Lecture 2
summary of Java SE & Network Programming – section 2

presentation

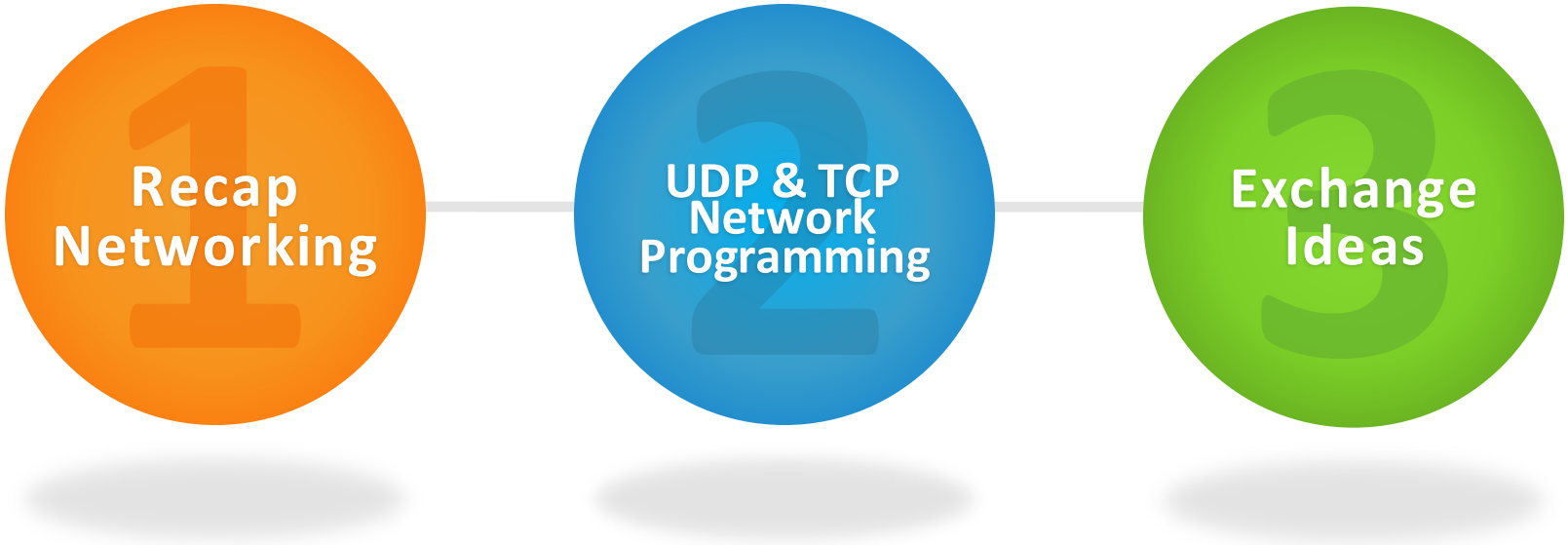
DAD – Distributed Applications Development

Cristian Toma

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

Cristian Toma – Business Card

Agenda for Lecture 2

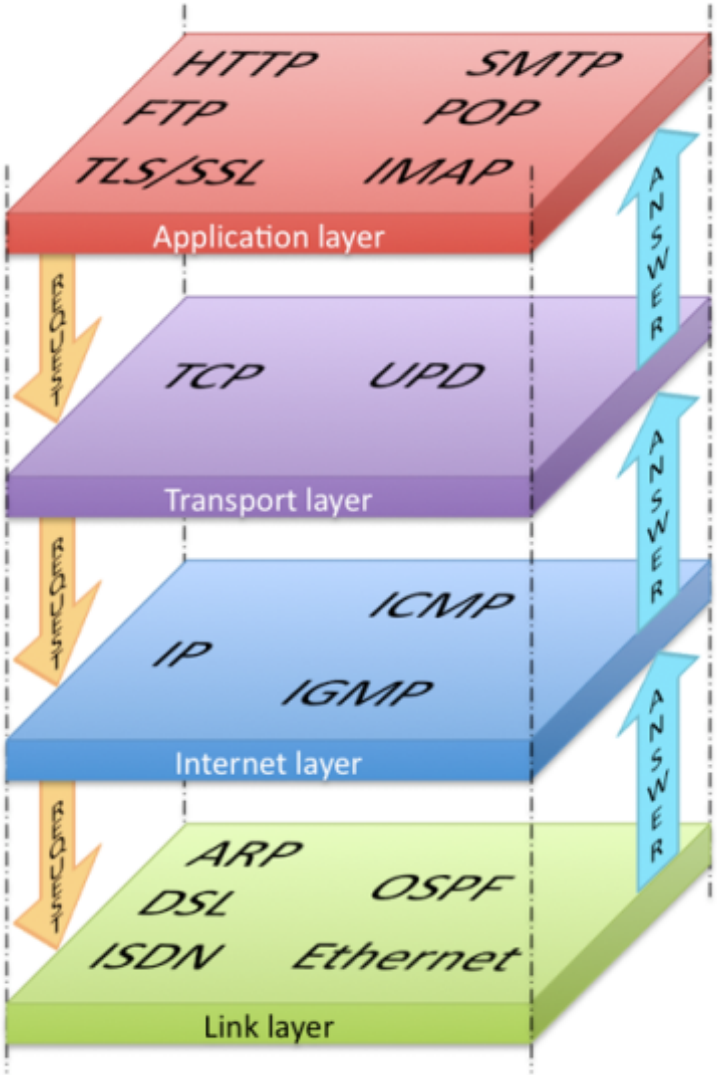


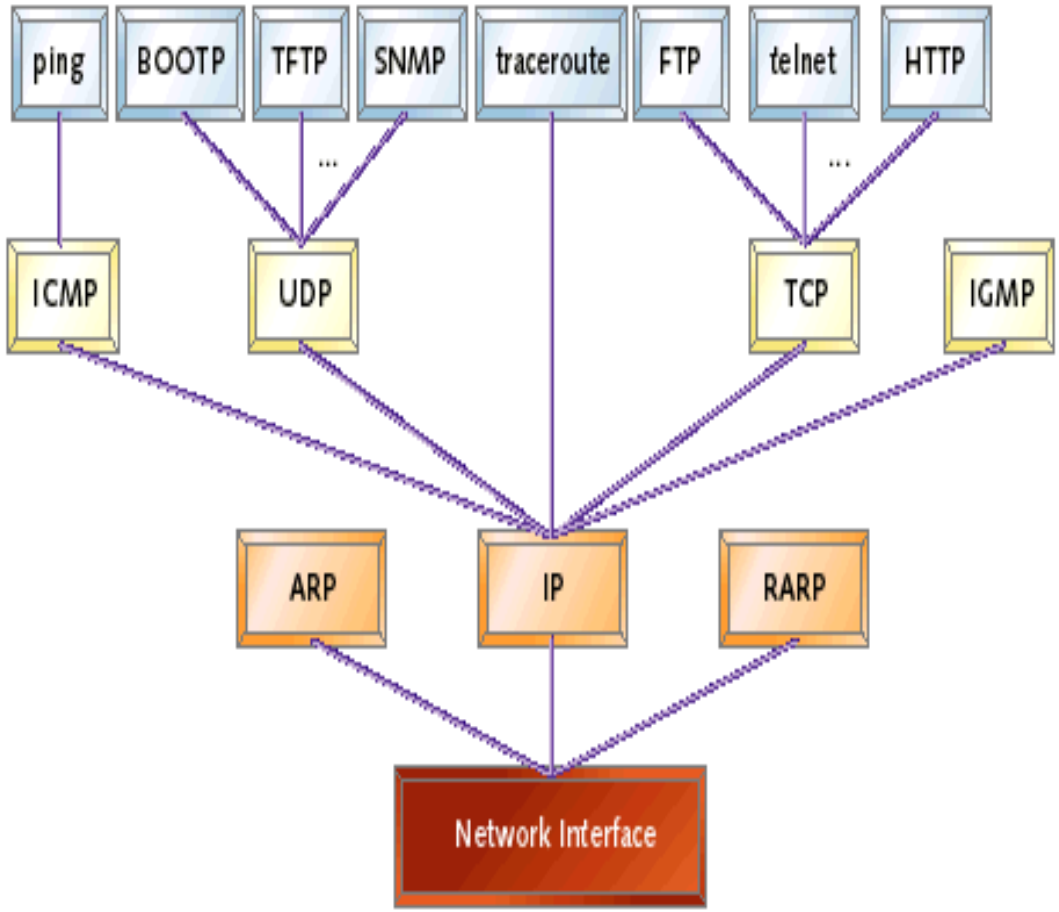
Networking IP, UDP and TCP programming, TCP/IP state machine, SNMP, SMTP

Networking Recapitulation

1. Networking TCP/IP Stack

HOW TCP/IP Works:



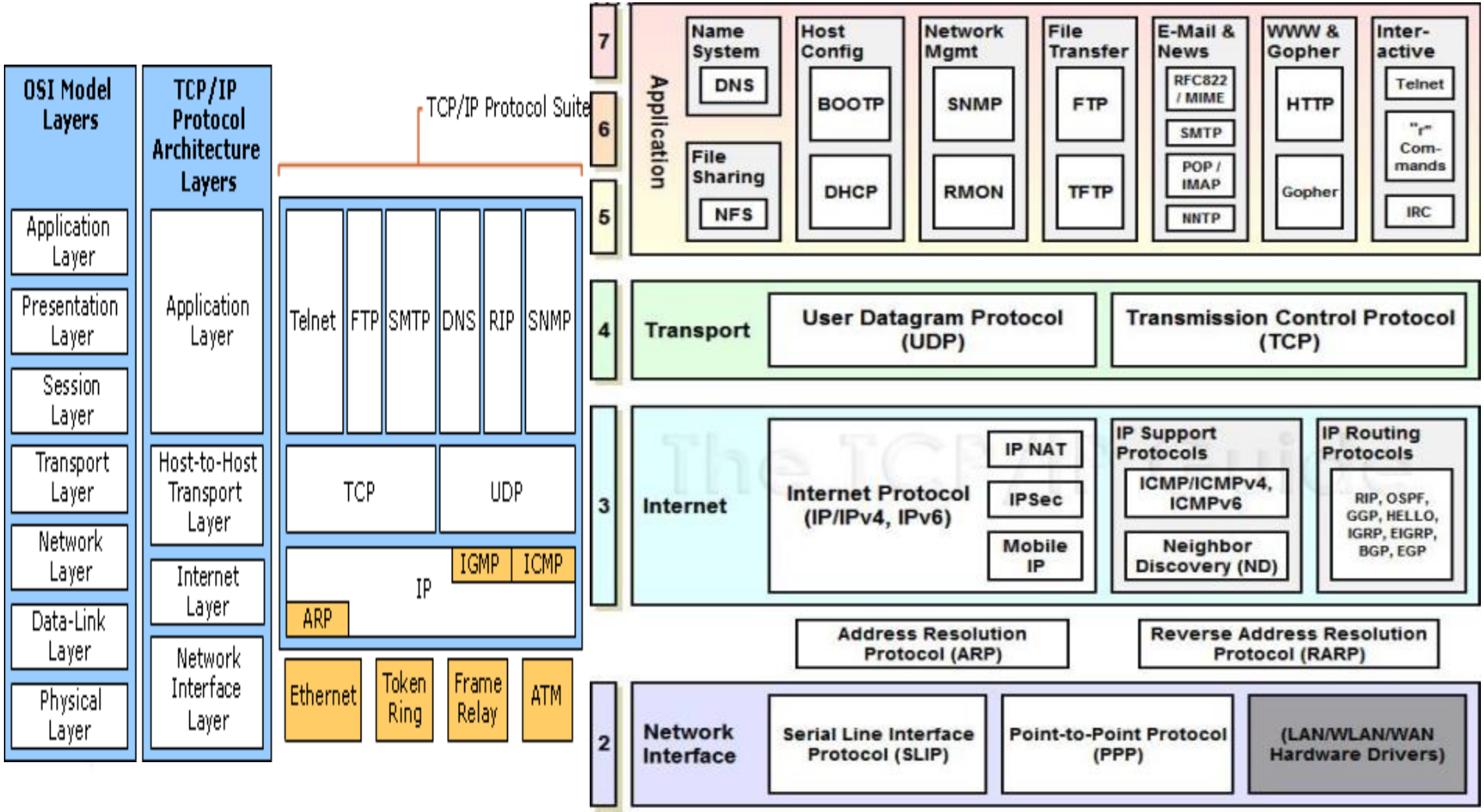


http://buildingautomationmonthly.com/tcpip-an-overview/

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

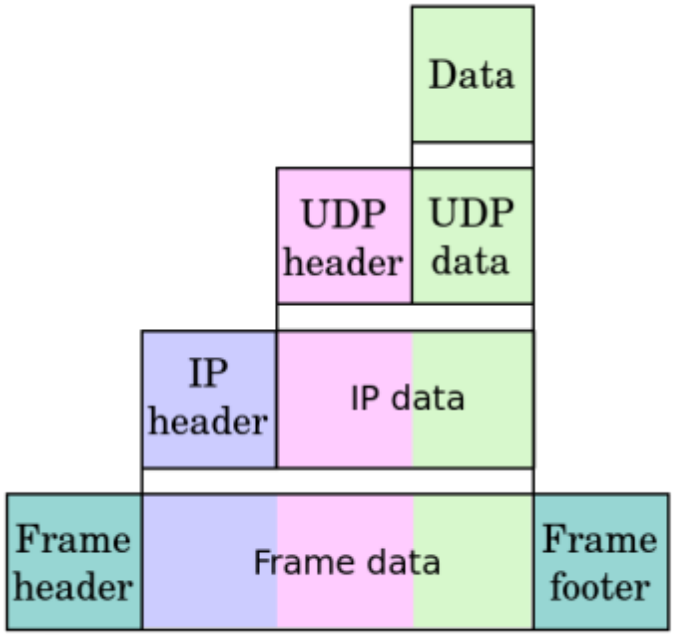
ISO/OSI Model vs. TCP/IP:

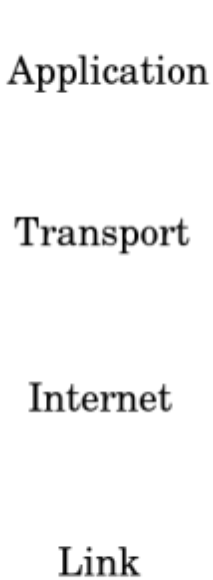
TCP/IP DoD Model			OSI Model
Application Layer (Services Layers 5,6,7) **PDU: Data**	HTTP: port **80** HTTPS/TLS/SSL: port **443** NNTP: port **119** FTP: port **21, 20** Telnet: port **23** SSH: port **22** POP3: port **110** IMAP4: port **143** SMTP: port **25**	DNS: port **53** TFTP: port **69** DHCP/BootP: port **67,68** SNMP: port **162, 161** NTP: port **123** Syslog: port **514**	Application Layer (7) **Scribe.** APIs, network services Serves the King/User Presentation Layer (6) **Translator.** Reformats, encrypts/de-crypts, compress/de-compress 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.

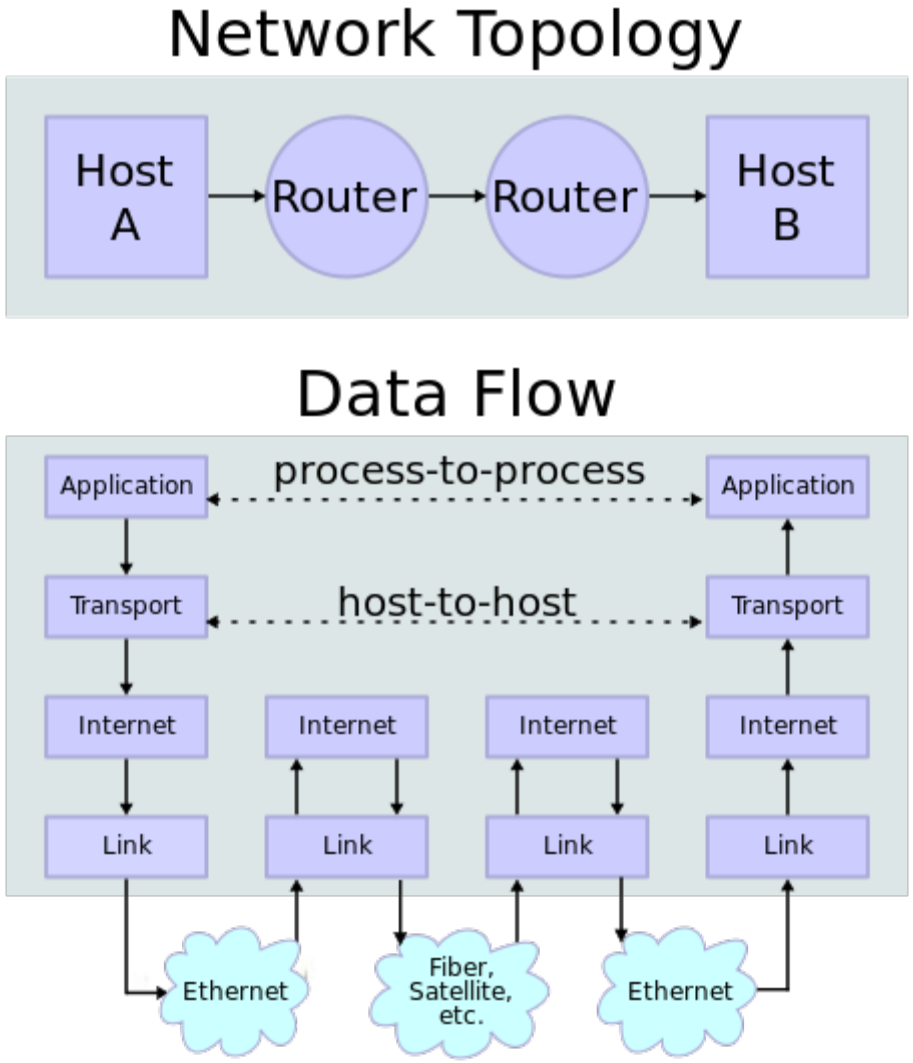
http://buildingautomationmonthly.com/tcpip-an-overview/

1. Networking TCP/IP Stack

TCP/IP Message Encapsulation:



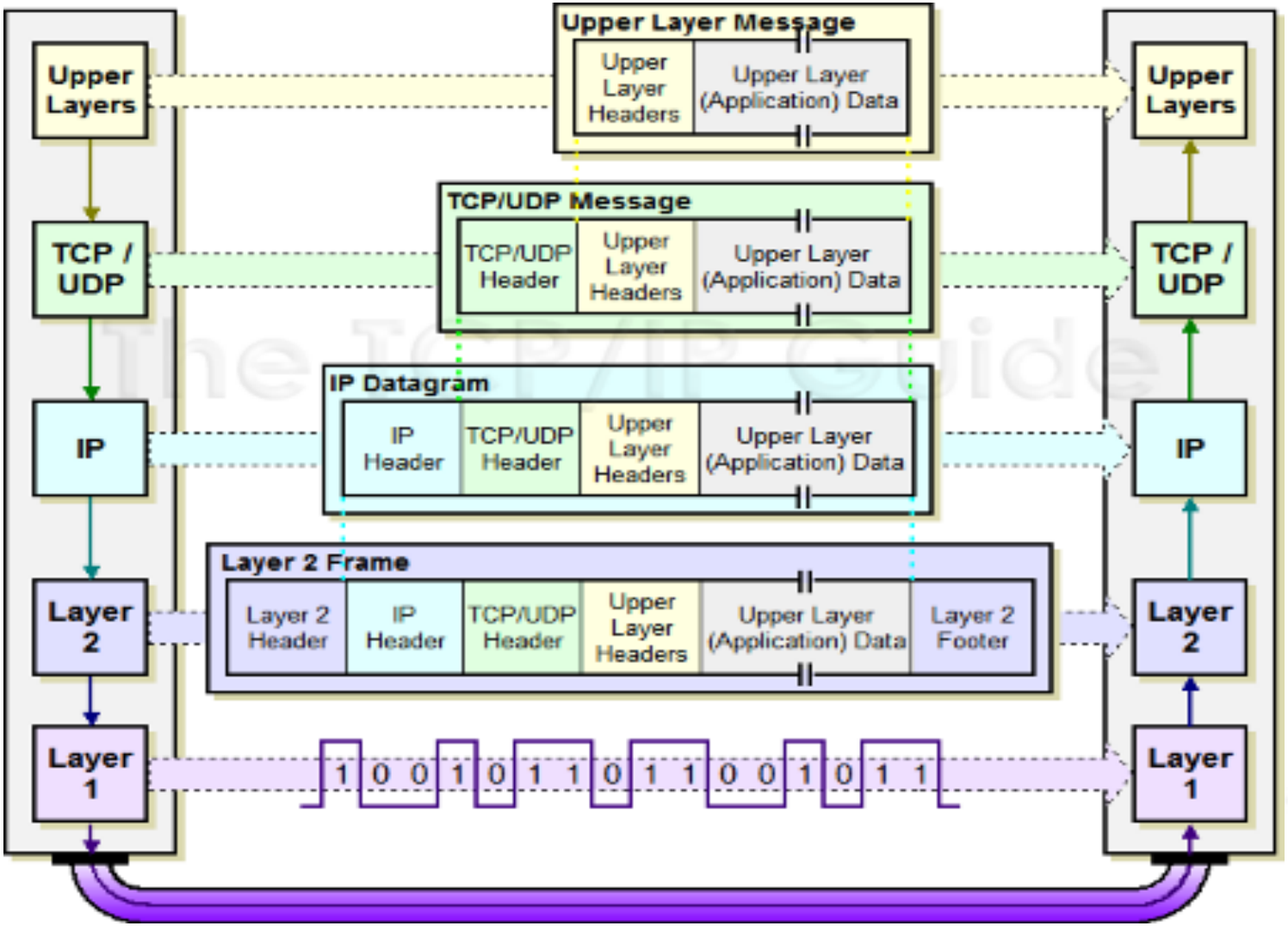




http://en.wikipedia.org/wiki/Internet_protocol_suite

1. Networking TCP/IP Stack

TCP/IP Message Flow:

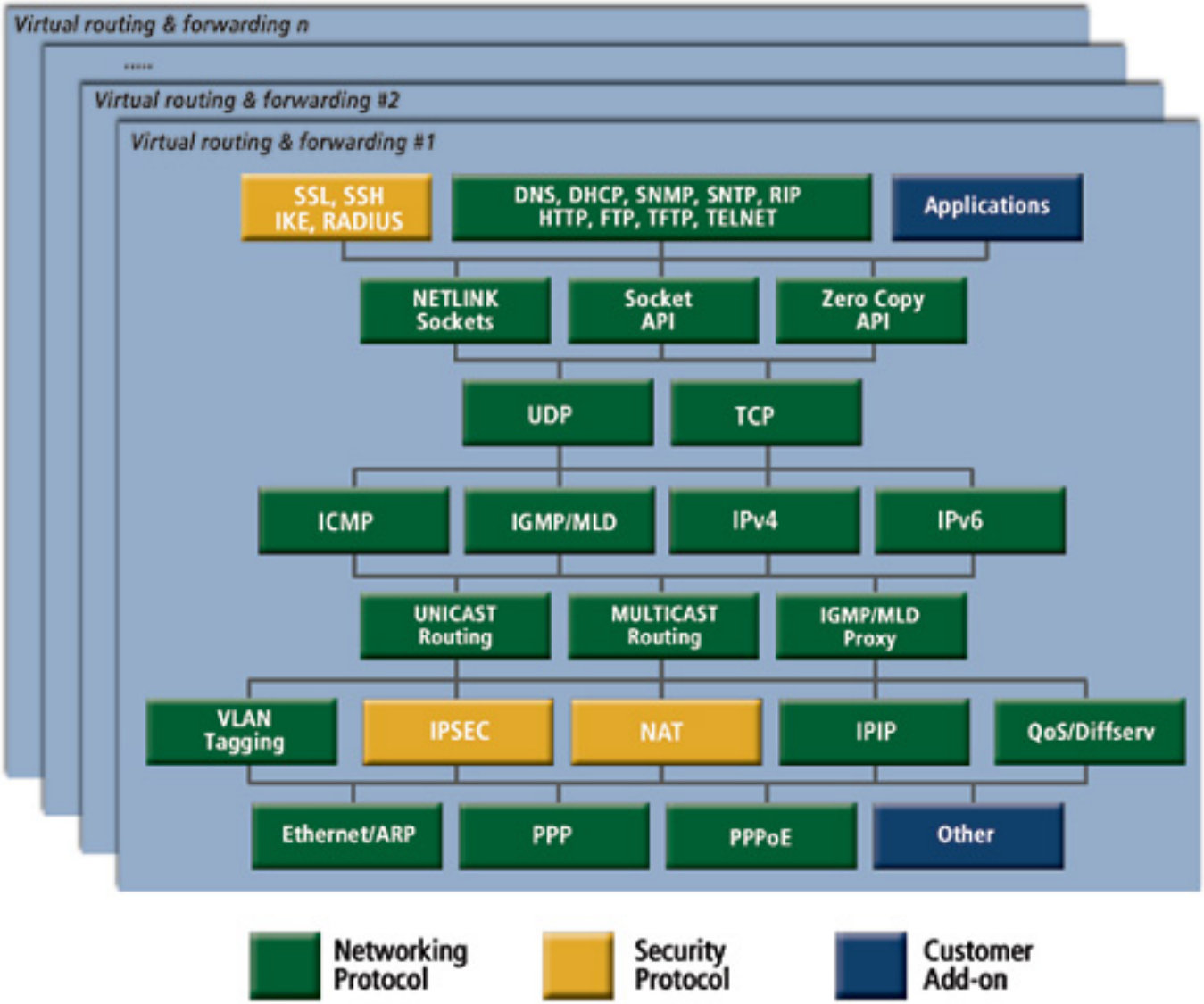


http://buildingautomationmonthly.com/tcpip-an-overview/

1. Networking TCP/IP Stack

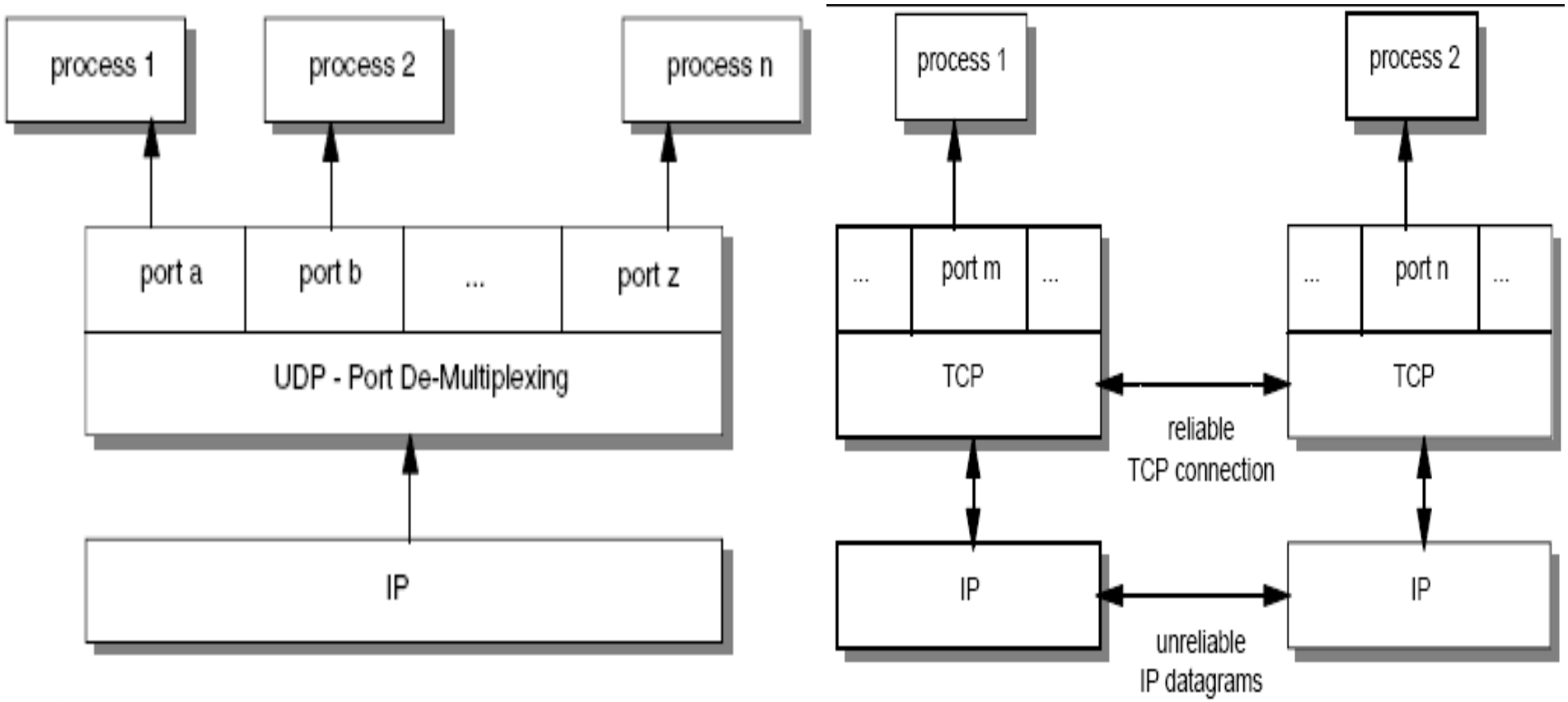
TCP/IP Virtual LAN/Networks:

http://www.ghs.com/products/comm_tcp-ip.html



1. Networking TCP/IP Stack

TCP/IP App/Port Multiplexing:

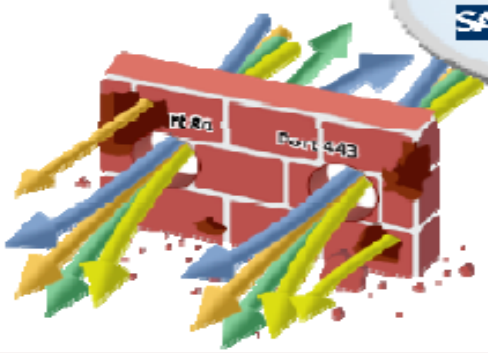


1. Networking TCP/IP Stack

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

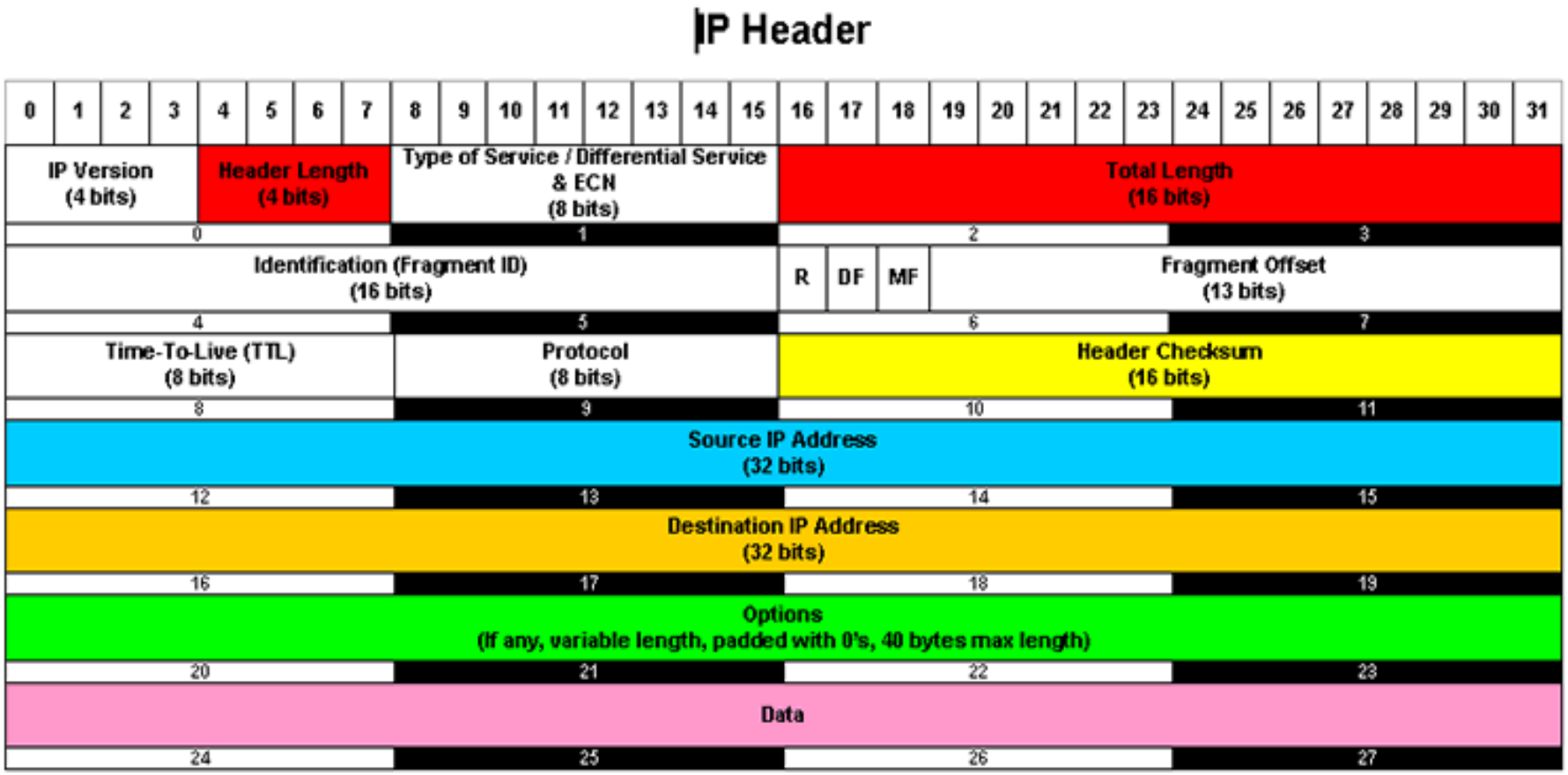


- BUT…Applications Have Changed
  - Ports ≠Applications
  - IP Addresses ≠Users
  - Packets ≠Content

Need to Restore Visibility and Control in the Firewall

1. Networking TCP/IP Stack

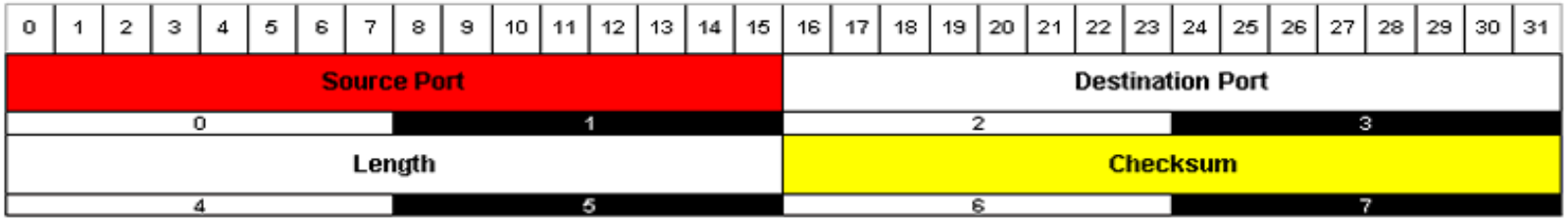
IP Header - RFC 791 – Submasking + Routing + NAT:



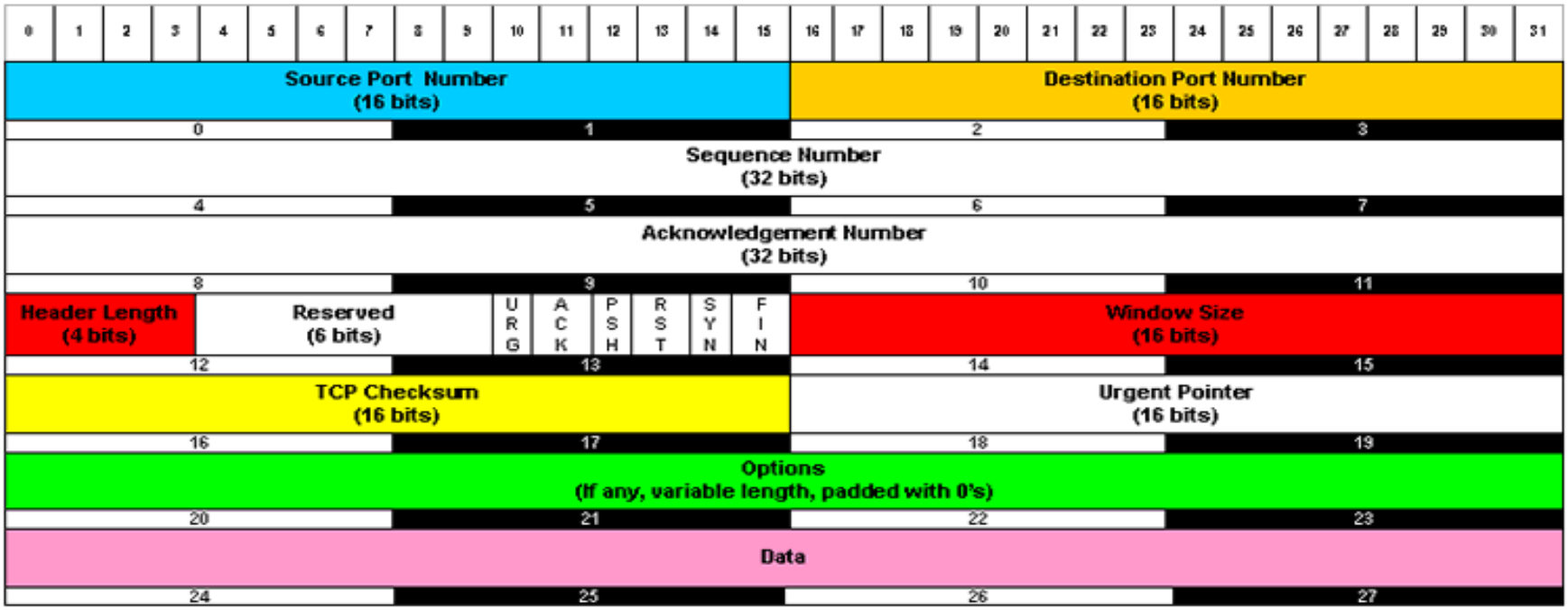
1. Networking TCP/IP Stack

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

UDP Header

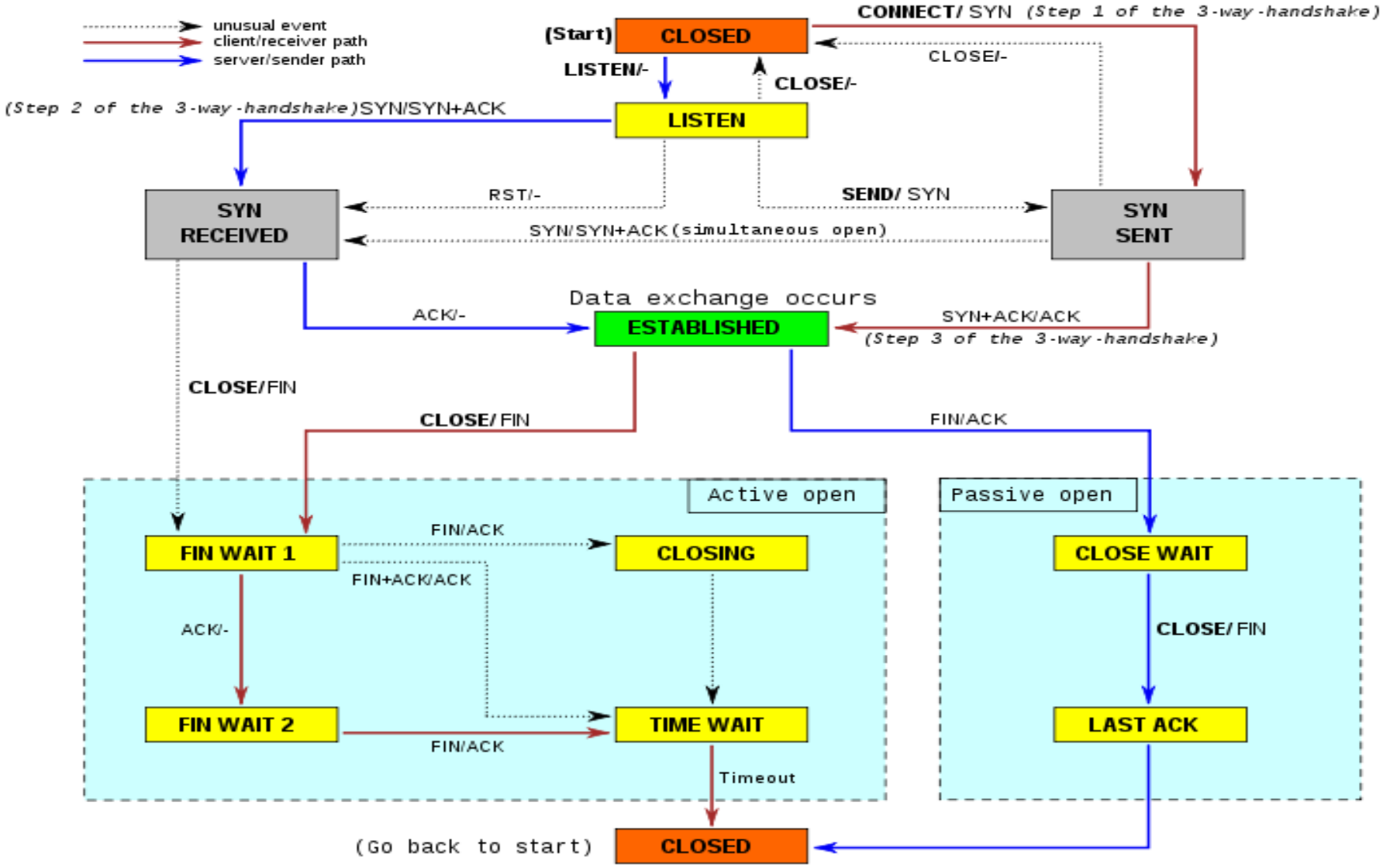


TCP Header



1. Networking TCP/IP Stack

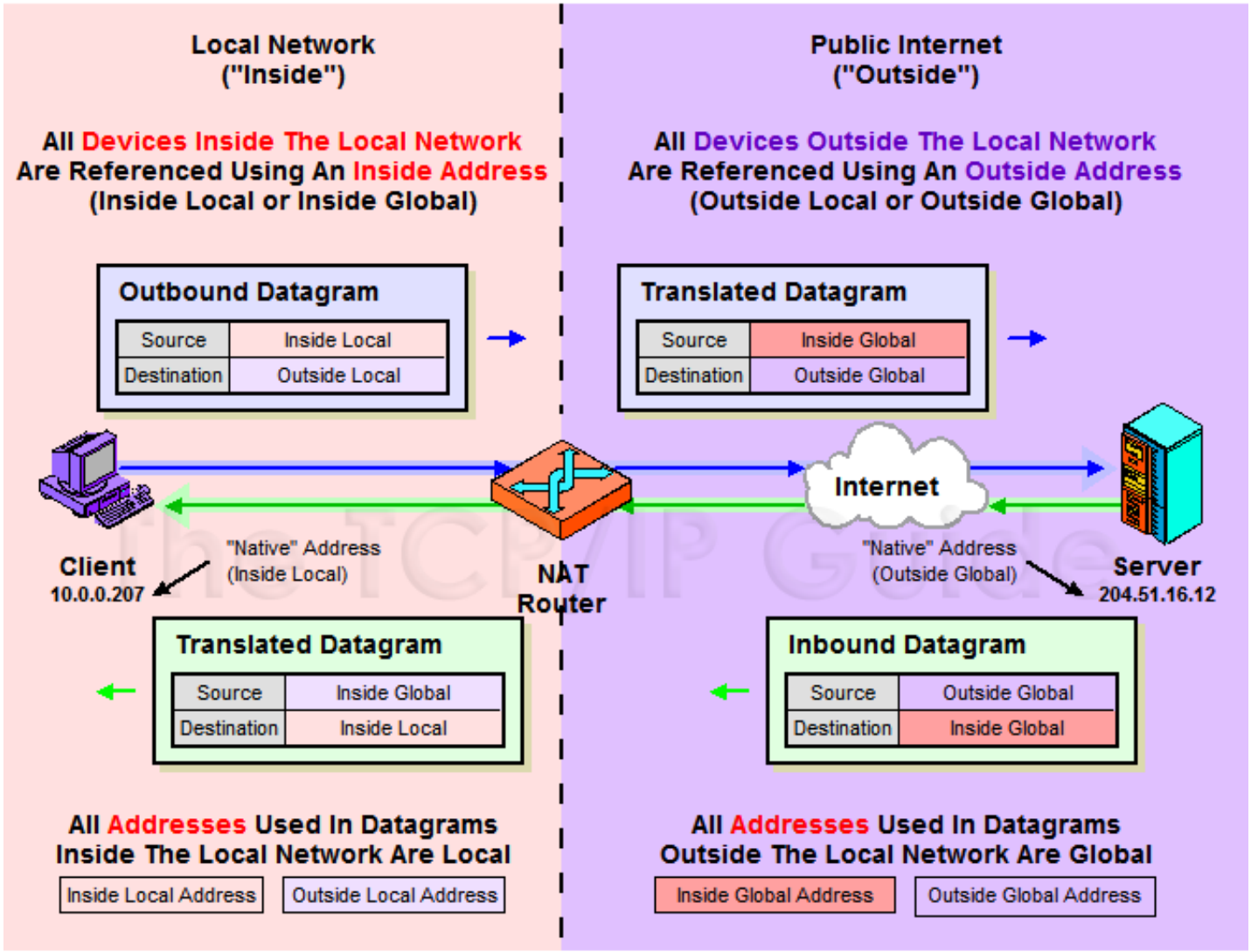
TCP State Machine – RFC 793:



1. Networking TCP/IP Stack

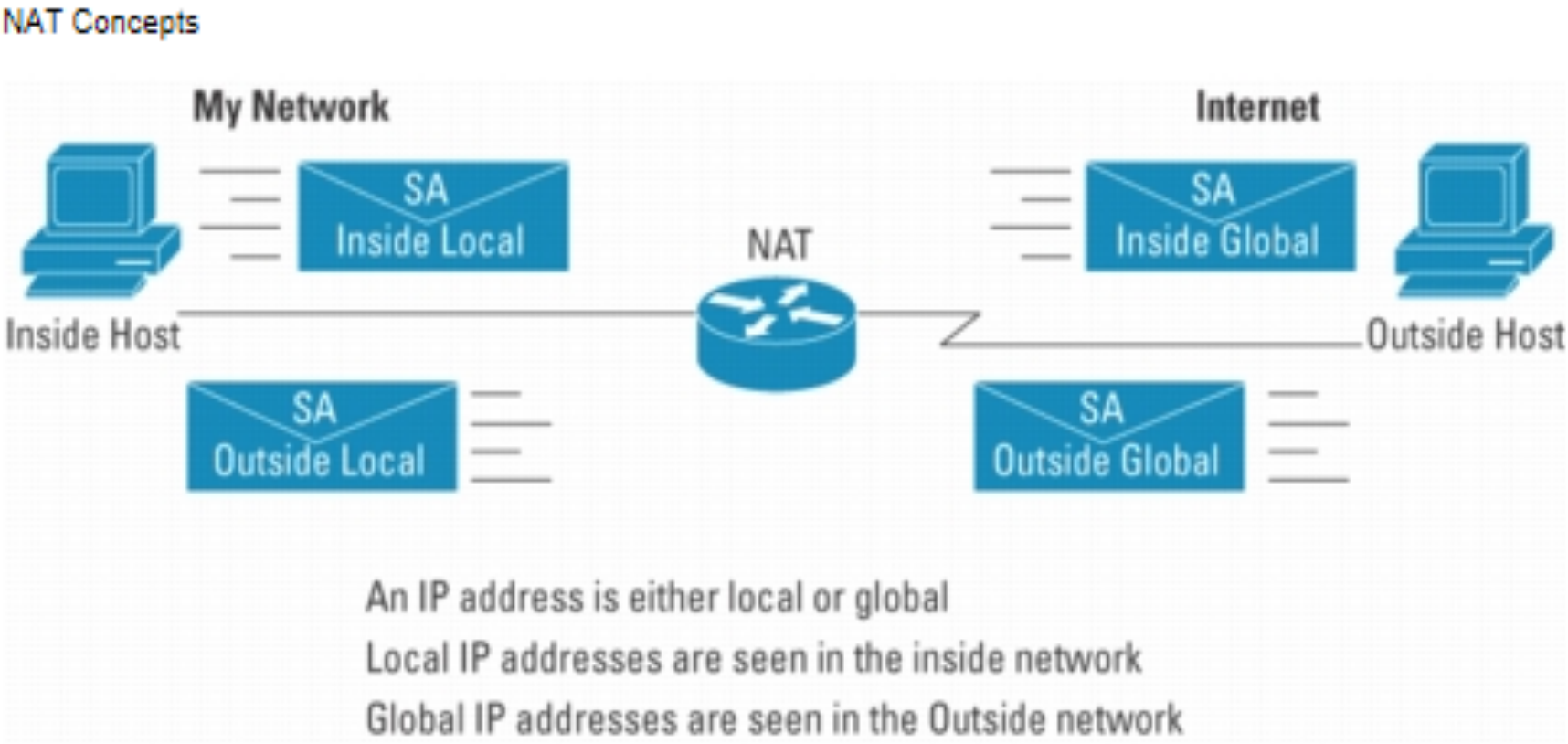
TCP/IP NAT & PAT:

http://www.tcpipguide.com/free/t_IPNATAddressTerminology-3.htm



1. Networking TCP/IP Stack

TCP/IP NAT:

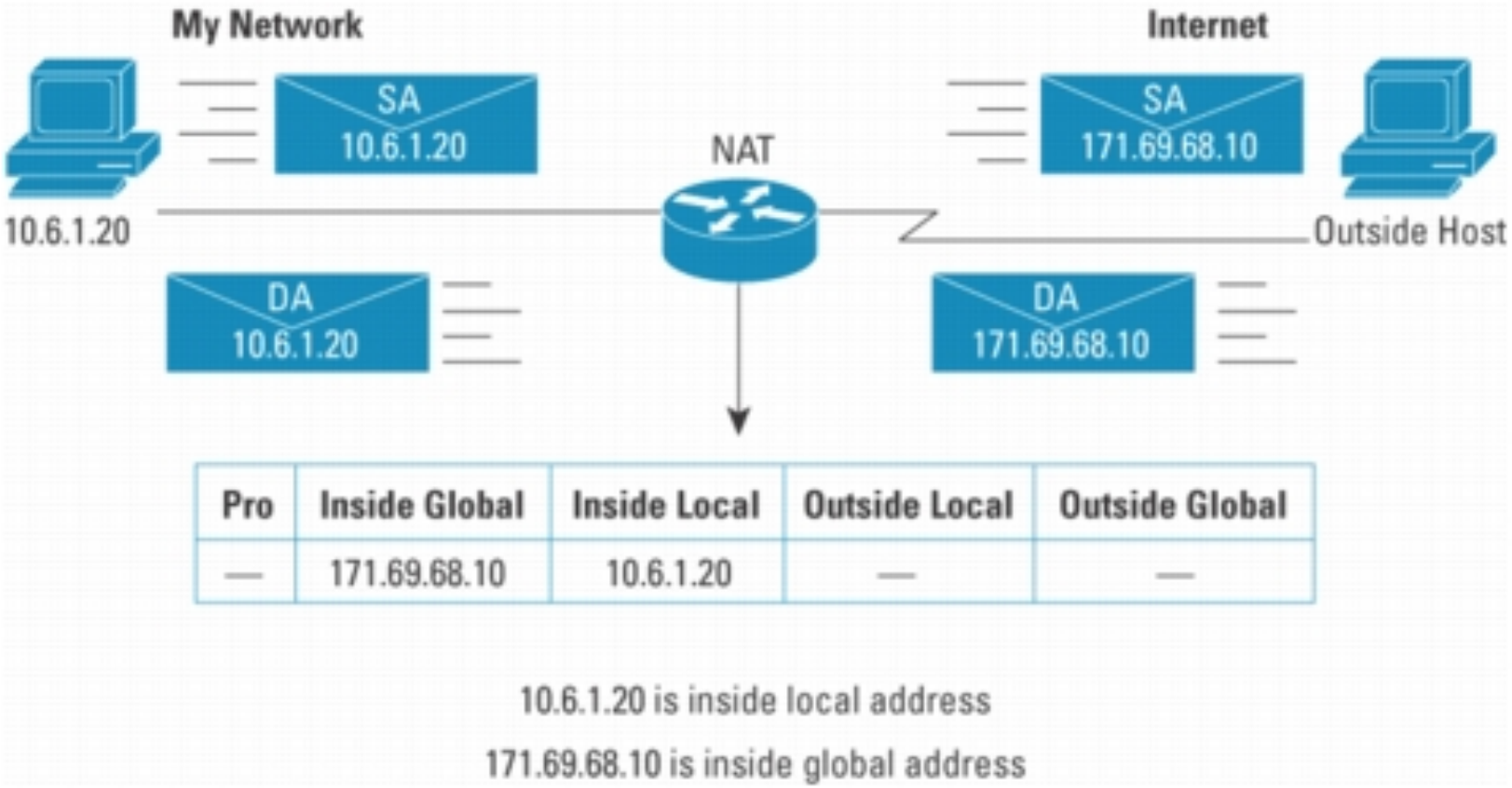


http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP NAT:

NAT Terminology « Inside Addressing»



Pro	Inside Global	Inside Local	Outside Local	Outside Global
—	171.69.68.10	10.6.1.20	—	—

10.6.1.20 is inside local address

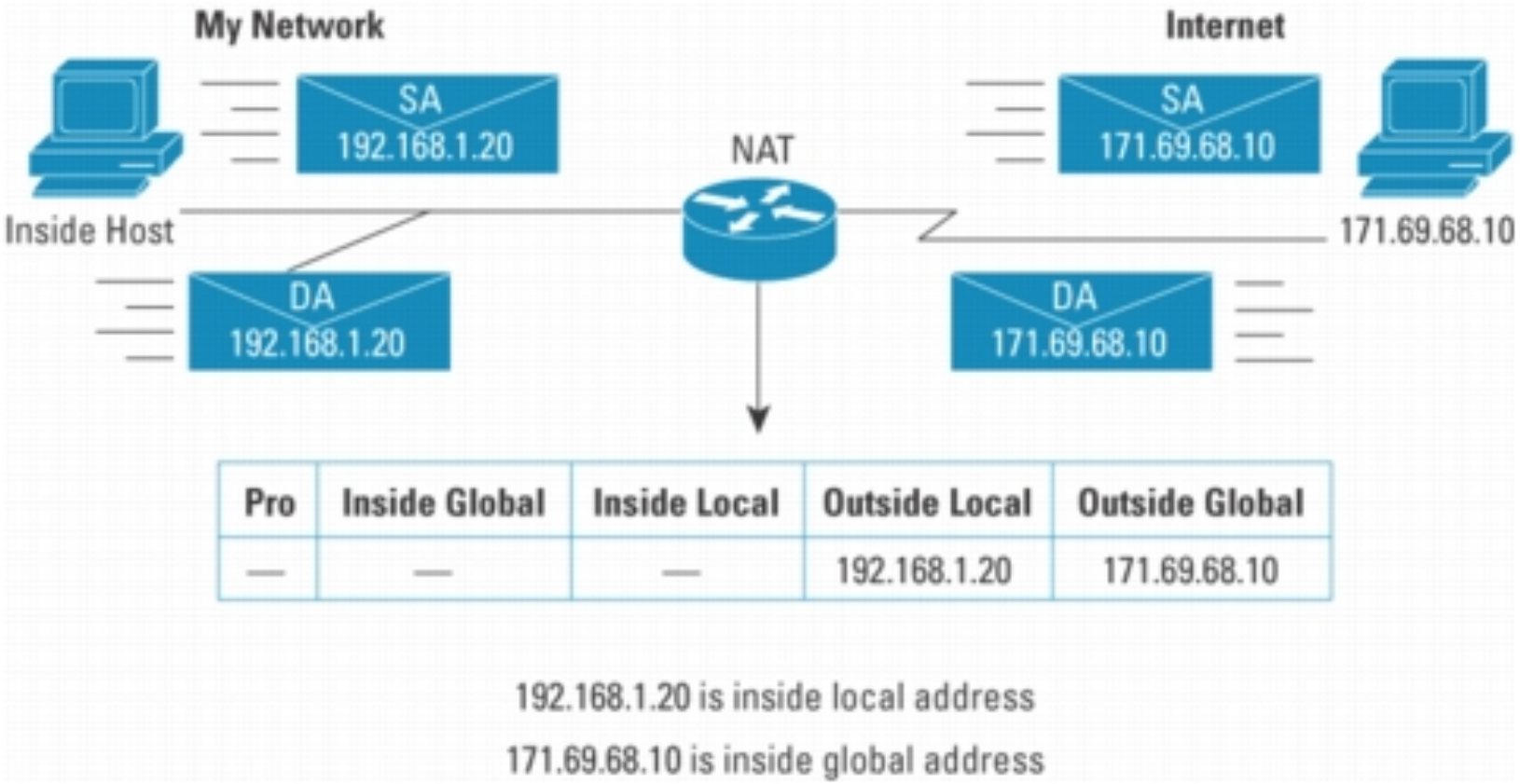
171.69.68.10 is inside global address

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP NAT:

NAT Terminology "Outside Addressing"



Pro	Inside Global	Inside Local	Outside Local	Outside Global
—	—	—	192.168.1.20	171.69.68.10

192.168.1.20 is inside local address

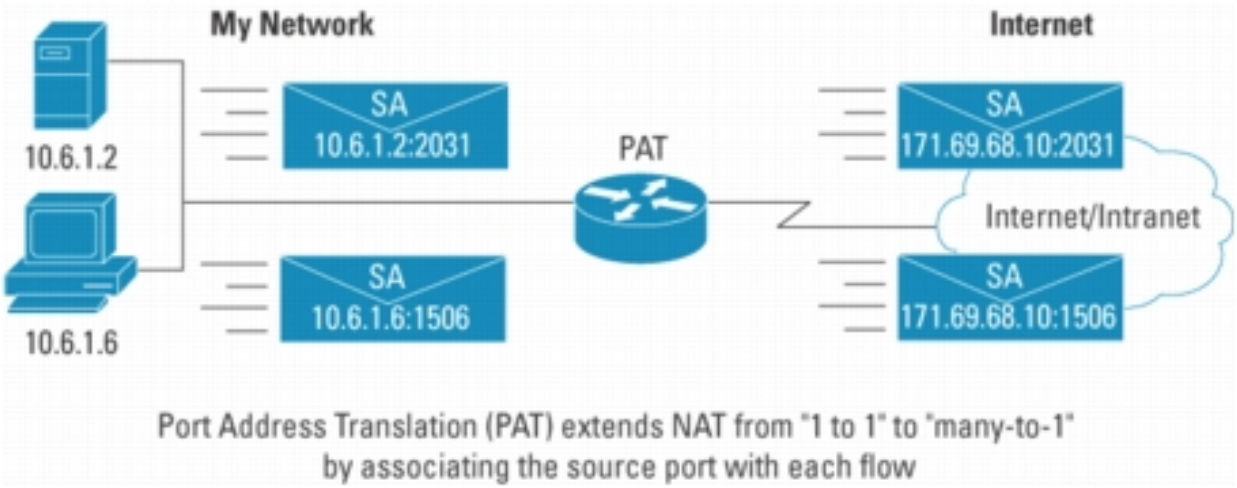
171.69.68.10 is inside global address

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP PAT:

Basic Concepts of PAT



Port Address Translation (PAT) extends NAT from "1 to 1" to "many-to-1" by associating the source port with each flow

Figure 5
Unique Source Port per Translation Entry

Pro	Inside Global	Inside Local	Outside Local	Outside Global
tcp	171.69.68.5:1405	10.6.15.2:1405	204.71.200.69:80	204.71.200.69:80

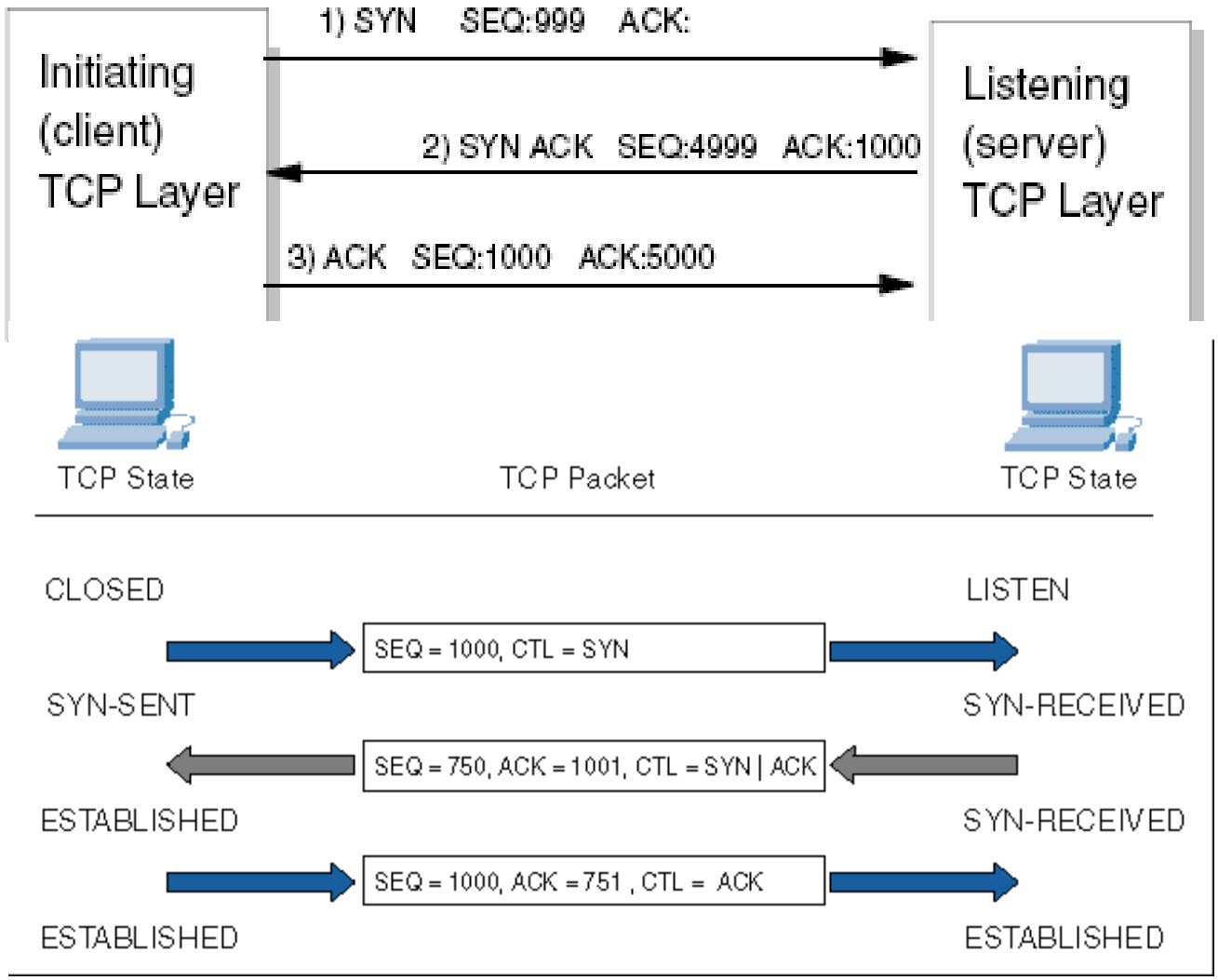
PAT (Port Address Translation) includes ports in addition to IP addresses

- Many-to-one translation
- Maps multiple IP addresses to 1 or a few IP addresses
- Unique source port number identifies each session
- Conserves registered IP addresses
- Also called NAPT in IETF documents

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

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

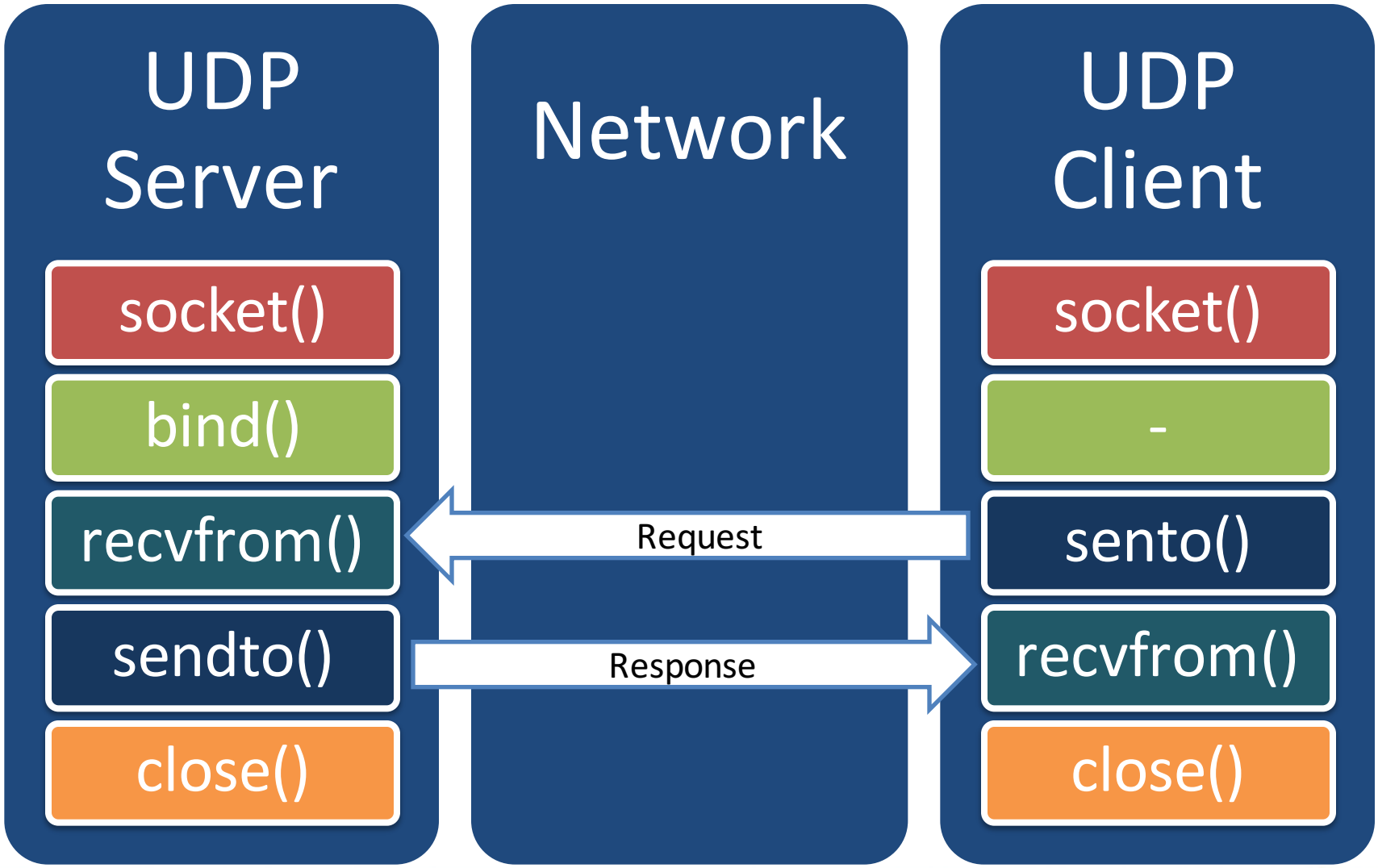
Fact: DAD needs Networking

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.

UDP Client-Server programming, SNMP client sample, TCP Client-Server programming, SMTP client sample

Network UDP & TCP Programming

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



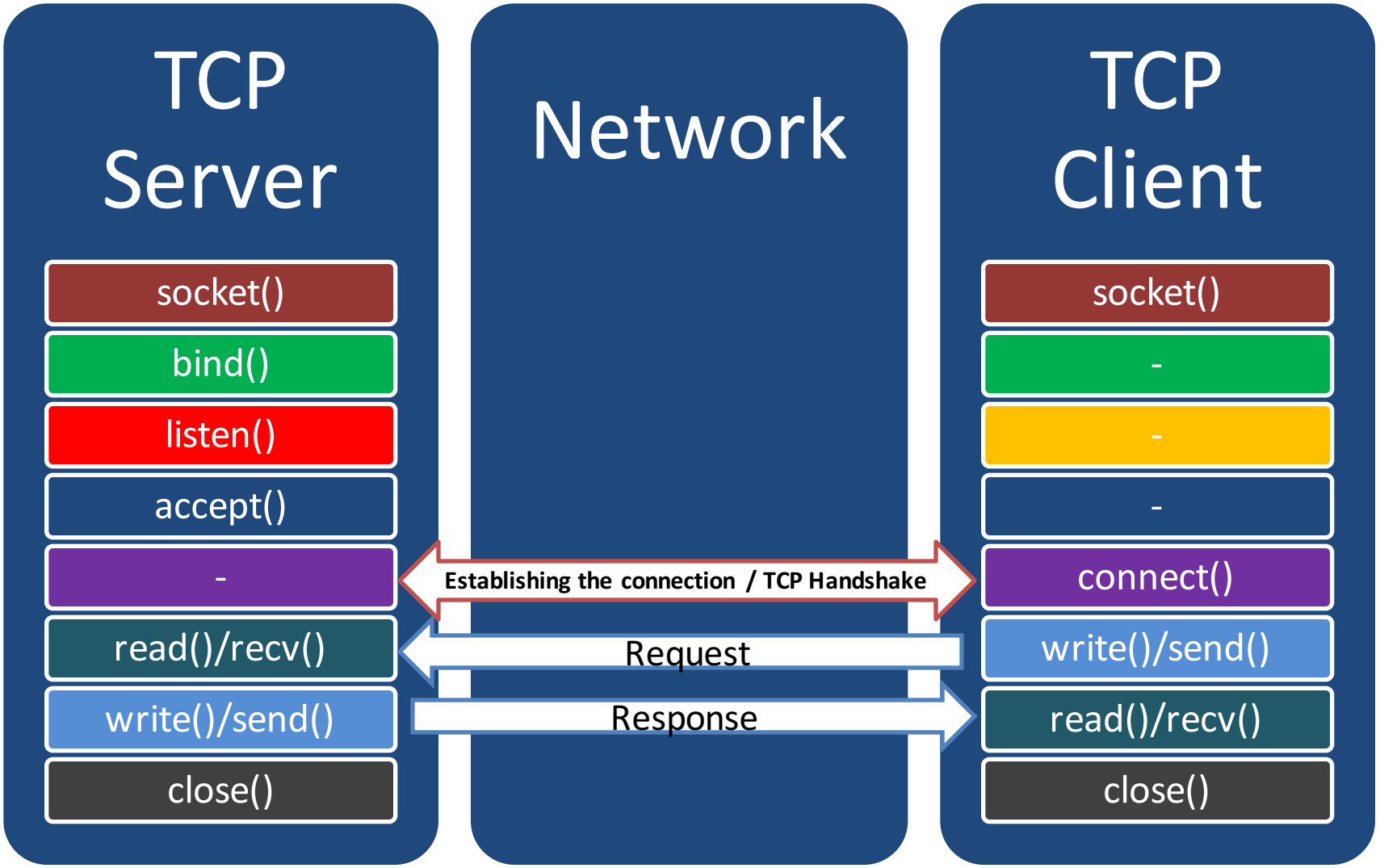
2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

```
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:



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

Server

Client

```
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;}
 }
```

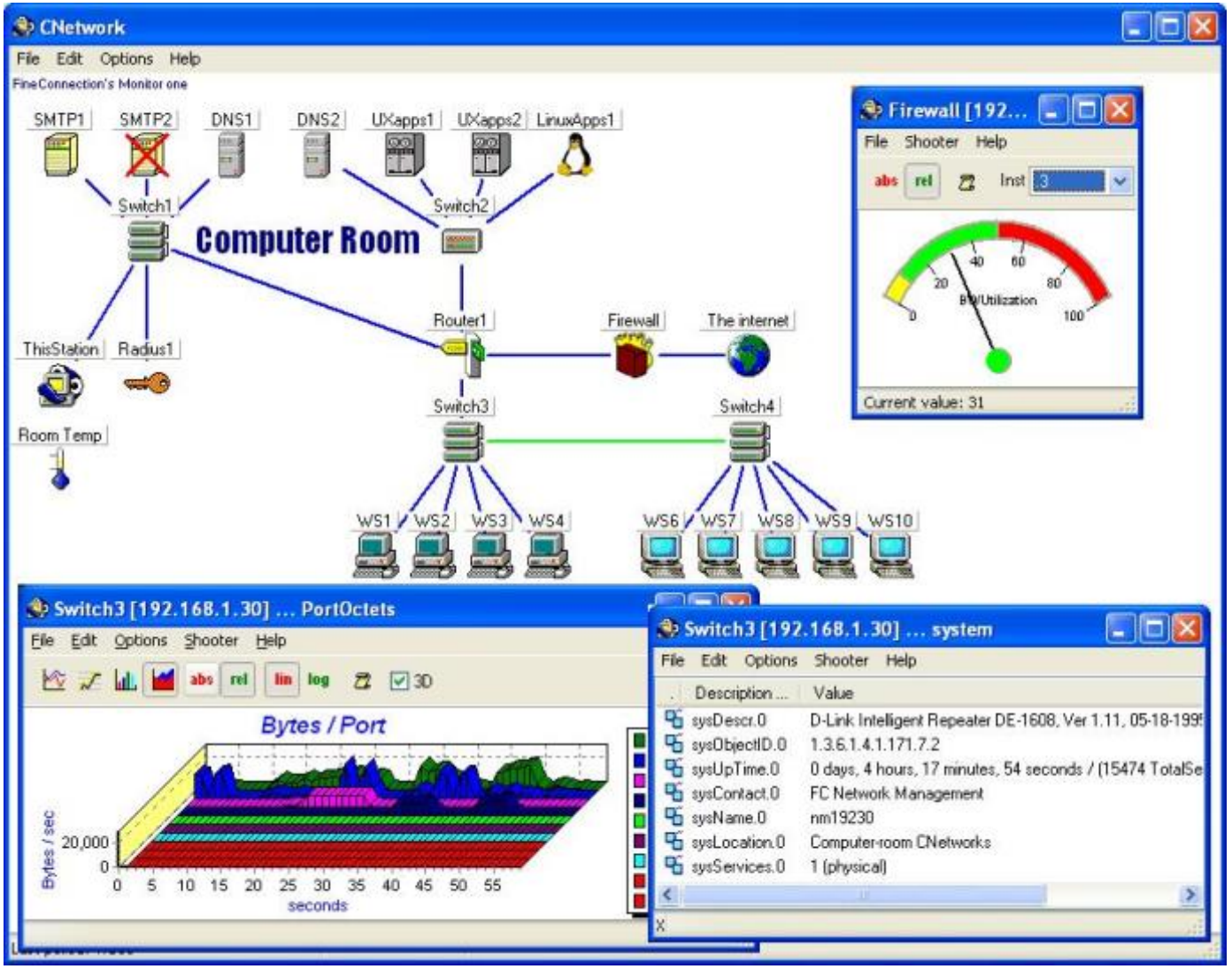
in <- outC

out -> inC

```
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);
```

2.3 SNMP – RFC 1157

What is the necessity for SNMP – Simple? Network Management Protocol?



http://www.filedudes.com/images/screenshots/27179.jpg

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

```
RFC1157-SNMP DEFINITIONS ::= BEGIN

    IMPORTS
        ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
            FROM RFC1155-SMI;

        -- top-level message

        Message ::=
                SEQUENCE {
                    version          -- version-1 for this RFC
                        INTEGER {
                            version-1(0)
                        },

                    community        -- community name
                        OCTET STRING,

                    data             -- e.g., PDUs if trivial
                        ANY          -- authentication is being used
                }

        -- protocol data units

        PDUs ::=
                CHOICE {
                            get-request
                                GetRequest-PDU,

                            get-next-request
                                GetNextRequest-PDU,

                            get-response
                                GetResponse-PDU,

                            set-request
                                SetRequest-PDU,

                            trap
                                Trap-PDU
                        }

        -- PDUs

        GetRequest-PDU ::=
            [0]
                IMPLICIT PDU

        GetNextRequest-PDU ::=
            [1]
                IMPLICIT PDU

        GetResponse-PDU ::=
            [2]
                IMPLICIT PDU

        SetRequest-PDU ::=
            [3]
                IMPLICIT PDU

        PDU ::=
                SEQUENCE {
                   request-id
                        INTEGER,

                    error-status      -- sometimes ignored
                        INTEGER {
                            noError(0),
                            tooBig(1),
                            noSuchName(2),
                            badValue(3),
```

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

Some of the **data types** used in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
BOOLEAN	Primitive	0x01	Boolean value: yes/no
INTEGER	Primitive	0x02	Negative and positive integers
BIT STRING	Primitive	0x03	Bit sequence
OCTET STRING	Primitive	0x04	Byte sequence (1 byte=8 bits=1octet)
NULL	Primitive	0x05	A null value
OBJECT IDENTIFIER	Primitive	0x06	An object identifier, which is a sequence of integer components that identify an object such as an algorithm or attribute type
PrintableString	Primitive	0x13	An arbitrary string of printable characters
T61String	Primitive	0x14	An arbitrary string of T.61 (eight-bit) characters
IA5String	Primitive	0x16	An arbitrary string of IA5 (ASCII) characters
UTCTime	Primitive	0x17	A "coordinated universal time" or Greenwich Mean Time (GMT) value

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

Some of **structured types** defined in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
SEQUENCE	Constructed	0x30	An ordered collection of one or more types
SEQUENCE OF	Constructed	0x10	An ordered collection of zero or more occurrences of a given type
SET	Constructed	0x31	An unordered collection of one or more types
SET OF	Constructed	0x11	An unordered collection of zero or more occurrences of a given type
A0, A1, ...	Constructed	0xAz	Where z = 0..F in hex and it represents the z-th element in SEQUENCE data type

2.3 SNMP – RFC 1157

ASN.1 TLV:

T - tag	L - length	V - value
1-2 bytes	1-3 bytes	n bytes

TLV Object

Tag	Length	Value				
0x85	0x05	0x46	0x72	0x61	0x6E	0x6B

Tag for first names | Length of the first name | First name "Frank" in ASCII

Tag for constructed data object | **Length** of the constructed data object | **Value** of the constructed data object

Tc	Lc	Vc

T_{p1}	L_{p1}	V_{p1}	T_{p2}	L_{p2}	V_{p2}	T_{p3}	L_{p3}	V_{p3}

2.3 SNMP – RFC 1157

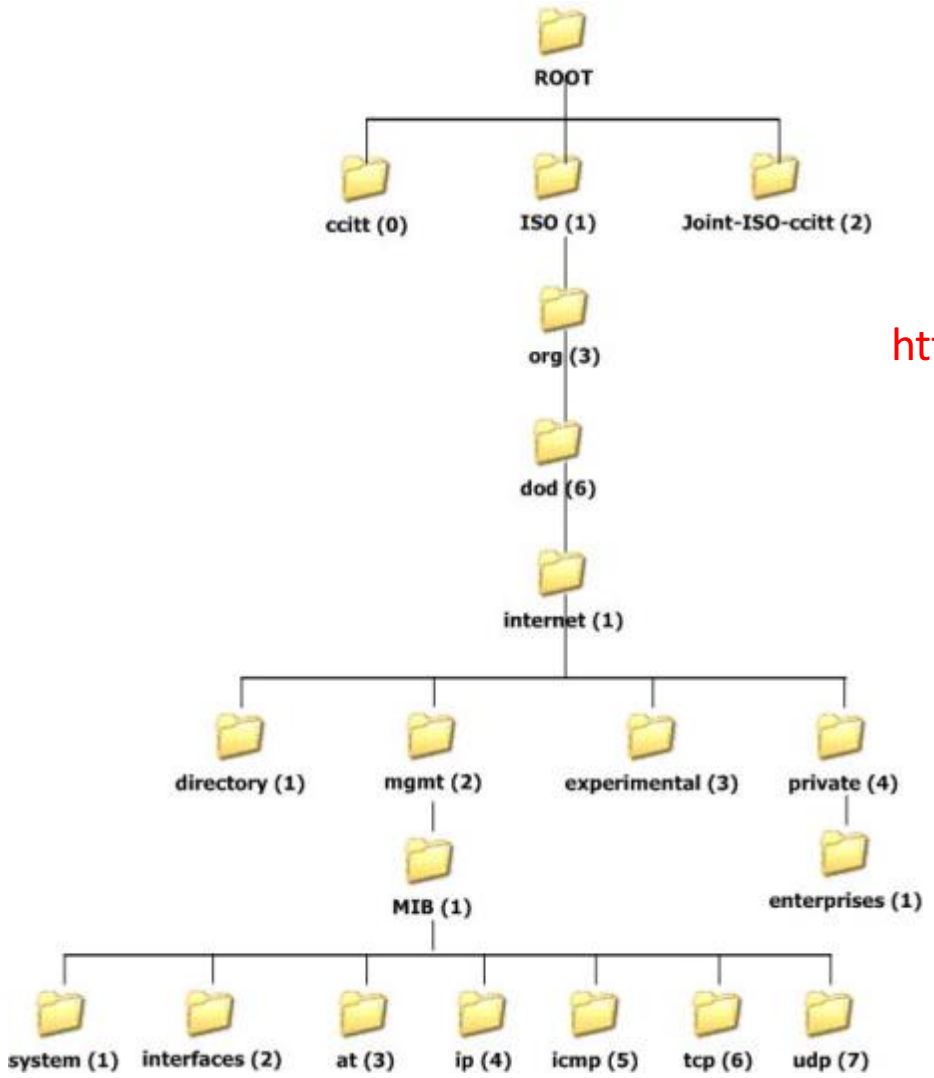
ASN.1 DER and BER:

T - tag			L - length			
b7 b6 b5 b4 b3 b2 b1 b0	Meaning	b7 b6 b5 b4 b3 b2 b1 b0	Byte 1	Byte 2	Byte 3	Meaning
0 0	Universal class	0	0 - 127	...	...	One byte is needed
0 1	Application class		0x81	128-255		Two bytes are needed
1 0	Context-specific class	The value of tag code in range 31-127	0x82	256-65535		Three bytes are needed
1 1	Private class					
... ... 0	Primitive data object					
... ... 1	Constructed data obj.					
... ... Y Y Y Y Y	Tag code (0-30)					
... ... 1 1 1 1 1	Pointer to the following byte (byte 2), which specifies the tag code					

Byte 1 | Byte 2 | Byte 3,4,5

2.3 SNMP – RFC 1157

ASN.1 OID:



http://www.oid-info.com/cgi-bin/display

2.3 SNMP – RFC 1157

ASN.1 OID:

1.2.840.113549.1.1.5
({iso(1) member-body(2) us(840) rsadsi(113549)
pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)})

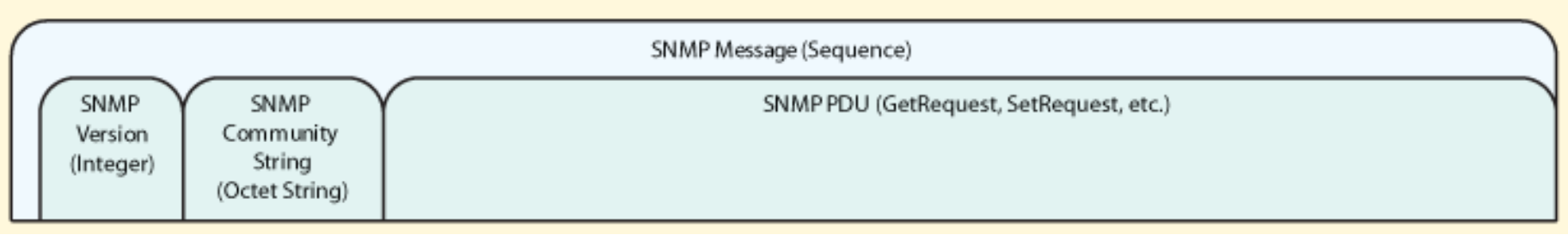
sample of OID is 1.2.840.113549.1.1.5 ({iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}) for signature obtained from SHA-1 digest on the message and RSA algorithm applied to the digest. This encoding will be at byte level: 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x05. The first byte shows that here there is a OID – OBJECT IDENTIFIER field. The OID has 9 bytes length because of second byte in array. Because the first bit in length field is not set the length field has only one byte. The length field has value 9 which means the OID structure has the payload data in 9 bytes.

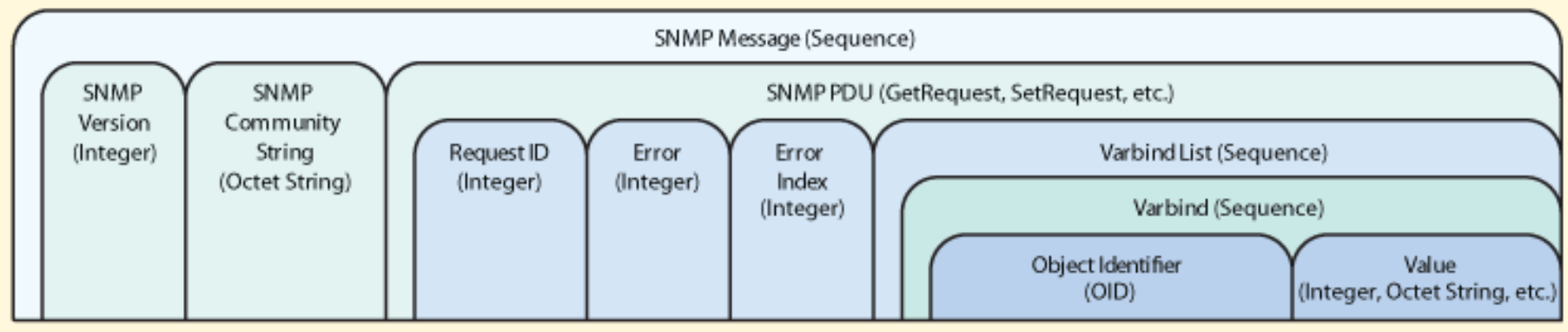
2.3 SNMP – RFC 1157

ASN.1 OID:

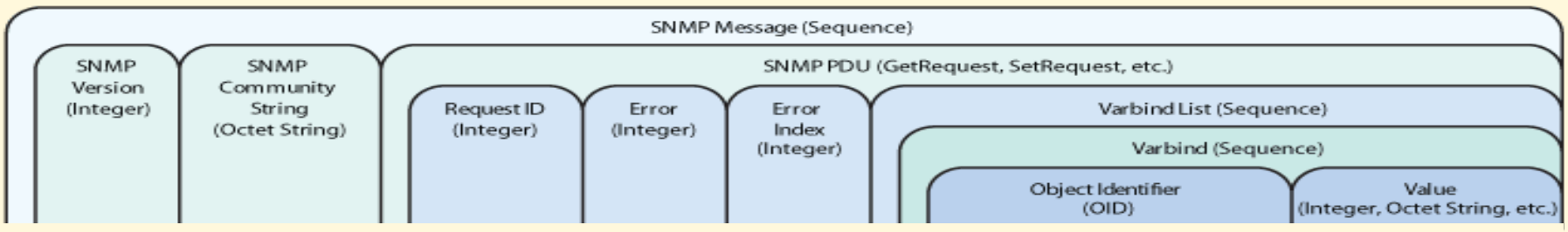
According to BER, the first two numbers of any OID (x.y) are encoded as one value using the formula (40*x)+y. The first two numbers in an OID are here 1.2. Therefore, the first two numbers of an OID are encoded as 42 or 0x2A, because (40*1)+2 = 42. After the first two numbers are encoded, the subsequent numbers in the OID are each encoded as a byte. However, a special rule is required for large numbers because one byte (eight bits) can only represent a number from 0-255. This is the case for 840 and 113549. For 840 from the OID, the first bit of the first byte should be set. The number occupies enough number of bytes till the last byte of representation is not having the first bit set. In 840 case 0x48 has the first bit NOT set. So, the formula is the last hex digit from the first byte multiplied with 2^7 = 128 (1 bit is for multiple bytes representation) and added with the value from the second byte as long as the second byte has a value less than 0x80. In 840 case, the formula is: 0x06*128 + 0x48 (form 0x86 0x48)= 768 + 72 = 840. In case of 113549 we choose the bytes 0x86 0xF7 0x0D because 0x0D is the last byte with first bit (sign bit) not set. The formula for 113549 is:
0x06*2^14 + 0x77*2^7 + 0x0d*2^0 = 6*16384 + 119*128 + 13*1 = 98304 + 15232 + 13 = 113549. For the remaining encoding {...pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}, we will have only one byte for each number: 0x01 0x01 0x05.

2.3 SNMP – RFC 1157



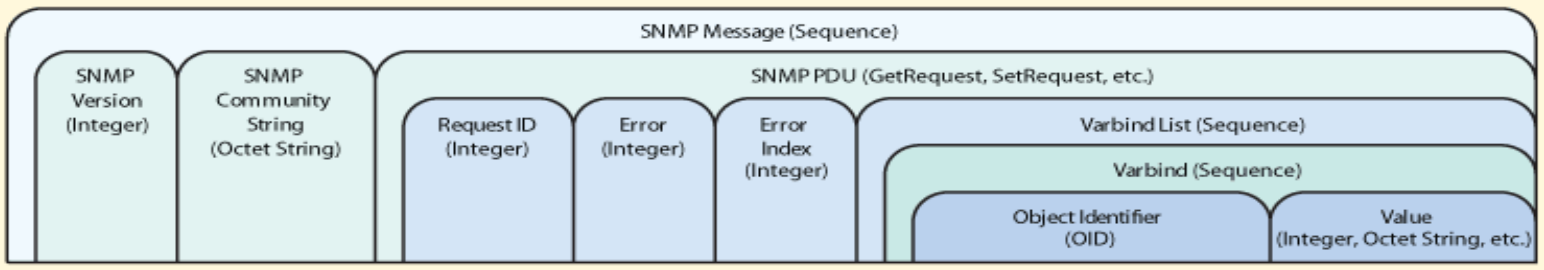


2.3 SNMP – RFC 1157



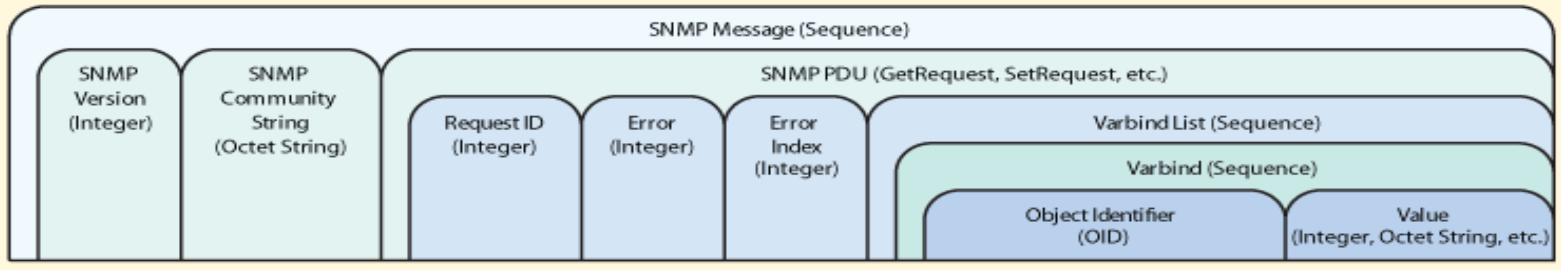
Field	Description
SNMP message	A Sequence representing the entire SNMP message consisting of the SNMP version, Community String, and SNMP PDU.
SNMP Version	An Integer that identifies the version of SNMP. SNMPv1 = 0
SNMP Community String	An Octet String that may contain a string used to add security to SNMP devices.
SNMP PDU	An SNMP PDU contains the body of the SNMP message. There are several types of PDUs. Three common PDUs are GetRequest, GetResponse, SetRequest.
Request ID	An Integer that identifies a particular SNMP request. This index is echoed back in the response from the SNMP agent, allowing the SNMP manager to match an incoming response to the appropriate request.

2.3 SNMP – RFC 1157



Error	An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:
	• 0x00 -- No error occurred
	• 0x01 -- Response message too large to transport
	• 0x02 -- The name of the requested object was not found
	• 0x03 -- A data type in the request did not match the data type in the SNMP agent
	• 0x04 -- The SNMP manager attempted to set a read-only parameter
	• 0x05 -- General Error (some error other than the ones listed above)

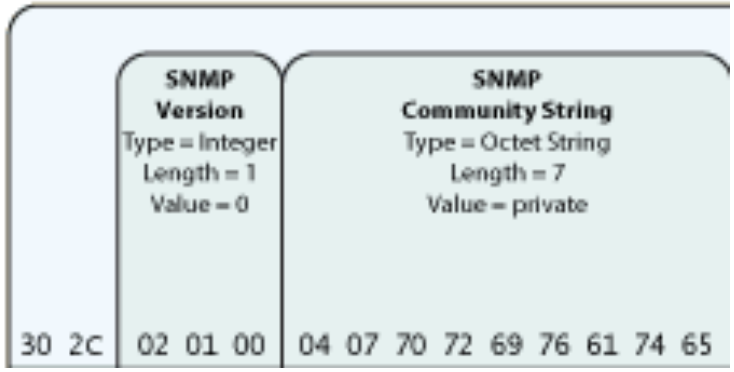
2.3 SNMP – RFC 1157

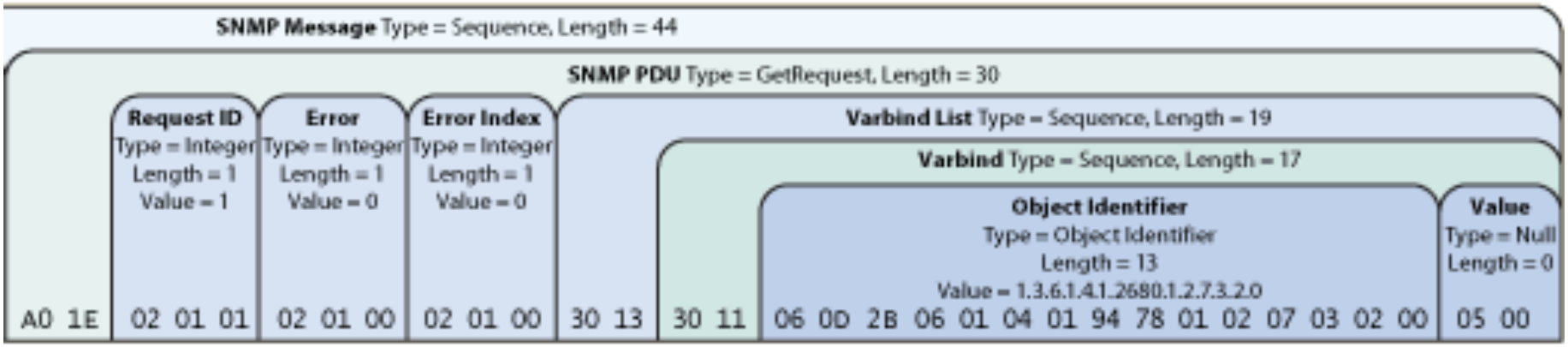


Error Index	If an Error occurs, the Error Index holds a pointer to the Object that caused the error, otherwise the Error Index is 0x00.
Varbind List	A Sequence of Varbinds.
Varbind	A Sequence of two fields, an Object ID and the value for/from that Object ID.
Object Identifier	An Object Identifier that points to a particular parameter in the SNMP agent.

Value	SetRequest PDU -- Value is applied to the specified OID of the SNMP agent.
	GetRequest PDU -- Value is a Null that acts as a placeholder for the return data.
	GetResponse PDU -- The returned Value from the specified OID of the SNMP agent.

2.3 SNMP – RFC 1157





2.4 SMTP – RFC 2821

D.1 A Typical SMTP Transaction Scenario

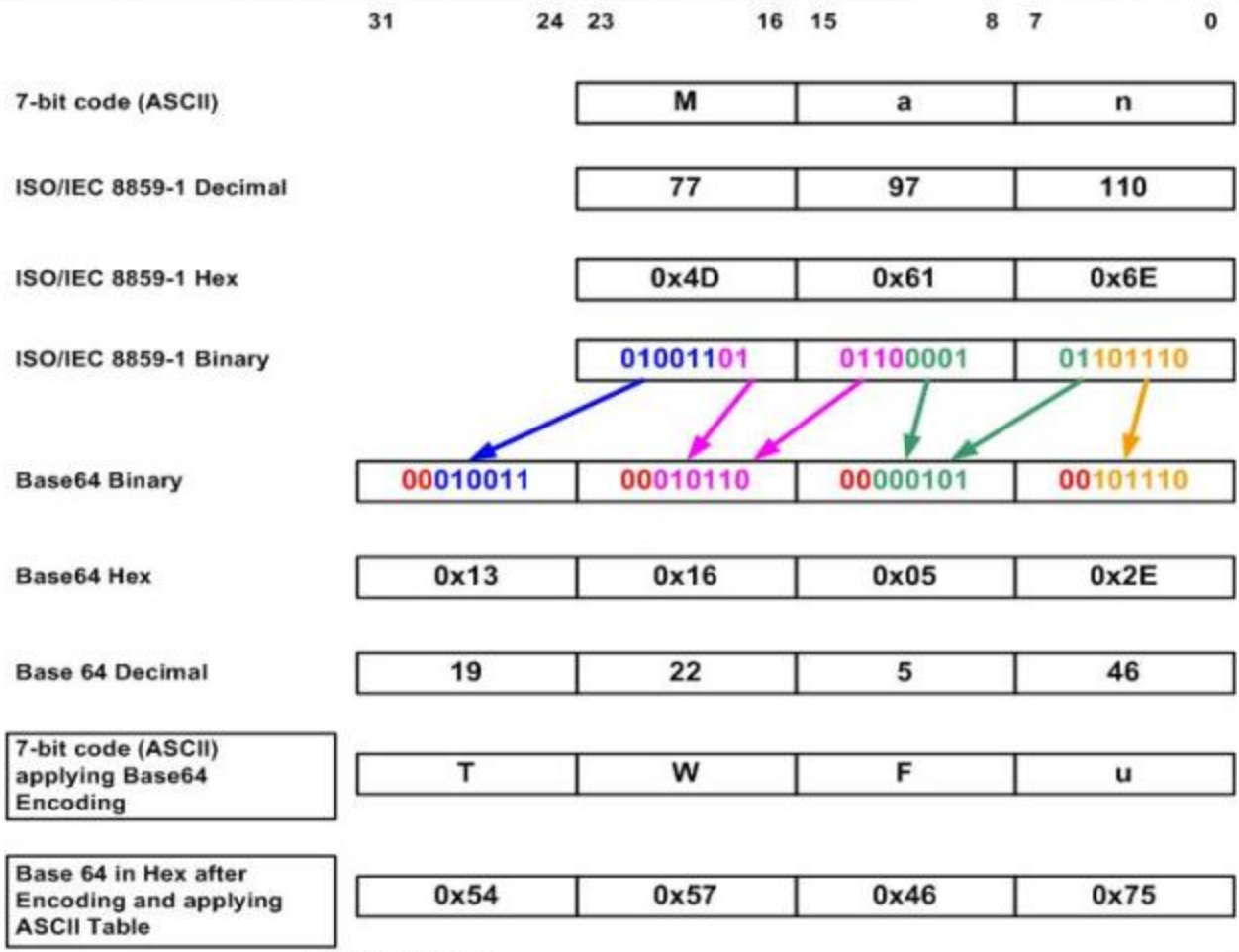
This SMTP example shows mail sent by Smith at host bar.com, to Jones, Green, and Brown at host foo.com. Here we assume that host bar.com contacts host foo.com directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host foo.com.

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>

S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

2.4 SMTP – RFC 2821

Base64 encoding is used in practice usually for transport over the network and heterogeneous environments binary code such as pictures or executable code. The techniques is very simple: to transform each 3 bytes values into 4 bytes value in order to avoid to obtain values greater then 127 per byte.

For instance, if the scope is to encode the word “Man” into Base64 encoding then it is encoded as “TWFu”. Encoded in ASCII (in ISO 8859-1, one value per byte), M, a, n are stored as the bytes 77 (0x4D), 97 (0x61), 110 (0x6E), which are 01001101, 01100001, 01101110 in base 2.



As this example illustrates, the encoding converts 3 not encoded bytes (in this case, ASCII characters) into 4 encoded ASCII characters. If the output is not multiple of 4 bytes then the '=' sign is put in order to supply the padding. Must be considered the padding with = (64 value) in case of multiple 4 bytes number.

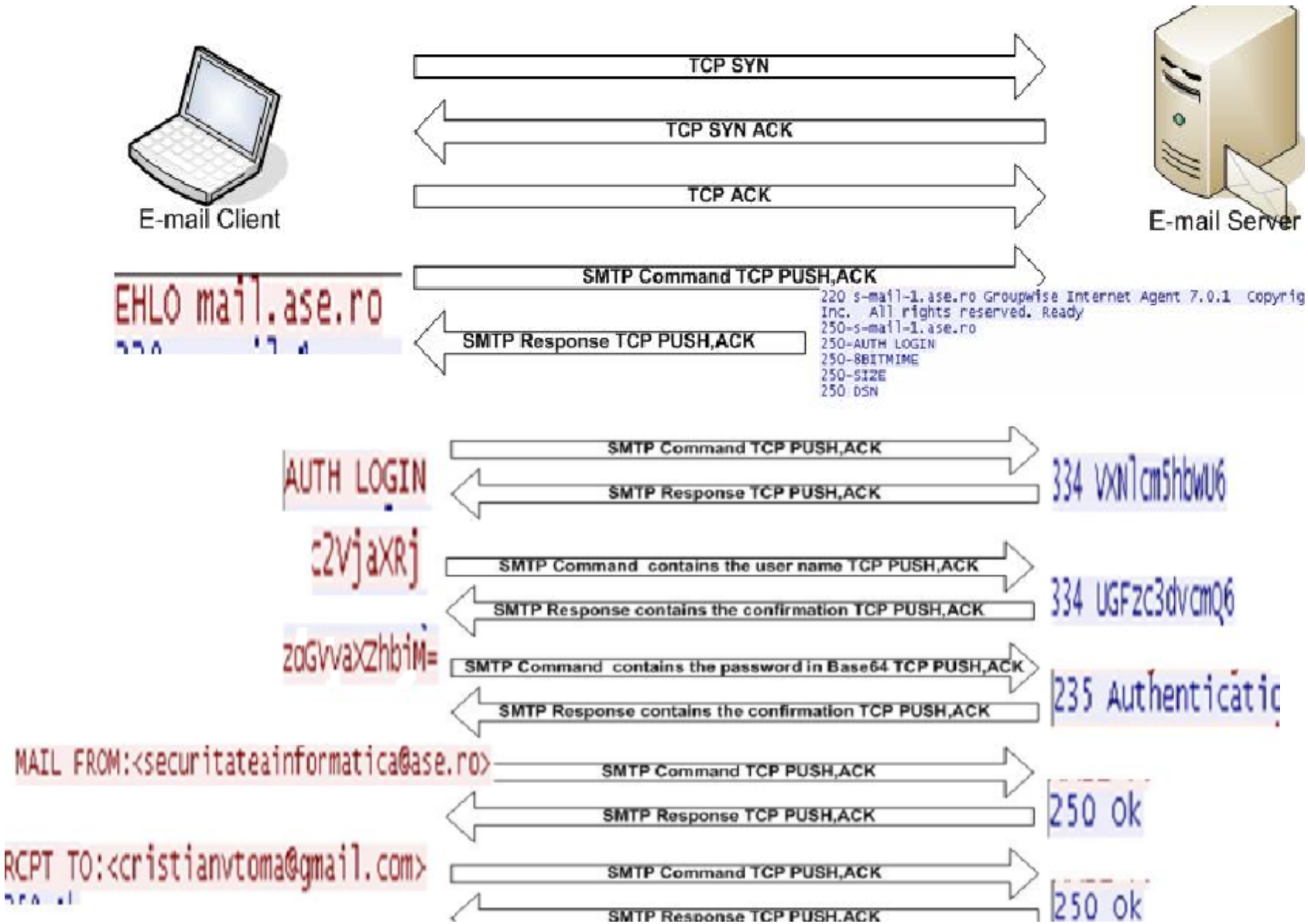
Base 64 Code

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

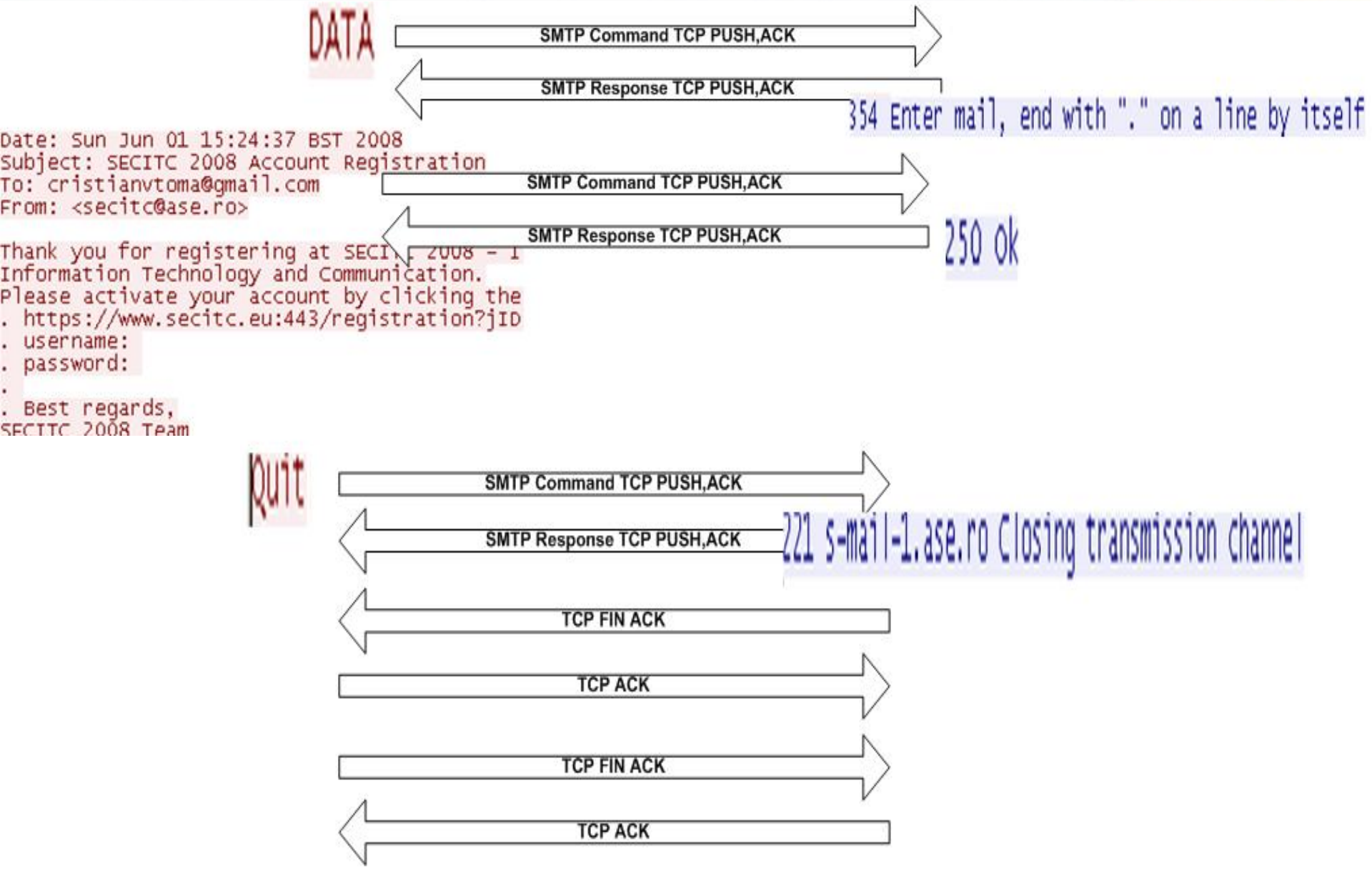
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51

0	1	2	3	4	5	6	7	8	9	+	/	=
52	53	54	55	56	57	58	59	60	61	62	63	64 (64 is used as padding)

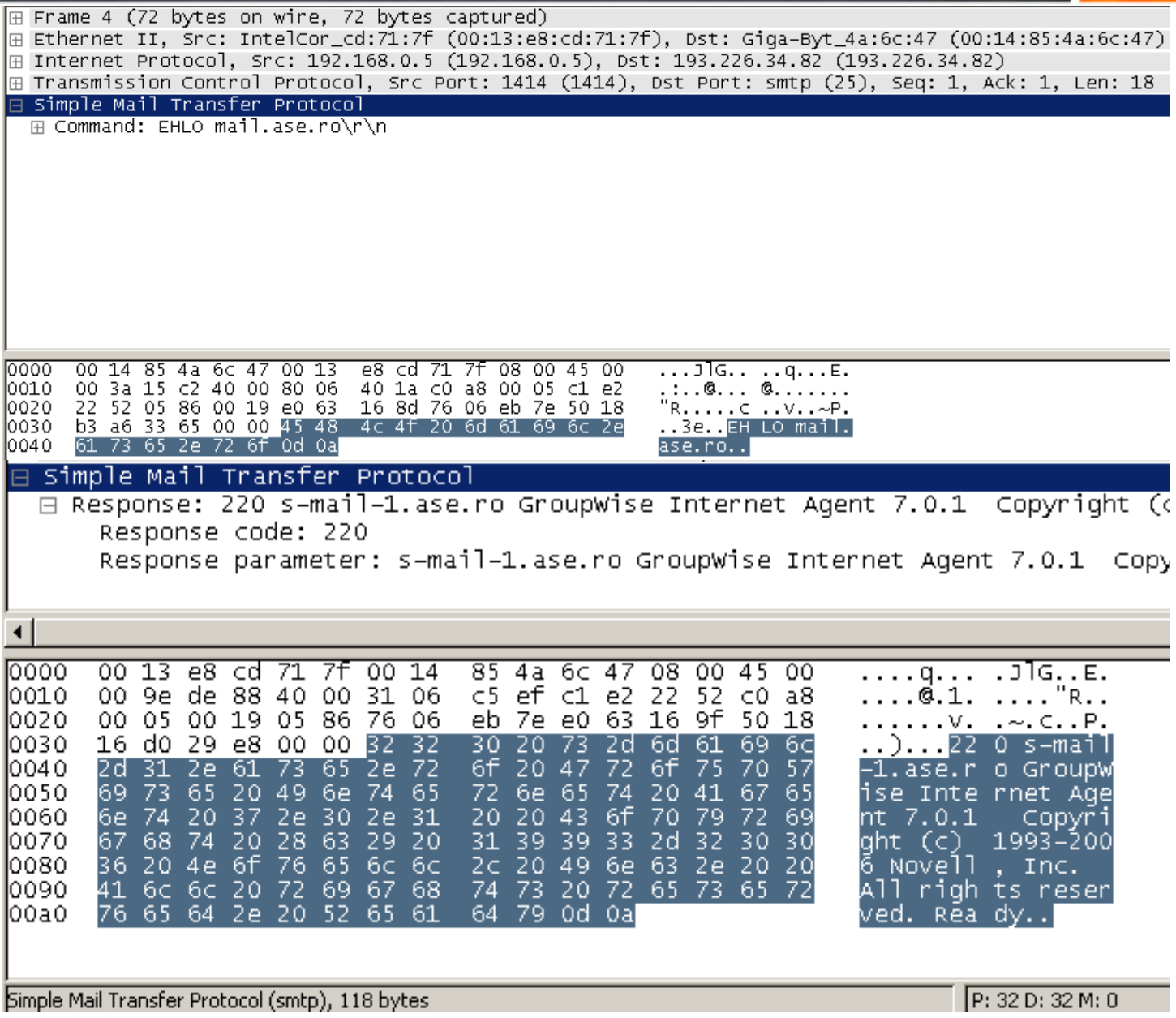
2.4 SMTP – RFC 2821



2.4 SMTP – RFC 2821



2.4 SMTP – RFC 2821



Java Network Programming
for easy sharing

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.

As practical approach, also developed in the laboratory activities, there is a need to implement "raw" SNMP request, but third party libraries – *snmp4j.org* or product – *Naggios, Cacti, Zabbix* may be used.

As practical approach, there is a need to implement "raw" SMTP in order to send e-mails, but Sun/Oracle or third party libraries may be used – java.mail.*.

Network Programming & Java Sockets

Communicate & Exchange Ideas

Questions & Answers!

But wait…

There's More!

Thanks!

DAD – Distributed Application Development
End of Lecture 2 – Summary of Java SE & Network Programming – section 2