

LABORATORY MANUAL



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING U19CS504 – COMPUTER NETWORKS LABORATORY BATCH: 2019-2023

	File No :FRB/ U19CS504 /01	Revision no:01	Date: 03.11.2021
	Prepared by	Verified by	Approved by
Name of the Faculty member	Dr.A.Prithiviraj, ASP / CSE Dr.G.Kirubasri, AP / CSE Prof.V.Vinodhini, AP / CSE Prof.R.Priyadharshini, AP / CSE	Dr D Balamurugan, ASP / CSE	Dr B Sathiyabhama, HOD / CSE
Signature			

COLLEGE VISION

To become an institute of great repute, in the fields of Science, Applied Science, Engineering, Technology and Management studies, by offering a full range of programmes of global standard to foster research, and to transform the students into globally competent personalities

COLLEGE MISSION

- To offer Graduate, Post-graduate, Doctoral and other value-added programmes beneficial for the students
- To establish state-of-the-art facilities and resources required to achieve excellence in teaching-learning, and supplementary processes
- To provide Faculty and Staff with the required qualification and competence and to provide opportunity to upgrade their knowledge and skills
- To motivate the students to pursue higher education, appear for competitive exams, and other value added programmes for their holistic development
- To provide opportunity to the students and bring out their inherent talent
- To establish Centres of excellence in the emerging areas of research
- To have regular interaction with the Industries in the area of R & D, and offer consultancy, training and testing services
- To offer Continuing education, and Non-formal vocational education programmes that are beneficial to the society

Computer Science and Engineering - Programme Educational Objectives

Sona College of Technology, Computer Science and Engineering program will prepare its graduates to,

1. Work productively as successful Computer professionals in diverse career paths including supportive and leadership roles on multidisciplinary teams or be active in higher studies,
2. Communicate effectively, recognize and incorporate societal needs and constraints in their professional endeavors, and practice their profession with high regard to ethical responsibilities,
3. Engage in life-long learning and to remain current in their profession to foster personal and organizational growth.

Computer Science and Engineering - Programme Specific Outcomes

Programme Specific Outcomes - On completion of the B.E (Computer Science & Engineering) degree the graduates will be able to

- Apply standard Software Engineering practices and strategies in real-time software project development using open-source programming environment or commercial environment to deliver quality product for the organization success
- Design and develop computer programs/computer-based systems in the areas related to algorithms, networking, web design, cloud computing, IoT and data analytics of varying complexity
- Acquaint with the contemporary trends in industrial/research settings and thereby innovate novel solutions to existing problems

LIST OF EXPERIMENTS

1. Simulation of HTTP protocol using TCP Socket.
2. Programs using TCP and UDP Sockets (like getting date and time from server, Chat application, etc...)
3. Programs using RMI.
4. Network analysis using TCP Dump, Netstat, Trace Route tools.
5. Simulating a simple LAN using CISCO Packet tracer.
6. Simulating an organization LAN with multiple subnets using CISCO Packet tracer.
7. Simulation of a web server based network using CISCO Packet tracer.
8. Simulation of smart home network with IoT devices using CISCO Packet tracer.
9. Network topology configuration using ns2.
10. Packet sniffing and traffic analysis using WIRESHARK.

TABLE OF CONTENTS

S. No	TITLE	PAGE NO
1	INTRODUCTION TO SOCKET PROGRAMMING IN JAVA	1
2	RMI (Remote Method Invocation)	22
3	Simulation of Error Correction Codes	34
4	Probing the Internet (netstat, Traceroute)	35
5	CISCO Packet tracer	39
6	Introduction to NS2	42
7	WIRESHARK INTRODUCTION	48

Computer Science and Engineering - Programme Outcomes

Programme Outcomes describe what students are expected to know or be able to do by the time of graduation from the programme. On Completion of Computer Science and Engineering programme, the graduates will be able to

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

GENERAL INSTRUCTIONS

- Get the observation signed from the lab in charge in the first 15 minutes.
- Observation should contain aim and program for the corresponding experiment for the day.
- Students should proceed with program thereafter.
- Student should attend the viva voce simultaneously.
- Students should complete the program within their stipulated time.
- Record work of the program should be submitted in the next week.
- Students should maintain strict silence within the lab.

1. INTRODUCTION TO SOCKET PROGRAMMING IN JAVA

SOCKET PROGRAMMING AND JAVA.NET CLASS

A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called java.net, to enable the rapid development of network applications. Key classes, interfaces, and exceptions in java.net package simplifying the complexity involved in creating client and server programs are:

The Classes

1. ContentHandler
2. DatagramPacket
3. DatagramSocket
4. DatagramSocketImpl
5. HttpURLConnection
6. InetAddress
7. MulticastSocket
8. ServerSocket
9. Socket
10. SocketImpl
11. URL
12. URLConnection
13. URLEncoder
14. URLStreamHandler

The Interfaces

1. ContentHandlerFactory
2. FileNameMap
3. SocketImplFactory
4. URLStreamHandlerFactory

Exceptions

1. BindException
2. ConnectException
3. MalformedURLException
4. NoRouteToHostException
5. ProtocolException
6. SocketException
7. UnknownHostException
8. UnknownServiceException

TCP/IP SOCKET PROGRAMMING

The two key classes from the java.net package used in creation of server and client programs are:

- ServerSocket
- Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it. Figure 13.5 illustrates key steps involved in creating socket-based server and client programs.

A simple Server Program in Java

The steps for creating a simple server program are:

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

Socket Programming 351

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
```

```
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();
```

```
Send to client: os.writeBytes("Hello\n");
```

5. Close socket:

```
client.close();
```

An example program illustrating creation of a server socket, waiting for client request, and then responding to a client that requested for connection by greeting it is given below:

Program 1

```
// SimpleServer.java: A simple server program.

import java.net.*;

import java.io.*;

public class SimpleServer {

    public static void main(String args[]) throws IOException {

        // Register service on port 1254

        ServerSocket s = new ServerSocket(1254);

        Socket s1=s.accept(); // Wait and accept a connection

        // Get a communication stream associated with the socket

        OutputStream s1out = s1.getOutputStream();

        DataOutputStream dos = new DataOutputStream (s1out);

        // Send a string!

        dos.writeUTF("Hi there");

        // Close the connection, but not the server socket

        dos.close();

        s1out.close();

        s1.close();

    }

}
```

A simple Client Program in Java

The steps for creating a simple client program are:

1. Create a Socket Object:

```
Socket client = new Socket(server, port_id);
```

2. Create I/O streams for communicating with the server.

```
is = new DataInputStream(client.getInputStream());
```

```
os = new DataOutputStream(client.getOutputStream());
```

3. Perform I/O or communication with the server:

Receive data from the server: `String line = is.readLine();`

Send data to the server: `os.writeBytes("Hello\n");`

4. Close the socket when done:

```
client.close();
```

An example program illustrating establishment of connection to a server and then reading a message sent by the server and displaying it on the console is given below:

Program 2

// SimpleClient.java: A simple client program.

```
import java.net.*;
```

```
import java.io.*;
```

```
public class SimpleClient {
```

```
    public static void main(String args[]) throws IOException {
```

```
        // Open your connection to a server, at port 1254
```

```
        Socket s1 = new Socket("localhost",1254);
```

```
        // Get an input file handle from the socket and read the input
```

```
        InputStream s1In = s1.getInputStream();
```

```
        DataInputStream dis = new DataInputStream(s1In);
```

```
        String st = new String (dis.readUTF());
```

```
        System.out.println(st);
```

```
        // When done, just close the connection and exit
```

```
        dis.close();
```

```
        s1In.close();
```

```
s1.close();  
  
}  
  
}
```

Running Socket Programs Compile both server and client programs and then deploy server program code on a machine which is going to act as a server and client program, which is going to act as a client. If required, both client and server programs can run on the same machine. To illustrate execution of server and client programs, let us assume that a machine called mundroo.csse.unimelb.edu.au on which we want to run a server program as indicated below:

```
[raj@mundroo] java SimpleServer
```

The client program can run on any computer in the network (LAN, WAN, or Internet) as long as there is no firewall between them that blocks communication. Let us say we want to run our client program on a machine called gridbus.csse.unimelb.edu.au as follows:

```
[raj@gridbus] java SimpleClient
```

The client program is just establishing a connection with the server and then waits for a message. On receiving a response message, it prints the same to the console. The output in this case is:

Hi there which is sent by the server program in response to a client connection request.

It should be noted that once the server program execution is started, it is not possible for any other server program to run on the same port until the first program which is successful using it is terminated. Port numbers are a mutually exclusive resource. They cannot be shared among different processes at the same time.

UDP SOCKET PROGRAMMING

The previous two example programs used the TCP sockets. As already said, TCP guarantees the delivery of packets and preserves their order on destination. Sometimes these features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished by the UDP protocol which conveys datagram packets.

Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

The format of datagram packet is:

| Msg | length | Host | serverPort |

Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

The class DatagramPacket contains several constructors that can be used for creating packet object.

One of them is:

`DatagramPacket(byte[] buf, int length, InetAddress address, int port);`

This constructor is used for creating a datagram packet for sending packets of length to the specified port number on the specified host. The message to be transmitted is indicated in the first argument.

The key methods of DatagramPacket class are:

`byte[] getData()`

Returns the data buffer.

`int getLength()`

Returns the length of the data to be sent or the length of the data received.

`void setData(byte[] buf)`

Sets the data buffer for this packet.

`void setLength(int length)`

Sets the length for this packet.

The class DatagramSocket supports various methods that can be used for transmitting or receiving data a datagram over the network. The two key methods are:

`void send(DatagramPacket p)`

Sends a datagram packet from this socket.

`void receive(DatagramPacket p)`

Receives a datagram packet from this socket.

A simple UDP server program that waits for client's requests and then accepts the message (datagram) and sends back the same message is given below. Of course, an extended server program can manipulate client's messages/request and send a new message as a response.

Program 3

// UDPServer.java: A simple UDP server program.

```
import java.net.*;

import java.io.*;

public class UDPServer {

    public static void main(String args[]){

        DatagramSocket aSocket = null;

        if (args.length < 1) {

            System.out.println("Usage: java UDPServer <Port Number>");

            System.exit(1);

        }

        try {

            int socket_no = Integer.valueOf(args[0]).intValue();

            aSocket = new DatagramSocket(socket_no);

            byte[] buffer = new byte[1000];

            while(true) {

                DatagramPacket request = new DatagramPacket(buffer,

                    buffer.length);

                aSocket.receive(request);

                DatagramPacket reply = new DatagramPacket(request.getData(),

                    request.getLength(),request.getAddress(),

                    request.getPort());

                aSocket.send(reply);
```

```

    }
}

catch (SocketException e) {

System.out.println("Socket: " + e.getMessage());

}

catch (IOException e) {

System.out.println("IO: " + e.getMessage());

}

finally {

if (aSocket != null)

aSocket.close();

}

}

}

```

A corresponding client program for creating a datagram and then sending it to the above server and then accepting a response is listed below.

Program 4

// UDPClient.java: A simple UDP client program.

```

import java.net.*;

import java.io.*;

public class UDPClient {

public static void main(String args[]){

// args give message contents and server hostname

DatagramSocket aSocket = null;

if (args.length < 3) {

```



```

System.out.println(
    "Usage: java UDPClient <message> <Host name> <Port number>");
System.exit(1);
}
try {
    aSocket = new DatagramSocket();
    byte [] m = args[0].getBytes();
    InetAddress aHost = InetAddress.getByName(args[1]);
    int serverPort = Integer.valueOf(args[2]).intValue();
    DatagramPacket request =
        new DatagramPacket(m, args[0].length(), aHost, serverPort);
    aSocket.send(request);
    byte[] buffer = new byte[1000];
    DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
    aSocket.receive(reply);
    System.out.println("Reply: " + new String(reply.getData()));
}
catch (SocketException e) {
    System.out.println("Socket: " + e.getMessage());
}
catch (IOException e) {
    System.out.println("IO: " + e.getMessage());
}
finally {
    if (aSocket != null)

```

```
aSocket.close();  
  
}  
  
}  
  
}
```

MATH SERVER

It is time to implement a more comprehensive network application by using the socket programming APIs you have learned so far.

The basic math interface is shown as follows:

Program 5

// MathService.java: A basic math interface.

```
public interface MathService {  
  
    public double add(double firstValue, double secondValue);  
  
    public double sub(double firstValue, double secondValue);  
  
    public double div(double firstValue, double secondValue);  
  
    public double mul(double firstValue, double secondValue);  
  
}
```

The implementation of this interface is not related to any network operation. The following code shows a very simple implementation of this interface:

Program 6

// PlainMathService.java: An implementation of the MathService interface.

```
public class PlainMathService implements MathService {  
  
    public double add(double firstValue, double secondValue) {  
  
        return firstValue+secondValue;  
  
    }  
  
    public double sub(double firstValue, double secondValue) {  
  
        return firstValue-secondValue;  
  
    }  
  
}
```

```

    }

    public double mul(double firstValue, double secondValue) {
        return firstValue * secondValue;
    }

    public double div(double firstValue, double secondValue) {
        if (secondValue != 0)
            return firstValue / secondValue;
        return Double.MAX_VALUE;
    }
}

```

The implementation of the MathServer is quite straightforward, which looks pretty similar to the echo server mentioned previously. The difference is that the MathServer have to consider the specific protocol defined by the math server and client communication. The program uses a very simple protocol operator:

first_value:second_value. It is the math server's responsibility to understand this protocol and delegate to the proper methods such as add, sub, mul, or div.

Program 7

// MathServer.java : An implementation of the MathServer.

```

import java.io.*;

import java.net.*;

public class MathServer{
    protected MathService mathService;

    protected Socket socket;

    Socket Programming 357

    public void setMathService(MathService mathService) {
        this.mathService = mathService;
    }
}

```

```

public void setSocket(Socket socket) {
    this.socket = socket;
}

public void execute() {
    try {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        // read the message from client and parse the execution
        String line = reader.readLine();
        double result = parseExecution(line);
        // write the result back to the client
        BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream()));
        writer.write(""+result);
        writer.newLine();
        writer.flush();
        // close the stream
        reader.close();
        writer.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

// the predefined protocol for the math operation is

```

```

// operator:first value:second value
protected double parseExecution(String line)
throws IllegalArgumentException {
    double result = Double.MAX_VALUE;
    String [] elements = line.split(":");
    if (elements.length != 3)
        throw new IllegalArgumentException("parsing error!");
    double firstValue = 0;
    double secondValue = 0;
    try {
        firstValue = Double.parseDouble(elements[1]);
        secondValue = Double.parseDouble(elements[2]);
    }
    catch(Exception e) {
        throw new IllegalArgumentException("Invalid arguments!");
    }
    switch (elements[0].charAt(0)) {
        case '+':
            result = mathService.add(firstValue, secondValue);
            break;
        case '-':
            result = mathService.sub(firstValue, secondValue);
            break;
        case '*':
            result = mathService.mul(firstValue, secondValue);

```

```

break;

case '/':

result = mathService.div(firstValue, secondValue);

break;

default:

throw new IllegalArgumentException("Invalid math operation!");

}

return result;

}

public static void main(String [] args)throws Exception{

int port = 10000;

if (args.length == 1) {

try {

port = Integer.parseInt(args[0]);

}

catch(Exception e){

}

}

System.out.println("Math Server is running...");

// create a server socket and wait for client's connection

ServerSocket serverSocket = new ServerSocket(port);

Socket socket = serverSocket.accept();

// run a math server that talks to the client

MathServer mathServer = new MathServer();

mathServer.setMathService(new PlainMathService());

```

```

mathServer.setSocket(socket);

mathServer.execute();

System.out.println("Math Server is closed...");

}

}

```

A test client program that can access the math server is shown below:

Program 8

// MathClient.java: A test client program to access MathServer.

```

import java.io.*;

import java.net.Socket;

public class MathClient {

    public static void main(String [] args){

        String hostname = "localhost";

        int port = 10000;

        if (args.length != 2) {

            System.out.println("Use the default setting...");

        }

        else {

            Socket Programming 359

            hostname = args[0];

            port = Integer.parseInt(args[1]);

        }

        try {

            // create a socket

            Socket socket = new Socket(hostname, port);

```

```

// perform a simple math operation "12+21"

BufferedWriter writer = new BufferedWriter(
new OutputStreamWriter(socket.getOutputStream()));

writer.write("+:12:21");

writer.newLine();

writer.flush();

// get the result from the server

BufferedReader reader = new BufferedReader(
new InputStreamReader(socket.getInputStream()));

System.out.println(reader.readLine());

reader.close();

writer.close();

}

catch (Exception e) {

e.printStackTrace();

}

}

}

```

URL ENCODING

It is very important that the Web can be accessed via heterogeneous platforms such as Windows, Linux, or Mac. The characters in the URL must come from a fixed subset of ASCII in order to maintain the interoperability between various platforms. Specifically, the capital letters A–Z, the lowercase letters a–z, the digits 0–9 and the punctuation characters. Encoding is very simple, any characters that are not ASCII numerals, letters, or the punctuation marks allowed are converted into bytes and each byte is written as a percentage sign followed by two hexadecimal digits. For example, the following program helps encode a query string if non-ASCII characters are present.

Program 9

// QueryStringFormatter.java: encodes a string with non-ASCII characters.

```
import java.io.UnsupportedEncodingException;

import java.net.URLEncoder;

public class QueryStringFormatter {

    private String queryEngine;

    private StringBuilder query = new StringBuilder();

    public QueryStringFormatter(String queryEngine) {

        this.queryEngine = queryEngine;

    }

    public String getEngine() {

        return this.queryEngine;

    }

    public void addQuery(String queryString, String queryValue)

        throws Exception {

        query.append(queryString+"="+

            URLEncoder.encode(queryValue,"UTF-8")+"&");

    }

    public String getQueryString(){

        return "?" + query.toString();

    }

    public static void main(String[] args) throws Exception {

        QueryStringFormatter formatter =

            new QueryStringFormatter("http://www.google.com.au/search");

        formatter.addQuery("newwindow","1");

    }

}
```

```

formatter.addQuery("q","Xingchen Chu & Rajkumar Buyya");

System.out.println(formatter.getEngine()+

formatter.getQueryString());

}

}

```

The output of this program is shown as follows:

```
http://www.google.com.au/search?newwindow=1&q=Xingchen+Chu+%26+Rajkumar+Buyya&
```

It can be seen that the whitespace has been encoded as “+” and the “&” has been encoded as “%26” the percentage sign following by its byte value. Other characters remain the same. These conversions have been performed by the URLEncoder class, which is part of the Java base class library and provides facilities for encoding strings (urls) in different formats. There is also an URLDecoder class that performs the reverse process. These two classes are useful not only for URLs but also for managing HTML form data.

Writing and Reading Data via URLConnection

Besides the socket and datagram introduced in the previous section, java.net package provides another useful class that can be used to write and read data between the server and the client: URLConnection. It is not possible to instantiate the URLConnection class directly. Instead you have to create the URLConnection via the URL object.

```
URLConnection connection = new URL("www.yahoo.com").openConnection();
```

Then the URLConnection class provides the getInputStream and getOutputStream methods that are very similar to the getInputStream and getOutputStream provided by the Socket class. The following example shows how to use the URLConnection and the URLEncoder class to send queries to the Yahoo search engine.

Program 10

```
// TextBasedSearchEngine.java:
```

```

import java.io.*;

import java.net.*;

public class TextBasedSearchEngine {

private String searchEngine;

```

```

public TextBasedSearchEngine(String searchEngine) {
    this.searchEngine = searchEngine;
}

public void doSearch(String queryString) {
    try {
        // open a url connection

        URL url = new URL(searchEngine);

        URLConnection connection = url.openConnection();

        connection.setDoOutput(true);

        // write the query string to the search engine

        PrintStream ps = new PrintStream(connection.getOutputStream());

        ps.println(queryString);

        ps.close();

        // read the result

        DataInputStream input =
            new DataInputStream(connection.getInputStream());

        String inputLine = null;

        while((inputLine = input.readLine()) != null) {

            System.out.println(inputLine);

        }

    }

    catch(Exception e) {

        e.printStackTrace();

    }

}

```

```

public static void main(String[] args) throws Exception{
    QueryStringFormatter formatter =
    new QueryStringFormatter("http://search.yahoo.com/search");
    formatter.addQuery("newwindow","1");
    formatter.addQuery("q","Xingchen Chu & Rajkumar Buyya");
    // search it via yahoo

    TextBasedSearchEngine search =
    new TextBasedSearchEngine(formatter.getEngine());
    search.doSearch(formatter.getQueryString());
}
}

```

Output:

The program first creates an encoded query string that can be used by the web application, then it utilizes the `URLConnection` class to send/receive data from the Yahoo search engine. The output in the entire HTML page will be something like this:

```

<!doctype html public "-//W3C/DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">

<html>

<head>

...

...

</html>

```

Exercises:

1. Write a client – server application using which the client sends a “hello” message to the server. In response to this message, the server replies the same “hello” message to the client. Likewise whatever the message the client sends, the server responds with the same message to the client.

2. Write a file transfer application in which the file server has a file repository. When the client connects to the server, the server asks the client whether the client wants to upload a file or download a file. According to the choice made by the client, the client will be able to either upload or download the file.
3. Write a file transfer application in which the file server has a file repository. When the client connects to the server, the server asks the client whether the client wants to upload a file or download a file. According to the choice made by the client, the client will be able to either upload or download the file. This application should use connectionless service of the transport layer.

2. RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

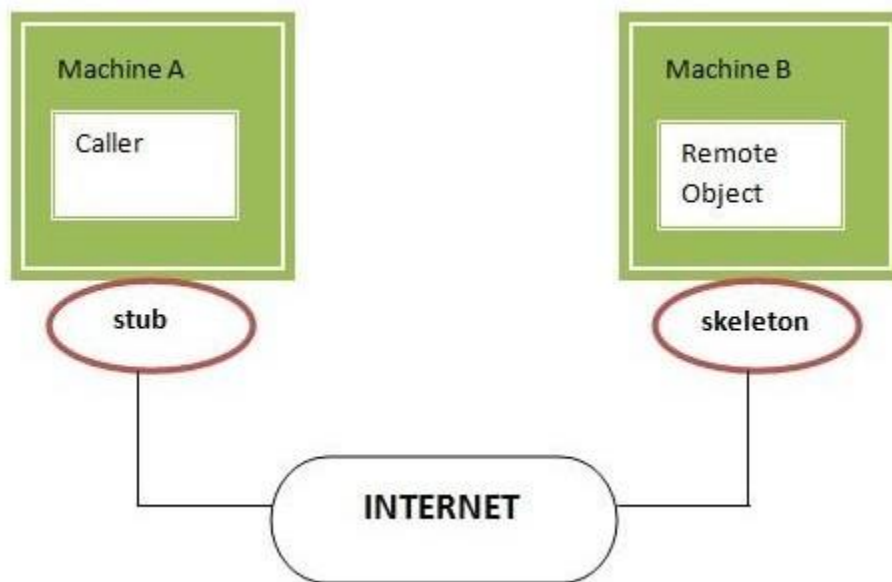
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

- 1. The application need to locate the remote method
- 2. It need to provide the communication with the remote objects, and
- 3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Steps to write the RMI program

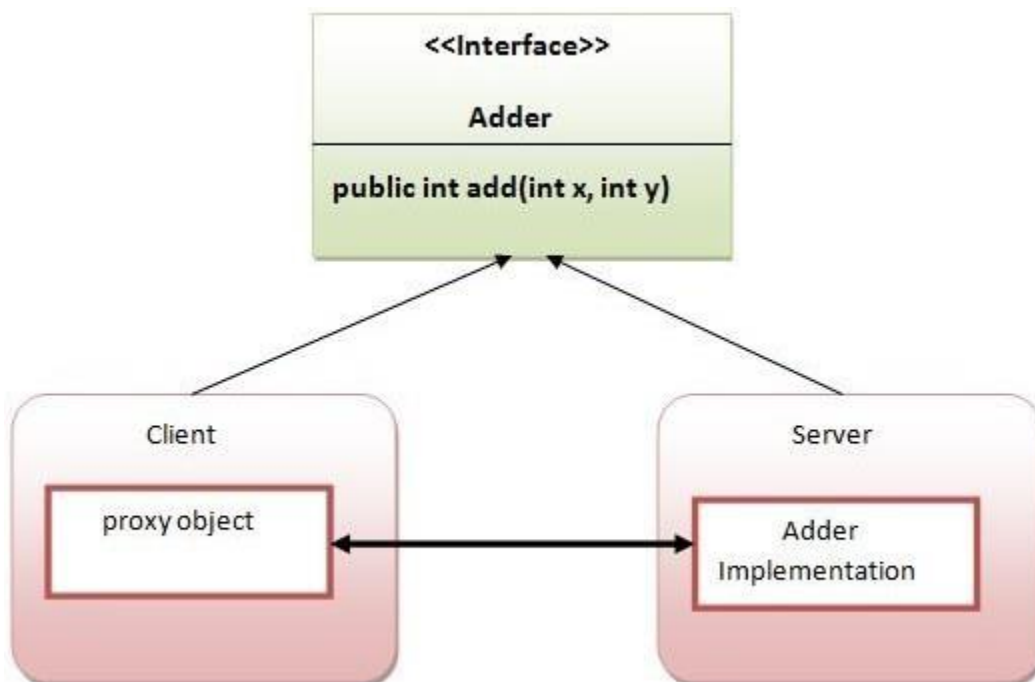
The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application

6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

1. **import** java.rmi.*;
2. **public interface** Adder **extends** Remote{
3. **public int** add(**int** x,**int** y)**throws** RemoteException;

4. }

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
1. import java.rmi.*;
2. import java.rmi.server.*;
3. public class AdderRemote extends UnicastRemoteObject implements Adder{
4. AdderRemote()throws RemoteException{
5. super();
6. }
7. public int add(int x,int y){return x+y;}
8. }
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

1. rmic AdderRemote

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

1. rmiregistry 5000

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object.
2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name.
3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name.
4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name.
5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
1. import java.rmi.*;
2. import java.rmi.registry.*;
3. public class MyServer{
4. public static void main(String args[]){
5. try{
6. Adder stub=new AdderRemote();
7. Naming.rebind("rmi://localhost:5000/sonoo",stub);
8. }catch(Exception e){System.out.println(e);}
9. }
10. }
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
1. import java.rmi.*;
2. public class MyClient{
```

```
3. public static void main(String args[]){  
4. try{  
5. Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");  
6. System.out.println(stub.add(34,4));  
7. }catch(Exception e){ }  
8. }  
9. }
```

[download this example of rmi](#)

1. For running **this** rmi example,
- 2.
3. 1) compile all the java files
- 4.
5. javac *.java
- 6.
7. 2)create stub and skeleton object by rmic tool
- 8.
9. rmic AdderRemote
- 10.
11. 3)start rmi registry in one command prompt
- 12.
13. rmiregistry 5000
- 14.
15. 4)start the server in another command prompt
- 16.
17. java MyServer
- 18.
19. 5)start the client application in another command prompt
- 20.
21. java MyClient

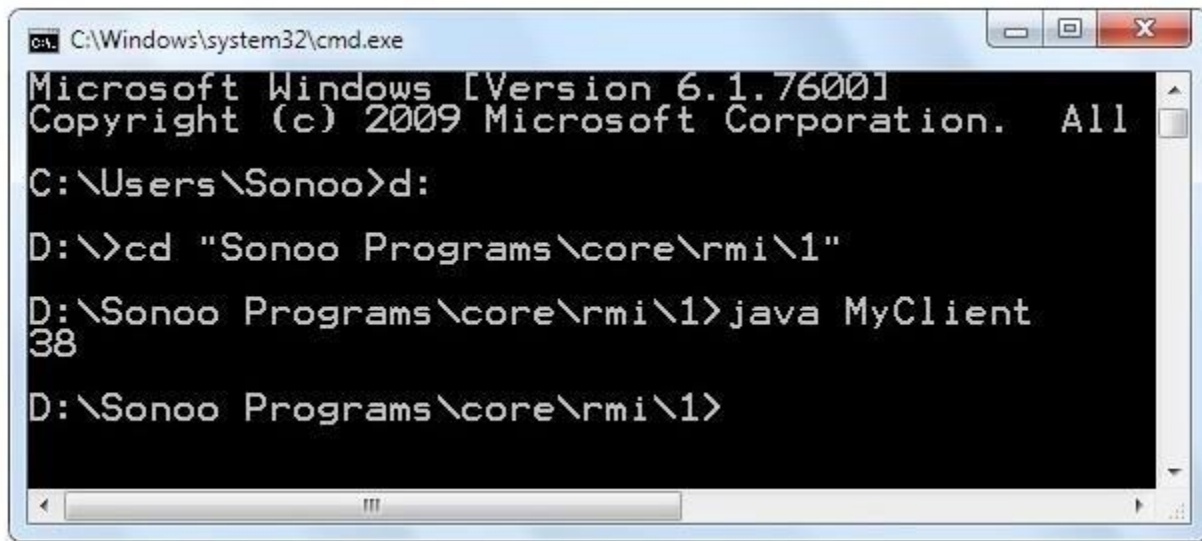
Output of this RMI example

```
C:\Windows\system32\cmd.exe - rmiregistry 5000
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>javac *.java
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```

```
C:\Windows\system32\cmd.exe - java MyServer
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyServer
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All
C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```

Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.




1) Create the table

First of all, we need to create the table in the database. Here, we are using Oracle10 database.

CUSTOMER400

TableDataIndexesModelConstraintsGrantsStatisticsUI DefaultsTriggersDependenciesSQL

QueryCount RowsInsert Row

EDIT	ACC_NO	FIRSTNAME	LASTNAME	EMAIL	AMOUNT
	67539876	James	Franklin	franklin1james@gmail.com	500000
	67534876	Ravi	Kumar	ravimalik@gmail.com	98000
	67579872	Vimal	Jaiswal	jaiswalvimal32@gmail.com	9380000
					row(s) 1 - 3 of 3

Download

2) Create Customer class and Remote interface

File: Customer.java

1. **package** com.javatpoint;
2. **public class** Customer **implements** java.io.Serializable{
3. **private int** acc_no;
4. **private String** firstname,lastname,email;
5. **private float** amount;
6. //getters and setters
7. }

Note: Customer class must be Serializable.

File: Bank.java

1. **package** com.javatpoint;
2. **import** java.rmi.*;
3. **import** java.util.*;
4. **interface** Bank **extends** Remote{
5. **public List**<Customer> getCustomers()**throws** RemoteException;
6. }

3) Create the class that provides the implementation of Remote interface

File: BankImpl.java

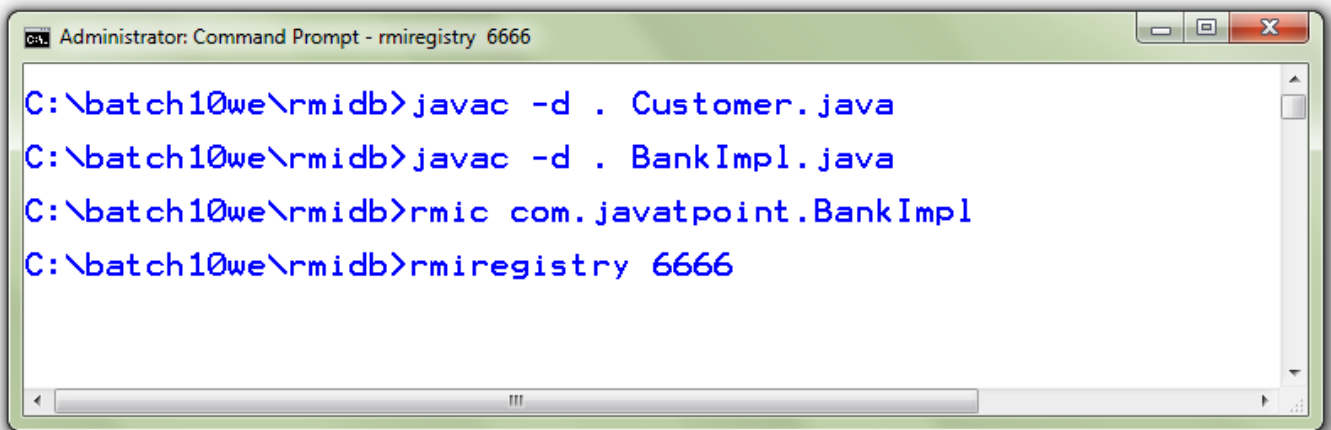
1. **package** com.javatpoint;

```

2. import java.rmi.*;
3. import java.rmi.server.*;
4. import java.sql.*;
5. import java.util.*;
6. class BankImpl extends UnicastRemoteObject implements Bank{
7. BankImpl()throws RemoteException{ }
8.
9. public List<Customer> getCustomers(){
10. List<Customer> list=new ArrayList<Customer>();
11. try{
12. Class.forName("oracle.jdbc.driver.OracleDriver");
13. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
    "oracle");
14. PreparedStatement ps=con.prepareStatement("select * from customer400");
15. ResultSet rs=ps.executeQuery();
16.
17. while(rs.next()){
18. Customer c=new Customer();
19. c.setAcc_no(rs.getInt(1));
20. c.setFirstname(rs.getString(2));
21. c.setLastname(rs.getString(3));
22. c.setEmail(rs.getString(4));
23. c.setAmount(rs.getFloat(5));
24. list.add(c);
25. }
26.
27. con.close();
28. }catch(Exception e){System.out.println(e);}
29. return list;
30. }//end of getCustomers()
31. }

```

4) Compile the class rmic tool and start the registry service by rmiregistry tool



```
Administrator: Command Prompt - rmiregistry 6666

C:\batch10we\rmidb>javac -d . Customer.java
C:\batch10we\rmidb>javac -d . BankImpl.java
C:\batch10we\rmidb>rmic com.javatpoint.BankImpl
C:\batch10we\rmidb>rmiregistry 6666
```

5) Create and run the Server

File: MyServer.java

1. **package** com.javatpoint;
2. **import** java.rmi.*;
3. **public class** MyServer{
4. **public static void** main(String args[])**throws** Exception{
5. Remote r=**new** BankImpl();
6. Naming.rebind("rmi://localhost:6666/javatpoint",r);
7. }}



```
Administrator: C:\Windows\system32\cmd.exe - java com.javatpoint.MyServer

C:\batch10we\rmidb>javac -d . MyServer.java
C:\batch10we\rmidb>java com.javatpoint.MyServer
```

6) Create and run the Client

File: MyClient.java

1. **package** com.javatpoint;
2. **import** java.util.*;
3. **import** java.rmi.*;
4. **public class** MyClient{
5. **public static void** main(String args[])**throws** Exception{
6. Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");
- 7.
8. List<Customer> list=b.getCustomers();
9. **for**(Customer c:list){
10. System.out.println(c.getAcc_no()+" "+c.getFirstname()+" "+c.getLastname()
11. +" "+c.getEmail()+" "+c.getAmount());
12. }
- 13.
14. }}

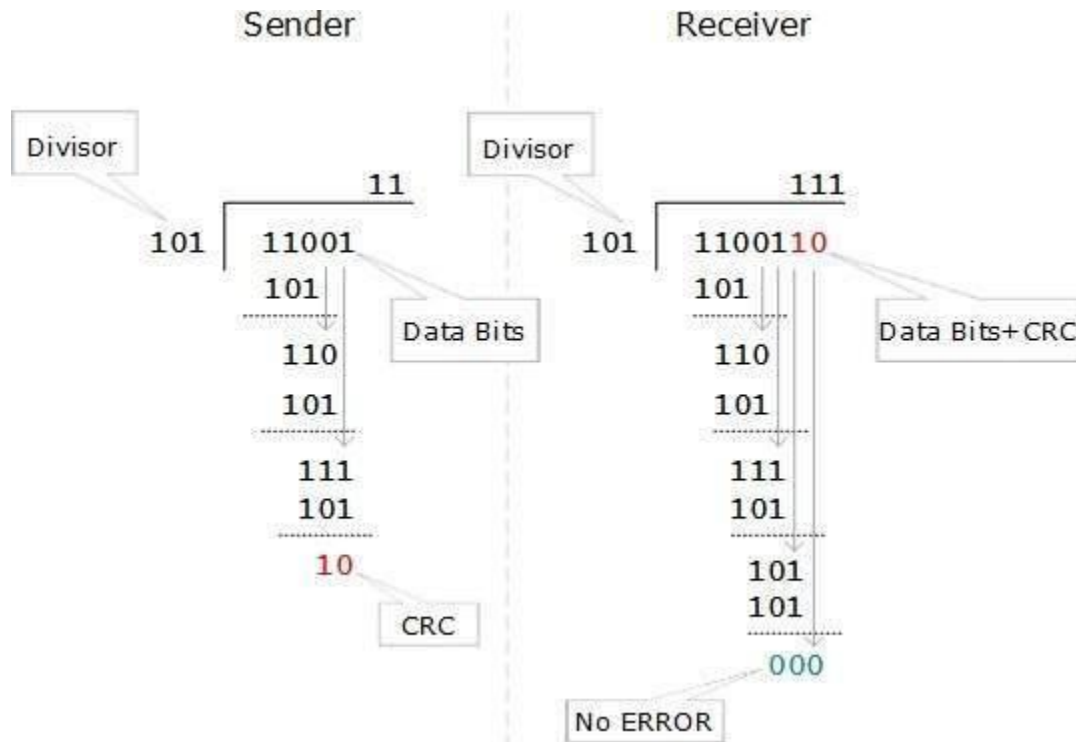
Exercise:

1. Write a client server application in which the server maintains a list of services such as CGPA calculation, data conversion (eg. °C to °F) and simple calculator. The client chooses the required service and sends the required number of parameters for that service as object and the server calculates the values and sends the result back to the client as an object. Use Remote Method Invocation (RMI) for this application.

3. Simulation of Error Correction Codes

Cyclic Redundancy Check

CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords.



At the other end, the receiver performs division operation on codewords using the same CRC divisor. If the remainder contains all zeros the data bits are accepted, otherwise it is considered as there some data corruption occurred in transit.

Error Correction

In the digital world, error correction can be done in two ways:

- **Backward Error Correction:** When the receiver detects an error in the data received, it requests back the sender to retransmit the data unit.
- **Forward Error Correction:** When the receiver detects some error in the data received, it executes error-correcting code, which helps it to auto-recover and to correct some kinds of errors.

The first one, Backward Error Correction, is simple and can only be efficiently used where retransmitting is not expensive. For example, fiber optics. But in case of wireless transmission retransmitting may cost too much. In the latter case, Forward Error Correction is used.

To correct the error in data frame, the receiver must know exactly which bit in the frame is corrupted. To locate the bit in error, redundant bits are used as parity bits for error detection. For example, we take ASCII words 7bitsdata7bitsdata, then there could be 8 kind of information we need: first seven bits to tell us which bit is error and one more bit to tell that there is no error.

For m data bits, r redundant bits are used. r bits can provide 2^r combinations of information. In m+r bit codeword, there is possibility that the r bits themselves may get corrupted. So the number of r bits used must inform about m+r bit locations plus no-error information, i.e. m+r+1.

$$2^r \geq m+r+1$$

Exercise

1. Machine A sends “1101011111” to machine B. To add the error detection and correction code machine A uses “10011” as generator polynomial. Write a java program to add the CRC code with the message at machine A and verify the appended CRC at machine B.

4. Probing the Internet (netstat, Traceroute)

Objective

In this exercise we investigate two applications of the Internet Control Message Protocol (ICMP)

1. PING uses ICMP to determine whether a host is reachable:
2. Traceroute uses ICMP to allow users to determine the route that an IP packet takes from a local host to a remote host.

Protocols Examined

- ICMP: Echo, Echo Reply, Time Exceeded messages.
- IP Time-to-Live
- PING application
- Traceroute application

Procedure:

PING

1. Prepare Wireshark for a packet capture.
2. Open a Command Prompt window. (In Windows XP, from the “All Programs” Menu, select “Accessories” and then “Command Prompt”).
3. Type “PING <website>”; Do NOT press ENTER.
4. Start Wireshark packet capture.
5. Press ENTER in Command Prompt window. You should obtain a series of replies
6. Stop packet capture when Command Prompt returns.
7. Save the contents in the Command Prompt window using a screen capture (Alt PrtSc)

Traceroute – Repeat the above steps by replacing step 3 with `tracert` command

Protocol Analysis Questions

To answer the following questions, start Wireshark and open the packet capture file created above.

1. *PING Protocols Captured.*

- Examine the protocol column in the top pane of the Wireshark window. You will find a series of ICMP packets. It is likely that these ICMP packets are preceded by a DNS query/response message pair.
- Identify the IP address returned in the DNS response message.

2. *ICMP Echo Request*

- Examine the IP packet that carries the first ICMP Echo Request. What is the destination IP address in the IP packet? What is the protocol type? What is the Time-to-Live?
- Next examine the ICMP message. What is the ICMP message type? What is the message identifier and sequence number?
- Highlight the data bytes carried in the request message. Note the corresponding character sequence in the third pane of the Wireshark window.

3. ICMP Echo Reply

- What are the source and destination addresses in the IP packet that carries the first ICMP Echo Reply? What are the protocol type and the Time-to-Live?
- Now examine the ICMP reply message. What is the ICMP message type? Compare the message identifier and sequence number in the reply message with the corresponding numbers in the request message?
- Highlight the data bytes in the reply message and compare the data sequence with that in the request message.

4. Repeat steps 2 and 3 for the remaining Echo request and Echo reply messages.

- How do the identifier and sequence numbers change with time?
- Does the data sequence in the request and reply messages change?
- Calculate the time that elapses between the sending of each Echo request and the receipt of the corresponding Echo reply. Compare the maximum, average, and minimum of the delays with those provided by the PING command.

5. Traceroute Protocols Captured.

- Examine the protocol column in the top pane of the Wireshark window. You will find a series of ICMP packets. Once again, it is likely that these ICMP packets are preceded by a DNS query/response message pair.
- Identify the IP address returned in the DNS response message.

6. ICMP Echo Request

- Determine the destination address in the IP packet that carries the first ICMP Echo Request. Compare to the address returned by the DNS response message. What are the protocol type and the Time-to-Live in the IP packet?
- Record the header of the IP packet for future reference.
- Examine the ICMP message. What is the ICMP message type? What are the message identifier and sequence number?
- How many data bytes are carried in the request message? Note the character sequence corresponding to the data bytes in the third pane of the Wireshark window.

7. ICMP Time Exceeded

- What are the source and destination addresses in the IP packet that carries the ICMP Time Exceeded message?
- Now examine the ICMP message. What is the ICMP message type?
- The ICMP Type, Code, and Checksum are followed by 32 zeros and then by the IP header of the ICMP Echo Request Message. Compare the returned IP header to the IP header noted in step 6.
- Does the ICMP message carry any additional data?
- Next compare the message identifier and sequence number in the Time Exceeded message with the corresponding numbers in the request message?

8. Repeat steps 6 and 7 for the remaining Echo request and Time Exceeded messages.

- Track the evolution of the TTL in the Echo request packets. Are there any repeated values of TTL? Is there a pattern to the repetitions?
- List the sequence of the source IP addresses in the packets that carry the ICMP Time Exceeded messages. Compare to the list provided by Traceroute.
- What is the received ICMP message when the ICMP Echo reply finally reaches the desired host?
- Calculate the time that elapses between the sending of each Echo request and the receipt of the corresponding Time-Exceeded message. Compare the delay values obtained with the results provided by the Traceroute command.

Conclusion Report:

Write a report on Probing tools based on your observation from the above queries

5. CISCO Packet Tracer

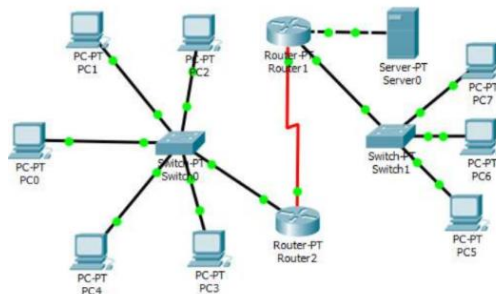
Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows users to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface. Packet Tracer makes use of a drag and drop user interface, allowing users to add and remove simulated network devices as they see fit. The software is mainly focused towards Certified Cisco Network Associate Academy students as an educational tool for helping them learn fundamental CCNA concepts. Previously students enrolled in a CCNA Academy program could freely download and use the tool free of charge for educational use.

Packet Tracer can be run on Linux, Microsoft Windows, and macOS. Similar Android and iOS apps are also available. Packet Tracer allows users to create simulated network topologies by dragging and dropping routers, switches and various other types of network devices. A physical connection between devices is represented by a 'cable' item. Packet Tracer supports an array of simulated Application Layer protocols, as well as basic routing with RIP, OSPF, EIGRP, BGP, to the extents required by the current CCNA curriculum. As of version 5.3, Packet Tracer also supports the Border Gateway Protocol.

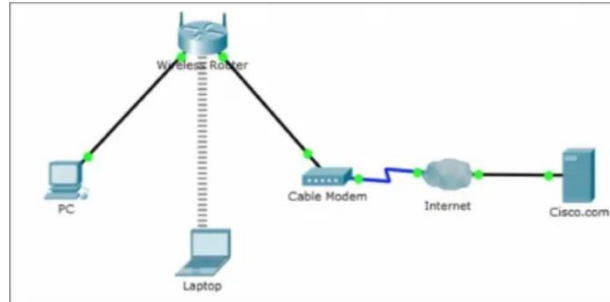
In addition to simulating certain aspects of computer networks, Packet Tracer can also be used for collaboration. As of Packet Tracer 5.0, Packet Tracer supports a multi-user system that enables multiple users to connect multiple topologies together over a computer network. Packet Tracer also allows instructors to create activities that students have to complete. Packet Tracer is often used in educational settings as a learning aid. Cisco Systems claims that Packet Tracer is useful for network experimentation.

Exercises:

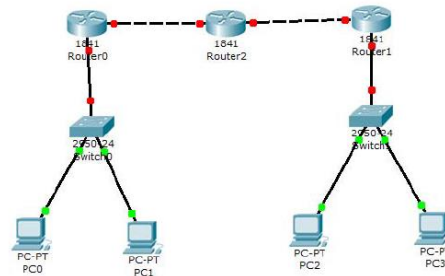
1. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



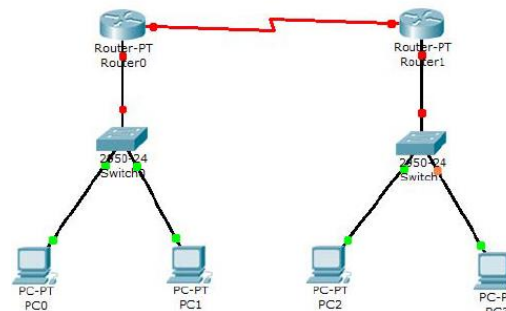
2. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



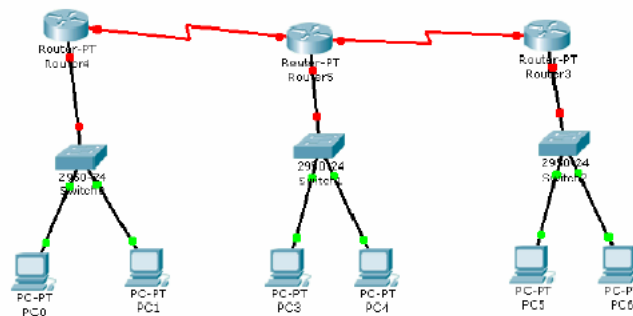
3. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



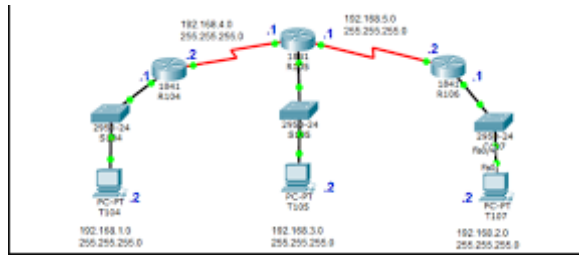
4. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



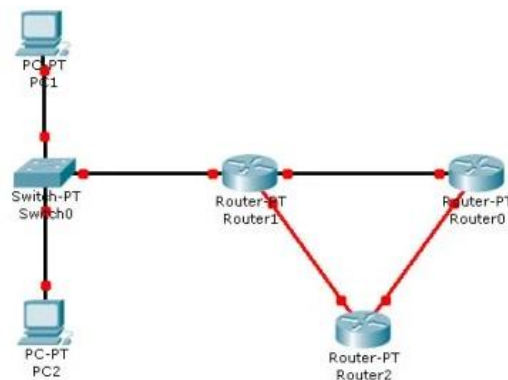
5. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



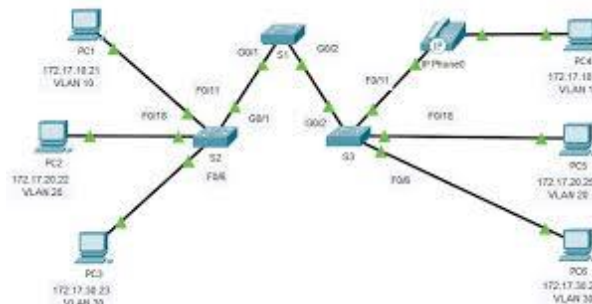
6. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



7. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



8. Simulate the creation of simple network topology using Cisco Packet Tracer. Configure the network devices. Test the connectivity between network devices.



6. Introduction to NS2

Network simulator is a piece of software or hardware that predicates the performance of a network without a real network being there. NS2 is a vital simulation tool for networks. It supports a number of algorithms for routing and queuing. It can set up packet traffic related to internet and measure a variety of parameters. NS2 is very helpful because it is very costly to verify viability of new algorithms, test architectures, check topologies etc. network simulator is a name for series of discrete event network simulator. Simulators are used in the simulation of routing protocols, and are heavily used in ad-hoc networking research, and support popular network protocols, offering simulation results for wired and wireless networks alike

Features of NS2

NS2 can be employed in most unix systems and windows. Most of the NS2 code is in C++. It uses TCL as its scripting language, Otcl adds object orientation to TCL. NS(version 2) is an object oriented, discrete event driven network simulator that is freely distributed and open source.

- Traffic Models: CBR, VBR, Web etc
- Protocols: TCP, UDP, HTTP, Routing algorithms, MAC etc
- Error Models: Uniform, bursty etc
- Misc: Radio propagation, Mobility models , Energy Models
- Topology Generation tools
- Visualization tools (NAM), Tracing

Structure of NS

- NS is an object oriented discrete event simulator
 - Simulator maintains list of events and executes one event after another
 - Single thread of control: no locking or race conditions
- Back end is C++ event scheduler
 - Protocols mostly
 - Fast to run, more control
 - Front end is OTCL

Creating scenarios, extensions to C++ protocols

fast to write and change

Platforms

It can be employed in most Unix systems(FreeBSD, Linux, Solaris) and Windows.

Source code

Most of NS2 code is in C++

Scripting language

It uses TCL as its scripting language OTcl adds object orientation to TCL.

Protocols implemented in NS2

Transport layer (Traffic Agent) – TCP, UDP

Network layer (Routing agent)

Interface queue – FIFO queue, Drop Tail queue, Priority queue

Logic link control layer – IEEE 802.2, AR

How to use NS2

Design Simulation – Determine simulation scenario

Build ns-2 script using tcl.

Run simulation

Simulation with NS2

Ø Define objects of simulation.

Ø Connect the objects to each other

Ø Start the source applications. Packets are then created and are transmitted through network.

Ø Exit the simulator after certain fixed time.

NS programming Structure

- Create the event scheduler
- Turn on tracing
- Create network topology
- Create transport connections
- Generate traffic
- Insert errors

Creating Event Scheduler

- Create event scheduler: set ns [new simulator]
 - Schedule an event: \$ns at <time> <event>
- Event is any legitimate ns/tcl function

\$ns at 5.0 “finish”

```
proc finish {} {
```

```
    global ns nf
```

```
    close $nf
```

```
    exec nam out.nam &
```

```
    exit 0
```

```
}
```

- Start Scheduler

\$ns run

Tracing and Animation

- Network Animator

```
set nf [open out.nam w]
```

\$ns namtraceall

```

$nf

proc finish {} {

global ns nf

close $nf

exec nam out.nam &

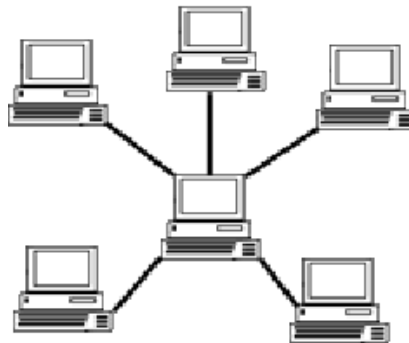
exit 0

}

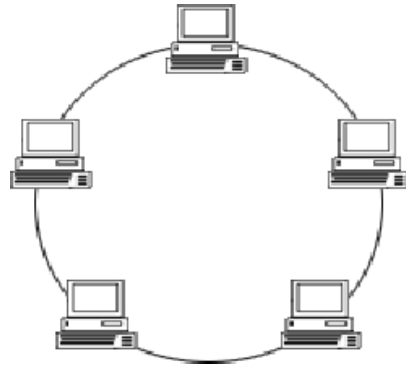
```

Exercises:

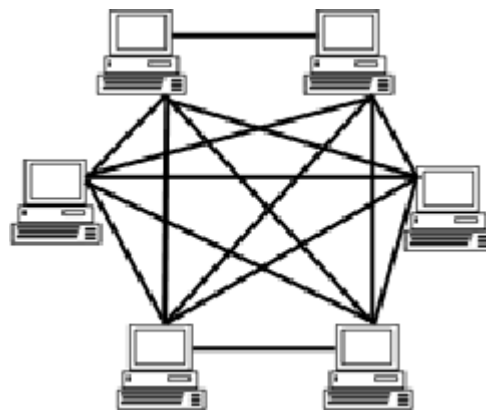
1. Simulate a network consisting of 5 nodes, numbered from 0 to 4, forming a star topology. The links have a 2Mbps bandwidth with 10ms delay. Send TCP packets from node 2 to node 4 and node 1 to node 3 at the rate of 1000 packets/sec using default packet size. Start transmission at 0.01. Finish the transmission at 15.000. Then run “nam” to view the results.



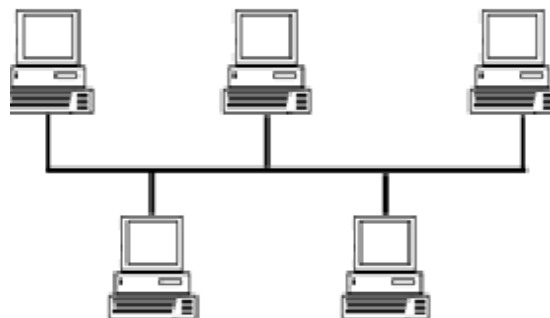
2. Simulate a network consisting of 5 nodes, numbered from 0 to 4, forming a ring topology. The links have a 5Mbps bandwidth with 10ms delay. Send TCP packets from node 0 to node 3 with the rate of 1000 packets/sec using default packet size. Start transmission at 0.01. Finish the transmission at 5.000. Then run “nam” to view the results.



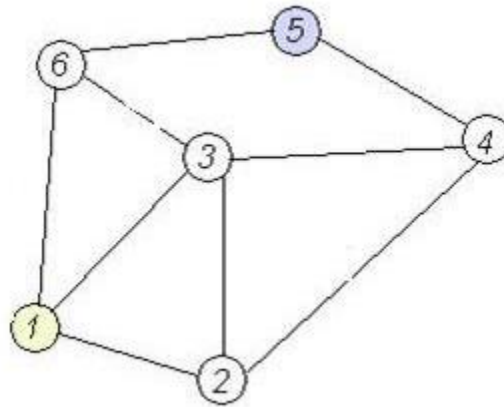
3. Simulate a network consisting of 5 nodes, numbered from 0 to 5, forming a mesh topology. The links have a 5Mbps bandwidth with 15ms delay. Send UDP packets from node 2 to node 4 and node 1 to node 5 with the rate of 1000 packets/sec using default packet size. Start transmission at 0.01. Finish the transmission at 5.000. Then run “nam” to view the results.



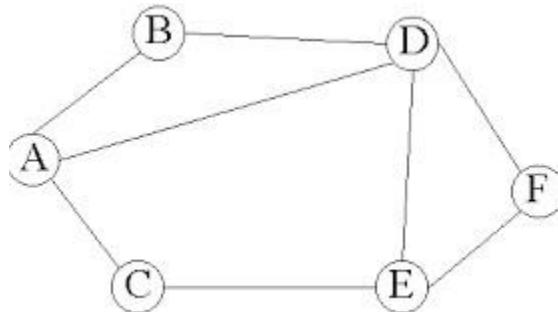
4. Simulate a network consisting of 5 nodes, numbered from 0 to 4, forming a bus topology. The links have a 12Mbps bandwidth with 10ms delay. Send TCP packets from node 2 to node 3 with the rate of 1000 packets/sec using default packet size. Start transmission at 0.01. Finish the transmission at 5.000. Then run “nam” to view the results.



5. Simulate a network consisting of n nodes. The links between the node 2 and node 5 should have a 10Mbps bandwidth with 5ms delay and the links between the node 1 and node 4 should have a 5Mbps bandwidth with 15ms delay. Send UDP/TCP packets at the rate of 100 packets/sec using default packet size. Apply Link State Routing Protocol and Distance vector routing for the given links and compare the performance of the routing protocols using “xgraph”



6. Simulate a network consisting of 6 nodes; the links that connects the network nodes should have a 2Mbps bandwidth with 10ms delay. Send TCP packets from node B to node E and Send UDP packets from the node A to node F with the rate of 1000 packets/sec using default packet size. Increase the transmission rate gradually until packet loss occurs due to congestion and check the transmission rate after the congestion occurs in both the paths. Start transmission at 0.01.Finish the transmission at 5.000. Then compare the performance of TCP and UDP during congestion period.



7. Wireshark

Introduction

Wireshark, previously known as Ethereal, is a full-function network analyzer that is available for free download at www.wireshark.org. In this lab, you will use Wireshark to sniff and analyze various types of traffic in order to become familiar with its usage and capabilities. One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Features

The following are some of the many features Wireshark provides:

- Available for **UNIX** and **Windows**.
- **Capture** live packet data from a network interface.
- **Open** files containing packet data captured with tcpdump/WinDump, Wireshark, and a number of other packet capture programs.
- **Import** packets from text files containing hex dumps of packet data.
- Display packets with **very detailed protocol information**.
- **Save** packet data captured.
- **Export** some or all packets in a number of capture file formats.
- **Filter packets** on many criteria.
- **Search** for packets on many criteria.
- **Colorize** packet display based on filters.
- Create various **statistics**.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet, and so all upper layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

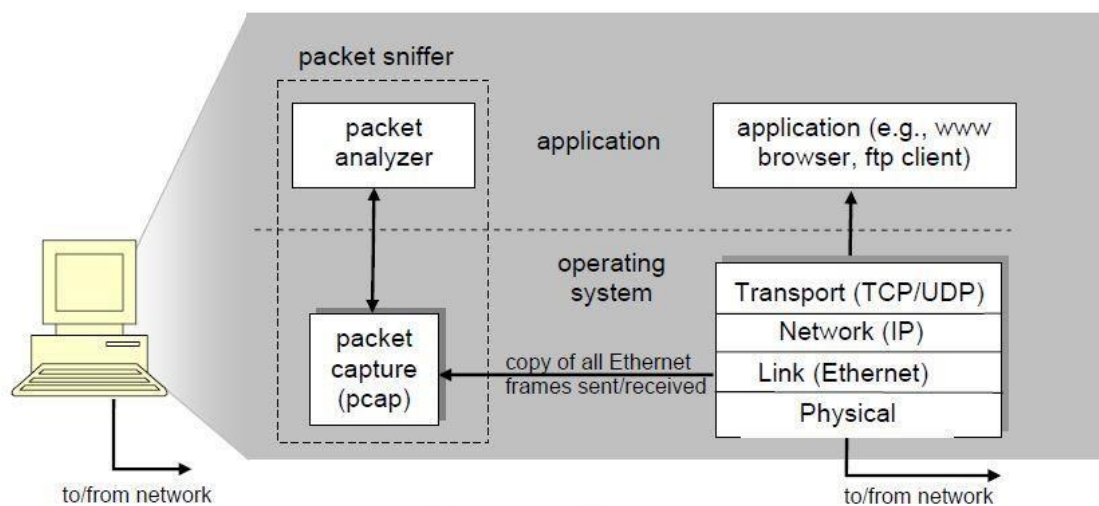


Figure 1: Packet sniffer structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must “understand” the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string “GET,” “POST,” or “HEAD,”.

Getting Wireshark

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the libpcap or WinPCap packet capture library. The libpcap software will be installed for you, if it is not installed within your operating system, when you install Wireshark

Running Wireshark

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 2 will be displayed. Initially, no data will be displayed in the various windows

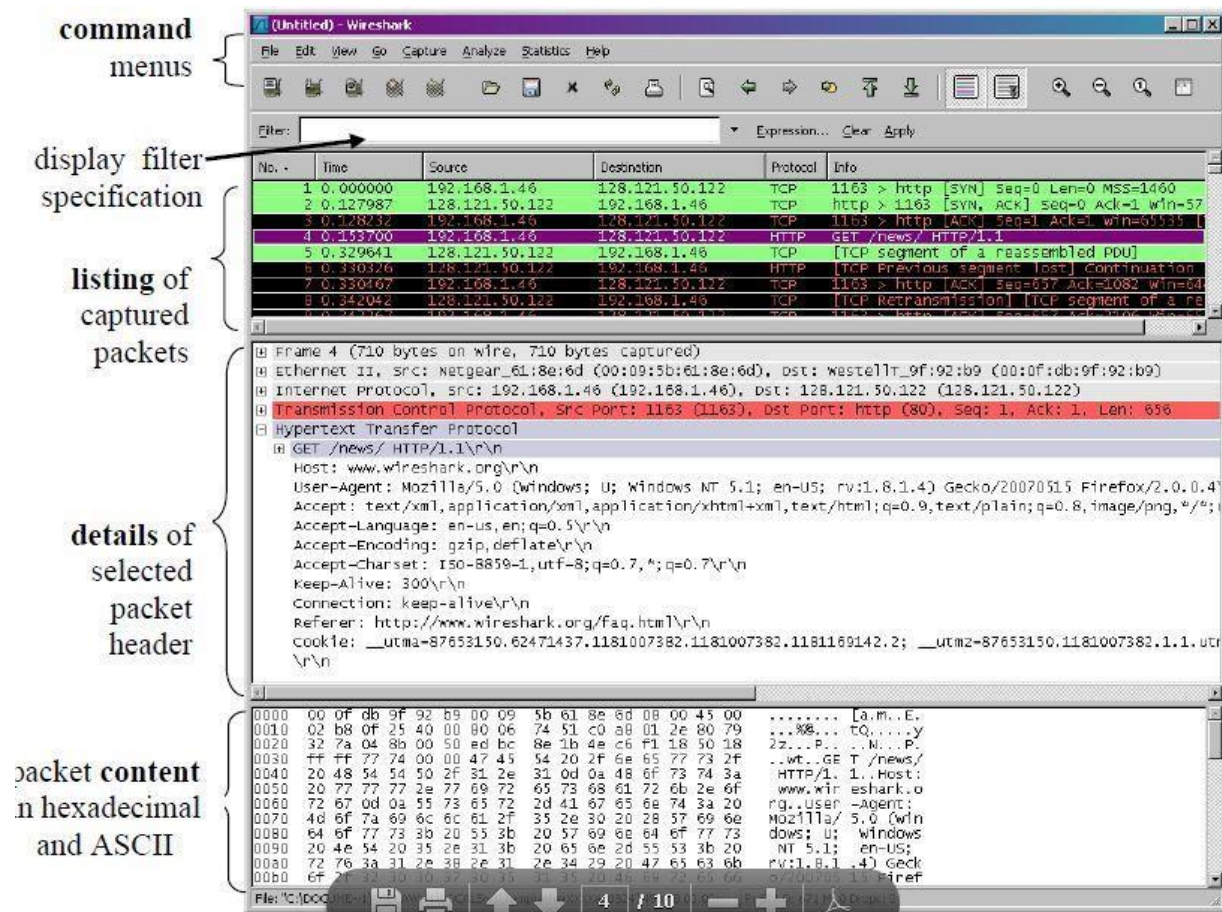


Figure 2: Wireshark Graphical User Interface

Exercises:

1. From your node please open the browser and do web surfing of your choice. Use “wireshark” to analyze the web traffic and show the communication from client to server in the application layer. Set the Ethernet card in promiscuous mode and capture all the packets that are transmitted through your node and do an analysis at each layer
2. From your node please open the browser and do web surfing of your choice. Use “Wireshark” to analyze the web traffic and show the communication from client to server in the application layer. Set the Ethernet card in promiscuous mode and capture all the packets that are transmitted through your node and do an analysis at each layer.
3. Make a Google search for “apple”, use Wireshark to analyze the web traffic and show the communication from client to server in the application layer.
4. Analyze the web traffic which passes through port 21, transfer files and highlight the message transfer which is captured.
5. Analyze the web traffic which passes through port 80, visit our college website and highlight the message transfer which is captured.