Project Title: Scripting from Packet Tracer Controller using REST APIs

Group members: Siddharth Pradipbhai Dhanak(119845220), Oluwadamilola Ogundipe(120759221),Mariam Odutayo(140164237)

Course: Software Defined Networks

Course code: DCN420

Course Section: NAA

Professor: Andres Lombo

Date: April 11, 2025

Institution: Seneca College of Applied Arts and Technology

Project Overview:

This project simulates how modern network controllers (like Cisco DNA Center or SDN platforms) interact with network devices using REST APIs. Since Cisco Packet Tracer does not support external REST API integration directly, our project simulates that functionality using Python scripts and mock endpoints. Each script represents a real-world controller task such as ticket handling, device inventory, or network configuration updates.

Key Functional Goals

- Simulate API-based automation as seen in real SDN controllers
- Build individual Python scripts to mimic network operations
- Apply Agile methodology using MoSCoW technique to prioritize features
- Ensure all code is fully commented and modular
- Demonstrate realistic outputs using simulated data

Tools & Technologies Used

- Python 3.13.3 (Programming Language)
- Requests library (to simulate HTTP REST APIs)
- Static data structures & mock APIs (e.g., JSONPlaceholder)
- Text Editors:
    - Dammy: Replit (Online IDE for scripting and collaboration)
    - Mariam: Visual Studio Code

- Siddharth: Visual Studio Code

Development approach:

We applied the MoSCoW methodology to prioritize project features. Must-have features included modular Python scripts that simulate controller tasks using RESTful API operations like GET, POST, and PUT. These were essential to demonstrate ticket handling, inventory retrieval, flow analysis, and configuration updates.

Should-have features included dynamic input handling, such as IP address detection and output formatting to mimic SDN controller behavior. Could-have features, like user-driven flow hop definitions and enhanced error handling, were partially implemented for demonstration. Won't-have items, such as live integration with Cisco Packet Tracer APIs, were excluded due to tool limitations.

Deliverables:

Our submission includes five fully commented Python scripts, a PowerPoint presentation with outputs and tool descriptions, a written project report, and a live presentation script — all packaged in a properly named .zip folder.

Procedures:

1. Ticket Handling Script – Developed by Oluwadamilola Ogundipe

- Simulates the lifecycle of a support ticket (create, view, update)
- Used POST and PUT operations on mock REST API
- Demonstrates how controllers can automate service desk workflows
    - File: ticket_handeling.py

2. Inventory Management Script – Developed by Oluwadamilola Ogundipe

- Simulates fetching network device inventory using a GET request
- Parses JSON to show hostname, device ID, company, and status
- Emulates real-time controller inventory views
    - File: inventory_management.py

3. Flow Analysis Script – Developed by Siddharth Pradipbhai Dhanak

- Simulates a traceroute-like flow from source to destination
- Displays device hops with interfaces
- Used static path representation for flow visibility
    - File: flow_analysis.py

4. Network-Wide Settings Update – Developed by Siddharth

- Simulates pushing NTP and SYSLOG settings across the network
- Uses socket module to dynamically detect host IP
- Demonstrates controller-based configuration updates
    - File: update_network-wide settings.py

5. Device & Host Configuration – Developed by Mariam Odutayo

- Displays running configurations of simulated devices (switches and routers)
- Shows host IPs, MAC addresses, and their connected devices

- Mimics controller-level topology and device insights
    - File: device_host_config.py

Pseudocode for scripts:

1.Ticket handling:

- Import the requests library (used to send HTTP requests)
- Set the fake API URL that we will use to simulate the ticket system
- Define a function called create_ticket:
    - Takes an issue description as input
    - Creates a dictionary with a title, issue text, and user ID
    - Sends this as a POST request to the fake API
    - Returns the response from the API (simulated ticket info)

- Define a function called list_tickets:
    - Takes the ticket we created
    - Returns it inside a list (just to act like we're listing multiple tickets)

- Define a function called update_ticket:
    - Takes a ticket ID and a new status (like "closed")
    - Prints a confirmation that the ticket was updated
    - Returns a fake update result with ticket ID and status

- In the main program (only runs if this file is executed directly): a. Print “Creating ticket…” b. Call the create_ticket function with a sample issue (e.g., "Switch down in Lab A") c. Print the response (ticket info)
- Print “Listing ticket…” e. Use list_tickets to simulate listing the ticket f. Print ticket ID and title from the returned list
- Print “Updating ticket…” h. Call update_ticket to fake a ticket update i. Print the update confirmation

2.Inventory Management:

- Import the requests module so we can make web requests to an online API.
- Set the URL for a fake API that returns a list of “users.”
  (We’re pretending these users are network devices.)
- Define a function called get_inventory: a. Send a GET request to the fake API to get the data. b. Convert the response into a list of device data. c. For each device in the list:
  - Print its ID
  - Print its username (used as a hostname)
  - Print “Device Type: Router” (hardcoded)
  - Print “Status: Active” (also hardcoded)
  - Print a line separator
- In the main part of the script: a. Print “Fetching mock network inventory…” to show what’s happening b. Call the get_inventory function to display all the device info

3.Flow Analysis:

Define a function called trace_flow that takes two inputs:

- src_ip: the source IP address (where data starts)
- dst_ip: the destination IP address (where data ends)

- Print a message showing the flow is being traced from the source to the destination.
- Create a list called mock_hops to simulate the devices the data travels through:
  - The first device is a router ("R1") using the interface "Gig0/0"
  - The second is a switch ("SW1") using the interface "Gig0/1"
  - The third is a firewall using the interface "Eth0"
  - The final hop is a server using the interface "Eth1"
- Go through each hop in the list and:
  - Print the device name and the interface it's using
- At the bottom, check if the script is being run directly.
  - If yes, call the trace_flow function with a sample source IP and destination IP.

4. Device and Host Cobnfiguration

Define a function called get_device_configurations:

- Create a list with two mock devices:
  - One is a switch (SW1) with model 2960 and a simulated IP config

    - The other is a router (R1) with model 4321 and a different IP config
- For each device in the list:
    - Print the device's hostname
    - Print its model number
    - Print the simulated configuration
    - Print a line separator for clarity

Define a second function called get_host_details:

- Create a list with two simulated hosts:
    - Each host has an IP address, MAC address, and the device it connects to
- For each host in the list:
    - Print the IP address
    - Print the MAC address
    - Print which device it is connected to
    - Print a line separator

In the main part of the script:

- Call the get_device_configurations function to display device info
- Print a blank line to separate the output
- Call the get_host_details function to display host info

5. Network wide setting Update Script:

Define a function called update_settings that takes two inputs:

- ntp_ip: the IP address of the NTP (time) server
- syslog_ip: the IP address of the SYSLOG (logging) server

Inside the function:

- Print a message saying it's sending configuration to the network
- Print the IP address of the NTP server
- Print the IP address of the SYSLOG server
- Print a message saying the configuration was successful (simulated)

In the main part of the script:

- Call the update_settings function with example IP addresses:
    - 172.16.10.10 for the NTP server
    - 172.16.20.20 for the SYSLOG server

Troubleshooting:

During testing, we initially attempted to simulate real IP addresses for tasks such as ticket handling and network-wide configuration updates. However, because Cisco Packet Tracer does not support external REST API integration, and due to restrictions in live networking environments, these real IPs did not respond as expected. To overcome this, we used fake or placeholder IP addresses (e.g., 172.16.x.x) and mock API services like JSONPlaceholder to simulate realistic behavior. This allowed us to demonstrate the controller logic effectively without needing a live backend or real devices.

Presentation Roles:

For the final presentation and in-class demo of our DCN420 scripting project, each team member was assigned a specific script to explain and demonstrate. Although the project was developed collaboratively, the breakdown of responsibilities during the presentation was as follows:

- Oluwadamilola Ogundipe:
    - Delivered the project introduction and overview
    - Presented Script 1 (Ticket Handling) in detail
    - Demonstrated ticket creation, listing, and update via a mock API
- Mariam Odutayo:
    - Explained and demoed Script 2 (Inventory Management)
    - Followed up with Script 3 (Flow Analysis) to show traffic path tracing
    - Focused on real-world applications like asset tracking and traffic mapping
- Siddharth Pradipbhai Dhanak:
    - Covered Script 4 (Update Network-Wide Settings), explaining dynamic IP detection and configuration push
    - Presented Script 5 (Device & Host Configuration), highlighting how network topology and device data are retrieved

This division allowed each team member to focus on specific functionality and provide a hands-on walkthrough of their assigned scripts. We ran live simulations using Replit and ensured all output was clearly explained to reflect real-world network controller tasks.

Conclusion:

In this project, we effectively used the MoSCoW methodology to prioritize and manage our development tasks. We focused first on Must-have features like ticket handling, inventory display, and configuration scripts — all of which are core to SDN automation. Should-have features, such as dynamic IP detection and flow analysis, enhanced realism. Could-have features like deeper error handling were noted for future improvement, while Won't-have items (like full Packet Tracer API integration) were acknowledged as limitations of the platform. By applying MoSCoW, we kept our project structured, efficient, and aligned with real-world network management scenarios.