

NETWORKING TRAINING - MODULE 3 & 4

Akash S, embedUR Systems

Task 1:

Simulate a small network with switches and multiple devices. Use ping to generate traffic and observe the MAC address table of the switch. Capture packets using Wireshark to analyze Ethernet frames and MAC addressing

Explanation:

- Installed GNS3 & Cisco Packet Tracer and necessary packages.
Created a new workspace inside the GNS3 & Cisco Packet Tracer.
- Added two PCs and a switch.
- Configured IP: **192.168.1.10** for PC1 and **192.168.1.20** for PC2

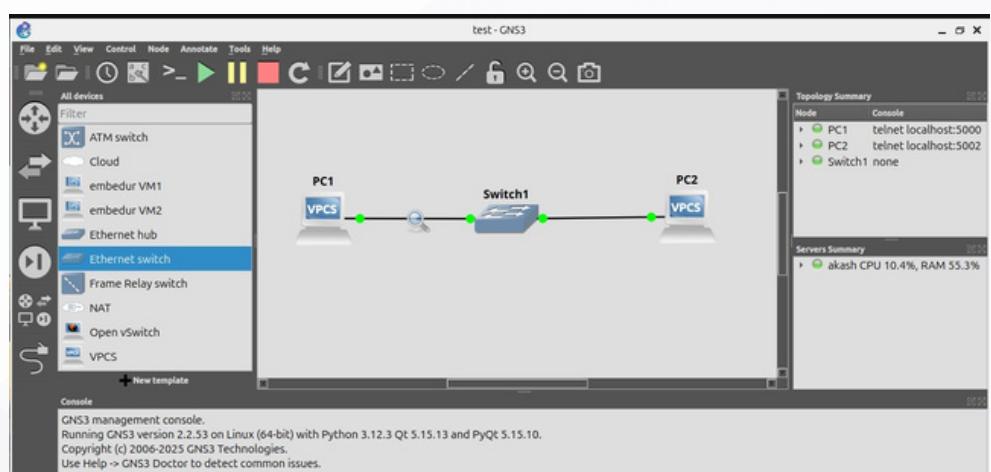


- Pinged PC2 from PC1 to check the connectivity
- Added two PCs and a switch.
- Configured IP: **192.168.1.10** for PC1 and **192.168.1.20** for PC2
- Checking the MAC address of the table using
\$ show mac address-table

- Packets are captured using Wireshark tool and Simulation is verified

Outputs:

GNS3



Capturing from - [PC1 Ethernet0 to PC2] (192.168.1.10)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Len
1	0.000000	00:50:79:66:68:00	Broadcast	ARP	64
2	0.000493	00:50:79:66:68:01	00:50:79:66:68:00	ARP	64
3	0.001144	192.168.1.10	192.168.1.20	ICMP	64
4	0.001548	192.168.1.20	192.168.1.10	ICMP	64
5	1.004639	192.168.1.10	192.168.1.20	ICMP	64
6	1.004975	192.168.1.20	192.168.1.10	ICMP	64
7	2.026875	192.168.1.10	192.168.1.20	ICMP	64
8	2.027551	192.168.1.20	192.168.1.10	ICMP	64
9	3.030868	192.168.1.10	192.168.1.20	ICMP	64
10	3.031271	192.168.1.20	192.168.1.10	ICMP	64
11	4.034643	192.168.1.10	192.168.1.20	ICMP	64
12	4.035190	192.168.1.20	192.168.1.10	ICMP	64
13	5.043249	192.168.1.10	192.168.1.20	ICMP	64

```

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^].
ip 192.168.1.10 255.255.255.0
Checking for duplicate address...
PC1 : 192.168.1.10 255.255.255.0

PC1> ping 192.168.1.20 -c 8

84 bytes from 192.168.1.20 icmp_seq=1 ttl=64 time=0.773 ms
84 bytes from 192.168.1.20 icmp_seq=2 ttl=64 time=0.614 ms
84 bytes from 192.168.1.20 icmp_seq=3 ttl=64 time=1.702 ms
84 bytes from 192.168.1.20 icmp_seq=4 ttl=64 time=1.203 ms
84 bytes from 192.168.1.20 icmp_seq=5 ttl=64 time=1.174 ms
84 bytes from 192.168.1.20 icmp_seq=6 ttl=64 time=1.140 ms
84 bytes from 192.168.1.20 icmp_seq=7 ttl=64 time=2.332 ms
84 bytes from 192.168.1.20 icmp_seq=8 ttl=64 time=1.658 ms

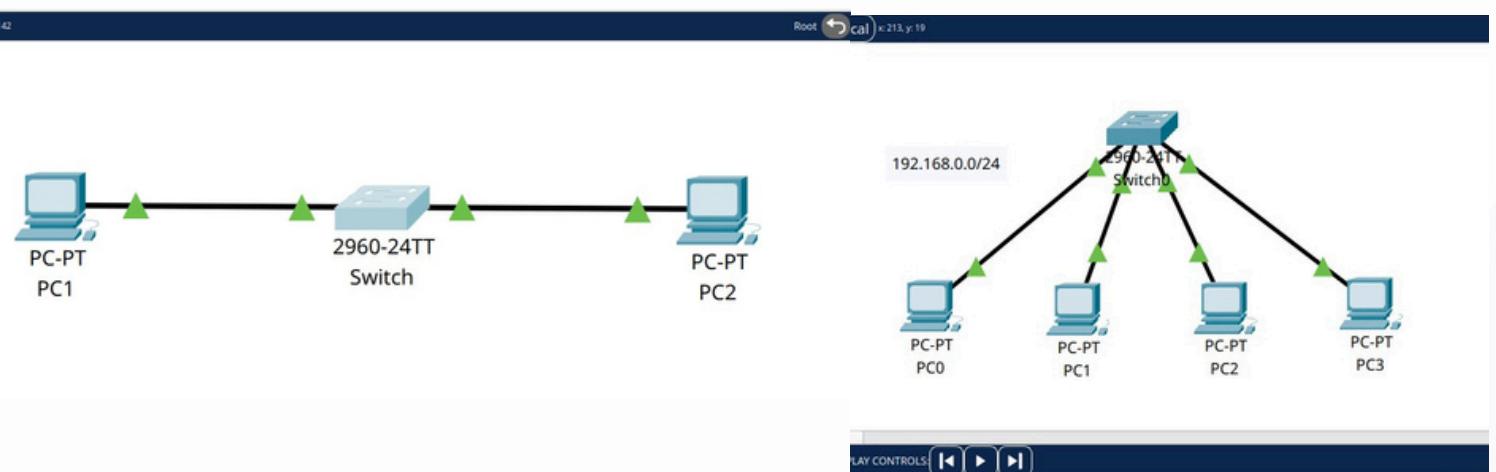
PC1> show arp

00:50:79:66:68:01 192.168.1.20 expires in 108 seconds

```

Wireshark Packet Capture - Pinging PC2 from PC1

Cisco Packet Tracer:



PC1

Physical Config Desktop Programming Attributes

Command Prompt

```

Cisco Packet Tracer PC Command Line 1.0
C:>>ping 192.168.1.20

Pinging 192.168.1.20 with 32 bytes of data:
Reply from 192.168.1.20: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.20:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
  Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:>

```

Switch

Physical Config CLI Attributes

IOS Command Line Interface

```

Switch>show mac address-table
Mac Address Table
-----
Vlan   Mac Address      Type      Ports
----  -----
  1    000d.bd34.438e  DYNAMIC   Fa0/2
  1    00e0.f9cb.7e35  DYNAMIC   Fa0/1

Switch>
Switch>
Switch>
Switch>
Switch>
Switch>
Switch>

```

MAC Address table of Switch

Task 2:

Capture and analyze Ethernet frames using Wireshark. Inspect the structure of the frame, including destination and source MAC addresses, Ethertype, payload, and FCS. Use GNS3 or Packet Tracer to simulate network traffic.

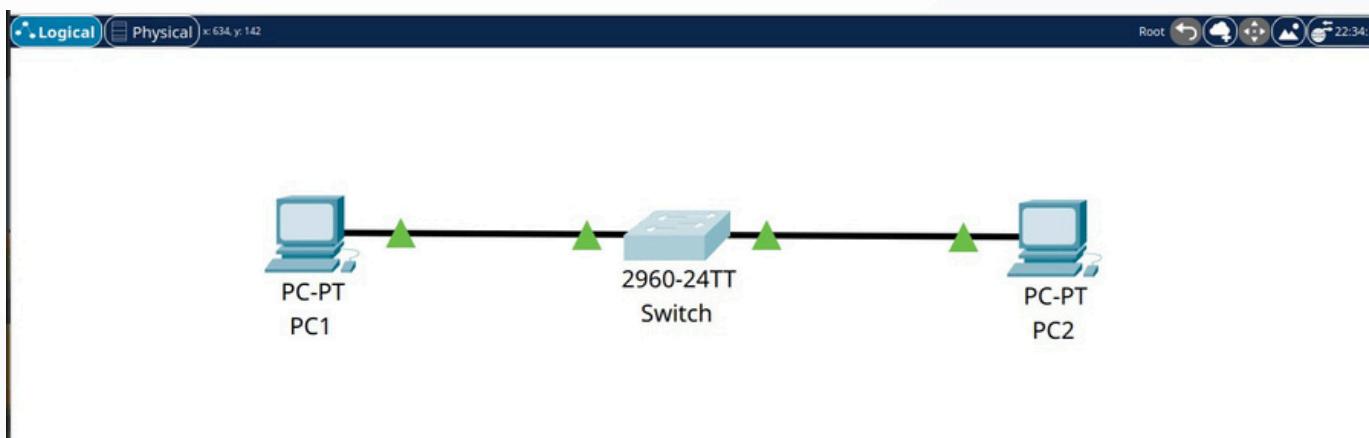
Explanation:

- In the Cisco Packet Tracer, create a network setup with two VPCs and a switch.
- Configured IP: **192.168.1.1** for PC1 and **192.168.1.2** for PC2

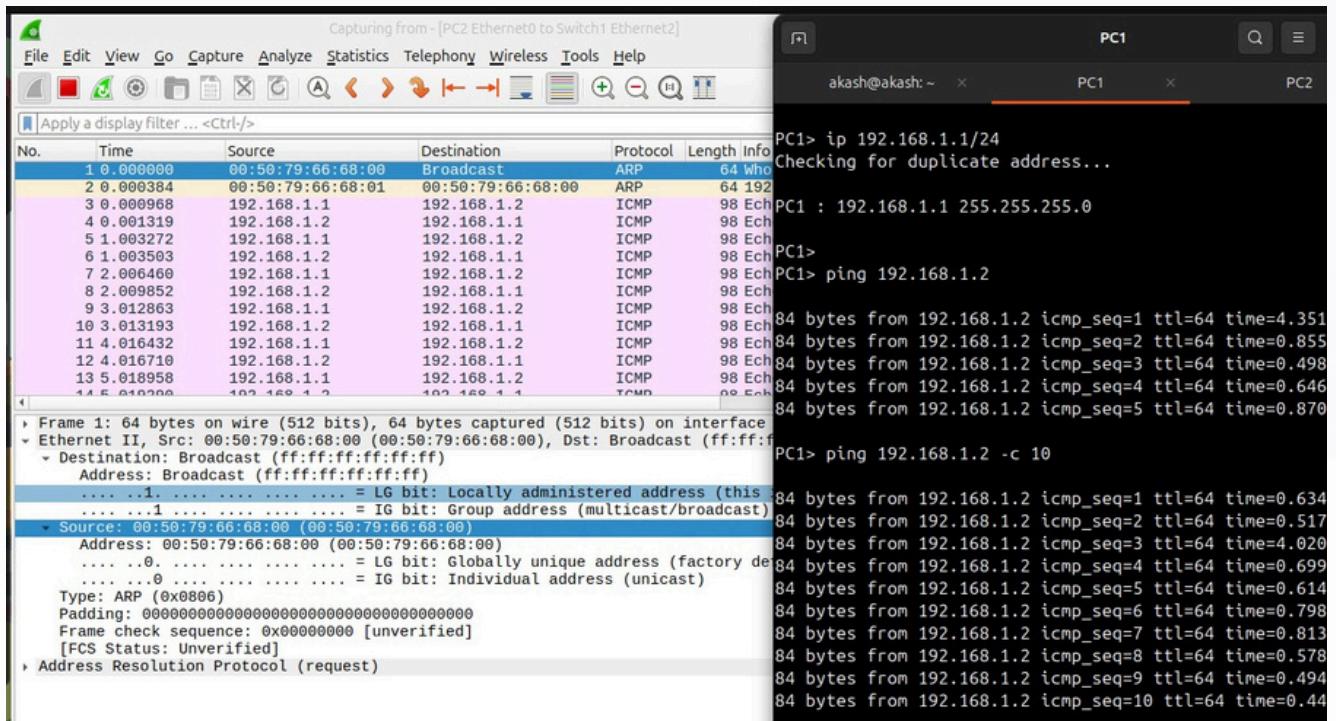


- Ping PC2 (192.168.1.2) from PC1 using Ping command
\$ ping 192.168.1.2
- Once ping is successful, open the simulation mode and play.
- We can now examine the movement of packets from switch to PCs
- We can capture the packets in **Wireshark** and on packets to see Detailed Ethernet Frame Format including Source MAC, Destination MAC, FCS, Padding, Payload, TTL, etc...

Outputs:

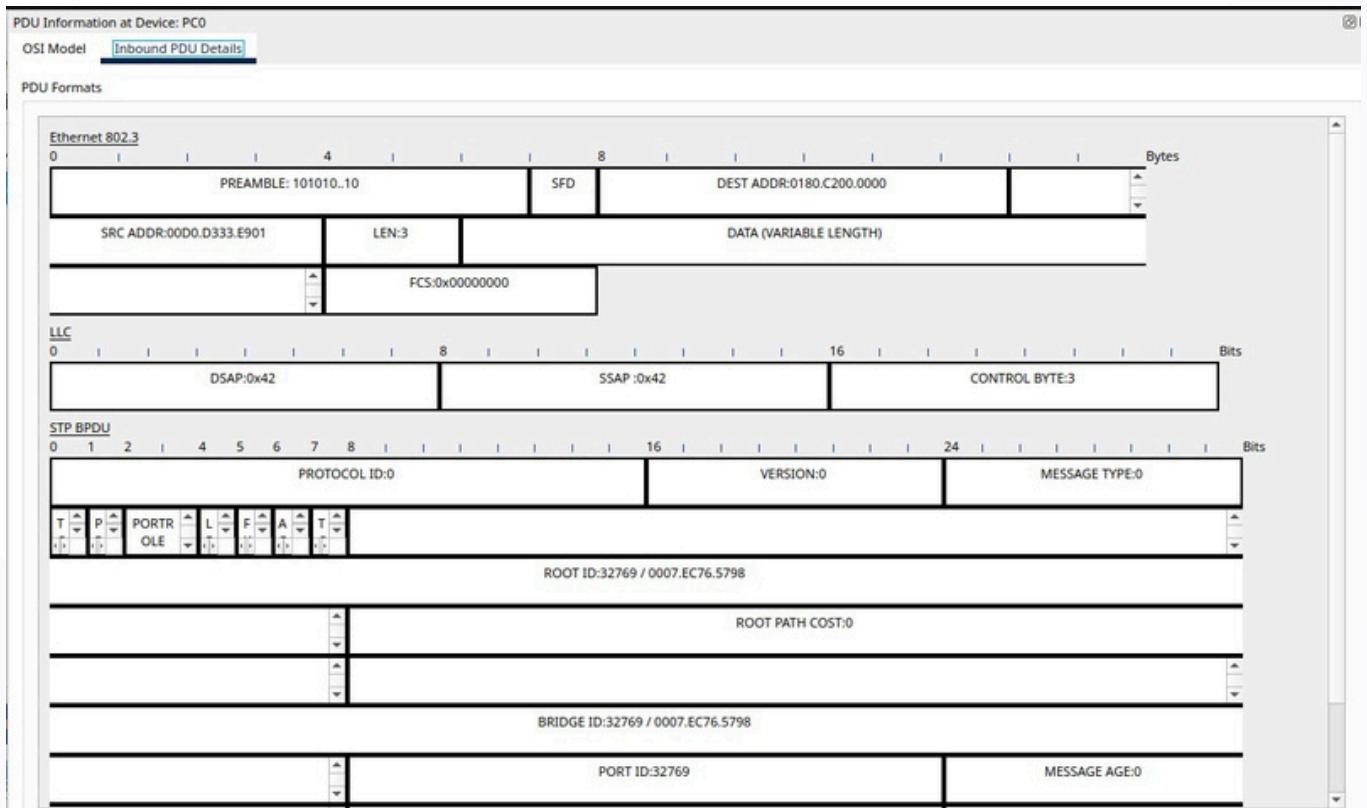


Wireshark:



Pinging PC2 from PC1

Message Frame:



Detailed Ethernet Frame Format

Task 3:

Configure static IP addresses, modify MAC addresses, and verify network connectivity using ping and ifconfig commands.

Explanation:

Static IP:

- Created a clone of a Virtual Machine for this task - **embedUR VM**
- First, the interface is found using the command
\$ ifconfig
- In my case, the interface is **enp0s3**.
- Next, the static IP is set using NAT/Bridged adapter.
 - ipv4 address: **10.0.2.100/24 | 192.168.29.10**
 - default gateway: **10.0.2.2 | 192.168.29.1**
- Once it is set, we can confirm the static IP using the command,
\$ ifconfig enp0s3 | grep inet

MAC address:

- A MAC address is the one which is set by the IEEE and the manufacturer.
- If we change the MAC address, we can ensure privacy - meaning it is hard to track the device.
- First step is to find the interface using the command,
\$ ifconfig
- In my case the interface is **enp0s3**
- To configure MAC address,
 - Make the interface down
\$ sudo ifconfig enp0s3 down
 - Set the address
\$ sudo ifconfig enp0s3 hw ether 00:11:22:33:44:55
 - Bring the interface up
\$ sudo ifconfig enp0s3 up

Outputs:

Static IP:

```
akash@akash:~$ sudo nmcli con mod "netplan-enp0s3" ipv4.addresses 10.0.2.100/24
akash@akash:~$ sudo nmcli con mod "netplan-enp0s3" ipv4.gateway 10.0.2.2
akash@akash:~$ sudo nmcli con mod "netplan-enp0s3" ipv4.dns "8.8.8.8"
akash@akash:~$ sudo nmcli con mod "netplan-enp0s3" ipv4.method manual
akash@akash:~$ sudo nmcli con down "netplan-enp0s3"
Connection 'netplan-enp0s3' successfully deactivated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/2)
akash@akash:~$ sudo nmcli con up "netplan-enp0s3"
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/4)
akash@akash:~$ ifconfig enp0s3 | grep inet
    inet 10.0.2.100 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fd00::a00:27ff:fe65:656e prefixlen 64 scopeid 0x0<global>
        inet6 fd00::7851:6888:3093:53c7 prefixlen 64 scopeid 0x0<global>
        inet6 fe80::a00:27ff:fe65:656e prefixlen 64 scopeid 0x20<link>
akash@akash:~$
```

MAC Address:

```
akash@akash:~$ sudo ip link set enp0s3 down
akash@akash:~$ sudo ip link set enp0s3 address 00:16:17:33:44:55
akash@akash:~$ sudo ip link set enp0s3 up
akash@akash:~$ ifconfig enp0s3 | grep ether
    ether 00:16:17:33:44:55 txqueuelen 1000 (Ethernet)
akash@akash:~$ ifconfig enp0s3 | grep -E "inet |ether"
    inet 10.0.2.100 netmask 255.255.255.0 broadcast 10.0.2.255
    ether 00:16:17:33:44:55 txqueuelen 1000 (Ethernet)
akash@akash:~$ ping -c 4 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=255 time=30.9 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=255 time=48.9 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=255 time=163 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=255 time=25.0 ms

--- 1.1.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3011ms
rtt min/avg/max/mdev = 25.041/67.014/163.193/56.222 ms
akash@akash:~$
```

↑
Checking Connectivity using Ping

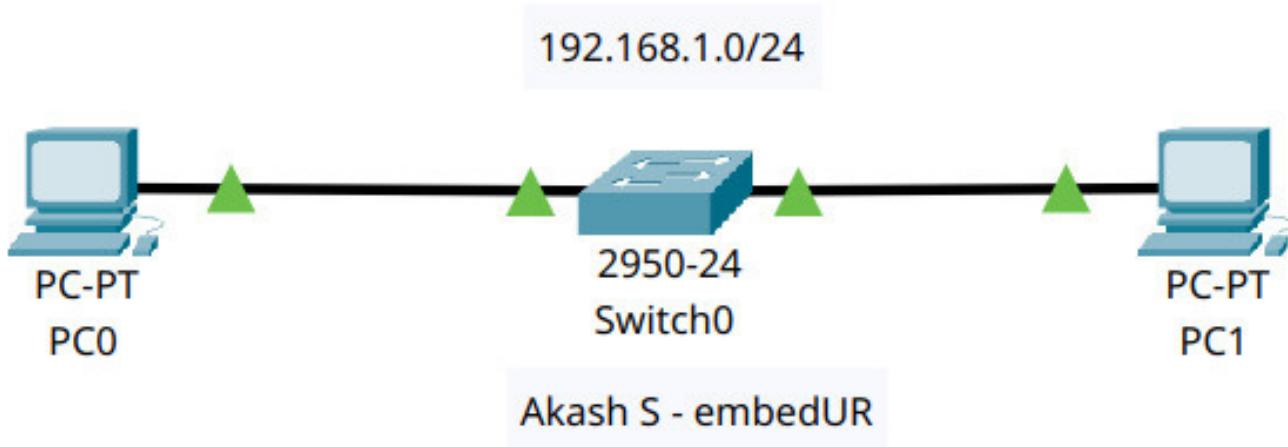
Task 4, 5, 6, 7:

Troubleshoot Ethernet Communication with ping and traceroute -> Using cisco packet tracer. Create a simple LAN setup with two Linux machines connected via a switch. Ping from one machine to the other. If it fails, use ifconfig to ensure the IP addresses are configured correctly. Use traceroute to identify where the packets are being dropped if the ping fails.

Explanation:

LAN setup:

- Created a simple **Local Area Network** setup in Cisco Packet Tracer with Two VPCs and a Switch (2950-24).
- Configured IP:
 - PC1: **192.168.1.10**
 - PC2: **192.168.1.20**
- To check the connectivity between the end devices, I pinged from PC1 to PC2 and from PC2 to PC1.

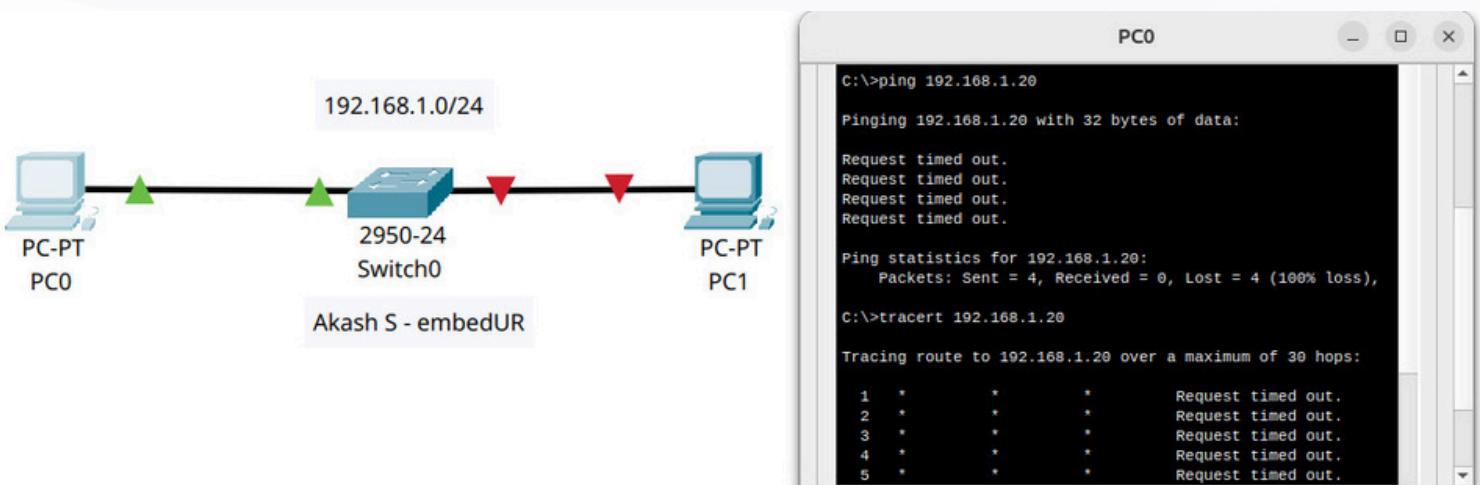


Check the configuration

- Checked whether the IPs are configured correctly using
 - **ifconfig enp0s3 | grep inet**

Troubleshooting:

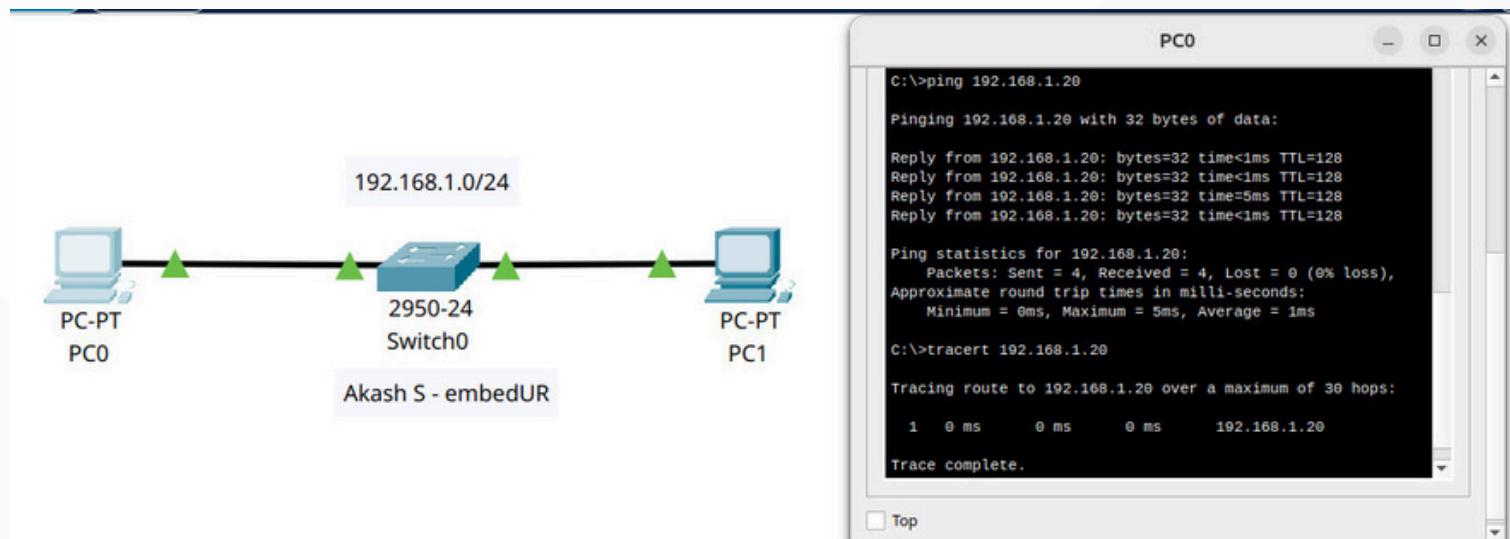
- From switch, inside CLI, disabled the link between Switch and PC2
- So now the ping from PC1 to PC2 fails as the link is down
- Since it fails, the **ping** shows “*Request timed out*”
- Also, we can’t find the routes using **tracert**, it shows “* * *”



*Failure - Request Timed Out! and * * **

End Results:

- Upon enabling the link between the switch and the PC2, it works!
- Pinging from PC1 to PC2 to check the connectivity
- Finding the intermediate routes using **tracert**
- We can know upto 30 intermediate hops



Success - ping and tracert

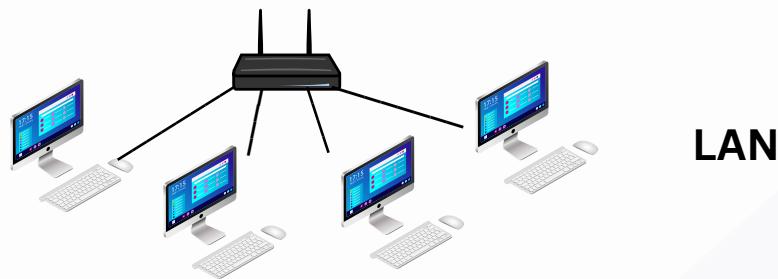
Task 8:

Research the Linux kernel's handling of Ethernet devices and network interfaces. Write a short report on how the Linux kernel supports Ethernet communication (referencing kernel.org documentation).

Report:

Introduction to Ethernet and Linux:

- I found out that Ethernet is a technology that connects devices in a local network (like a LAN at home or school).
- It's super important because it lets computers talk to each other using cables or Wi-Fi.
- The Linux kernel, which is the core of the Linux system, has special features to support Ethernet.
- I learned this is part of its “networking subsystem” that makes communication smooth.



Device Detection & Driver Loading:

- The kernel uses the PCI subsystem to spot Ethernet devices and matches them to drivers using unique PCI IDs.
- It then loads the right driver (e.g., for Intel or Realtek cards) and sets up a network interface like eth0.
- I can configure this interface with tools like ip or ifconfig to assign an IP address.

Data Handling & Management tools:

- The kernel's networking stack processes data through four layers, adding or removing headers for transmission.
- Tools like ifconfig, ip, and ethtool let me manage interfaces, check settings, or troubleshoot issues.
- This setup ensures smooth data flow and easy network control.



ifconfig



ip



ethtool

Kernel handling Network Data:

- I checked out kernel.org for more details:
- The [Networking Documentation](#) explains the whole system.
- The [Ethernet Drivers Section](#) talks about specific drivers.
- These pages helped me confirm what I learned and dig deeper if I want.



* **Networking**

Refer to [Networking subsystem \(netdev\)](#) for a guide on netdev development process specifics.

Contents:

- [AF_XDP](#)
 - Overview
 - Concepts
 - Libbpf
 - XSKMAP / BPF_MAP_TYPE_XSKMAP
 - Configuration Flags and Socket Options
 - Multibuffer Support
 - Sample application
 - FAQ
 - Credits
- [Bare UDP Tunnelling Module Documentation](#)
 - Special Handling
 - Usage
- [batman-adv](#)
 - Configuration
 - Usage
 - Logging Debugging
 - batctl
 - Contact

Networking

NetLabel
InfraBond
ISDN
MBD
Storage interfaces
Other subsystems

Locking

Licensing rules
Writing documentation
Development tools
Testing guide
Hacking guide
Tracing
Fault injection
Livepatching
Rust

Administration
Build system
Reporting issues
Userspace tools
Userspace API

Firmware
Firmware and DeviceTree
CPU architectures

* **Ethernet Device Drivers**

Device drivers for Ethernet and Ethernet-based virtual function devices.

Contents:

- [Linux and the JCom EtherLink III Series Ethercards \(driver v1.18c and higher\)](#)
 - Introduction
 - Special Driver Features
 - Full-duplex mode
 - Available Transceiver Types
 - Revision history (dm file)
- [JCom Vortex device driver](#)
 - Module parameters
- [Linux kernel driver for Elastic Network Adapter \(ENA\) family](#)
 - Overview
 - ENA Source Code Directory Structure
 - Management Interface
 - Data Path Interface
 - Interrupt Modes
 - Interrupt Moderation
 - RX copybreak
 - Statistics
 - MTU
 - Stateless Offloads

My Takeaway:

- I think it's awesome how the Linux kernel handles Ethernet so smartly.
- It finds devices, loads drivers, sets up interfaces, and moves data—all automatically!
- The tools and docs make it easier to understand and control.
- I feel like I get the basics now and could even try setting up a network myself.

Task 9:

Describe how you would configure a basic LAN interface using the ip command in Linux (kernel.org).

Documentation:

Find Interface:

- Open a terminal and type: \$ **ip link show**.
- Find the available interface.
- In my case, the interface is **enp0s3**
- Also we need to identify the status (UP or DOWN)



Bring the interface UP:

- Use the command: \$ **ip link set enp0s3**
- This activates the interface so it's ready to use.
- Verify it's up with: ip link show <interface>.

Bring the interface UP:

- Set a static IP with: (e.g., **ip addr add 192.168.1.10/24 dev enp0s3**)
- The /24 means a subnet mask of 255.255.255.0 (LAN)
- Check it's applied: **ip addr show enp0s3**.

Confirm Routing:

- Confirm routing with: **\$ip route show**

Test the configuration:

- Ping using **ping <ip>**
- If it works
 - Successfully set
- If it fails
 - Troubleshoot with **ip** command

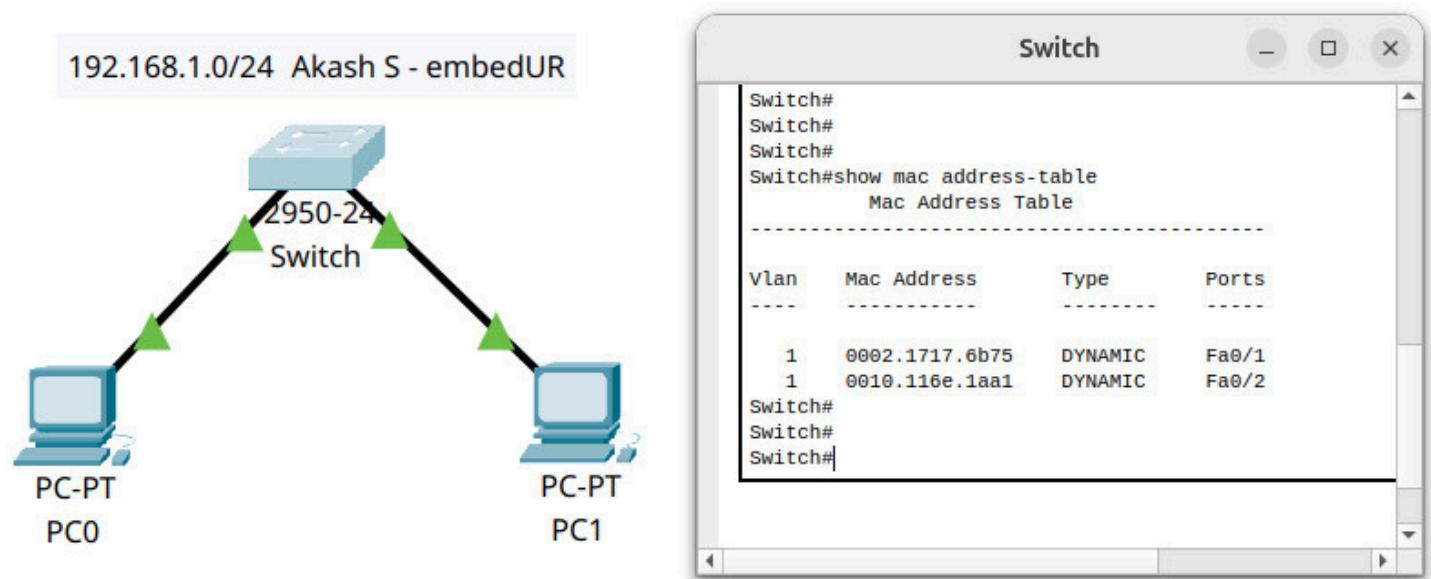
Task 10:

Use Linux to view the MAC address table of a switch (if using a Linux-based network switch). Use the bridge or ip link commands to inspect the MAC table and demonstrate a basic switch's operation.a

Explanation:

Switch MAC table:

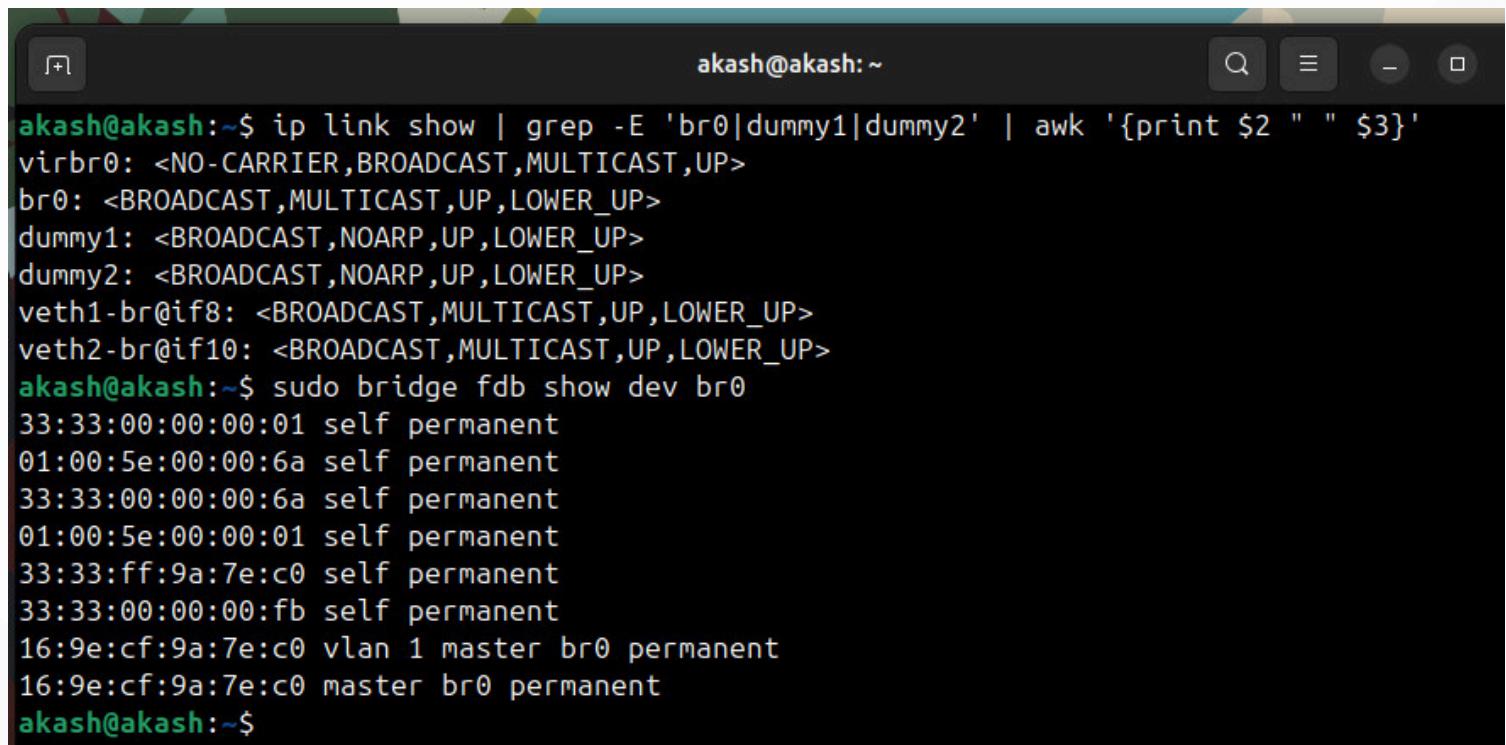
- Created a simple LAN setup in Cisco Packet Tracer with Two VPCs and a Switch (2950-24).
- Configured IP:
 - PC1: **192.168.1.10**
 - PC2: **192.168.1.20**
- To check the connectivity between the end devices, I pinged from PC1 to PC2 and from PC2 to PC1.
- Click on the switch to see the MAC address by executing the command
 - **\$ show mac address-table**



Switch MAC Address table

Bridge:

- Create a bridge named br0 with **sudo ip link add name br0 type bridge** and turn it on with **sudo ip link set br0 up** to act as a switch in Linux.
- Add two dummy ports with
 - dummy1
 - dummy2
- Link the ports to bridge using **master** command.
- Verify the entry in the
 - **\$ ip addr show | grep -E “br0|dummy1|dummy2”**
- The bridge forwards frames using the table
 - if it knows the destination MAC, it sends to the right port
 - if not, it sends to all ports except the sender, working like a switch.



```
akash@akash:~$ ip link show | grep -E 'br0|dummy1|dummy2' | awk '{print $2 " " $3}'  
virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP>  
br0: <BROADCAST,MULTICAST,UP,LOWER_UP>  
dummy1: <BROADCAST,NOARP,UP,LOWER_UP>  
dummy2: <BROADCAST,NOARP,UP,LOWER_UP>  
veth1-br@if8: <BROADCAST,MULTICAST,UP,LOWER_UP>  
veth2-br@if10: <BROADCAST,MULTICAST,UP,LOWER_UP>  
akash@akash:~$ sudo bridge fdb show dev br0  
33:33:00:00:00:01 self permanent  
01:00:5e:00:00:6a self permanent  
33:33:00:00:00:6a self permanent  
01:00:5e:00:00:01 self permanent  
33:33:ff:9a:7e:c0 self permanent  
33:33:00:00:00:fb self permanent  
16:9e:cf:9a:7e:c0 vlan 1 master br0 permanent  
16:9e:cf:9a:7e:c0 master br0 permanent  
akash@akash:~$
```

Bridge as Linux Switch