RPS™
learn..evolve

WIPRO NGA Program – Connectivity Datacomm Developer Batch

Capstone Project Presentation – 13th August 2024

Project Title – Implement Device Discovery and Basic Functionality

Presented by – TEAM A

WIPRO NGA Program-Datacom Connectivity Developer Batch

Capstone Project Presentation-13 August 2024

Device Discovery & Basic Functionality

Presented by – TEAM A

1. Arun Kumar (TL)	-Topology/PPT
2. Priyal Soni	-code
3. Prachi Patni	-Topology
4. Priyanshu Shit	-code
5. Sravani Muvva	-PPT/code
6. Payal Patil	-Research
7. Nandini Deshmukh	-Research
8. Nikita Pawar	-Test/Research

Contents

Introduction

- Device discovery and management use protocols like ICMP(ping) for network diagnostics and SNMP for comprehensive monitoring and configuration.
- ICMP checks device reachability, while SNMP manages and gathers information on network devices, ensuring effective network operation and troubleshooting.

Objectives

- Develop functionality to scan a range of IP addresses and identify network devices.
- Implement handling for different network protocols (e.g., ICMP, SNMP).

Overview

- This project uses ICMP to check if devices are reachable on a network and SNMP to monitor and manage those devices.
- By combining these tools, we aim to improve network visibility, performance, and troubleshooting, ensuring a smooth and reliable network operation.

Understanding network protocols

1. **ICMP (Internet Control Message Protocol)**

   Purpose : Diagnoses network connectivity issues.

   Function: Sends echo requests and receives echo replies to check if devices are reachable.

   Usage : Tools like ping and traceroute use ICMP for network diagnostics and latency measurement.

2. **SNMP (Simple Network Management Protocol)**

   Purpose : Manages and monitors network devices.

   Function: Collects and organizes information from devices and allows configuration changes.

   Usage : Enables network administrators to track performance, configure settings, and manage devices efficiently.

Scanning IP Address

The foundation of device discovery lies in scanning a defined range of IP addresses to identify active devices on the network.

Range Definition - Specify the IP address range to scan, taking into account network segmentation and device density.

Packet Transmission - Send ICMP echo requests to each IP address in the range, attempting to elicit a response.

Response Analysis - Analyze received responses to identify active devices and record their IP addresses, MAC addresses, and other relevant details.

Task 1 - IP Range Scanning

Objective	: Scan a specified range of IP addresses.
Methodology	: Use network scanning techniques to identify active devices within the range.
Tools	: Nmap, custom scripts, or network libraries.

Task 2 - Device Identification

Objective : Identify and classify devices on the network.

Methodology: Analyze responses from scanned IP addresses.

Protocols : ICMP (ping), SNMP (Simple Network Management Protocol).

Task 3 - Handling Network Protocols

ICMP:

- Send echo requests (ping) and wait for replies.
- Determine device availability and latency.

SNMP:

- Query devices for specific information.
- Collect data like device type, status, and more.

Handling Network Protocols

1. **ICMP**:

   Internet Control Message Protocol (ICMP) is commonly used for basic device detection, utilizing echo requests and replies.

2. **SNMP**:

   Simple Network Management Protocol (SNMP) provides detailed information about devices, including hardware configuration, software version, and performance metrics.

3. **DHCP**:

   Dynamic Host Configuration Protocol (DHCP) can be used to identify devices that obtain IP addresses dynamically from a DHCP server.

4. **Other Protocols**:

   Depending on specific requirements, additional protocols suchas Telnet, SSH, and HTTP can be utilized for device communication and data collection.

Project Outline

- **IP Range Scanning:**
  - Implement a function to scan a given range of IP addresses.
  - Use multithreading to speed up the scanning process.
- **Network Protocol Handling:**
  - **ICMP:** Implement ping functionality to check if a device is reachable.
  - **SNMP:** Implement Simple Network Management Protocol to gather information from devices.
- **Edge Case Handling:**
  - Handle invalid IP ranges gracefully.
  - Provide feedback if no devices are found within the given range.

Code Explanation

1.Ping Function

```
bool ping(const std::string& ipAddress)
 {
   std::string command = "ping -n 1 " +
 ipAddress + " > nul 2>&1";  // Windows ping
 command
   return system(command.c_str()) == 0;
}
```

Purpose:

To check if an IP address is reachable by sending a ping request.

Implementation:

Uses the Windows ping command. The -n 1 option sends one packet, and > nul 2>&1 suppresses output.

Return Value:

Returns true if the ping is successful, otherwise false.

2. SNMP Query Function

```
std::string snmpQuery(const std::string& ipAddress,
const std::string& oid) {
    std::string command = "snmpget -v 2c -c public " +
ipAddress + " " + oid + " 2>/dev/null";
    char buffer[128];
    std::string result = "";
    FILE* pipe = popen(command.c_str(), "r");
    if (!pipe) return "ERROR";
    while (fgets(buffer, sizeof buffer, pipe) != NULL) {
        result += buffer;
    }
    pclose(pipe);
    return result;
}
```

Purpose:

To perform an SNMP query to retrieve information from a network device.

Implementation:

Uses the snmpget command-line tool. The output is captured from a pipe.

Return Value:

Returns the result of the SNMP query as a string.

3. IP Address Increment Function

```
std::string incrementIpAddress(const std::string& ipAddress)
{
    struct sockaddr_in sa;
    inet_pton(AF_INET, ipAddress.c_str(), &(sa.sin_addr));
    sa.sin_addr.s_addr = htonl(ntohl(sa.sin_addr.s_addr) + 1);
    char str[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);
    return std::string(str);
}
```

Purpose:

To increment an IP address by one.

Implementation:

Converts the IP address to a numerical format, increments it, and converts it back to a string.

Return Value:

The next IP address in sequence.

4. Device Discovery Function

```
std::vector<std::string> discoverDevices(const std::string& startIp, const
std::string& endIp) {
    std::vector<std::string> discoveredDevices;
    std::string currentIp = startIp;
    struct sockaddr_in sa;
    if (inet_pton(AF_INET, startIp.c_str(), &(sa.sin_addr)) != 1) {
        std::cerr << "Invalid start IP address: " << startIp << std::endl;
        return discoveredDevices;}
    if (inet_pton(AF_INET, endIp.c_str(), &(sa.sin_addr)) != 1) {
        std::cerr << "Invalid end IP address: " << endIp << std::endl;
        return discoveredDevices;}
    while (currentIp <= endIp) {
        std::cout << "Pinging: " << currentIp << std::endl;
        if (ping(currentIp)) {
            std::cout << "Ping successful: " << currentIp << std::endl;
            std::string sysDescr = snmpQuery(currentIp, "1.3.6.1.2.1.1.1.0");
            std::cout << "SNMP Query Result: " << sysDescr << std::endl;
            discoveredDevices.push_back(currentIp + " - " + sysDescr);
        } else {
            std::cout << "Ping failed: " << currentIp << std::endl}
        currentIp = incrementIpAddress(currentIp);
    }return discoveredDevices;
}
```

Purpose:

To scan a range of IP addresses, ping each address, and perform SNMP queries to discover devices.

Implementation:

Validates IP addresses.
Iterates from startIp to endIp, pinging each address.If the ping is successful, performs an SNMP query.Collects and returns the list of discovered devices.

5. Test Scenarios

Test Case 1 (Valid IP Range):

```
void testValidIPRange() {
    std::cout << "\nTest Case 1: Valid IP Range\n";
    std::string startIp = "172.20.0.10";
    std::string endIp = "172.20.0.20";

    std::vector<std::string> devices = discoverDevices(startIp,
endIp);
    std::cout << "Discovered Devices:\n";
    for (const auto& device : devices) {
        std::cout << device << std::endl;
    }
    if (devices.empty()) {
        std::cout << "No devices found in the range.\n";
    }
}
```

Purpose:

These functions test the discoverDevices function under different conditions.

Explanation:

Tests discovery on a valid range, expecting to find devices.

Test Case 2 (Invalid IP Range)

```
void testInvalidIPRange() {
    std::cout << "\nTest Case 2: Invalid IP Range\n";
    std::string startIp = "192.168.1.256";
    std::string endIp = "192.168.1.260";

    std::vector<std::string> devices = discoverDevices(startIp, endIp);
    if (devices.empty()) {
        std::cout << "No devices found or invalid IP range provided.\n";
    }
}
```

Purpose:

These functions test the discoverDevices function under different conditions.

Explanation:

Test Case 2 (Invalid IP Range): Provides invalid IPs to ensure the code handles input validation correctly.

Test Case 3 (No Devices Found):

```
void testNoDevicesFound() {
    std::cout << "\nTest Case 3: No Devices Found\n";
    std::string startIp = "192.168.1.289";
    std::string endIp = "192.168.1.299";
    std::vector<std::string> devices = discoverDevices(startIp, endIp);
    std::cout << "Discovered Devices:\n";
    for (const auto& device : devices) {
        std::cout << device << std::endl;
    }
    if (devices.empty()) {
        std::cout << "No devices found in the range.\n";
    }
}
```

Purpose:

These functions test the discoverDevices function under different conditions.

Explanation:

Tests a valid range with no active devices to verify the absence of results.

6. Main Function

```
int main() {
    testValidIPRange();
    testInvalidIPRange();
    testNoDevicesFound();
    return 0;
}
```

Explanation : The main function calls each test scenario sequentially, helping to validate the functionality of the device discovery process under different conditions.

Testing Device Discovery

Thorough testing ensures the robustness and accuracy of the device discovery functionality.

1. **Valid IP Range**

   Test the discovery process with a known range of IP addresses containing active devices. Verify that all devices are detected and their information is accurately captured.

2. **Invalid IP Range**

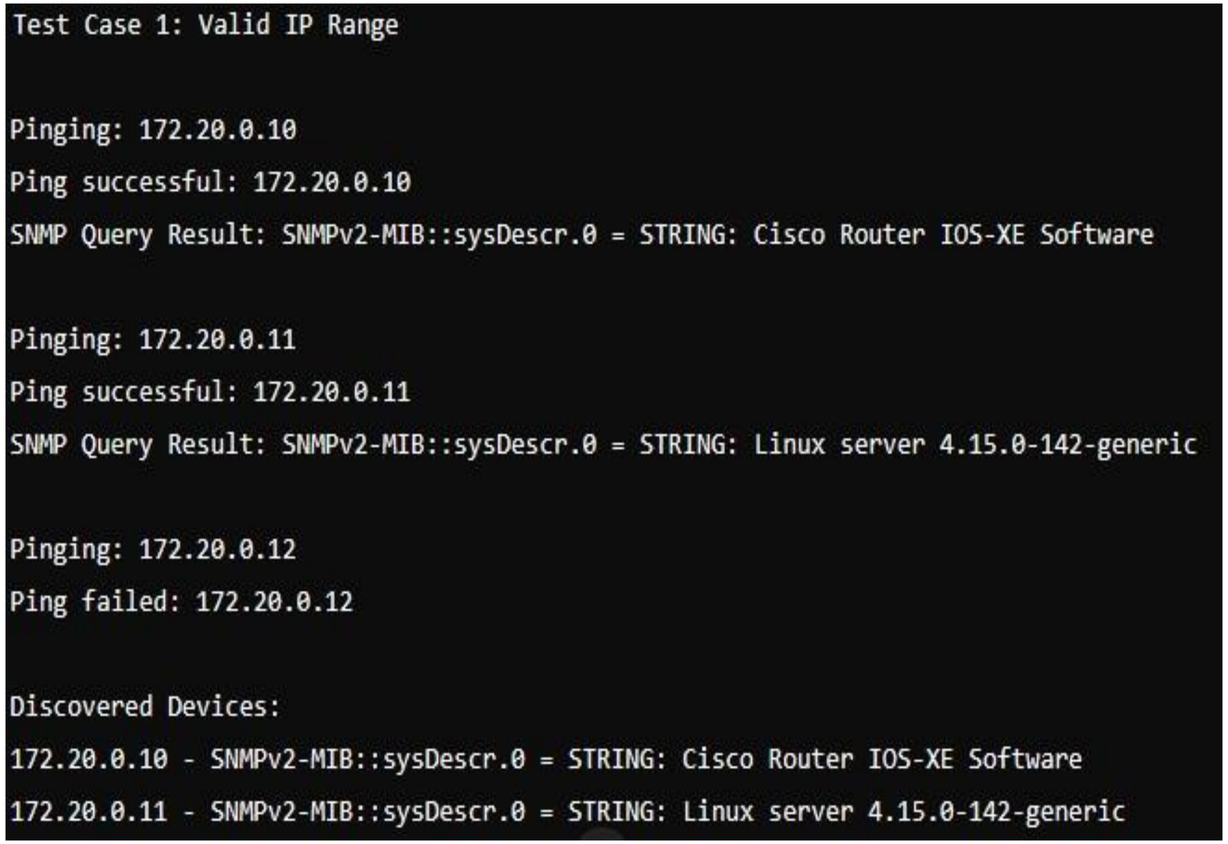
   Test with an IP range that does not contain any active devices. Ensure that the system correctly handles the absence of responses and does not report false positives.

3. **No Devices Found**

   Test a scenario where there are no active devices in the specified IP range. The system should report the absence of devices without encountering errors or unexpected behavior

Sample output

Test Case 1: Valid IP Range

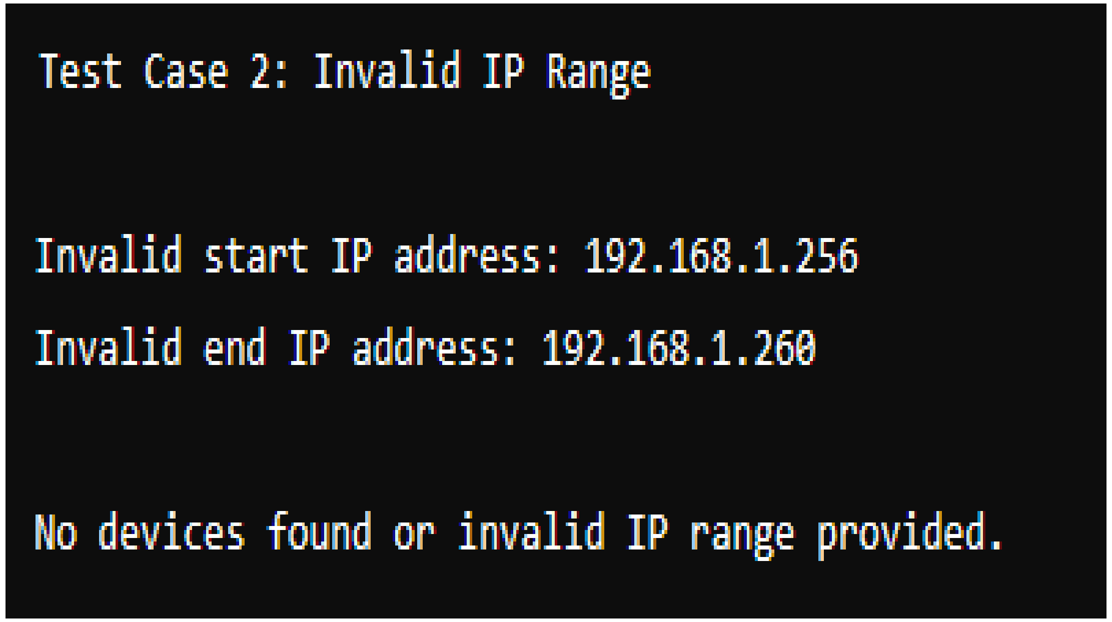


```
Test Case 1: Valid IP Range

Pinging: 172.20.0.10
Ping successful: 172.20.0.10
SNMP Query Result: SNMPv2-MIB::sysDescr.0 = STRING: Cisco Router IOS-XE Software

Pinging: 172.20.0.11
Ping successful: 172.20.0.11
SNMP Query Result: SNMPv2-MIB::sysDescr.0 = STRING: Linux server 4.15.0-142-generic

Pinging: 172.20.0.12
Ping failed: 172.20.0.12

Discovered Devices:
172.20.0.10 - SNMPv2-MIB::sysDescr.0 = STRING: Cisco Router IOS-XE Software
172.20.0.11 - SNMPv2-MIB::sysDescr.0 = STRING: Linux server 4.15.0-142-generic
```

Explanation:

Simulates a scenario where some devices respond to pings and SNMP queries, while others do not. The successful pings return the system descriptions of the devices.

Test Case 2: Invalid IP Range

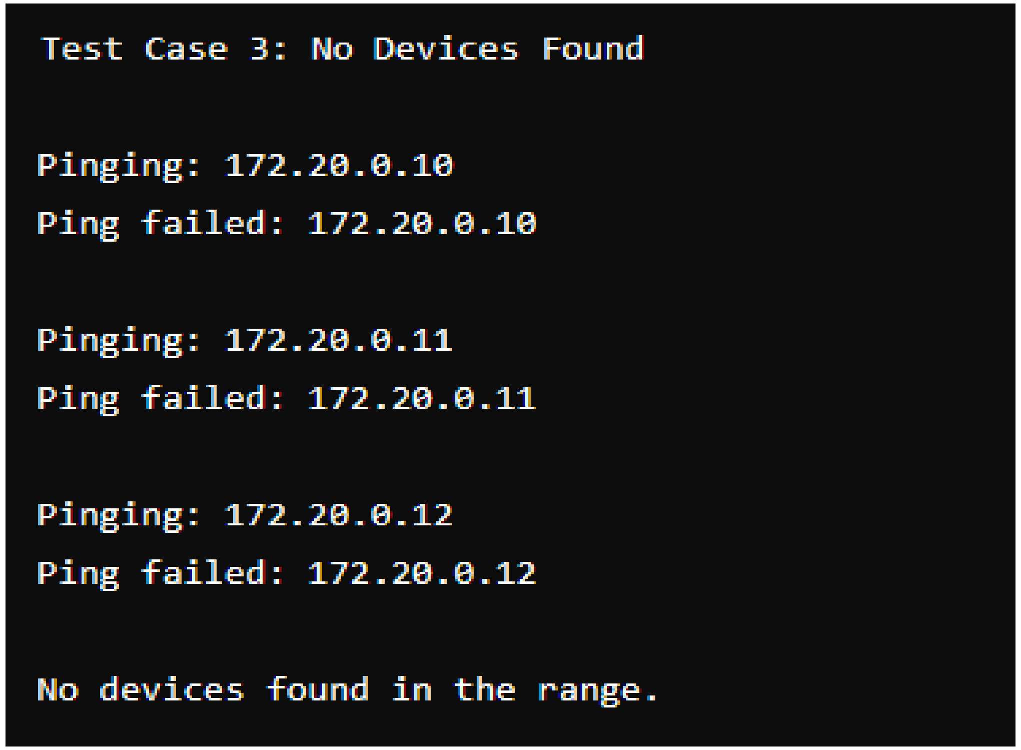


```
Test Case 2: Invalid IP Range


Invalid start IP address: 192.168.1.256

Invalid end IP address: 192.168.1.260


No devices found or invalid IP range provided.
```

Explanation:

Demonstrates handling of an invalid IP range. Since the IP addresses provided are invalid, the code identifies this and does not proceed with scanning.

Test Case 3: No Devices Found

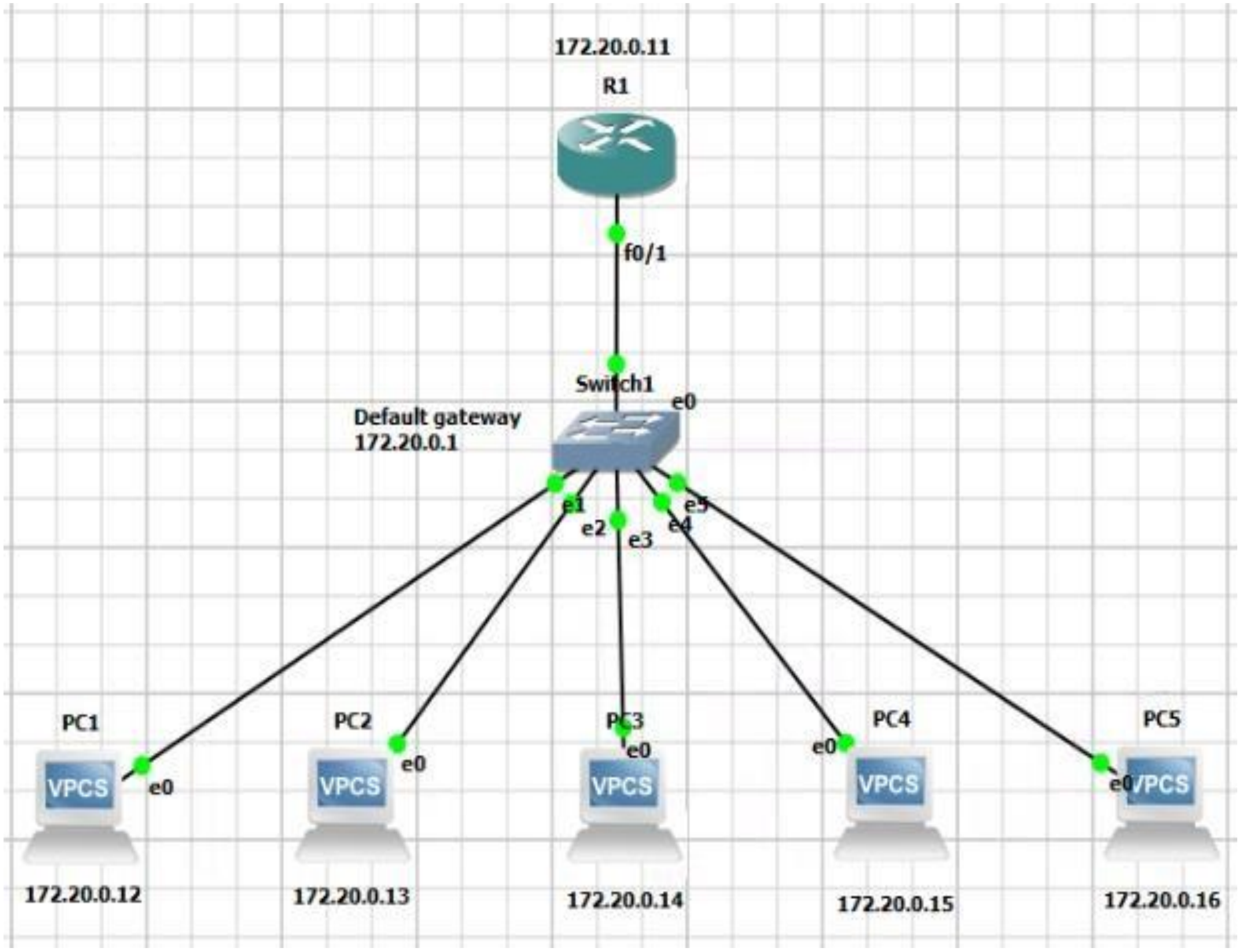


```
Test Case 3: No Devices Found

Pinging: 172.20.0.10
Ping failed: 172.20.0.10

Pinging: 172.20.0.11
Ping failed: 172.20.0.11

Pinging: 172.20.0.12
Ping failed: 172.20.0.12

No devices found in the range.
```

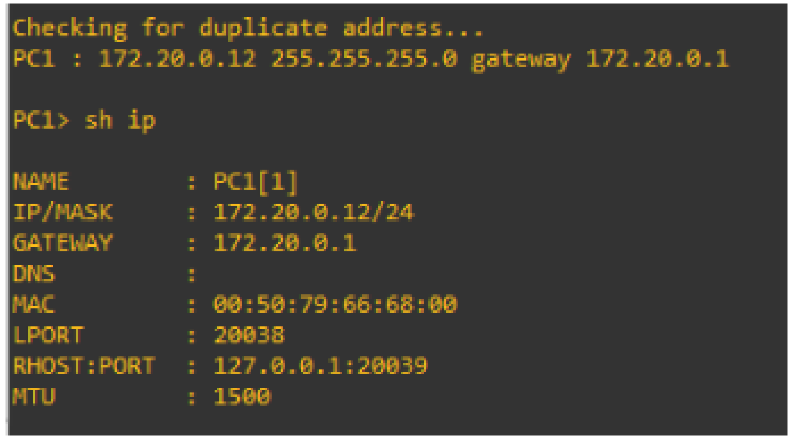
Explanation:

Simulates a situation where no devices respond to pings, resulting in an empty list of discovered devices.

Topology for Device Discovery

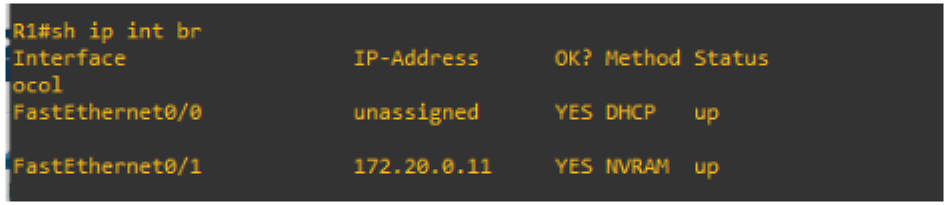


IP configuration for PCs



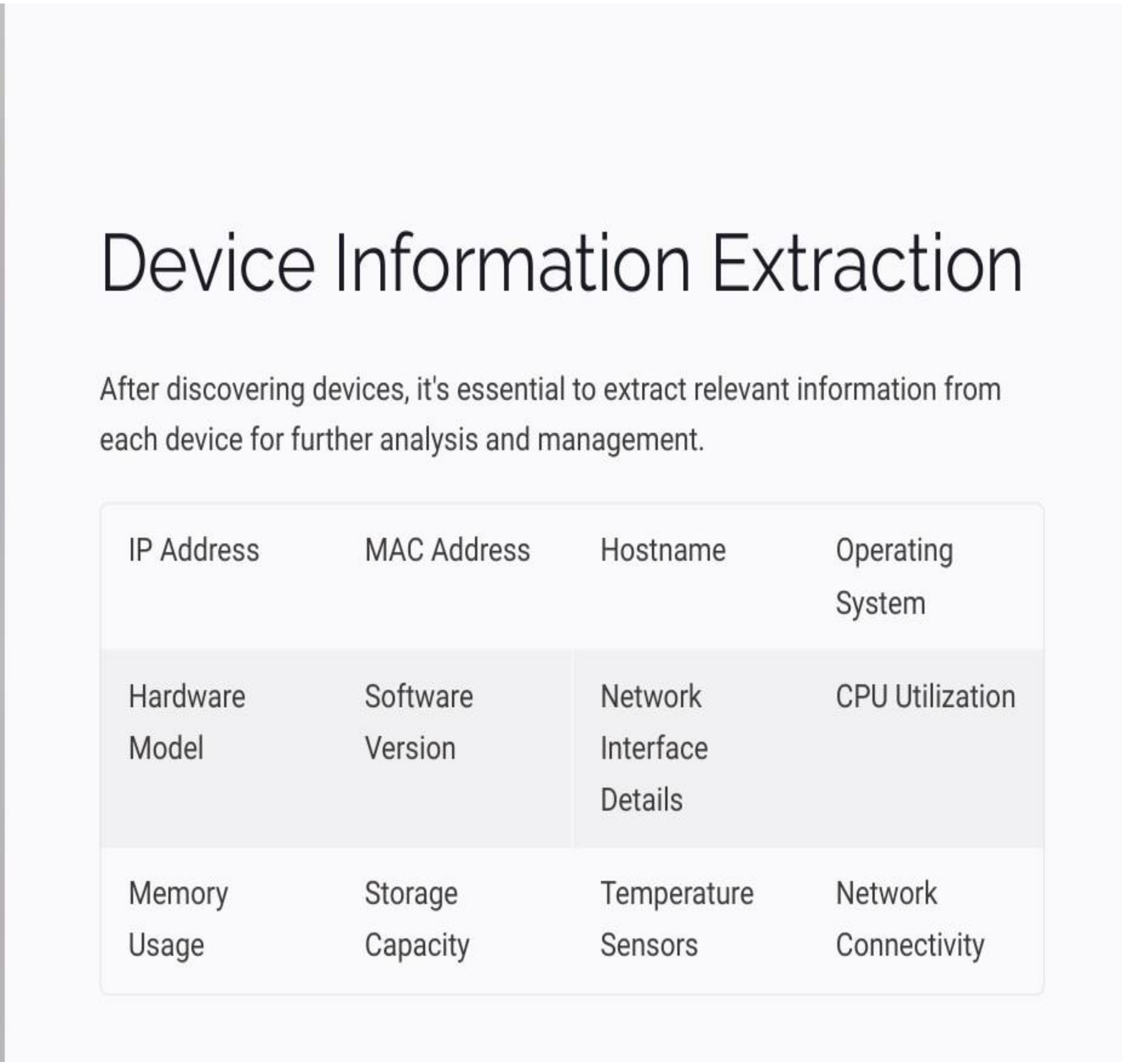
```
Checking for duplicate address...
PC1 : 172.20.0.12 255.255.255.0 gateway 172.20.0.1

PC1> sh ip

NAME        : PC1[1]
IP/MASK     : 172.20.0.12/24
GATEWAY     : 172.20.0.1
DNS         :
MAC         : 00:50:79:66:68:00
LPORT       : 20038
RHOST:PORT  : 127.0.0.1:20039
MTU         : 1500
```

IP configuration for Router



```
R1#sh ip int br
Interface                  IP-Address      OK? Method Status
ocol
FastEthernet0/0            unassigned      YES DHCP   up

FastEthernet0/1            172.20.0.11     YES NVRAM  up
```

Device Information



Device Information Extraction

After discovering devices, it's essential to extract relevant information from each device for further analysis and management.

IP Address	MAC Address	Hostname	Operating System
Hardware Model	Software Version	Network Interface Details	CPU Utilization
Memory Usage	Storage Capacity	Temperature Sensors	Network Connectivity

Future Enhancements

Network Topology Mapping:

Automatically generate a visual representation of the network topology based on discovered devices and their interconnections.

Real-Time Monitoring:

Implement real-time monitor of discovered devices to track performance metrics, identify anomalies, anfi generate alerts.

Automated Configuration Management:

Automate device configuration tasks, such as firmware updates, security settings, and network configuration changes.

Security Vulnerability Scanning:

Integrate security vulnerability scanning tools to identify potential security threats and proactively mitigate risks

Detailed Testing Steps

Setup Devices and Network:

- Connect devices to the switch/router
- Ensure correct IP assignments

Deploy and Compile Code:

- Transfer and compile the code on a testing device

Run the Code:

- Execute the IP scanning program

Verify and Troubleshoot:

- Check results and compare with actual devices
- Address any issues with connectivity or code

Troubleshooting

Common Issues:

- No response from devices
- Incorrect IP address assignments
- Code not handling edge cases

Troubleshooting Steps:

- Verify device connections and configurations
- Check and debug code for errors
- Use network monitoring tools to diagnose issues

Conclusion

Efficient Network Scanning:

Successfully pings and retrieves SNMP data across specified IP ranges.

Robust Error Handling:

Handles invalid IPs and scenarios with no devices, ensuring stability.

Modular and Extendable:

Functions are reusable and adaptable for further network management tasks.

Cross-Platform Considerations:

Initially designed for Windows, adaptable for other OS with minor adjustments.

Educational Value:

Provides a practical introduction to network programming with ICMP and SNMP in C++.

THANK YOU