SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 6th Sem

COMPUTER NETWORKS (USA23602J)

Lab Manual

Laboratory 1: Basic Network Commands and their functionalities

Title: Basic Network Commands and their functionalities **Aim:** To understand and practice basic network commands and their functionalities in a command-line interface. **Procedure:**

1. Open the command prompt or terminal on your operating system.
2. Execute various network commands such as `ping`, `ipconfig` (Windows) / `ifconfig` (Linux/macOS), `tracert` (Windows) / `traceroute` (Linux/macOS), `netstat`, `nslookup`, and `arp`.
3. Observe the output of each command and understand its purpose, such as checking connectivity, viewing network configuration, tracing routes, and displaying network statistics.
4. Experiment with different options and parameters for each command. **Source Code:**

```
(No source code for this lab, as it involves command-line execution.)
```

Input:

```
Commands like:
ping google.com
ipconfig /all (or ifconfig)
tracert 8.8.8.8 (or traceroute 8.8.8.8)
netstat -a
nslookup example.com
arp -a
```

Expected Output:

```
Output showing network connectivity, IP addresses, routing paths, active
connections, DNS resolution, and ARP cache entries.
```

Laboratory 2: Introduction to CISCO Packet Tracer (CPT)

Title: Introduction to CISCO Packet Tracer (CPT) **Aim:** To get familiar with the Cisco Packet Tracer simulation tool and its interface for designing and simulating network topologies.
Procedure:

1. Launch Cisco Packet Tracer.
2. Explore the user interface, including the device selection area, logical and physical workspaces, and simulation/real-time modes.
3. Drag and drop various network devices (e.g., PCs, switches, routers) onto the workspace.
4. Connect devices using different types of cables.
5. Assign IP addresses to end devices.
6. Test basic connectivity using the `ping` command within the simulated environment.
   Source Code:

```
(No source code for this lab, as it involves using a simulation tool.)
```

Input:

```
Configuration of devices within Cisco Packet Tracer, such as IP addresses for
PCs.
```

Expected Output:

```
A simple network topology displayed in Cisco Packet Tracer with connected
devices and successful ping results between them.
```

Laboratory 3: Build a Peer to Peer N/W using Hub and Switch. Analyse the difference between the working of a Hub and a Switch

Title: Build a Peer to Peer Network using Hub and Switch. Analyze the difference between the working of a Hub and a Switch. **Aim:** To construct peer-to-peer networks using a hub and a switch, and to analyze the differences in their operation, particularly regarding collision domains and broadcast domains. **Procedure:**

1. **Hub Network:**
   - In Cisco Packet Tracer, create a network with multiple PCs connected to a Hub.
   - Assign IP addresses to all PCs in the same subnet.
   - Send packets between PCs and observe the packet flow in simulation mode, noting how the hub broadcasts traffic.
2. **Switch Network:**
   - Create another network with multiple PCs connected to a Switch.
   - Assign IP addresses to all PCs in the same subnet.
   - Send packets between PCs and observe the packet flow in simulation mode, noting how the switch forwards traffic intelligently.
3. Compare the simulation results to understand the differences in collision domains and efficiency. **Source Code:**

```
(No source code for this lab, as it involves using a simulation tool.)
```

Input:

```
Configuration of PCs (IP addresses) and connections to Hub/Switch in Cisco
Packet Tracer.
```

Expected Output:

```
Observation of packet broadcasting by the hub and intelligent forwarding by
the switch, demonstrating the difference in network efficiency and collision
domains.
```

Laboratory 4: Construct Network using bus topology, Star topology

Title: Construct Network using Bus Topology, Star Topology **Aim:** To design and implement network topologies, specifically bus and star, using Cisco Packet Tracer and understand their characteristics. **Procedure:**

1. **Bus Topology:**
   - In Cisco Packet Tracer, place multiple PCs and connect them using a single coaxial cable (represented by a straight-through cable in CPT, but conceptually a shared medium).
   - Assign IP addresses to all PCs.
   - Test connectivity between devices.
2. **Star Topology:**
   - Place a central device (e.g., a Switch or Hub) and connect multiple PCs to it individually.
   - Assign IP addresses to all PCs.
   - Test connectivity between devices.
3. Analyze the advantages and disadvantages of each topology. **Source Code:**

```
(No source code for this lab, as it involves using a simulation tool.)
```

Input:

```
Configuration of PCs (IP addresses) and connections for bus and star
topologies in Cisco Packet Tracer.
```

Expected Output:

```
Visual representation of bus and star topologies in Cisco Packet Tracer with
successful connectivity tests.
```

Laboratory 5: Construct Network using Ring topology, Mesh topology

Title: Construct Network using Ring Topology, Mesh Topology **Aim:** To design and implement network topologies, specifically ring and mesh, using Cisco Packet Tracer and understand their characteristics. **Procedure:**

1. **Ring Topology:**
   - In Cisco Packet Tracer, connect multiple PCs or switches in a closed loop, forming a ring.
   - Assign IP addresses to all devices.
   - Test connectivity and observe packet flow in the ring.
2. **Mesh Topology:**
   - For a full mesh, connect every device directly to every other device. For a partial mesh, connect critical devices directly.
   - Assign IP addresses to all devices.
   - Test connectivity and observe the multiple paths available.
3. Analyze the advantages and disadvantages of each topology, focusing on redundancy and complexity. **Source Code:**

```
(No source code for this lab, as it involves using a simulation tool.)
```

Input:

```
Configuration of devices (IP addresses) and connections for ring and mesh
topologies in Cisco Packet Tracer.
```

Expected Output:

```
Visual representation of ring and mesh topologies in Cisco Packet Tracer with
successful connectivity tests and observation of their unique traffic
patterns.
```

Laboratory 6: Connecting two LANs using router with static Route

Title: Connecting two LANs using Router with Static Route **Aim:** To configure a router to connect two distinct Local Area Networks (LANs) and implement static routing to enable communication between them. **Procedure:**

1. In Cisco Packet Tracer, create two separate LANs, each with a switch and several PCs.
2. Connect each LAN to a different interface of a router.
3. Assign IP addresses to the PCs and the router interfaces, ensuring each LAN is in a different subnet.
4. Configure static routes on the router to specify the path for packets destined for the other LAN.
5. Test connectivity by pinging devices across the two LANs. **Source Code:**

```
(No source code for this lab, as it involves configuring a router in a
simulation tool.)
```

Input:

```
Router configuration commands for interfaces and static routes (e.g., `ip
route <destination_network> <subnet_mask> <next_hop_ip>`), and IP
configurations for PCs.
```

Expected Output:

```
Successful ping replies between PCs located in different LANs, demonstrating
inter-LAN communication via static routing.
```

Laboratory 7: Multi-routing connection with static router

Title: Multi-routing connection with static router **Aim:** To implement a network with multiple routers and configure static routes to enable communication across several interconnected networks. **Procedure:**

1. In Cisco Packet Tracer, design a network topology with three or more interconnected LANs, each connected via a router.
2. Ensure each LAN and inter-router link is in a unique subnet.
3. Configure IP addresses on all PCs and router interfaces.
4. Carefully configure static routes on each router to ensure all networks are reachable from each other. This will involve defining routes for networks that are not directly connected.
5. Test end-to-end connectivity by pinging devices across all interconnected networks.
   Source Code:

```
(No source code for this lab, as it involves configuring routers in a
simulation tool.)
```

Input:

```
Extensive router configuration commands for interfaces and multiple static
routes on each router, along with IP configurations for all PCs.
```

Expected Output:

```
Successful ping replies between any two devices in the entire multi-network
topology, confirming proper static routing configuration.
```

Laboratory 8: Connecting 2 LANs Using Dynamic Routing

Title: Connecting 2 LANs Using Dynamic Routing **Aim:** To configure a router to connect two distinct Local Area Networks (LANs) and implement dynamic routing (e.g., RIP or OSPF) to enable automatic route discovery and communication between them. **Procedure:**

1. In Cisco Packet Tracer, create two separate LANs, each with a switch and several PCs.
2. Connect each LAN to a different interface of a router.
3. Assign IP addresses to the PCs and the router interfaces, ensuring each LAN is in a different subnet.
4. Configure a dynamic routing protocol (e.g., RIPv2 or OSPF) on the router, advertising the connected networks.
5. Test connectivity by pinging devices across the two LANs.
6. Observe the routing table to see the dynamically learned routes. **Source Code:**

```
(No source code for this lab, as it involves configuring a router in a
simulation tool.)
```

Input:

```
Router configuration commands for interfaces and dynamic routing protocol
(e.g., `router rip`, `network <network_address>` or `router ospf
<process_id>`, `network <network_address> <wildcard_mask> area <area_id>`),
and IP configurations for PCs.
```

Expected Output:

```
Successful ping replies between PCs located in different LANs, and the
routing table showing dynamically learned routes.
```

Laboratory 9: Implementing a simple application using TCP

Title: Implementing a simple application using TCP **Aim:** To develop a basic client-server application using TCP sockets to demonstrate reliable, connection-oriented communication.
Procedure:

1. **Server Program:**
   - Write a server program that creates a TCP socket, binds it to a specific IP address and port, and listens for incoming client connections.
   - Upon accepting a connection, the server should receive data from the client, process it (e.g., print it), and optionally send a response back.
2. **Client Program:**
   - Write a client program that creates a TCP socket and connects to the server's IP address and port.
   - Once connected, the client should send data to the server and optionally receive a response.
3. Compile and run the server program first, then the client program.
4. Observe the communication between the client and server. **Source Code:**

```
# Example Python TCP Server
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 65432        # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    print(f"Server listening on {HOST}:{PORT}")
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            print(f"Received from client: {data.decode()}")
            conn.sendall(b"Message received!")

# Example Python TCP Client
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 65432        # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b"Hello, server!")
    data = s.recv(1024)
print(f"Received from server: {data.decode()}")
```

Input:

```
Client sends a message (e.g., "Hello, server!").
```

Expected Output:

```
Server console:
```

```
Server listening on 127.0.0.1:65432
Connected by ('127.0.0.1', <client_port>)
Received from client: Hello, server!

Client console:
Received from server: Message received!
```

Laboratory 10: Implementing a simple application using UDP

Title: Implementing a simple application using UDP **Aim:** To develop a basic client-server application using UDP sockets to demonstrate unreliable, connectionless communication. **Procedure:**

1. **Server Program:**
   - Write a server program that creates a UDP socket, binds it to a specific IP address and port.
   - The server should continuously listen for incoming datagrams, receive data from clients, process it (e.g., print it), and optionally send a response back to the client's address.
2. **Client Program:**
   - Write a client program that creates a UDP socket.
   - The client should send data (datagrams) to the server's IP address and port.
   - Optionally, the client can listen for a response from the server.
3. Compile and run the server program first, then the client program.
4. Observe the communication, noting the connectionless nature. **Source Code:**

```
# Example Python UDP Server
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.bind((HOST, PORT))
    print(f"UDP Server listening on {HOST}:{PORT}")
    while True:
        data, addr = s.recvfrom(1024)
        print(f"Received from {addr}: {data.decode()}")
        s.sendto(b"Message received!", addr)

# Example Python UDP Client
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.sendto(b"Hello, UDP server!", (HOST, PORT))
    data, addr = s.recvfrom(1024)
print(f"Received from server: {data.decode()}")
```

Input:

```
Client sends a message (e.g., "Hello, UDP server!").
```

Expected Output:

```
Server console:
UDP Server listening on 127.0.0.1:65432
Received from ('127.0.0.1', <client_port>): Hello, UDP server!

Client console:
Received from server: Message received!
```

Laboratory 11: Analyzing the Working of RIP

Title: Analyzing the Working of RIP (Routing Information Protocol) **Aim:** To understand and analyze the operation of the Routing Information Protocol (RIP) in a simulated network environment, focusing on route advertisement, convergence, and hop count. **Procedure:**

1. In Cisco Packet Tracer, create a network with multiple routers and LANs, ensuring there are at least two paths between some networks to observe RIP's path selection.
2. Configure RIP (e.g., RIPv2) on all routers, advertising their directly connected networks.
3. Observe the routing tables on each router as RIP updates are exchanged.
4. Introduce a network change (e.g., disconnect a link, add a new network) and observe how RIP converges and updates the routing tables.
5. Analyze the hop count metric and how RIP selects the best path. **Source Code:**

```
(No source code for this lab, as it involves configuring a routing protocol
in a simulation tool.)
```

Input:

```
Router configuration commands for RIP (e.g., `router rip`, `version 2`,
`network <network_address>`), and network topology changes.
```

Expected Output:

```
Routing tables displaying dynamically learned routes via RIP, demonstration
of route convergence after network changes, and understanding of hop count as
a metric.
```

Laboratory 12: ARP simulation in CPT

Title: ARP Simulation in CPT (Cisco Packet Tracer) **Aim:** To simulate and analyze the Address Resolution Protocol (ARP) process in Cisco Packet Tracer, understanding how MAC addresses are discovered from IP addresses. **Procedure:**

1. In Cisco Packet Tracer, create a simple LAN with at least two PCs connected to a switch.
2. Assign IP addresses to both PCs.
3. In simulation mode, send a `ping` request from one PC to the other for the first time.
4. Observe the ARP request and ARP reply packets in the simulation, noting the source and destination MAC and IP addresses.
5. Check the ARP cache on the PCs before and after the ping using the `arp -a` command in the PC's command prompt. **Source Code:**

```
(No source code for this lab, as it involves using a simulation tool.)
```

Input:

```
Ping command from one PC to another in Cisco Packet Tracer.
```

Expected Output:

```
Observation of ARP request (broadcast) and ARP reply (unicast) packets in
simulation mode, and the ARP cache on PCs showing learned MAC addresses.
```

Laboratory 13: Analyze the Working of a DNS

Title: Analyze the Working of a DNS (Domain Name System) **Aim:** To understand and analyze the operation of the Domain Name System (DNS) in resolving domain names to IP addresses. **Procedure:**

1. In Cisco Packet Tracer, create a network with a PC, a switch, and a DNS server. Optionally, include a web server.
2. Configure the DNS server with a domain name and its corresponding IP address (e.g., `www.example.com` mapping to the web server's IP).
3. Configure the PC to use the DNS server for name resolution.
4. From the PC, try to access the web server using its domain name (e.g., in a web browser or by pinging the domain name).
5. In simulation mode, observe the DNS query and DNS reply packets, tracing the name resolution process. **Source Code:**

```
(No source code for this lab, as it involves configuring services in a
simulation tool.)
```

Input:

```
DNS server configuration (domain name to IP mapping), and a request from the
PC to resolve a domain name (e.g., `ping www.example.com`).
```

Expected Output:

```
Successful resolution of the domain name to an IP address on the PC, and
observation of DNS query/reply packets in simulation mode.
```

Laboratory 14: Implementing a simple web server

Title: Implementing a simple web server **Aim:** To develop and configure a basic web server to serve static HTML content over HTTP. **Procedure:**

1. **Server Program (e.g., Python):**
   - Write a simple HTTP server program that listens on a specific port (e.g., 80 or 8000).
   - When a client connects, the server should respond with an HTTP header and the content of an HTML file (e.g., `index.html`).
2. **HTML File:**
   - Create a simple `index.html` file with some basic content (e.g., "Hello, World! This is my web server.").
3. Run the server program.
4. From a web browser on another machine (or the same machine, using `localhost`), navigate to the server's IP address and port (e.g., `http://localhost:8000`).
5. Observe the HTML content displayed in the browser. **Source Code:**

```
# Example Python Simple HTTP Server
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print(f"serving at port {PORT}")
    httpd.serve_forever()

# Create a file named index.html in the same directory as the Python script:
# # <!DOCTYPE html>
# <html>
# <head>
#     <title>My Simple Web Server</title>
# </head>
# <body>
#     <h1>Hello from my Web Server!</h1>
#     <p>This is a static HTML page served by Python.</p>
# </body>
# </html>
```

Input:

```
Accessing the server's URL in a web browser (e.g., `http://localhost:8000`).
```

Expected Output:

```
The content of `index.html` displayed in the browser.
```

Laboratory 15: Emulate Working of a complete Network using CPT

Title: Emulate Working of a complete Network using CPT (Cisco Packet Tracer) **Aim:** To design, configure, and simulate a comprehensive network scenario using Cisco Packet Tracer, integrating various devices, routing protocols, and services to emulate a real-world network.
Procedure:

1. **Design:** Plan a complex network topology that includes multiple LANs, WAN links, routers, switches, end devices (PCs, servers), and potentially wireless access points.
2. **IP Addressing:** Implement a structured IP addressing scheme for all networks and devices.
3. **Device Configuration:** Configure all devices (routers, switches, PCs) with appropriate settings.
4. **Routing:** Implement dynamic routing protocols (e.g., OSPF, EIGRP) across the network to ensure full connectivity.
5. **Services:** Configure network services such as DNS, DHCP, and a web server on appropriate server devices.
6. **Testing:** Thoroughly test connectivity between all end devices using ping, tracert, and application-level tests (e.g., accessing the web server).
7. **Analysis:** Use simulation mode to observe packet flow, analyze routing decisions, and troubleshoot any connectivity issues. **Source Code:**

```
(No source code for this lab, as it involves extensive configuration within a
simulation tool.)
```

Input:

```
Detailed network design plan, configuration commands for all devices
(routers, switches, servers, PCs), and various connectivity tests.
```

Expected Output:

```
A fully functional simulated network in Cisco Packet Tracer where all devices
can communicate, services are accessible, and routing operates correctly,
mimicking a real-world network environment.
```