# WIPRO NGA Program – Connectivity Datacom Developer Batch

A capstone project on

# C++ MONITORING AND COMPLIANCE CHECKING

**A PROJECT REPORT IS SUBMITTED BY**

ARFATH BASHA(TL)

PRAVEEN

LOGESH KUMAR

KOWSHIK

MURALI PRASANTH

ALPITA MANDAVKAR

ANUJ JAISWAL

**Presentation Date– 13th Aug 2024**

## 1.1 DEVELOP FUNCTIONALITY TO MONITOR DEVICE STATUS

### 1. Understand the 5G Architecture

- **User Equipment (UE):** Devices that connect to the 5G network.
- **Radio Access Network (RAN):** Manages the wireless connection between the UE and the core network.
- **Core Network (CN):** Manages data routing, user sessions, and mobility.

### 2. Define Monitoring Metrics

Decide what metrics you need to monitor for device status. Common metrics include:

- Signal Strength (RSRP, RSRQ)
- Connection Status (Connected, Idle, Disconnected)
- Data Throughput
- Latency
- Error Rates (e.g., dropped packets, connection failures)
- Battery Status (for IoT devices)

### 3. Implement Data Collection Mechanisms

- **Network Slicing:** Use network slicing to create isolated logical networks that can monitor specific device types or services.
- **Network Functions:** Utilize network functions such as the Network Function Virtualization (NFV) and Software-Defined Networking (SDN) for data collection and management.
- **Monitoring Interfaces:** Leverage standard interfaces like the Network Management Interface (NMI) for data collection.

### 4. Real-Time Data Collection and Processing

- **Telemetry:** Implement telemetry systems to gather data from network elements.
- **Network Management System (NMS):** Use NMS for central management and real-time monitoring.

- **Data Aggregation:** Collect data from various network nodes and aggregate it for analysis.

**5. Analyze Device Status**

- **Event Correlation:** Correlate events from different network elements to determine the status of devices.
- **Machine Learning:** Implement machine learning algorithms to predict and identify potential issues based on historical data.

**6. Visualize and Alert**

- **Dashboards:** Create dashboards to visualize real-time data, trends, and alerts.
- **Alerts:** Set up alerting mechanisms for critical issues, such as device disconnections or high error rates.

**7. Implement Feedback Mechanisms**

- **Automated Actions:** Develop automated actions for common issues, such as reconnecting devices or adjusting network parameters.
- **User Notifications:** Notify users or administrators of significant issues that require manual intervention.

**8. Compliance and Security**

- **Data Privacy:** Ensure that monitoring solutions comply with data protection regulations.
- **Security:** Implement security measures to protect monitoring data from unauthorized access.

**9. Testing and Optimization**

- **Testing:** Conduct extensive testing in different network conditions to ensure that the monitoring solution performs well.
- **Optimization:** Continuously optimize the monitoring process based on performance metrics and feedback.

## 1.2 IMPLEMENT COMLIANCE CHECKING TO ENSURE DEVICES MATCH SPECIFIED CONFIGURATIONS

Implementing compliance checking to ensure devices match specified configurations in a 5G network involves several steps. This process ensures that devices adhere to predetermined standards and configurations, which is crucial for network performance, security, and regulatory compliance. Here's a step-by-step guide to achieve this:

### 1. Define Compliance Standards and Configurations

- **Configuration Standards:** Specify the exact configurations that devices should adhere to, including settings for radio frequencies, encryption protocols, software versions, etc.
- **Regulatory Requirements:** Ensure that the configurations comply with relevant regulatory standards and industry best practices.

### 2. Develop a Configuration Management Plan

- **Baseline Configurations:** Create and document baseline configurations for each device type or category.
- **Configuration Change Management:** Implement procedures to handle configuration changes and ensure that updates are compliant with standards.

### 3. Implement Monitoring and Data Collection

- **Configuration Data Collection:** Develop mechanisms to collect configuration data from devices. This can be done using SNMP, APIs, or network management tools.
- **Regular Audits:** Schedule regular audits to collect configuration data and compare it against the baseline.

### 4. Automate Compliance Checking

- **Configuration Management Tools:** Use tools such as Ansible, Puppet, or Chef to automate configuration management and compliance checks.

- **Compliance Scripts:** Develop scripts to automatically verify that device configurations match the specified standards. These can be implemented in languages like Python or Bash.

**5. Perform Compliance Checks**

- **Data Comparison:** Compare the collected configuration data against the defined standards. This can be done using scripts or configuration management tools.
- **Generate Reports:** Produce reports that highlight any discrepancies or non-compliant configurations.

**6. Handle Non-Compliance**

- **Alerting:** Set up alerts for non-compliance issues so that administrators can take corrective actions.
- **Automated Remediation:** Implement automated remediation actions to correct configurations or notify administrators for manual intervention.

**7. Ensure Security and Access Control**

- **Access Management:** Ensure that only authorized personnel can modify configurations or perform compliance checks.
- **Audit Trails:** Maintain logs of configuration changes and compliance checks to ensure accountability and traceability.

**8. Review and Update**

- **Regular Reviews:** Periodically review compliance standards and update them as needed to reflect changes in regulatory requirements or technological advancements.
- **Feedback Loop:** Implement a feedback loop to continuously improve the compliance checking process based on findings and evolving requirements.

## 1.3 DEVICE STATUS MONITORING USING GNS3

NS3 (Graphical Network Simulator-3) is a network simulation tool that allows you to create complex network topologies and simulate real-world networking environments. While GNS3 is typically used for simulating traditional network devices and configurations, it can also be adapted to monitor device status in a 5G network simulation. Here's how you can set up device status monitoring in a 5G environment using GNS3:

### 1. Set Up GNS3 for 5G Network Simulation

1. **Install GNS3:**
    - Download and install GNS3 from the official website.
    - Ensure you have the necessary dependencies and virtual machines (VMs) installed.
2. **Create a 5G Network Topology:**
    - Use GNS3 to design a network topology that includes 5G network elements such as:
        - **User Equipment (UE):** Simulated devices (e.g., VMs or routers configured to act as 5G devices).
        - **gNodeB (gNB):** 5G base stations.
        - **5G Core Network (5GC):** Components like the Access and Mobility Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF).
    - Use images of routers, switches, and servers that support 5G functionalities. You may need to use VMs with specific network functions or integrate with other network simulation tools.

### 2. Implement Monitoring Tools

To monitor device status in your 5G simulation, you can integrate various monitoring tools:

1. **Network Monitoring Systems:**
    - Use network monitoring systems like **Nagios**, **Zabbix**, or **PRTG**. You can simulate these systems within GNS3 or integrate them from external sources.
2. **Custom Scripts and Tools:**

    - Develop custom scripts to collect and analyze data from your simulated devices. Use Python or other scripting languages to interact with devices and gather status metrics.

**3. Configure Data Collection**

1. **Set Up SNMP or API Access:**
    - Configure simulated devices to support SNMP (Simple Network Management Protocol) or REST APIs for data collection.
    - Use SNMP agents or APIs to gather metrics such as signal strength, connection status, and data throughput.
2. **Implement Telemetry:**
    - If supported, configure devices to send telemetry data to a central monitoring system.

**4. Monitor Device Status**

1. **Create Monitoring Dashboards:**
    - Use tools like **Grafana** or **Kibana** to create dashboards that visualize the status of simulated devices. Integrate these tools with your data sources (e.g., databases, APIs).
2. **Analyze Metrics:**
    - Collect and analyze metrics such as:
        - **Signal Strength:** Track received signal strength indicators.
        - **Connection Status:** Monitor whether devices are connected or disconnected.
        - **Throughput and Latency:** Measure data rates and latency.

## 1.4 COMPLIANCE CHECK SUCCESS

**Compliance Check Success** refers to the outcome of a process where a system or device is evaluated against predefined standards or requirements, and it meets those standards. In the context of network management and device monitoring, a successful compliance check means that the device's configuration, performance, and operational behavior align with the specified guidelines or policies.

### 1. Configuration Compliance

- **Correct Configuration:** The device's settings align with the approved configuration parameters. This includes:
  - **Network Settings:** Proper IP addressing, routing configurations, and network segmentation.
  - **Radio Parameters:** Correct settings for frequency bands, bandwidth, transmission power, and other radio configuration parameters.
  - **Protocol Settings:** Proper implementation of 5G protocols and standards.
- **Software and Firmware:** The device is running the approved or recommended software and firmware versions, ensuring compatibility and functionality.

### 2. Performance Compliance

- **Latency:** The device meets the required latency benchmarks, ensuring that it supports low-latency applications and services as per 5G requirements (e.g., latency below 50ms).
- **Throughput:** The data throughput matches or exceeds the performance standards defined for the device, providing sufficient bandwidth for users (e.g., throughput of 1Gbps or more).
- **Reliability:** The device operates reliably without frequent errors or performance degradation. Metrics like packet loss, jitter, and error rates are within acceptable limits.

### 3. Security Compliance

- **Encryption:** Proper encryption methods are in place to protect data transmission and ensure confidentiality and integrity.

- **Authentication and Authorization:** The device enforces strong authentication and authorization mechanisms to prevent unauthorized access.
- **Patch Management:** The device has the latest security patches and updates applied, protecting it from known vulnerabilities.

**4. Regulatory Compliance**

- **Data Privacy:** The device and network adhere to relevant data privacy regulations, such as GDPR, ensuring that user data is handled appropriately.
- **Industry Standards:** The device complies with industry standards and specifications for 5G technology, such as those defined by the 3rd Generation Partnership Project (3GPP).
- **Documentation and Reporting:** All necessary documentation and reporting requirements are met, providing transparency and accountability for compliance.

**5. Operational Compliance**

- **Connection Stability:** The device maintains a stable connection without frequent disconnections or failures.
- **Resource Utilization:** The device's resource usage (e.g., CPU, memory, storage) is within acceptable limits, ensuring efficient operation.

## 1.5 COMPLIANCE CHECK FAILURE

**Compliance Check Failure** in a 5G network refers to the situation where a network device or system does not meet the established standards, configurations, or regulatory requirements. This failure can have implications for performance, security, and regulatory adherence. Here's a detailed look at what constitutes a compliance check failure in the context of a 5G network:

### 1. Configuration Non-Compliance

- **Incorrect Settings:** The device settings do not match the predefined or required configurations. For example:
    - **Network Configuration:** IP addressing, routing, or VLAN settings do not align with network design specifications.
    - **Radio Parameters:** Incorrect settings for frequency bands, bandwidth, or transmission power that deviate from the 5G standards or operator requirements.
- **Version Mismatch:** The device firmware or software version is outdated or does not comply with the recommended or mandatory versions.

### 2. Performance Issues

- **Latency:** The device exhibits latency that exceeds acceptable thresholds, affecting the quality of service (QoS). For instance, if the latency is higher than the required 50ms, it can impact user experience.
- **Throughput:** The data throughput is below the expected performance levels, which could be due to misconfiguration or hardware issues.
- **Error Rates:** High error rates or packet loss that affect network reliability and performance.

### 3. Security Policy Violations

- **Unauthorized Access:** Device configurations allow unauthorized access or do not implement proper authentication and authorization mechanisms.
- **Lack of Encryption:** Security protocols such as encryption are not properly applied, leading to potential data breaches.
- **Outdated Security Patches:** The device has not applied critical security patches or updates, making it vulnerable to known threats.

**4. Regulatory Non-Compliance**

- **Data Privacy:** The device or network does not comply with regulations related to data privacy, such as GDPR or HIPAA, potentially leading to legal issues.
- **Standards Adherence:** The device does not meet industry standards or specifications for 5G technology, affecting interoperability and performance.

**5. Operational Issues**

- **Connection Problems:** The device frequently disconnects or fails to maintain a stable connection, affecting network stability and user connectivity.
- **Resource Utilization:** The device exceeds acceptable thresholds for resource utilization (CPU, memory, etc.), leading to performance degradation.

## 2.1 HARDWARE REQUIREMENTS

| | |
|---|---|
| Processor | Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM |
| RAM | Minimum 16 GB |
| HARD DISK | 100 GB |

## 2.2 SOFTWARE REQUIREMENTS

## 2.3 SOFTWARE ENVIRONMENT

C++ is a powerful, versatile programming language that is widely used for system/software development, game development, high-performance applications, and more. Developed by Bjarne Stroustrup in the early 1980s as an enhancement to the C programming language, C++ introduces object-oriented programming (OOP) features along with other enhancements.

**Key Features of C++**

1. **Object-Oriented Programming (OOP):**
   - **Classes and Objects:** C++ introduces classes and objects, which allow for the encapsulation of data and functions into reusable and modular units.
   - **Inheritance:** Classes can inherit attributes and methods from other classes, enabling code reuse and the creation of hierarchical relationships.
   - **Polymorphism:** C++ supports both compile-time and runtime polymorphism, allowing for function overloading and dynamic method binding.
   - **Encapsulation:** Data hiding and access control through private, protected, and public access specifiers.
2. **Low-Level Manipulation:**
   - **Pointers:** C++ provides direct access to memory and the ability to manipulate addresses through pointers.
   - **Memory Management:** Supports dynamic memory allocation and deallocation using new and delete operators.

3. **Performance:**
    - **Efficiency:** C++ is known for its performance and efficiency, making it suitable for system-level programming, real-time systems, and applications requiring high computational power.
    - **Control:** It offers fine-grained control over system resources and hardware.
4. **Standard Template Library (STL):**
    - **Containers:** Provides a collection of pre-written classes for common data structures such as vectors, lists, queues, and maps.
    - **Algorithms:** Includes a wide range of algorithms for operations such as sorting and searching.
    - **Iterators:** Allows for traversal and manipulation of container elements.
5. **Generic Programming:**
    - **Templates:** C++ supports generic programming through templates, enabling the creation of functions and classes that work with any data type.
6. **Multi-Paradigm Language:**
    - **Procedural Programming:** Supports traditional procedural programming with functions and procedural logic.
    - **Functional Programming:** Provides features such as lambda expressions for functional programming techniques.
7. **Exception Handling:**
    - **Error Handling:** C++ includes built-in support for handling errors and exceptions using try, catch, and throw keywords.
8. **Compatibility:**
    - **C Compatibility:** C++ is largely compatible with C, meaning that most C code can be compiled with a C++ compiler. This allows for the use of existing C libraries and codebases.

**GNS3 (Graphical Network Simulator-3)** is a powerful network simulation tool used primarily for designing, configuring, and testing network topologies. It is widely utilized by network engineers, administrators, and students for learning and practicing networking concepts, troubleshooting network configurations, and preparing for networking certifications.

## Key Features of GNS3

1. **Network Simulation and Emulation:**
    - **Simulation:** GNS3 allows users to simulate various network devices such as routers, switches, firewalls, and more, using virtualized images.
    - **Emulation:** Unlike simple network simulators, GNS3 can emulate real network device software, providing a closer approximation to real-world behavior.
2. **Device Integration:**
    - **Virtual Machines (VMs):** Supports integration with VMs running network operating systems like Cisco IOS, Juniper JUNOS, or Linux-based systems.
    - **Real Devices:** Can connect to real hardware devices for hybrid simulations combining virtual and physical components.
3. **Graphical Interface:**
    - **Drag-and-Drop:** Provides a graphical user interface where users can drag and drop network devices to create topologies.
    - **Configuration:** Users can configure devices using CLI (Command Line Interface) or GUI-based methods.
4. **Support for Various Network Devices:**
    - **Router and Switch Images:** Compatible with a wide range of router and switch images from vendors like Cisco, Juniper, and others.

    - **Network Appliances:** Supports various network appliances and security devices for more complex scenarios.
5. **Interactivity and Testing:**
    - **Real-Time Testing:** Enables real-time interaction with network devices for configuration, troubleshooting, and testing.
    - **Capture and Analysis:** Allows packet capture and analysis using tools like Wireshark integrated within the simulation environment.
6. **Topology Design:**
    - **Custom Topologies:** Users can design custom network topologies and test different network configurations and scenarios.
    - **Scalability:** Supports the creation of complex network designs with multiple devices and interconnections.

## 3.Ping Method

**ICMP Protocol:**

**Description:**

The ping method is a network utility that uses the Internet Control Message Protocol (ICMP) to test the reachability of a network device, such as a router or computer, and measure the round-trip time for messages sent from the originating host to a destination host. It provides a quick and simple way to check if a device is reachable on the network and if it is responding to network requests.

**How It Works**:

1. **ICMP Echo Request**:
    - When a ping command is executed, an ICMP Echo Request packet is sent from the source device to the target device. This packet includes a payload of data and a unique identifier to match the request with its response.
2. **ICMP Echo Reply**:
    - Upon receiving the ICMP Echo Request, the target device sends back an ICMP Echo Reply packet to the source device. This reply packet contains the same data as the request packet, allowing the source device to measure the round-trip time and check for any packet loss.
3. **Timeouts and Errors**:
    - If the target device does not respond within a specified timeout period, or if the request is not received, the ping command will indicate that the device is unreachable or that there is a network issue.

**Command Example:**

```
ping -c 1 <IP_Address>
```

- -c 1 specifies that only one ping request should be sent. The <IP_Address> is the IP address of the target device.

**Advantages:**

- **Simplicity:** The ping command is easy to use and does not require any special configuration on the target device.
- **Widely Supported:** It is a standard network utility available on most operating systems and network devices.
- **Quick Diagnostics:** Provides immediate feedback on network connectivity and basic troubleshooting.

**Limitations:**

- **Limited Information:** Ping only verifies reachability and measures response time but does not provide detailed information about the state or performance of the device.
- **Firewall and Security:** Some devices may block ICMP traffic or be configured to ignore ping requests, making them appear unreachable even if they are online.

**4.SNMP METHOD:**

**Description:**

SNMP (Simple Network Management Protocol) is a protocol used for network management, monitoring, and configuration. It enables administrators to query and manage network devices such as routers, switches, servers, and other infrastructure components. SNMP provides a structured approach to retrieve detailed information about device status, configuration, and performance**.**

**How It Works:**

1. **SNMP Manager and Agent:**
   - SNMP Manager: The management system that sends requests and receives responses. It is typically a network management software running on a server or workstation.
   - SNMP Agent: The software running on network devices that responds to queries and sends notifications (traps) to the SNMP Manager.
2. **SNMP Requests:**
   - GET Request: The manager queries the agent for specific information from the device's Management Information Base (MIB). For example, retrieving the device's system name or interface status.
   - SET Request: The manager sends configuration changes to the agent, such as modifying device settings.
   - GETNEXT Request: Retrieves the next item in the MIB hierarchy, useful for walking through MIB tables.
   - TRAP: The agent sends unsolicited notifications to the manager about certain events or conditions, such as device failures or thresholds exceeded**.**
3. **Management Information Base (MIB):**

    - MIB is a hierarchical database used by SNMP to store information about network devices. It organizes data into objects, each with a unique Object Identifier (OID) that the manager uses to query or configure the device.

4. **OID (Object Identifier):**
    - An OID is a numerical label used to uniquely identify managed objects in the MIB. Each object represents a specific piece of information or configuration parameter on the device.

**Command Example:**

To retrieve information using SNMP, you might use a command like:

```
snmpget -v2c -c <community_string> <IP_Address> <OID>
```

- -v2c specifies the SNMP version (e.g., SNMPv2c).
- -c <community_string> specifies the SNMP community string for authentication.
- <IP_Address> is the IP address of the target device.
- <OID> is the Object Identifier for the information being queried.

**Advantages:**

- **Detailed Information:** Provides comprehensive details about device status, configuration, and performance.
- **Management and Configuration:** Allows for both monitoring and management of devices, including configuration changes.
- **Standardized Protocol:** Widely used and supported across different network devices and vendors.

**Limitations:**

- **Configuration Required:** Devices must be configured to support SNMP, including setting community strings and enabling SNMP services.
- **Complexity:** SNMP can be more complex to configure and use compared to simpler tools like ping.

5. **Device Status Monitoring**

   **Description:**
   Device status monitoring refers to the process of continuously checking and tracking the operational state of network devices, such as routers, switches, servers, and other critical infrastructure components. This is essential for ensuring that devices are functioning correctly and are reachable within the network. Monitoring device status helps in early detection of issues, preventing downtime, and maintaining network health.
   **Methods:**

1. **ICMP (Internet Control Message Protocol):**
   - **Ping:** The most common method for checking the status of a device is using the ping command, which utilizes ICMP. The ping command sends ICMP Echo Request packets to the target device and waits for Echo Reply packets to determine if the device is reachable.

Advantages:

Simple and Quick: Ping provides a straightforward way to check device availability.
Widely Supported: Most network devices and operating systems support ICMP and respond to ping requests.
Limitations:
Limited Information: Ping only indicates whether a device is reachable or not; it does not provide detailed status or configuration information.
Firewall and Security Settings: Some devices may block ICMP packets for security reasons, causing ping tests to fail even if the device is operational.
SNMP (Simple Network Management Protocol):

SNMP Queries: For more detailed monitoring, SNMP can be used to query devices for their status and performance metrics. SNMP agents on devices provide information about various status parameters, such as interface up/down status, CPU usage, memory usage, and more.

1. 
    - **Advantages:**
        - **Detailed Status:** SNMP provides in-depth information about device status, including performance metrics and operational parameters.
        - **Comprehensive Monitoring:** Allows for ongoing monitoring and collection of performance data over time.
    - **Limitations:**
        - **Configuration Required:** Devices must be configured to support SNMP, and proper community strings or authentication must be set.
        - **Complexity:** SNMP can be more complex to set up and use compared to simple ping commands.

**Implementation:**

1. **ICMP Ping Implementation:**
    - Use the ping command in scripts or monitoring tools to periodically check if devices are reachable.
    - Integrate with alerting systems to notify administrators of unreachable devices.
2. **SNMP Monitoring Implementation:**
    - Set up SNMP agents on network devices and configure SNMP managers to collect and analyze device status information.
    - Use SNMP monitoring tools or custom scripts to query devices and gather performance metrics.

## 6. COMPLIANCE CHECKING

### Compliance Checking

**Description:**

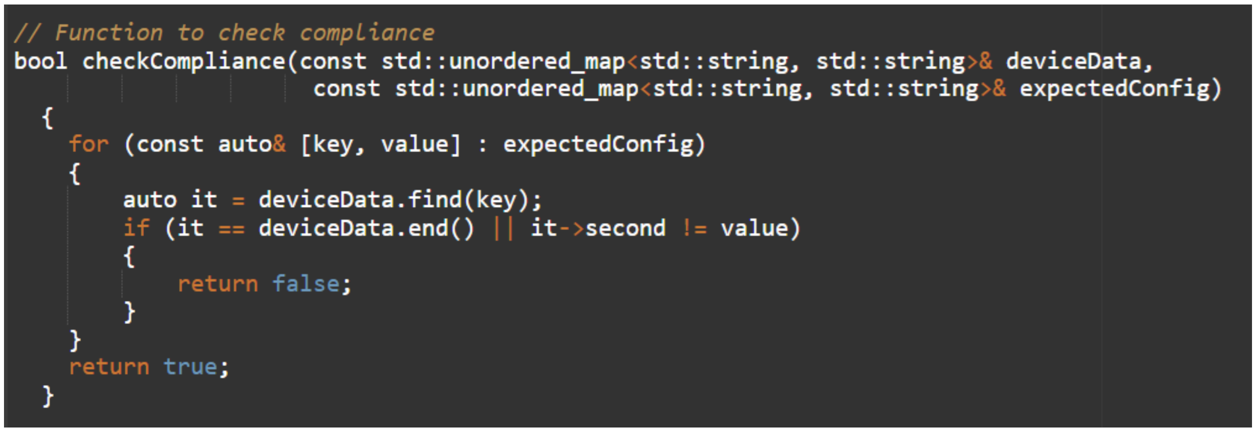
Compliance checking in the context of network management refers to the process of verifying that network devices adhere to predefined configuration standards and operational policies. This involves comparing the actual configuration and status of devices against expected or required criteria to ensure they meet organizational or regulatory requirements.

**Methods:**

1. **SNMP-Based Compliance Checking:**
   - **Using SNMP Queries:** SNMP can be utilized to retrieve configuration and status information from network devices. By querying specific Object Identifiers (OIDs) related to device settings, you can compare the retrieved data against expected values to determine compliance.

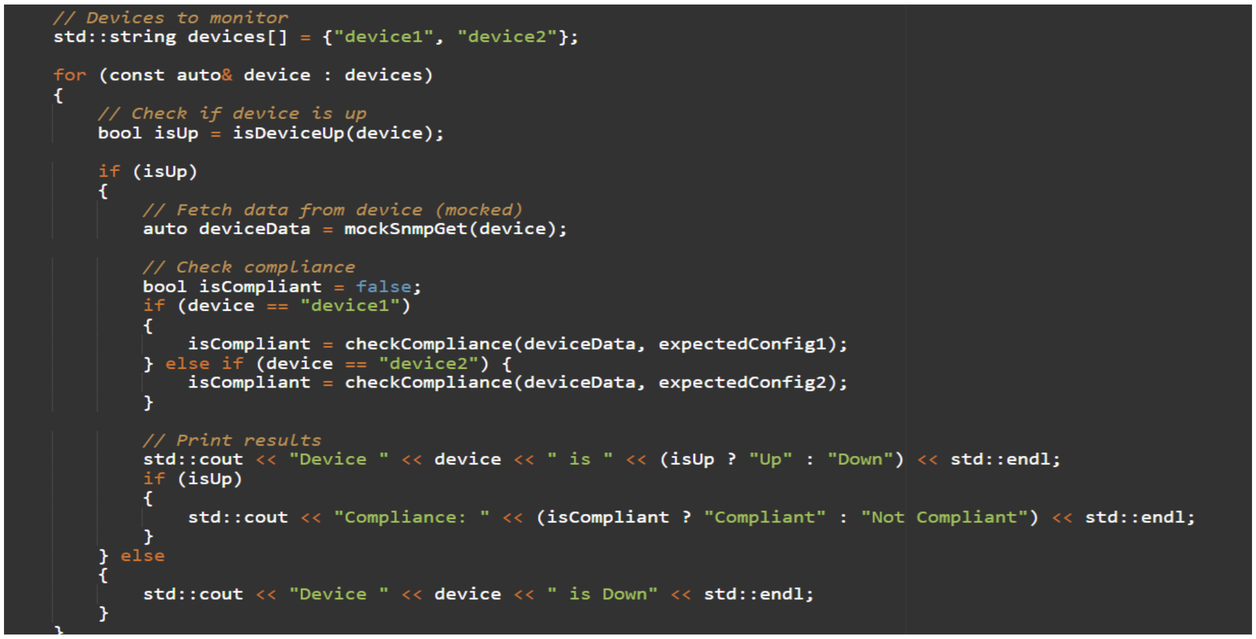
**Advantages:**

- **Detailed Data Retrieval:** SNMP allows for detailed retrieval of device configurations and status information.
- **Automated Checking:** SNMP-based checks can be automated, making it easier to continuously monitor compliance.



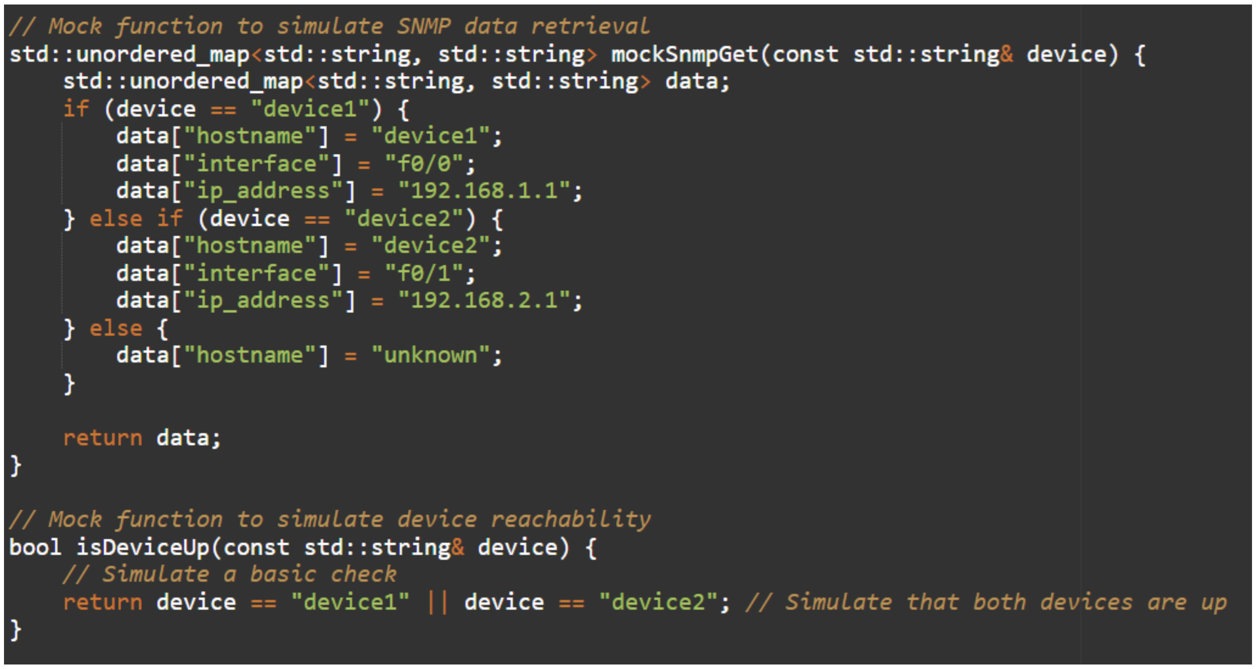
```
// Function to check compliance
bool checkCompliance(const std::unordered_map<std::string, std::string>& deviceData,
                      const std::unordered_map<std::string, std::string>& expectedConfig)
  {
    for (const auto& [key, value] : expectedConfig)
    {
        auto it = deviceData.find(key);
        if (it == deviceData.end() || it->second != value)
        {
            return false;
        }
    }
    return true;
  }
```

## 7. DEVICE MONITORING AND COMPLIANCE CHECKING IMPLEMENTATION



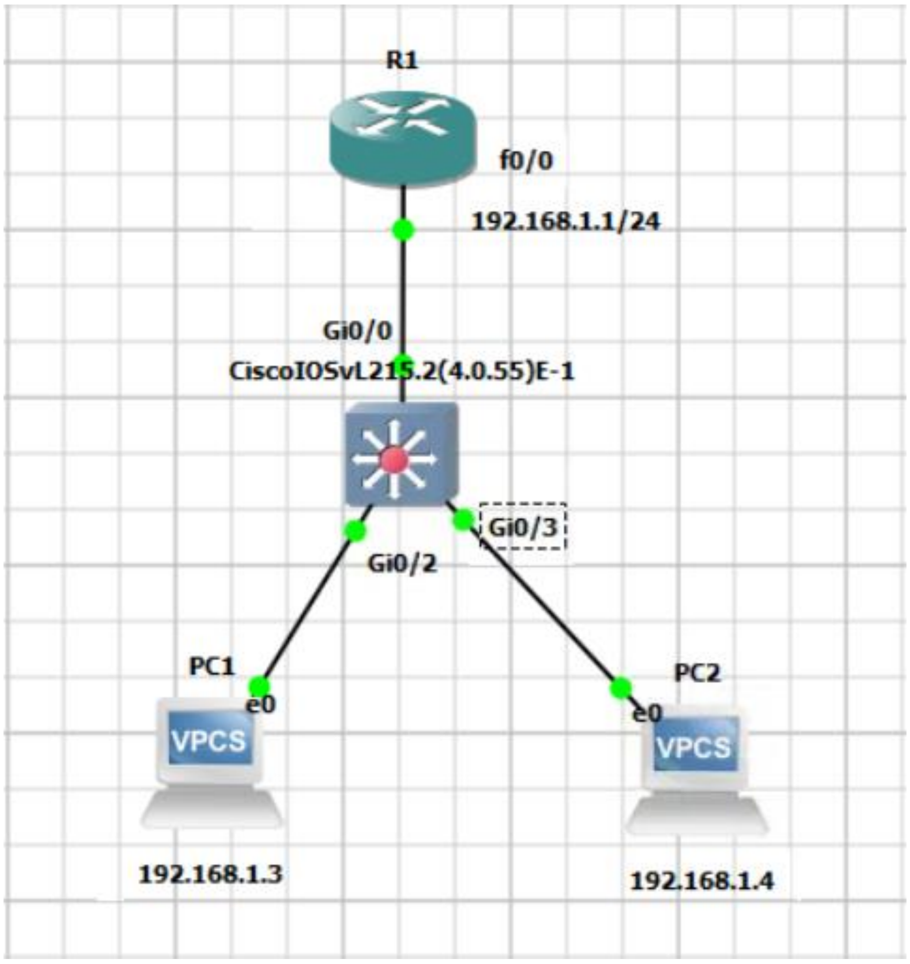
```cpp
// Devices to monitor
std::string devices[] = {"device1", "device2"};

for (const auto& device : devices)
{
    // Check if device is up
    bool isUp = isDeviceUp(device);

    if (isUp)
    {
        // Fetch data from device (mocked)
        auto deviceData = mockSnmpGet(device);

        // Check compliance
        bool isCompliant = false;
        if (device == "device1")
        {
            isCompliant = checkCompliance(deviceData, expectedConfig1);
        } else if (device == "device2") {
            isCompliant = checkCompliance(deviceData, expectedConfig2);
        }

        // Print results
        std::cout << "Device " << device << " is " << (isUp ? "Up" : "Down") << std::endl;
        if (isUp)
        {
            std::cout << "Compliance: " << (isCompliant ? "Compliant" : "Not Compliant") << std::endl;
        }
    } else
    {
        std::cout << "Device " << device << " is Down" << std::endl;
    }
```

## 8. Mock SNMP Function



```cpp
// Mock function to simulate SNMP data retrieval
std::unordered_map<std::string, std::string> mockSnmpGet(const std::string& device) {
    std::unordered_map<std::string, std::string> data;
    if (device == "device1") {
        data["hostname"] = "device1";
        data["interface"] = "f0/0";
        data["ip_address"] = "192.168.1.1";
    } else if (device == "device2") {
        data["hostname"] = "device2";
        data["interface"] = "f0/1";
        data["ip_address"] = "192.168.2.1";
    } else {
        data["hostname"] = "unknown";
    }

    return data;
}

// Mock function to simulate device reachability
bool isDeviceUp(const std::string& device) {
    // Simulate a basic check
    return device == "device1" || device == "device2"; // Simulate that both devices are up
}
```

## 9.TOPOLOGY


R1
f0/0
192.168.1.1/24
Gi0/0
CiscoIOSvL215.2(4.0.55)E-1
Gi0/3
Gi0/2
PC1
e0
VPCS
PC2
e0
VPCS
192.168.1.3
192.168.1.4

## 10.CODING

```
#include <iostream>

#include <unordered_map>

#include <string>


// Mock function to simulate  data retrievalSNMP

std::unordered_map<std::string, std::string> mockSnmpGet(const std::string&
device) {

   std::unordered_map<std::string, std::string> data;

   if (device == "device1") {

      data["hostname"] = "device1";

      data["interface"] = "f0/0";

      data["ip_address"] = "192.168.1.1";

   } else if (device == "device2") {

      data["hostname"] = "device2";

      data["interface"] = "f0/1";

      data["ip_address"] = "192.168.2.1";

   } else {

      data["hostname"] = "unknown";

   }


   return data;

}


// Mock function to simulate device reachability
```

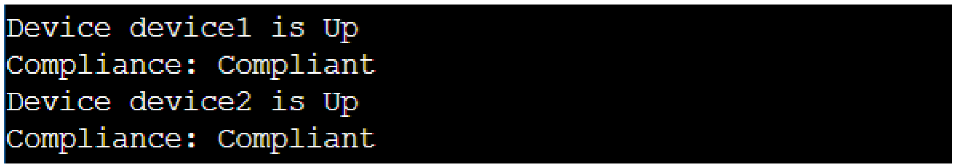
```
bool isDeviceUp(const std::string& device) {
    // Simulate a basic check
    return device == "device1" || device == "device2"; // Simulate that both devices
are up
}

// Function to check compliance
bool checkCompliance(const std::unordered_map<std::string, std::string>&
deviceData,
                     const std::unordered_map<std::string, std::string>& expectedConfig)
{
    for (const auto& [key, value] : expectedConfig) {
        auto it = deviceData.find(key);
        if (it == deviceData.end() || it->second != value) {
            return false;
        }
    }
    return true;
}

int main() {
    // Define expected configurations
    std::unordered_map<std::string, std::string> expectedConfig1 = {
        {"hostname", "device1"},
        {"interface", "f0/0"},
        {"ip_address", "192.168.1.1"}

    };

    std::unordered_map<std::string, std::string> expectedConfig2 = {
        {"hostname", "device2"},
        {"interface", "f0/1"},
        {"ip_address", "192.168.2.1"}

    };

    // Devices to monitor
    std::string devices[] = {"device1", "device2","device3"};

    for (const auto& device : devices) {
        // Check if device is up
        bool isUp = isDeviceUp(device);

        if (isUp) {
            // Fetch data from device (mocked)
            auto deviceData = mockSnmpGet(device);

            // Check compliance
            bool isCompliant = false;
```

```
            if (device == "device1") {
                isCompliant = checkCompliance(deviceData, expectedConfig1);
            } else if (device == "device2") {
                isCompliant = checkCompliance(deviceData, expectedConfig2);
            }

            // Print results
            std::cout << "Device " << device << " is " << (isUp ? "Up" : "Down") <<
std::endl;
            if (isUp) {
                std::cout << "Compliance: " << (isCompliant ? "Compliant" : "Not
Compliant") << std::endl;
            }
        } else {
            std::cout << "Device " << device << " is Down" << std::endl;
        }
    }

    return 0;
}
```

## 11. Output

### Test Scenario-1:

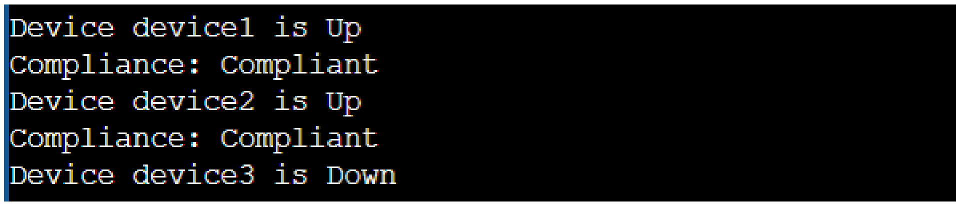
- If all devices are up and compliant



```
Device device1 is Up
Compliance: Compliant
Device device2 is Up
Compliance: Compliant
```

### Test Scenario-2:

- If two devices are up and third device is down



```
Device device1 is Up
Compliance: Compliant
Device device2 is Up
Compliance: Compliant
Device device3 is Down
```