Ex. No. : 1 **Retrieving Data using URL**

Procedure

URLs

A URL identifies the location of a resource on the Internet. It specifies the protocol used to access a server (e.g., FTP, HTTP), the name of the server, and the location of a file on that server.

The ***syntax of a URL*** is:

protocol://username@hostname:port/path/filename?query#fragment

The ***protocol*** is another word for what was called the scheme of the URI. In a URL, the protocol part can be file, ftp, http, https, gopher, news, telnet, wais, or various other strings.

The ***hostname*** part of a URL is the name of the server that provides the resource you want, such as www.oreilly.com or utopia.poly.edu. It can also be the server's IP address, such as 204.148.40.9 or 128.238.3.21.

The ***username*** is an optional username for the server. The ***port number*** is also optional. It's not necessary if the service is running on its default port (port 80 for HTTP servers).

The ***path*** points to a particular directory on the specified server. The path is relative to the document root of the server, not necessarily to the root of the file system on the server.

The ***filename*** points to a particular file in the directory specified by the path. It is often omitted—in which case, it is left to the server's discretion what file, if any, to send.

The ***query string*** provides additional arguments for the server. It's commonly used only in http URLs, where it contains form data for input to programs running on the server.

The ***fragment*** references a particular part of the remote resource. If the remote resource is HTML, the fragment identifier names an anchor in the HTML document. If the remote resource is XML, the fragment identifier is an XPointer.

ABSOLUTE

Absolute URLs refer to the entire URL.

http://www.someplace.com/page.html

No matter where the link originates or who types it into the address bar, this URL can be located precisely (absolutely). Ther is no doubt as to the server and the page being requested. Any references to an outside site must use an ABSOLUTE URL.

RELATIVE

Relative URLs refer to a file or page on the same server.

page.html
images/image.gif
../info/page2.html

RELATIVE URLs can only be used for files kept on the originating server. The above example shows the 3 types of Relative URLs.

page.html	is a file located in the "root" directory.
image.gif	is a file located in a subdirectory called "images" off the root directory.
../	is used when the subdirectory is not part of the default root directory. It states : Take the main root directory, find the folder called "info", then access the file called "page2.html"

It depends on the server and file setup as to which type to use. If there is any doubt, using an ABSOLUTE URL will work every time.

Algorithm

1. Create a URL.
2. Retrieve the URLConnection object.
3. Set output capability on the URLConnection.
4. Open a connection to the resource.
5. Get an output stream from the connection.
6. Write to the output stream.
7. Close the output stream.

Parsing a URL

The URL class provides several methods that let you query URL objects. You can get the protocol, authority, host name, port number, path, query, filename, and reference from a URL using these accessor methods:

getProtocol

Returns the protocol identifier component of the URL.

getHost

Returns the host name component of the URL.

getPort

Returns the port number component of the URL. The getPort method returns an integer that is the port number. If the port is not set, getPort returns -1.

getPath

Returns the path component of this URL.

getFile

Returns the filename component of the URL. The getFile method returns the same as getPath, plus the concatenation of the value of getQuery, if any.

getContent

Gets the contents of this URL.

getContentType

Returns A string containing the value of the "content type" header in the request.

getContentLength

Returns An integer representing the "content length" header contained in the request.

getDate

Returns the value of the date header field.

getLastModified

Returns the value of the last-modified header field.

getURL

Returns the value of this URLConnection's URL field.

Ex. No.: 2a Implementation of Socket Programming using TCP/IP

Procedure:

TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network (either an intranet or an extranet).

TCP/IP is a two-layer program. The higher layer, Transmission Control Protocol, manages the assembling of a message or file into smaller packets that are transmitted over the Internet and received by a TCP layer that reassembles the packets into the original message. The lower layer, Internet Protocol, handles the address part of each packet so that it gets to the right destination.

TCP/IP uses the client/server model of communication in which a computer user (a client) requests and is provided a service (such as sending a Web page) by another computer (a server) in the network.

TCP Ports

The port numbers from 0 to 255 are well-known ports, and the use of these port numbers in your application is highly discouraged. Many well-known services you use have assigned port numbers in this range.

Opening and closing a socket

To create a TCP/IP socket, we use the socket( ) API.

```
SOCKET hSock = socket( AF_INET, SOCK_STREAM, IPPROTO_IP );
```

If the socket API fails, it will return a value of INVALID_SOCKET, otherwise it will return a descriptor value that we will need to use when referring to this socket. Once we are done with the socket we must remember to call the closesocket( ) API.

```
closesocket( hSock );
```

<u>TCP Server</u>

The steps to get a server up and running are shown below (read from top to bottom). This is how our sample code is written, so it's a good idea to get familiar with the process.

The steps for creating a simple server program are:

1. Open the Server Socket:

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

1. Wait for the Client Request:

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

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

```
DataInputStream is = new DataInputStream(client.getInputStream());
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

1. Perform communication with client

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

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

1. Close socket:

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

TCP Client

Now let's take a look at what steps the client needs to take in order to communicate with the server.

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());

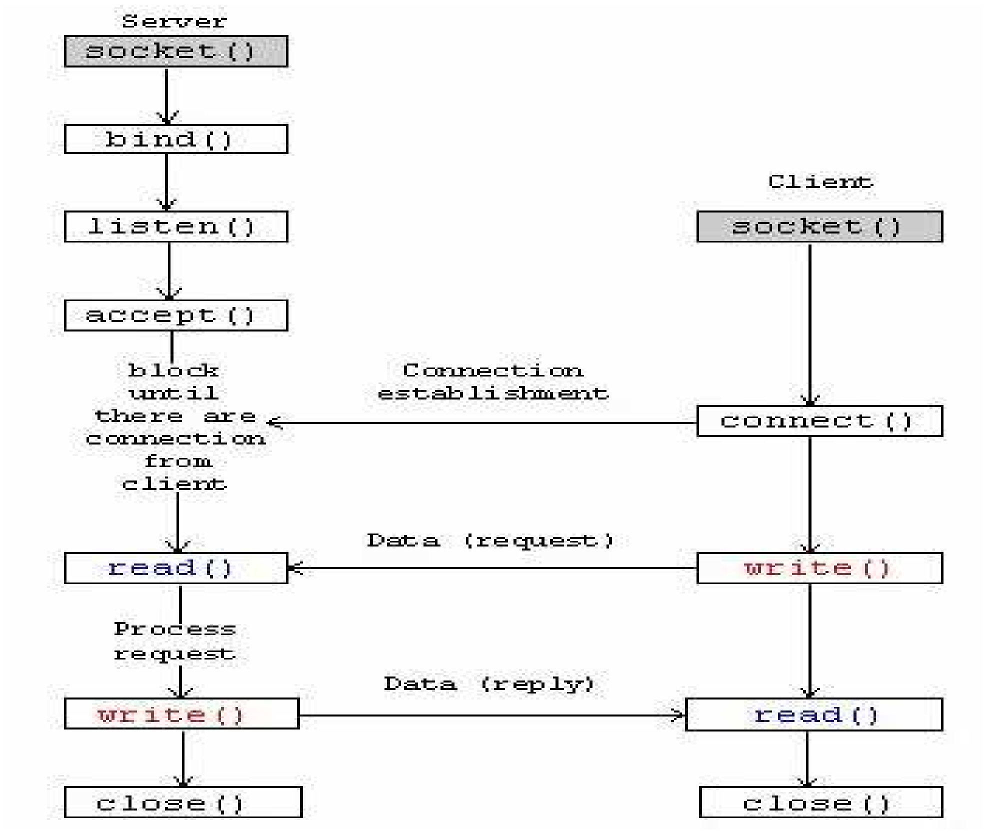
3. Perform I/O or communication with the server:

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();



Ex. No.: 2a Implementation of Socket Programming using UDP

Procedure

UDP

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP.

Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end.

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 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, extended server program can manipulate client's messages/request and send a new message as a response.

Algorithm

Server:

1. Create a datagram socket.

2. Get the message from the user, using Input Stream.

3. Convert the message to byte data type and store it in a byte array.

4. Send the byte array, length of the array, destination and the port number of the client as a datagram packet using datagram socket created.

5. If the message from the user equals "end", quit the program.

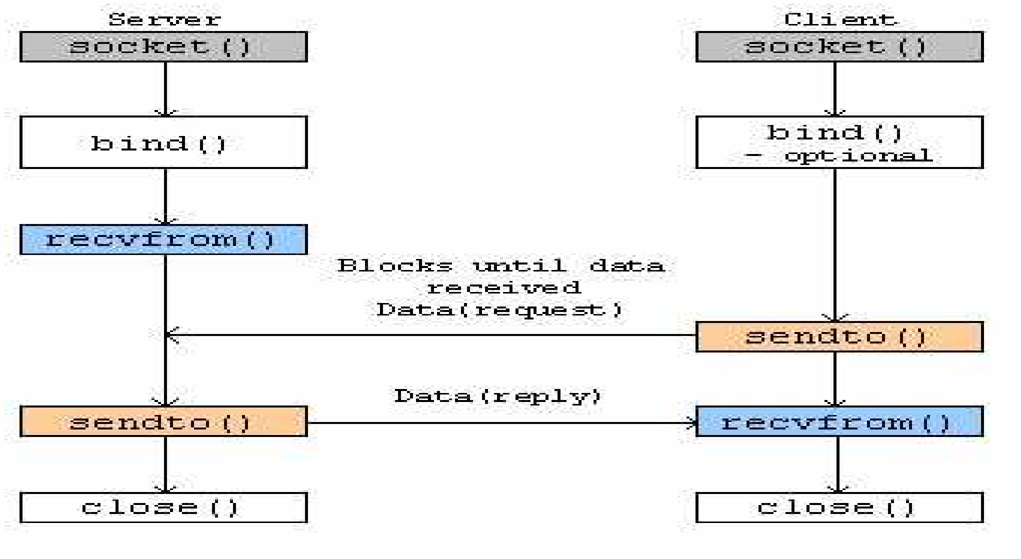
Client:

1. Create a Datagram Socket and Datagram Packet.

2. Using the socket, receive the Datagram Packet from the server (receive() method).

3. The Datagram Packet contains the message from the server.

4. If the message from the user equals "end", quit the program.

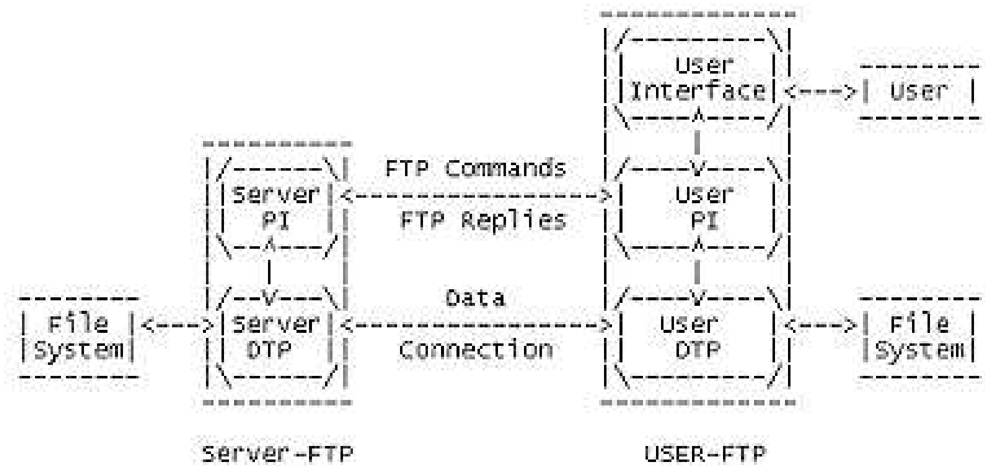


Ex.No.: 3 **Implementation of FTP**

Procedure

FTP

File Transfer Protocol (FTP), a standard Internet protocol, is the simplest way to exchange files between computers on the Internet. FTP is commonly used to transfer Web page files from their creator to the computer that acts as their server for everyone on the Internet. It's also commonly used to download programs and other files to your computer from other servers.



In the model, the user-protocol interpreter initiates the control connection. The control connection follows the Telnet protocol. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the control connection.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.).

The user- DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters.

It should be noted that the data port need not be in the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted that the data connection may be used for simultaneous sending and receiving.

Algorithm

1) Start your FTP client session.
2) Specify the name of the remote system to which you want to send the file.
3) Tell the remote system your user name for the remote server.
4) Tell the remote system your password for the remote server.
5) Locate the directory on Their Co server from which you want to transfer the file.
6) Navigate to the directory on the local server to which you want to transfer the file.
7) Specify file type, ASCII or BINARY. Default file type is ASCII.
8) Request a file transfer from the remote server system to the client system.
9) When finished, Exit from FTP.

The server MUST close the data connection under the following conditions:

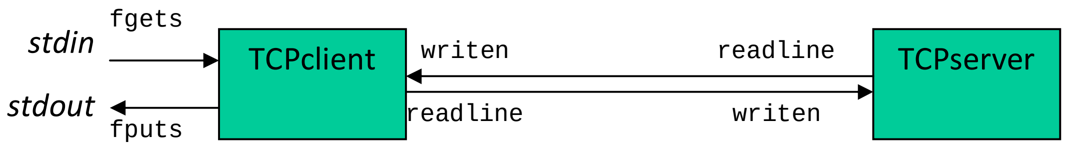
1) The server has completed sending data in a transfer mode that requires a close to indicate EOF.
2) The server receives an ABORT command from the user.
3) The port specification is changed by a command from the user.
4) The control connection is closed legally or otherwise.
5) An irrecoverable error condition occurs.

Ex.No: 4a **Implementation of echo**

A simple method for measuring the round-trip delay (RTT in seconds); of a segment would be: the sender places a timestamp in the segment and the receiver returns that timestamp in the corresponding ACK segment.

When the ACK segment arrives at the sender, the difference between the current time and the timestamp is the RTT. To implement this timing method, the receiver must simply reflect or echo selected data (the timestamp) from the sender's segments. This idea is the basis of the "TCP Echo" and "TCP Echo reply" options.

To use the TCP Echo and Echo Reply options, a TCP must send a TCP Echo option in its own SYN segment and receive a TCP Echo option in a SYN segment from the other TCP.



1. The Client reads a line of text from its standard input and writes the line to the server.
2. The server reads the line from its network input and echoes the line back to the client.
3. The client reads the echoed line and prints it on its standard output.

Simple TCP/IP echo server

This program demonstrates a simple TCP/IP server. It will accept a connection from a client application, receive one line of text, echo that line back to the client and close the connection.

Programming Issues

This program illustrates the classic process for a TCP/IP server program. Summarized, it is as follows:

- Create a socket with a call to socket()
- Create and initialize a socket address strucure with the IP address set to INADDR_ANY (the server will listen on any IP address) and the port to whichever one you wish to use.
- Call bind() to bind the socket address to the socket.
- Call listen() to indicate that this is a passive socket, that we want to accept incoming requests rather than make outgoing ones.
- Enter a loop in which we:
    1. Call accept() to wait for an incoming connection
    2. Service the request on our new connection
    3. close() our connection, and continue the loop to wait for a new one

Simple TCP/IP echo client

This program demonstrates a simple TCP/IP client.

Programming Issues

This program illustrates the classic process for a TCP/IP client program. Summarized, it is as follows:

- Create a socket with a call to socket()
- Create and initialize a socket address structure with the address family (AF_INET here), the server's remote IP address, and the port the server is listening on.
- Actively connect to the server by calling connect()
- Communicate with the server, and (in this case) allow the server to perform the active close.

Ex.no. 4b Implementation of Ping

Ping is a basic Internet program that allows a user to verify that a particular IP address exists and can accept requests.

Ping is used diagnostically to ensure that a host computer the user is trying to reach is actually operating. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specified interface on the network and waiting for a reply. Ping can be used for troubleshooting to test connectivity and determine response time.

Ping operates by sending Internet Control Message Protocol (ICMP) *echo request* packets to the target host and waiting for an ICMP response. In the process it measures the time from transmission to reception (*round-trip time*) and records any packet loss.

A successful ping implicitly indicates that a working path exists both to and from the destination. Ping is actually indicating the following when it is successful:

- There is a functioning path from the source to the destination.
- The IP device with the destination IP address is up.
- There is a functioning path from the destination back to the source.

An unsuccessful ping may have failed because:

- there is no available path to the destination,
- the destination is down, or
- there is no return path from the destination.

Syntax:

Ping hostname (or) ipaddress

Ex.no. 4c **Implementation of talk**

Procedure:

Talk is a visual communication program which copies lines from your terminal to that of another user. It is a UNIX Command communication method for split screen textual conversations between two users. Talk requests an actively sent to a user on a particular machine, unlike ERC which is simply joined by interested user. It copies line from the terminal to that of another user.

Syntax: talk<username> @ <their machine>

Algorithm:

Server:

1. Create a server socket and client socket.
2. Use input stream to receive the message sent by the client.
3. Use the output stream to send a message to the client.
4. Wait for client to display this message and write a new one to be sent by the server through the socket.
5. Display message sent by the client.
6. If the message sent by the server or the message received from the client is "end", terminate the application.

Client:

1. Create a client socket that connects to the server using host name and port number.
2. Use input stream to receive the message sent by the server.
3. Use the output stream to send a message to the client.
4. Send a message to the server and listen the port to get a new message from server.
5. Display the message sent by the server.
6. If the message sent by the client or the message received from the server is "end", terminate the application

Ex.No.: 5 Implementation of Remote command Execution

Procedure:

It allows users to execute commands on remote systems. It involves how to run commands on a remote windows machine and get access to the standard output of the command.

Algorithm:

Server:

1. Create a datagram socket.
2. Wait for the client to request for commands to be executed.
3. Get the command from the client through the datagram socket.
4. Execute the command using the method exec() of class Runtime.
5. If the message received is exit, terminate the server.

Client:

1. Create a Datagram Socket and Datagram Packet.
2. Display the choice of commands to be executed.
3. Define the datagram packet with the command.
4. Send the datagram packet to the server through the socket.
5. To end the application, send the message exit to the remote server.

Ex.No.: 6 **Implementation of ARP**

Procedure:

Computers on an Ethernet-based Local Area Network, or LAN, communicate by sending data to each other in the form of packets. These packets have headers that identify the sender and the recipient of each one.

In an Ethernet LAN, traffic is addressed to a target Media Access Control address, or MAC address, as it's called. Internet traffic is sent using IP addresses to specify the sender and the recipient of the packets. This creates problems for traffic being sent to and from the Internet.

To translate a LAN IP address to its MAC address owner, we have something called the Address Resolution Protocol, or ARP. ARP is used to translate packets between IP address and MAC address.

ALGORITHM:

1. Import the packages java.io and java.util.
2. Define the class with 3 methods ARPget(), display() and search().
3. In ARPget() method, determine the IP addresses and physical address of the systems available in the network and store them in a hash table.
4. Use the method hasMoreTokens() of class StringTokenizer to separate the tokens- IP address, Physical address and type.
5. In search() method, with the IP address obtained from the user, use method containsKey() of hash table to find the matching IP address and print the corresponding Physical address.
6. If the IP address is not found in the hash table print , "No entry in ARP cache ".
7. In display() method, print all the machines IP address and Physical address using the methods hasMoreElements() and nextElement().

Ex.No.: 7 **Implementation of RARP**

Procedure:

RARP (Reverse Address Resolution Protocol) is a protocol by which a physical machine in a local area network can request to learn its IP address from a gateway server's Address Resolution Protocol (ARP) table or cache.

A network administrator creates a table in a local area network's gateway router that maps the physical machine (or Media Access Control - MAC address) addresses to corresponding Internet Protocol addresses.

When a new machine is set up, its RARP client program requests from the RARP server on the router to be sent its IP address. Assuming that an entry has been set up in the router table, the RARP server will return the IP address to the machine which can store it for future use.

<u>ALGORITHM:</u>

1. Import the packages java.io and java.util.
2. Define the class with 3 methods RARPget(), display() and search().
3. In RARPget() method, determine the IP addresses and physical address of the systems available in the network and store them in a hash table.
4. Use the method hasMoreTokens() of class StringTokenizer to separate the tokens- IP address, Physical address and type.
5. In search() method, with the Physical address obtained from the user, use method containsKey() of hash table to find the matching Physical address and print the corresponding IP address.
6. If the Physical address is not found in the hash table print , "No entry in RARP cache ".
7. In display() method, print all the machines IP address and Physical address using the methods hasMoreElements() and nextElement().

Ex.No.: 8 **Implementation of RMI / RPC**

Procedure:

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines*, possibly on different hosts.

RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

ALGORITHM:

1. Import the necessary packages.
2. Define a server class that implements the set of methods add(), sub(), mul() and div().
3. Create an interface to the server class that declares the methods add(), sub(), mul() and div().
4. Define a server class that implements the main method.
5. Create an object for the server class and bind it to the class using rebind() method.
6. Design a client interface to access the remote object.
7. Generate Stub and Skeleton.
8. Access the remote methods from the client.
9. Terminate the application.

Ex.No.: 9 Implementation of Shortest Path Routing Algorithm

Procedure:

Routing: control plane

- Computing paths the packets will follow
- Routers talking amongst themselves
- Individual router *creating* a forwarding table

Shortest-Path Routing

- Given: network topology with link costs
  - **c(x,y)**: link cost from node x to node y
  - Infinity if x and y are not direct neighbors
- Compute: least-cost paths to all nodes
  - From a given source u to all other nodes
  - **p(v)**: predecessor node along path from source to v

Dijkstra's Shortest-Path Algorithm

- Iterative algorithm
  - After k iterations, know least-cost path to k nodes
- **S**: nodes whose least-cost path definitively known
  - Initially, **S = {u}** where u is the source node
  - Add one node to S in each iteration
- **D(v)**: current cost of path from source to node v
  - Initially, **D(v) = c(u,v)** for all nodes v adjacent to u
  - … and **D(v) =** ∞ for all other nodes v
  - Continually update D(v) as shorter paths are learned

Dijkstra’s Algorithm:

```
  *Initialization:*
    S = {u}
    for all nodes v
      if v adjacent to u {
        D(v) = c(u,v)
      else D(v) = ∞
  *Loop*
    find w not in S with the smallest D(w)
    add w to S
    update D(v) for all v adjacent to w and not in S:
      D(v) = min{D(v), D(w) + c(w,v)}
  *until all nodes in S*
```

ALGORITHM:

1. Import the required packages.
2. Get the number of nodes in the network
3. Get the Source and Destination nodes.
4. Generate the Cost matrix for the given topology.
5. Use Dijkstra’s algorithm to find the shortest path between the two nodes using the Cost matrix.
6. Record the shortest route between the source and destination and display the path.
7. Terminate the application.

Ex.No.: 10 Implementation Of Sliding Window Protocol

Procedure:

Stop & Wait:

→ stream of bulk data

Data:

→ data can be pipelined

→ transmit window of date

→ donot worry about getting ack immediately

How do we handle errors?

Sender and receiver maintain – buffer space

Receiver window =1

Sender window=n

<u>ALGORITHM</u>:

1. Import packages java.io , java.net and java.lang.
2. Establish connection between the sliding window server and sliding window client using TCP connection.
3. Initialize the sender window size (SWS) and receiver window size(RWS).
4. Get the number of frames to be transferred from the sender.
5. From the receiver side, send acknowledgment for a group of frames.
6. At the sender side, for each frame sent, decrement the SWS and for each acknowledgement received increment the SWS based on the ACK number.
7. At the receiver side, for each frame received decrement the RWS and for each acknowledgment sent increment the RWS based on the ACK number.
8. Print all the frames received at the receiver.