**SRI VENKATESWARA COLLEGE OF ENGINEERING**

**(AUTONOMOUS)**

**(Affiliated to J.N.T. University Anantapur, Ananthapuramu, Accredited by NBA & NAAC “A”)**

### Karakambadi Road, Tirupati - 517 507

** DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

CN LAB MANUAL

**(CS20APC503)Computer Networks Laboratory**

**Regulation – R20**

**Academic Year (2024 – 25)**

**Year / Semester : III / I**

**Prepared by**

**N.NARAYANAMMA , Assistant Professor**

****

**Syllabus for (R20 Regulations)**

**Computer Networks Laboratory**

**(CS20APC503)**

List of Experiments

1. Work with the commands Ping, Tracert, Ipconfig, pathping, telnet, ftp, getmac, ARP,

Hostname, Nbtstat, netdiag, and Nslookup

2. Use Sniffers for monitoring network communication (Ethereal)

3. Find all the IP addresses on your network. Unicast, Multicast, and Broadcast on your

Network.

4. Use Packet tracer software to build network topology and configure using

Distance vector routing protocol.

5. Use Packet tracer software to build network topology and configure using Link

State Routing protocol.

6. Using JAVA RMI Write a program to implement Basic Calculator

7. Implement a Chatting application using JAVA TCP and UDP sockets.

8. 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 neighbor. Implement Hello

And Echo commands using JAVA.

9. Use ethereal tool to capture the information about packets.

10. 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, and jitter and packet loss.

11. 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, and jitter and packet loss.

12. 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.

**SRI VENKATESWARA COLLEGE OF ENGINEERING**

**(AUTONOMOUS)**

**(Affiliated to J.N.T. University Anantapur, Ananthapuramu, Accredited by NBA & NAAC “A”)**

### Karakambadi Road, Tirupati - 517 507

**Computer Networks Laboratory**

## (INDEX)

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Week wise** | **Name of the Experiment** | **Page No** |
| 1 | Week-1 | Work with the commands Ping, Tracert, Ipconfig, pathping, telnet, ftp, getmac, ARP,  Hostname, Nbtstat, netdiag, and Nslookup | 5-7 |
| 2 | Week-2 | Use Sniffers for monitoring network communication (Ethereal) | 8-9 |
| 3 | Week-3 | Find all the IP addresses on your network. Unicast, Multicast, and Broadcast on your Network.  . | 10-11 |
| 4 | Week-4 | Use Packet tracer software to build network topology and configure using Distance vector routing protocol. | 12-15 |
| 5 | Week-5 | Use Packet tracer software to build network topology and configure using Link State Routing protocol. | 16-20 |
| 6 | Week-6 | Using JAVA RMI Write a program to implement Basic Calculator | 21-25 |
| 7 | Week-7 | Implement a Chatting application using JAVA TCP and UDP sockets. | 26-28 |
| 8 | Week-8 | 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 neighbor. Implement Hello  And Echo commands using JAVA | 29 |
| **9** | Week-9 | Use ethereal tool to capture the information about packets | 30-37 |
| **10** | Week-10 | 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, and jitter and packet loss. | 38-39 |
| **11** | Week-11 | 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, and jitter and packet loss. | 40-42 |
| **12** | Week-12 | 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 | 43-48 |

# GENERAL INSTRUCTIONS FOR LABORATORY CLASSES

**DO‘S**

1. Without Prior permission do not enter into the Laboratory.
2. While entering into the LAB students should wear their ID cards.
3. The Students should come with proper uniform.
4. Students should sign in the LOGIN REGISTER before entering into the laboratory.
5. Students should come with observation and record note book to the laboratory.
6. Students should maintain silence inside the laboratory.
7. After completing the laboratory exercise, make sure to shutdown the system properly

**DONT‘S**

1. Students bringing the bags inside the laboratory.
2. Students wearing slippers/shoes insides the laboratory.
3. Students using the computers in an improper way.
4. Students scribbling on the desk and mishandling the chairs.
5. Students using mobile phones inside the laboratory.
6. Students making noise inside the laboratory.

|  |  |
| --- | --- |
| **EXP NO: 1** | **Networking commands** |
| **Date:** |

**AIM:** Work with the commands Ping, Tracert, Ipconfig, pathping, telnet, ftp, getmac, ARP, Hostname, Nbtstat, netdiag, and Nslookup

### Procedure:

### 1. ****Ping****

* **Usage**: ping [hostname or IP address]
* **Description**: Sends ICMP Echo Request packets to a target IP address or domain to check if it's reachable. It also measures round-trip time.
* **Example**: ping google.com — This will check if Google’s servers are reachable.

### 2. ****Tracert****

* **Usage**: tracert [hostname or IP address]
* **Description**: Traces the path that packets take to reach a target host. It shows the routers the data passes through and the time taken for each hop.
* **Example**: tracert google.com — This will show the path packets take to reach Google.

### 3. ****Ipconfig****

* **Usage**: ipconfig
* **Description**: Displays detailed network configuration information, including IP address, subnet mask, and gateway.
* **Example**: ipconfig — displays the IP configuration of all network adapters.
* **Additional Options**:
  + ipconfig /all — Shows detailed information for all network interfaces.
  + ipconfig /release — Releases the current DHCP lease.
  + ipconfig /renew — Renews the DHCP lease.
  + ipconfig /flushdns — Clears the DNS cache.

### 4. ****Pathping****

* **Usage**: pathping [hostname or IP address]
* **Description**: A combination of ping and tracert. It provides information about network latency and packet loss along the route to a target.
* **Example**: pathping google.com — Analyzes packet loss and latency from your computer to Google.

### 5. ****Telnet****

* **Usage**: telnet [hostname or IP address] [port]
* **Description**: Establishes a connection to a remote system over the Telnet protocol. It is often used to check if a port is open on a remote server.
* **Example**: telnet example.com 80 — Connects to a web server on port 80.

### 6. ****FTP****

* **Usage**: ftp [hostname or IP address]
* **Description**: Starts an FTP (File Transfer Protocol) session to transfer files between computers.
* **Example**: ftp example.com — Connects to an FTP server for file transfers.

### 7. ****Getmac****

* **Usage**: getmac
* **Description**: Displays the MAC (Media Access Control) address of all network interfaces on the local machine.
* **Example**: getmac — Shows the MAC address for each network adapter.

### 8. ****ARP****

* **Usage**: arp -a
* **Description**: Displays or modifies the ARP (Address Resolution Protocol) cache. The ARP cache maps IP addresses to MAC addresses.
* **Example**: arp -a — Shows the current ARP cache.

### 9. ****Hostname****

* **Usage**: hostname
* **Description**: Displays the name of the current machine or device.
* **Example**: hostname — Shows the local machine's hostname.

### 10. ****Nbtstat****

* **Usage**: nbtstat -a [hostname]
* **Description**: Displays NetBIOS over TCP/IP information. It's useful for troubleshooting network connectivity and looking up the NetBIOS name of a computer.
* **Example**: nbtstat -a 192.168.1.1 — Displays NetBIOS information for the device with IP 192.168.1.1.

### 11. ****Netdiag****

* **Usage**: netdiag
* **Description**: A diagnostic tool that checks the health of the network environment. It is often used to troubleshoot network connectivity issues.
* **Example**: netdiag — Runs a set of diagnostic tests on your network setup.

### 12. ****Nslookup****

* **Usage**: nslookup [hostname or IP address]
* **Description**: Queries DNS servers to obtain domain name or IP address mappings. It's useful for troubleshooting DNS-related issues.
* **Example**: nslookup google.com — Queries DNS to find the IP address of google.com.

#### Practical Examples of How These Commands Can Be Used:

1. **Network Troubleshooting**:
   * If you're unable to reach a website, you might use ping to check if it’s reachable, tracert to trace the path, and nslookup to check the DNS resolution.
   * If you suspect issues with routing or packet loss, use pathping.
2. **Check Network Configuration**:
   * Use ipconfig to confirm your device’s network settings and ensure it's correctly connected to the network.
3. **Testing Network Services**:
   * Use telnet to check if a specific port is open (e.g., check if port 80 is open on a web server).
4. **Device Identification**:
   * Use getmac to retrieve the MAC address of your network card or a remote device.
5. **Name Resolution**:
   * Use nbtstat for NetBIOS name resolution on a local network.
6. **Remote Connectivity**:
   * Use ftp to transfer files to and from remote servers or telnet for testing connectivity to remote services.

# Result:

Thus the Configure Internet connection and use IPCONFIG, PING / Tracer and Net stat utilities to establish interconnection between systems have been done successful.

|  |  |
| --- | --- |
| **EXP NO: 2** | **monitoring network communication** |
| **Date:** |

**AIM:** Use Sniffers for monitoring network communication (Ethereal)

### Steps to Use Wireshark (formerly Ethereal) for Monitoring Network Communication:

#### 1. **Install Wireshark**

* Download and install Wireshark from the official website: Wireshark Download.
* During installation, ensure that you install the necessary network drivers (like WinPcap or Npcap) that allow Wireshark to capture network traffic.

#### 2. **Start Wireshark**

* Launch the **Wireshark** application.
* You'll be presented with a list of available network interfaces on your system (e.g., Ethernet, Wi-Fi, etc.). Choose the network interface you want to monitor.

#### 3. **Select a Network Interface**

* If you are on a wired network, select your **Ethernet** adapter.
* If you are on a wireless network, select your **Wi-Fi** adapter.
* In Wireshark, click on the interface you want to capture traffic from. You will see the live traffic displayed in the main window.

#### 4. **Start Capturing Network Traffic**

* Once you've selected the interface, click the **"Start Capturing Packets"** button (shaped like a shark fin).
* Wireshark will begin displaying real-time network traffic.
* The traffic is captured in packets, and each packet contains detailed information about the communication, such as the source and destination IP addresses, ports, protocols, and more.

#### 5. **Apply Filters**

* Wireshark allows you to filter the captured traffic to focus on specific types of communication.
* To filter traffic, use **display filters** in the filter bar at the top. Here are some common filters:
  + **ip.addr == [IP Address]** — Filters packets for a specific IP address.
  + **tcp.port == 80** — Filters packets on HTTP (port 80).
  + **udp.port == 53** — Filters packets on DNS (port 53).
  + **http** — Filters only HTTP packets.
  + **ip.src == [IP Address]** — Filters packets from a specific source IP.
  + **ip.dst == [IP Address]** — Filters packets to a specific destination IP.
* Example: To capture only traffic to and from Google’s servers, you can filter by ip.addr == 172.217.0.0/16.

#### 6. **Inspect Packet Details**

* When you select a packet, you can see detailed information, including:
  + **Packet Headers**: Displays detailed breakdowns of protocols used (Ethernet, IP, TCP, UDP, HTTP, etc.).
  + **Payload**: The actual data transmitted in the packet.
  + **Protocol Hierarchy**: The various protocols and layers involved in the communication.

#### 7. **Analyze the Traffic**

* **Protocol Analysis**: Wireshark decodes different protocols so you can analyze network behavior, detect network performance issues, and identify potential security risks (e.g., suspicious traffic or vulnerabilities).
* **Timing Analysis**: You can also see the timing between packets, which can help diagnose issues like network delays or performance bottlenecks.
* **Packet Reconstruction**: Wireshark can reconstruct conversations (such as HTTP sessions, FTP sessions) to help you analyze the full communication flow.

#### 8. **Stop Capturing**

* To stop the packet capture, click the **"Stop Capturing"** button (red square).
* Once stopped, you can review the captured packets in more detail.

#### 9. **Save the Capture File**

* You can save the captured traffic for later analysis by selecting **File → Save As**.
* Wireshark saves captures in .pcap (Packet Capture) format, which can be reopened later for inspection.

#### 10. **Export Packet Data**

* If needed, you can export specific data or packets to a different format for analysis or reporting. Go to **File → Export Specified Packets** to do this.

### Example Scenario: Capturing HTTP Requests

1. **Start Wireshark** and select your network interface.
2. In the **filter bar**, enter http to capture only HTTP traffic.
3. Visit a website in your browser, like http://example.com.
4. In Wireshark, you'll see HTTP packets (e.g., GET requests, HTTP responses).
5. Click on a packet to view details of the HTTP request, including headers, cookies, and any data sent.

**Result**: Wireshark is a powerful tool for network analysis and troubleshooting, providing deep insights into how devices and services communicate over a network. It is an essential tool for network administrators, security experts, and developers looking to understand or diagnose network behavior.

|  |  |
| --- | --- |
| **EXP NO: 3** | **Find all the IP addresses on your network** |
| **Date:** |

**AIM:** To find all the IP addresses on your network. Unicast, Multicast, and Broadcast on your network.

### Description:

There are two ways to find IP addresses on network systems. You can find them manually or you can use an IP scanner, which is designed to automatically find the IP addresses within a certain range.

With a scanner such as SolarWinds IPAM, you can run automated scans to identify new devices and more easily manage IP addresses.

The basic steps for manually creating a list of device IP addresses on a network include:

* Get to the command line by opening a terminal window.
* Type the right command for your system. On Linux, type the command “ifconfig” and press Return. On Windows, type the command “ipconfig” and press Return.
* Get more information by typing the command “arp -a.”
* You should now see a basic list of the IP addresses for devices connected to your network.
* You can then input this information into an information storage tool, like a spreadsheet, that you’ll need to update by hand each time you attempt a new discovery.

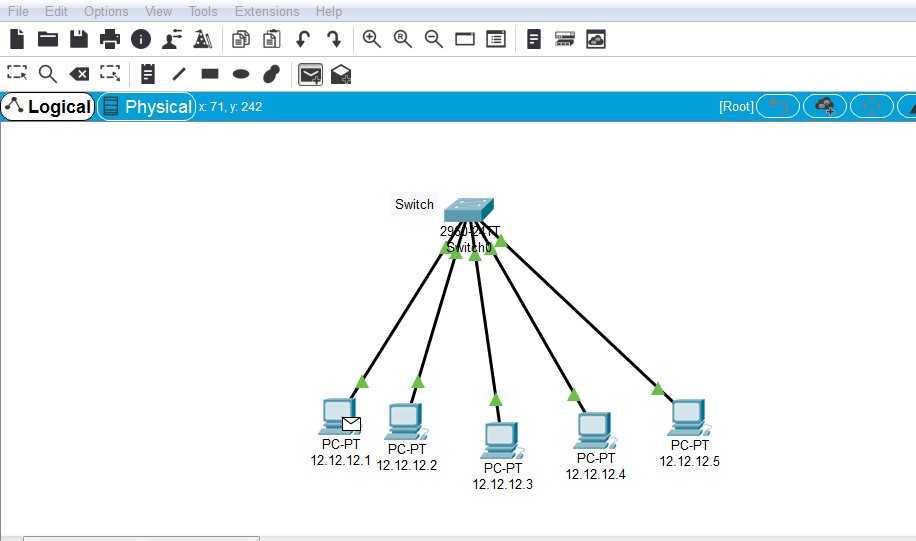
This process is unnecessarily time consuming and vulnerable to errors. Not only are you forced to manually run commands to find IP addresses whenever you want to see the devices connected to your network, but you also need to identify which of the IP addresses you find are new or have changed.

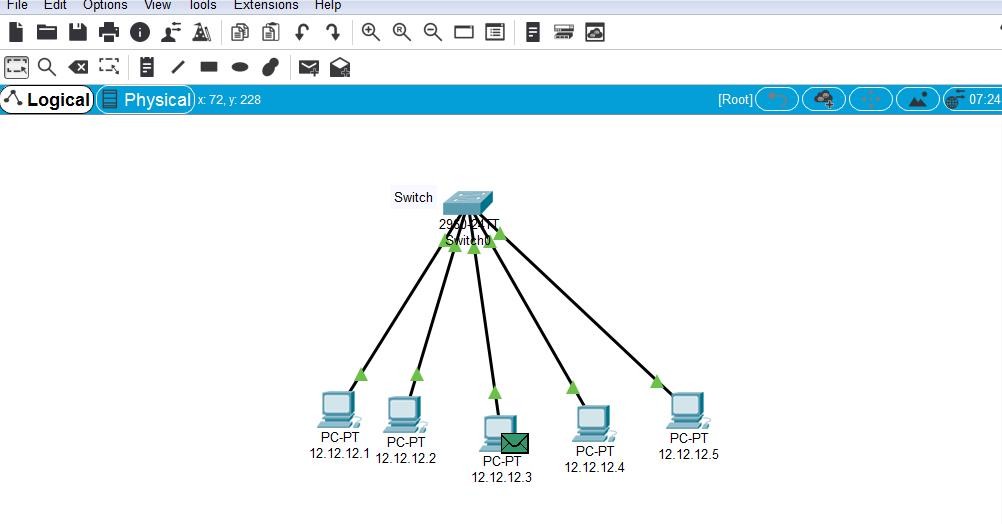
# Use an IP discovery tool to automatically locate devices and collect critical data

SolarWinds® IP Address Manager (IPAM) is designed to eliminate the need to manually manage IP address data with complicated spreadsheets. By quickly identifying and updating device information, an automated IP address discovery tool can help make IP discovery quicker and a more accurate process.

SolarWinds IPAM is built to scan your network for all IP addresses, checks their statuses, and reports the details. IP Address Manager also uses automated subnet discovery and neighbor discovery techniques to help identify and accurately manage IP subnets, address blocks, and connected hosts to present real-time IP address information.

### Output:

****

****

|  |  |
| --- | --- |
| **EXP NO: 4** | **Packet tracer software to build network topology** |
| **Date:** |

**AIM:** To Use Packet tracer software to build network topology and configure using Distance vector routing protocol.

### Description:

It is under dynamic routing algorithm. This algorithm operates by having each route maintains a table giving the least known distance to reach destination and include line in used to get these. These are updated by changing information with neighbour. This is called “Bell mann ford algorithm” and “fod fick” algorithm.

**build a network topology in Packet Tracer** and **configure it using a Distance Vector Routing Protocol (RIP)**:

### 1. ****Open Packet Tracer and Create a New Project****

* Launch **Cisco Packet Tracer**.
* Create a new project by selecting **File → New** from the top menu.

### 2. ****Create a Basic Network Topology****

* **Add Devices**:
  + **Routers**: Drag and drop two or more routers onto the workspace (from the **Routers** category in the bottom-left panel).
  + **Switches**: Drag and drop switches (from the **Switches** category) to connect devices.
  + **PCs**: Add PCs from the **End Devices** section. These will be connected to the routers or switches.
* **Connect Devices**:
  + Use **copper straight-through cables** to connect routers and switches, and **copper crossover cables** to connect PCs to switches or routers.
  + Connect the devices by clicking on the **cable** icon, selecting the appropriate cable type, and clicking on the interfaces to connect them.

### 3. ****Assign IP Addresses to Devices****

Each device in your topology will need an IP address to communicate with others:

* **PC Configuration**:
  + Click on each **PC** and go to the **Desktop** tab.
  + Select **IP Configuration** and assign an IP address and subnet mask to the PC’s network adapter.
  + Example:
    - PC1: IP address 192.168.1.2, subnet mask 255.255.255.0.
    - PC2: IP address 192.168.2.2, subnet mask 255.255.255.0.
  + Repeat this for all the PCs, ensuring they are on the same subnet as their corresponding router.
* **Router Interfaces**:
  + Click on each **Router** and go to the **CLI (Command Line Interface)** tab.
  + Assign IP addresses to the router interfaces connected to the PCs and the other routers.
  + Example:
    - Router1 (Interface GigabitEthernet0/0): 192.168.1.1/24.
    - Router1 (Interface GigabitEthernet0/1): 192.168.2.1/24.
    - Router2 (Interface GigabitEthernet0/0): 192.168.2.2/24.
    - Router2 (Interface GigabitEthernet0/1): 192.168.3.1/24.

### 4. ****Enable RIP on the Routers****

The Distance Vector Routing protocol **RIP** (Routing Information Protocol) allows routers to exchange routing tables. You will configure RIP on each router to enable communication between the devices across the different subnets.

#### Steps to Configure RIP on Routers:

* **Access Router CLI**:
  + Click on **Router1** (or any other router), go to the **CLI** tab.
  + Enter the configuration mode by typing enable and then configure terminal.
* **Enable RIP**:
  + To enable RIP on Router1, type the following commands:

Router1# configure terminal

Router1(config)# router rip

Router1(config-router)# version 2

Router1(config-router)# network 192.168.1.0

Router1(config-router)# network 192.168.2.0

**5.Enable RIP on Router2:**

* Repeat the same steps for **Router2**, with the appropriate network addresses:
* Router2# configure terminal
* Router2(config)# router rip
* Router2(config-router)# version 2
* Router2(config-router)# network 192.168.2.0
* Router2(config-router)# network 192.168.3.0
* Router2(config-router)# exit

### ****Verify RIP Configuration****

After configuring RIP on both routers, it's important to verify that the routers have learned each other’s routes and can communicate properly.

* **Show IP Routes**:
  + On **Router1**, enter the following command:

Router1# show ip route

**Ping Between PCs**:

* From **PC1** (which is connected to Router1), go to **Desktop → Command Prompt** and type:

ping 192.168.2.2

 This will check if **PC1** can communicate with **PC2** (connected to Router2).

 Similarly, ping from **PC2** to **PC1**.

### ****6.Optional: Troubleshoot the Network****

* If the PCs cannot ping each other:
  + Verify the IP addresses and subnet masks on the PCs and routers.
  + Check if RIP is properly configured and that both routers have the correct network statements for the relevant networks.
  + Make sure that both routers are using **RIP version 2** (use show ip protocols to verify).

### 7. ****Save Your Configuration****

After configuring the network and verifying the communication, save your work:

* On each router, use the following command to save the configuration

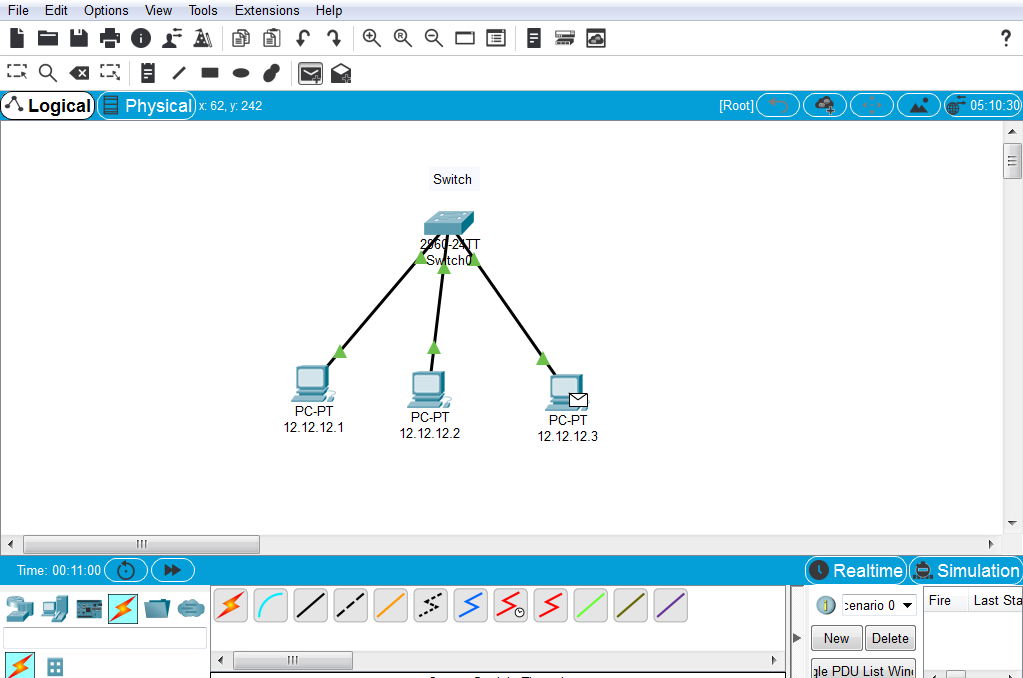
Router1# copy running-config startup-config

Router2# copy running-config startup-config

### Example Configuration Summary:

* **Router1**:
  + 192.168.1.1/24 on interface connected to **PC1**.
  + 192.168.2.1/24 on interface connected to **Router2**.
  + RIP configuration includes network 192.168.1.0 and network 192.168.2.0.
* **Router2**:
  + 192.168.2.2/24 on interface connected to **Router1**.
  + 192.168.3.1/24 on interface connected to **PC2**.
  + RIP configuration includes network 192.168.2.0 and network 192.168.3.0.

### Output:

****

|  |  |
| --- | --- |
| **EXP NO: 5** | **Link State routing protocol** |
| **Date:** |

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

### Procedure:

In **Packet Tracer**, you can configure **Link-State Routing Protocols** like **Open Shortest Path First (OSPF)**, which is widely used in real networks for its scalability and efficiency in calculating the best paths. **OSPF** (Open Shortest Path First) is a Link-State Routing Protocol that works by having routers send information about their own interfaces to neighboring routers. This information is then used to build a comprehensive view of the network, allowing each router to make intelligent routing decisions.

### Steps to Build a Network Topology and Configure OSPF in Packet Tracer:

### ****1. Open Packet Tracer and Create a New Project****

* **Open Cisco Packet Tracer**.
* Select **File → New** to create a new project.

### ****2. Create a Basic Network Topology****

For this example, let's create a simple network with two routers, two switches, and two PCs, and configure OSPF to route between the routers.

#### **Add Devices to the Workspace:**

1. **Routers**: Drag two routers (e.g., **Router0** and **Router1**) from the **Routers** section onto the workspace.
2. **Switches**: Add two switches (e.g., **Switch0** and **Switch1**) from the **Switches** section.
3. **PCs**: Add two PCs (e.g., **PC0** and **PC1**) from the **End Devices** section.
4. **Connections**: Use **copper straight-through cables** to connect:
   * Router0 to Switch0
   * Router1 to Switch1
   * PC0 to Switch0
   * PC1 to Switch1
   * Router0 to Router1

### ****3. Configure IP Addresses on Devices****

Each device needs an IP address to communicate. We'll assign IP addresses on the PCs and router interfaces.

#### **PC Configuration**:

1. **PC0**:
   * Go to **PC0 → Desktop → IP Configuration**.
   * Assign the following IP address:
     + **IP Address**: 192.168.1.2
     + **Subnet Mask**: 255.255.255.0
   * Leave the **Default Gateway** empty for now; it will be filled automatically when OSPF is configured.
2. **PC1**:
   * Go to **PC1 → Desktop → IP Configuration**.
   * Assign the following IP address:
     + **IP Address**: 192.168.2.2
     + **Subnet Mask**: 255.255.255.0

#### **Router Configuration**:

* **Router0**:
  + Click **Router0** → go to **CLI**.
  + Enter the following commands to configure Router0:
  + Router0> enable
  + Router0# configure terminal
  + Router0(config)# interface gig0/0
  + Router0(config-if)# ip address 192.168.1.1 255.255.255.0
  + Router0(config-if)# no shutdown
  + Router0(config-if)# exit
  + Router0(config)# interface gig0/1
  + Router0(config-if)# ip address 192.168.3.1 255.255.255.0
  + Router0(config-if)# no shutdown
  + Router0(config-if)# exit
* **Router1**:
  + Click **Router1** → go to **CLI**.
  + Enter the following commands to configure Router1:
  + Router1> enable
  + Router1# configure terminal
  + Router1(config)# interface gig0/0
  + Router1(config-if)# ip address 192.168.3.2 255.255.255.0
  + Router1(config-if)# no shutdown
  + Router1(config-if)# exit
  + Router1(config)# interface gig0/1
  + Router1(config-if)# ip address 192.168.2.1 255.255.255.0
  + Router1(config-if)# no shutdown
  + Router1(config-if)# exit

### ****4. Configure OSPF (Open Shortest Path First) on Routers****

OSPF is a link-state routing protocol that allows routers to share detailed information about the network's topology.

#### **Configure OSPF on Router0**:

1. On **Router0**, go to the **CLI** and enter the following commands:
2. Router0# configure terminal
3. Router0(config)# router ospf 1
4. Router0(config-router)# network 192.168.1.0 0.0.0.255 area 0
5. Router0(config-router)# network 192.168.3.0 0.0.0.255 area 0
6. Router0(config-router)# exit
   * Explanation:
     + router ospf 1: Configures OSPF process 1.
     + network [IP address] [wildcard mask] area 0: This command tells the router to advertise the network in the specified area (area 0 in this case).
     + wildcard mask: A wildcard mask is the inverse of the subnet mask. In this case, for 192.168.1.0/24, the wildcard mask is 0.0.0.255.

#### **Configure OSPF on Router1**:

1. On **Router1**, go to the **CLI** and enter the following commands:
2. Router1# configure terminal
3. Router1(config)# router ospf 1
4. Router1(config-router)# network 192.168.2.0 0.0.0.255 area 0
5. Router1(config-router)# network 192.168.3.0 0.0.0.255 area 0
6. Router1(config-router)# exit
   * Similar to Router0, the network command configures OSPF on the networks directly connected to Router1.

### ****5. Verify OSPF Configuration****

After configuring OSPF on both routers, verify that the routers are exchanging OSPF information and have learned the routes.

#### **Check OSPF Neighbor Relationships**:

* On **Router0**, enter the following command to check if the OSPF neighbor relationship has been established:
* Router0# show ip ospf neighbor
  + This will display the OSPF neighbors. If successful, you should see **Router1** listed as a neighbor.
* On **Router1**, enter the same command:
* Router1# show ip ospf neighbor

#### **Check Routing Table**:

* On **Router0**, use the following command to check the routing table:
* Router0# show ip route ospf
  + This will display routes learned via OSPF.
* On **Router1**, use the following command:
* Router1# show ip route ospf

### ****6. Test Connectivity Between PCs****

* From **PC0**, ping **PC1** to test the connectivity between different subnets:
  + Go to **PC0 → Desktop → Command Prompt** and type:
  + ping 192.168.2.2
  + If OSPF is configured correctly and the routers have exchanged routing information, **PC0** should successfully ping **PC1**.
* Similarly, you can test the reverse (ping from **PC1** to **PC0**).

### ****7. Save the Configuration****

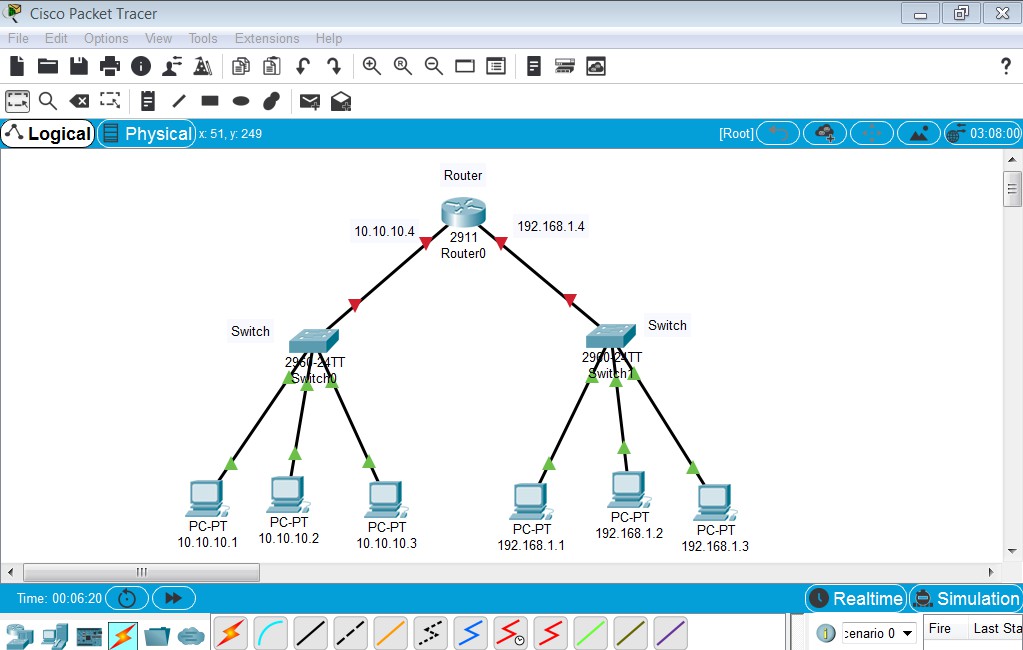
Once everything is working, don’t forget to save your configuration on the routers:

* On **Router0**:
* Router0# copy running-config startup-config
* On **Router1**:
* Router1# copy running-config startup-config

### ****Summary of Configuration****

* **Router0**:
  + Interfaces:
    - 192.168.1.1/24 on gig0/0 (connected to **PC0**).
    - 192.168.3.1/24 on gig0/1 (connected to **Router1**).
  + OSPF Networks:
    - network 192.168.1.0 0.0.0.255 area 0
    - network 192.168.3.0 0.0.0.255 area 0
* **Router1**:
  + Interfaces:
    - 192.168.3.2/24 on gig0/0 (connected to **Router0**).
    - 192.168.2.1/24 on gig0/1 (connected to **PC1**).
  + OSPF Networks:
    - network 192.168.2.0 0.0.0.255 area 0
    - network 192.168.3.0 0.0.0.255 area 0

### Output:

****

### ****Result:****

Successfully created a simple network topology in Packet Tracer and configured **OSPF** as the **Link-State Routing Protocol**. This allows for efficient routing between different networks with dynamic updates as the network topology changes

|  |  |
| --- | --- |
| **EXP NO: 6** | **JAVA RMI Basic Calculator** |
| **Date:** |

**AIM:** To using JAVA RMI write a program to implement Basic Calculator.

RMI called as remote method innovaction, by using rmi to develop calculator program in java

### Program:

Calc.java

import java.rmi.Remote;

import java.rmi.RemoteException; public interface Calc extends Remote

{

public long addition(long a, long b)throws RemoteException;

public long subtraction(long a, long b) throws RemoteException;

public long multiplication(long a, long b) throws RemoteException;

public long divition(long a, long b) throws RemoteException;

}

Calcimpl.java

public long multiplication(long a,long b)throws RemoteException

{

return a\*b;

}

public long divition(long a, long b)throws java.rmi.RemoteException

{

return a/b;

}

}

Calcserv.java

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class Calcimpl extends UnicastRemoteObject implements Calc

{

protected Calcimpl()throws RemoteException

{

super();

}

public long addition(long a,long b)throws RemoteException

{

return a+b;

}

public long subtraction(long a,long b)throws RemoteException

{

return a-b;

}

import java.rmi.Naming;

public class CalcServer

{

CalcServer()

{

try

{

Calc c = new Calcimpl(); Naming.rebind("rmi://localhost:1099/CalculatorService", c);

}

catch (Exception e)

{

System.out.println("Exception:"+e);

}

}

public static void main(String args[])

{

new CalcServer();

}

}

CalculatorClient

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

{

Calc c = (Calc)

Naming.lookup("rmi://localhost/CalculatorService");

System.out.println( c.subtraction(4, 3) ); System.out.println( c.addition(4, 5) ); System.out.println( c.multiplication(3, 6) ); System.out.println( c.divition(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.ArithmeticExceptionae)

{

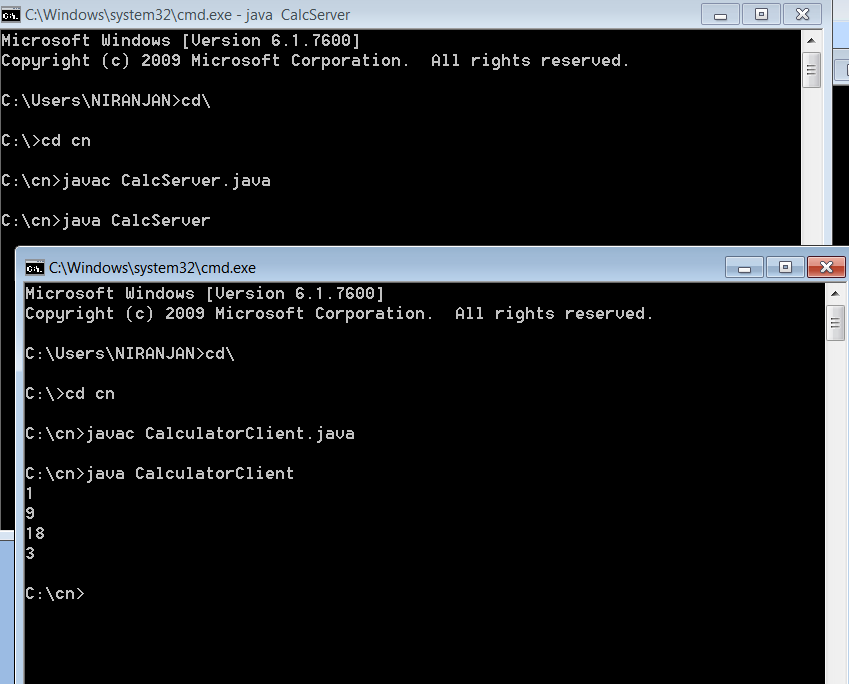
System.out.println(); System.out.println("java.lang.ArithmeticException"); System.out.println(ae);

}

}

}

### Output:



|  |  |
| --- | --- |
| **EXP NO: 7** | **Chatting application using JAVA TCP and UDP sockets** |
| **Date:** |

**AIM:** Implement a Chatting application using JAVA TCP and UDP sockets.

### Procedure:

1. In any Client/Server Application, we need to run the server before the client, because the server keeps waiting for the client to be connected.
2. Server keeps listening for the client on an assigned IP & Port
3. For establishing connection client must know the IP & Port of the server.
4. When we start Client Application, It creates a connection to the server.
5. After the Successful connection Client & Server Applications can send & receive messages.

### Source Code:

**charsever.java**

import java.net.\*; import java.io.\*; public class chatserver

{

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

{

ServerSocket ss=new ServerSocket(2000); Socket sk=ss.accept();

BufferedReader cin=new BufferedReader(new InputStreamReader(sk.getInputStream()));

PrintStream cout=new PrintStream(sk.getOutputStream());

BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in)); String s;

while ( true )

{

s=cin.readLine();

if (s.equalsIgnoreCase("END"))

{

cout.println("BYE"); break;

}

System. out.print("Client : "+s+"\n"); System.out.print("Server : "); s=stdin.readLine();

cout.println(s);

}

ss.close();

sk.close();

cin.close();

cout.close(); stdin.close();

}

}

**chatclient.java**

import java.net.\*; import java.io.\*; public class chatclient

{

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

{

Socket sk=new Socket("127.0.0.1",2000); BufferedReader sin=new BufferedReader(new InputStreamReader(sk.getInputStream()));

PrintStream sout=new PrintStream(sk.getOutputStream());

BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in)); String s;

while ( true )

{

System.out.print("Client : "); s=stdin.readLine(); sout.println(s); s=sin.readLine();

System.out.print("Server : "+s+"\n"); if ( s.equalsIgnoreCase("BYE") ) break;

}

sk.close();

sin.close();

sout.close(); stdin.close();

}

**output:**

Server:

E:\nwlab>javac \*.java E:\nwlab>java chatserver Client : hi

Server : hi Client:

E:\nwlab>java chatclient Client : hi

Server : hi Client :

|  |  |
| --- | --- |
| **EXP NO: 8** | **Hello and Echo commands using JAVA** |
| **Date:** |

**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.

# Source Code:

import java.util.Scanner; public class Echo

{

public static void main (String[] args)

{

String inData;

Scanner scan = new Scanner( System.in );

System.out.println("Enter the data:"); inData = scan.nextLine();

System.out.println("You entered:" + inData );

}

}

### Output:

C:\> javac Echo.java C:\> java Echo Enter the data:

This is what the user typed.

You entered: This is what the user typed.

|  |  |
| --- | --- |
| **EXP NO: 9** | **Inspect HTTP Traffic** |
| **Date:** |

**AIM:** Use Ethereal tool to capture the information about packets..

### Procedure:

Wireshark is a software protocol analyzer, or “packet sniffer” application, used for network troubleshooting, analysis, software and protocol development, and education. As data streams travel back and forth over the network, the sniffer “captures” each protocol data unit (PDU) and can decode and analyze its content according to the appropriate RFC or other specifications.

Wireshark is a useful tool for anyone working with networks and can be used with most labs in the CCNA courses for data analysis and troubleshooting. This lab provides instructions for downloading and installing Wireshark.

### Required Resources

* PC (Windows 7, 8, or 10 with internet access)

### Download and Install Wireshark

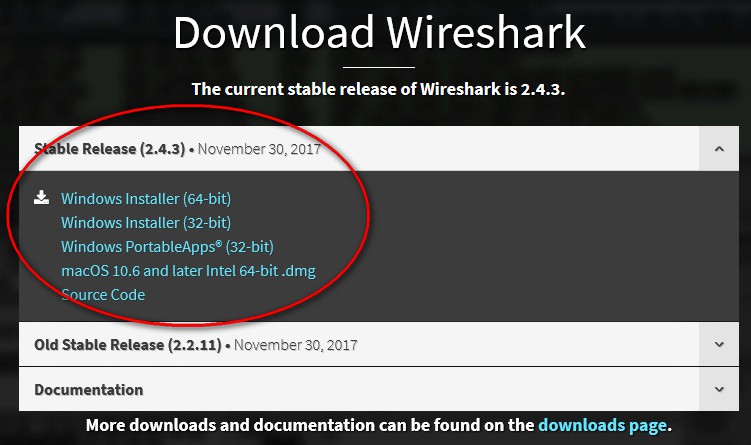
Wireshark has become the industry standard packet-sniffer program used by network engineers. This open source software is available for many different operating systems, including Windows, Mac, and Linux. In this lab, you will download and install the Wireshark software program on your PC.

### Step 1: Download Wireshark.

* 1. Wireshark can be downloaded from [www.wireshark.org.](https://www.wireshark.org/)
  2. Click the icon above **Download**



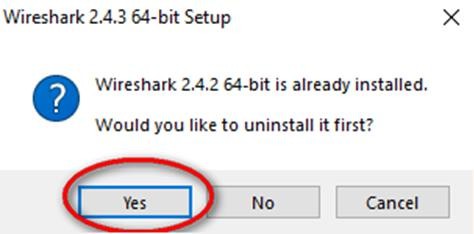
C. Choose the software version you need based on your PC architecture and operating system. For instance, if you have a 64-bit PC



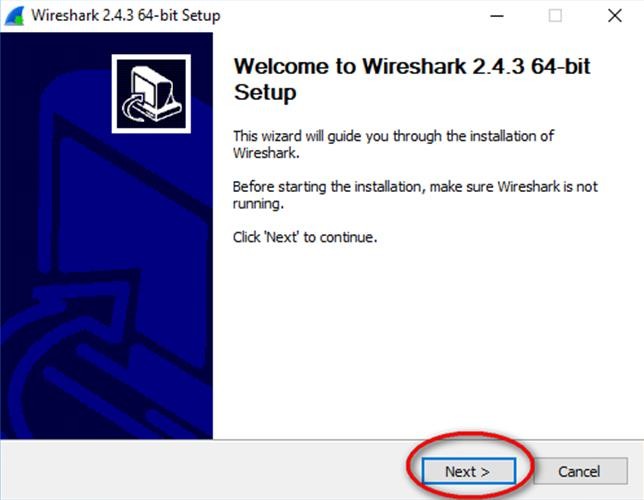
**Step 2: Install Wireshark.**

1. The downloaded file is named **Wireshark-win64-x.x.x.exe**, where **x** represents the version number. Double-click the file to start the installation process.
2. Respond to any security messages that may display on your screen. If you already have a copy of Wireshark on your PC, you will be prompted to uninstall the old version before

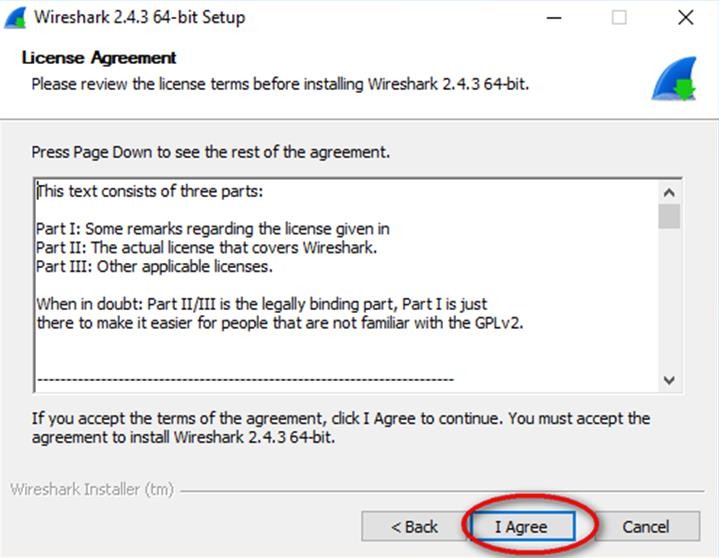
installing the new version. It is recommended that you remove the old version of Wireshark prior to installing another version. Click Yes to uninstall the previous version of Wireshark.



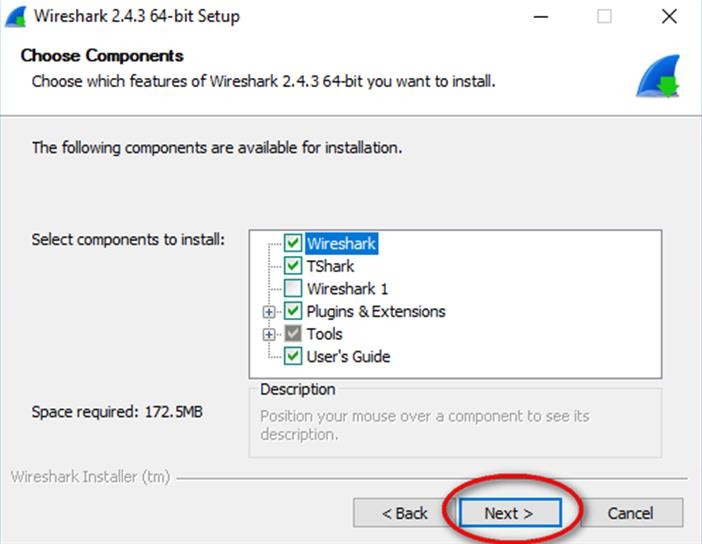
1. If this is the first time that you have installed Wireshark, or after you have completed the uninstall process, you will navigate to the **Wireshark Setup** wizard. Click **Next**.



1. Continue advancing through the installation process. Click **I Agree** when the License Agreement window displays

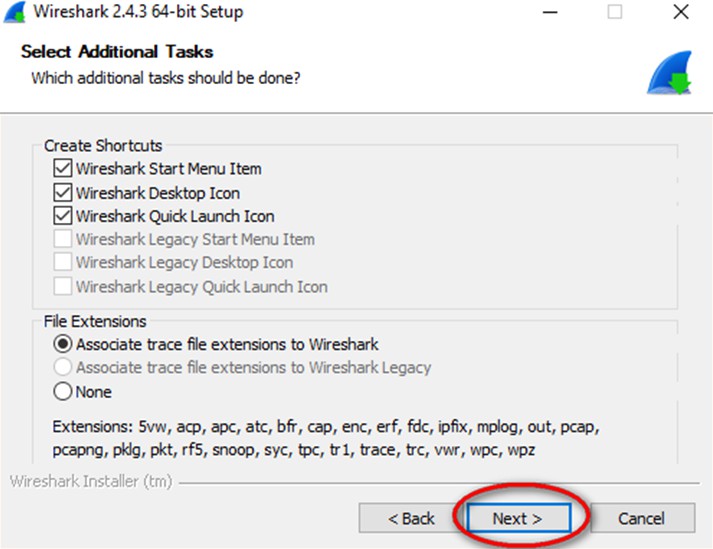


1. Keep the default settings on the **Choose Components** window and click **Next**

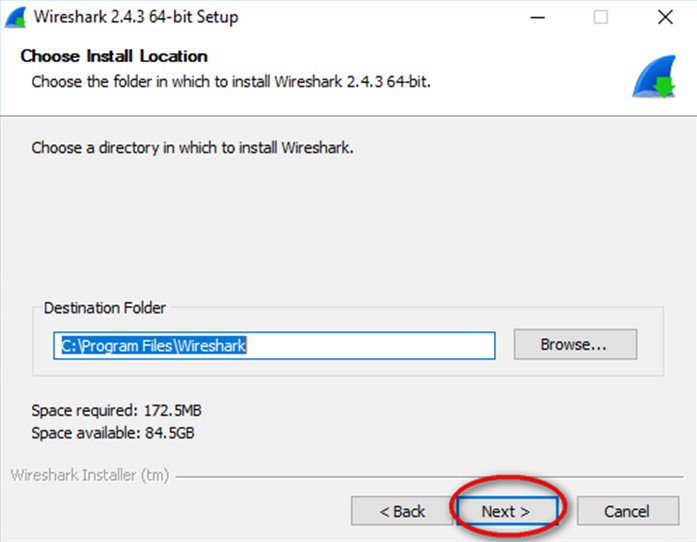


1. Choose your desired shortcut options and click **Next**.

33



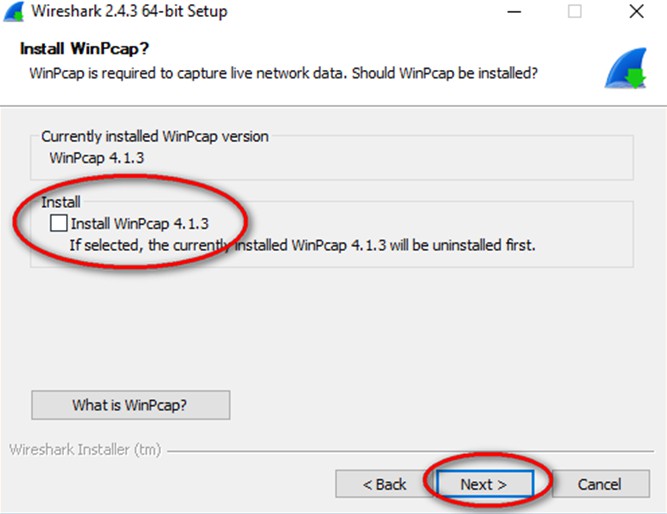
1. You can change the installation location of Wireshark, but unless you have limited disk space, it is recommended that you keep the default location.



1. To capture live network data, WinPcap must be installed on your PC. If WinPcap is already installed on your PC, the Install check box will be unchecked. If your installed version of WinPcap is older than the version that comes with Wireshark, it is recommended that you allow the newer version to be installed by clicking the Install

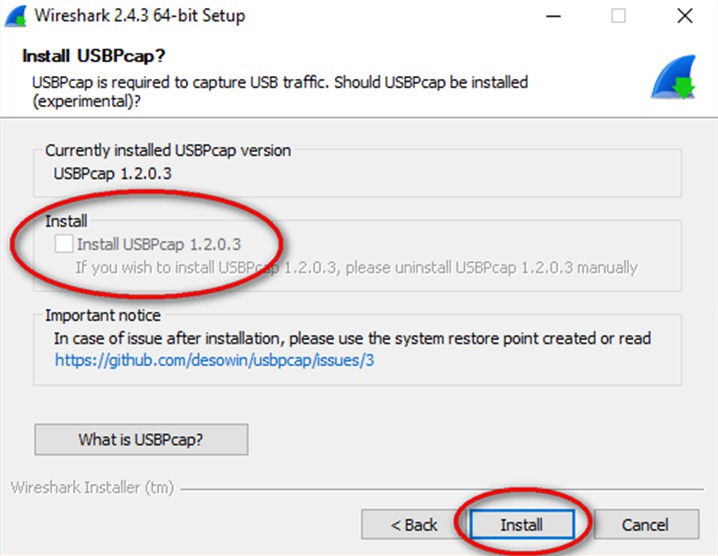
34

WinPcap x.x.x (version number) check box.

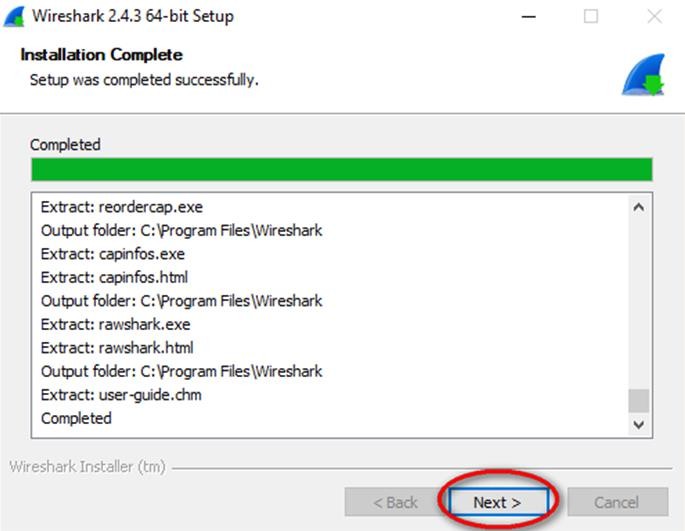
1. ****Finish the WinPcap Setup wizard if installing WinPcap
2. In addition, USBPcap can be installed on your PC. If USBPcap is already installed on your PC, the Install check box will be unchecked. If your installed version of USBPcap is older than the version that comes with Wireshark, it is recommended that you allow the newer version to be installed by clicking the **Install USBPcap x.x.x** (version number) check box.

**Note**: Because USBcap is still experimental, it is recommended that you **DO NOT** install USBcap unless you need to capture USB traffic.

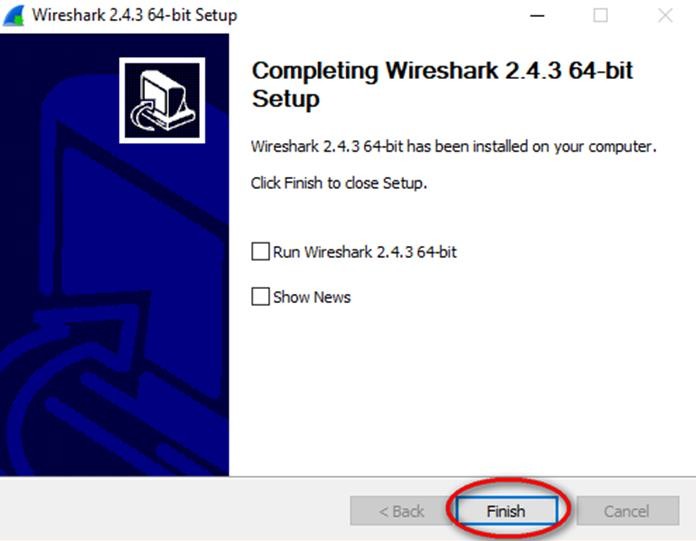
1. Finish the **USBPcap Setup** wizard if installing USBPcap



1. Wireshark starts installing its files, and a separate window displays with the status of the installation. Click **Next** when the installation is complete.



36

1. ****Click **Finish** to complete the Wireshark install process.

### Result:

37

|  |  |
| --- | --- |
| **EXP NO: 10** | **Network Simulater-2** |
| **Date:** |

**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.

### Description:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,1 the foundation which NS is based on. Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual Inter Network Testbed (VINT) project . Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers and developers in the community are constantly working to keep NS2 strong and versatile.

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. NS meets both of these needs with two languages, C++ and OTcl.

### Tcl scripting

* Tcl is a general purpose scripting language. [Interpreter]
* Tcl runs on most of the platforms such as Unix, Windows, and Mac.
* The strength of Tcl is its simplicity.
* It is not necessary to declare a data type for variable prior to the usage.

### Basics of TCL

Syntax: command arg1 arg2 arg3

### Hello World!

puts stdout{Hello, World!} Hello, World!

**Variables** Command Substitution set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

### Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc) Set the time of traffic generation (e.g., CBR, FTP) Terminate the simulation

### NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

### Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

### set ns [new Simulator]

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

### Output:

|  |  |
| --- | --- |
| **EXP NO: 11** | **Network Simulater-2 FTP & CBR Traffic** |
| **Date:** |

**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.

DESCRIPTION

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. NS meets both of these needs with two languages, C++ and OTcl.

### Tcl scripting

* Tcl is a general purpose scripting language. [Interpreter]
* Tcl runs on most of the platforms such as Unix, Windows, and Mac.
* The strength of Tcl is its simplicity.
* It is not necessary to declare a data type for variable prior to the usage.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using ―open command:

**#Open the Trace file**

**#Open the NAM trace file**

**set tracefile1 [open out.tr w]**

**$ns trace-all $tracefile1**

**set namfile [open out.nam w]**

**$ns namtrace-all $namfile**

The above creates a dta trace file called out.tr and a nam visualization trace file called out. nam.Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―namfile respectively. Remark that

they begins with a # symbol. The second line open the file ―out.tr to be used for writing, declared with the letter ―w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

**Define a “finish‟ procedure Proc finish { } {**

**global ns tracefile1 namfile**

**$ns flush-trace Close $tracefile1 Close $namfile**

**Exec nam out.nam & Exit 0**

**}**

### Definition of a network of links and nodes

The way to define a node is

### set n0 [$ns node]

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace ―duplex-link by

―simplex-link.

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

### #set Queue Size of link (n0-n2) to 20

**FTP over TCP**

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

**#Setup a UDP connection**

**set udp [new Agent/UDP]**

**$ns attach-agent $n1 $udp set null [new Agent/Null]**

**$ns attach-agent $n5 $null**

**$ns connect $udp $null**

**$udp set fid\_2**

**#setup a CBR over UDP connection**

The below shows the definition of a CBR application using a UDP agent

The command $ns attach-agent $n4 $sink defines the destination node. The command $ns connect $tcp $sink finally makes the TCP connection between the source and destination nodes.

**set cbr [new Application/Traffic/CBR]**

**$cbr attach-agent $udp**

**$cbr set packetsize\_ 100**

**$cbr set rate\_ 0.01Mb**

**$cbr set random\_ false**

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command $tcp set packetSize\_ 552.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command $tcp set fid\_ 1 that assigns to the TCP connection a flow identification of ―1.We shall later give the flow identification of ―2‖ to the UDP connection.

|  |  |
| --- | --- |
| **EXP NO: 12** | **Network Simulater-2 FTP & CBR Traffic** |
| **Date:** |

**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.

### Description:

A wireless sensor network (WSN) consists of a large number of small sensor nodes that are deployed in the area in which a factor is to be monitored. In wireless sensor network, energy model is one of the optional attributes of a node. The energy model denotes the level of energy in a mobile node. The components required for designing energy model includes initialEnergy, txPower, rxPower, and idlePower. The “initialEnergy” represents the level of energy the node has at the initial stage of simulation. “txPower” and “rxPower” denotes the energy consumed for transmitting and receiving the packets. If the node is a sensor, the energy model should include a special component called “sensePower”. It denotes the energy consumed during the sensing operation. Apart from these components, it is important to specify the communication range (RXThresh\_) and sensing range of a node (CSThresh\_). The sample 18.tcl designs a WSN in which sensor nodes are configured with different communication and sensing range. Base Station is configured with highest communication range. Data Transmission is established between nodes using UDP agent and CBR traffic.

### Algorithm:

1. Create a simulator object
2. Define setting options for wireless channel
3. Create trace file and name file
4. Setup topography object and nodes
5. Provide initial location of mobile nodes
6. Setup a UDP connection between nodes
7. Printing the window size

### Program:

# Define setting options

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation model set val(netif) Phy/WirelessPhy ;# network interface type

set val(mac) Mac/802\_11 ;# MAC type

set val(ifq) Queue/DropTail/PriQueue ;# interface queue type set val(ll) LL ;# link layer type

set val(ant) Antenna/OmniAntenna ;# antenna model set val(ifqlen) 50 ;# max packet in ifq

set val(nn) 10 ;# number of mobilenodes set val(rp) DSDV ;# routing protocol

set val(x) 500 ;# X dimension of topography set val(y) 400 ;# Y dimension of topography set val(stop) 150 ;# time of simulation end set ns [new Simulator]

#Creating trace file and nam file set tracefd [open dsdv.tr w]

set windowVsTime2 [open win.tr w] set namtrace [open dsdv.nam w]

$ns trace-all $tracefd

$ns namtrace-all-wireless $namtrace $val(x) $val(y) # set up topography object

set topo [new Topography]

$topo load\_flatgrid $val(x) $val(y) create-god $val(nn)

# configure the nodes

$ns node-config -adhocRouting $val(rp) \

-llType $val(ll) \

-macType $val(mac) \

-ifqType $val(ifq) \

-ifqLen $val(ifqlen) \

-antType $val(ant) \

-propType $val(prop) \

-phyType $val(netif) \

-channelType $val(chan) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF \

-movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } { set node\_($i) [$ns node]

}

# Provide initial location of mobilenodes

$node\_(0) set X\_ 5.0

$node\_(0) set Y\_ 800.0

$node\_(0) set Z\_ 0.0

$node\_(1) set X\_ 8.0

$node\_(1) set Y\_ 650.0

$node\_(1) set Z\_ 0.0

$node\_(2) set X\_ 60.0

$node\_(2) set Y\_ 450.0

$node\_(2) set Z\_ 0.0

$node\_(3) set X\_ 10.0

$node\_(3) set Y\_ 480.0

$node\_(3) set Z\_ 0.0 #another four

$node\_(4) set X\_ 350.0

$node\_(4) set Y\_ 500.0

$node\_(4) set Z\_ 0.0

$node\_(5) set X\_ 150.0

$node\_(5) set Y\_ 850.0

$node\_(5) set Z\_ 0.0

$node\_(6) set X\_ 200.0

$node\_(6) set Y\_ 500.0

$node\_(6) set Z\_ 0.0 #$node\_(7) set X\_ 320.0

$node\_(7) set X\_ 320.0

$node\_(7) set Y\_ 650.0

$node\_(7) set Z\_ 0.0 #another four

$node\_(8) set X\_ 250.0

$node\_(8) set Y\_ 700.0

$node\_(8) set Z\_ 0.0

$node\_(9) set X\_ 400.0

$node\_(9) set Y\_ 800.0

$node\_(9) set Z\_ 0.0 ##################################

#copy of the data ##################################

#####Attack Node

$node\_(5) color Green

$ns at 10.0 "$node\_(5) color Green"

$ns at 10.0 "$node\_(5) label EndUser"

# Set a udp connection between node\_(5) and node\_(8)

set udp [new Agent/UDP]

set sink [new Agent/LossMonitor] #$ns attach-agent $node\_(5) $udp #$ns attach-agent $node\_(8) $sink

$ns attach-agent $node\_(8) $udp

$ns attach-agent $node\_(5) $sink

$ns connect $udp $sink

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$node\_(8) color Green

$ns at 10.0 "$node\_(8) color yellow"

$ns at 10.0 "$node\_(5) label node5 "

$ns at 10.0 "$node\_(8) label node8 "

$ns at 10.0 "$cbr start"

$ns at 10.45 "$cbr stop"

# Set a udp connection between node\_(8) and node\_(6) set udp1 [new Agent/UDP]

set sink1 [new Agent/LossMonitor]

$ns attach-agent $node\_(8) $udp1

$ns attach-agent $node\_(6) $sink1

$ns connect $udp1 $sink1

set cbr1 [new Application/Traffic/CBR]

$cbr1 attach-agent $udp1

$node\_(6) color yellow

$ns at 10.0 "$node\_(6) color yellow"

$ns at 10.0 "$node\_(6) label node6 "

$ns at 10.0 "$cbr1 start"

$ns at 10.45 "$cbr1 stop"

# Set a udp connection between node\_(8) and node\_(6) set udp2 [new Agent/UDP]

set sink2 [new Agent/LossMonitor]

$ns attach-agent $node\_(5) $udp2

$ns attach-agent $node\_(8) $sink2

$ns connect $udp2 $sink2

set cbr2 [new Application/Traffic/CBR]

$cbr2 attach-agent $udp2

$node\_(6) color yellow

$ns at 12.0 "$cbr2 start"

$ns at 13.45 "$cbr2 stop"

# Set a udp connection between node\_(8) and node\_(6)

set udp3 [new Agent/UDP]

set sink3 [new Agent/LossMonitor]

$ns attach-agent $node\_(8) $udp3

$ns attach-agent $node\_(6) $sink3

$ns connect $udp3 $sink3

set cbr3 [new Application/Traffic/CBR]

$cbr3 attach-agent $udp1

$node\_(6) color yellow

$ns at 14.0 "$cbr3 start"

$ns at 16.45 "$cbr3 stop" # Printing the window size

proc plotWindow {tcpSource file} { global nss

set time 0.01

set now [$ns now]

set cwnd [$tcpSource set cwnd\_] puts $file "$now $cwnd"

$ns at [expr $now+$time] "plotWindow $tcpSource $file" } #$ns at 10.1 "plotWindow $tcp $windowVsTime2"

# Define node initial position in nam for {set i 0} {$i < $val(nn)} { incr i } { # 30 defines the node size for nam

$ns initial\_node\_pos $node\_($i) 30

}

# Telling nodes when the simulation ends for {set i 0} {$i < $val(nn) } { incr i } { #$ns at $val(stop) "$node\_($i) reset"; #$ns at 200.25 "$node\_($i) reset";

$ns at 21.25 "$node\_($i) reset";

}

$ns at 0.00 "$ns trace-annotate \"Wireless Mac Ptotocol \""

$ns at 10.0 "$ns trace-annotate \" Data send in node5 to node8 \""

$ns at 10.0 "$ns trace-annotate \" Data send in node6 to node8 \""

$ns at 10.45 "$ns trace-annotate \" Data Collision \""

$ns at 12.00 "$ns trace-annotate \"Data send in node5 to node8\""

$ns at 14.00 "$ns trace-annotate \"Data send in node6 to node8\""

$ns at 21.25 "$ns trace-annotate \"End simulation\""

$ns at 21.25 "$ns nam-end-wireless 21.01"

$ns at 21.25 "stop"

$ns at 22.01 "puts \"end simulation\" ; $ns halt"

proc stop {} {

global ns tracefd namtrace

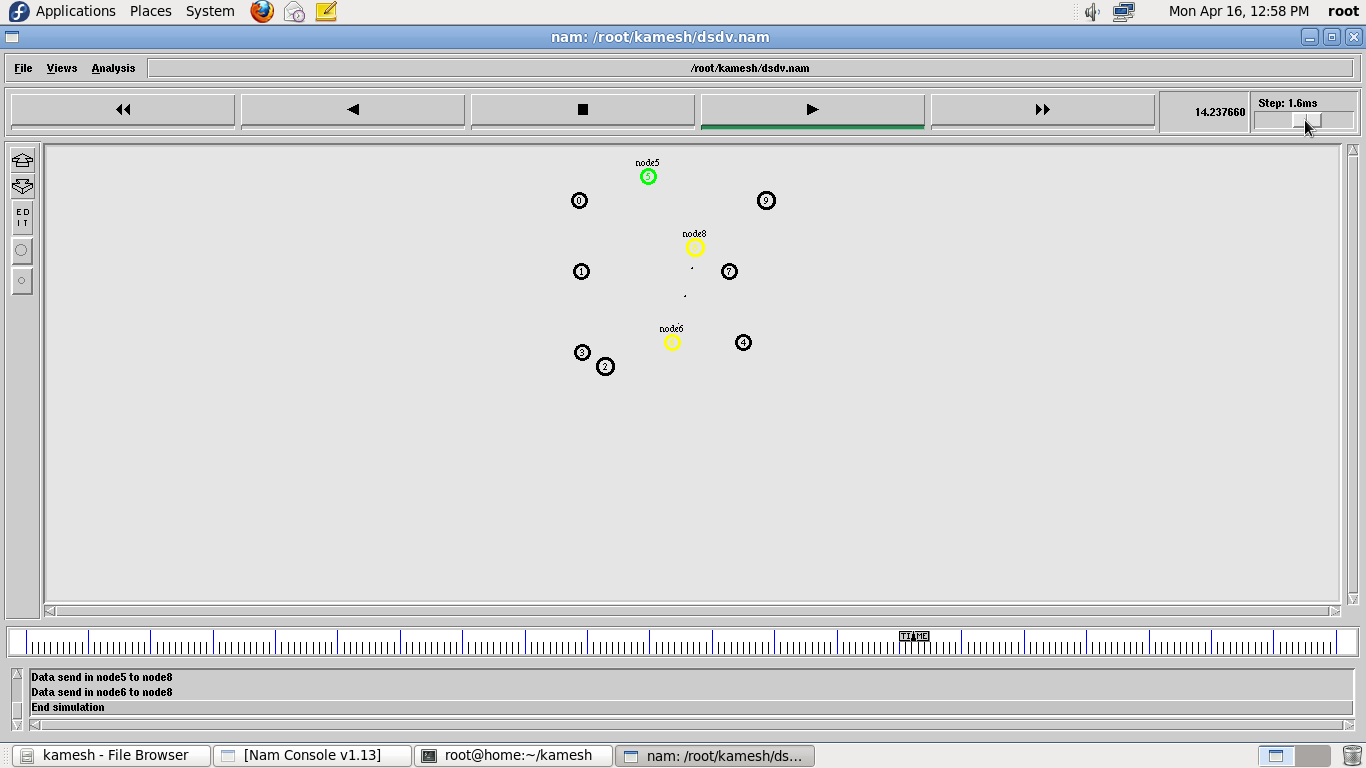
$ns flush-trace close $tracefd close $namtrace

exec nam dsdv.nam & exit 0

}

$ns run

### Expected Output:

****