

Advance Networks

CT60A4700

Prabhat Kumar, Ph.D., MIEEE

Department of Software Engineering

Email:prabhat.kumar@lut.fi



November 15, 2022



Prabhat Kumar: received his Ph.D. degree in Information Technology, National Institute of Technology (NIT) Raipur, Raipur, India, under the prestigious fellowship of Ministry of Human Resource and Development (MHRD) funded by the Government of India. Thereafter, he worked with Indian Institute of Technology (IIT) Hyderabad, India, as a Post-doctoral Researcher under project "Development of Indian Telecommunication Security Assurance Requirements for IoT devices". He is currently working as Post-doctoral Researcher with the Department of Software Engineering, LUT School of Engineering Science, LUT University, Lappeenranta, Finland. He has many research contributions in the area of Machine Learning, Deep Learning, Federated Learning, Big Data Analytics, Cybersecurity, Blockchain, Cloud Computing, Internet of Things and Software Defined Networking.

Marks Distribution

- 4 Assignments will have 10 marks each (total=40)
- Final Exam will have 60 marks (total=60)

Note: All video lectures and assignments will be available through the starting of this course. Deadlines for submitting assignment will be 1 week after the last lecture mentioned on Moodle.

Course Overview

- 1 Introduction to Computer Networks
Client/Server Model
- 2 Basic Network Concepts and Java I/O
OSI vs TCP/IP layer model
IP, TCP and UDP
IP Addresses, Domain Names and Ports
Internet, Firewalls and Proxy Servers
Streams
- 3 Basic Web Concepts and Network Programming
Internet Addresses
Uniform Resource Identifier (URI)
Uniform Resource Locator (URL)
Hypertext Transfer Protocol (HTTP)
URLConnections
Sockets for Clients
Sockets for Servers
The UDP Protocol

Client/Server Model

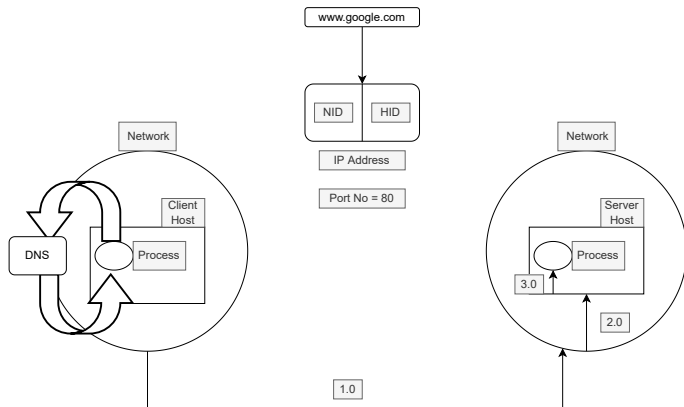


Figure 1: Client Server model

Client/Server Model

Basic Network Concept:

Network

- A *network* is a collection of computers and other devices that can send data to and receive data from one another, more or less in real time.
- A network is often connected by wires, and the bits of data are turned into electromagnetic waves that move through the wires.

Node

Each machine on a network is called a *node*. Most nodes are computers, but printers, routers, bridges and gateways can also be nodes.

Client/Server Model (cont'd)

Host

- Nodes that are fully functional computers are also called *Host*.
- We will use the word *node* to refer to any device on the network, and the word *host* to refer to a node that is a general-purpose computer.

Address

- Every network node has an *address*, a sequence of bytes that uniquely identifies it.
- More bytes means more addresses are available and the more devices can be connected to the network simultaneously.

Note: All modern computer networks are packet-switched networks: data traveling on the network is broken into chunks called *packets* and each packet is handled separately. Each packet contains information about who sent it and where it's going.

Protocol

- A *protocol* is a precise set of rules defining how computers communicate: the format of addresses, how data is split into packets, and so on.
- For example, the Hypertext Transfer Protocol (HTTP) defines how web browsers servers communicate.

Challenges:

- Sending data across a network is a complex operation due to the *physical characteristics of the network* as well as the *logical character of the data* being sent.
- Software that sends data across a network must understand: *how to avoid collisions between packets, convert digital data to analog signals, detect and correct errors, route packets from one host to another.*
- How to support multiple operating systems and heterogeneous network cabling.

OSI vs TCP/IP layer model

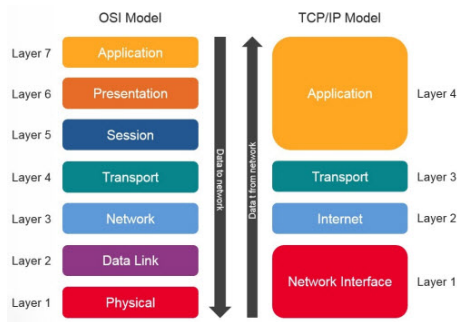


Figure 2: The OSI model vs TCP/IP layer model

The OSI model and TCP/IP model is shown in Figure 2.

OSI vs TCP/IP layer model (cont'd)

Application Layer

- To allow access to network resources.

Presentation Layer

- To translate, encrypt, and compress data.

Session Layer

- To establish, manage, and terminate session.

OSI vs TCP/IP layer model (cont'd)

Transport Layer

- To provide reliable process-to-process message delivery and error recovery.

Network Layer

- To move packets from source to destination.
- To provide internetworking.

Data Link Layer

- To organize bits into frames.
- To provide hop-to-hop delivery.

Physical Layer

- To transmit bits over a medium.
- To provide mechanical and electrical specifications.

IP, TCP and UDP

IP

- IP was designed to allow multiple routes between any two points and to route packets of data around damaged routers.

TCP

- TCP was layered on top of IP to give each end of a connection the ability to acknowledge receipt of IP packets and request retransmission of lost or corrupted packets.

UDP

- TCP, however, carries a fair amount of overhead. Packets are sometimes sent without the guarantees that TCP provides using the UDP protocol.
- UDP is an unreliable protocol that does not guarantee that packets will arrive at their destination or that they will arrive in the same order they were sent.

IP Addresses

- As a Java programmer, we should know about addressing.
- Every computer on an IPv4 network is identified by a four-byte number.
- This is normally written in a dotted quad format like 199.1.32.90, where each of the four numbers is one unsigned byte ranging in value from 0 to 255.
- There are a little more than four billion possible IP addresses.
- IPv6 addresses are customarily written in eight blocks of four hexadecimal digits separated by colons, such as FEDC:BA98:7654:3210:FEDC:BA98:7654:3210.
- In mixed networks of IPv6 and IPv4, the last four bytes of the IPv6 address are sometimes written as an IPv4 dotted quad address. For example, FEDC:BA98:7654:3210:FEDC:BA98:7654:3210 could be written as FEDC:BA98:7654:3210:FEDC:BA98:118.84.50.16.

IP Addresses, Domain Names and Ports (cont'd)

Domain Names

- Domain Name System (DNS) was developed to translate hostnames that humans can remember, such as “www.oreilly.com,” into numeric Internet addresses such as 208.201.239.101.
- When Java programs access the network, they need to process both these numeric addresses and their corresponding hostnames. This is done with the help of `java.net.InetAddress` class

Ports

- A port is a communication endpoint. Typically, ports identify a specific network service assigned to them.
- Within an operating system, the primary use of a port number is to transfer the data between a computer network and an application.

IP Addresses, Domain Names and Ports (cont'd)

Protocol	Port	Protocol	Purpose
echo	7	TCP/UDP	Echo is a test protocol used to verify that two machines are able to connect by having one echo back the other's input.
discard	9	TCP/UDP	Discard is a less useful test protocol in which all data received by the server is ignored.
daytime	13	TCP/UDP	Provides an ASCII representation of the current time on the server.
FTP data	20	TCP	FTP uses two well-known ports. This port is used to transfer files.
FTP	21	TCP	This port is used to send FTP commands like put and get.
SSH	22	TCP	Used for encrypted, remote logins.
Telnet	23	TCP	Used for interactive, remote command-line sessions.
smtp	25	TCP	The Simple Mail Transfer Protocol is used to send email between machines.
time	37	TCP/UDP	A time server returns the number of seconds that have elapsed on the server since midnight, January 1, 1900, as a four-byte, unsigned, big-endian integer.
whois	43	TCP	A simple directory service for Internet network administrators.
finger	79	TCP	A service that returns information about a user or users on the local system.
HTTP	80	TCP	The underlying protocol of the World Wide Web.
POP3	110	TCP	Post Office Protocol version 3 is a protocol for the transfer of accumulated email from the host to sporadically connected clients.
NNTP	119	TCP	Usenet news transfer; more formally known as the "Network News Transfer Protocol."
IMAP	143	TCP	Internet Message Access Protocol is a protocol for accessing mailboxes stored on a server.
dict	2628	TCP	A UTF-8 encoded dictionary service that provides definitions of words.

Figure 3: Well-known port assignments

Internet, Firewalls and Proxy Servers (cont'd)

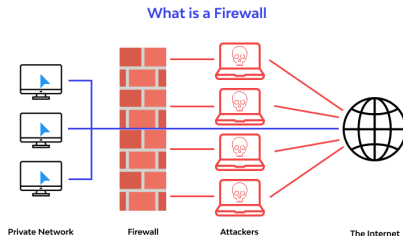


Figure 5: Working of Firewalls

Firewall

- Firewall is responsible for inspecting each packet that passes into or out of its network interface and accepting it or rejecting it according to a set of rules.

Internet, Firewalls and Proxy Servers (cont'd)

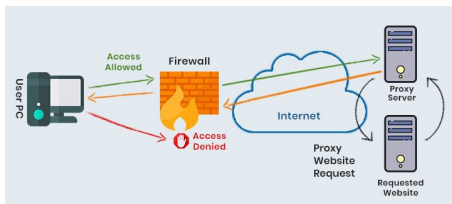


Figure 6: Working of Proxy Servers

Proxy Servers

- Proxy servers are related to firewalls.
- Advantages of using a proxy server is that external hosts only find out about the proxy server. They do not learn the names and IP addresses of the internal machines, making it more difficult to hack into internal systems.

Streams

Streams

Network programs do is simple input and output: moving bytes from one system to another. Input streams read data; output streams write data.

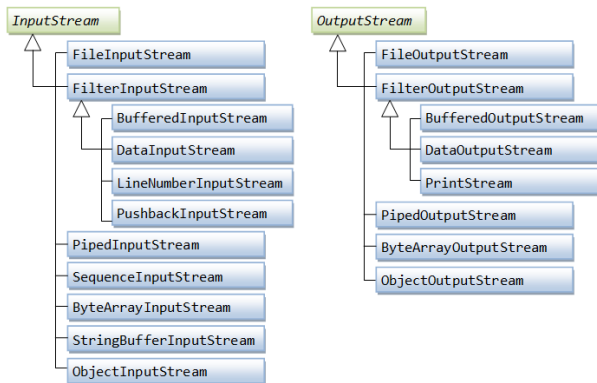


Figure 7: Classification Chart for `InputStream` and `OutputStream`

Output Streams

Subclasses of `OutputStream` use these methods to write data onto particular media. The output class is `java.io.OutputStream` that provides the fundamental methods needed to write data. These are:

`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`

Note: `OutputStream`'s fundamental method is `write(int b)`. This method takes an integer from 0 to 255 as an argument and writes the corresponding byte to the output stream.

Output Stream Example

In this example, we are writing the textual information in the `BufferedOutputStream` object which is connected to the `FileOutputStream` object. The `flush()` flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

```
1 import java.io.*;
2 public class BufferedOutputStreamExample{
3     public static void main(String args[])throws Exception{
4         FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
5         BufferedOutputStream bout=new BufferedOutputStream(fout);
6         String s="Welcome to Networking Classes.";
7         byte b[]=s.getBytes();
8         bout.write(b);
9         bout.flush();
10        bout.close();
11        fout.close();
12        System.out.println("success");
13    }
14 }
```

Figure 8: Example of `BufferedOutputStream` class

Input Streams

Java's basic input class is `java.io.InputStream`. This class provides the fundamental methods needed to read data as raw bytes. These are:

`public abstract int read() throws IOException`

`public int read(byte[] input) throws IOException`

`public int read(byte[] input, int offset, int length) throws IOException`

Note: This method reads a single byte of data from the input stream's source and returns it as an `int` from 0 to 255. End of stream is signified by returning -1.

Input Stream Example

Let's see the simple example to read data of file using `BufferedInputStream`:

```
1 import java.io.*;
2 public class BufferedInputStreamExample{
3     public static void main(String args[]){
4         try{
5             FileInputStream fin=new FileInputStream("D:\\testout.txt");
6             BufferedInputStream bin=new BufferedInputStream(fin);
7             int i;
8             while((i=bin.read())!=-1){
9                 System.out.print((char)i);
10            }
11            bin.close();
12            fin.close();
13        }catch(Exception e){System.out.println(e);}
14    }
15 }
```

Figure 9: Example of `BufferedInputStream` class

Internet Addresses

Internet Addresses

Creating New InetAddress Objects: There are no public constructors in the InetAddress class. Instead, InetAddress has static factory methods that connect to a DNS server to resolve a hostname. The most common is `InetAddress.getByName()`. <https://onecompiler.com/java/3yft2aa4s>

```
1 import java.net.*;
2 public class lappeenrantaByName {
3     public static void main (String[] args) {
4         try {
5             InetAddress address = InetAddress.getByName("www.lappeenranta.fi");
6             System.out.println(address);
7         } catch (UnknownHostException ex) {
8             System.out.println("Could not find");
9         }
10    }
11 }
```

STDIN
Input for the program (Optional)

Output:
www.lappeenranta.fi/194.89.230.214

Figure 10: Example program that prints the address of "www.lappeenranta.fi"

Internet Addresses

getHostName()

We can also find the hostname with ip address using `getHostName()` method. If the address you look up does not have a hostname, `getHostName()` simply returns the dotted quad address you supplied.

```
InetAddress address = InetAddress.getByName("194.89.230.214");  
System.out.println(address.getHostName());
```

getLocalHost()

The `getLocalHost()` method returns an `InetAddress` object for the host on which your code is running:

```
InetAddress me = InetAddress.getLocalHost();  
System.out.println(me);
```

Getter Methods

The `InetAddress` class contains four getter methods that return the hostname as a string and the IP address as both a string and a byte array:

```
public String getHostName()  
public String getCanonicalHostName()  
public byte[] getAddress()  
public String.getHostAddress()
```

Definition

- The `getHostName()` method returns a `String` that contains the name of the host with the IP address and will only call DNS if it doesn't think it already knows the hostname.

Definition

- The `getCanonicalHostName()` method is similar but calls DNS if it can, and may replace the existing cached hostname.
- To know the IP address of a machine we use the `getAddress()` method, which returns an IP address as an array of bytes in network byte order.
- The `getHostAddress()` method returns a string containing the dotted quad format of the IP address.

Uniform Resource Identifier (URI)

URI

- A Uniform Resource Identifier (URI) is a string of characters in a particular syntax that identifies a resource.
- The resource identified may be a file on a server; but it may also be an email address, a news message, a book, a person's name, an Internet host, the current stock price of Oracle, or something else.
- A resource is a thing that is identified by a URI. A URI is a string that identifies a resource.
- A Uniform Resource Locator (URL) is the most common type of URI.
- **The syntax of a URI is composed of a scheme and a scheme-specific part, separated by a colon**, like this: *scheme:scheme-specific-part*

Note: The syntax of the scheme-specific part depends on the scheme being used. Current schemes include: data, file, ftp, http, mailto, magnet, telnet and urn.

Uniform Resource Identifier (URI)

Note: There is no specific syntax that applies to the scheme-specific parts of all URIs. However, many have a hierarchical form, like this:

//authority/path?query

Example 1: For instance, the URI *http : //www.ietf.org/rfc/rfc3986.txt* has the scheme *http*, the authority *www.ietf.org*, and the path */rfc/rfc3986.txt* (initial slash included). This means the server at *www.ietf.org* is responsible for mapping the path */rfc/rfc3986.txt* to a resource. This URI does not have a query part.

Example 2: The URI *http : //www.powells.com/cgi-bin/biblio?inkey = 62 – 1565928709 – 0* has the scheme *http*, the authority *www.powells.com*, the path */cgi – bin/biblio*, and the query *inkey = 62 – 1565928709 – 0*.

Example 3: The URI *urn : isbn : 156592870* has the scheme *urn* but doesn't follow the *hierarchical //authority/path?query* form for scheme-specific parts.

Uniform Resource Locator (URL)

URL

- URI may tell us what a resource is, but not actually tells where or how to get that resource.
- A URL is a URI that, as well as identifying a resource, provides a specific network location for the resource that a client can use to retrieve a representation of that resource.
- In Java, it's the difference between the `java.net.URI` class that only identifies resources and the `java.net.URL` class that can both identify and retrieve resources.
- **The syntax of a URL is:** `protocol://userInfo@host:port/path?query#fragment`

Example: A URL looks like `http://www.ibiblio.org/javafaq/javatutorial.html`. This specifies that there is a file called `javatutorial.html` in a directory called `javafaq` on the server `www.ibiblio.org`, and that this file can be accessed via the `HTTP` protocol.

Uniform Resource Locator (URL)

URL

- The `java.net.URL` class is an abstraction of a URL. URLs are immutable. After a URL object has been constructed, its fields do not change.

Creating New URLs

- We can construct instances of `java.net.URL`. The constructors differ in the information they require:

`public URL(String url) throws MalformedURLException`

`public URL(String protocol, String hostname, String file) throws MalformedURLException`

`public URL(String protocol, String host, int port, String file) throws MalformedURLException`

Uniform Resource Locator (URL)

Note: All these constructors throw a `MalformedURLException` if you try to create a URL for an unsupported protocol or if the URL is syntactically incorrect.

Constructing a URL from a string

- The simplest URL constructor just takes an absolute URL in string form as its single argument:

`public URL(String url) throws MalformedURLException`

Uniform Resource Locator (URL)

</> source code

```
1  import java.net.*;
2  class ProtocolTester {
3  private static void testProtocol(String url) {
4      try {
5          URL u = new URL(url);
6          System.out.println(u.getProtocol() + " is supported");
7      } catch (MalformedURLException ex) {
8          String protocol = url.substring(0, url.indexOf(':'));
9          System.out.println(protocol + " is not supported");
10     }
11 }
12 public static void main(String[] args) {
13     // hypertext transfer protocol
14     testProtocol("http://www.adc.org");
15     // secure http
16     testProtocol("https://www.amazon.com/exec/obidos/order2/");
17     // file transfer protocol
18     testProtocol("ftp://ibiblio.org/pub/languages/java/javafaq/");
19     // Simple Mail Transfer Protocol
20     testProtocol("mailto:elharo@ibiblio.org");
21     // telnet
22     testProtocol("telnet://dibner.poly.edu/");
23     // local file access
24     testProtocol("file:///etc/passwd");
25 }
26 }
27 }
```

input output

Success #stdin #stdout 0.14s 50924KB
http is supported
https is supported
ftp is supported
mailto is supported
telnet is not supported
file is supported

Figure 11: Example program for determining which protocols a virtual machine supports

Uniform Resource Locator (URL)

Constructing a URL from its component parts

- We can also build a URL by specifying the protocol, the hostname, and the file:

*public URL(String protocol, String hostname, String file) throws
MalformedURLException*

```
29 try {  
30     URL u = new URL("http", "www.eff.org", "/blueribbon.html#intro");  
31 } catch (MalformedURLException ex) {  
32     throw new RuntimeException("shouldn't happen; all VMs recognize http");  
33 }
```

Note: For the rare occasions when the default port isn't correct, the next constructor lets you specify the port explicitly as an int

```
34 try {  
35     URL u = new URL("http", "fourier.dur.ac.uk", 8000, "/~dma3mjh/jsci/");  
36 } catch (MalformedURLException ex) {  
37     throw new RuntimeException("shouldn't happen; all VMs recognize http");  
38 }
```

Hypertext Transfer Protocol (HTTP)

HTTP

- The Hypertext Transfer Protocol (HTTP) is a standard that defines how a web client talks to a server and how data is transferred from the server back to the client.
- It can be used to transfer TIFF pictures, Microsoft Word documents, Windows .exe files, or anything else that can be represented in bytes.

Note: HTTP connections use the TCP/IP protocol for data transfer. For each request from client to server, there is a sequence of four steps:

- 1 The client opens a TCP connection to the server on port 80, by default; other ports may be specified in the URL.
- 2 The client sends a message to the server requesting the resource at a specified path. The request includes a header.
- 3 The server sends a response to the client. The response begins with a response code, followed by a header full of metadata, a blank line, and the requested document or an error message.
- 4 The server closes the connection.

Hypertext Transfer Protocol (HTTP)

Note: This is the basic HTTP 1.0 procedure. In HTTP 1.1 and later, multiple requests and responses can be sent in series over a single TCP connection. That is, steps 2 and 3 can repeat multiple times in between steps 1 and 4. Furthermore, in HTTP 1.1, requests and responses can be sent in multiple chunks.

Request Structure

A typical client request looks something like this:

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:20.0)
Gecko/20100101 Firefox/20.0
Host: en.wikipedia.org
Connection: keep-alive
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Hypertext Transfer Protocol (HTTP)

Response Structure

A typical successful response looks something like this:

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2013 15:12:46 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-length: 115
```

```
<html>
<head>
<title>
A Sample HTML file
</title>
</head>
<body>
The rest of the document goes here
</body>
</html>
```

HTTP Methods

There are four main HTTP methods: GET, POST, PUT and DELETE.

Hypertext Transfer Protocol (HTTP)

GET

The GET method retrieves a representation of a resource.

PUT

The PUT method uploads a representation of a resource to the server at a known URL.

POST

The POST method is the most general method. It too uploads a representation of a resource to a server at a known URL, but it does not specify what the server is to do with the newly supplied resource.

DELETE

The DELETE method removes a resource from a specified URL.

URLConnections

- URLConnection is an abstract class that represents an active connection to a resource specified by a URL.
- It provides more control over the interaction with a server (especially an HTTP server) than the URL class.

Opening URLConnections

A program that uses the URLConnection class directly follows this basic sequence of steps:

- 1 Construct a URL object.
- 2 Invoke the URL object's `openConnection()` method to retrieve a URLConnection object for that URL.
- 3 Get an input stream and read data.
- 4 Get an output stream and write data.
- 5 Close the connection

URLConnections

Reading Data from a Server

The `getInputStream()` method returns a generic `InputStream` that lets you read and parse the data that the server sends.

```
1  import java.io.*;
2  import java.net.*;
3
4  class URLConnection1 {
5  public static void main (String[] args) {
6      if (args.length > 0) {
7          try {
8              // Open the URLConnection for reading
9              URL u = new URL(args[0]);
10             java.net.URLConnection uc = u.openConnection();
11             try (InputStream raw = uc.getInputStream()) { // autoclose
12                 InputStream buffer = new BufferedInputStream(raw);
13                 // chain the InputStream to a Reader
14                 Reader reader = new InputStreamReader(buffer);
15                 int c;
16                 while ((c = reader.read()) != -1) {
17                     System.out.print((char) c);
18                 }
19             } catch (MalformedURLException ex) {
20                 System.err.println(args[0] + " is not a parseable URL");
21             } catch (IOException ex) {
22                 System.err.println(ex);
23             }
24         }
25     }
26 }
27
28 }
```

Figure 12: Example program for downloading a web page with a `URLConnection`

URLConnections

```
C:\PC\Desktop\Network>java URLConnection1 "https://www.google.com"
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="fi"
><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta
content="/images/branding/google/1x/google_standard_color_128dp.png" itemprop=
"image"><title>Google</title><script nonce="iY_eDZrKEmU1P60DiuW_g"><function><<
window.google=(kEl:'des1Y_K2ndfV1s0P9dSv000',kEXPI:'0.1302536.56073.6058.207.400
4.2316.303.246.5.5367.1123753.1197775.626.300009.16109.20690.22431.1361.12319.17
500.4990.13220.3047.10622.22741.5001.005.700.1279.2742.149.1103.040.2197.4100.35
14.606.2023.1777.520.14670.3227.2845.7.33770.1051.15756.3.346.230.4306.2073.149.
13975.4.1528.2304.32112.2658.7357.13658.21223.5821.2536.4094.17.4035.3.3541.1.42
154.2.14022.14116.11623.5679.1020.2381.22660.6074.4567.6256.23421.1252.5035.1496
0.4332.7484.17756.9326.0155.6502.799.129.14551.1209.874.19633.7.1922.9779.19130.
5292.6900.4832.5761.2173.8097.984.123.700.4.1.2.2.2.2.1.7390.1009.1567.3947.12
00.3927.5206.2019.2604.5.3890.751.446.1622.3285.760.2800.649.77.90.1104.766.38.7
24.922.72.419.16.14.146.195.8.4.300.10.752.524.39.256.987.1242.476.87.147.904.13
27.126.3.3.2.2.1473.48.924.945.760.102.2046.161.607.623.660.079.11.640.13.159.50
9.850.26.1447.3209.2.5344714.509.160.2.34.8799318.3452.795.2.19733.1.1.346.1527.
43.79.5.3.3.1.2.7.8725104.15223711.2773819.1268323.1964.3094.13579.3405.5540.405
1.1375748.47703'.kBL:'o7pG');google.sn='webhp';google.kHL='fi';><></function><<
```

Figure 13: Output of a web page with a URLConnection

Sockets

- Sockets allow the programmer to treat a network connection as just another stream onto which bytes can be written and from which bytes can be read.
- Sockets shield the programmer from low-level details of the network, such as error detection, packet sizes, packet splitting, packet retransmission, network addresses, and more.

Sockets for Clients

```
1  import java.net.*;
2  import java.io.*;
3  class DaytimeClient {
4  public static void main(String[] args) {
5  String hostname = args.length > 0 ? args[0] : "time.nist.gov";
6  Socket socket = null;
7  try {
8  socket = new Socket(hostname, 13);
9  socket.setSoTimeout(15000);
10 InputStream in = socket.getInputStream();
11 StringBuilder time = new StringBuilder();
12 InputStreamReader reader = new InputStreamReader(in, "ASCII");
13 for (int c = reader.read(); c != -1; c = reader.read()) {
14 time.append((char) c);
15 }
16 System.out.println(time);
17 } catch (IOException ex) {
18 System.err.println(ex);
19 } finally {
20 if (socket != null) {
21 try {
22 socket.close();
23 } catch (IOException ex) {
24 // ignore
25 }
26 }
27 }
28 }
29 }
```

Figure 14: Example program for printing daytime protocol client

Sockets for Clients

```
C:\PC\Desktop\Network>java DaytimeClient
59842 22-09-20 13:08:04 50 0 0 745.9 UTC(NIST) *
C:\PC\Desktop\Network>
```

Figure 15: Output for printing daytime protocol client

Sockets for Servers

A server socket runs on the server and listens for incoming TCP connections.

Basic life cycle

- 1 A new `ServerSocket` is created on a particular port using a `ServerSocket()` constructor.
- 2 The `ServerSocket` listens for incoming connection attempts on that port using its `accept()` method.
- 3 Depending on the type of server, either the `Socket`'s `getInputStream()` method, `getOutputStream()` method, or both are called to get input and output streams that communicate with the client.
- 4 The server and the client interact according to an agreed-upon protocol until it is time to close the connection.
- 5 The server, the client, or both close the connection.

Note: Once a `ServerSocket` has set up the connection, the server uses a regular `Socket` object to send data to the client. Data always travels over the regular socket.

Sockets for Servers

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.Date;
4  class DaytimeServer {
5      public final static int PORT = 13;
6      public static void main(String[] args) {
7          try (ServerSocket server = new ServerSocket(PORT)) {
8              while (true) {
9                  try (Socket connection = server.accept()) {
10                     Writer out = new OutputStreamWriter(connection.getOutputStream());
11                     Date now = new Date();
12                     out.write(now.toString() + "\r\n");
13                     out.flush();
14                     connection.close();
15                 } catch (IOException ex) {}
16             }
17             } catch (IOException ex) {
18                 System.err.println(ex);
19             }
20         }
21     }
```

Figure 16: Example program for daytime protocol server

Note: The `accept()` method is called within an infinite loop to watch for new connections; like many servers, this program never terminates but continues listening until an exception is thrown or you stop it manually.

TCP Echo Server Example

```
1  import java.net.*;
2  import java.io.*;
3  class Tcpechoserver
4  {
5      public static void main(String[] arg) throws IOException
6      {
7
8          ServerSocket sock = null;
9          BufferedReader fromClient = null;
10         OutputStreamWriter toClient = null;
11         Socket client = null;
12
13         try
14         {
15             sock = new ServerSocket(4000);
16             System.out.println("Server Ready");
17             client = sock.accept();
18             System.out.println("Client Connected");
19             fromClient = new BufferedReader(new InputStreamReader(client.getInputStream()));
20             toClient = new OutputStreamWriter(client.getOutputStream());
21             String line;
22             while (true)
23             {
24                 line = fromClient.readLine();
25                 if ((line == null) || line.equals("bye")) break;
26                 System.out.println("Client [ " + line + " ]");
27                 toClient.write("Server [ "+ line + " ]\n");
28                 toClient.flush();
29             }
30             fromClient.close();
31             toClient.close();
32             client.close();
33             sock.close();
34             System.out.println("Client Disconnected");
35         }
36         catch (IOException ioe)
37         {
38             System.err.println(ioe);
39         }
40     }
41 }
42
```

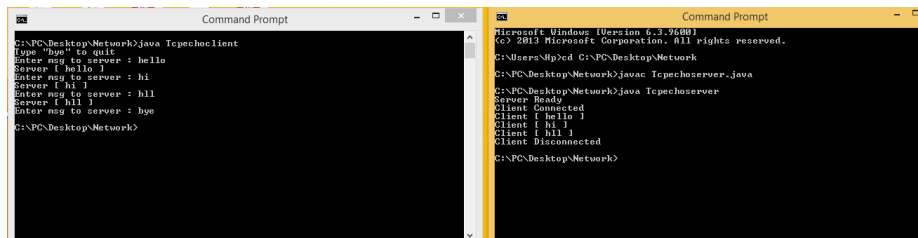
Figure 17: Program for TCP Echo Server – Tcpechoserver.java

TCP Echo Client Example

```
1  import java.net.*;
2  import java.io.*;
3  class Tcpechoclient
4  {
5      public static void main(String[] args) throws IOException
6      {
7          BufferedReader fromServer = null;
8          BufferedReader fromUser = null;
9          PrintWriter toServer = null;
10         Socket sock = null;
11
12         try
13         {
14             if (args.length == 0)
15                 sock = new Socket(InetAddress.getLocalHost(), 4000);
16             else
17                 sock = new Socket(InetAddress.getByName(args[0]), 4000);
18             fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));
19             fromUser = new BufferedReader(new InputStreamReader(System.in));
20             toServer = new PrintWriter(sock.getOutputStream(), true);
21             String Usermsg;
22             System.out.println("Type \"bye\" to quit");
23             while (true)
24             {
25                 System.out.print("Enter msg to server : ");
26                 Usermsg = fromUser.readLine();
27                 if (Usermsg == null || Usermsg.equals("bye"))
28                 {
29                     toServer.println("bye"); break;
30                 }
31                 else
32                     toServer.println(Usermsg);
33                 Srvmsg = fromServer.readLine();
34                 System.out.println(Srvmsg);
35             }
36             fromUser.close();
37             fromServer.close();
38             toServer.close();
39             sock.close();
40         }
41         catch (IOException ioe)
42         {
43             System.err.println(ioe);
44         }
45     }
46 }
47
```

Figure 18: Program for TCP Echo Client – Tcpechoclient.java

TCP Echo Client-Server Output



```
C:\PC\Desktop\Network>java TcpEchoClient
Type "bye" to quit
Enter msg to server : hello
Server [ hello ]
Enter msg to server : hi
Server [ hi ]
Enter msg to server : hll
Server [ hll ]
Enter msg to server : bye
C:\PC\Desktop\Network>
```

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Jip>cd C:\PC\Desktop\Network
C:\PC\Desktop\Network>javac TcpEchoServer.java
C:\PC\Desktop\Network>java TcpEchoServer
Server Ready
Client Connected
Client [ hello ]
Client [ hi ]
Client [ hll ]
Client Disconnected
C:\PC\Desktop\Network>
```

Figure 19: Output for TCP Echo Client-Server

RESULT: Thus data from client to server is echoed back to the client to check reliability/noise level of the channel.

The UDP Protocol

User Datagram Protocol (UDP)

- TCP is designed for reliable transmission of data.
- If data is lost or damaged in transmission, TCP ensures that the data is resent. However, this reliability comes at a price of speed.
- The UDP is an alternative transport layer protocol for sending data over IP that is very quick, but not reliable.
- Java's implementation of UDP is split into two classes: DatagramPacket and DatagramSocket.
- The DatagramPacket class stuffs bytes of data into UDP packets called datagrams and lets you unstuff datagrams that you receive.
- A DatagramSocket sends as well as receives UDP datagrams.
- To send data, you put the data in a DatagramPacket and send the packet using a DatagramSocket.
- To receive data, you take a DatagramPacket object from a DatagramSocket and then inspect the contents of the packet.

Acknowledgements

Text Books

- Learning Network Programming with Java by Richard M Reese.
- Java Network Programming, 4th Edition by Elliotte Rusty Harold.
- Head First Java: A Brain-Friendly Guide 3rd Edition.
- Data Communications and Networking (McGraw-Hill Forouzan Networking) 4th Edition.

Lab

- ChainLab, Software Engineering, LUT School of Engineering Science, LUT University, Lappeenranta, Finland.

The End