

## **List of Experiments:-**

- 1(a). Study different types of Network cables (Copper and Fiber) and prepare cables (Straight and Cross) to connect Two or more systems. Use crimping tool to connect jacks. Use LAN tester to connect the cables.
- 1(b). Install and configure Network Devices: HUB, Switch and Routers. Consider both manageable and non-manageable switches. Do the logical configuration of the system. Set the bandwidth of different ports.
- 1(c). Install and Configure Wired and Wireless NIC and transfer files between systems in Wired LAN and Wireless LAN. Consider both adhoc and infrastructure mode of operation.
2. Work with the commands Ping, Tracert, Ipconfig, pathping, telnet, ftp, getmac, ARP, Hostname, Nbtstat, netdiag, and Nslookup
3. Use Sniffers for monitoring network communication (Ethereal)
4. Find all the IP addresses on your network. Unicast, Multicast, and Broadcast on your network.
5. Use Packet tracer software to build network topology and configure using Distance vector routing protocol.
6. Use Packet tracer software to build network topology and configure using Link State routing protocol.
7. Using JAVA RMI Write a program to implement Basic Calculator
8. Implement a Chatting application using JAVA TCP and UDP sockets.
9. Hello command is used to know whether the machine at the other end is working or not. Echo command is used to measure the round trip time to the neighbour. Implement Hello and Echo commands using JAVA.
10. Use Ethereal tool to capture the information about packets.
11. Install Network Simulator 2/3. Create a wired network using dumbbell topology. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.
12. Create a static wireless network. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.
13. Create a mobile wireless network. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.

## **Additional List of Experiments:-**

1. Implement File Transfer in Client & Server Using TCP/IP.
2. Implement Address Resolution Protocol Using JAVA.
3. Implement download a webpage using Java

## **Experiment-1(A)**

**Aim:- Study different types of Network cables(Copper and Fiber) and prepare cables (Straight and Cross) to connect two or more systems. Use crimping tool to connect jacks. Use LAN testers to connect the cables.**

### **Types of Network Cables:-**

There are three types of Network cables: **coaxial, twisted pair, and fiber-optic cabling**. In modern Local Area Networks, twisted pair cable cabling is the most popular type of cabling, but fiber-optic cabling usage is increasing, especially in high-end networks. Coaxial cabling is usually used for cable over the Internet. Let's explain all three *Types of Ethernet Cables* in detail.

#### **1: Coaxial Cabling:-**

A coaxial cable has an internal conductor that runs down the middle of the cable. The conductor is surrounded by a layer of insulation which is then surrounded by another carrying conductor shield, which makes this type of cable resistant to external obstruction. This type of cable comes in two types – thinnet and thicknet. Each type has a maximum transmission speed of 10 Mbps. Coaxial cables were previously used in computer networks, but are now replaced by twisted pair cables.

A **single-core** coaxial cable uses a single central metal (usually copper) conductor, while a **multi-core** coaxial cable uses multiple thin strands of metal wires. The following image shows both types of cable.

#### **2: Twisted-pair Cabling:-**

The twisted-pair has four pairs of wires. These wires are twisted almost to each other to reduce crosstalk and external interference. This type of cabling is common in current LANs.

Twisted pair cables can be used for telephone and network cables. It comes in two versions: **UTP (Unshielded Twisted-Pair)** and **STP (Shielded Twisted-Pair)**. The difference between these is that the STP cable has an additional layer of protection to protect the data from external interference.

#### **Similarities and differences between STP and UTP cables**

- Both STP and UTP can transmit data over 10Mbps, 100Mbps, 1Gbps, and 10Gbps.
- STP cables are more expensive than UTP cables because they contain more material.
- Both cables use the same RJ-45 (registered slot) module connector.
- STP provides more noise and EMI resistance than UTP cable.
- The maximum length of the two cable segments is 100 meters or 328 feet.
- The two cables can accommodate a maximum of 1,024 nodes per segment.

The following image shows both types of twisted-pair cable.

#### **3: Fiber-optic Cabling:-**

This type of cable uses optical fibers to transmit data in the form of a light signal. The cables have fiberglass strands surrounded by cladding material.

The core is wrapped in cladding; The cladding is wrapped in a buffer, and the buffer is wrapped in a jacket.

1. The key is to transmit information signals in the form of light.
2. The cladding reflects light back into the core.
3. The buffer prevents light from leaking.
4. This jacket protects the cable from physical damage.

Fiber optic cables are fully immune to EMI and RFI. This cable can transmit data over long distances at maximum speed. It can transmit 40 km of data at 100Gbps.

Fiber optic cables use light to transmit data. It reflects light from one point to another. There are two types of fiber optic cables based on how much light they transmit at a given time; SMF to MMF.

This type of cable can support longer cable lengths than other cable types (a few miles). The cable has no electromagnetic interference. As you can see, this cable method has many advantages over other methods, but its main disadvantage is that it is more expensive.

There are two types of fiber-optic cables:

- **Single-mode fiber (SMF)** – uses only one light beam to transmit information. Used for longer distances.
- **Multi-mode fiber (MMF)** – uses multiple light beams to transmit data. Less expensive than SMF.

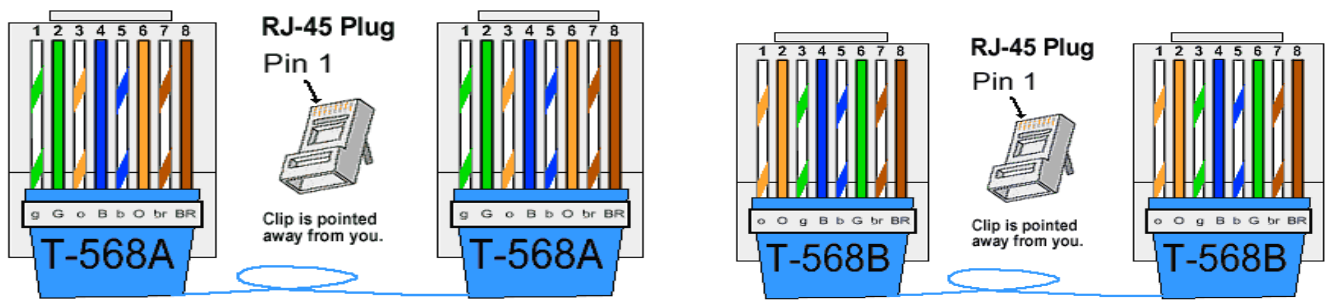
### **prepare cables (Straight and Cross):-**

**Components:** RJ-45 connector, Crimping Tool, Twisted pair Cable

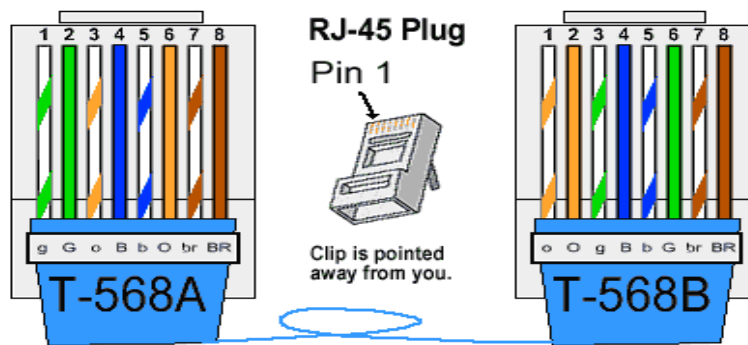
**Procedure: To do these practical following steps should be done:**

1. Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be very careful at this point, as to not nick or cut into the wires, which are inside. Doing so could alter the characteristics of your cable, or even worse render it useless. Check the wires, one more time for nicks or cuts. If there are any, just whack the whole end off, and start over.
2. Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand. You do not want the wires to become untwisted down inside the jacket. Category 5 cable must only have ½ of an inch of ‘untwisted’ wire at the end. Otherwise it will be ‘out of spec’. At this point, you obviously have A LOT more than ½ of an inch of un-twisted wire.
3. You have 2 end jacks, which must be installed on your cable. If you are using a pre-made cable, with one of the ends whacked off, you only have one end to install – the crossed over end. Below are two diagrams, which show how you need to arrange the cables for each type of cable end. Decide at this point which end you are making and examine the associated picture below.

**Diagram show you how to prepare Straight wired connection**



**Diagram show you how to prepare Cross wired connection**



### **LAN testers:-**

A **LAN cable tester** is a tool that is used for testing LAN cables. It has the capability to find certain problems associated with the cable. Any defect can easily be detected with this small but very useful instrument. Generally, a LAN cable tester consists of a source of electric current, a voltmeter and switching matrix that is used to connect the source and the voltmeter to the contact points.

Usually, the testing process is done in two stages. The first stage is called the open test, and this stage makes sure that every planned connections is properly set. The second, called the short test, helps find unintended connections. The LAN cable testers are not only used for testing LAN cables, but for telecommunication and data communication cables as well. In short, a LAN cable tester is a tool that offers a lot and is a must-have network testing tool.

## **Experiment-1(B)**

**Aim:- Install and configure Network Devices: HUB, Switch and Routers. Consider both manageable and non-manageable switches. Do the logical configuration of the system. Set the bandwidth of different ports.**

### **Hub:**

An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or optic fiber Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer(layer1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

### **Switch:**

A network switch connects devices within a network (often a local area network, or LAN) and forwards data packets to and from those devices. Unlike a router, a switch only sends data to the single device it is intended for (which may be another switch, a router, or a user's computer), not to networks of multiple devices.

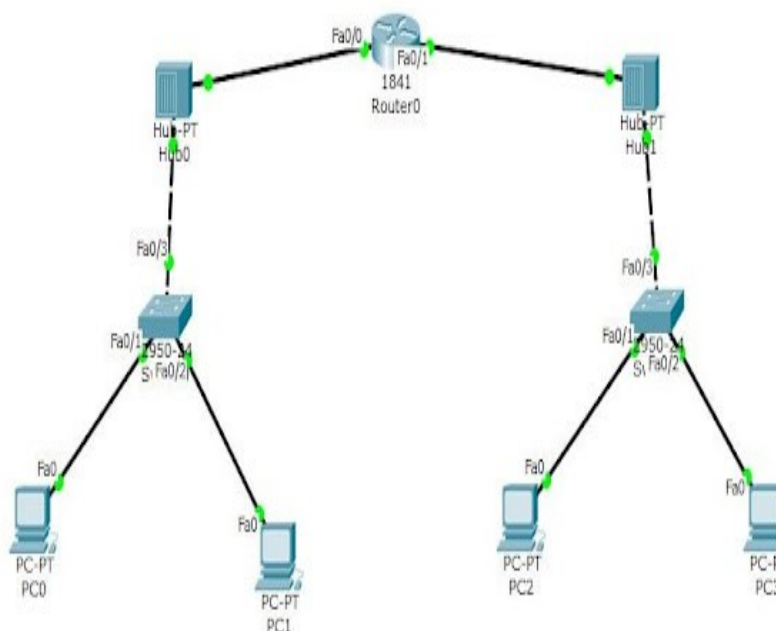
An unmanaged switch simply creates more Ethernet ports on a LAN, so that more local devices can access the Internet. Unmanaged switches pass data back and forth based on device MAC addresses.

A managed switch fulfills the same function for much larger networks, and offers network administrators much more control over how traffic is prioritized. They also enable administrators to set up Virtual LANs (VLANs) to further subdivide a local network into smaller chunks.

### **Router:**

A Router is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network. Or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of

interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.



0

**Step 1:** Take a router of model 1841.

**Step 2:** Take two hub of model Hub-PT.

**Step 3:** Take two switch of model 2950-24.

**Step 4:** Take pc0, pc1 for switch0, take pc2, pc3 for switch1.

**Step 5:** Connect pc0, pc1 to Switch0 with Straight through cable, connect pc2, pc3 to Switch1 with straight through cable, and connect Switch0 to Hub0 with cross over cable, connect Switch1 to Hub1 with cross over cable. Connect Hub0, Hub1 to Router0 with straight through cable.

**Step 6:** Router configuration:

Go to **Router0** → **CLI**

Router>enable

Router#configure terminal

Router(config)#interface fastEthernet 0/0

Router(config-if)#ip address 10.10.1.1 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface fastEthernet 0/0  
Router(config-if)#ip address 192.168.1.1 255.255.255.0  
Router(config-if)#no shutdown

**Step 7: All pc Configuration:**

Go to **PC0 → Desktop → IP Configuration → Static**

IP Address	10.10.1.5
Subnet Mask	255.0.0.0
Default Gateway	10.10.1.1

Go to **PC1 → Desktop → IP Configuration → Static**

IP Address	10.10.1.6
Subnet Mask	255.0.0.0
Default Gateway	10.10.1.1

Go to **PC2 → Desktop → IP Configuration → Static**

IP Address	192.168.1.5
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.1

Go to **PC3 → Desktop → IP Configuration → Static**

IP Address	192.168.1.6
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.1

## **Experiment-1(c)**

**Aim:- install and configure wired and wireless NIC and transfer files between systems in wired LAN and wireless LAN. Consider both adhoc and infrastructure mode of operation.hn**

### **Install and Configure of wired LAN:-**

- After connecting A computer to an ethernet cable, click the **Start** button, and select **Settings**.
- Select **Network and Internet**.
- In **Status**, click **Network and Sharing Center**.
- Choose **Change adapter settings** on the left side menu.
- Right-click Ethernet and then choose Properties.
- Select **Internet Protocol Version 4 (TCP/IPv4)**, then click **Properties**.
- Set up the IP to use, then click **OK** to save your settings. You may want to restart your computer.
- Open a web browser to test your connection. In some instances, you may be asked to register your computer onto CU Boulder's network. If you register your computer, you should clear your web browser's cache after registration.

### **Install and Configure of wireless LAN:-**

- The optimal placement is in a central location, free from obstructions that could cause wireless interference..
- Plug an Ethernet cable (typically provided with the router) into the router **WAN port**. Then, connect the other end of the Ethernet cable to the modem.
- plug one end of another Ethernet cable into the router LAN port (any port will work) and the other end of the Ethernet cable into the Ethernet port of a laptop.
- It's best if you turn on these devices in the proper order. Turn on the modem first. When the modem lights are all on, turn on the router. When the router is on, turn on the computer.
- Configure the router name, static or dynamic ip addresses and security password to the router and save configuration.
- After saving the router's configuration settings, unplug the cable that connects the computer to the router. Then, plug a USB or PC card wireless adapter into the laptop if it doesn't have a wireless adapter installed or built-in.
- Your computer may automatically install the drivers, or you may have to use the setup CD that came with the adapter.
- On your computer and other wireless-enabled devices, find the new network you set up and **connect to the network**.

### **Ad-Hoc Mode & Infrastructure Mode**

- Infrastructure mode is the most common mode to operate and is the factory default setting.
- Infrastructure mode is used when a PC is connected to a wireless access point or to a wireless router.
- If the hosts are connected to the network via a base station or an access point (fixed infrastructure), then the network is said to be operating in infrastructure mode.
- Cellular networks work in infrastructure mode because all the cell phones connect to the network through the base stations (cell towers).

When a network is not in infrastructure mode, then it is said to be operating in Ad hoc mode of operation.

- In an Ad hoc mode of operation, the wireless hosts can communicate with each other without an access point.
- No fixed infrastructure such as access point is required for the wireless hosts.

## **Experiment-2**

**Aim:-Work with the commands Ping, Tracert, Ipconfig, pathping, telnet, ftp, getmac, ARP, Hostname, nbtstaff, netdiag, and nslookup.**

### **Basic networking commands:-**

C:\>ping: Ping is the most basic TCP/IP command, and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with "Hello" on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers



name or  
IP address.

C:\>tracert: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

C:\>ipconfig: The ipconfig command displays information about the host (the computer you're sitting at) computer TCP/IP configuration.

C:\>ipconfig /all: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

C:\>Ipconfig /renew: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

C:\>Ipconfig /release: This command allows you to drop the IP lease from the DHCP server.

C:\>ipconfig /flushdns: This command is only needed if you're having trouble with your network's DNS configuration. The best time to use this command is after network configuration frustration sets in, and you really need the computer to reply with flushed.

C:\>pathping: Pathping is unique to Windows, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25 second test of each router along the way, gathering statistics on the rate of data loss along each hop.

C:\telnet

Telnet is a text-based program you can use to connect to another computer using the Internet. You'll be able to access programs and services that are on the remote computer as if you were sitting right in front of it.

C:\ftp

It is a protocol

C:\>getmac: This is the simplest of all TCP/IP commands. It simply displays the MAC or physical address of your computer.

C:\>arp -a: ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.

C:\>hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

C:\>nbtstat -a: This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)

C:\>netdiag: Netdiag is a network testing utility that performs a variety of network diagnostic tests, allowing you to pinpoint problems in your network. Netdiag isn't installed by default, but can be installed from the Windows XP CD after saying no to the install. Navigate to the CD ROM drive letter and open the support\tools folder on the XP CD and click the setup.exe icon in the support\tools folder.

C:\>netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

C:\>nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

5

C:\>route: The route command displays the computers routing table. A typical computer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn't pose any network problems. But if you're having trouble accessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

### **Experimnet-3**

**Aim:- use sniffers for monitoring and network communication(Eatheral).**

#### **What is Eatheral:-**

Eatheral is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

You could think of a network packet analyzer as a measuring device used to examine what's going on inside a network cable, just like a voltmeter is used by an electrician to examine what's going on inside an electric cable(but as higher level, of course).

In the past such tools either very expensive, proprietary, or both, however, with the advent of Eatheral, all that has changed.

Ethereal is perhaps one of the best open source packet analyzers available today.

Some intended purposes:-

Here are some examples people use Eatheral for;

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

Beside these examples, Eatheral can be helpful in many other situations too.

Features:-

The following are some of the many features Eatheral provides:

- Available for UNIX and WINDOWS.
- Capture live packet data from a network interface.
- Display packets with very detailed protocol information.
- Open and save packet data captured.
- Import and Export packet data from and to a lot of other capture programs.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Create various statistics.

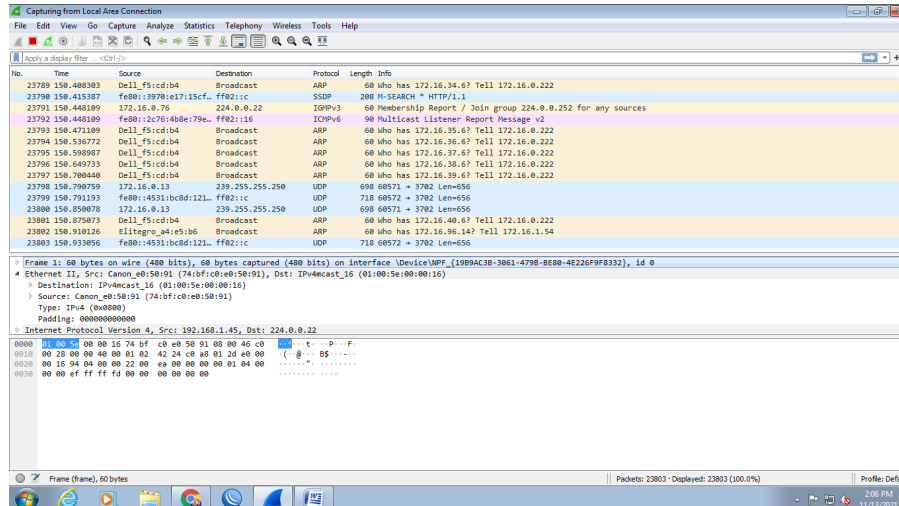
- .... And a lot more!

Ethereal can be used to capture the network packets and displays their details.

The actual process is going through the following steps:

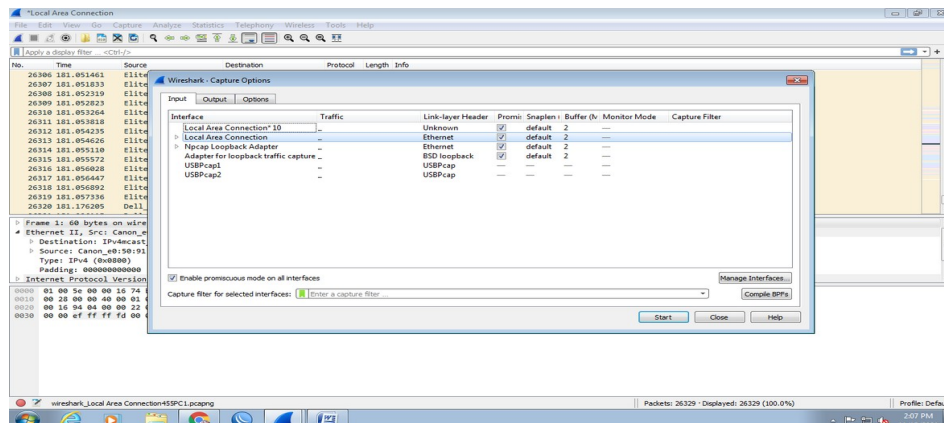
Step 1:-

Open the Ethereal folder/click the ethereal icon, then you can see the window shown below.



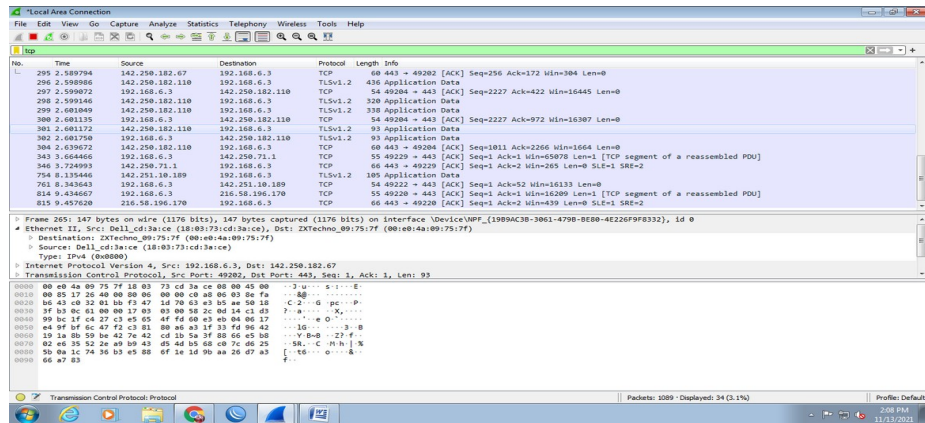
Step 2:-

You can press the Capture menu then press Interfaces, then you can see the window shown below.



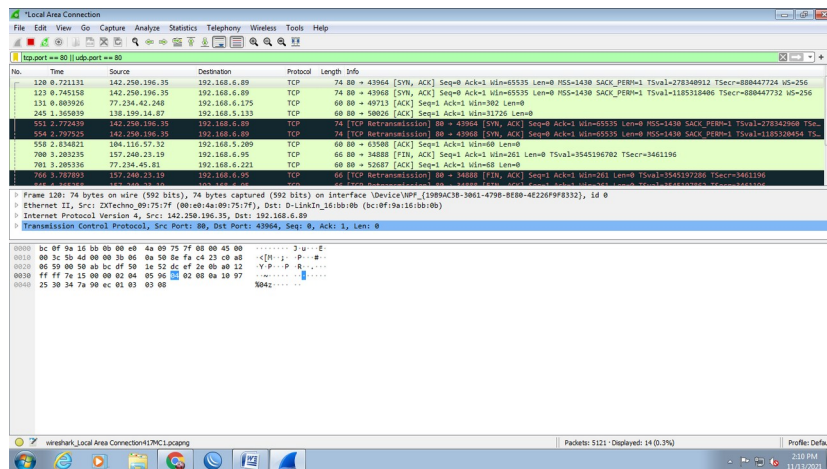
Step 3:-

- Then the packets are preparing as shown in the figure. In our example 29 packets are prepared.
- Now press Capture button, then you can see the window shown below.



#### Step 4:-

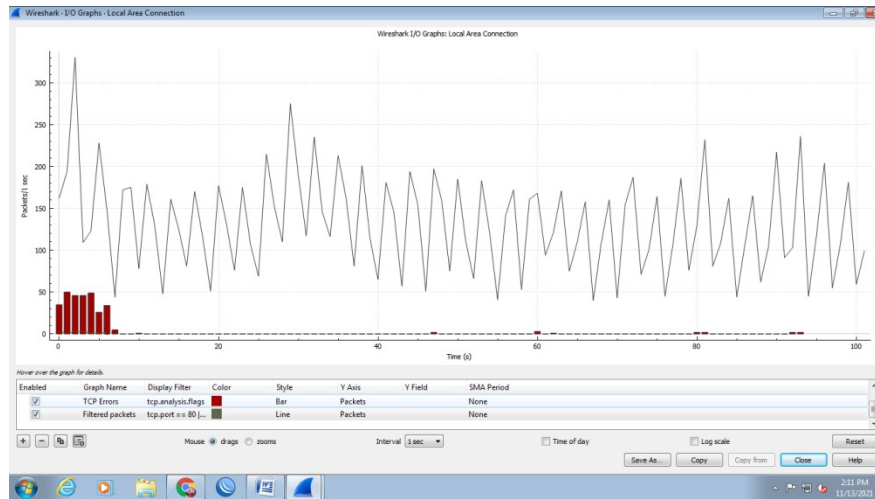
After some time click stop, then you can see three windows shown below.



- The first window shows all the general frames and their protocol, etc
- The second window shows the information about the particular frame which is selected in the first window.
- The third window shows the corresponding output.

#### Ethereal IO graphs:-

- ✓ In the graph time is taken along the x-axis and numbers of packets are taken along the y-axis.
- ✓ To obtain IO graphs click on Status menu and then click on IOgraphs, then you can see the window shown below.
- ✓ In the above window( Ethereal IO Graph) we can able to check the graphs corresponding to the protocols which are present in the previous three windows.
- ✓ You observe the protocols in the previous window and then type those in the filler columns.
- ✓ Now you click on the buttons graphs 1, graphs 2 etc., then you can observe the corresponding graphs in the above display space as shown below.



Result:-

Using Ethernet packets are captured, protocols are observed and the corresponding IO graphs are observed successfully.

## **Experiment-4**

**Aim:-** Find all ip addresses on your network. Unicast, multicast and Broadcast on your network.

### **IP Scanning:-**

Operating Systems, like Windows and Linux, come with their own native simple networking set of tools. Commands such as “ipconfig”, “arp -a”, or “ping” allow simple scanning and troubleshooting. The simplest way to get a quick list of IP addresses and their devices connected to your network is with those OS native commands found in the command line. With a list of the assigned IP address and their devices, you can easily find the devices that are causing the most problems.

#### ***ipconfig***

This command displays all network settings assigned to one or all adapters in the computer. You can find information such as your own IP, subnet, and Gateway. For Linux and MacOS is “ifconfig”.

#### ***arp -a***

When you issue the “arp -a”, you’ll get IP-address-to-mac conversion and the allocation type (whether dynamic or static) of all devices in your network.

#### ***Ping***

It helps determine connectivity between two hosts and find the IP address of a hostname.

### **Finding your own network adapter configuration:-**

Go to Run > type **cmd** > type **ipconfig**

This Windows computer has 5 network adapters, but the last one (Wireless LAN adapter Wi-Fi) is the only one connected to a network. The rest are disconnected.

```
>ipconfig
Windows IP Configuration

Ethernet adapter Ethernet 2:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
Ethernet adapter Ethernet:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
Wireless LAN adapter Local Area Connection* 6:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
Wireless LAN adapter Local Area Connection* 7:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
Wireless LAN adapter Wi-Fi:
    Connection-specific DNS Suffix  . : Home
    Link-local IPv6 Address . . . . . : fe80::ad4b:d267:7d36:9f26%24
    IPv4 Address. . . . . : 192.168.1.41
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1
```

- In this network, the router (or Default Gateway) is playing the role of the DHCP server. It is assigning the IP address dynamically and giving access to the Internet.
- You are reading two of the most important IP addresses for your device; Your own device's IP (IPv4 and IPv6) and your Gateway. The Subnet Mask is also very important, it shows that you are on the same subnet as the gateway.

Now you know your subnet, which in this case is 192.168.1.0/24 . Now you need to find the rest of the IP address in your network.

**Scanning the Network:-**

The job of the ARP protocol is to map IPs to MAC addresses. It provides a method for hosts on a LAN to communicate without knowing any address and create a cache of information. When a new computer enters the LAN, it receives an IP and updates its ARP cache with the Gateway information. This ARP cache can be found using the “arp-a” command.

- Use the command line to enter the “arp -a” command.

```
>arp -a
Interface: 192.168.1.41 --- 0x18
Internet Address      Physical Address      Type
192.168.1.1           b0-46-fc-7c-b0-50    dynamic
192.168.1.25          00-1b-a9-58-04-4e    dynamic
192.168.1.33          00-18-f3-23-50-d6    dynamic
192.168.1.36          0c-8f-ff-4c-92-c6    dynamic
192.168.1.37          60-30-d4-76-b8-c8    dynamic
192.168.1.40          b4-99-ba-de-66-90    dynamic
192.168.1.200         4c-9e-ff-c0-67-bc    dynamic
224.0.0.2             01-00-5e-00-00-02    static
224.0.0.22            01-00-5e-00-00-16    static
224.0.0.251           01-00-5e-00-00-fb    static
224.0.0.252           01-00-5e-00-00-fc    static
239.0.2.2             01-00-5e-00-02-02    static
239.0.2.30            01-00-5e-00-02-1e    static
239.0.2.129           01-00-5e-00-02-81    static
239.0.2.153           01-00-5e-00-02-99    static
239.0.5.185           01-00-5e-00-05-b9    static
239.255.255.250       01-00-5e-7f-ff-fa    static
```

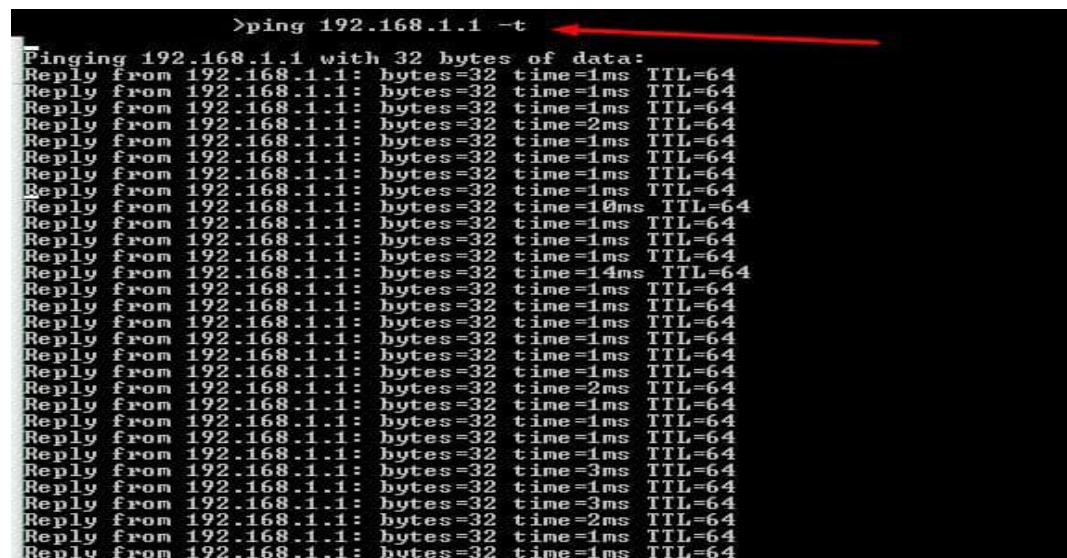
- This computer has been connected for some time into the LAN, so its ARP cache is very precise and complete. The first IP address shown in the display is the Gateway (the same we found through the ipconfig command).



- The output shows the IP, the MAC addresses, and their assignation type. The addresses displayed here were dynamically assigned by the DHCP server in the LAN. All of these IPs are devices connected to the LAN (192.168.1.0/24). The other static addresses are reserved for Multicasting.

## Testing Connectivity

Finally, with some information, you can test connectivity. In the following test, we tried an extended ping with “ping -t” to the gateway. With this, you can learn some simple insights about delay and latency.



```
>ping 192.168.1.1 -t
Pinging 192.168.1.1 with 32 bytes of data:
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=2ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=10ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=14ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=2ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=3ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=3ms TTL=64
Reply from 192.168.1.1: bytes=32 time=2ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
Reply from 192.168.1.1: bytes=32 time=1ms TTL=64
```

From the list generated by the ARP command, you could ping all the live hosts. Or you can go beyond and ping the entire subnet to find hosts not found by the ARP (but that would be too much manual work...). Later, we’ll discuss how to automatically ping entire subnets at once.

**Unicast:** traffic, many streams of IP packets that move across networks flow from a single point, such as a website server, to a single endpoint such as a client PC. This is the most common form of information transference on networks.

**Broadcast:** Here, traffic streams from a single point to all possible endpoints within reach on the network, which is generally a LAN. This is the easiest technique to ensure traffic reaches its destinations. This mode is mainly utilized by television networks for video and audio distribution. Even if the television network is a cable television (CATV) system, the source signal reaches all possible destinations, which is the key reason that some channels’ content is scrambled. Broadcasting is not practicable on the public Internet due to the massive amount of unnecessary data that would continually reach each user’s device, the complications and impact of scrambling, and related privacy issues.

**Multicast:** In this method traffic recline between the boundaries of unicast (one point to one destination) and broadcast (one point to all destinations). And multicast is a “one source to many destinations” way of traffic distribution, which means that only the destinations that openly point to their requisite to accept the data from a specific source to receive the traffic stream.

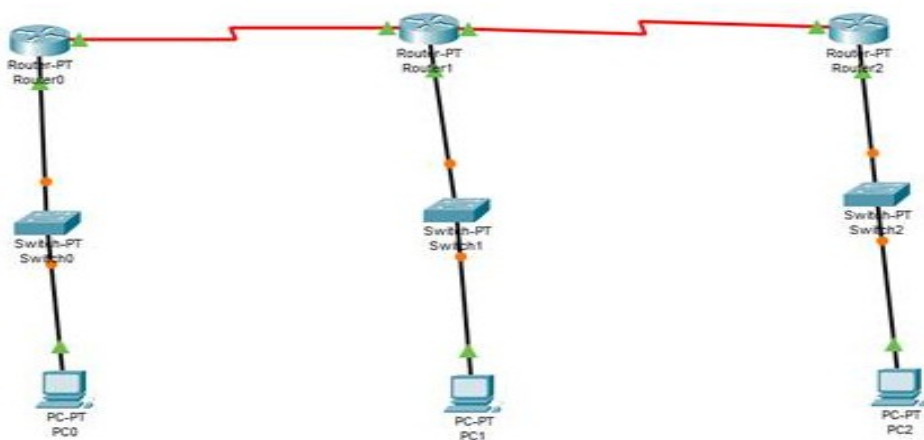
On an IP network, destinations (i.e. clients) do not regularly communicate straight to sources (i.e. servers), because the routers between source and destination must be able to regulate the topology of the network from unicast or multicast side to avoid disordered routing traffic. Multicast routers replicate packets received on one input interface and send the replicas out on multiple output interfaces.



In the multicast model, the source and destinations are almost every time “Host” and not “Routers”. The multicast traffic is spread by multicast routers across the network from source to destination. The multicast routers must find multicast sources on the network, send out copies of packets on a number of interfaces, avoid loops, connect interested destinations with accurate sources and keep the flow of unsolicited packets to a minimum. The standard protocols of multicast routing provide most of these facilities, but some router architecture cannot send multiple copies of packets and so do not support direct multicasting.

### **Experiment-5**

**Aim:-**Use Packet tracer software to build network topology and configure using Distance vector routing protocol



**Step 1:** Take three model of router Router-PT.

**Step 2:** Take three model of Switch-PT.

**Step 3:** Take three pc's PC-0, PC-1 and PC-2.

**Step 4:** Take pc0, pc1 for switch0, take pc2, pc3 for switch1.

**Step 5:** Connect pc0 to Switch0, pc1 to Switch1 and pc2 to Switch2 with Straight through cable, connect Switch0 to Router0, Switch1 to Router1 and Switch2 to Router2 with straight through cable, and connect Router0 to Router1 and Router1 to Router2 with cross over cable.

**Step 6:** Routers configuration:

Go to **Router0** → **CONFIG**

- FastEthernet0/0 ip address 192.168.10.1 subnet mask 255.255.255.0 and port status on.
- Serial port2/0 ip address 10.0.0.2 subnet mask 255.0.0.0 clock rate 64000 and port status on.
- RIP Network addresses 192.168.10.1 and 10.0.0.2
- Pc0 ip address 192.168.10.2 255.255.255.0 default gateway 192.168.10.1

Go to **Router1** → **CONFIG**

- FastEthernet0/0 ip address 192.168.20.1 subnet mask 255.255.255.0 and port status on.
- Serial port2/0 ip address 10.0.0.3 subnet mask 255.0.0.0 clock rate 64000 and port status on.
- Serial port3/0 ip address 20.0.0.2 subnet mask 255.0.0.0 clock rate 64000 and port status on.
- RIP Network addresses 192.168.20.1, 10.0.0.3 and 20.0.0.2
- Pc1 ip address 192.168.20.2 255.255.255.0 default gateway 192.168.20.1

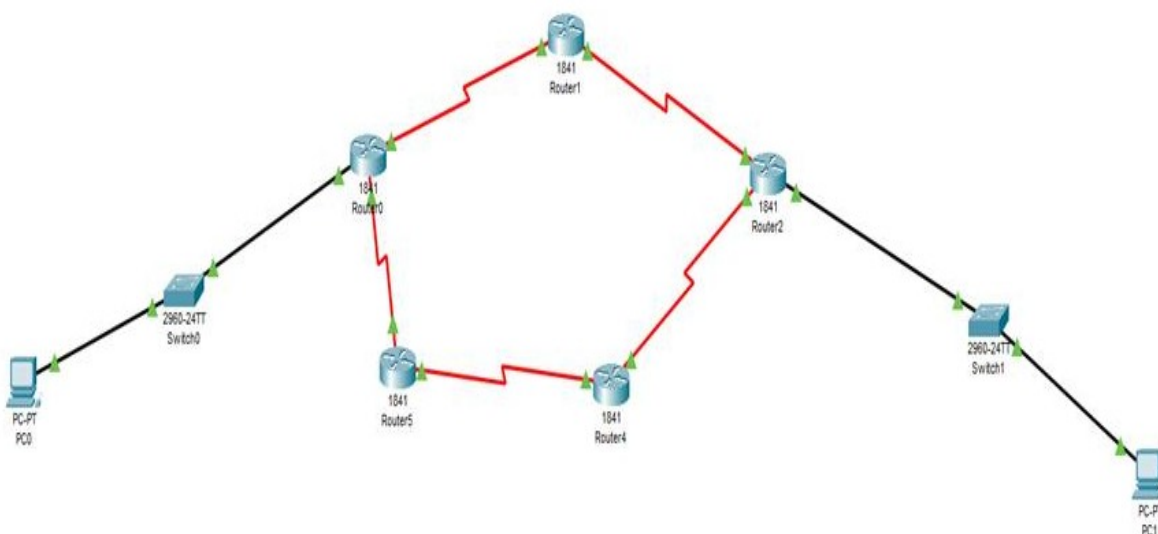
Go to **Router1** → **CONFIG**

- FastEthernet0/0 ip address 192.168.30.1 subnet mask 255.255.255.0 and port status on.
- Serial port2/0 ip address 20.0.0.3 subnet mask 255.0.0.0 clock rate 64000 and port status on.
- RIP Network addresses 192.168.30.1 and 20.0.0.3
- Pc2 ip address 192.168.30.2 255.255.255.0 and default gateway 192.168.30.1

**Step 7:** Start the simulation and analyze the distance vector routing protocol.

## Experiment-6

**Aim:-**Use Packet tracer software to build network topology and configure using Link State routing protocol.



**Step 1:** Take Five routers model 1841.

**Step 2:** Take two switch of model 2960-24.

**Step 3:** Take two pc's pc0 and pc1.

**Step 4:** Connect pc0 to Switch0 with Straight through cable, connect pc2 to Switch1 with straight through cable, and connect Switch0 to Router with Straight cable, connect Switch1 to Router2 with Straight cable. Connect Router0, Router1, Router2, Router3 and Router4 with each other through Cross cable.

**Step 5:** Routers configuration:

Go to **Router0** → **CLI**

```
Router>enable
Router#configure terminal
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 192.168.10.2 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0
Router(config-if)#ip address 10.0.0.3 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/1
Router(config-if)#ip address 17.0.0.3 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

Go to **Router1** → **CLI**

```
Router>enable
Router#configure terminal
Router(config)#interface serial 0/0
Router(config-if)#ip address 10.0.0.2 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/1
Router(config-if)#ip address 11.0.0.1 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

Go to **Router2** → **CLI**

```
Router>enable
Router#configure terminal
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 192.168.20.2 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0
Router(config-if)#ip address 17.0.0.2 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/1
Router(config-if)#ip address 15.0.0.2 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

Go to **Router3** → **CLI**

Router>enable

Router#configure terminal

Router(config)#interface serial 0/0

Router(config-if)#ip address 14.0.0.2 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/1

Router(config-if)#ip address 15.0.0.2 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

Go to **Router4** → **CLI**

Router>enable

Router#configure terminal

Router(config)#interface serial 0/0

Router(config-if)#ip address 10.0.0.2 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/1

Router(config-if)#ip address 17.0.0.2 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

**Step 7:** Check link state routing protocol.

## **Experiment-7**

**Aim:-** . Use JAVA RMI Write a program to implement Basic Calculator.

**Programmes:-**

### **Calculator.java**

```
public interface Calculator extends java.rmi.Remote {  
    public long add(long a, long b) throws java.rmi.RemoteException;  
    public long sub(long a, long b) throws java.rmi.RemoteException;  
    public long mul(long a, long b) throws java.rmi.RemoteException;  
    public long div(long a, long b) throws java.rmi.RemoteException;  
}
```

### **CalculatorClient.java**

```
import java.rmi.Naming;  
import java.rmi.RemoteException;  
import java.net.MalformedURLException;  
import java.rmi.NotBoundException;
```

```
public class CalculatorClient {
```

```

public static void main(String[] args) {
    try {
        Calculator c = (Calculator)Naming.lookup("rmi://localhost/CalculatorService");
        System.out.println( c.sub(4, 3) );
        System.out.println( c.add(4, 5) );
        System.out.println( c.mul(3, 6) );
        System.out.println( c.div(9, 3) );
    }
    catch (MalformedURLException murle) {
        System.out.println( );
        System.out.println("MalformedURLException");
        System.out.println(murle);
    }
    catch (RemoteException re) {
        System.out.println();
        System.out.println("RemoteException");
        System.out.println(re);
    }
    catch (NotBoundException nbe) {
        System.out.println();
        System.out.println( "NotBoundException");
        System.out.println(nbe);
    }
    catch (
        java.lang.ArithmeticException ae) {
        System.out.println();
        System.out.println("java.lang.ArithmeticException");
        System.out.println(ae);
    }
}
}

```

### **CalculatorImpl.java**

```

public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject
implements Calculator {
    // Implementations must have an
    //explicit constructor
    // in order to declare the
    //RemoteException exception
    public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }

    public long add(long a, long b) throws java.rmi.RemoteException {
        return a + b;
    }

    public long sub(long a, long b) throws java.rmi.RemoteException {
        return a - b;
    }

    public long mul(long a, long b) throws java.rmi.RemoteException {
        return a * b;
    }

    public long div(long a, long b) throws java.rmi.RemoteException {
        return a / b;
    }
}

```

### **CalculatorServer.java**

```

import java.rmi.Naming;

```

```

public class CalculatorServer {
    public CalculatorServer() {
        try {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }
    public static void main(String args[]) {
        new CalculatorServer();
    }
}

```

### ***Compiling the code***

Use **javac** to compile all 4 programs.

```

> javac Calculator.java
> javac CalculatorImpl.java
> javac CalculatorServer.java
> javac CalculatorClient.java

```

Now use **rmic** to create the stub and skeleton class files.

```

> rmic CalculatorImpl

```

### ***Running the RMI System***

You are now ready to run the system! You need to start three consoles, one for the server, one for the client, and one for the RMIRegistry.

Start with the Registry. You must be in the directory that contains the classes you have written. From there, enter the following:

```

> rmiregistry

```

If all goes well, the registry will start running and you can switch to the next console.

In the second console start the server hosting the `CalculatorService`, and enter the following:

```

> java CalculatorServer

```

It will start, load the implementation into memory and wait for a client connection.

In the last console, start the client program.

```

> java CalculatorClient

```

If all goes well you will see the following output:

```

1
9
18
3

```

## **Experiment-8**

**Aim:** Create chat application using either TCP protocol.

### **Programmes:-**

#### **Server Code:-**

```
import java.net.*;
import java.util.Scanner;
public class Server {
    public static void main(String[] args) throws Exception
    {
        ServerSocketss=new ServerSocket(7888);
```

```

        Socket s=ss.accept();
        DataInputStream din=new DataInputStream(s.getInputStream());
        String str;
        str=din.readUTF();
        System.out.println("Client:\t"+str);
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
        DataInputStream msg=new DataInputStream(System.in);
        while(true)
        {
            str=din.readUTF();
            System.out.print("Client:\t"+str);
            System.out.print("Server:\t");
            str=msg.readLine();
            dout.writeUTF(str);
        }
    }
}

```

### **Client Code:-**

```

import java.io.*;
import java.net.Socket;
import java.util.Scanner;
public class Client
{
    public static void main(String[] args) throws Exception
    {
        Socket s=new Socket("127.0.0.1",7888);
        if(s.isConnected())
        {
            System.out.println("Connected to server");
        }
        DataInputStream msg=new DataInputStream(System.in);
        String str="Start Chat.....";
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
        dout.writeUTF(str);
        System.out.println(str);
        DataInputStream din=new DataInputStream(s.getInputStream());
        while(true)

```



```

    {
        System.out.print("Client:\t");
        str=msg.readLine();
        dout.writeUTF(str+"\n");
        str=din.readUTF();
        System.out.println("Server:\t"+str);
    }
}
}

```

### **Sample Output:-**

#### **Server**

```

C:\Windows\system32\cmd.exe - java Server
H:\Subjects\Advance JAVA\Practicals\1>java Server
Client: Start Chat.....
.....
Client: hi
Server: hello
Client: what's going on?
Server: Server-client chat application
Client: great
Server: _

```

#### **Client**

```

C:\Windows\system32\cmd.exe - java Client
H:\Subjects\Advance JAVA\Practicals\1>java Client
Connected to server
Start Chat.....
.....
Client: hi
Server: hello
Client: what's going on?
Server: Server-client chat application
Client: great

```

**Aim: Create chat application using either UDP protocol.**

### **Programmes:-**

#### **Server code:-**

```

import java.io.*;
import java.net.*;

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

```

```

{
    DatagramSocket socket=new DatagramSocket(9861);
    byte receiveByte[]=new byte[1024];
    byte sendByte[]=new byte[1024];
    while(true)
    {
        //data receiving
        DatagramPacketreceivePacket=new DatagramPacket(receiveByte,receiveByte.length);
        socket.receive(receivePacket);
        String receiveStr=new String(receivePacket.getData());
        receiveStr=receiveStr.trim();
        System.out.println("Client:"+receiveStr);
        //data sending
        DataInputStream din=new DataInputStream(System.in);
        System.out.print("Server:");
        String sendStr=din.readLine();
        sendByte=sendStr.getBytes();
        InetAddressip=receivePacket.getAddress();
        int port=receivePacket.getPort();
        DatagramPacketsend Packet=new DatagramPacket(sendByte,sendByte.length,ip,port);
        socket.send(sendPacket);
    }
}
}

```

### **Client Code:-**

```

import java.net.*;
import java.io.*;
public class Client {
    public static void main(String[] args) throws Exception
    {
        DatagramSocket socket=new DatagramSocket();
        InetAddressip=InetAddress.getLocalHost();
        byte sendByte[]=new byte[1024];
        byte receiveByte[]=new byte[1024];
        while(true)
        {
            //Data Sending

```

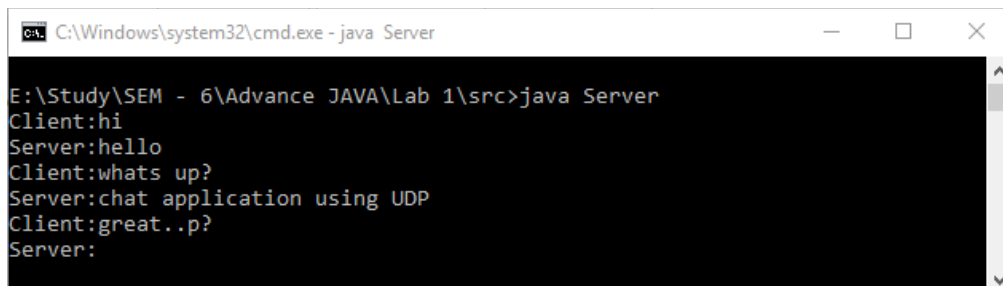
```

        DataInputStream din=new DataInputStream(System.in);
        System.out.print("Client:");
        String sendStr=din.readLine();
        sendByte=sendStr.getBytes();
        DatagramPacket sendPacket=new DatagramPacket(sendByte,sendByte.length,ip,9861);
        socket.send(sendPacket);
        //Data Receiving
        DatagramPacket receivePacket=new DatagramPacket(receiveByte,receiveByte.length);
        socket.receive(receivePacket);
        String receiveStr=new String(receivePacket.getData());
        receiveStr=receiveStr.trim();
        System.out.println("Server:"+receiveStr);
    }
}
}

```

### **Sample Output:-**

#### **Server**

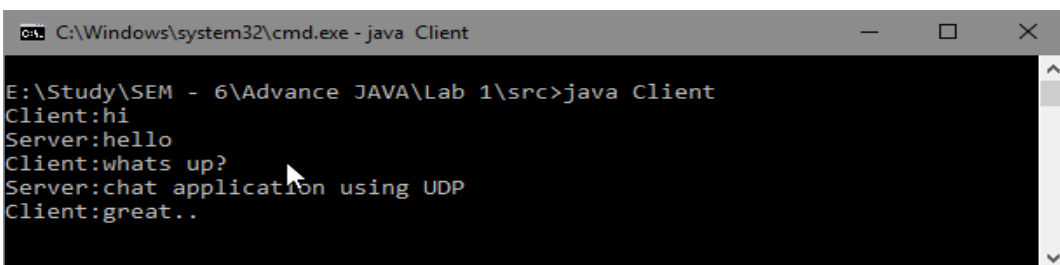


```

C:\Windows\system32\cmd.exe - java Server
E:\Study\SEM - 6\Advance JAVA\Lab 1\src>java Server
Client:hi
Server:hello
Client:whats up?
Server:chat application using UDP
Client:great..p?
Server:

```

#### **Client**



```

C:\Windows\system32\cmd.exe - java Client
E:\Study\SEM - 6\Advance JAVA\Lab 1\src>java Client
Client:hi
Server:hello
Client:whats up?
Server:chat application using UDP
Client:great..

```

### **Experiment No:-9**

**Aim:-** Hello command is used to know whether the machine at the other end is working or not. Echo command is used to measure the round trip time to the neighbour. Implement Hello and Echo commands using JAVA.

#### **Programme:-**

```

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader

public class ProcessBuilderPing{

public static void main(String [] args){

ProcessBuilder processBuilder =new ProcessBuilder();

//windows

processBuilder command("cmd.exe","/c", "ping -n 3 google.com");

try

{

Process process= processBuilder.start();

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

String line;

While((line=reader.readLine())!=null){

System.out.println("\n Exited with error code"+exitCode);

}

Catch (IOException e){

e.printStackTrace();

}

Catch(InterruptedExceotion e){

e.printStackTrace();

}

}

}

}

```

Output:-

Pinging google.com[172.217.31.78]with 32 bytes of data:

Reply from 172.217.31.78:bytes=32 time=13ms TTL=55

Reply from 172.217.31.78:bytes=32 time=13ms TTL=55

Reply from 172.217.31.78:bytes=32 time=13ms TTL=55

Reply from 172.217.31.78:bytes=32 time=13ms TTL=55

Ping statistics for 172.217.31.78:

Packets: Sent=4, Received=4, Loss=0(0%loss),

Approximate round trip times in milli-seconds;

Minimum=6ms, Maximum=13ms, Average=8ms

### **SOCKET PROGRAM FOR ECHO.**

AIM To write a socket program for implementation of echo.

#### **ALGORITHM**

##### **CLIENT SIDE**

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

##### **SERVER SIDE**

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

#### **PROGRAM**

##### **ECHO CLIENT**

```

import java.io.*;

import java.net.*;

public class eclient
{
    public static void main(String args[])
    {
        Socket c=null;

        String line;

        DataInputStream is,is1;

        PrintStream os;

        try
        {
            c=new Socket("localhost",8080);
        }
        catch(IOException e)
        {
            System.out.println(e);
        }

        try
        {
            7

            os=new PrintStream(c.getOutputStream());

            is=new DataInputStream(System.in);

            is1=new DataInputStream(c.getInputStream());

            do
            {
                System.out.println("client");

                line=is.readLine();

                os.println(line);

```

```

if(!line.equals("exit"))

System.out.println("server:"+is1.readLine());

}while(!line.equals("exit"));

}

catch(IOException e)

{

System.out.println("socket closed");

}}

```

Echo Server:

```

import java.io.*;

import java.net.*;

import java.lang.*;

public class eserver

{

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

{

ServerSocket s=null;

String line;

DataInputStream is;

PrintStream ps;

Socket c=null;

try

{

s=new ServerSocket(8080);

}

catch(IOException e)

{

System.out.println(e);

}

```

```

try
{
c=s.accept();

is=new DataInputStream(c.getInputStream());

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

while(true)
{
line=is.readLine();

System.out.println("msg received and sent back to client");

ps.println(line);

}

8

}

catch(IOException e)
{
System.out.println(e);
}
}
}

```

OUTPUT

CLIEN

Enter the IP address 127.0.0.1

CONNECTION ESTABLISHED

Enter the data SRM

Client received SRM

SERVER

CONNECTION ACCEPTED

Server received SRM



## Experiment No:10

**Aim:- Use Ethereal tool to Capture information about packets.**

When you launch Wireshark, a welcome screen lists the available network connections on your current device. Displayed to the right of each is an EKG-style line graph that represents live traffic on that network.

To begin capturing packets with Wireshark:

1. Select one or more of networks, go to the menu bar, then select **Capture**.

To select multiple networks, hold the **Shift** key as you make your selection.

2. In the **Wireshark Capture Interfaces** window, select **Start**.

There are other ways to initiate packet capturing. Select the **shark fin** on the left side of the Wireshark toolbar, press **Ctrl+E**, or double-click the network.

3. Select **File > Save As** or choose an **Export** option to record the capture.
4. To stop capturing, press **Ctrl+E**. Or, go to the Wireshark toolbar and select the red **Stop** button that's located next to the shark fin.

### **View and Analyze Packet Contents**

The captured data interface contains three main sections:

- The packet list pane (the top section)
- The packet details pane (the middle section)
- The packet bytes pane (the bottom section)

### **Packet List**

The packet list pane, located at the top of the window, shows all packets found in the active capture file. Each packet has its own row and corresponding number assigned to it, along with each of these data points:

- **No:** This field indicates which packets are part of the same conversation. It remains blank until you select a packet.
- **Time:** The timestamp of when the packet was captured is displayed in this column. The default format is the number of seconds or partial seconds since this specific capture file was first created.
- **Source:** This column contains the address (IP or other) where the packet originated.
- **Destination:** This column contains the address that the packet is being sent to.
- **Protocol:** The packet's protocol name, such as TCP, can be found in this column.
- **Length:** The packet length, in bytes, is displayed in this column.

- **Info:** Additional details about the packet are presented here. The contents of this column can vary greatly depending on packet contents.

To change the time format to something more useful such as the actual time of day, select **View > Time Display Format**.

When a packet is selected in the top pane, you may notice one or more symbols appear in the **No.** column. Open or closed brackets and a straight horizontal line indicate whether a packet or group of packets are part of the same back-and-forth conversation on the network. A broken horizontal line signifies that a packet is not part of the conversation.

## Packet Details

The details pane, found in the middle, presents the protocols and protocol fields of the selected packet in a collapsible format. In addition to expanding each selection, you can apply individual Wireshark filters based on specific details and follow streams of data based on protocol type by right-clicking the desired item.

## Packet Bytes

At the bottom is the packet bytes pane, which displays the raw data of the selected packet in a hexadecimal view. This [hex dump](#) contains 16 hexadecimal bytes and 16 ASCII bytes alongside the data offset.

Selecting a specific portion of this data automatically highlights its corresponding section in the packet details pane and vice versa. Any bytes that cannot be printed are represented by a period.

To display this data in bit format as opposed to hexadecimal, right-click anywhere within the pane and select **as bits**.

## Wireshark Filters:-

Capture filters instruct Wireshark to only record packets that meet specified criteria. Filters can also be applied to a capture file that has been created so that only certain packets are shown. These are referred to as display filters.

Wireshark provides a large number of predefined filters by default. To use one of these existing filters, enter its name in the **Apply a display filter** entry field located below the Wireshark toolbar or in the **Enter a capture filter** field located in the center of the welcome screen.

For example, if you want to display TCP packets, type **tcp**. The Wireshark autocomplete feature shows suggested names as you begin typing, making it easier to find the correct moniker for the filter you're seeking.

Another way to choose a filter is to select the **bookmark** on the left side of the entry field. Choose **Manage Filter Expressions** or **Manage Display Filters** to add, remove, or edit filters.

You can also access previously used filters by selecting the down arrow on the right side of the entry field to display a history drop-down list.

Capture filters are applied as soon as you begin recording network traffic. To apply a display filter, select the right arrow on the right side of the entry field.

## Wireshark Color Rules:-

While Wireshark's capture and display filters limit which packets are recorded or shown on the screen, its colorization function takes things a step further: It can distinguish between different packet types based on their individual hue. This quickly locates certain packets within a saved set by their row color in the packet list pane.

Wireshark comes with about 20 default coloring rules, each can be edited, disabled, or deleted. Select **View > Coloring Rules** for an overview of what each color means. You can also add your own color-based filters.

Select **View > Colorize Packet List** to toggle packet colorization on and off.

## **Statistics in Wireshark**

Other useful metrics are available through the **Statistics** drop-down menu. These include size and timing information about the capture file, along with dozens of charts and graphs ranging in topic from packet conversation breakdowns to load distribution of HTTP requests.

Display filters can be applied to many of these statistics via their interfaces, and the results can be exported to common file formats, including [CSV](#), [XML](#), and TXT.

## **Experiment NO:-11**

**Aim:- Install Network Simulator 2/3. Create a wired network using dumbbell topology. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.**

### **Network Simulator 2.35 installation:-**

#### Step 1:

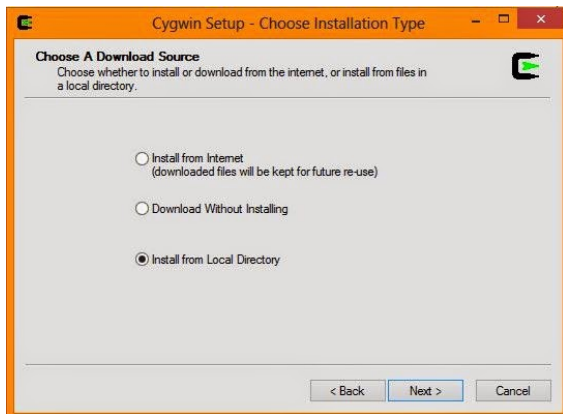
- Go to these link and download the three .rar files and place them in a single folder.
- Link for part 1(100Mb):[NS 2.35-Part1](#)
- Link for part 2(100Mb):[NS 2.35-Part2](#)
- Link for part 3(19.49Mb):[NS 2.35-Part3](#)

#### Step 2:

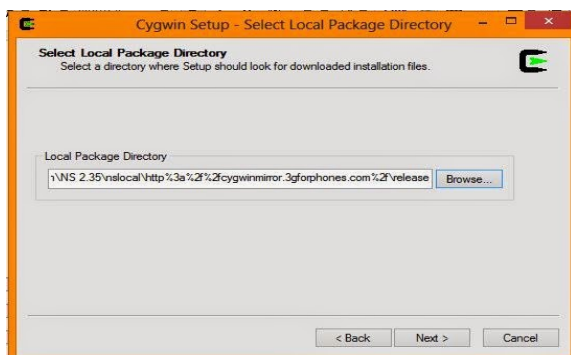
- Now Extract the [NS 2.35.part1](#) only.It will extract the part2 an part3 automatically.
- Note:
- Extract after complete download
- Don't Extract the part2 and part3

#### Step 3:

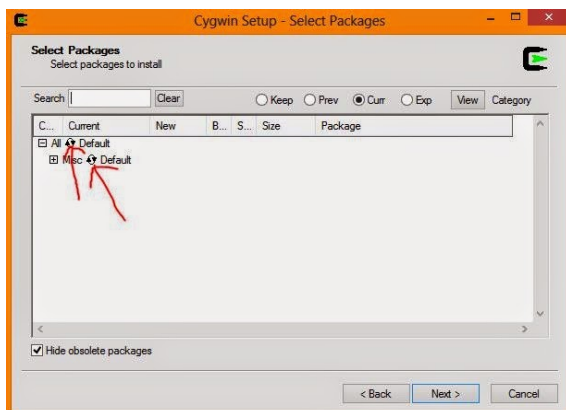
- After extraction you'll get a folder named NS 2.35
- Inside this folder you get cygwin setup , .bashrc etc..
- Now install the setup. Choose install from local directory



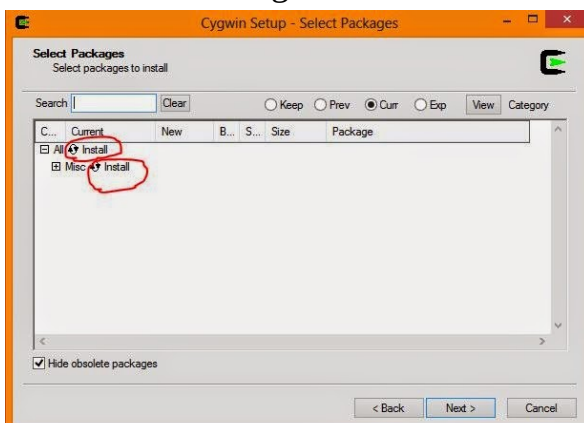
- Press Next
- Don't change the default Root Directory
- In Local Package Repository field browse to: NS 2.35/nslocal/release



- Click Next
- Click on the Circle



- After clicking the circle the default turn to install

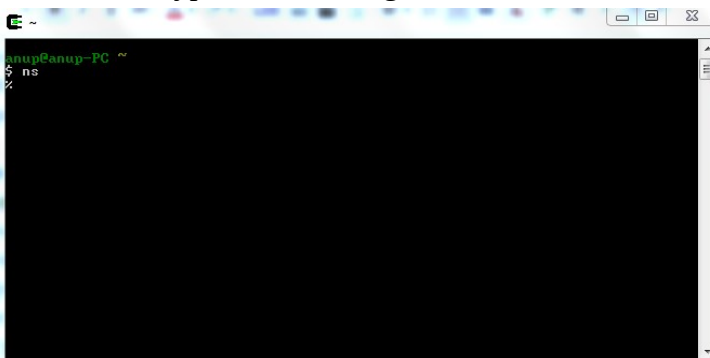


- After this press Next it will install the necessary packages for NS2

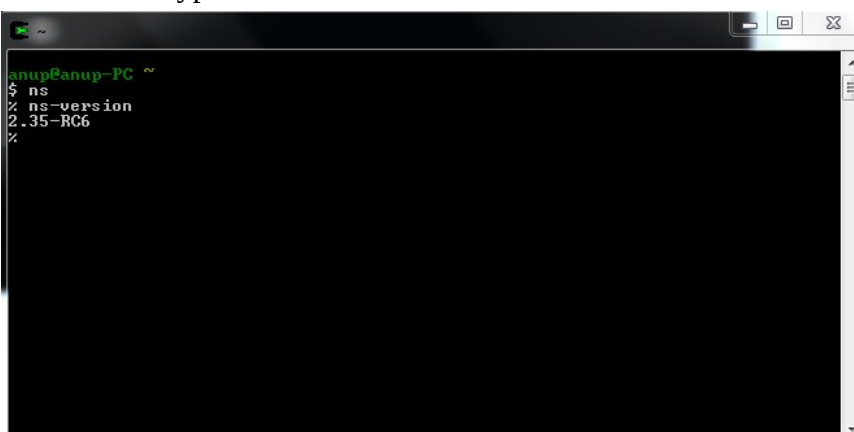
- check create a desktop shortcut and start menu shortcut
- Then installation finished.:)p
- Now you are halfway done !!!
- Now You'll get a screen like this:



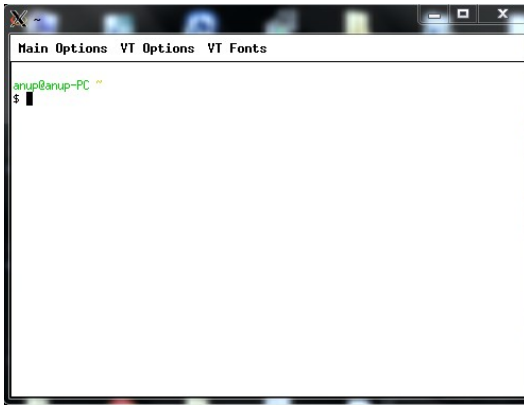
- Now create a folder Named Nouredidinein C:\cygwin\home
- Now copy the ns-allinone-2.35-RC7avecxgraph.rarand Extract there. You'll get ns-allinone-2.35-RC7 folder.
- Now cpoy the .bashrc file from the downloaded Extract folder
- Now go to the "C:\cygwin\home\prittam" here"prittam" indicate my user name in your case it will be your user name.And replace(i.e paste) the .bashrc file that you copied.
- Now open the Cygwin command window which is on your desktop and type source .bashrc
- Now type ns "\$" change to "%"



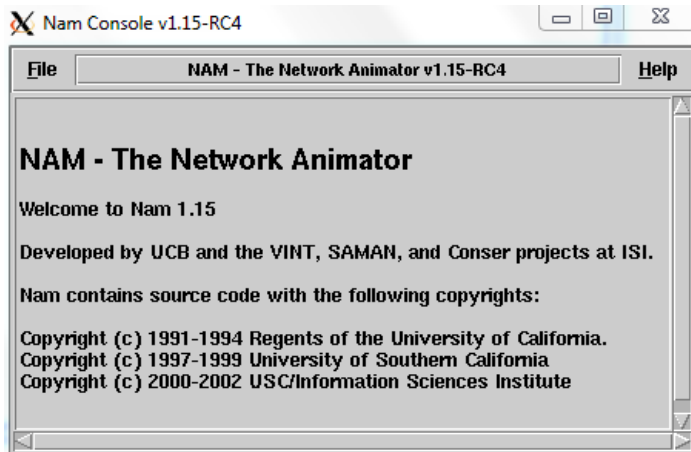
- Now type ns-version it will show like:



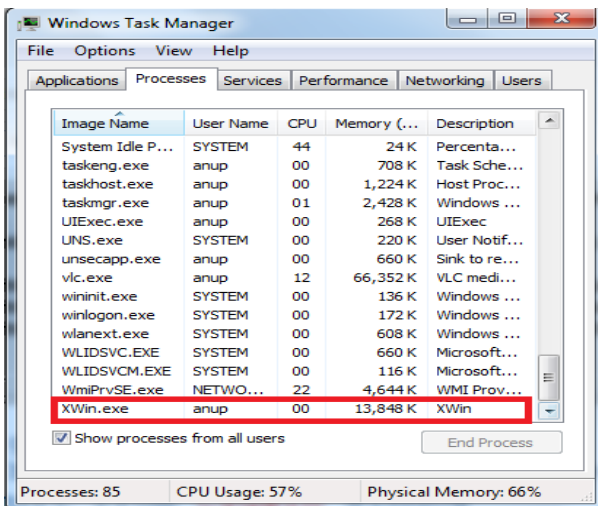
- Now press exit.
- Now type startxwin you'll get a window like this:



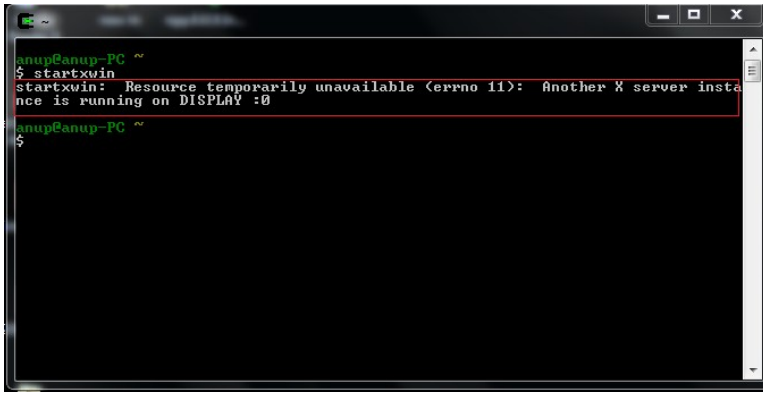
- Now type nam you'll get a screen:



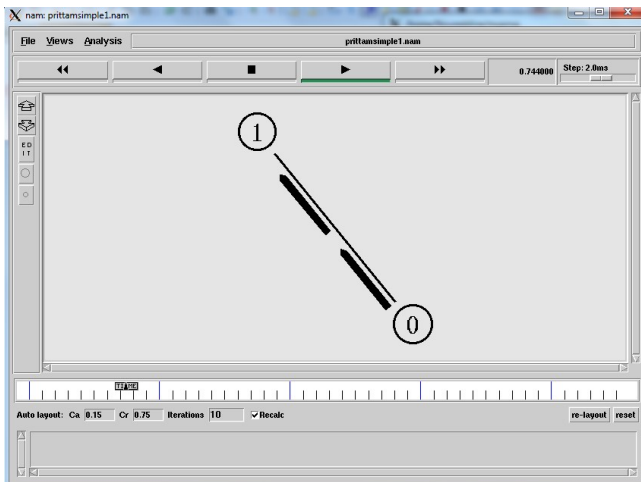
- Now go to file and quit the nam window
- Now hold the main tab and drag your mouse to quit from Cygwin window(the white window)
- And exit from 'cygwin command' window (typeexit)
- Important:
- go to Taskmanager(Alt+Ctrl+Del) and goto process tab and search for XWin.exe



- End process
- If you not do this when you press startxwin inCygwin command window for the second time then you get this:



- Now open the cygwin command window startxwin and type ns yourfilename.tcl then nam file is generated and type nam yournamfile.nam
- For my case :



### TCL Script:-

```
set ns [new Simulator]
```

```
set namfile [open ex_05a.nam w]
```

```
$ns namtrace-all $namfile
```

```
set tracefile [open ex_05a.tr w]
```

```
$ns trace-all $tracefile
```

```
Agent/TCP set packetSize_ 1460
```

```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

```
set n2 [$ns node]
```

```
set n3 [$ns node]
```

```
set n4 [$ns node]
```

```
set n5 [$ns node]
```

```

set n6 [$ns node]

set n7 [$ns node]

$n1 color blue

$n1 shape box

$n0 color red

$ns duplex-link $n0 $n2 1Mb 10ms DropTail

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link-op $n1 $n2 orient right-up

$ns duplex-link $n2 $n3 0.5Mb 10ms DropTail

$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link $n3 $n4 1Mb 10ms DropTail

$ns duplex-link-op $n3 $n4 orient right-up

$ns duplex-link $n3 $n5 1Mb 10ms DropTail

$ns duplex-link-op $n3 $n5 orient right-down

$ns duplex-link $n4 $n6 1Mb 10ms DropTail

$ns duplex-link-op $n4 $n6 orient right

$ns duplex-link $n5 $n7 1Mb 10ms DropTail

$ns duplex-link-op $n5 $n7 orient right

set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n6 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

```



```

set null [new Agent/Null]

$ns attach-agent $n7 $null

$ns connect $udp $null

$udp set class_ 1

$ns color 1 Blue

$tcp set class_ 2

$ns color 2 Red

set cbr [new Application/Traffic/CBR]

$cbr set packetSize_ 500

$cbr set interval_ 0.005

$cbr attach-agent $udp

$ns at 0.0 "$cbr start"

$ns at 9.0 "$cbr stop"

set filesize [expr 4*1024*1024]

$ns at 0.0 "$ftp send $filesize"

proc finish {}

{

global ns namfile tracefile

$ns flush-trace

close $namfile

close $tracefile

exec nam ex_05a.nam &

exit 0

}

$ns at 100.0 "finish"

$ns run

```

Note:-

In the underlined line put the bandwidth from 0.5Mb, 0.75Mb, 1.0Mb, 1.25Mb, 1.5Mb, 1.75Mb, 2.0Mb, 2.25Mb and 2.5Mb and execute the program.

### Calculatin ThroughPut:

For every time you change the bandwidth execute the following code with this command in terminal:-  
awk -f throughput.awk ex\_05.tr

BEGIN

```
{
recvbytes = 0 starttime = $2
}

{
if ( $1 == "r" && $4 == 3 && $6 >100)
{
recvbytes += $6 endt = $2
}
}

END

{
printf(" %f \n", ((recvbytes/(endt-starttime))* (8/(1024*1024))))
}
```

### Calculatin Delay:

For every time you change the bandwidth execute the following code with this command in terminal:-  
awk -f delay.awk ex\_05.tr

BEGIN

```
{
tdelay=0 count=0 pktid=-1
}

{
if($1=="+" && $3==0 && $12>pktid)
{
pktid=$12 sTime[pktid]=$2
}
}
```

```

if($1=="r" && $4==6)
{
pktid=$12 eTime[$12]=$2 count++
}
if($1=="d")
{
eTime[$12]--1
}
}
END

{
for(i=0;i<pktid;i++)
{
delay[i]=eTime[i]-sTime[i] if(delay[i]>0) tdelay=tdelay+delay[i]
}
print("%f\n", (tdelay/count))
}

```

### **Experiment No:-12**

**Aim:- Create a static wireless network. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.**

#### **TCL Script:-**

```
set ns [new Simulator]

set namfile [open ex_06.nam w]

$ns namtrace-all $namfile

set tracefile [open ex_06.tr w]

$ns trace-all $tracefile

Agent/TCP set packetSize_ 1460

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]
```

```

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

$ns1 color blue

$ns1 shape box

$ns0 color red

$ns duplex-link $ns0 $n2 4Mb 10ms DropTail

$ns duplex-link-op $ns0 $n2 orient right-down

$ns duplex-link $ns1 $n2 4Mb 10ms DropTail

$ns duplex-link-op $ns1 $n2 orient right-up

$ns duplex-link $ns2 $n3 4Mb 10ms DropTail

$ns duplex-link-op $ns2 $n3 orient right

$ns duplex-link $ns3 $n4 4Mb 10ms DropTail

$ns duplex-link-op $ns3 $n4 orient right-up

$ns duplex-link $ns3 $n5 4Mb 10ms DropTail

$ns duplex-link-op $ns3 $n5 orient right-down

$ns duplex-link $ns4 $n6 4Mb 10ms DropTail

$ns duplex-link-op $ns4 $n6 orient right

$ns duplex-link $ns5 $n7 4Mb 10ms DropTail

$ns duplex-link-op $ns5 $n7 orient right

set tcp [new Agent/TCP]

$ns attach-agent $ns0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $ns6 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

set udp [new Agent/UDP]

```

```

$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n7 $null
$ns connect $udp $null
$udp set class_ 1
$ns color 1 Blue
$tcp set class_ 2
$ns color 2 Red
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1500
$cbr set interval_ 0.005
$cbr attach-agent $udp
$ns at 0.0 "$cbr start"
$ns at 9.0 "$cbr stop"
$ns at 0.0 "$ftp start"
$ns at 9.0 "$ftp stop"
proc finish {}
{
global ns namfile tracefile
$ns flush-trace
close $namfile
close $tracefile
set awkCode {
BEGIN{}
{
if($1 == "r" && $4 == 6 && $6 >= 1460)
{
count_bytes = count_bytes + $6 - ($6 % 1460);

```

```

print $2,count_bytes >> "prgm6_tcp0.005.data";
}

else

if($1 == "r" && $4 == 7 && $6 >= 500)
{
count_bytes = count_bytes + $6 - ($6 % 500);
print $2,count_bytes >> "prgm6_udp0.005.data";
}
}

END{}

}

exec awk $awkCode ex_06.tr

exec nam ex_06.nam &

exec xgraph -bb -tk -x Time -y Bytes_Received_at_0.5 prgm6_tcp0.005.data -bg white &
exec xgraph -bb -tk -x Time -y Bytes_Received_at_0.5 prgm6_udp0.005.data -bg white &

exit 0

}

$ns at 100.0 "finish"

$ns run

```

### **Experiment No:-13**

**Aim:- Create a mobile wireless network. Attach agents, generate both FTP and CBR traffic, and transmit the traffic. Vary the data rates and evaluate the performance using metric throughput, delay, jitter and packet loss.**

#### **TCL Script:-**

```
set ns [new Simulator]

set namfile [open ex_03.nam w]

$ns namtrace-all $namfile

set tracefile [open ex_03.tr w]

$ns trace-all $tracefile

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]
```



```

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

set n6 [$ns node]

set lan [$ns newLan "$n0 $n1 $n2 $n3 $n4 $n5 $n6" 100Mb 0.5ms LL Queue/DropTail Mac/802_3
Channel Phy/WiredPhy]

set udp [new Agent/UDP]

$ns attach-agent $n0 $udp

set null [new Agent/Null]

$ns attach-agent $n6 $null

$ns connect $udp $null

set cbr [new Application/Traffic/CBR];

$cbr set packetSize_ 500

$cbr set interval_ 0.005

$cbr attach-agent $udp

$ns at 0.0 "$cbr start"

$ns at 9.0 "$cbr stop"

proc finish {}

{

global ns namfile tracefile

$ns flush-trace close

$namfile close

$tracefile exec nam ex_03.nam &

exit 0

}

$ns at 0.25 "finish" $ns run

```

### **Experiment:-14**

**AIM:- To Perform File Transfer in Client & Server Using TCP/IP.**

ALGORITHM

CLIENT SIDE

1. Start.
2. Establish a connection between the Client and Server.
3. `Socketss=new Socket(InetAddress.getLocalHost(),1100);`
4. Implement a client that can send two requests.
  - i) To get a file from the server.
  - ii) To put or send a file to the server.
5. After getting approval from the server ,the client either get file from the server or send
6. file to the server.

## SERVER SIDE

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the 'get' request.
4. It reads the data from the input stream and writes it to a file in the server for the 'put' instruction.
5. Exit upon client's request.
6. Stop.

## PROGRAM

### CLIENT SIDE

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

public class FileClient{

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

        int filesize=6022386; // filesize temporary hardcoded

        long start = System.currentTimeMillis();

        int bytesRead;

        int current = 0;

        // localhost for testing

        Socket sock = new Socket("127.0.0.1",13267);

        System.out.println("Connecting...");

        // receive file

        byte [] mybytearray = new byte [filesize];

        InputStream is = sock.getInputStream();

        FileOutputStream fos = new FileOutputStream("source-copy.pdf");

        BufferedOutputStream bos = new BufferedOutputStream(fos);

        bytesRead = is.read(mybytearray,0,mybytearray.length);
```

14

```

current = bytesRead;

// thanks to A. Cádiz for the bug fix

do {

bytesRead =

is.read(mybytearray, current, (mybytearray.length-current));

if(bytesRead >= 0) current += bytesRead;

} while(bytesRead > -1);

bos.write(mybytearray, 0 , current);

bos.flush();

long end = System.currentTimeMillis();

System.out.println(end-start);

bos.close();

sock.close();

}}

```

## SERVER SIDE

```

import java.net.*;

import java.io.*;

public class FileServer

{

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

        ServerSocket servsock = new ServerSocket(13267);

        while (true) {

            System.out.println("Waiting...");

            Socket sock = servsock.accept();

            System.out.println("Accepted connection : " + sock);

            File myFile = new File ("source.pdf");

            byte [] mybytearray = new byte [(int)myFile.length()];

            FileInputStream fis = new FileInputStream(myFile);

```

```

BufferedInputStream bis = new BufferedInputStream(fis);
bis.read(mybytearray,0,mybytearray.length);
OutputStream os = sock.getOutputStream();
System.out.println("Sending...");
os.write(mybytearray,0,mybytearray.length);
os.flush();
sock.close();
}}}

```

15

## OUTPUT

### SERVEROUTPUT

C:\Program Files\Java\jdk1.6.0\bin>javac FServer.java

C:\Program Files\Java\jdk1.6.0\bin>java FServer

Waiting for clients...

Connection Established

Client wants file:network.txt

### CLIENTOUTPUT

C:\Program Files\Java\jdk1.6.0\bin>javac FClient.java

C:\Program Files\Java\jdk1.6.0\bin>java FClient

Connection request.....Connected

Enter the filename: network.txt

Computer networks: A computer network, often simply referred to as a network, is a collection of computers and devices connected by communications channels that facilitates communications among users and allows users to share resources with other user

## RESULT

Thus the File transfer Operation is done & executed successfully.

## **Experiment:-15**

**Aim:-To implement Address Resolution Protocol .**

**ALGORITHM:-**

**CLIENT SIDE:-**

1. Establish a connection between the Client and Server.

Socket ss=new Socket(InetAddress.getLocalHost(),1100);

2. Create instance output stream writer

PrintWriter ps=new PrintWriter(s.getOutputStream(),true);

3. Get the IP Address to resolve its physical address.
4. Send the IPAddress to its output Stream.ps.println(ip);
5. Print the Physical Address received from the server.

#### **SERVER SIDE:-**

1. Accept the connection request by the client.

```
ServerSocket ss=new ServerSocket(2000);Socket s=ss.accept();
```

2. Get the IPAddress from its inputstream.

```
BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
```

```
ip=br1.readLine();
```

3. During runtime execute the processRuntime r=Runtime.getRuntime();

```
Process p=r.exec("arp -a "+ip);
```

4. Send the Physical Address to the client.

#### **PROGRAMME:-**

##### **ARP CLIENT:-**

```
importjava.io.*;
```

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

```
class ArpClient
```

```
{
```

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

```
{
```

```
try
```

```
{
```

```
Socketss=newSocket(InetAddress.getLocalHost(),1100);
```

```
PrintStream ps=new PrintStream(ss.getOutputStream());
```

```
BufferedReaderbr=newBufferedReader(newInputStreamReader(System.in));
```

```
Stringip;
```

```
System.out.println("Enter the IPADDRESS:");
```

```
ip=br.readLine();
```

```

ps.println(ip);

Stringstr,data;

BufferedReaderbr2=newBufferedReader(newInputStreamReader(ss.getInputStream()));

System.out.println("ARP From Server::");

do

{

str=br2.readLine();

System.out.println(str);

}

while(!(str.equalsIgnoreCase("end")));

}

catch(IOException e)

{

System.out.println("Error"+e);

}}}
```

### **ARP SERVER:-**

```

importjava.io.*;

import java.net.*;

classArpServer

{

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

{

try

{

ServerSocketss=newServerSocket(1100);

Sockets=ss.accept();

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

BufferedReaderbr1=newBufferedReader(newInputStreamReader(s.getInputStream()));
```



```

String ip;

ip=br1.readLine();

Runtime r=Runtime.getRuntime();

Process p=r.exec("arp-a "+ip);

BufferedReaderbr2=newBufferedReader(newInputStreamReader(p.getInputStream()));

Stringstr;

while((str=br2.readLine())!=null)

{

ps.println(str);

}}

catch(IOException e)

{

System.out.println("Error"+e); }}

24

```

## **OUTPUT**

```

C:\NetworkingPrograms>javaArpServer
C:\NetworkingPrograms>javaArpClient
Enterthe IPADDRESS:
192.168.11.58
ARPFromServer::
Interface: 192.168.11.57 on Interface 0x1000003
InternetAddress PhysicalAddress Type
192.168.11.58 00-14-85-67-11-84 dynamic

```

## **RESULT**

Thus the implementation of ARP is done & executed successfully.

## **Experiment No-16**

**AIM:-To download a webpage using Java**

**ALGORITHM:-**

**CLIENT SIDE:-**

1) Start the program.

- 2) Create a socket which binds the Ip address of server and the port address to acquire service.
- 3) After establishing connection send the url to server.
- 4) Open a file and store the received data into the file.
- 5) Close the socket.
- 6) End the program.

#### **SERVER SIDE:-**

- 1) Start the program.
- 2) Create a server socket to activate the port address.
- 3) Create a socket for the server socket which accepts the connection.
- 4) After establishing connection receive url from client.
- 5) Download the content of the url received and send the data to client.
- 6) Close the socket.
- 7) End the program.

#### **PROGRAMME:-**

```
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class Client{
    public static void main(String args[]) throws Exception{
        Socket soc;
        BufferedImage img = null;
        soc=new Socket("localhost",4000);
        System.out.println("Client is running. ");
```

```

try {
System.out.println("Reading image from disk. ");
img = ImageIO.read(new File("digital_image_processing.jpg"));
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ImageIO.write(img, "jpg", baos);
baos.flush();
byte[] bytes = baos.toByteArray();
baos.close(); System.out.println("Sending image to server. ");
OutputStream out = soc.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);
dos.writeInt(bytes.length);
dos.write(bytes, 0, bytes.length);
System.out.println("Image sent to server. ");
dos.close();
out.close();
}catch (Exception e) {
System.out.println("Exception: " + e.getMessage());
soc.close();
}soc.close();
}}

```

### **SERVER PROGRAM:-**

```

import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;

class Server {
public static void main(String args[]) throws Exception{
ServerSocket server=null;
Socket socket;
server=new ServerSocket(4000);
System.out.println("Server Waiting for image");
socket=server.accept();
System.out.println("Client connected.");

```

```
InputStream in = socket.getInputStream();
DataInputStream dis = new DataInputStream(in);
int len = dis.readInt();
System.out.println("Image Size: " + len/1024 + "KB");
byte[] data = new byte[len];
dis.readFully(data);
dis.close();
in.close();
InputStream ian = new ByteArrayInputStream(data);
BufferedImage bImage = ImageIO.read(ian);
JFrame f = new JFrame("Server");
ImageIcon icon = new ImageIcon(bImage);
JLabel l = new JLabel();
l.setIcon(icon);
f.add(l);
f.pack();
f.setVisible(true); }}
```