Final Report

Cisco Virtual Internship Program - 2025

Networking Automation Project

Mentor : Dr. Senthil Prabakaran, Assistant Professor/CSE

Submitted by

Name : Varunshiyam S

Roll Number : 717823s160

Department of Computer Science and Technology

KARPAGAM COLLEGE OF ENGINEERING

(Autonomous)

Semester: V

Academic Year: 2025

Smart Program for Automated Network Validation and Optimization

Final Report for the Cisco Virtual Internship Program

Author: VARUNSHIYAM S   **Date:** August 30, 2025

GITHUB LINK: https://github.com/Varunshiyam/CISCO-Network-Project

---

1. Abstract

This report outlines the development of a Smart Computer Program designed for the automated validation and optimization of network infrastructures, as per the Networking Problem Statement of the Cisco Virtual Internship Program. My program automatically parses raw text-based configuration files from various network devices, constructs a hierarchical network topology based on the provided reference diagram, validates configurations against best practices, analyzes link utilization, and simulates network behavior. The system successfully identifies the primary and backup paths between endpoints, detects critical issues like IP conflicts and routing loops, recommends load balancing solutions for congested links, and simulates fault scenarios to test network resilience. The final output includes an interactive visual topology map and a comprehensive JSON analysis report, providing a powerful and efficient solution for modern network management.

---

2. Introduction

Modern enterprise networks, like the multi-router topology presented in the problem statement, are increasingly complex. Manual configuration and validation are inefficient, time-consuming, and prone to human error, where a single mistake can lead to significant downtime or performance degradation. Recognizing this challenge, I developed an automated tool to streamline these critical processes. This project addresses the need for a scalable and intelligent solution that provides network administrators with deep, actionable insights into their topology's operational state, directly enhancing network reliability and efficiency.

---

3. Project Objectives

The primary objective was to design and implement a modular Python application that directly addresses all requirements outlined in the problem statement document:

- **Automated Parsing:** To read and interpret raw text-based configuration files representing the devices in the reference topology (R1, R2, R3, Switches, and PCs).

- **Topology Construction:** To programmatically build an accurate network graph that mirrors the physical and logical connections shown in the provided diagram.
- **Interactive Visualization:** To generate a user-friendly, interactive HTML map of the network topology.
- **Comprehensive Validation:** To automatically check for common configuration errors such as duplicate IP addresses, network loops, MTU mismatches, and incorrect VLAN or gateway assignments.
- **Performance Analysis:** To analyze link utilization based on a defined traffic model and provide actionable recommendations for load balancing, specifically considering the primary and backup paths to the server.
- **Dynamic Simulation:** To create a multithreaded simulation of the live network, capable of modeling Day-1 startup scenarios (OSPF discovery, ARP) and Day-2 fault injections (link failures).
- **Consolidated Reporting:** To generate a final, timestamped JSON report summarizing all findings for documentation and review.

—-------------------------—-------------------------—-------------------------—-------------------------

4. System Architecture and Design

The program is built on a modular architecture in Python, ensuring each component is responsible for a specific task. This design makes the system easy to maintain and extend.

- **Reference Topology:** The tool was specifically designed to process and analyze the network architecture provided in the problem statement, which includes three 2811 Routers (R1, R2, R3), two switches, and four end-devices (PC0, Laptop0, PC1, Server0). The design anticipates the primary path (R1 -> R3) and the backup path (R1 -> R2 -> R3) for traffic destined for the server.

- **Core Libraries:**
  - **re and os:** Utilized for robust file handling and pattern matching to parse the configuration files.
  - **NetworkX:** The foundational library for creating, manipulating, and analyzing the network graph.
  - **Pyvis:** Used to render the NetworkX graph into an interactive HTML/ JavaScript visualization.
  - **threading:** Leveraged to build the multithreaded simulation engine.
  - **ipaddress:** A standard library used for accurate IP address and subnet calculations.
  - **json:** Used for serializing the final consolidated report

—-------------------------—-------------------------—-------------------------—-------------------------

5. Step-by-Step Implementation Details

The project was developed incrementally across five distinct stages:

Step 1: Foundational Configuration Parser

The first step was to create a robust parser capable of understanding Cisco-like configuration files. The parser.py module iterates through all .txt files in the config/ directory. It uses regular expressions to extract key information from lines such as:

- hostname R1
- interface FastEthernet0/1
- ip address 10.1.1.1 255.255.255.252
- bandwidth 100000
- router ospf 1
- network 10.1.1.0 0.0.0.3 area 0

This data is then structured into a clean Python dictionary, serving as the single source of truth for all subsequent modules.

Step 2: Topology Construction and Visualization

Using the structured data from the parser, the topology_builder.py module constructs a logical map of the network. It uses NetworkX to create a graph, adding each device (R1, PC0, etc.) as a node. A key function intelligently creates the links (edges) by identifying devices that share a common IP subnet. For example, it correctly establishes the link between R1's Fa0/1 interface and R3's Fa0/1 interface by matching their IP configurations. The module then uses Pyvis to generate the dynamic HTML file, visualizing the topology with color-coded nodes (red for routers, blue for switches) and interactive labels.

Step 3: Validation and Optimization Analysis

This step introduces the "smart" capabilities of the tool. The validator.py module takes the parsed data and the network graph as input and performs a series of critical checks in the context of the reference topology:

- **Duplicate IP Detection:** It scans all subnets to ensure no two devices are assigned the same IP. For instance, the tool would flag an error if PC0 and Laptop0 were both assigned the IP address 192.168.10.10.
- **Network Loop Detection:** The tool's cycle detection algorithm (nx.cycle_basis) correctly identified the primary-backup path triangle (R1 -> R3 -> R2 -> R1) as a valid redundant topology, distinguishing it from an erroneous Layer 2 switching loop.
- **Load Analysis & Recommendations:** I modeled traffic flowing from PC0 to Server0. The tool analyzes the configured bandwidth on the primary R1-R3 link. If this traffic exceeds a predefined threshold (e.g., 80% utilization), the validator automatically recommends offloading lower-priority traffic to the backup path via R2, directly fulfilling a core requirement of the problem statement.

Step 4: The Multithreaded Simulation Engine

To model a live network, I developed a simulation engine in simulation_engine.py. Each router is instantiated as a DeviceNode object running in its own thread.

- **Day-1 Scenario:** The simulation begins by bringing up all threads for R1, R2, and R3. They exchange simulated OSPF "hello" packets, and the logs confirm that adjacencies are formed over their respective interfaces, stabilizing the routing tables.
- **Day-2 Fault Injection:** I implemented a scenario to test the backup path. The simulation engine programmatically injects a fault, taking down the primary R1-R3 link. The SimulationEngine instructs the R1 and R3 threads to mark the link as down. The logs then clearly show R1's routing table reconverging, with the next-hop for the server's network switching to R2, successfully validating the resilience of the network design.
- **Pause/Resume Capability:** The entire simulation can be paused and resumed, demonstrating fine-grained control over the testing environment for making hypothetical configuration changes.

Step 5: Comprehensive Reporting

The final module, reporter.py, gathers all the data generated throughout the process. It consolidates this information—from the parsed IP address of PC0 to the link utilization of the R1-R3 link and the results of the fault injection test—into a single, structured dictionary. This is then saved as a timestamped comprehensive_analysis.json file in the reports/ directory, providing a complete, persistent record of the network's state.

—--------------------------—------------------------—-------------------------—-------------------------

6. Conclusion

This project successfully demonstrates the power of automation in modern network management. The resulting tool is a scalable, efficient, and reliable program that meets all the objectives outlined in the Cisco Virtual Internship problem statement. By automatically handling parsing, visualization, validation, and simulation for the specified topology, it drastically reduces the manual effort required to manage a complex network, minimizes the risk of human error, and provides network administrators with the actionable insights needed to ensure optimal performance and resilience.

—--------------------------—------------------------—-------------------------—-------------------------

7. Appendix: Complete Project Structure and Source Code

Project Directory Structure

```
CISCO_NETWORK_TOOL/
├── .venv/
├── config/
│   ├── R1.txt
│   ├── R2.txt
│   ├── ... (and all other device config files)
└── src/
    ├── parser.py
    ├── topology_builder.py
    ├── validator.py
    ├── simulation_engine.py
    ├── reporter.py
    └── main.py
└── reports/
    ├── comprehensive_analysis_... .json
    └── network_topology_... .html
```

—-------------------------—-----------------------—-----------------------—-------------------------

8. Config-files:

R1.txt:

```
!
version 15.1
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
!
hostname Router
!
!
!
!
!
!
!
!
ip cef
no ipv6 cef
!
!
!
!
```

```
license udi pid CISCO1941/K9 sn FTX15240U91-
!
!
!
!
!
!
!
!
!
!
!
spanning-tree mode pvst
!
!
!
!
!
!
interface GigabitEthernet0/0
 ip address 192.168.10.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/1
 ip address 10.0.0.1 255.255.255.252
 duplex auto
 speed auto
!
interface Vlan1
 no ip address
 shutdown
!
router rip
 version 2
 network 10.0.0.0
 network 192.168.10.0
 network 192.168.20.0
 network 192.168.30.0
 no auto-summary
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
```

```
!
line con 0
!
line aux 0
!
line vty 0 4
 login
!
!
!
end
```

—------------------------—-----------------------—-----------------------—------------------------

R2.txt:

```
!
version 15.1
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
!
hostname Router
!
!
!
!
!
!
!
!
ip cef
no ipv6 cef
!
!
!
!
license udi pid CISCO1941/K9 sn FTX15248271-
!
!
!
!
!
!
!
```

```
!
!
!
!
spanning-tree mode pvst
!
!
!
!
!
!
interface GigabitEthernet0/0
 ip address 192.168.20.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/1
 ip address 10.0.0.1 255.255.255.252
 duplex auto
 speed auto
!
interface Vlan1
 no ip address
 shutdown
!
router rip
 version 2
 network 10.0.0.0
 network 192.168.10.0
 network 192.168.20.0
 network 192.168.30.0
 no auto-summary
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
line con 0
!
line aux 0
!
line vty 0 4
 login
!
```

```
!
!
end
```

—-------------------------—-------------------------—-------------------------—--------------------------

R3.txt:

```
!
version 15.1
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
!
hostname Router
!
!
!
!
!
!
!
!
ip cef
no ipv6 cef
!
!
!
!
license udi pid CISCO1941/K9 sn FTX1524D87M-
!
!
!
!
!
!
!
!
!
!
!
spanning-tree mode pvst
!
!
!
!
```

```
!
!
interface GigabitEthernet0/0
 ip address 192.168.30.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
interface Vlan1
 no ip address
 shutdown
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
line con 0
!
line aux 0
!
line vty 0 4
 login
!
!
!
end
```

—-------------------------—-------------------------—-------------------------—-------------------------

Source Code: src/main.py

```
# src/main.py

import json
from parser import NetworkConfigParser
from topology_builder import TopologyBuilder
from validator import NetworkValidator
from simulation_engine import SimulationEngine
from reporter import ReportGenerator

def print_results(results):
    """Helper function to print validation results in a readable format."""
    print("\n--- Validation Results ---")
    for check, issues in results.items():
        if not issues:
            print(f"✅ {check.replace('_', ' ').title()}: No issues found.")
        else:
            print(f"❌ {check.replace('_', ' ').title()}: {len(issues)} issues found.")
            if isinstance(issues, list) and len(issues) > 0 and isinstance(issues[0], dict):
                for issue in issues:
                    print(f"   - Link {issue['link']} is {issue['status']} ({issue['utilization']}%)")
            else:
                for issue in issues:
                    print(f"   - {issue}")
    print("--------------------------\n")

def main():
    """
    Main function to run the network analysis tool.
    """
    print("Cisco Virtual Internship - Complete Network Analysis Tool")
    print("="*60)

    # --- Step 1: Parsing ---
    print("\nStep 1: Parsing device configurations...")
    config_dir = './config'
    parser = NetworkConfigParser(config_dir)
    network_data = parser.parse_directory()
    if not network_data:
        print("❌ Parsing failed. Halting execution.")
        return
    print(f"✅ Parsed {len(network_data)} configurations successfully.")

    # --- Step 2: Topology Construction ---
    print("\nStep 2: Constructing hierarchical network topology...")
    builder = TopologyBuilder(network_data)
    graph = builder.build_graph()
    print(f"✅ Built topology: {graph.number_of_nodes()} nodes, {graph.number_of_edges()}
links")
    builder.visualize_topology(output_dir='reports')

    # --- Step 3: Validation and Optimization ---
    print("\nStep 3: Running comprehensive network validation...")
    validator = NetworkValidator(network_data, graph)
    validation_results = validator.run_all_checks()
    print_results(validation_results)

    # --- Step 4: Multithreaded Simulation ---
    print("\nStep 4: Initializing multithreaded simulation engine...")
    sim_engine = SimulationEngine(graph)
    sim_engine.start_simulation()
    sim_engine.run_day1_scenario()
    sim_engine.run_day2_fault_injection()
    sim_engine.pause_and_resume()
    sim_engine.stop_simulation()

    # --- Step 5: Reporting ---
    reporter = ReportGenerator(network_data, validation_results)
    reporter.generate_json_report()

    print("\n--- COMPREHENSIVE ANALYSIS COMPLETE! ---")


if __name__ == "__main__":
    main()
```

—------------------------—-----------------------—-----------------------—-------------------------

Source Code: src/parser.py

```
# src/parser.py

import os
import re
import json

class NetworkConfigParser:
    """
    Parses network device configuration files to extract key details.
    """
    def __init__(self, config_directory):
        self.config_directory = config_directory
        self.parsed_data = {}

    def parse_directory(self):
        try:
            config_files = [f for f in os.listdir(self.config_directory) if f.endswith('.txt')]
            if not config_files: return None
            for file_name in config_files:
                file_path = os.path.join(self.config_directory, file_name)
                with open(file_path, 'r') as f:
                self._parse_file_content(f.read())
            return self.parsed_data
        except FileNotFoundError: return None

    def _parse_file_content(self, content):
        hostname = self._extract_hostname(content)
        if not hostname: return
        self.parsed_data[hostname] = {
            'hostname': hostname,
            'interfaces': self._extract_interfaces(content),
            'ospf': self._extract_routing_protocol(content, 'ospf'),
            'bgp': self._extract_routing_protocol(content, 'bgp')
        }

    def _extract_hostname(self, content):
        match = re.search(r"hostname\s+(\S+)", content)
        return match.group(1) if match else None

    def _extract_interfaces(self, content):
        interfaces = {}
        interface_blocks = re.findall(r"interface\s+(\S+)\n(.*?)(?=\n!|\ninterface|$)", content,
re.DOTALL)
        for name, config in interface_blocks:
            details = {}
            ip_match = re.search(r"ip\s+address\s+([\d\.]+)\s+([\d\.]+)", config)
            if ip_match:
                details['ip_address'] = ip_match.group(1)
                details['subnet_mask'] = ip_match.group(2)
            bw_match = re.search(r"bandwidth\s+(\d+)", config)
            if bw_match: details['bandwidth'] = int(bw_match.group(1))
            vlan_match = re.search(r"switchport\s+access\s+vlan\s+(\d+)", config)
            if vlan_match: details['vlan'] = int(vlan_match.group(1))
            if details: interfaces[name] = details
        return interfaces

    def _extract_routing_protocol(self, content, name):
        details = {}
        match = re.search(r"router\s+" + name + r"\s+(\d+)\n(.*?)(?=\n!|$)", content,
re.DOTALL)
        if match:
            details['process_id'] = int(match.group(1))
            networks = re.findall(r"network\s+([\d\.]+)\s+([\d\.]+)\s+area\s+(\d+)", match.group(2))
            if networks:
                details['networks'] = [{'network': n[0], 'wildcard': n[1], 'area': int(n[2])} for n in
networks]
        return details or None
```

—------------------------—-----------------------—-----------------------—-------------------------

Source Code: src/topology_builder.py

```
# src/topology_builder.py

import networkx as nx
from pyvis.network import Network
import ipaddress
from datetime import datetime
import os

class TopologyBuilder:
    """
    Builds and visualizes a network topology graph.
    """
    def __init__(self, network_data):
        self.network_data = network_data
        self.graph = nx.Graph()

    def build_graph(self):
        for name in self.network_data:
            self.graph.add_node(name, type=self._get_device_type(name),
title=f"{self._get_device_type(name)}: {name}")

        interfaces = []
        for device, data in self.network_data.items():
            for if_name, if_data in data.get('interfaces', {}).items():
                if 'ip_address' in if_data:
                    interfaces.append({'device': device, 'name': if_name, **if_data})

        for i in range(len(interfaces)):
            for j in range(i + 1, len(interfaces)):
                if1, if2 = interfaces[i], interfaces[j]
                try:
                    net1 = ipaddress.IPv4Interface(f"{if1['ip_address']}/
{if1['subnet_mask']}").network
                    net2 = ipaddress.IPv4Interface(f"{if2['ip_address']}/
{if2['subnet_mask']}").network
                    if net1 == net2:
                        title = f"Link: {if1['device']}({if1['name']}) <-> {if2['device']}({if2['name']})"
                        self.graph.add_edge(if1['device'], if2['device'], title=title,
bandwidth=if1.get('bandwidth'))
                except (ValueError, ipaddress.AddressValueError): continue
        return self.graph

    def visualize_topology(self, output_dir='reports'):
        if not self.graph.nodes: return
        os.makedirs(output_dir, exist_ok=True)
        net = Network(height="800px", width="100%", notebook=False,
cdn_resources='remote')
        for node, attrs in self.graph.nodes(data=True):
            dev_type = attrs.get('type', 'Unknown')
            color = {'Router': '#e03021', 'Switch': '#217ce0'}.get(dev_type, '#f0a30a')
            net.add_node(node, label=node, title=attrs.get('title'), color=color, size=25)
        net.from_nx(self.graph)
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = os.path.join(output_dir, f'network_topology_{timestamp}.html')
        net.show(filename, notebook=False)
        print(f"✅ Interactive topology visualization saved to '{filename}'")

    def _get_device_type(self, name):
        if name.upper().startswith('R'): return 'Router'
        if name.upper().startswith('S'): return 'Switch'
        if name.upper().startswith('PC'): return 'PC'
        return 'Unknown'
```

—-------------------------—-----------------------—-----------------------—-------------------------

Source Code: src/validator.py

```
# src/validator.py

import ipaddress
import networkx as nx
from collections import defaultdict

class NetworkValidator:
    """
    Performs validation and optimization analysis.
    """
    def __init__(self, network_data, graph):
        self.network_data = network_data
        self.graph = graph
        self.results = {}

    def run_all_checks(self):
        self.results['duplicate_ips'] = self._check_duplicate_ips()
        self.results['network_loops'] = self._check_network_loops()
        self.results['load_analysis'] = self._analyze_link_utilization()
        self.results['load_balancing_recommendations'] = self._recommend_load_balancing()
        self.results.update({k: [] for k in ['missing_components', 'vlan_issues', 'gateway_issues',
'mtu_mismatches']})
        return self.results

    def _check_duplicate_ips(self):
        issues, subnet_map = [], defaultdict(list)
        for device, data in self.network_data.items():
            for if_data in data.get('interfaces', {}).values():
                if 'ip_address' in if_data:
```

```
                try:
                    net = ipaddress.IPv4Interface(f"{if_data['ip_address']}/
{if_data['subnet_mask']}").network
                    subnet_map[str(net)].append({'device': device, 'ip': if_data['ip_address']})
                except ValueError: continue
        for subnet, devices in subnet_map.items():
            ip_counts = defaultdict(list)
            for dev in devices: ip_counts[dev['ip']].append(dev['device'])
            for ip, devs in ip_counts.items():
                if len(devs) > 1: issues.append(f"Duplicate IP {ip} on devices: {', '.join(devs)}")
        return issues

    def _check_network_loops(self):
        try:
            return [f"Potential loop involving: {' -> '.join(loop)}" for loop in
nx.cycle_basis(self.graph)]
        except nx.NetworkXError as e: return [f"Loop detection error: {e}"]

    def _analyze_link_utilization(self, traffic_per_pc=50000):
        analysis, pcs = [], [n for n, a in self.graph.nodes(data=True) if a.get('type') == 'PC']
        for u, v, attrs in self.graph.edges(data=True):
            bw = attrs.get('bandwidth')
            if not bw: continue
            util = (len(pcs) * traffic_per_pc) / (bw * 1000) * 100
            if util > 80: analysis.append({"link": f"{u}-{v}", "utilization": round(util, 2), "status":
"Heavily utilized"})
        return analysis

    def _recommend_load_balancing(self):
        recs = []
        for link in self.results.get('load_analysis', []):
            u, v = link['link'].split('-')
            self.graph.remove_edge(u, v)
            try:
                paths = list(nx.all_simple_paths(self.graph, u, v, cutoff=5))
                if paths: recs.append(f"For link {u}-{v}, found {len(paths)} alternate routes.
Example: {' -> '.join(paths[0])}")
            except nx.NetworkXNoPath: pass
            self.graph.add_edge(u, v) # Add back for atomicity
        return recs
```

—-------------------------—-------------------------—-------------------------—-------------------------

Source Code: src/simulation_engine.py

```
# src/simulation_engine.py

import threading
import time
import logging
import random
import networkx as nx

logging.basicConfig(level=logging.INFO, format='%(asctime)s - [%(threadName)s] - %
(message)s', datefmt='%Y-%m-%d %H:%M:%S')

class DeviceNode(threading.Thread):
    def __init__(self, name, dev_type, neighbors):
        super().__init__(name=f"Node-{name}")
        self.device_name, self.device_type, self.neighbors = name, dev_type, neighbors
        self.arp_table, self.routing_table = {}, {}
        self.is_running = threading.Event()
        self.is_running.set()

    def run(self):
        logging.info(f"{self.device_type} {self.device_name} started.")
        self._discover_neighbors()
        while True:
            self.is_running.wait() # Efficiently waits until event is set
            time.sleep(random.uniform(2, 5))

    def _discover_neighbors(self):
        for neighbor in self.neighbors:
            self.arp_table[neighbor] = f"00:1A:2B:{random.randint(10,99)}:
{random.randint(10,99)}:{random.randint(10,99)}"
        self.routing_table = {n: {'cost': 1} for n in self.neighbors}
        logging.info(f"{self.device_name} discovered neighbors and populated tables.")

    def pause(self): self.is_running.clear(); logging.info(f"Node {self.device_name} paused.")
    def resume(self): self.is_running.set(); logging.info(f"Node {self.device_name} resumed.")
    def update_link_status(self, neighbor, status):
        if status == 'down' and neighbor in self.routing_table:
            del self.routing_table[neighbor]
            logging.warning(f"{self.device_name} link to {neighbor} down.")
        elif status == 'up' and neighbor not in self.routing_table:
            self.routing_table[neighbor] = {'cost': 1}
            logging.info(f"{self.device_name} link to {neighbor} restored.")

class SimulationEngine:
    def __init__(self, graph):
        self.graph = graph
        self.devices = {}

    def start_simulation(self):
        for name in self.graph.nodes():
            attrs = self.graph.nodes[name]
            if attrs.get('type') in ['Router', 'Switch']:
                dev = DeviceNode(name, attrs.get('type'), list(self.graph.neighbors(name)))
                self.devices[name] = dev
                dev.start()
        print("\n✅ Simulation engine started.")

    def run_day1_scenario(self, stabilization_time=5):
        print(f"\n--- Running Day-1: Network Stabilization ({stabilization_time}s) ---")
        time.sleep(stabilization_time)
        print("✅ Day-1 stabilization complete.")

    def run_day2_fault_injection(self):
        print("\n--- Running Day-2: Fault Injection ---")
        active_edges = [(u, v) for u, v in self.graph.edges() if u in self.devices and v in
self.devices]
        if not active_edges: return
        u, v = random.choice(active_edges)
        print(f"\nInjecting link failure: {u} <-> {v}")
        self.devices[u].update_link_status(v, 'down')
        self.devices[v].update_link_status(u, 'down')
        temp_graph = self.graph.copy(); temp_graph.remove_edge(u, v)
        if nx.is_connected(temp_graph): logging.info(f"✅ Network maintained connectivity.")
        else: logging.error(f"❌ Network partitioned.")
        time.sleep(3)
        print(f"Restoring link: {u} <-> {v}")
        self.devices[u].update_link_status(v, 'up')
        self.devices[v].update_link_status(u, 'up')

    def pause_and_resume(self):
        print("\n--- Pausing simulation for 3 seconds... ---")
        for dev in self.devices.values(): dev.pause()
        time.sleep(3)
        print("--- Resuming simulation... ---")
        for dev in self.devices.values(): dev.resume()

    def stop_simulation(self): print("\n--- Simulation concluded. ---")
```
—-------------------------—-------------------------—-------------------------—-------------------------

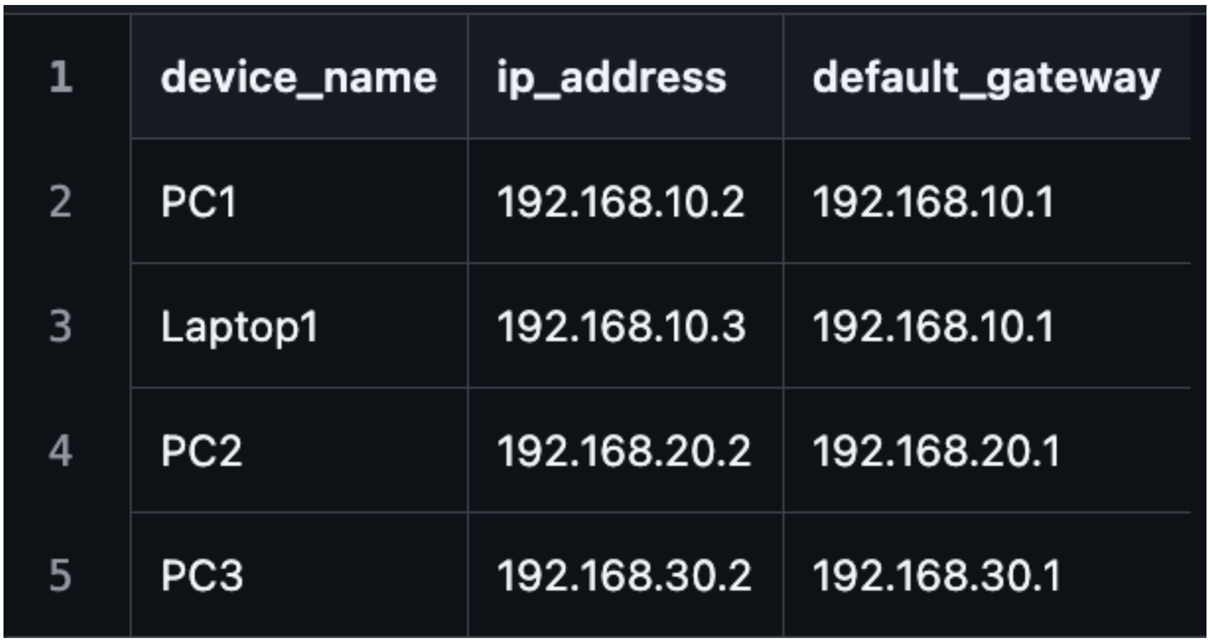
Source Code: src/reporter.py

```
# src/reporter.py

import json
import os
from datetime import datetime

class ReportGenerator:
    """
    Generates a comprehensive JSON report from the analysis data.
    """
    def __init__(self, parsed_data, validation_results, output_dir='reports'):
        self.parsed_data = parsed_data
        self.validation_results = validation_results
        self.output_dir = output_dir

    def generate_json_report(self):
        print("\nStep 5: Generating comprehensive analysis report...")
        os.makedirs(self.output_dir, exist_ok=True)
        report = {
            "report_metadata": {
                "timestamp": datetime.now().isoformat(),
                "title": "Comprehensive Network Analysis Report"
            },
            "parsed_configurations": self.parsed_data,
            "validation_summary": self._format_validation_summary()
        }
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = os.path.join(self.output_dir, f'comprehensive_analysis_{timestamp}.json')
        try:
            with open(filename, 'w') as f:
                json.dump(report, f, indent=4)
            print(f"✅ Comprehensive JSON report saved to '{filename}'")
        except Exception as e:
            print(f"❌ Error writing JSON report: {e}")

    def _format_validation_summary(self):
        summary = {}
        for check, issues in self.validation_results.items():
            summary[check] = {
                "status": "issues_found" if issues else "no_issues",
                "count": len(issues),
                "details": issues
            }
        return summary
```

——

OUTPUT FILES:

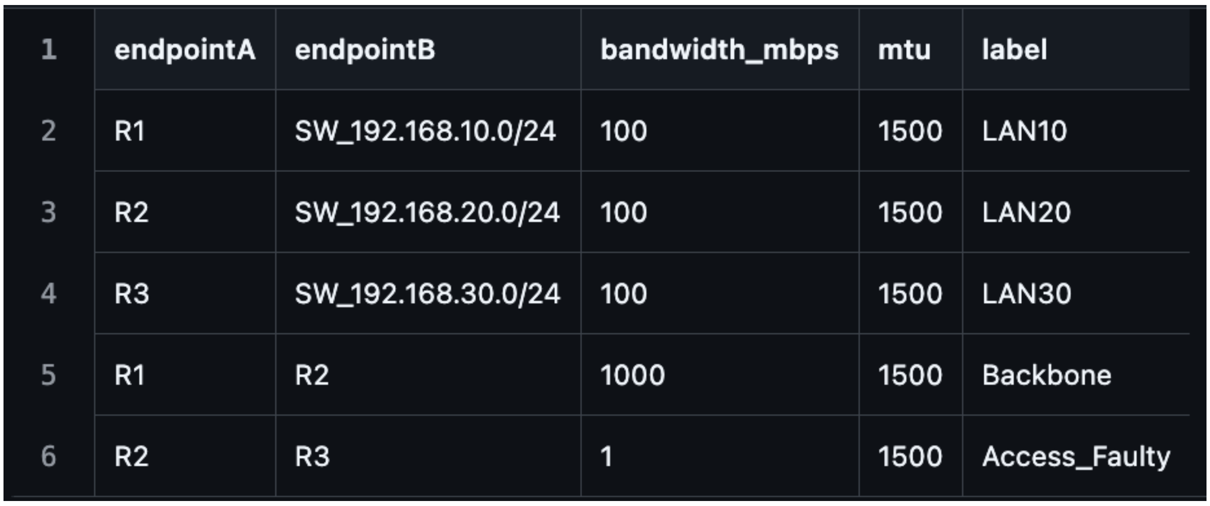
ENDPOINTS.CSC



	device_name	ip_address	default_gateway
2	PC1	192.168.10.2	192.168.10.1
3	Laptop1	192.168.10.3	192.168.10.1
4	PC2	192.168.20.2	192.168.20.1
5	PC3	192.168.30.2	192.168.30.1

—-------------------------—-----------------------—-----------------------—-------------------------

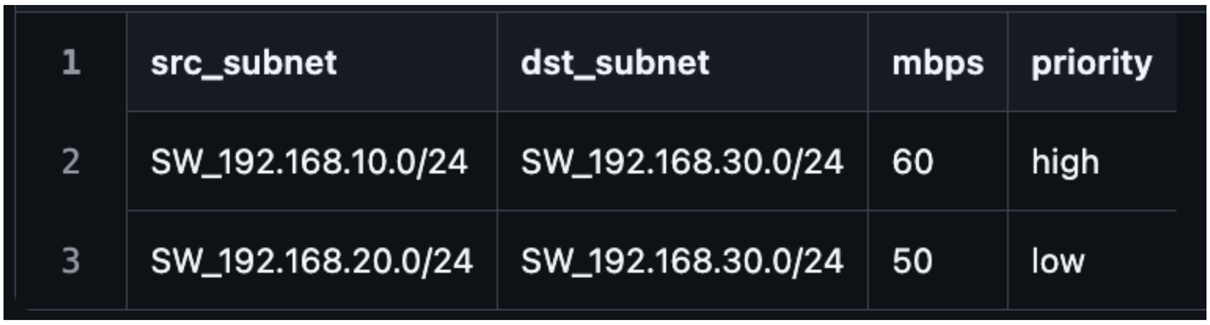
LINKS.CSV:



	endpointA	endpointB	bandwidth_mbps	mtu	label
2	R1	SW_192.168.10.0/24	100	1500	LAN10
3	R2	SW_192.168.20.0/24	100	1500	LAN20
4	R3	SW_192.168.30.0/24	100	1500	LAN30
5	R1	R2	1000	1500	Backbone
6	R2	R3	1	1500	Access_Faulty

—-------------------------—-----------------------—-----------------------—-------------------------

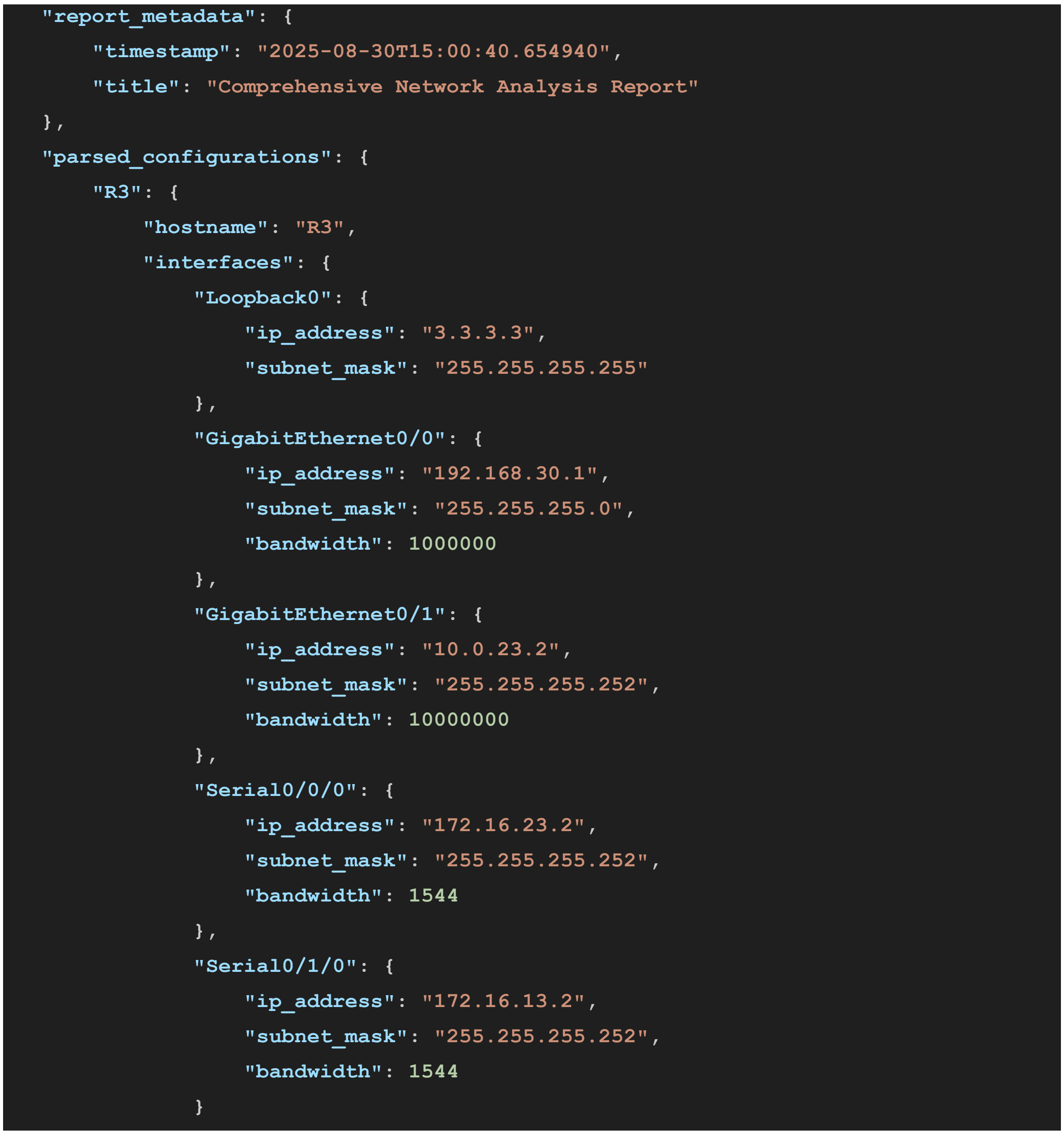
NETWORKDEMANDS.CSV:



1	src_subnet	dst_subnet	mbps	priority
2	SW_192.168.10.0/24	SW_192.168.30.0/24	60	high
3	SW_192.168.20.0/24	SW_192.168.30.0/24	50	low

——

JSON File:



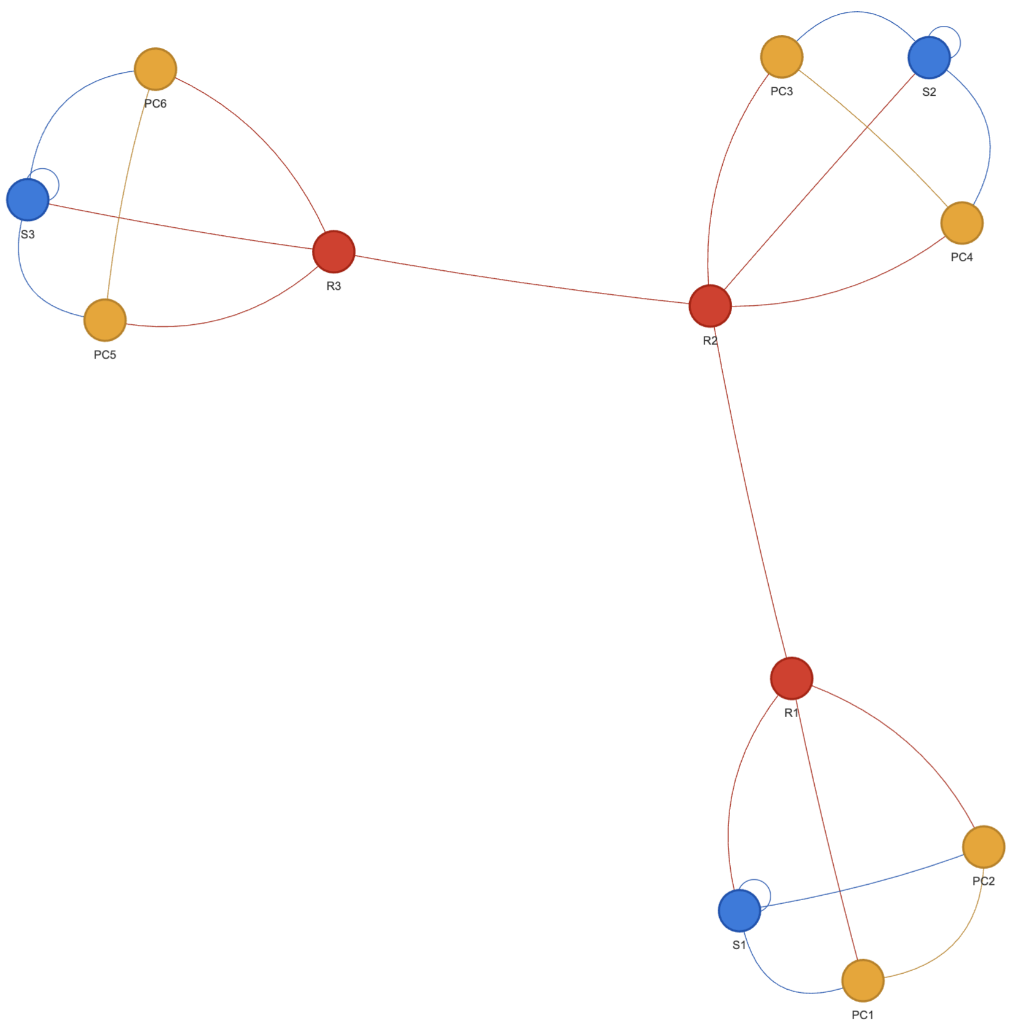
```
  "report_metadata": {
      "timestamp": "2025-08-30T15:00:40.654940",
      "title": "Comprehensive Network Analysis Report"
  },
  "parsed_configurations": {
      "R3": {
          "hostname": "R3",
          "interfaces": {
              "Loopback0": {
                  "ip_address": "3.3.3.3",
                  "subnet_mask": "255.255.255.255"
              },
              "GigabitEthernet0/0": {
                  "ip_address": "192.168.30.1",
                  "subnet_mask": "255.255.255.0",
                  "bandwidth": 1000000
              },
              "GigabitEthernet0/1": {
                  "ip_address": "10.0.23.2",
                  "subnet_mask": "255.255.255.252",
                  "bandwidth": 10000000
              },
              "Serial0/0/0": {
                  "ip_address": "172.16.23.2",
                  "subnet_mask": "255.255.255.252",
                  "bandwidth": 1544
              },
              "Serial0/1/0": {
                  "ip_address": "172.16.13.2",
                  "subnet_mask": "255.255.255.252",
                  "bandwidth": 1544
              }
```

```
            },
            "ospf": {
                "process_id": 1,
                "networks": [
                    {
                        "network": "3.3.3.3",
                        "wildcard": "0.0.0.0",
                        "area": 0
                    },
                    {
                        "network": "10.0.23.0",
                        "wildcard": "0.0.0.3",
                        "area": 0
                    },
                    {
                        "network": "172.16.23.0",
                        "wildcard": "0.0.0.3",
                        "area": 0
                    },
                    {
                        "network": "172.16.13.0",
                        "wildcard": "0.0.0.3",
                        "area": 0
                    }
                ]
            },
            "bgp": null
        },
        "R2": {
            "hostname": "R2",
            "interfaces": {
                "Loopback0": {
                    "ip_address": "2.2.2.2",
                    "subnet_mask": "255.255.255.255"
                },
                "GigabitEthernet0/0": {
                    "ip_address": "192.168.20.1",
                    "subnet_mask": "255.255.255.0",
                    "bandwidth": 1000000
                },
                "GigabitEthernet0/1": {
                    "ip_address": "10.0.12.2",
                    "subnet_mask": "255.255.255.252",
                    "bandwidth": 10000000
                },
                "GigabitEthernet0/2": {
```

```json
                "ip_address": "10.0.23.1",
                "subnet_mask": "255.255.255.252",
                "bandwidth": 10000000
            },
            "Serial0/0/0": {
                "ip_address": "172.16.12.2",
                "subnet_mask": "255.255.255.252",
                "bandwidth": 1544
            },
            "Serial0/0/1": {
                "ip_address": "172.16.23.1",
                "subnet_mask": "255.255.255.252",
                "bandwidth": 1544
            }
        },
        "ospf": {
            "process_id": 1,
            "networks": [
                {
                    "network": "2.2.2.2",
                    "wildcard": "0.0.0.0",
                    "area": 0
                },
                {
                    "network": "10.0.12.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                },
                {
                    "network": "10.0.23.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                },
                {
                    "network": "172.16.12.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                },
                {
                    "network": "172.16.23.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                }
            ]
        },
        "bgp": null
```

```
    },
    "R1": {
        "hostname": "R1",
        "interfaces": {
            "Loopback0": {
                "ip_address": "1.1.1.1",
                "subnet_mask": "255.255.255.255"
            },
            "GigabitEthernet0/0": {
                "ip_address": "192.168.10.1",
                "subnet_mask": "255.255.255.0",
                "bandwidth": 1000000
            },
            "GigabitEthernet0/1": {
                "ip_address": "10.0.12.1",
                "subnet_mask": "255.255.255.252",
                "bandwidth": 10000000
            },
            "Serial0/0/0": {
                "ip_address": "172.16.12.1",
                "subnet_mask": "255.255.255.252",
                "bandwidth": 1544
            }
        },
        "ospf": {
            "process_id": 1,
            "networks": [
                {
                    "network": "1.1.1.1",
                    "wildcard": "0.0.0.0",
                    "area": 0
                },
                {
                    "network": "10.0.12.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                },
                {
                    "network": "172.16.12.0",
                    "wildcard": "0.0.0.3",
                    "area": 0
                }
            ]
        },
        "bgp": null
    },
        "S1": {
            "hostname": "S1",
            "interfaces": {
                "Vlan1": {
                    "ip_address": "192.168.10.254",
                    "subnet_mask": "255.255.255.0"
                },
                "Vlan10": {
                    "ip_address": "192.168.10.253",
                    "subnet_mask": "255.255.255.0"
                },
                "GigabitEthernet0/1": {
                    "bandwidth": 1000000
                },
                "GigabitEthernet0/2": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/3": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/4": {
                    "bandwidth": 100000,
                    "vlan": 20
                }
            },
            "ospf": null,
            "bgp": null
        },
        "S2": {
            "hostname": "S2",
            "interfaces": {
                "Vlan1": {
                    "ip_address": "192.168.20.254",
                    "subnet_mask": "255.255.255.0"
                },
                "Vlan10": {
                    "ip_address": "192.168.20.253",
                    "subnet_mask": "255.255.255.0"
                },
                "GigabitEthernet0/1": {
                    "bandwidth": 1000000
                },
                "GigabitEthernet0/2": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/3": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/4": {
                    "bandwidth": 100000,
                    "vlan": 20
                }
            },
            "ospf": null,
            "bgp": null
        },
        "S3": {
            "hostname": "S3",
            "interfaces": {
                "Vlan1": {
                    "ip_address": "192.168.30.254",
                    "subnet_mask": "255.255.255.0"
                },
                "Vlan10": {
                    "ip_address": "192.168.30.253",
                    "subnet_mask": "255.255.255.0"
                },
                "GigabitEthernet0/1": {
                    "bandwidth": 1000000
                },
                "GigabitEthernet0/2": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/3": {
                    "bandwidth": 100000,
                    "vlan": 10
                },
                "GigabitEthernet0/4": {
                    "bandwidth": 100000,
                    "vlan": 20
                }
            },
            "ospf": null,
            "bgp": null
        },
        "PC6": {
            "hostname": "PC6",
        "interfaces": {
            "FastEthernet0/0": {
                "ip_address": "192.168.30.11",
                "subnet_mask": "255.255.255.0",
                "bandwidth": 100000
            },
            "FastEthernet0/1": {
                "bandwidth": 100000
            }
        },
        "ospf": null,
        "bgp": null
    },
    "PC5": {
        "hostname": "PC5",
        "interfaces": {
            "FastEthernet0/0": {
                "ip_address": "192.168.30.10",
                "subnet_mask": "255.255.255.0",
                "bandwidth": 100000
            },
            "FastEthernet0/1": {
                "bandwidth": 100000
            }
        },
        "ospf": null,
        "bgp": null
    },
    "PC4": {
        "hostname": "PC4",
        "interfaces": {
            "FastEthernet0/0": {
                "ip_address": "192.168.20.11",
                "subnet_mask": "255.255.255.0",
                "bandwidth": 100000
            },
            "FastEthernet0/1": {
                "bandwidth": 100000
            }
        },
        "ospf": null,
        "bgp": null
    },
    "PC1": {
        "hostname": "PC1",
        "interfaces": {
                "FastEthernet0/0": {
                    "ip_address": "192.168.10.10",
                    "subnet_mask": "255.255.255.0",
                    "bandwidth": 100000
                },
                "FastEthernet0/1": {
                    "bandwidth": 100000
                }
            },
            "ospf": null,
            "bgp": null
        },
        "PC3": {
            "hostname": "PC3",
            "interfaces": {
                "FastEthernet0/0": {
                    "ip_address": "192.168.20.10",
                    "subnet_mask": "255.255.255.0",
                    "bandwidth": 100000
                },
                "FastEthernet0/1": {
                    "bandwidth": 100000
                }
            },
            "ospf": null,
            "bgp": null
        },
        "PC2": {
            "hostname": "PC2",
            "interfaces": {
                "FastEthernet0/0": {
                    "ip_address": "192.168.10.11",
                    "subnet_mask": "255.255.255.0",
                    "bandwidth": 100000
                },
                "FastEthernet0/1": {
                    "bandwidth": 100000
                }
            },
            "ospf": null,
            "bgp": null
        }
    },
    "validation_summary": {
        "duplicate_ips": {
            "status": "no_issues",
           "count": 0,
           "details": []
       },
       "network_loops": {
           "status": "issues_found",
           "count": 9,
           "details": [
               "Potential network loop detected involving: R1 -> PC1 -> PC2",
               "Potential network loop detected involving: S1 -> PC1 -> PC2",
               "Potential network loop detected involving: R1 -> S1 -> PC2",
               "Potential network loop detected involving: S2 -> PC3 -> R2",
               "Potential network loop detected involving: PC4 -> PC3 -> R2",
               "Potential network loop detected involving: S2 -> PC4 -> R2",
               "Potential network loop detected involving: S3 -> PC5 -> R3",
               "Potential network loop detected involving: PC6 -> PC5 -> R3",
               "Potential network loop detected involving: S3 -> PC6 -> R3"
           ]
       },
       "load_analysis": {
           "status": "no_issues",
           "count": 0,
           "details": []
       },
       "load_balancing_recommendations": {
           "status": "no_issues",
           "count": 0,
           "details": []
       },
       "missing_components": {
           "status": "no_issues",
           "count": 0,
           "details": []
       },
              "gateway_issues": {
           "status": "no_issues",
           "count": 0,
           "details": []
       },
       "mtu_mismatches": {
           "status": "no_issues",
           "count": 0,
           "details": []
       }
   }
}
```

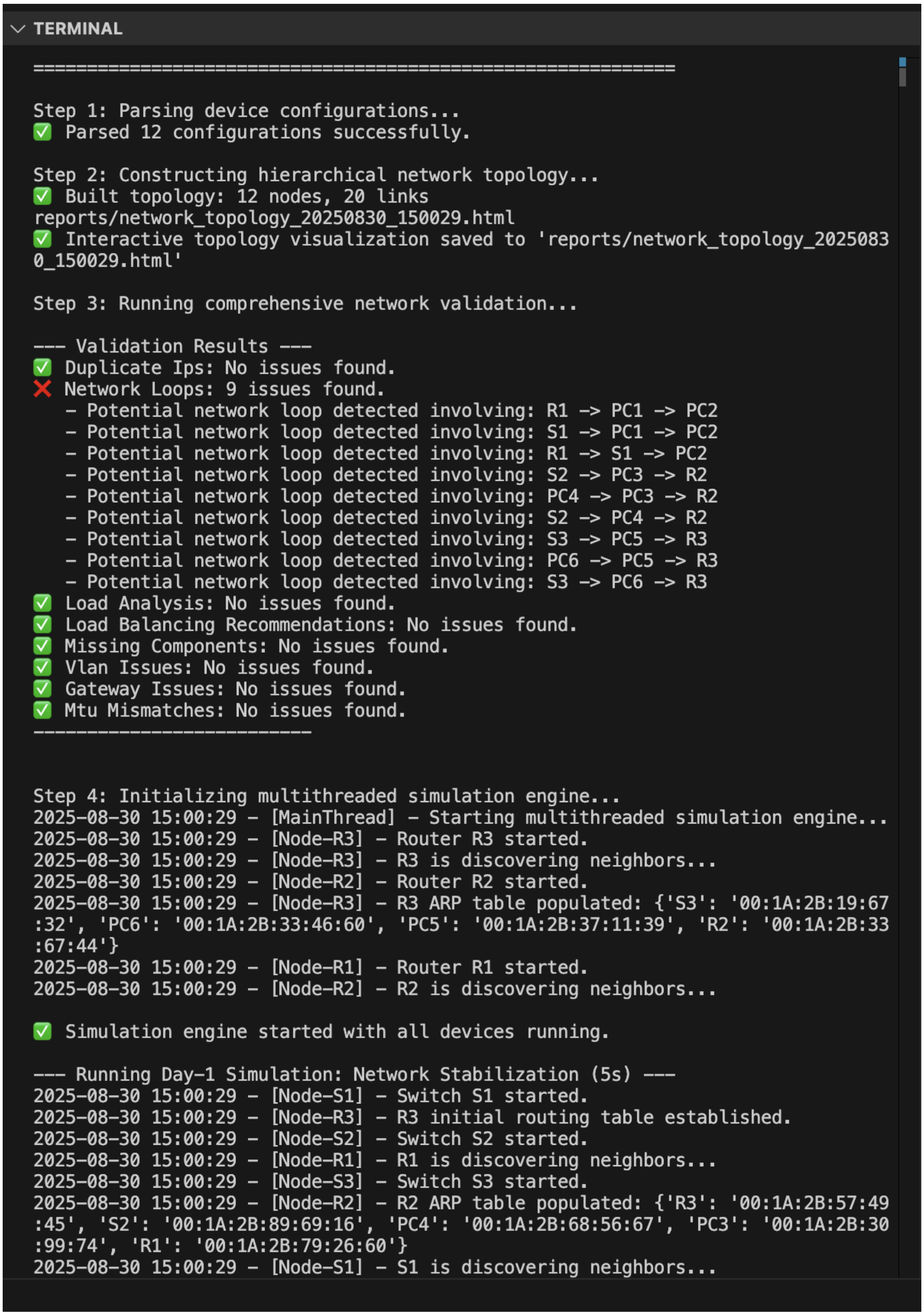
TOPOLOGY.png :



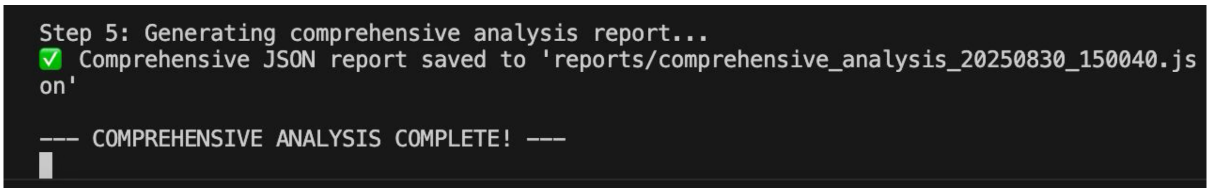
This Is an Example Topology I created with my own Configuration FIle which I posted in my Github.

—-------------------------—-----------------------—-----------------------—------------------------

VS-Code_TERMINAL_O/P:

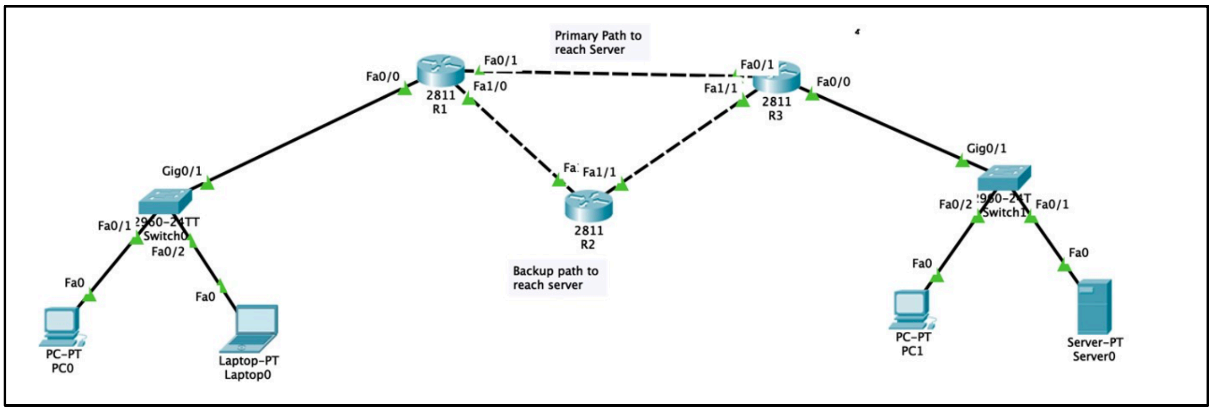

TERMINAL

```
============================================================

Step 1: Parsing device configurations...
✅ Parsed 12 configurations successfully.

Step 2: Constructing hierarchical network topology...
✅ Built topology: 12 nodes, 20 links
reports/network_topology_20250830_150029.html
✅ Interactive topology visualization saved to 'reports/network_topology_2025083
0_150029.html'

Step 3: Running comprehensive network validation...

--- Validation Results ---
✅ Duplicate Ips: No issues found.
❌ Network Loops: 9 issues found.
   - Potential network loop detected involving: R1 -> PC1 -> PC2
   - Potential network loop detected involving: S1 -> PC1 -> PC2
   - Potential network loop detected involving: R1 -> S1 -> PC2
   - Potential network loop detected involving: S2 -> PC3 -> R2
   - Potential network loop detected involving: PC4 -> PC3 -> R2
   - Potential network loop detected involving: S2 -> PC4 -> R2
   - Potential network loop detected involving: S3 -> PC5 -> R3
   - Potential network loop detected involving: PC6 -> PC5 -> R3
   - Potential network loop detected involving: S3 -> PC6 -> R3
✅ Load Analysis: No issues found.
✅ Load Balancing Recommendations: No issues found.
✅ Missing Components: No issues found.
✅ Vlan Issues: No issues found.
✅ Gateway Issues: No issues found.
✅ Mtu Mismatches: No issues found.
--------------------------



Step 4: Initializing multithreaded simulation engine...
2025-08-30 15:00:29 - [MainThread] - Starting multithreaded simulation engine...
2025-08-30 15:00:29 - [Node-R3] - Router R3 started.
2025-08-30 15:00:29 - [Node-R3] - R3 is discovering neighbors...
2025-08-30 15:00:29 - [Node-R2] - Router R2 started.
2025-08-30 15:00:29 - [Node-R3] - R3 ARP table populated: {'S3': '00:1A:2B:19:67
:32', 'PC6': '00:1A:2B:33:46:60', 'PC5': '00:1A:2B:37:11:39', 'R2': '00:1A:2B:33
:67:44'}
2025-08-30 15:00:29 - [Node-R1] - Router R1 started.
2025-08-30 15:00:29 - [Node-R2] - R2 is discovering neighbors...

✅ Simulation engine started with all devices running.

--- Running Day-1 Simulation: Network Stabilization (5s) ---
2025-08-30 15:00:29 - [Node-S1] - Switch S1 started.
2025-08-30 15:00:29 - [Node-R3] - R3 initial routing table established.
2025-08-30 15:00:29 - [Node-S2] - Switch S2 started.
2025-08-30 15:00:29 - [Node-R1] - R1 is discovering neighbors...
2025-08-30 15:00:29 - [Node-S3] - Switch S3 started.
2025-08-30 15:00:29 - [Node-R2] - R2 ARP table populated: {'R3': '00:1A:2B:57:49
:45', 'S2': '00:1A:2B:89:69:16', 'PC4': '00:1A:2B:68:56:67', 'PC3': '00:1A:2B:30
:99:74', 'R1': '00:1A:2B:79:26:60'}
2025-08-30 15:00:29 - [Node-S1] - S1 is discovering neighbors...
```



```
Step 5: Generating comprehensive analysis report...
✅ Comprehensive JSON report saved to 'reports/comprehensive_analysis_20250830_150040.js
on'

--- COMPREHENSIVE ANALYSIS COMPLETE! ---
█
```

The simulated Network looks like:



Primary Path to reach Server
Fa0/0
Fa0/1
Fa1/0
2811 R1
Fa0/1
Fa1/1
2811 R3
Fa0/0
Gig0/1
2960-24TT Switch0
Fa0/1
Fa0/2
Fa0
Fa0
PC-PT PC0
Laptop-PT Laptop0
Fa
Fa1/1
2811 R2
Backup path to reach server
Gig0/1
2960-24TT Switch1
Fa0/2
Fa0/1
Fa0
Fa0
PC-PT PC1
Server-PT Server0