CN LAB

Practical 3: Using Cisco Packet Tracer for Network Analysis

- **Aim**: To study the Cisco Packet Tracer tool, install it, and understand its user interface for designing simple networks.
- **Overview**: Cisco Packet Tracer is a network simulation tool that allows users to create network topologies, configure devices, and simulate network communication. This practical involves setting up the tool, exploring its features, and using it to design basic network structures. Users will learn how to interact with different network devices like hubs and switches and observe packet flow.
- **Procedure**:
    1. Install Cisco Packet Tracer.
    2. Open the tool and familiarize yourself with the user interface.
    3. Drag and drop devices such as PCs and a hub into the workspace.
    4. Connect devices using appropriate cables.
    5. Assign IP addresses to devices.
    6. Use the simulation mode to observe packet behavior.
    7. Document the observed communication behavior.

Practical 4: Setting up a LAN with a Switch

- **Aim**: To configure a LAN using switches and Ethernet cables to enable communication among devices.
- **Overview**: A Local Area Network (LAN) connects devices in a small area, like a lab or office, allowing data sharing and communication. This exercise demonstrates setting up a LAN using a switch, where devices are directly connected to the switch and assigned unique IP addresses, enabling local communication and resource sharing.
- **Procedure**:
    1. Choose four PCs and a switch.
    2. Connect PCs to the switch using Ethernet cables.
    3. Assign IP addresses to each PC (e.g., 192.168.0.1, 192.168.0.2).
    4. Test connectivity between PCs using the ping command.
    5. Share and access files between PCs.
    6. Observe network behavior and log results.

Practical 5: Packet Capture with Wireshark

- **Aim**: To capture and analyze packets using the Wireshark tool.
- **Overview**: Wireshark is a packet analyzer used to capture and examine packets on a network. This practical demonstrates how to use Wireshark to capture live network traffic, apply filters to display specific protocols, and understand packet details, providing insight into data encapsulation and protocols at various OSI layers.
- **Procedure**:
    1. Open Wireshark and select the network interface.
    2. Start packet capture.
    3. Open a web browser and visit a website to generate traffic.
    4. Stop the capture after some packets are recorded.
    5. Filter the packets to view HTTP or DNS packets.
    6. Analyze packet details and note the source, destination, and protocol.
    7. Save the capture for documentation.

Practical 8(a): VLAN Configuration in Cisco Packet Tracer

- **Aim**: To configure VLANs (Virtual LANs) using Cisco Packet Tracer for network segmentation.
- **Overview**: VLANs segment a single network into smaller virtual networks to reduce congestion and increase security. This exercise involves creating VLANs, assigning devices to specific VLANs, and ensuring isolated communication within VLANs.
- **Procedure**:
    1. Create a topology with two switches and four PCs.
    2. Assign IP addresses to PCs.
    3. Configure VLANs on each switch (e.g., VLAN 10 and VLAN 20).
    4. Assign ports on the switch to different VLANs.
    5. Enable trunking between switches.
    6. Test communication within VLANs by pinging devices in the same VLAN.
    7. Document observations.

Practical 8(b): Configuring a Wireless LAN

- **Aim**: To configure a Wireless LAN in Cisco Packet Tracer.
- **Overview**: Wireless LANs allow devices to connect to a network without cables, using wireless signals instead. This practical involves setting up a wireless router, configuring wireless security, and enabling PCs to connect wirelessly.
- **Procedure**:
    1. Place a wireless router and two laptops in the workspace.
    2. Configure the router's SSID and wireless security settings.
    3. Assign IPs to devices or enable DHCP.
    4. Connect laptops to the router using the SSID and password.
    5. Test connectivity between devices.
    6. Observe and document network behavior.

Practical 9: Subnetting in Cisco Packet Tracer

- **Aim**: To implement IP subnetting and connect subnets using routers.
- **Overview**: Subnetting divides a large network into smaller, manageable subnetworks. This exercise involves creating subnets, assigning IPs, and configuring routers to connect these subnets, allowing devices on different networks to communicate.
- **Procedure**:
    1. Create a network topology with two routers and multiple PCs.
    2. Subnet a class C address (e.g., 192.168.1.0/24) into smaller networks.
    3. Assign IPs within each subnet to devices.
    4. Configure routing on routers to enable inter-subnet communication.
    5. Test connectivity between subnets using ping.
    6. Observe packet flow and document results.

Practical 10(a): Internetworking with Routers

- **Aim**: To configure basic routing between networks using a router.
- **Overview**: This practical demonstrates the basics of routing in Packet Tracer, showing how routers connect multiple networks, direct traffic, and allow communication across different IP subnets.

- **Procedure**:
    1. Add two PCs and a router to the workspace.
    2. Assign IPs to the PCs and configure the router with corresponding IPs.
    3. Connect PCs to the router.
    4. Set up routing entries to enable connectivity.
    5. Test communication by pinging between PCs.
    6. Document observations on routing behavior.

Practical 10(b): Configuring Wireless Router with DHCP

- **Aim**: To configure a wireless router with DHCP and test connectivity.
- **Overview**: DHCP automatically assigns IP addresses to devices, simplifying network management. This practical involves configuring a wireless router to act as a DHCP server, providing IPs to connected devices.
- **Procedure**:
    1. Add a wireless router and two PCs.
    2. Configure the router with DHCP enabled.
    3. Connect PCs to the router wirelessly.
    4. Verify that IPs are assigned by DHCP.
    5. Test connectivity between PCs.
    6. Document the DHCP process and connectivity results.

Practical 11(a): Static Routing Protocol Configuration

- **Aim**: To configure static routing between networks using Cisco Packet Tracer.
- **Overview**: Static routing involves manually setting up routes on routers to enable communication between different networks. This exercise demonstrates the setup and configuration of static routes.
- **Procedure**:
    1. Create a network with two routers and four PCs.
    2. Assign IP addresses to routers and PCs.
    3. Add static routes on each router.

4. Test connectivity between PCs on different networks.
5. Verify routing tables.
6. Document results and note how static routing operates.

Practical 11(b): RIP Protocol Configuration

- **Aim**: To configure the RIP protocol for dynamic routing.
- **Overview**: RIP is a dynamic routing protocol that allows routers to update routing tables automatically. This exercise shows how to set up RIP in Packet Tracer, enabling routers to learn routes dynamically.
- **Procedure**:
    1. Set up two routers and connect PCs.
    2. Assign IPs to all devices.
    3. Enable RIP on routers with relevant network addresses.
    4. Test connectivity across networks.
    5. Verify dynamic route updates in routing tables.
    6. Document observations on RIP's functionality.

EX_12_A

Aim:

To implement an echo client-server program using TCP/UDP sockets, which allows the server to echo back any messages it receives from the client.

Algorithm:

1. **Server Side**:
    - Create a socket with TCP/UDP protocol.
    - Bind the socket to a specific IP address and port number.
    - Listen for incoming connections.
    - Accept a connection from a client.
    - Receive data from the client.
    - Send back the same data to the client (echo).

- Close the connection.

2. **Client Side**:
    - Create a socket with TCP/UDP protocol.
    - Connect to the server's IP and port number.
    - Send a message to the server.
    - Receive the echoed message from the server.
    - Display the message.
    - Close the connection.

Server Code (TCP):

```
import socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 12345))
server_socket.listen(1)
print("Server is ready and listening...")
conn, addr = server_socket.accept()
print(f"Connected to {addr}")
data = conn.recv(1024).decode()
print(f"Received: {data}")
conn.send(data.encode())
conn.close()
server_socket.close()
```

Client Code (TCP):

```
import socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("localhost", 12345))
message = "Hello, Server!"
client_socket.send(message.encode())
echo = client_socket.recv(1024).decode()
print(f"Echo from server: {echo}")
```

client_socket.close()

Sample Output:

- **Server Output**:

  Server is ready and listening...

  Connected to ('127.0.0.1', 54321)

  Received: Hello, Server!

- **Client Output**:

  Echo from server: Hello, Server!

EX_12_B

Aim:

To develop a chat program that enables two-way communication between client and server using TCP/UDP sockets.

Algorithm:

1. **Server Side**:
   - Initialize a socket.
   - Bind the socket to an IP address and port.
   - Start listening for client connections.
   - Accept a connection.
   - Enter a loop to receive and send messages alternately with the client.
   - Close the connection.
2. **Client Side**:
   - Initialize a socket.
   - Connect to the server's IP and port.
   - Enter a loop to send and receive messages alternately with the server.
   - Close the connection.

Server Code:

import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```
server_socket.bind(("localhost", 12345))
server_socket.listen(1)
print("Chat server started...")
conn, addr = server_socket.accept()
print(f"Connected to {addr}")
while True:
    message = conn.recv(1024).decode()
    if message.lower() == 'bye':
        print("Connection closed.")
        break
    print(f"Client: {message}")
    conn.send(input("Server: ").encode())
conn.close()
server_socket.close()
```

Client Code:

```
import socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("localhost", 12345))
while True:
    client_socket.send(input("Client: ").encode())
    message = client_socket.recv(1024).decode()
    if message.lower() == 'bye':
        print("Connection closed.")
        break
    print(f"Server: {message}")
client_socket.close()
```

Sample Output:

- **Client:**
  Client: Hello, Server!
  Server: Hello, Client!

- **Server:**
  Client: Hello, Server!
  Server: Hello, Client!

EX_13

Aim:

To create a custom ping program that sends ICMP echo requests to a remote host and measures the response time.

Algorithm:

1. Create a socket using the raw socket type.
2. Form an ICMP echo request packet.
3. Send the ICMP packet to the specified IP address.
4. Wait for the echo reply from the target.
5. Calculate the round-trip time.
6. Print the results, such as response time and packet loss.

Program:

```
import os
import socket
import struct
import time
def checksum(data):
    s = sum(data[i] << 8 | data[i+1] for i in range(0, len(data), 2))
    s = (s >> 16) + (s & 0xffff)
    return ~s & 0xffff
icmp = socket.getprotobyname("icmp")
sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
packet = struct.pack("!BBHHH", 8, 0, 0, os.getpid() & 0xFFFF, 1)
sock.sendto(packet, ("8.8.8.8", 1))
start = time.time()
reply, _ = sock.recvfrom(1024)
print("Ping time:", time.time() - start, "seconds")
```

Sample Output: Ping time: 0.0145 seconds

EX_14

Aim:

To write a program that uses raw sockets for packet sniffing, capturing data packets on the network interface.

Algorithm:

1. Create a raw socket for packet capture.
2. Bind the socket to the network interface.
3. Enter a loop to receive packets.
4. Display packet contents, such as source and destination addresses.
5. Print packet data for inspection.

Program:

```
import socket
sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
sniffer.bind(("localhost", 0))
sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
try:
   while True:
      packet = sniffer.recvfrom(65565)[0]
      print(packet)
finally:
   sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

Sample Output:

Packet captured:

```
b'\x45\x00\x00\x54\xa6\xf2\x40\x00\x40\x01\xa6\xac\xc0\xa8\x00\x01\xc0\xa8\x00\x02..
.'
```