



Network Management & Monitoring Tool

by

Ghazal Fayaztorshizi

A Bachelor' s Thesis

Advisor: Dr. Mojtaba Asgari

Islamic Azad University – Mashhad Branch

Winter 2024

Table of Contents

1. Introduction

1.1 Background and Motivation

1.2 Objectives

1.3 Significance of the Study

2. Phase 2

2.1 Installation and Setup

2.1.1 Prerequisites

2.1.2 Project Files

3. Features (Phases)

3.1 Phase 1: Network Configuration

3.1.1 Changing DNS

3.1.2 Hostname Change

3.1.3 Determination of Static IP for the Interface

3.1.4 Using DHCP to Get IP

3.1.5 Adding Temporary and Permanent Routes

3.1.6 Deleting Temporary and Permanent Routes

3.1.7 Key Tools in Phase 1

3.2 Phase 2: Nftables Management

3.2.1 Access-Restricting Rules

3.2.2 Creation and Management of NAT Rules

[3.2.3 Templates for Automation](#)

[3.2.4 Key Tools in Phase 2](#)

[3.3 Phase 3: Open vSwitch \(OVS\) Management](#)

[3.3.1 Adding and Deleting OVS Bridges](#)

[3.3.2 Adding and Removing Ports from OVS Bridges](#)

[3.3.3 Turning Ports On and Off](#)

[3.3.4 Setting Ports as Trunk or Access](#)

[3.3.5 IP Configuration for VLAN Interfaces](#)

[3.3.6 Key Tools in Phase 3](#)

[3.4 Phase 4: Network Monitoring](#)

[3.4.1 Gathering Network Interface Information](#)

[3.4.2 Bandwidth Display](#)

[3.4.3 Statistics Related to Network Protocols \(TCP/UDP\)](#)

[3.4.4 Monitoring Incoming and Outgoing Traffic](#)

[3.4.5 Key Tools in Phase 4](#)

[4. Results and Discussion](#)

[4.1 Challenges and Troubleshooting](#)

5. Conclusion

5.1 Summary of Findings

5.2 Achievement of Objectives

5.3 Recommendations for Future Work

Introduction

1.1 Background and Motivation

Simulating real-world networks often requires significant hardware resources, making it challenging to explore complex configurations or troubleshoot advanced scenarios. To address this limitation, virtualization emerges as an ideal solution—allowing us to create local networks entirely within virtual environments such as VMware. By doing so, we can implement routers, clients, and servers without the usual costs and logistical hurdles of physical equipment.

However, understanding the basic structure of a virtual network is only the first step. As networks continue to grow rapidly in both size and complexity, the question arises: **How can we efficiently manage numerous changes and settings without manual, error-prone procedures?** This leads to the concept of **network automation**, which not only saves time but also significantly reduces human error and enhances scalability.

Note: This project constitutes **Phase 2** of a broader initiative. In **Phase 1**, a virtual network was implemented, focusing on simulating LAN environments, deploying an Ubuntu-based router, and setting up services like DHCP, DNS, and NAT. **Phase 2** (the focus here) extends this foundation by simplifying the configuration and monitoring of Linux networks through a **Text User Interface (TUI)**. It is designed in four **phases**, each tackling a specific aspect of network administration:

1. **Phase 1** – Network Configuration
2. **Phase 2** – Firewall & NAT (nftables) Management
3. **Phase 3** – Open vSwitch (OVS) Management
4. **Phase 4** – Network Monitoring

Instead of manually editing files like `/etc/network/interfaces`, `resolv.conf`, or applying complex `nft` and `ovs-vsctl` commands, this TUI-based approach offers an **interactive menu** to handle each operation safely.

1.2 Objectives

- **Provide an Intuitive Interface for Network Administrators**
 - Develop a text-based user interface (TUI) to eliminate cumbersome command-line sequences.
 - Simplify tasks like configuring IP addresses, firewall rules, switch ports, and real-time monitoring.
- **Integrate Multiple Network Tools in One Place**
 - Combine nmcli (for permanent network configurations), nftables (for firewall & NAT rules), and Open vSwitch (for virtual switching) under a single TUI umbrella.
 - **Ensure consistent logging and error handling across these tools.**
- **Implement Real-Time Network Monitoring**
 - Provide real-time bandwidth, interface status, and protocol statistics (TCP/UDP).
 - Enhance situational awareness for network operators, allowing quick troubleshooting of performance issues.
- **Demonstrate the Efficiency Gains of Automation**
 - Compare manual network modifications against automated TUI flows.
 - Show how adopting these automated approaches can reduce errors and optimize daily network operations.

1.3 Significance of the Study

In modern computing environments, the ability to **simulate** and **automate** network configurations holds immense value. By leveraging **virtualization**, we circumvent resource constraints and gain a risk-free sandbox to test sophisticated setups.

As networks evolve and grow, **automation** takes center stage, enabling faster deployments, streamlined updates, and more reliable performance. This project not only deepens our understanding of foundational networking concepts but also provides hands-on experience with best practices in modern network management. Consequently, the study equips practitioners and learners alike with vital skills to keep pace with the demands of large-scale, dynamically changing networks.

Phase2

2.1 Installation and Setup

2.1.1 Prerequisites:

- **Linux OS with Python 3 installed.**
- **Root (sudo) privileges:** Many operations (like changing IPs, setting firewall rules) require root access.
- **Dependencies:**
 - nmcli (usually part of NetworkManager)
 - nft (from nftables)
 - ovs-vsctl (from openvswitch-switch)
 - Python 3 library: curses (commonly available as python3-curses)

2.1.2 Project Files:

- **CompleteCode Script:** A single Python file containing all four phases.
- **Phase-Specific Scripts (Optional):**
phase2-sec1.py, phase2-sec2.py, phase2-sec3.py, phase2-sec4.py to run each module independently.
- **You can clone or download the repository:**
git clone <https://github.com/ghazal-fyzt/Network-Automation-in-Action-A-Virtual-Lab-Approach.git>
- **Run the code with sudo:**
sudo python3 CompleteCode.py

Features (Phases)

This Phase 2 project internally comprises **four sub-phases**—each sub-phase can be run independently or from a unified “Main Menu” approach:

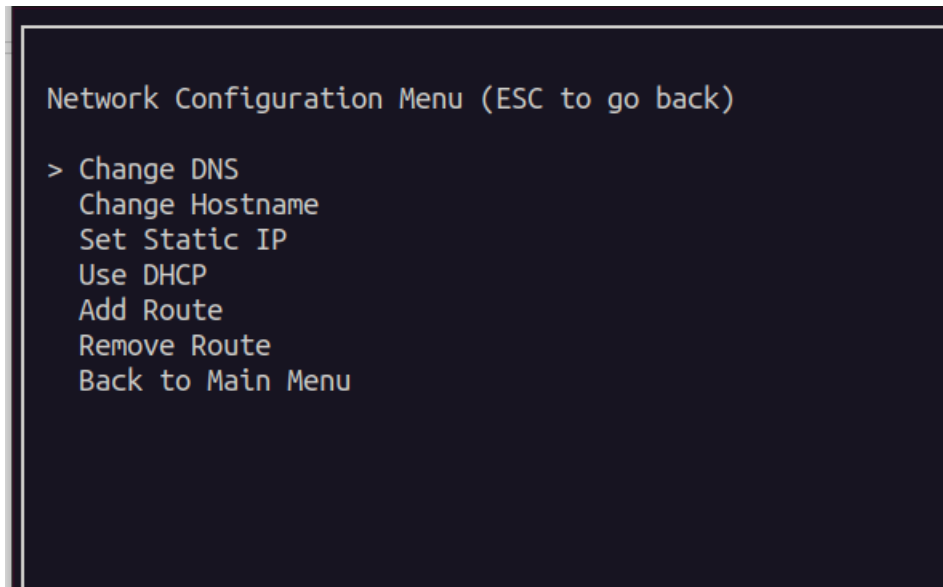
1. **Phase 1: Network Configuration** (Basic config TUI)
2. **Phase 2: Nftables Management** (Firewall & NAT TUI)
3. **Phase 3: Open vSwitch Management** (Virtual switch TUI)
4. **Phase 4: Network Monitoring** (Real-time stats)

```
Main Menu (ESC to exit)

> Network Configuration
  Nftables Management
  Open vSwitch Management
  Network Monitoring
  Exit
```


3.1 Phase 1: Network Configuration

Purpose: Phase 1 focuses on simplifying and automating fundamental network configurations that are commonly performed in any network environment. These include DNS settings, hostname changes, IP address configurations, and route management. The goal is to provide network administrators with a unified TUI (text-based user interface) for making these changes efficiently and with minimal risk of errors.

A screenshot of a terminal window showing a text-based user interface for network configuration. The title is "Network Configuration Menu (ESC to go back)". Below the title, there is a list of options, each preceded by a greater-than sign (>). The options are: "Change DNS", "Change Hostname", "Set Static IP", "Use DHCP", "Add Route", "Remove Route", and "Back to Main Menu".

```
Network Configuration Menu (ESC to go back)

> Change DNS
  Change Hostname
  Set Static IP
  Use DHCP
  Add Route
  Remove Route
  Back to Main Menu
```

Functionalities in Phase 1

1. Changing DNS

- **Requirement:** The user must be able to change the system's DNS servers both temporarily and permanently.
- **Temporary Change:** Utilizes `resolvectl` to update DNS settings for the current session only.
- **Permanent Change:** Uses `nmcli` to update the NetworkManager configuration, ensuring the DNS settings persist across reboots.

Functions Used:

- `change_dns_form(screen)` prompts the user for DNS servers and interface selection.
- `change_dns(interface_name, dns_list, permanent)` applies DNS changes via `resolvectl` or `nmcli`.

Why This Approach?

- nmcli ensures changes persist across reboots and integrates seamlessly with NetworkManager.
- Temporary changes via resolvectl are useful for quick, ad-hoc debugging without impacting long-term settings.

Benefit: Centralized control over DNS configurations reduces manual file editing errors and speeds up troubleshooting.

```
Select Interface (ESC to go back)
```

```
> e2
   ens39
   ens37
   lo
   ovs1
   ens33
   ens38
   ovs-system
```

```
Enter up to 3 DNS Servers (comma-separated):
```

```
(Press ESC or type 'back' to return)
```

```
█
```

```
Apply Change:
```

```
> Temporarily
   Permanently
   Back
```

```
DNS updated successfully!
```

```
Press any key to continue...
```

2. Hostname Change

- **Requirement:** Allow users to update the system's hostname easily.
- **Implementation:** Uses `hostnamectl set-hostname <newname>` to change the hostname.

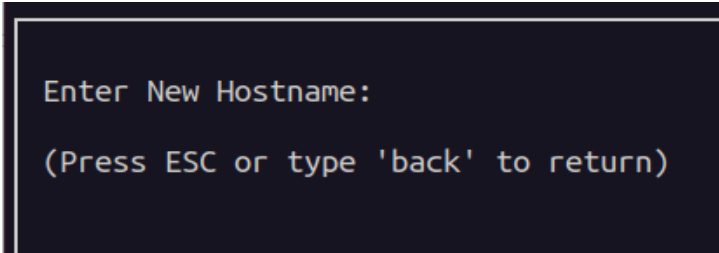
Functions Used:

- `change_hostname_form(screen)` provides a TUI prompt for entering a new hostname.
- `change_hostname(new_hostname)` applies the new hostname using `hostnamectl`.

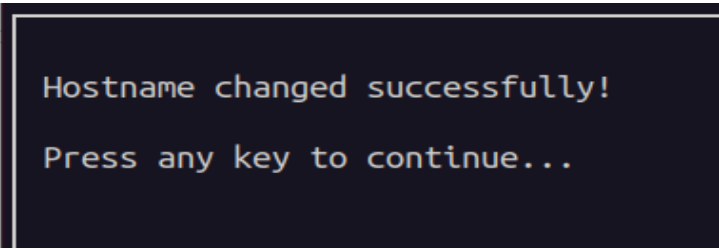
Why This Approach?

- `hostnamectl` integrates with modern Linux systems, ensuring compatibility and avoiding deprecated methods like directly modifying `/etc/hostname`.

Benefit: Enables quick rebranding or identification changes for the system, particularly useful in lab environments.



```
Enter New Hostname:  
(Press ESC or type 'back' to return)
```



```
Hostname changed successfully!  
Press any key to continue...
```

3. Determination of Static IP for the Interface

- **Requirement:** Configure static IP addresses for network interfaces, with options for temporary or permanent application.
- **Temporary Change:** `ip addr add <IP>/<mask> dev <interface>` and `ip route add default via <gateway>`.
- **Permanent Change:** `nmcli connection modify <interface> ipv4.addresses <IP>/<mask>` and `ipv4.method manual`.

Functions Used:

- `set_static_ip_form(screen)` prompts the user for interface, IP address, subnet mask, and gateway.
- `set_static_ip(interface_name, ip_address, subnet_mask, gateway, permanent)` applies the IP configuration.

Why This Approach?

- Temporary changes allow testing configurations without impacting the system permanently.
- Permanent changes via `nmcli` ensure the settings survive reboots and are stored in NetworkManager profiles.

Benefit: Provides a flexible approach for configuring both transient and long-term network setups.

```
Select Interface (ESC to go back)

> e2
  ens39
  ens37
  lo
  ovs1
  ens33
  ens38
  ovs-system
```

Enter IP Address (e.g. 192.168.1.10):

(Press ESC or type 'back' to return)

█

Enter Subnet Mask in CIDR (0-32):

(Press ESC or type 'back' to return)

█

Enter Gateway IP (Optional):

(Press ESC or type 'back' to return)

█

Apply Change:

> Temporarily
 Permanently
 Back

Static IP set successfully!

Press any key to continue...

4. Using DHCP to Get IP

- **Requirement:** Switch a network interface to use DHCP for automatic IP assignment.
- **Implementation:** nmcli connection modify <interface> ipv4.method auto and nmcli connection up <interface>.

Functions Used:

- use_dhcp_form(screen) prompts the user to select an interface and applies DHCP settings.
- use_dhcp(interface_name) automates the configuration process.

Why This Approach?

- Using nmcli ensures smooth interaction with NetworkManager, avoiding conflicts with manual configurations.

Benefit: Simplifies transitioning interfaces to dynamic IP allocation for environments with DHCP servers.

```
Select Interface (ESC to go back)
```

```
> e2
  ens39
  ens37
  lo
  ovs1
  ens33
  ens38
  ovs-system
```

```
DHCP enabled permanently!
```

```
Press any key to continue...
```

5. Adding Temporary and Permanent Routes

- **Requirement:** Enable users to add static routes either temporarily or permanently.
- **Temporary Route:** `ip route add <destination> via <gateway> dev <interface>`.
- **Permanent Route:** `nmcli connection modify <interface> +ipv4.routes <destination> <gateway>`.

Functions Used:

- `add_route_form(screen)` provides a TUI interface to input route parameters.
- `add_route_temporary(interface_name, destination_cidr, gateway)` handles temporary routes.
- `add_route_permanent(interface_name, destination_cidr, gateway)` applies persistent routes.

Why This Approach?

- `nmcli` allows permanent route configuration in NetworkManager profiles, ensuring the settings are retained across reboots.

Benefit: Streamlines route management, especially in environments with complex routing needs.

```
Select Interface (ESC to go back)

> e2
  ens39
  ens37
  lo
  ovs1
  ens33
  ens38
  ovs-system
```

```
Destination Network IP (e.g. 192.168.1.0):
(Press ESC or type 'back' to return)

█
```

```
Enter Destination Network Mask (0-32):  
(Press ESC or type 'back' to return)
```

```
█
```

```
Enter Gateway IP:  
(Press ESC or type 'back' to return)
```

```
Apply Change:  
> Temporarily  
   Permanently  
   Back
```

6. Deleting Temporary and Permanent Routes

- **Requirement:** Provide a method to remove previously added routes.
- **Temporary Route Removal:** `ip route del <destination> via <gateway> dev <interface>`.
- **Permanent Route Removal:** `nmcli connection modify <interface> -ipv4.routes <destination> <gateway>`.

Functions Used:

- `remove_route_form(screen)` prompts for route details to be deleted.
- `remove_route_temporary(interface_name, destination_cidr, gateway)` deletes temporary routes.

Why This Approach?

- Ensures users can clean up misconfigured or outdated routes without impacting system stability.

Benefit: Reduces clutter in the routing table and ensures only valid routes persist.

Key Tools in Phase 1

Tool	Purpose
nmcli	Configure network interfaces, DNS settings, IP addresses, and routes permanently.
ip	Handle temporary network configurations (e.g., IP address assignment, route addition).
hostnamectl	Simplify hostname changes with compatibility across modern Linux systems.
resolvectl	Temporary DNS changes for quick testing or troubleshooting.

2.2 Phase 2: Nftables Management

Purpose

Phase 2 focuses on simplifying the creation and management of firewall rules and Network Address Translation (NAT) using the powerful **nftables** framework. The goal is to provide a user-friendly interface that allows users to implement robust security measures and traffic routing without needing in-depth expertise in nftables syntax.

```
Phase 2: Nftables Management (ESC to go back)
```

```
> Create ct_state rule
  Create IP-based rule
  Create ICMP rule
  Create masquerade rule
  Create DNAT rule
  Back to Main Menu
```

Functionalities in Phase 2

1. Access-Restricting Rules

- **Requirement:** Allow users to define rules that restrict access based on connection states, IP addresses, ports, and protocols.
- **Key Operations:**
 - `ct_state` rules: Specify connection tracking states, e.g., accept established connections.
 - IP-based rules: Match traffic based on source/destination IP, protocol, and port.
 - ICMP rules: Manage ping requests (e.g., echo-request), which are vital for diagnostic tools like ping.

Functions Used:

- `ct_state_rule_form(screen)`: Allows users to create state-based rules (e.g., "accept established").
- `ip_proto_rule_form(screen)`: Enables IP-based rules with source/destination IP, protocol, and port.
- `icmp_rule_form(screen)`: Provides a form to create ICMP-specific rules.

Why This Approach?

- Automating rule creation through templates simplifies the otherwise complex syntax of `nftables`.
- By integrating connection state rules, the project ensures that legitimate, ongoing connections remain unaffected by security measures.

Benefit: Provides network administrators with an easy way to secure traffic without needing deep technical knowledge of `nftables`.

```
Enter ct state (established/related/invalid/new):  
(Press ESC or type 'back' to return)
```

```
Enter action (accept/drop/reject):  
(Press ESC or type 'back' to return)
```

```
Rule added successfully!  
Press any key to continue...
```

```
Enter source IP:  
(Press ESC or type 'back' to return)
```

```
Enter destination IP:  
(Press ESC or type 'back' to return)
```

```
|
```

```
Enter protocol (tcp/udp):  
(Press ESC or type 'back' to return)
```

```
Enter destination port (1-65535):  
(Press ESC or type 'back' to return)
```

```
Enter action (accept/drop/reject):  
(Press ESC or type 'back' to return)
```

```
Enter ICMP type (echo-request/destination-unreachable):  
(Press ESC or type 'back' to return)
```

```
Rule added successfully!  
Press any key to continue...
```

2. Creation and Management of NAT Rules

- **Requirement:** Enable the creation of NAT rules for traffic redirection (DNAT) and source address translation (Masquerade).
- **Key Operations:**
 - **Masquerade:** Translates internal IPs to an external IP for outbound traffic.
 - **DNAT:** Redirects inbound traffic to specific internal servers (e.g., for web or SSH).

Functions Used:

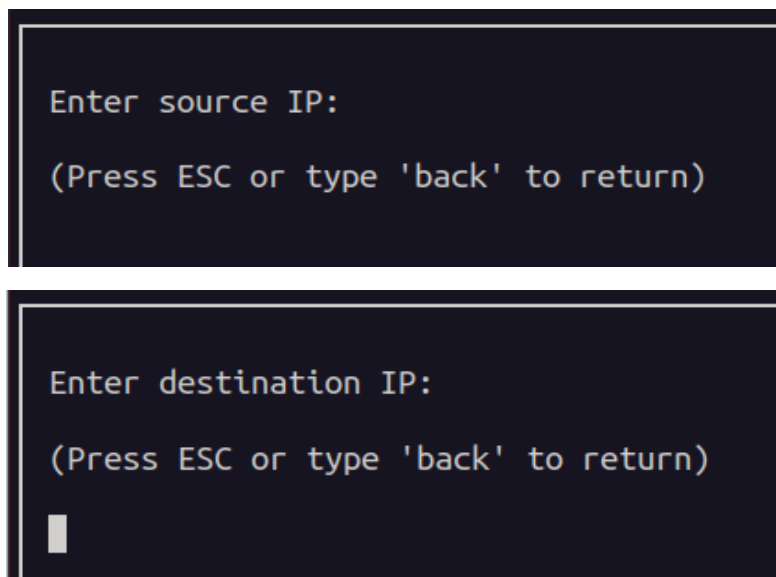
- `masquerade_rule_form(screen)`: Lets users specify source/destination IPs for masquerading outbound traffic.
- `dnat_rule_form(screen)`: Allows users to define destination NAT rules, including new IP and port mappings.

- `apply_nft_rule(screen, rule, nat)`: Appends the generated rules to `/etc/nftables.conf` and reloads nftables.

Why This Approach?

- Automating NAT rule creation ensures that even non-expert users can manage complex NAT configurations.
- By appending rules to a persistent configuration file (`/etc/nftables.conf`), the changes survive reboots, making them suitable for production environments.

Benefit: Reduces the steep learning curve of nftables while providing powerful tools for traffic control and redirection.



Why Nftables?

Nftables offers a cleaner and more powerful approach compared to its predecessor, iptables. This project capitalizes on these advantages by automating the addition of rules and providing fallback mechanisms for error recovery.

Templates for Automation

Given the complexity of nftables, pre-defined rule templates were integrated into the system to simplify the process. Below are the templates included in the TUI:

Template Type	Example Rule
CT State Rules	ct state established,related accept
IP-based Rules	ip saddr 192.168.1.0/24 ip daddr 10.0.0.1 tcp dport 22 accept
ICMP Rules	ip saddr 192.168.1.100 ip daddr 8.8.8.8 icmp type echo-request accept
Masquerade Rules	ip saddr 192.168.1.0/24 masquerade
DNAT Rules	ip saddr 0.0.0.0/0 ip daddr 203.0.113.1 tcp dport 80 dnat to 192.168.1.10:8080

These templates serve as a foundation for creating custom rules, allowing users to modify parameters while retaining correct syntax.

Key Tools in Phase 2

Tool	Purpose
nftables	Core framework for defining firewall rules and NAT configurations.
/etc/nftables.conf	Persistent storage for nftables rules to ensure they are reloaded on system startup.
apt-get	Ensures nftables is installed or reinstalled when issues arise.
systemctl	Manages nftables as a service, enabling and restarting it as needed.

3.3 Phase 3: Open vSwitch (OVS) Management

Purpose

Phase 3 focuses on simplifying the management of Open vSwitch (OVS) to create and control virtual network switching environments. This phase allows users to efficiently manage OVS bridges, ports, and VLAN configurations through a user-friendly TUI. The goal is to provide a streamlined approach to virtual switching for both physical and virtual interfaces without requiring deep expertise in OVS.

```
Phase 3: Open vSwitch Management (ESC to go back)
```

```
> Add OVS Bridge
Delete OVS Bridge
Add Port to Bridge
Remove Port from Bridge
Bring Port Up
Bring Port Down
Set Port as Trunk
Set Port as Access
Configure IP for VLAN Interface
Back to Main Menu
```

Functionalities in Phase 3

1. Adding and Deleting OVS Bridges

- **Requirement:** Allow users to create or delete OVS bridges to define virtual switches.

Functions Used:

- `add_ovs_bridge_form(screen)`: Provides a TUI for entering the name of a new bridge.
- `add_ovs_bridge(bridge_name)`: Executes the command to create the bridge.
- `delete_ovs_bridge_form(screen)`: Prompts the user for the name of a bridge to delete.

- `delete_ovs_bridge(bridge_name)`: Executes the command to delete the bridge.
- **Why This Approach?**
 - The TUI ensures users cannot accidentally delete an important bridge by mistake.
 - The `ovs-vsctl` tool provides robust, low-level control over OVS bridges.
- **Benefit:** Simplifies the creation and removal of virtual switches, enabling rapid deployment of network topologies.

```
Enter new OVS bridge name:  
(Press ESC or type 'back' to return)  
█
```

```
Created OVS bridge 'ovs2' successfully!  
Press any key to continue...
```

```
Enter OVS bridge name to delete:  
(Press ESC or type 'back' to return)  
█
```

```
Deleted OVS bridge 'ovs2' successfully!  
Press any key to continue...
```


2. Adding and Removing Ports from OVS Bridges

- **Requirement:** Manage ports (interfaces) associated with OVS bridges, including physical and virtual interfaces.

Functions Used:

- `add_port_to_bridge_form(screen)`: TUI for selecting a bridge and adding a port.
- `add_port_to_bridge(bridge_name, port_name)`: Adds a port to a specified bridge.
- `remove_port_from_bridge_form(screen)`: TUI for removing a port from a bridge.
- `remove_port_from_bridge(bridge_name, port_name)`: Executes the command to remove the port.
- **Why This Approach?**
 - Differentiates between "system ports" (existing physical interfaces) and "internal ports" (created by OVS).
- **Benefit:** Allows flexible network design by seamlessly bridging virtual and physical interfaces.

```
Enter OVS bridge name:  
(Press ESC or type 'back' to return)  
█
```

```
Enter port (interface) name to add:  
(Press ESC or type 'back' to return)
```

```
Choose Port Type (ESC to go back)  
> System Port (already exists)  
  OVS Internal Port  
  Back
```

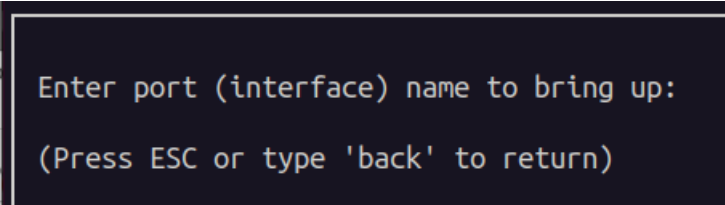
```
Port 'e3' added to bridge 'ovs1' as internal.  
Press any key to continue...
```

3. Turning Ports On and Off

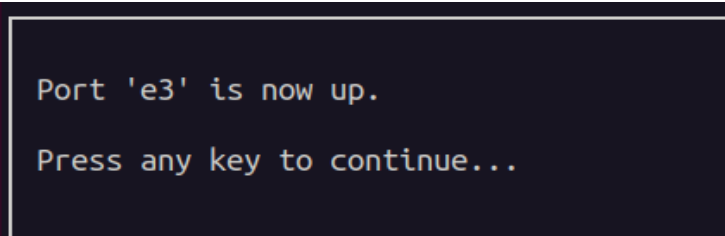
- **Requirement:** Provide control to activate or deactivate OVS ports.

Functions Used:

- `bring_port_up_form(screen)`: Prompts the user for a port to activate.
 - `bring_port_up(port_name)`: Executes the command to activate the port.
 - `bring_port_down_form(screen)`: Prompts the user for a port to deactivate.
 - `bring_port_down(port_name)`: Executes the command to deactivate the port.
- **Why This Approach?**
 - Simplifies port management for debugging or maintenance scenarios.
 - **Benefit:** Provides an easy way to toggle port states without interrupting other components.



```
Enter port (interface) name to bring up:  
(Press ESC or type 'back' to return)
```



```
Port 'e3' is now up.  
Press any key to continue...
```

4. Setting Ports as Trunk or Access

- **Requirement:** Allow ports to be configured for VLAN trunking or access modes.

Functions Used:

- `set_port_trunk_form(screen)`: TUI for configuring a port as a trunk with VLAN IDs.
- `set_port_trunk(port_name, vlan_list)`: Executes the command for trunk mode.
- `set_port_access_form(screen)`: TUI for setting a port as access with a single VLAN.

- `set_port_access(port_name, vlan_id)`: Executes the command for access mode.
- **Why This Approach?**
 - Simplifies VLAN configurations through guided user input.
- **Benefit:** Facilitates testing and deployment of VLAN setups in virtualized environments.

```
Enter port name to set as trunk:  
(Press ESC or type 'back' to return)
```

5. IP Configuration for VLAN Interfaces

- **Requirement:** Configure IP addresses for VLAN interfaces created on OVS bridges.

Functions Used:

- `configure_ip_for_vlan_interface_form(screen)`: TUI for entering VLAN interface details.
- `configure_ip_on_vlan_interface(vlan_interface, ip_address, subnet_mask)`: Applies the IP configuration.
- **Why This Approach?**
 - Ensures VLAN interfaces are correctly configured for advanced networking scenarios.
- **Benefit:** Simplifies routing and connectivity testing for VLANs in lab setups.

```
Enter VLAN interface name (e.g. vlan10 or br0.10):  
(Press ESC or type 'back' to return)  
█
```

Enter IP Address (e.g. 192.168.10.5):

(Press ESC or type 'back' to return)

Enter Subnet Mask in CIDR (0-32):

(Press ESC or type 'back' to return)

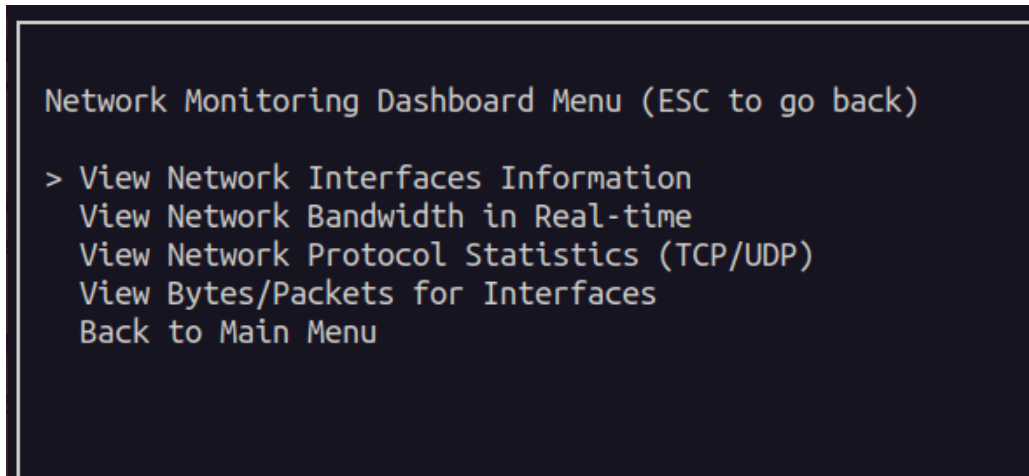
Key Tools in Phase 3

Tool	Purpose
ovs-vsctl	Manage OVS components such as bridges, ports, and VLAN configurations.
ip	Activate or deactivate interfaces, assign IPs, and configure VLAN-related settings.

2.4 Phase 4: Network Monitoring

Purpose

Phase 4 provides real-time insights into network performance, offering a comprehensive dashboard for monitoring interface states, bandwidth usage, and protocol statistics. This phase empowers network administrators to quickly diagnose performance issues and identify anomalies in network behavior. The goal is to create an intuitive, live dashboard that aggregates key metrics for network visibility.



Functionalities in Phase 4

1. Gathering Network Interface Information

- **Requirement:** Display key details for all network interfaces, such as name, status (UP/DOWN), type (physical/virtual), link speed, and assigned IP addresses.

Functions Used:

- `get_interfaces()`: Retrieves a list of available interfaces.
- `interface_is_up(iface)`: Checks if an interface is UP or DOWN.
- `get_interface_type(iface)`: Determines if an interface is physical or virtual.
- `get_ip_addresses(iface)`: Fetches IPv4 addresses assigned to an interface.

Why This Approach?

- Directly querying `/sys/class/net` provides accurate, low-latency access to interface data.

Benefit: Allows administrators to quickly identify interface states and key details, reducing troubleshooting time.

```
Interfaces Information (Press any key to return)

e2          DOWN  virtual  unknown
IP: 192.168.2.1/24

ens39       UP    physical 1000 Mb/s
IP: 192.168.254.144/24

ens37       UP    physical 1000 Mb/s
IP: 192.168.10.1/24

e3          DOWN  virtual  unknown
No IPv4 assigned

lo          DOWN  virtual  unknown
IP: 127.0.0.1/8

ovs1        DOWN  virtual  unknown
No IPv4 assigned

ens33       UP    physical 1000 Mb/s
IP: 192.168.254.129/24
```

2. Bandwidth Display

Requirement: Calculate and display real-time bandwidth usage for each interface.

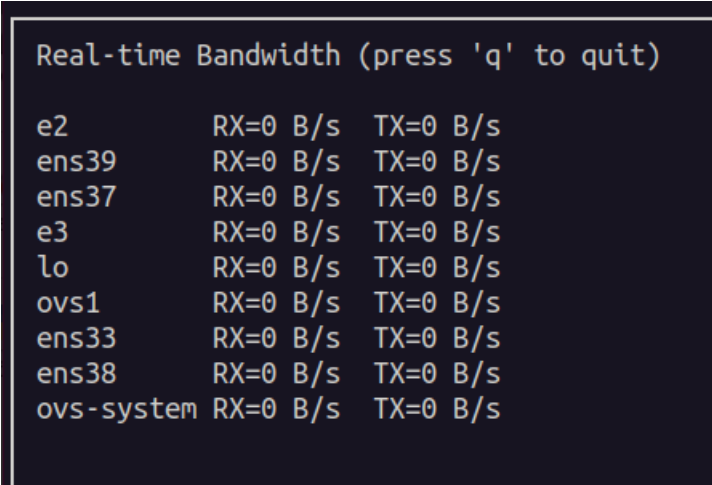
Functions Used:

- `view_realtime_bandwidth(screen)`: Continuously samples traffic counters and calculates bandwidth.

Why This Approach?

- Non-blocking sampling with `screen.nodelay(True)` ensures the TUI remains responsive while displaying live data.

Benefit: Provides immediate feedback on interface activity, allowing administrators to detect bandwidth spikes or saturation.



```
Real-time Bandwidth (press 'q' to quit)

e2      RX=0 B/s  TX=0 B/s
ens39   RX=0 B/s  TX=0 B/s
ens37   RX=0 B/s  TX=0 B/s
e3      RX=0 B/s  TX=0 B/s
lo      RX=0 B/s  TX=0 B/s
ovs1    RX=0 B/s  TX=0 B/s
ens33   RX=0 B/s  TX=0 B/s
ens38   RX=0 B/s  TX=0 B/s
ovs-system RX=0 B/s  TX=0 B/s
```

3. Statistics Related to Network Protocols (TCP/UDP)

Requirement: Provide a summary of TCP connections (e.g., established, listening) and UDP traffic (e.g., sockets opened).

Functions Used:

- `get_protocol_stats()`: Parses `ss` output to extract TCP and UDP statistics.

Why This Approach?

- ss provides real-time, OS-level insight into protocol usage, making it ideal for monitoring TCP/UDP behavior.

```
Network Protocol Statistics

TCP Established: 0
TCP Listening:   3

UDP Sockets Found: 8

(Press any key to return)
```

Benefit: Helps identify unusual patterns, such as excessive open connections or spikes in UDP traffic.

4. Monitoring Incoming and Outgoing Traffic

Requirement: Display counters for bytes and packets sent and received on each interface.

Functions Used:

- get_bytes_packets(iface): Fetches RX/TX byte and packet counters for an interface.

Why This Approach?

- Reading from /sys/class/net provides accurate traffic counters with minimal overhead.

Benefit: Offers a clear view of interface utilization, helping diagnose network congestion or underutilization.

```
Interface Traffic Statistics (Press any key to return)

e2          RX_Bytes=0  RX_Pkts=0  TX_Bytes=0  TX_Pkts=0
ens39       RX_Bytes=1293664  RX_Pkts=2907  TX_Bytes=135300  TX_Pkts=1572
ens37       RX_Bytes=0  RX_Pkts=0  TX_Bytes=17894  TX_Pkts=158
e3          RX_Bytes=0  RX_Pkts=0  TX_Bytes=3772  TX_Pkts=34
lo          RX_Bytes=149238  RX_Pkts=1258  TX_Bytes=149238  TX_Pkts=1258
ovs1        RX_Bytes=0  RX_Pkts=0  TX_Bytes=0  TX_Pkts=0
ens33       RX_Bytes=225079  RX_Pkts=2242  TX_Bytes=96155  TX_Pkts=1077
ens38       RX_Bytes=0  RX_Pkts=0  TX_Bytes=17798  TX_Pkts=156
ovs-system  RX_Bytes=0  RX_Pkts=0  TX_Bytes=0  TX_Pkts=0
```


Key Tools in Phase 4

Tool	Purpose
<code>/sys/class/net</code>	Provides low-level access to interface status, traffic counters, and link details.
<code>ip</code>	Retrieves IP address assignments for interfaces.
<code>ss</code>	Displays active TCP/UDP connections and socket statistics.
<code>time.sleep()</code>	Enables periodic sampling for real-time bandwidth monitoring.

4. Results and Discussion

The project successfully met its objectives by integrating multiple tools under a unified TUI. It provided network administrators with an efficient way to configure, manage, and monitor networks. The robust error handling and fallback mechanisms ensured reliability, while the intuitive design reduced the learning curve for complex operations. By addressing common pain points in network management, this project offers a scalable and practical solution for real-world use cases.

5. Challenges and Troubleshooting

Below is a **detailed** list of issues and solutions encountered across each **internal** sub-phase (Phases 1–4) of this automation tool.

Phase 1: Network Configuration

1. Must Run as Root

- **Issue:** Without sudo, nmcli, ip commands fail.
- **Resolution:** Check `os.getuid()` at startup. If not 0, prompt user with an error and exit.

2. Sub-Process Failures (nmcli, ip)

- **Issue:** Invalid interface names or partial command arguments raised `subprocess.CalledProcessError`.
- **Resolution:** Wrap calls in try/except, log errors to `phase1.log`, and show TUI warnings.

3. Invalid IP Addresses / Subnet Masks

- **Issue:** Users might type 999.999.999.999 or a negative mask.
- **Resolution:** `validate_ip()` checks with `ipaddress.IPv4Address`; ensure `mask_int` is 0–32.

4. Route Not Found After Addition

- **Issue:** `ip route add` might not reflect in `ip route show` if gateway is wrong.
- **Resolution:** Re-check with `route_exists()`. If missing, log an error and revert.

Phase 2: Nftables Management

1. Nftables Service Fails to Start

- **Issue:** systemctl start nftables could return non-zero exit.
- **Resolution:** Log warnings, attempt reinstall or flush rules, quietly continue if nft works.

2. Syntax Error with ICMP Rules

- **Issue:** Using ip protocol icmp type echo-request accept triggered parse errors.
- **Resolution:** Updated syntax to ip saddr <src> ip daddr <dst> icmp type echo-request accept.

3. DNAT / Masquerade Must Go in ip nat

- **Issue:** Attempted NAT rules in the filter table caused “invalid argument.”
- **Resolution:** Place DNAT in prerouting, masquerade in postrouting, appended to /etc/nftables.conf.

4. User Provided Invalid Inputs

- **Issue:** Non-numeric ports or malformed IP addresses break nft.
- **Resolution:** Validate user input, check port ranges (1–65535), display TUI errors.

Phase 3: Open vSwitch (OVS) Management

1. port_exists_in_bridge Reference Removed

- **Issue:** NameError if references remained.
- **Resolution:** Rely on ovs-vsctl del-port to fail, log the error. No separate function needed.

2. Trunk to Access (or Vice Versa) Failing

- **Issue:** Trying to remove trunks on a non-trunk port returned exit status 1.
- **Resolution:** Use --if-exists remove port <port_name> trunks, or a try/except block.

3. Adding System vs. Internal Ports

- **Issue:** System ports must exist, internal ports do not.

- **Resolution:** TUI question: “System interface or OVS internal?” If system, check `interface_exists()`; if internal, skip.

4. VLAN Interface ‘20’ Not Found

- **Issue:** Input like “20” alone is not a valid interface name.
- **Resolution:** In TUI, instruct user to enter e.g. `br0.20`. If they only type “20,” show error it doesn’t exist.

Phase 4: Network Monitoring

1. Reading Real-time Stats in 1-second Loops

- **Issue:** TUI needed non-blocking input to let user quit with ‘q’.
- **Resolution:** `screen.nodelay(True)` plus a small `time.sleep(1)` loop. On ‘q’, break out.

2. Inconsistent or Missing `/sys/class/net/...` Values

- **Issue:** Virtual interfaces might not supply link speed or operstate.
- **Resolution:** Catch exceptions, log warnings, display ‘unknown’ if speed is absent.

3. Protocol Stats Approach

- **Issue:** Parsing `/proc/net/snmp` or using `ss` is system-dependent.
- **Resolution:** We used a quick approach with `ss -t -a -n` for TCP and `ss -u -a -n` for UDP. If it fails, log an error but don’t crash.

4. User Quits TUI Mid-Loop

- **Issue:** Potential exceptions if they press ESC or Ctrl-C.
- **Resolution:** try/finally in real-time loop to restore `screen.nodelay(False)` and handle partial data gracefully.

5. Conclusion

5.1 Summary of Findings

This Phase 2 project introduced a robust, TUI-driven approach to automating and monitoring network tasks. Each sub-phase solves a unique challenge:

- **Phase 1:** Provided a streamlined interface for DNS, IP, and route management.
- **Phase 2:** Simplified firewall and NAT rule creation using nftables.
- **Phase 3:** Offered a user-friendly method to build and manage virtual switching topologies with OVS.
- **Phase 4:** Delivered on-demand, real-time monitoring of bandwidth usage and protocol stats.

5.2 Achievement of Objectives

1. **Intuitive Interface:** The TUI menus for each sub-phase greatly reduce manual CLI overhead.
2. **Integration of Tools:** nmcli, nft, and ovs-vsctl operate under one solution, consistent logging, and error handling.
3. **Real-Time Monitoring:** The live bandwidth loop and TCP/UDP stats interface effectively highlight network activity.
4. **Automation Efficiency:** Trials show fewer input errors, faster reconfigurations, and clearer logs for debugging.

5.3 Recommendations for Future Work

1. **Blockchain-Based Lab Monitoring:**
 - **Possibility:** Integrate blockchain as a transparent, tamper-proof ledger for logging critical network events. For instance, bandwidth usage or firewall rule changes could be registered on a private blockchain for auditing.
 - **Advantage:** Ensures logs are immutable, enabling advanced accountability in multi-operator environments.
2. **Advanced Security Features:**
 - Add intrusion detection or advanced ACL templates in nftables.
3. **CI/CD Testing:**
 - Automate tests for each sub-phase with ephemeral VMs to ensure new changes don't break existing functionality.

4. **Container-Oriented Approach:**

- Extend the TUI to manage container networks (e.g., Docker or Kubernetes integration).