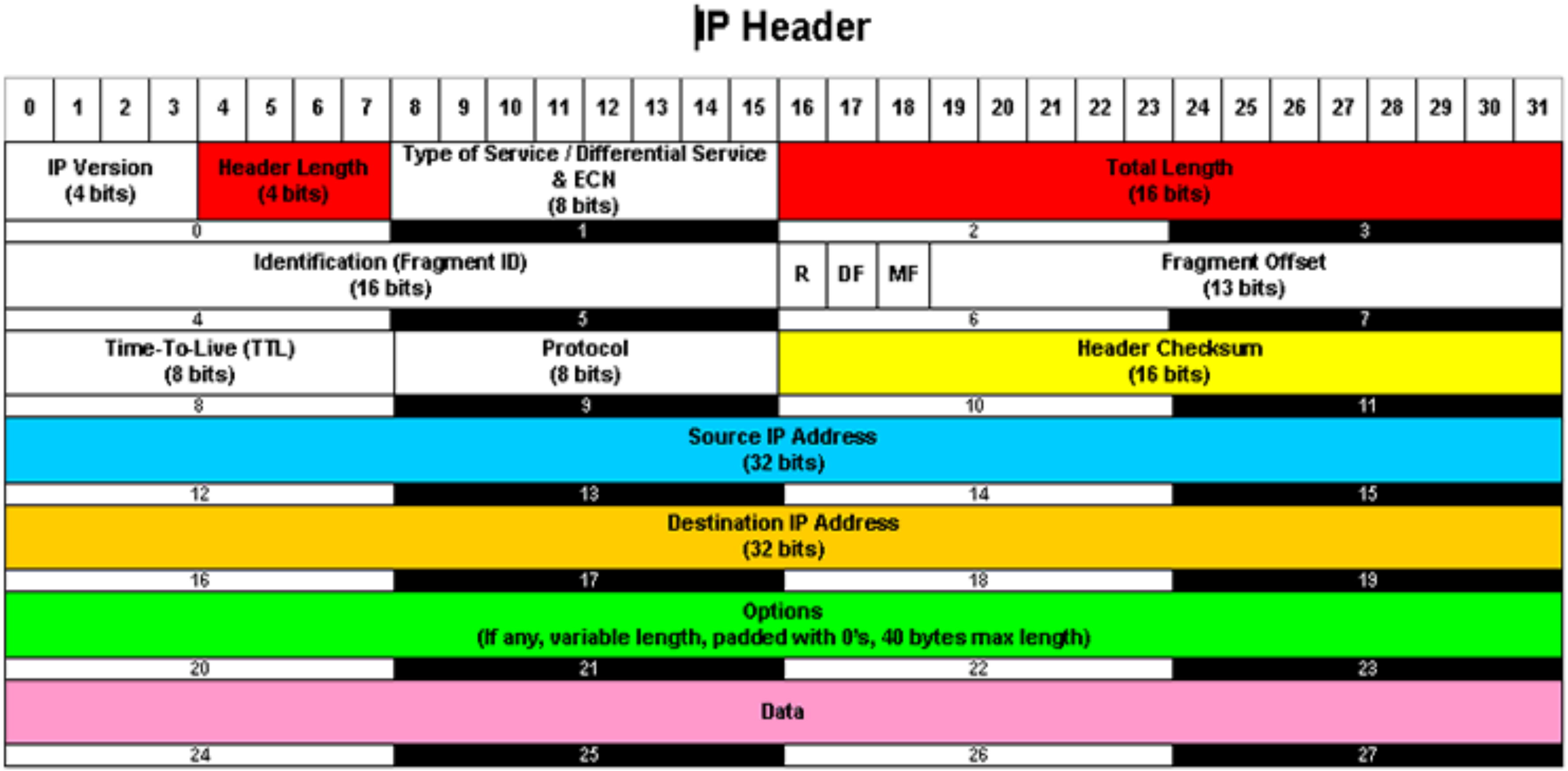
SaaS
Collaboration / Media
Personal
Pt 80
Port 443

- BUT…Applications Have Changed
  - Ports ≠Applications
  - IP Addresses ≠Users
  - Packets ≠Content

Need to Restore Visibility and Control in the Firewall

1. **Networking** TCP/IP Stack

IP Header - RFC 791 – Submasking + Routing + NAT:



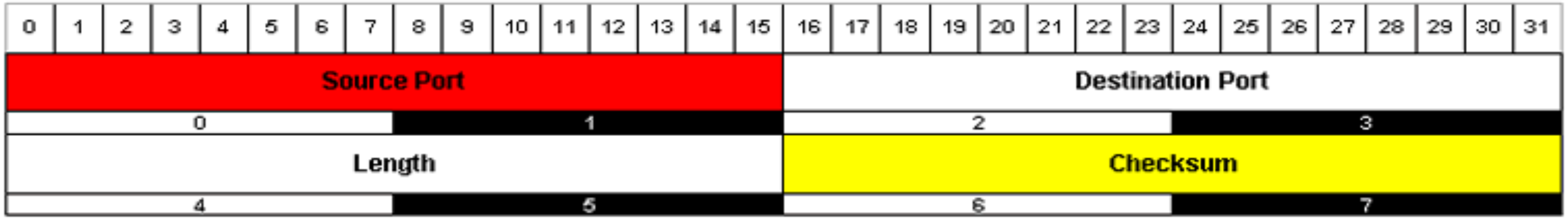
IP Header

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="4">IP Version
(4 bits)</td><td colspan="4">Header Length
(4 bits)</td><td colspan="8">Type of Service / Differential Service
& ECN
(8 bits)</td><td colspan="16">Total Length
(16 bits)</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="16">Identification (Fragment ID)
(16 bits)</td><td>R</td><td>DF</td><td>MF</td><td colspan="13">Fragment Offset
(13 bits)</td></tr>
<tr><td colspan="8">4</td><td colspan="8">5</td><td colspan="8">6</td><td colspan="8">7</td></tr>
<tr><td colspan="8">Time-To-Live (TTL)
(8 bits)</td><td colspan="8">Protocol
(8 bits)</td><td colspan="16">Header Checksum
(16 bits)</td></tr>
<tr><td colspan="8">8</td><td colspan="8">9</td><td colspan="8">10</td><td colspan="8">11</td></tr>
<tr><td colspan="32">Source IP Address
(32 bits)</td></tr>
<tr><td colspan="8">12</td><td colspan="8">13</td><td colspan="8">14</td><td colspan="8">15</td></tr>
<tr><td colspan="32">Destination IP Address
(32 bits)</td></tr>
<tr><td colspan="8">16</td><td colspan="8">17</td><td colspan="8">18</td><td colspan="8">19</td></tr>
<tr><td colspan="32">Options
(If any, variable length, padded with 0's, 40 bytes max length)</td></tr>
<tr><td colspan="8">20</td><td colspan="8">21</td><td colspan="8">22</td><td colspan="8">23</td></tr>
<tr><td colspan="32">Data</td></tr>
<tr><td colspan="8">24</td><td colspan="8">25</td><td colspan="8">26</td><td colspan="8">27</td></tr>
</table>

1. **Networking** TCP/IP Stack

UDP Header – RFC 768 – Connection-less vs. TCP Header – RFC 793 – Connection-oriented

UDP Header



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

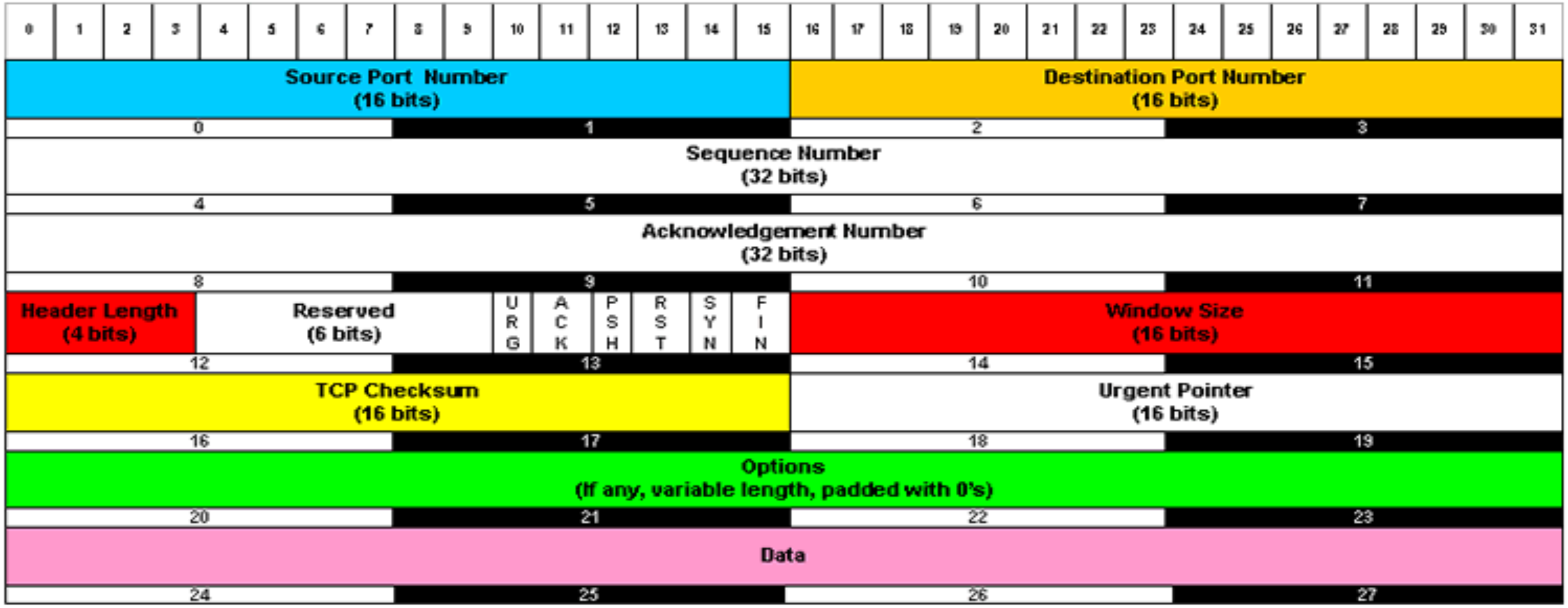
Source Port | Destination Port

0 1 2 3

Length | Checksum

4 5 6 7

TCP Header



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Source Port Number (16 bits) | Destination Port Number (16 bits)

0 1 2 3

Sequence Number (32 bits)

4 5 6 7

Acknowledgement Number (32 bits)

8 9 10 11

Header Length (4 bits) | Reserved (6 bits) | URG | ACK | PSH | RST | SYN | FIN | Window Size (16 bits)

12 13 14 15

TCP Checksum (16 bits) | Urgent Pointer (16 bits)

16 17 18 19

Options (If any, variable length, padded with 0's)

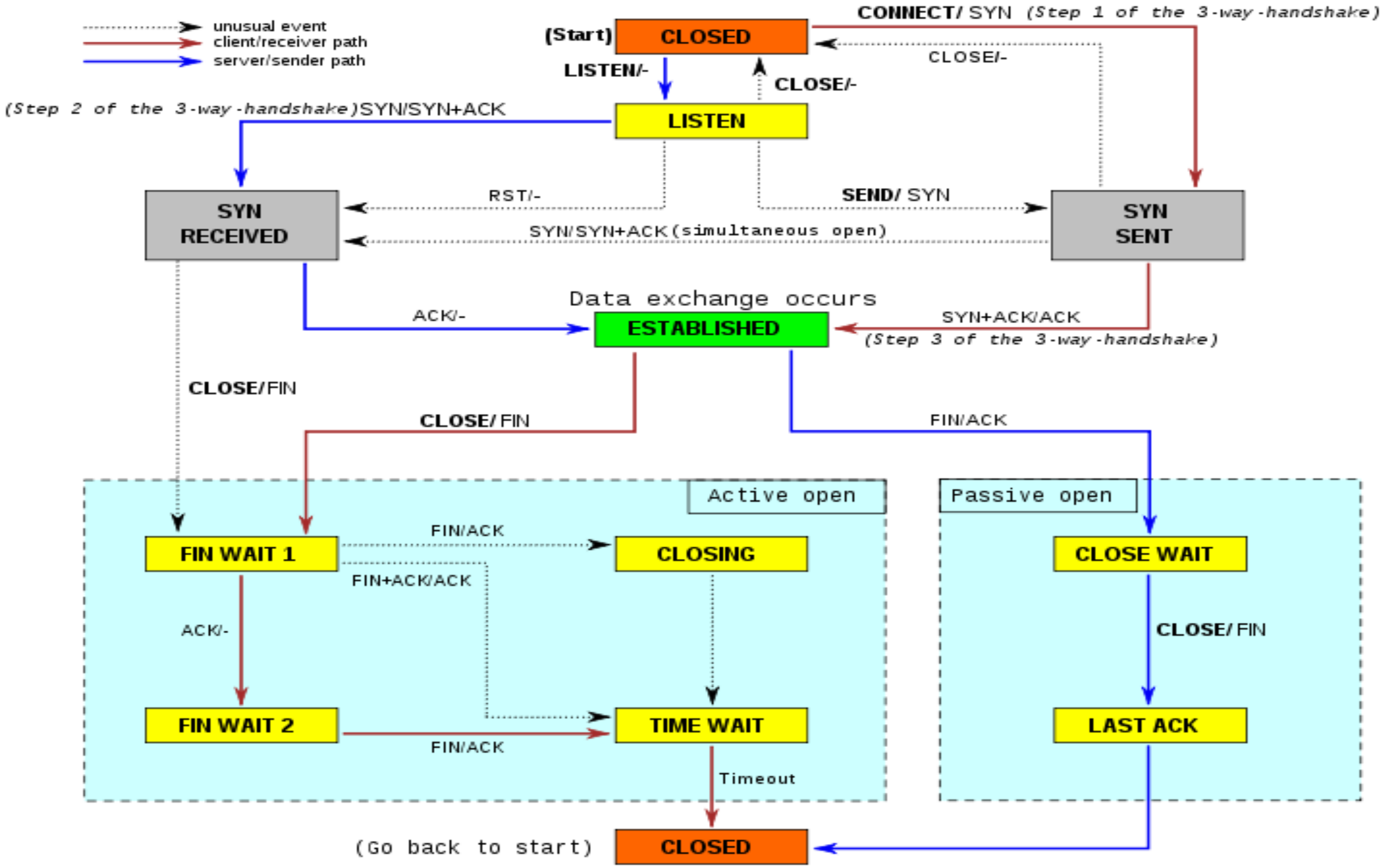
20 21 22 23

Data

24 25 26 27

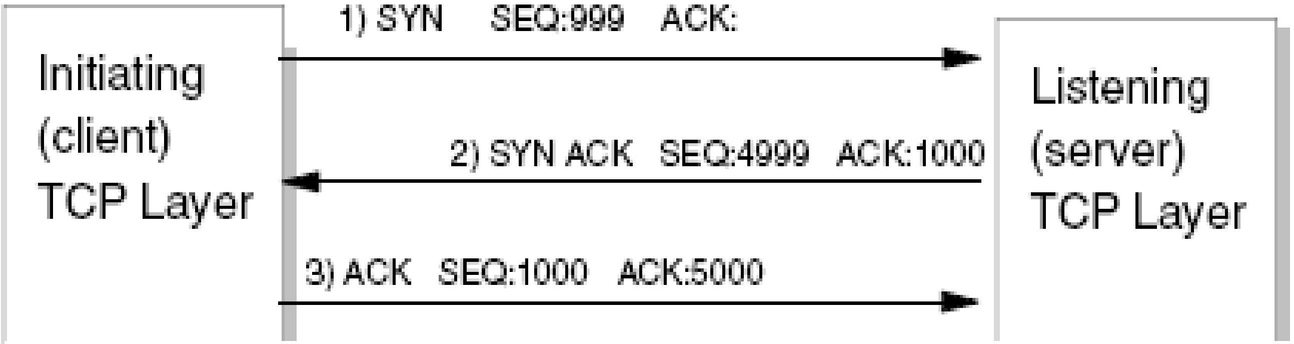
1. **Networking** TCP/IP Stack

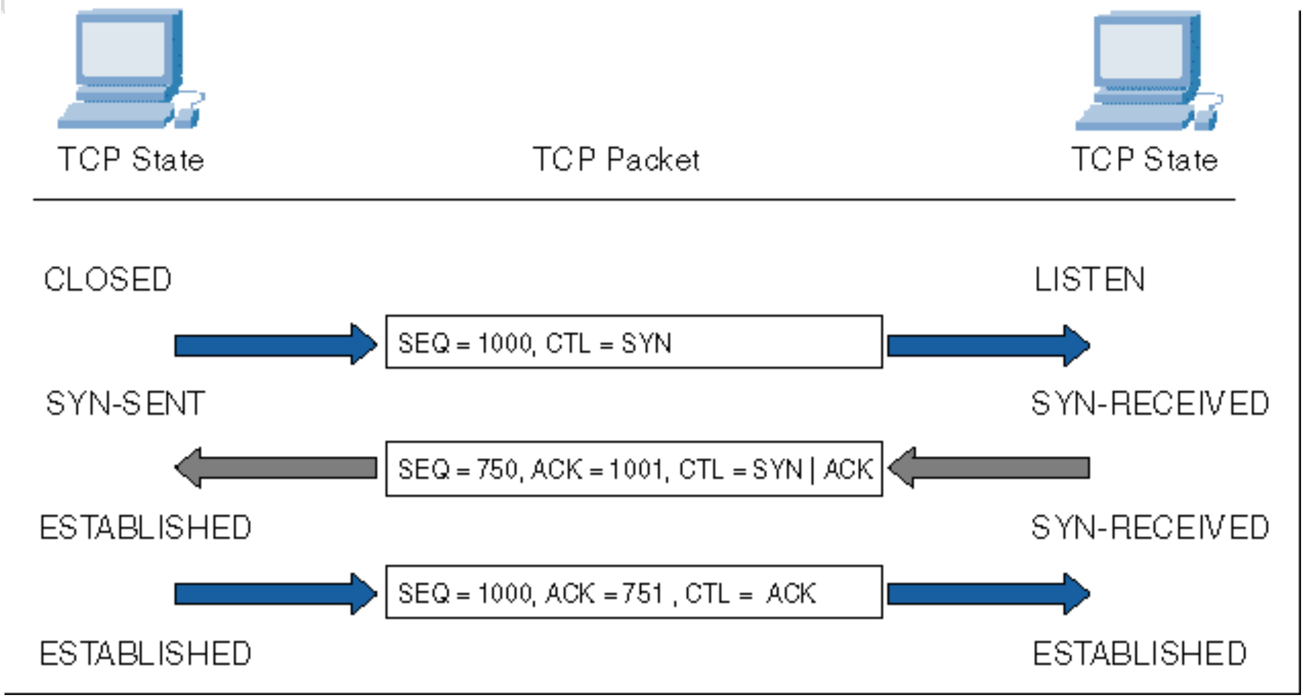
TCP State Machine – RFC 793:



1.2 TCP/IP Networking Programming

TCP Handshake:





http://www.cisco.com/web/about/ac123/ac147/ac174/ac196/about_cisco_ipj_archive_article09186a00800c8417.html

Section Conclusion

Fact: **Java is suitable for Networking**

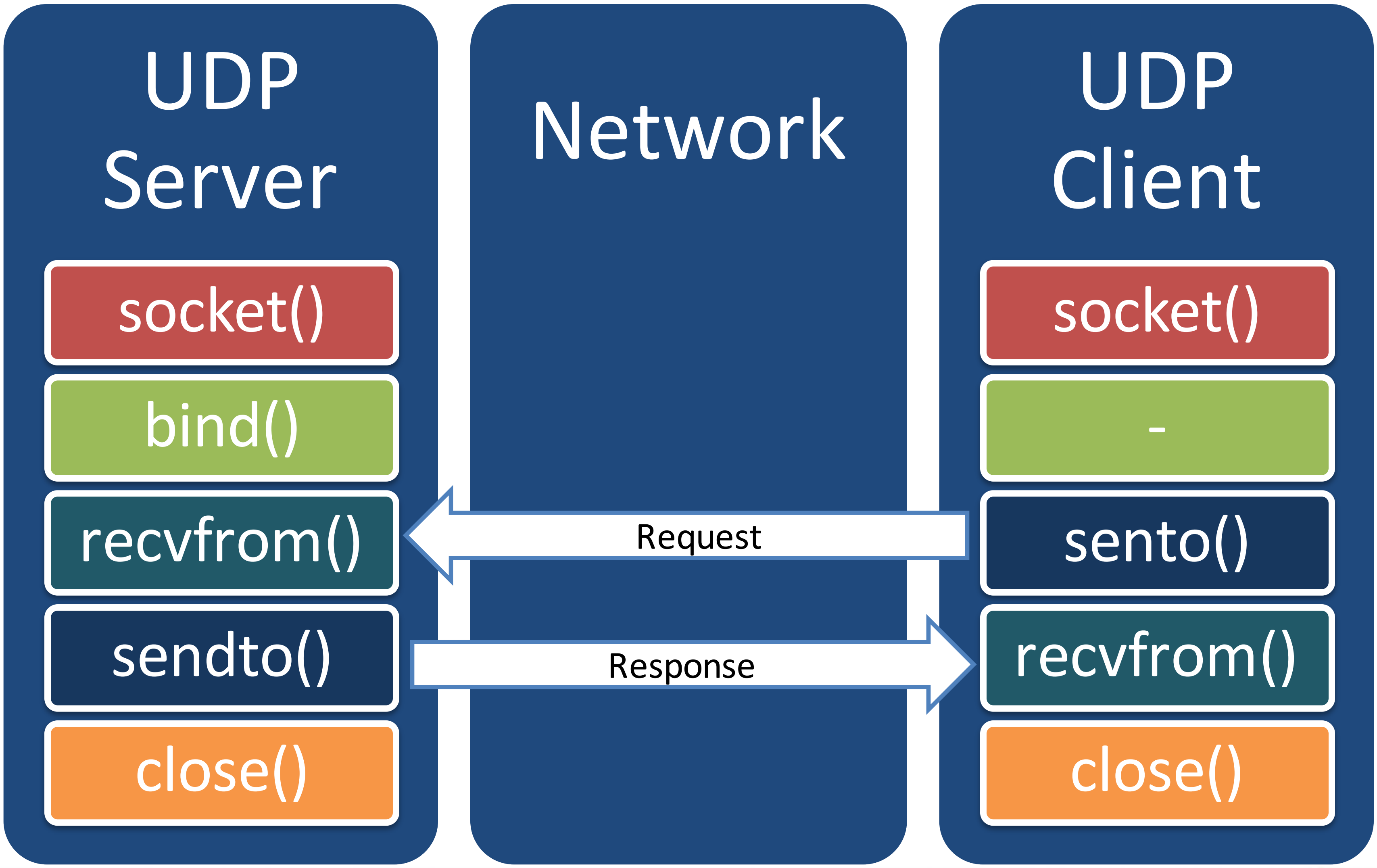
In few **samples** it is simple to understand: UDP and TCP programming is useful for HTC – High Throughput Computing (Distributed Computing), .

2

UDP Client-Server programming, TCP Client-Server programming, FTP Server

Network UDP & TCP Programming

2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:



UDP Server

socket()

bind()

recvfrom()

sendto()

close()

Network

Request

Response

UDP Client

socket()

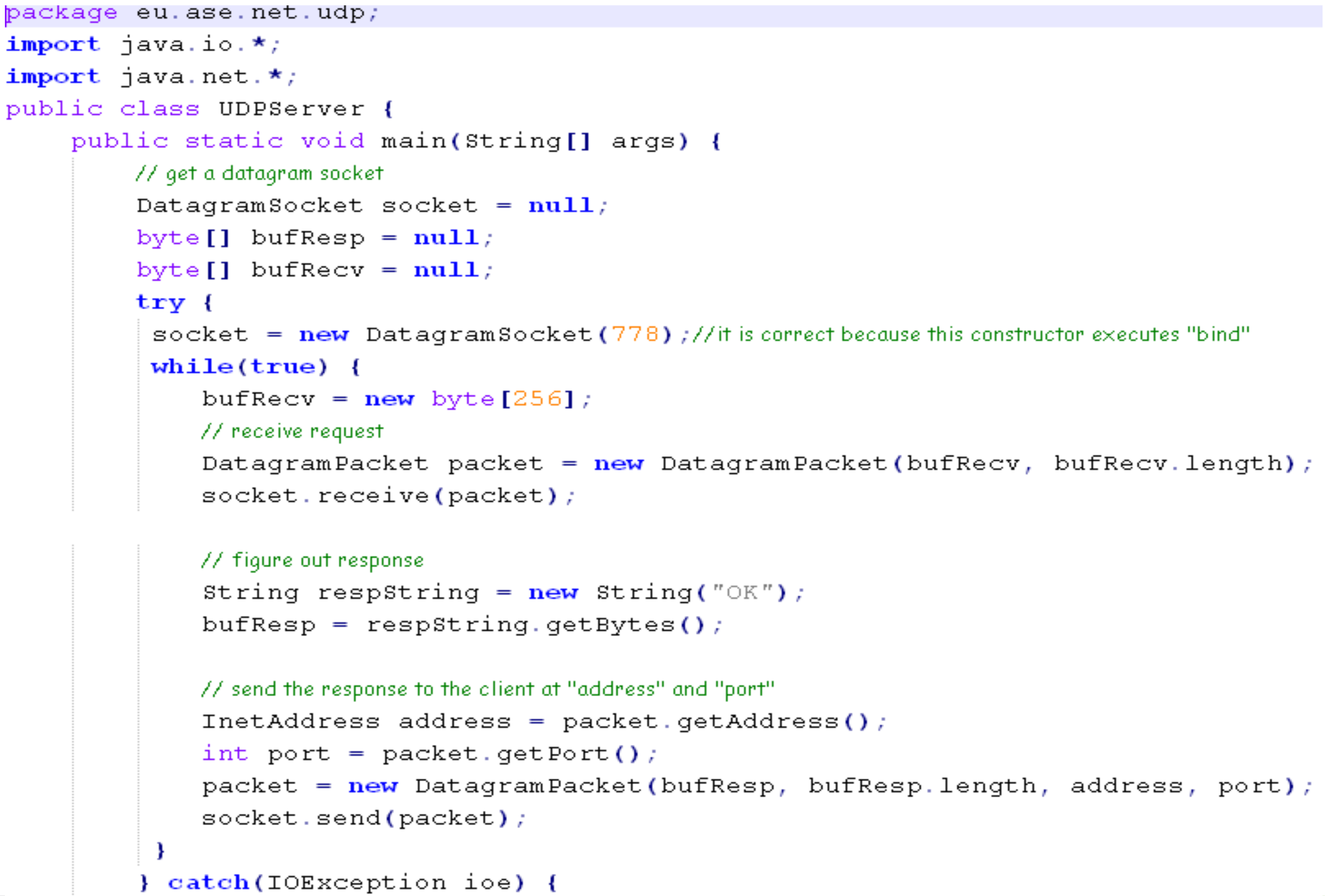
-

sento()

recvfrom()

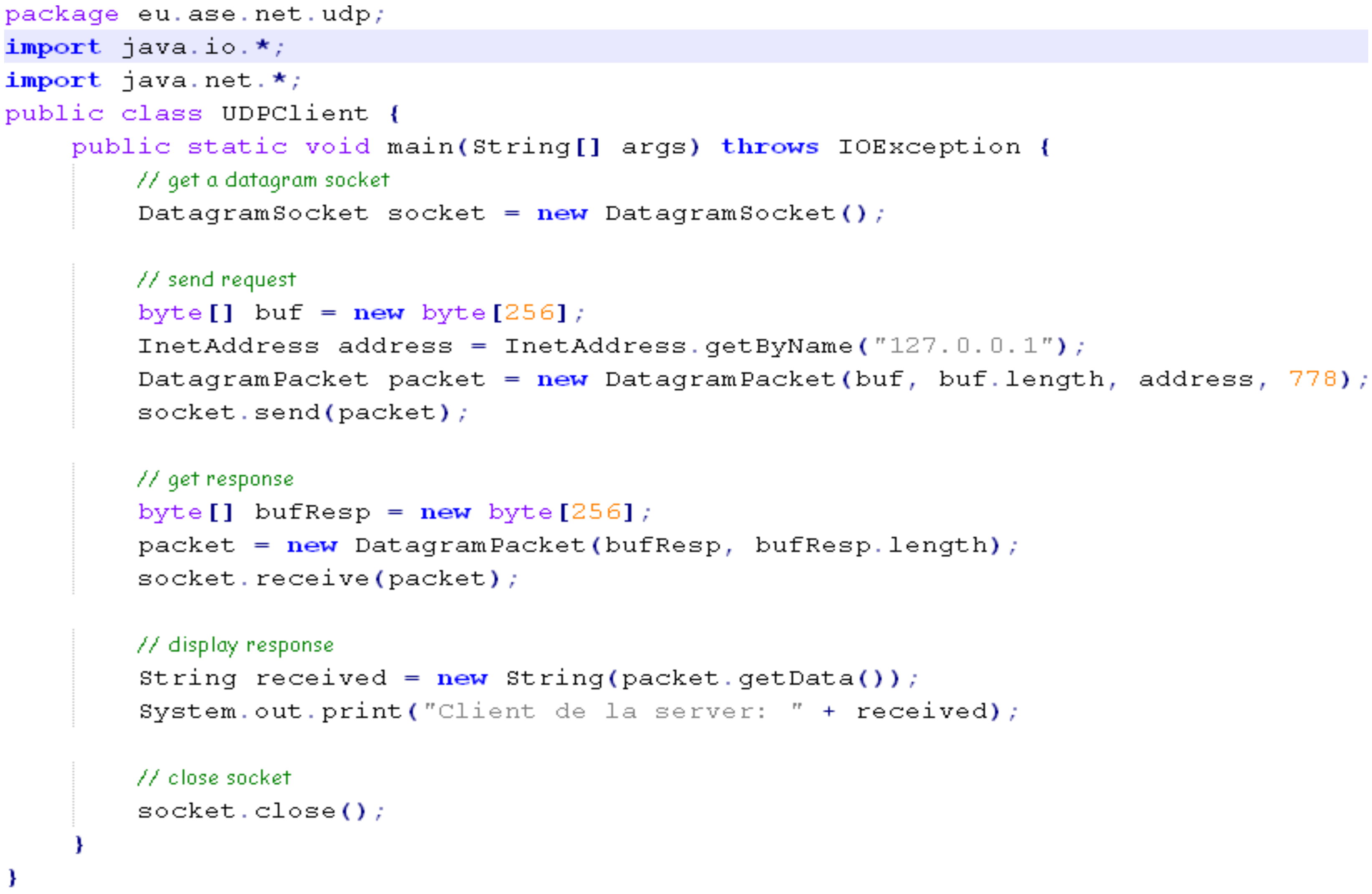
close()

2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:



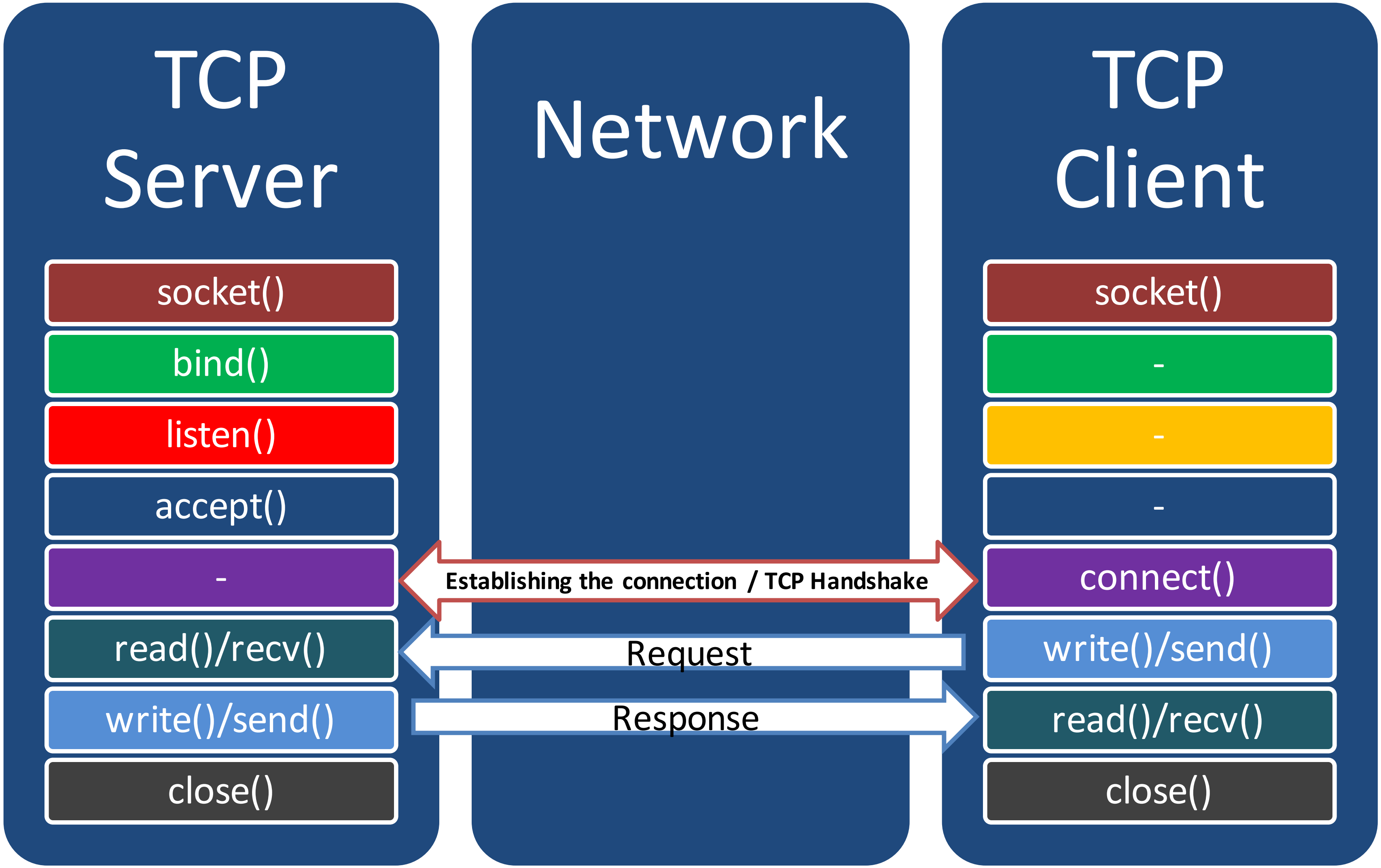
```
package eu.ase.net.udp;
import java.io.*;
import java.net.*;
public class UDPServer {
    public static void main(String[] args) {
        // get a datagram socket
        DatagramSocket socket = null;
        byte[] bufResp = null;
        byte[] bufRecv = null;
        try {
         socket = new 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) {
```

2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:



```java
package eu.ase.net.udp;
import java.io.*;
import java.net.*;
public class UDPClient {
    public static void main(String[] args) throws IOException {
        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();

        // send request
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName("127.0.0.1");
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 778);
        socket.send(packet);

        // get response
        byte[] bufResp = new byte[256];
        packet = new DatagramPacket(bufResp, bufResp.length);
        socket.receive(packet);

        // display response
        String received = new String(packet.getData());
        System.out.print("Client de la server: " + received);

        // close socket
        socket.close();
    }
}
```

2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:



TCP Server	Network	TCP Client
socket()		socket()
bind()		-
listen()		-
accept()		-
-	Establishing the connection / TCP Handshake	connect()
read()/recv()	Request	write()/send()
write()/send()	Response	read()/recv()
close()		close()

2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:

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;}
}
```

in <- outC

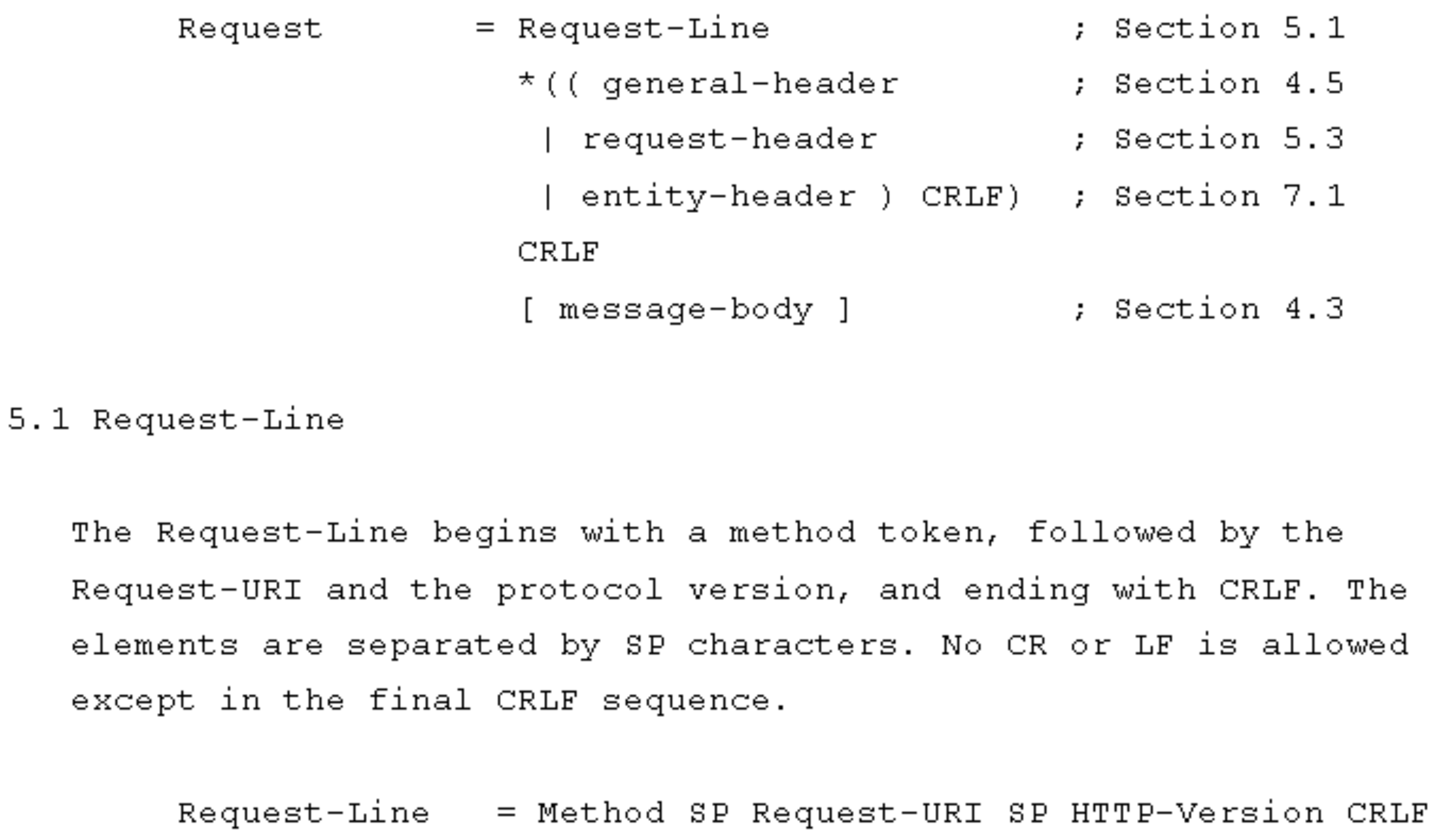
out -> inC

Client

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


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


//STABILIREA CONEXIUNII
//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("Sever: " + lin);
```

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
```

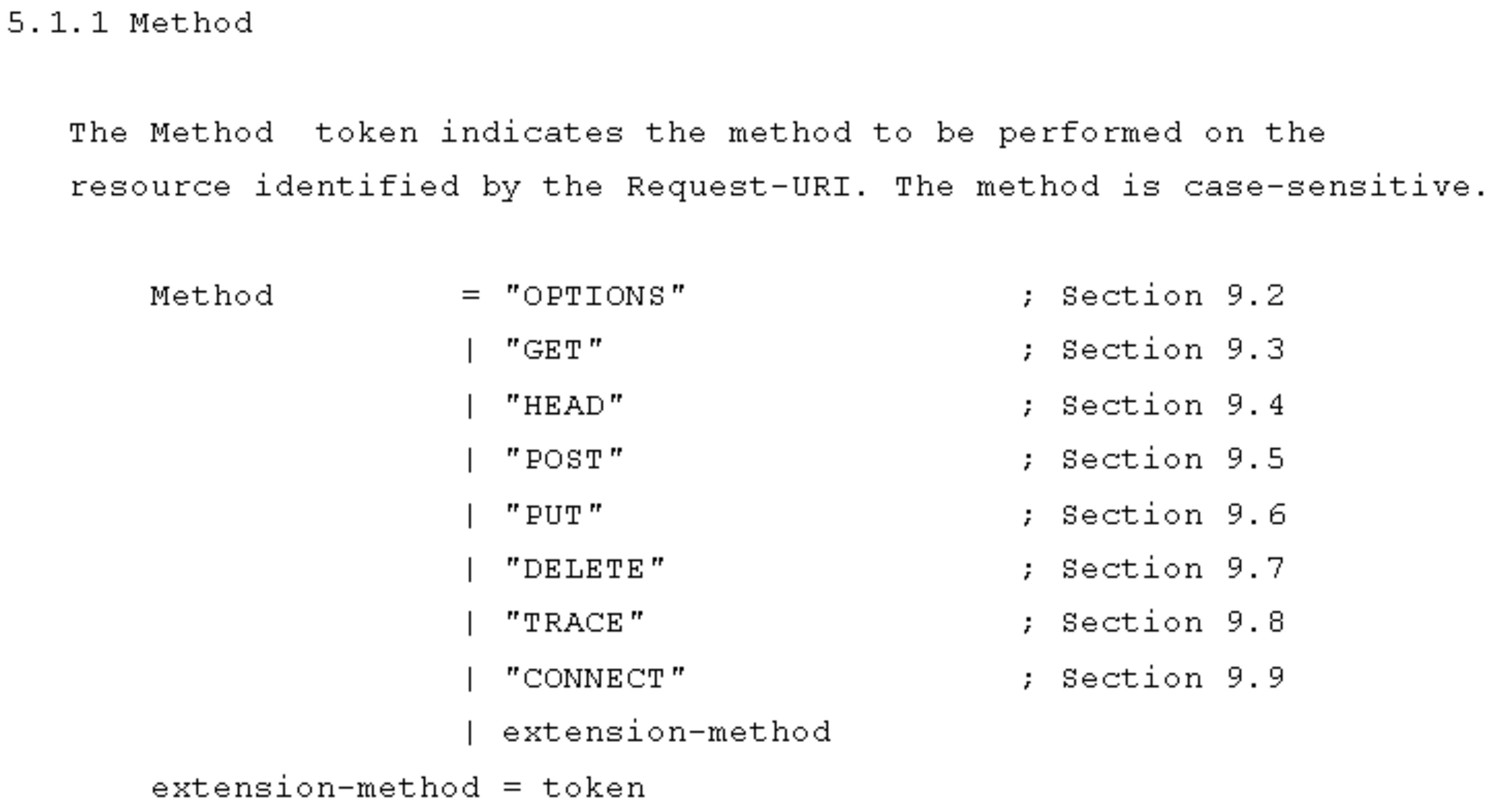
5.1 Request-Line

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

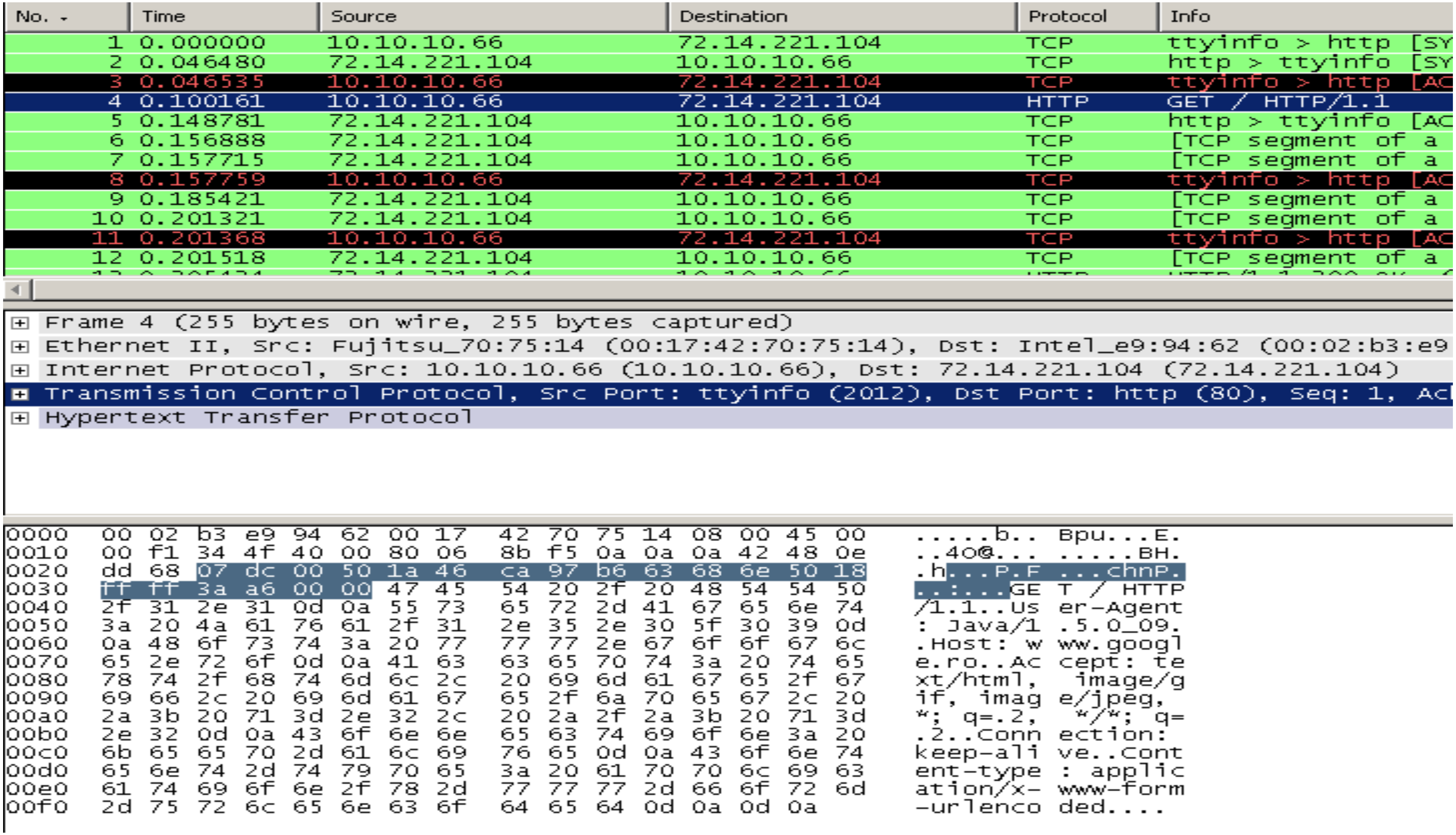
HTTP Programming – RFC 2616:



```
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:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.10.10.66	72.14.221.104	TCP	ttyinfo > http [SY
2	0.046480	72.14.221.104	10.10.10.66	TCP	http > ttyinfo [SY
3	0.046535	10.10.10.66	72.14.221.104	TCP	ttyinfo > http [AC
4	0.100161	10.10.10.66	72.14.221.104	HTTP	GET / HTTP/1.1
5	0.148781	72.14.221.104	10.10.10.66	TCP	http > ttyinfo [AC
6	0.156888	72.14.221.104	10.10.10.66	TCP	[TCP segment of a
7	0.157715	72.14.221.104	10.10.10.66	TCP	[TCP segment of a
8	0.157759	10.10.10.66	72.14.221.104	TCP	ttyinfo > http [AC
9	0.185421	72.14.221.104	10.10.10.66	TCP	[TCP segment of a
10	0.201321	72.14.221.104	10.10.10.66	TCP	[TCP segment of a
11	0.201368	10.10.10.66	72.14.221.104	TCP	ttyinfo > http [AC
12	0.201518	72.14.221.104	10.10.10.66	TCP	[TCP segment of a

```
⊞ Frame 4 (255 bytes on wire, 255 bytes captured)
⊞ Ethernet II, Src: Fujitsu_70:75:14 (00:17:42:70:75:14), Dst: Intel_e9:94:62 (00:02:b3:e9
⊞ Internet Protocol, Src: 10.10.10.66 (10.10.10.66), Dst: 72.14.221.104 (72.14.221.104)
⊞ Transmission Control Protocol, Src Port: ttyinfo (2012), Dst Port: http (80), Seq: 1, Ac
⊞ Hypertext Transfer Protocol
```

```
0000  00 02 b3 e9 94 62 00 17  42 70 75 14 08 00 45 00   .....b.. Bpu...E.
0010  00 f1 34 4f 40 00 80 06  8b f5 0a 0a 0a 42 48 0e   ..4O@... .....BH.
0020  dd 68 07 dc 00 50 1a 46  ca 97 b6 63 68 6e 50 18   .h...P.F ...chnP.
0030  ff ff 3a a6 00 00 47 45  54 20 2f 20 48 54 54 50   ..:...GE T / HTTP
0040  2f 31 2e 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74   /1.1..Us er-Agent
0050  3a 20 4a 61 76 61 2f 31  2e 35 2e 30 5f 30 39 0d   : Java/1 .5.0_09.
0060  0a 48 6f 73 74 3a 20 77  77 77 2e 67 6f 6f 67 6c   .Host: w ww.googl
0070  65 2e 72 6f 0d 0a 41 63  63 65 70 74 3a 20 74 65   e.ro..Ac cept: te
0080  78 74 2f 68 74 6d 6c 2c  20 69 6d 61 67 65 2f 67   xt/html,  image/g
0090  69 66 2c 20 69 6d 61 67  65 2f 6a 70 65 67 2c 20   if, imag e/jpeg,
00a0  2a 3b 20 71 3d 2e 32 2c  20 2a 2f 2a 3b 20 71 3d   *; q=.2,  */*; q=
00b0  2e 32 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20   .2..Conn ection:
00c0  6b 65 65 70 2d 61 6c 69  76 65 0d 0a 43 6f 6e 74   keep-ali ve..Cont
00d0  65 6e 74 2d 74 79 70 65  3a 20 61 70 70 6c 69 63   ent-type : applic
00e0  61 74 69 6f 6e 2f 78 2d  77 77 77 2d 66 6f 72 6d   ation/x- www-form
00f0  2d 75 72 6c 65 6e 63 6f  64 65 64 0d 0a 0d 0a      -urlenco ded....
```

Section Conclusions

Java Network Programming uses for UDP:

DatagramSocket and DatagramPacket classes on both server and client side.

Java Network Programming uses for TCP:

ServerSocket and Socket classes on server side.

Only Socket class on client side.

For both server and client, it is necessary to create byte/char Input (socket.getInputStream()) and Output (socket.getOutputStream()) streams between the Random Access Memory – RAM and the network communications channel.

Java Network Programming

for easy sharing

3

Network Programming & Java Sockets

Communicate & Exchange Ideas

Questions & Answers!

But wait…

There’s More!

Thanks!

Java SE
End of Lecture 10 – Summary of Java SE &
Network Programming