**Lab 1: Ethernet Network Simulation in Google Colab**

**Title:** Ethernet Network Simulation in Google Colab

**Objective:** To simulate a simple Ethernet network and analyze packet transmission, delay, and throughput using Python.

**Theory:** Ethernet is a wired LAN technology. Using Python’s `networkx` and `simpy`, we can simulate simple network topologies and transmission characteristics.

**Software Required:** - Google Colab - Python libraries: `networkx`, `simpy`, `matplotlib`

**Procedure:**

1. Install required libraries:

```python

!pip install networkx simpy matplotlib

```

2. Create a simple Ethernet-like network topology using `networkx`.

3. Simulate packet transmission using `simpy` and measure delay and throughput.

**Sample Code:**

```python

import networkx as nx

import matplotlib.pyplot as plt

# Create a simple network

G = nx.Graph()

G.add\_edges\_from([(0, 1), (1, 2), (2, 3)])

# Draw the network

nx.draw(G, with\_labels=True)

plt.show()

```

**Observation/Analysis:**

- Network topology created using `networkx`.

- Simulate delay by measuring time between sending and receiving packets.

**Conclusion:** Simple Ethernet-like network simulations can be implemented using `networkx` and `simpy`.

**Lab 2: IP Addressing and Subnetting in Google Colab**

Title: IP Addressing and Subnetting in Google Colab

Objective: To simulate IP addressing and subnetting using Python and understand packet flow between subnets.

Theory: IP addressing assigns a unique identifier to each network device. Subnetting divides a network into smaller subnetworks for better traffic management.

Software Required: - Google Colab - Python libraries: `ipaddress`

Procedure:

1. Install required libraries:

```python

!pip install ipaddress

```

2. Use Python’s `ipaddress` library to create subnets and assign IP addresses.

3. Simulate routing between subnets using simple packet transmission logic.

Sample Code:

```python

import ipaddress

# Define a network and subnet

network = ipaddress.ip\_network('192.168.1.0/24')

subnet = list(network.subnets(prefixlen\_diff=2))

Assign IP addresses to devices

device\_1 = subnet[0][1] # First IP of subnet

device\_2 = subnet[1][1] # First IP of second subnet

print(f"Device 1 IP: {device\_1}, Device 2 IP: {device\_2}")

```

Observation/Analysis:

- IP addresses assigned to devices in different subnets.

- Simple routing between subnets.

Conclusion: IP addressing and subnetting can be simulated using Python’s `ipaddress` module, offering flexibility for creating and managing subnets.

Lab 3: Wi-Fi Network Simulation in Google Colab

Title: Wi-Fi Network Simulation in Google Colab

Objective: To simulate a basic Wi-Fi network using Python and analyze network performance.

Theory:

Wi-Fi networks use radio frequencies to connect devices wirelessly. By simulating Wi-Fi signal strength and data transmission, we can understand how devices communicate wirelessly.

Software Required:

- Google Colab

- Python libraries: `networkx`, `simpy`

Procedure:

1. Create a wireless network with nodes.

2. Simulate data transmission between wireless devices.

3. Measure performance based on signal strength and distance.

Sample Code:

```python

import networkx as nx

import matplotlib.pyplot as plt

# Create a wireless network (similar to Wi-Fi)

G = nx.Graph()

G.add\_nodes\_from([0, 1, 2]) # Three devices

# Add wireless links (with assumed signal strength)

G.add\_edge(0, 1, weight=0.8) # Good signal

G.add\_edge(1, 2, weight=0.4) # Weaker signal

# Plot network

pos = nx.spring\_layout(G)

nx.draw(G, pos, with\_labels=True)

plt.show()

```

Observation/Analysis:

- Visualize network and signal strength between nodes.

- Simulate data transmission based on signal strength.

Conclusion:

Wi-Fi networks can be modeled as graphs in Python, allowing for easy simulation and analysis of wireless communication.

---

Lab 4: Static and Dynamic Routing in Google Colab

Title: Static and Dynamic Routing in Google Colab

Objective: To simulate static and dynamic routing protocols using Python.

Theory:

Routing determines how data is transmitted across networks. Static routing uses fixed routes, while dynamic routing protocols (like RIP and OSPF) adjust routes based on network conditions.

Software Required:

- Google Colab

- Python libraries: `networkx`, `simpy`

Procedure:

1. Create a network topology.

2. Implement static and dynamic routing.

3. Simulate packet transmission and compare routing performance.

Sample Code:

```python

import networkx as nx

# Create a simple network

G = nx.Graph()

G.add\_edges\_from([(0, 1), (1, 2), (2, 3)])

# Static routing: pre-defined routes

static\_route = nx.shortest\_path(G, source=0, target=3)

print("Static Route:", static\_route)

# Dynamic routing can be implemented using dynamic graphs

```

Observation/Analysis:

- Compare static and dynamic routes based on network conditions.

Conclusion:

Static routing uses fixed paths, while dynamic routing can adapt to network changes, improving efficiency.

---

Lab 5: Routing Protocol Performance Comparison in Google Colab

Title: Routing Protocol Performance Comparison in Google Colab

Objective: To compare the performance of different routing protocols in a simulated network.

Theory:

Ad-hoc routing protocols like AODV and DSR are used in mobile networks where nodes move frequently. Performance comparison includes metrics like packet delivery ratio and delay.

Software Required:

- Google Colab

- Python libraries: `networkx`

Procedure:

1. Create a mobile ad-hoc network topology.

2. Simulate routing using both AODV and DSR.

3. Compare packet delivery ratio, delay, and routing overhead.

Sample Code:

```python

import networkx as nx

# Create an ad-hoc network

G = nx.Graph()

G.add\_edges\_from([(0, 1), (1, 2), (2, 3)])

# Simulate AODV routing (shortest path)

aodv\_route = nx.shortest\_path(G, source=0, target=3)

print("AODV Route:", aodv\_route)

# Simulate DSR routing by considering route discovery

```

Observation/Analysis:

- Compare the two protocols in terms of routing efficiency.

Conclusion:

Dynamic routing protocols adapt to changing topologies and offer better performance in mobile networks.

---

Lab 6: Network Traffic Analysis with Python in Google Colab

Title: Network Traffic Analysis with Python in Google Colab

Objective: To analyze network traffic using Python and extract key parameters from packet capture files.

Theory:

Network traffic analysis helps in understanding communication patterns. Tools like `pyshark` and `scapy` can be used for packet-level analysis.

Software Required:

- Google Colab

- Python libraries: `scapy`, `pyshark`

Procedure:

1. Install required libraries:

```python

!pip install pyshark scapy

```

2. Upload a packet capture file (`.pcap`) and analyze its contents.

3. Extract and visualize key metrics like IP addresses, packet lengths, and protocols.

Sample Code:

```python

import pyshark

# Load the pcap file

capture = pyshark.FileCapture('example.pcap')

# Analyze packets

for packet in capture:

print(packet)

```

Observation/Analysis:

- Analyze packet sizes, protocols, and source/destination IPs.

Conclusion:

Packet analysis using Python tools provides insights into network traffic and helps diagnose performance issues.

Lab 7: IoT Network Simulation in Google Colab

Title: IoT Network Simulation in Google Colab

Objective: To simulate an IoT network and analyze sensor-to-gateway communication.

Theory: IoT networks connect sensors and actuators to a gateway. Communication between devices should be efficient and low-latency.

Software Required: - Google Colab - Python libraries: `networkx`, `simpy`

Procedure:

1. Create a network topology with IoT sensors and a central gateway.

2. Simulate data transmission between sensors and gateway.

3. Measure latency and packet delivery.

Sample Code:

```python

import networkx as nx

# Create an IoT network

G = nx.Graph()

G.add\_edges\_from([(0, 1), (1, 2)]) # Sensor to gateway

# Visualize network

nx.draw(G, with\_labels=True)

```

Observation/Analysis: - Measure packet delivery and communication delays.

Conclusion: IoT networks can be simulated in Python, helping analyze sensor data transmission and gateway efficiency.

Lab 8: SDN Simulation using Mininet in Google Colab

Title: SDN Simulation using Mininet in Google Colab

Objective: To simulate an SDN network using Mininet and control traffic using an SDN controller.

Theory: Software-Defined Networking (SDN) separates the control plane from the data plane. Mininet allows us to simulate SDN environments with programmable network behavior.

Software Required: - Google Colab - Mininet

Procedure:

1. Install Mininet on Colab.

2. Create a simple topology and connect it to an SDN controller (POX or Ryu).

3. Control traffic flow using the controller.

Sample Code:

```bash

!apt-get install -y mininet

!mn --topo=single,3 --mac --controller=remote --switch=ovsk

```

Observation/Analysis: - Analyze the traffic control managed by the SDN controller.

Conclusion: SDN enables flexible and programmable network control, as demonstrated through Mininet simulations.