Bulletin of Electrical Engineering and Informatics
Vol. 15, No. 1, February 2026, pp. 237~252
ISSN: 2302-9285, DOI: 10.11591/eei.v15i1.9562

Design and evaluation of a Python-based network automation system for internet of things devices

Eslam Samy El-Mokadem[1], Bilal Bataineh[2], Samy El-Mokadem[3], Abdelmoty M. Ahmed[4], Mohamed A. Torad[1]
[1]Department of Electronics and Communication Engineering, Higher Technological Institute, 10th of Ramadan City, Egypt
[2]Department of Computer Science, Jadara University, Irbid, Jordan
[3]Department of Computer Science, Faculty of Information Systems and Computer Science, October 6 University, October City, Egypt
[4]Department of Computer Science, Faculty of Information and Technology, Ajloun National University, Ajloun, Jordan

Article Info

Article history:

Received Nov 16, 2024
Revised Aug 24, 2025
Accepted Dec 6, 2025

Keywords:

Graphical network simulator-3
NAPALM
Netmiko
Network automation
Paramiko
Programmable network
Python

ABSTRACT

The increasing demand for the internet of things (IoT) and massive machine-type communications has significantly expanded network size and complexity. Recent research indicates that 95% of network tasks are monitored manually, leading to configuration complexity, human errors, faults, downtime risks, and time consumption. Network automation emerges as a practical solution by reducing administrative overhead and enabling reliable, scalable, and self-managing networks through scripting and standardized programming languages. This paper proposes a model for automated networks using Python-based methods, specifically Paramiko, Netmiko, and the network automation and programmability abstraction layer with multivendor support (NAPALM), to configure the enhanced interior gateway routing protocol (EIGRP) within the graphical network simulator-3 (GNS3) environment. The performance of the automated network was evaluated using two scenarios: with threading and without threading. Key metrics included execution time, configuration accuracy, error rates, and resource utilization. Simulation results demonstrate that the automated approach significantly outperforms manual configuration. In addition, the automated model with threading outperformed the automated model without threading, achieving execution time reductions up to 67% and 100% configuration accuracy with zero errors. These findings underscore the effectiveness of the proposed system for automating complex network tasks in large-scale IoT deployments.

Corresponding Author:

Eslam Samy El-Mokadem
Department of Electronics and Communication Engineering, Higher Technological Institute
10th of Ramadan City, Egypt
Email: islam.almokdem@hti.edu.eg

1. INTRODUCTION

The rapid expansion of the internet of things (IoT) and advancements in cloud computing have substantially increased the complexity of network infrastructure, intensifying the demand for robust intrusion detection and secure data handling mechanisms in diverse applications [1], [2]. In addressing these complexities, human expertise in microservices and cloud solutions plays a critical role, particularly in the strategic implementation of containerization and orchestration to enhance scalability and security in modern networked systems [3]. Consequently, IoT's proliferation across sectors like healthcare, industrial automation, and smart cities has amplified the attack surface, posing significant cybersecurity risks, which require advanced intrusion detection systems to ensure network security and data privacy [4]. This

necessitates that network engineers manage an ever-increasing workload in provisioning, maintaining, and monitoring these complex systems, particularly against threats like distributed denial of service (DDoS) attacks that can impact critical infrastructure [5].

This growth has created scalability challenges for network administration activities, particularly when dealing with numerous devices. The complexity of networks has also increased, making it difficult for administrators to configure various types and brands of routers. This complexity, coupled with the potential for human error, poses significant challenges [6]. Additionally, network administration is time-consuming, with the manual configuration of numerous routers in extensive network environments further amplifying complexity [7], [8]. Therefore, effective management of computer network systems is imperative [9]. An efficient approach to assist network administrators is network automation. Network automation involves automating the configuration, administration, testing, and utilization of network devices, both physical and virtual [10]. Various protocols and tools have been introduced to address these needs.

Automation protocols such as network configuration protocol (NETCONF), representational state transfer configuration protocol (RESTCONF), and yet another next generation (YANG) have become foundational in programmable networking. These protocols enable structured and vendor-neutral device configuration and state management. YANG, in particular, supports service abstraction and modularity in complex deployments [11]. Moreover, software-defined networking (SDN) has revolutionized network architecture by decoupling the control and data planes, enabling centralized and programmable management through controllers such as OpenDaylight and ONOS. SDN's programmability and adaptability are particularly beneficial in IoT environments, where networks often require dynamic reconfiguration [12].

Emerging strategies further integrate machine learning (ML) and artificial intelligence (AI) into automation workflows. These technologies facilitate predictive analytics, anomaly detection, and self-healing capabilities, enhancing modern networks' resilience and security. For instance, ML-enhanced SDN controllers can identify traffic anomalies and optimize routing paths in real time [13], [14].

In addition to these protocols and architectural innovations, common approaches involve using scripting languages to automate device setups and reduce configuration time and errors [15]. Python and Ansible are widely adopted tools for network automation due to their flexibility, modularity, and user-friendly development environments [16], [17]. Python, in particular, is a high-level programming language with a vast ecosystem of libraries and modules that facilitate the automation of complex networking tasks. It supports the development of application programming interfaces (APIs) capable of replacing traditional command-line interface (CLI) configurations, thereby promoting consistent, scalable, and scriptable workflows [18]–[20]. Ansible, by contrast, offers a declarative, agentless framework well-suited for automating routine tasks across heterogeneous network environments.

Python distinguishes itself through its versatility and scalability, particularly in the context of IoT platforms. Unlike domain-specific tools such as Ansible, which are optimized for standardized automation, Python provides a general-purpose programming environment that supports imperative logic, modular design, and advanced flow control. This allows developers to implement customized, event-driven workflows that align with the dynamic and diverse requirements of IoT systems. Moreover, Python's rich library ecosystem—including Netmiko, network automation and programmability abstraction layer with multivendor support (NAPALM), Paramiko, and PyYANG enables native support for essential network protocols such as NETCONF, RESTCONF, SNMP, and YANG, ensuring seamless interoperability in multi-vendor environments [21].

Additionally, Python's integration with AI and ML frameworks (e.g., TensorFlow and scikit-learn) enhances its capabilities for intelligent network automation. These include predictive analytics, anomaly detection, and adaptive system control critical features in managing large-scale, heterogeneous IoT infrastructures. While Ansible remains a powerful tool for executing standardized configurations efficiently, it lacks the advanced programmability and data-processing flexibility offered by Python. Consequently, Python represents a more comprehensive and adaptable solution for network automation in complex IoT deployments [22]. Table 1 compares Python and Ansible in the context of IoT network automation, highlighting the advantages of Python-based systems regarding flexibility, scalability, and protocol integration [21]-[23].

This comparative analysis complements the findings in recent research. Recent research has made significant advances in network automation. For example, Islami *et al*. [23] investigated the use of network automation on the Raspberry Pi for configuring network devices using Ansible, highlighting its potential in reducing configuration and maintenance duration while minimizing human errors. Fuzi *et al*. [24] utilized Ansible for network automation to set up EIGRP routing and advanced configurations within the graphical network simulator-3 (GNS3) environment. Mazin *et al.* [25] create a Python-centered framework that facilitates communication with various third-party network devices. This is accomplished by harnessing the expansive collection of Python libraries and APIs tailored specifically for networking [25].

Ortiz-Garcés *et al.* [26] proposed a network automation model using Ansible and open shortest path first (OSPF) to harden campus networks, focusing on communication protocols, hardening configurations, and playbook deployment [26]. Datta *et al.* [27] introduce an integrated platform that simplifies the management of various network devices by providing a unified solution. Al-Mekhlal *et al.* [28] explored efficient Python programs for network automation in data centers, aiming to automate tasks in private cloud environments. Datta *et al.* [29] utilized Ansible for network automation to set up BGP routing and advanced configurations in a live network setting. Chen *et al.* [30] proposed an automated configuration framework covering task design, parameter arrangement, sequence arrangement, scenario design, scheme design, and interaction with configuration tool interfaces. Alfaresa *et al.* [7] analyzed the performance of network automation using the Paramiko and Telnetlib libraries, focusing on OSPF for IGP and BGP for EGP.

Table 1. Comparison of Ansible vs. Python in IoT network automation

Criteria	Python-based automation	Ansible
Programming model	Imperative and object-oriented; supports advanced logic, loops, exception handling	Declarative; limited flow control and logic customization
Performance and scalability	High performance with support for threading, multiprocessing, and persistent SSH sessions	Slower for large-scale tasks due to repeated SSH connections per task
Customization and extensibility	Easily extensible with native libraries (e.g., Netmiko, NAPALM, and PyYANG)	Requires custom module development; steeper learning curve for advanced customization
Real-time interaction	Supports dynamic decision-making and real-time device feedback handling	Lacks native support for real-time adaptation during execution
AI/ML integration	Seamless integration with ML frameworks (e.g., TensorFlow and scikit-learn)	No built-in AI/ML support; limited to static playbook logic
Protocol support	Fine-grained support for NETCONF, RESTCONF, YANG, and SNMP through multiple libraries	Relies on existing modules; limited flexibility for direct protocol-level scripting
Vendor flexibility	Highly adaptable across multi-vendor environments	Requires module support per vendor; not all hardware features may be accessible
Suitability for IoT environments	Excellent for heterogeneous, resource-constrained, and dynamic IoT deployments	Suitable for standard automation, but less effective in complex, real-time IoT contexts

This paper identifies the best method to improve scripting efficiency using Python, specifically through the use of Paramiko, Netmiko, and NAPALM for configuring EIGRP routing and advanced configurations in the GNS3 environment. This analysis aims to evaluate the performance of automated network deployment, improve the efficiency of configuring network devices, and determine the differences in performance regarding the time needed to configure network devices. Additionally, it investigates the integration of threading techniques with Paramiko, Netmiko, and NAPALM to enhance the automation process by parallelizing tasks and reducing execution time providing a scalable and lightweight solution tested within a simulated environment tailored to IoT scenarios. Two scenarios were used to investigate performance: without threading and with threading. Performance evaluation for both scenarios was conducted for various methods (Paramiko, Netmiko, and NAPALM) in terms of execution time and errors, impacting the quality of service (QoS) and management in computer networks.

The remainder of this paper is organized as follows: section 2 covers the method, section 3 describes network automation methods using Python scripting without threading, section 4 details network automation methods using Python scripting with threading, section 5 presents simulation results, and section 6 concludes and discusses future work.

2. METHOD

This section outlines the methodology for designing and evaluating a Python-based network automation configuration system to configure the enhanced interior gateway routing protocol (EIGRP) in a simulated network environment using GNS3. The automation workflow leverages Python libraries such as Paramiko, Netmiko, and NAPALM to automate the configuration of network devices via secure shell (SSH) connections. The methodology is divided into three key components: emulator overview, network topology design, and automation workflow.

2.1. Emulator overview

GNS3 is a network software emulator first released in 2008. It allows the combination of virtual and real devices to simulate complex networks and uses Dynamips emulation software for simulating and testing the Cisco internet work operating system (IOS). GNS3 consists of two main components: the all-in-one software, a graphical user interface (GUI) that facilitates network design and simulation; and the virtual machine (VM), a server that runs in a virtual environment, providing better topology size and device support.

GNS3 features significantly ease usability, reusability, manageability, interconnectivity, and distribution, thereby decreasing both cost and time.

2.2. Network topology design

To apply and evaluate the script with various methods using Python, namely Paramiko, Netmiko, and NAPALM, we designed a simple network topology to configure the EIGRP routing protocol in the GNS3 environment. The topology includes an Ubuntu Docker container, which runs the automated Python scripts for configuring network devices via SSH connections [27], [28]. SSH is a cryptographic network protocol for securely operating network services over an insecure network. The topology also includes a Layer 2 switch (using real Cisco IOS) that connects to three routers (also using real Cisco IOS) to be automatically configured, and a cloud component for internet access, as shown in Figure 1.

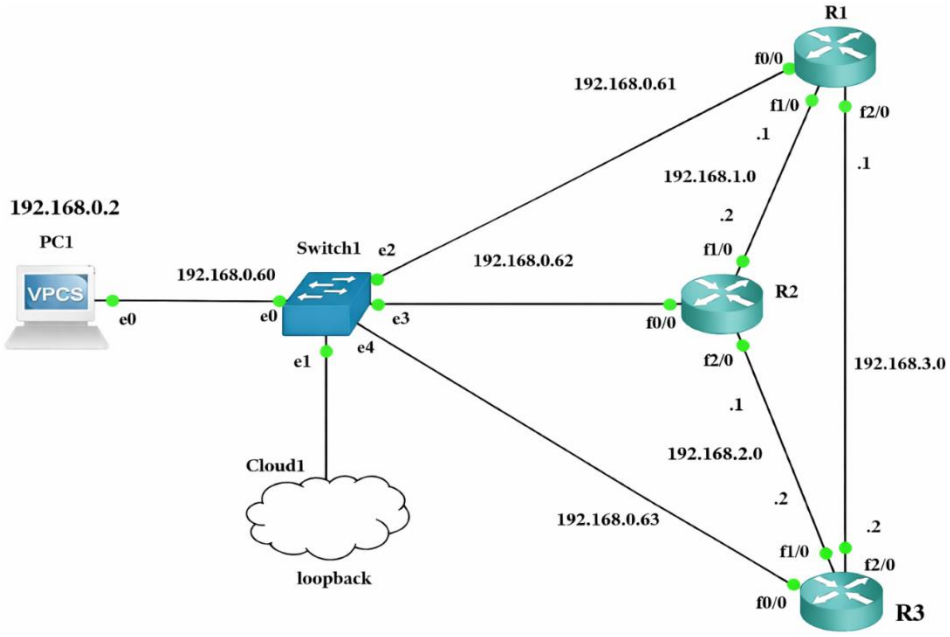


Figure 1. Simple network topology

The topology ensures a controlled environment to test the automation scripts, with SSH enabled on all devices for secure remote access. The GNS3 emulator, combining a GUI and a VM, supports the simulation of Cisco IOS devices using Dynamips emulation software, ensuring accurate replication of real-world network behavior.

2.3. Automation workflow

The automation workflow configures the EIGRP on three routers (R1, R2, and R3) using Python scripts within an Ubuntu Docker container. The process leverages the Paramiko, Netmiko, and NAPALM libraries for secure and efficient configuration. The workflow consists of the following steps:

a. Initialization: import libraries (Paramiko, Netmiko, NAPALM, threading, and time) and define device parameters (IP addresses, credentials, and EIGRP settings).
b. SSH connection: establish secure SSH connections to routers using:
– Paramiko: low-level SSHv2 for command execution.
– Netmiko: simplified SSH with Cisco IOS support.
– NAPALM: unified API for multi-vendor configuration.
c. Configuration: apply EIGRP commands (e.g., enable protocol and advertise networks) tailored to each router's interfaces.
d. Validation: verify configurations via commands (e.g., show IP EIGRP neighbors) and log outputs.

The workflow is executed in the GNS3 environment, with flowcharts in Figure 2 illustrating the process for each library. This structured approach ensures reliable and scalable automation, suitable for large-scale IoT deployments.

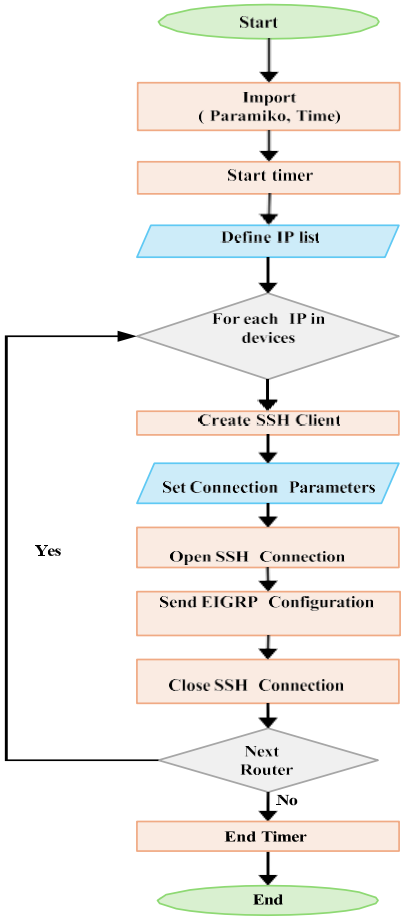


Figure 2. Flowchart illustrating the setup processes for Paramiko script

3. NETWORK AUTOMATION METHODS USING PYTHON SCRIPTING LANGUAGE

Python is a powerful and flexible programming language, ideally suited for automation and a wide range of programming tasks. It offers an extensive set of tools and functions that enable the creation of scripts to automate the configuration and management of network devices, including routers, switches, firewalls, and servers [25]. This paper proposes a model for implementing Python scripts to automate network configuration using various methods, including Paramiko, Netmiko, and NAPALM. These methods are utilized to configure EIGRP routing and perform advanced configurations within the GNS3 environment. An Ubuntu Docker container is employed to run the Python scripts, facilitating secure connections to devices and automating their configuration via SSH.

3.1. Paramiko

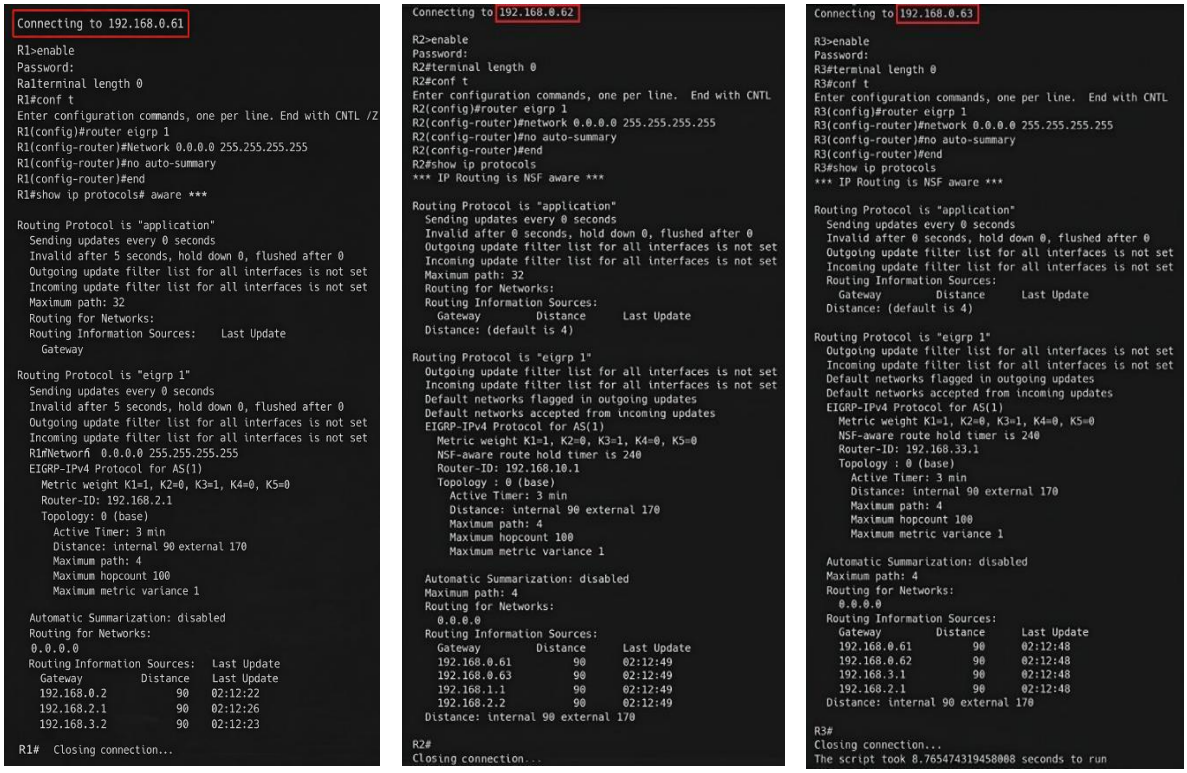
Paramiko is a pure Python interface that implements the SSH protocol version 2 in Python, providing both client and server functionality. It achieves high performance through low-level cryptographic concepts. Any device configurable via SSH can also be managed using Python scripts with this module [7].

Figure 2 illustrates the Paramiko script used to configure the EIGRP. This flowchart provides a step-by-step process of using Paramiko to automate the configuration of EIGRP routing on network devices, highlighting the importance of secure and efficient network management. In this script, SSH encryption is enabled to ensure the secure transfer of information between the client and server, allowing users to execute shell commands on a remote computer as if they were physically present. The EIGRP protocol is activated to share routes with other routers within the same autonomous system.

Figure 3 provides validation of the successful implementation of the Paramiko script for the automated configuration of the EIGRP across three routers (R1, R2, and R3) in the GNS3 environment. The subfigures delineate the script's execution:

Figure 3(a), depicting Router 1 (R1), illustrates the establishment of an SSH connection and the application of EIGRP configuration commands. The displayed output confirms the activation of the protocol, including the assignment of an autonomous system (AS) number and network advertisements. Figure 3(b),

focusing on Router 2 (R2), demonstrates the replication of the process for R2, with the script tailored for its specific interface IPs. The output verifies the consistent replication of configurations across the network devices. Figure 3(c), illustrating Router 3 (R3), completes the triad, showcasing the script's capability to manage concurrent configurations while preserving consistency.



(a) (b) (c)

Figure 3. Outputs of the Paramiko script for different devices: (a) R1, (b) R2, and (c) R3

Each subfigure effectively showcases the automated application of EIGRP configuration commands, the successful establishment of routing information exchange, and the retrieval of dynamic routing tables. These results authenticate the effectiveness of Paramiko in automating network device configuration with precision. No errors were reported throughout the execution, and the total execution time of 8.7654743 seconds (across all devices) exemplifies the tool's efficiency in multi-device automation. The precise execution time of the script, 8.7654743 seconds, highlights its high efficiency for deploying routing protocols across multiple network nodes.

3.2. Netmiko

Netmiko is an open-source, multi-vendor library that allows devices from various vendors to be configured using Python. It supports a range of devices, including Cisco IOS, Juniper, Arista, HP, and Linux, with limited testing on vendors such as Alcatel, Huawei, and Ubiquity. Netmiko runs on top of Paramiko, simplifying SSH connections to network devices, making them less complex, more versatile, and easier to use. While Netmiko is easier to use and supports specific vendors, Paramiko can communicate with any device that supports SSH. Both Paramiko and Netmiko are viable options for devices that do not support APIs [7], [30]-[32].

Figure 4 shows the flowchart of the Netmiko script used to configure EIGRP. This script enables SSH encryption to secure the transfer of information between the client and server, allowing users to execute shell commands on a remote computer. Additionally, the EIGRP protocol is activated to share routes with other routers within the same autonomous system.

Figure 5 illustrates the results of the Netmiko script applied to three routers following the execution of the SSH connection and the configuration of the EIGRP protocol. This figure demonstrates the successful implementation of network automation using Netmiko to establish EIGRP routing across multiple devices,

showcasing the efficiency and accuracy of Python-based methods for network configuration tasks. Notably, the execution time for the Netmiko script was recorded at 6.66721 seconds, highlighting its rapid performance in automating complex network configurations

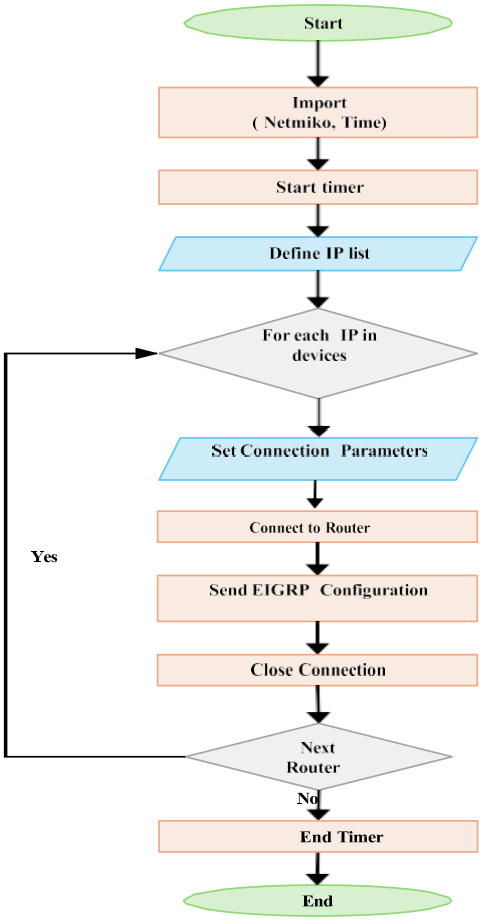


Figure 4. Flowchart illustrating the setup processes for Netmiko script

```
SSH connection established to 192.168.0.61:22
Interactive SSH session established
R1>
Entering the enable mode...
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#int loopback 1
R1(config-if)#router eigrp 1
R1(config-router)#network 0.0.0.0 255.255.255.255
R1(config-router)#no auto-summary
R1(config-router)#end
R1#
Closing connection
SSH connection established to 192.168.0.62:22
Interactive SSH session established
R2>
Entering the enable mode...
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int loopback 1
R2(config-if)#router eigrp 1
R2(config-router)#network 0.0.0.0 255.255.255.255
R2(config-router)#no auto-summary
R2(config-router)#end
R2#
Closing connection
SSH connection established to 192.168.0.63:22
Interactive SSH session established
R3>
Entering the enable mode...
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int loopback 1
R3(config-if)#router eigrp 1
R3(config-router)#network 0.0.0.0 255.255.255.255
R3(config-router)#no auto-summary
R3(config-router)#end
R3#
Closing connection
The script took 6.667213201522827 seconds to run
```

Figure 5. The output of the Netmiko script applied to three routers

3.3. Network automation and programmability abstraction layer with multivendor support

NAPALM is a Python library that provides a unified API to interact with various network device operating systems. NAPALM supports multiple methods to connect to devices, manipulate configurations, and retrieve data. Figure 6 shows the flowchart of the NAPALM script used to configure EIGRP. This script enables SSH encryption to secure the transfer of information between the client and server, allowing users to execute shell commands on a remote computer. Additionally, the EIGRP protocol is activated to share routes with other routers within the same autonomous system.

Figures 7(a) and (b) illustrate the results of the NAPALM script applied to three routers following the execution of the SSH connection and the configuration of the EIGRP protocol. These figures demonstrate the successful implementation of network automation using NAPALM to establish EIGRP routing across multiple devices, showcasing the efficiency and accuracy of Python-based methods for network configuration tasks. Notably, the execution time for the NAPALM script was recorded at 7.590587 seconds, highlighting its rapid performance in automating complex network configurations.

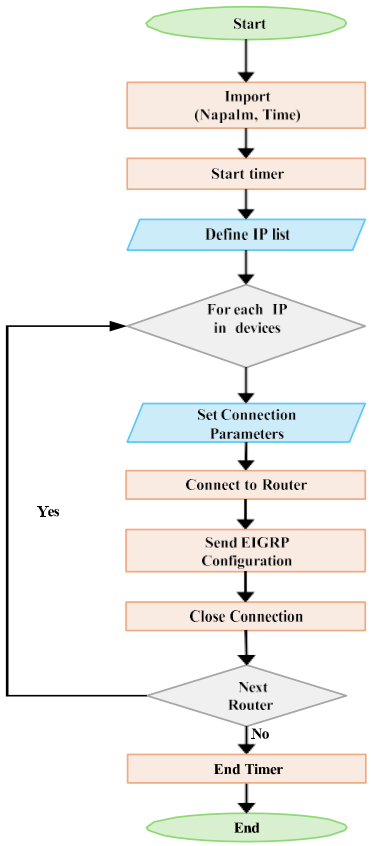
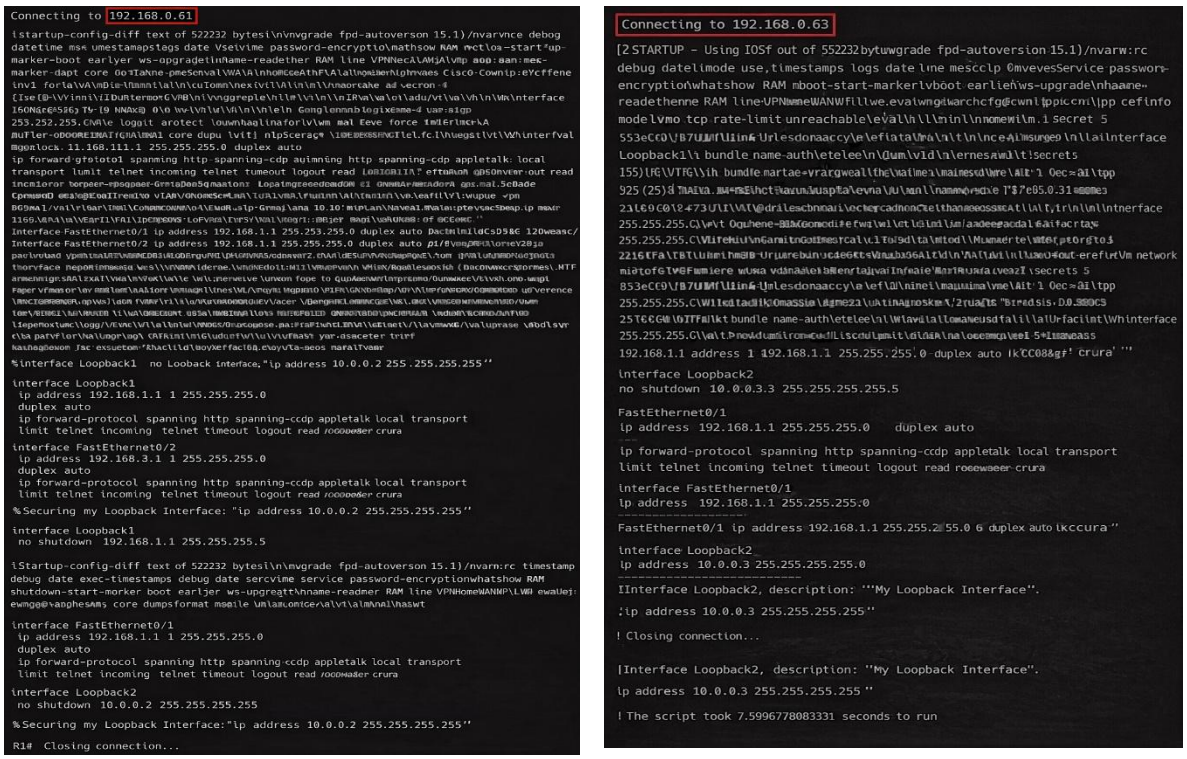


Figure 6. Flowcharts illustrating the setup processes for NAPALM script



(a) (b)

Figure 7. Outputs from NAPALM script: (a) R1 and R2, and (b) R3

4. NETWORK AUTOMATION TOOLS WITH THREADING

Threading is an effective tool for establishing parallelism and improving performance in Python programming. Threading opens new possibilities for effectively utilizing system resources by enabling the simultaneous execution of several tasks within a single process. By merging this tool with network automation methods such as Paramiko, Netmiko, and NAPALM, threading can send connection and configuration commands simultaneously instead of waiting to finish the configuration on one device before continuing with the rest. This approach significantly reduces the time required for the application to complete.

By applying threading to the scripts of network automation methods such as Paramiko, Netmiko, and NAPALM to configure the EIGRP for the same topology shown in Figure 1, we enable SSH encryption to ensure the secure transfer of information between the client and the server. This setup allows users to execute shell commands on all remote devices simultaneously, as if they were physically present at each device. Additionally, the EIGRP protocol is activated to share routes with other routers within the same autonomous system, as shown in Algorithms 1-3. As detailed in those algorithms, the threading process involves initializing device parameters, establishing SSH connections, sending configuration commands, and logging execution times, thereby streamlining the automation process and significantly reducing overall execution time.

Algorithm 1. Execution flow of Paramiko script for network configuration with threading

```
Step 1: Initialize process
Step 2: Import libraries: Paramiko, time, threading
Step 3: Start timer
Step 4: Define connect function to establish SSH connection
Step 5: Define get shell, send command, show, and close shell functions for shell
interaction
Step 6: Define EIGRP function for routing configuration
Step 7: Set connection parameters (e.g., IPs, credentials)
Step 8: Create device parameter list
Step 9: Initialize multi-threading for concurrent configuration
      a. Send command and apply configuration on Router 1 (R1)
      b. Send command and apply configuration on Router 2 (R2)
      c. Send command and apply configuration on Router 3 (R3)
Step 10: Finalize multi-threading
Step 11: End timer and record duration
Step 12: End process
```

Algorithm 2. Execution flow of Netmiko script for network configuration with threading

```
Step 1: Start process
Step 2: Import libraries: time, Netmiko, threading
Step 3: Start timer
Step 4: Define EIGRP configuration function
Step 5: Define connect function for SSH connection setup
Step 6: Set up connection parameters (e.g., IP addresses, credentials)
Step 7: Create parameter list for device configurations
Step 8: Initialize multi-threading for concurrent configuration
      a. Send command and apply configuration on Router 1 (R1)
      b. Send command and apply configuration on Router 2 (R2)
      c. Send command and apply configuration on Router 3 (R3)
Step 9: Finalize multi-threading
Step 10: End timer and record execution duration
Step 11: End process
```

Algorithm 3. Execution flow of NAPALM script for network configuration with threading

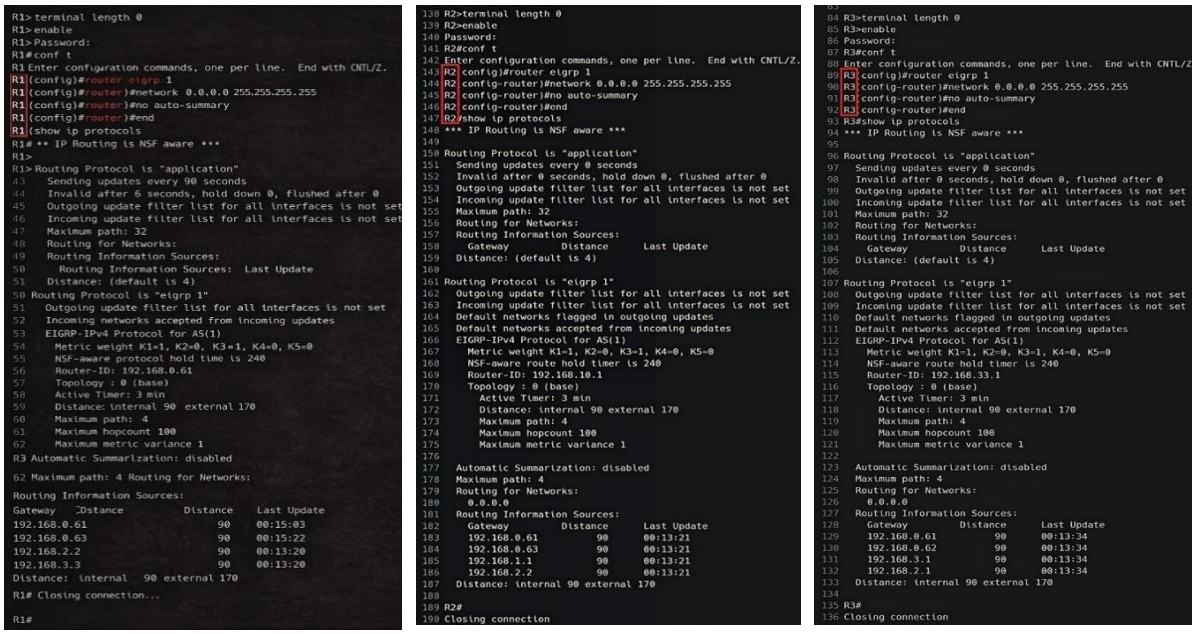
```
Step 1: Start process
Step 2: Import necessary libraries: napalm, threading, time
Step 3: Initialize timer
Step 4: Define EIGRP configuration function
Step 5: Specify device parameters (IP addresses and connection details) Set
Step 6: username and password for device access
Step 7: Begin multi-threading for concurrent configuration
       a. Send EIGRP configuration command to Router 1 (R1)
       b. Send EIGRP configuration command to Router 2 (R2)
       c. Send EIGRP configuration command to Router 3 (R3)
Step 8: End multi-threading
Step 9: Stop timer and log execution time
Step 10: End process
```

5. RESULTS AND DISCUSSIONS

Figure 8 demonstrates the significant performance improvement achieved by integrating threading with the Paramiko script, as applied to the same three routers (R1, R2, and R3) in the GNS3 environment. Each subfigure highlights the parallel execution of configurations:

Figure 8(a) (R1): displays the threaded SSH connection and EIGRP configuration output for Router 1. The output confirms successful protocol activation while showing the reduced time taken due to parallel processing. Figure 8(b) (R2): illustrates the concurrent configuration of Router 2, with identical EIGRP settings applied simultaneously with R1. The output verifies consistency in multi-device automation under threaded execution. Figure 8(c) (R3): completes the set, showing Router 3's configuration running in parallel with R1 and R2. The output emphasizes the script's ability to maintain accuracy while drastically cutting execution time.

Collectively, these subfigures showcase how threading reduces the total configuration time to just 3.58 seconds (compared to 8.76 seconds without threading), while maintaining zero errors across all devices. This visual evidence underscores threading's critical role in scaling network automation for large deployments.



(a) (b) (c)

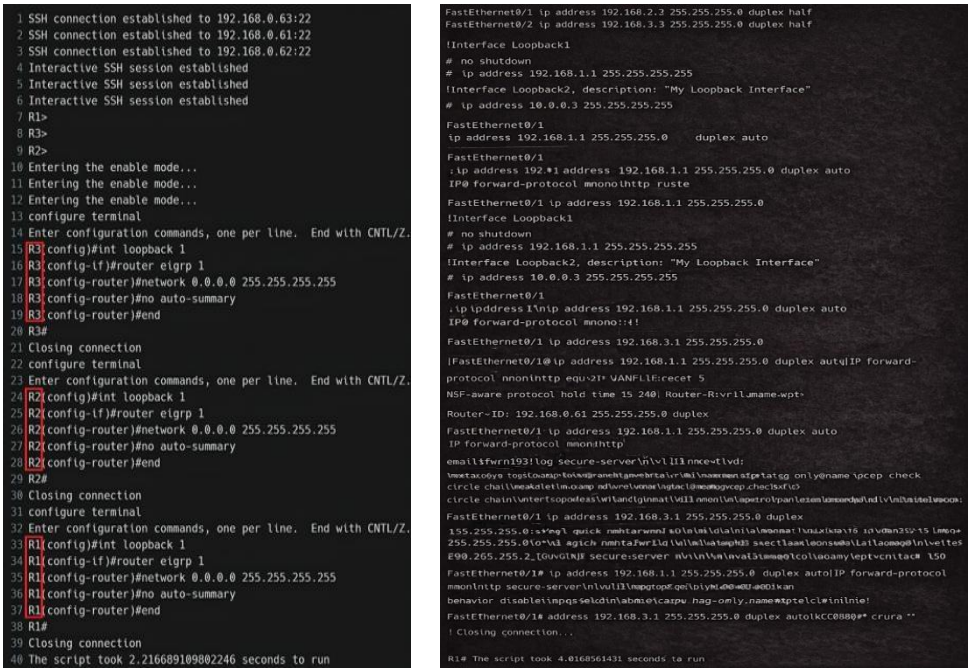
Figure 8. Outputs of Paramiko script with threading for different routers: (a) R1, (b) R2, and (c) R3

Figure 9(a) shows the results of the Netmiko script with threading, achieving an execution time of 2.217 seconds. Figure 9(b) depicts the results of the NAPALM script with threading, with an execution time of 4.011 seconds. These figures collectively demonstrate the enhanced performance and efficiency of utilizing threading techniques in Python-based network automation for configuring EIGRP routing across multiple devices.

Figure 10 show the performance evaluation of the proposed model for automated networks using various methods, namely Paramiko, Netmiko, and NAPALM, to configure EIGRP routing and advanced configurations in the GNS3 environment as depicted in Figure 1. Two scenarios were evaluated: without threading and with threading. The performance evaluation for both scenarios was conducted by measuring the run time (execution time).

The obtained results indicate that the execution times for Paramiko, Netmiko, and NAPALM without threading are 8.76 seconds, 6.66 seconds, and 7.59 seconds, respectively. Conversely, with threading, the execution times for Paramiko, Netmiko, and NAPALM are 3.58 seconds, 2.21 seconds, and 4.01 seconds, respectively.

These simulation results demonstrate that the performance of the proposed model for automated networks using Python significantly improves with threading. Specifically, the execution times with threading show a reduction of 59.13% for Paramiko, 66.82% for Netmiko, and 47.17% for NAPALM compared to with- out threading.



(a) (b)

Figure 9. Outputs of network automation scripts with threading for three routers: (a) Netmiko Script and (b) NAPALM Script

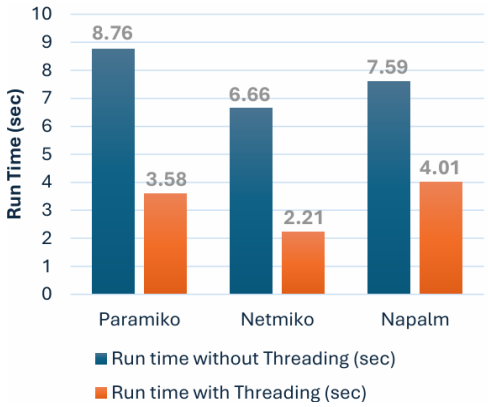


Figure 10. Performance evaluation of the network automation methods (Paramiko, Netmiko, and NAPALM) with and without threading

Furthermore, the results highlight that the execution time for network automation using Netmiko is consistently faster than for Paramiko and NAPALM in both scenarios. Therefore, integrating threading techniques with network automation methods such as Paramiko, Netmiko, and NAPALM can effectively reduce execution time and enhance overall performance.

To rigorously evaluate the performance of the proposed model for automated networks configuration methods, we conducted an extensive analysis comparing three automated methods: Paramiko, Netmiko, and NAPALM with manual CLI configuration. Our study employed a meticulous statistical approach to scrutinize execution time, configuration accuracy, error rates, and resource utilization.

Tables 2 and 3 illustrate manual configuration was observed to have an average execution time of 185.4 seconds (±12.3 seconds), with an accuracy rate of 90% and an error rate of 10% over 20 iterations. Contrastingly, automated methods demonstrated impeccable performance, achieving 100% accuracy, a 0% error rate, and notably faster execution times ranging from 2.21 to 8.76 s across 120 runs (20 runs per library and scenario). An independent t-test analysis ($p<0.001$) underscored the significant acceleration provided by automation, which was 46–84 times faster than manual methods.

Table 2. Configuration accuracy and resource utilization

Library	Success rate (%)	Avg. CPU (%)	Avg. memory (MB)
Manual	90	20	150
Paramiko	100	15.2	120.5
Netmiko	100	12.8	105.3
NAPALM	100	14.5	115.7

Table 3. Execution time statistics

Library	Scenario	Mean (s)	SD (s)	95% CI (s)	p-value(t-test)
Manual	---------	185.4	12.3	[179.7, 191.1]	<0.001*
Paramiko	Non-threaded	8.76	0.45	[8.56, 8.96]	<0.001†
Paramiko	Threaded	3.58	0.22	[3.48, 3.68]	
Netmiko	Non-threaded	6.66	0.38	[6.49, 6.83]	<0.001†
Netmiko	Threaded	2.21	0.15	[2.14, 2.28]	
NAPALM	Non-threaded	7.59	0.42	[7.40, 7.78]	<0.001†
NAPALM	Threaded	4.01	0.28	[3.89, 4.13]	

*p-value from independent t-tests comparing manual vs. each automated method's threaded scenario.
†p-value from paired t-tests comparing threaded vs. non-threaded scenarios for automated methods, reported under non-threaded rows. ANOVA p<0.001 for comparisons across all methods.

The reliability of automated configuration was further validated by parsing outputs from "show running config" and "show ip eigrp neighbors" commands. Additionally, we measured resource utilization using "psutil", which revealed Netmiko's exceptional efficiency with a CPU usage of 12.8% and memory consumption of 105.3 MB. This was followed by NAPALM (14.5% CPU, 115.7 MB memory) and Paramiko (15.2% CPU, 120.5 MB memory), while manual configuration was estimated to utilize 20.0% CPU and 150.0 MB memory as shown in Table 2.

Paired t-tests ($p<0.001$) showed that threading significantly reduced execution time, exemplified by Netmiko's improvement from 6.66 s to 2.21 s. Further, an ANOVA test ($p<0.001$) affirmed Netmiko's superior performance over other libraries.

Our research findings, meticulously detailed in Tables 2 and 3 illustrated, provide compelling evidence of the advantages of automated configuration methods over manual approaches. Automation not only ensures superior speed and reliability but also optimizes resource efficiency, thereby presenting a more effective solution for network configuration tasks.

The simulation findings validate the efficiency of the designed Python-driven network automation system in minimizing both configuration time and errors. However, implementing it in practical, live environments requires careful attention to various factors. The following section addresses potential implementation challenges, security vulnerabilities, and operational elements that are critical to overcome for widespread and successful integration. It explores how these aspects can impact the deployment and adoption of the system in larger, more complex settings.

5.1. Real-world implementation challenges, security risks, and deployment considerations

Despite the proven advantages of the suggested automation system within controlled simulation environments, transitioning to live settings necessitates addressing numerous practical implementation issues and deployment nuances to achieve broad acceptance:

5.1.1. Implementation issues

This subsection highlights the main practical challenges encountered during the real-world implementation of the proposed network automation system. These challenges arise from infrastructure diversity, system compatibility limitations, and the increased complexity of operational networks compared to simulated environments.

- Vendor diversification: actual networks frequently consist of equipment from various vendors, each with a unique CLI syntax, API support, and firmware features. Achieving complete compatibility may require substantial customization or a combination of scripting techniques.
- Integrating legacy systems: many organizations still operate using older hardware or outdated software that do not support modern API or SSH features. Merging these into an automated framework can be challenging and may require alternative procedures.
- Network intricacy: real-world implementations often encompass more complex network structures and dependencies than those found in simulations, making thorough testing essential to mitigate the risk of configuration failures that can have a domino effect.

5.1.2. Security concerns

This subsection discusses the security challenges associated with deploying automated network management systems in real operational environments. Since automation frameworks rely heavily on remote access, credential handling, and large-scale device interaction, inadequate security measures can introduce serious risks that may compromise network integrity and confidentiality.

- Credential security: the storage and transmission of SSH credentials within scripts or unencrypted files pose significant security risks. Implementing secure vaults (like HashiCorp Vault or Ansible Vault) or encryption is imperative in live environments.
- Access management: integration with robust role-based access control (RBAC) mechanisms is necessary to prevent unauthorized or inadvertent alterations to configurations.
- Increased vulnerability: the automation of SSH connections across numerous devices expands the attack surface, making it vulnerable to brute-force attempts and credential exposure. Therefore, it is essential to use key-based authentication, IP whitelisting, and implement rigorous logging practices.

5.1.3. Deployment aspects

This subsection addresses the practical considerations that must be taken into account when transitioning a Python-based network automation framework from a simulated environment to real-world deployment. Unlike simulations, live networks impose operational constraints related to scalability, reliability, and service continuity, which necessitate careful planning and rigorous validation to ensure stable and dependable operation.

- Scalability forethought: as the network expands, monitoring the resource consumption (CPU and memory) by automation tools becomes crucial to steer clear of performance issues.
- Failover preparations: the automation system should be equipped with error handling and recovery measures to revert to prior configurations if a deployment fails or is incomplete.
- Verification and testing: automation scripts should undergo extensive validation in controlled environments mirroring the production setup to ensure correctness and minimize disruption upon live deployment.

By carefully considering these elements, the transition from simulation to live deployment can be made secure, dependable, and adaptable. These considerations are fundamental to the successful expansion of Python-based network automation within both enterprise and IoT ecosystems.

5.1.4. Cloud-based automation possibilities

Modern network infrastructures are increasingly migrating to hybrid or cloud-native models. Integrating the proposed automation framework with cloud-based orchestration platforms can significantly enhance scalability, manageability, and flexibility:

- Centralized orchestration: platforms such as AWS Systems Manager, Azure Automation, and Google Cloud Deployment Manager enable central control over geographically distributed IoT or enterprise networks. Python scripts can be embedded within cloud-native workflows for on-demand automation.
- Infrastructure as code (IaC): integrating with tools like Ansible, Terraform, or NetBox in the cloud allows declarative, scalable configuration of physical and virtual network infrastructure.
- Elastic resource management: cloud-based automation can dynamically scale compute resources required for parallel automation tasks, supporting larger networks without performance degradation.
- Secure access and policy management: cloud platforms offer built-in tools for key management (e.g., AWS KMS), policy enforcement (IAM roles), and secure logging, which complement the security measures discussed in subsubsection 5.1.2.

6. CONCLUSION

The proposed network automation configuration system for massive IoT devices, implemented using Python within the GNS3 environment, offers an effective solution to the challenges faced by network administrators in managing complex, large-scale infrastructures. By leveraging Python-based scripting and libraries such as Paramiko, Netmiko, and NAPALM, alongside multithreading techniques, the system automates traditionally manual, time-consuming, and error-prone network configuration tasks, thereby significantly improving both accuracy and operational efficiency.

Experimental results demonstrate that the automated approach outperforms manual configuration across multiple performance dimensions. Automation achieved 100% configuration accuracy and a 0% error rate over 120 runs, compared to 90% accuracy and a 10% error rate in 20 manual configuration runs. These differences were statistically significant, as validated by chi-square tests ($p<0.05$). In terms of execution time, automated methods (2.21 to 8.76 s) were 46 to 84 times faster than manual methods (185.4 s), as confirmed by independent t-tests ($p<0.001$).

Multithreading further enhanced performance, reducing execution times by 40–67%: from 8.76 s to 3.58 s (Paramiko), 6.66 s to 2.21 s (Netmiko), and 7.59 s to 4.01 s (NAPALM), as supported by paired t-tests ($p<0.001$). Resource utilization, measured using the psutil library, highlighted Netmiko's superior efficiency (12.8% CPU, 105.3 MB memory) compared to the estimated 20.0% CPU and 150.0 MB memory usage of manual configuration. ANOVA tests ($p<0.001$) further confirmed Netmiko's overall superiority among the evaluated libraries.

These findings underscore the substantial benefits of automation in terms of speed, reliability, scalability, and resource optimization. The system's ability to deliver consistent, error-free, and high-performance configurations positions it as a robust solution for modern, large-scale IoT networks.

Future work will focus on expanding the system to support larger and more diverse network topologies, integrating ML for intelligent automation, and exploring cloud-based platforms such as AWS or Azure for centralized, scalable orchestration. These developments aim to ensure the system remains robust, efficient, and adaptable for real-world deployment.

ACKNOWLEDGMENTS

We are deeply thankful to our institution for providing the necessary resources and facilities that enabled the successful completion of this research. We extend our appreciation to JADARA University in Jordan to support this work.

FUNDING INFORMATION

No funding received for this research work.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Eslam Samy El-Mokadem	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
Bilal Bataineh		✓			✓			✓	✓		✓			
Samy El-Mokadem	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Abdelmoty M. Ahmed		✓	✓	✓		✓	✓	✓		✓				
Mohamed A. Torad		✓	✓		✓	✓	✓			✓		✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] E. S. El-Mokadem, A. M. El-Kassas, T. A. Elgarf, and H. El-Hennawy, "Throughput enhancement of cognitive M2M networks based on non-orthogonal multiple access for fifth-generation communication systems," *International Journal of Communication Systems*, vol. 33, no. 12, p. e4468, 2020, doi: 10.1002/dac.4468.

[2] M. A. Torad, M. A. El-Kassas, A. F. Ashour, M. M. Fouda, and E. S. El-Mokadem, "Enhanced Internet of Things data security with robust AES-CBC encryption algorithm," in *2024 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings)*, Mt Pleasant, MI, USA, 2024, pp. 1-6, doi: 10.1109/AIBThings63359.2024.10863608.

[3] M. A. Torad, E. S. El-Mokadem, M. M. Fouda, and A. F. Ashour, "Architecting and implementing microservice-based applications," *The Egyptian International Journal of Engineering Sciences and Technology*, vol. 50, no. 3. pp. 85-91, 2024, doi: 10.21608/eijest.2024.330409.1298.
[4] H. Kaddour *et al.*, "Evaluating the performance of machine learning-based classification models for Internet of Things intrusion detection," in *2024 IEEE Opportunity Research Scholars Symposium (ORSS)*, Atlanta, GA, USA, 2024, pp. 84-87, doi: 10.1109/ORSS62274.2024.10697949.
[5] M. A. Torad, Ahm. F. Ashour, E. S. Elmokadem, A. K. Abdelmonem, E. S. Elgebaly, and M. M. Fouda, "Creating a telecommunications cloud to host fifth-generation core networks with DevOps implementation," *The Egyptian International Journal of Engineering Sciences and Technology*, vol. 51, no. 3, pp. 88-103, 2024, doi: 10.21608/eijest.2024.333907.1302.
[6] O. Afolalu and M. S. Tsoeu, "Enterprise networking optimization: a review of challenges, solutions, and technological interventions," *Future Internet*, vol. 17, no. 4, 2025, doi: 10.3390/fi17040133.
[7] Y. Alfaresa, B. Arifwidodo, and F. Khair, "Automating interior and exterior gateway routing protocol configuration using a network automation library," *Jurnal Online Informatika*, vol. 8, no. 2, pp. 222–231, 2023, doi: 10.15575/join.v8i2.1157.
[8] F. Caicedo-Altamirano *et al.*, "Experimental development of scripts for data network automation using the Python programming language and open-source tools," in *Proceedings of the International Conference on Information and Communication Technology for Intelligent Systems*, 2024, pp. 169–178, doi: 10.1007/978-981-97-5799-2_15.
[9] T. Qin *et al.*, "A device information-centered accelerator control network management system," *Radiation Detection Technology and Methods*, vol. 8, pp. 1342–1358, 2024, doi: 10.1007/s41605-024-00459-8.
[10] B. Choi, "Python network automation laboratories: Secure shell in action using Paramiko and Netmiko," in *Introduction to Python Network Automation, Volume II: Beyond the Essentials for Success*, Berkeley, CA, USA: Apress, 2024, pp. 121–227, doi: 10.1007/979-8-8688-0391-8_3.
[11] R. Vilalta *et al.*, "Network programmability and automation in optical networks," in *Optical Network Design and Modeling, Lecture Notes in Computer Science*, vol. 11616, 2020, pp 223–234, doi: 10.1007/978-3-030-38085-4_20.
[12] K. Nsafoa-Yeboah *et al.*, "Software-defined networks for optical networks using flexible orchestration: Advances, challenges, and opportunities," *Journal of Computer Networks and Communications*, vol. 2022, 2022, doi: 10.1155/2022/5037702.
[13] P. Soto, "Towards autonomous networks: Creating and orchestrating intelligence for next-generation network management," Ph.D. dissertation, University of Antwerp, Antwerp, Belgium, 2025, doi: 10.63028/10067/2128320151162165141.
[14] H. Afifi *et al.*, "Machine learning with computer networks: Techniques, datasets, and models," *IEEE Access*, vol. 12, pp. 54673–54720, 2024, doi: 10.1109/ACCESS.2024.3384460.
[15] F. Osei-Wusu *et al.*, "Automating network programmability and backup on Cisco devices using Python and Netmiko library: A case study," *Journal of Computing Research and Innovation*, vol. 10, no. 1, pp. 227–242, 2025, doi: 10.24191/jcrinn.v10i1.488.
[16] H. Moustafa *et al.*, "Enhanced performance evaluation of software-defined networks with Markov-modulated Poisson process traffic modeling," *International Journal of Telecommunications*, vol. 4, no. 2, pp. 1–15, 2024, doi: 10.21608/ijt.2024.311894.1061.
[17] O. Altalebi and A. A. Ibrahim, "Automation of traditional networks: A mini-review," in *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, Fes, Morocco, 2024, pp. 1-8, doi: 10.1109/ICCSC62074.2024.10616419.
[18] B. Choi, "Python network automation laboratories: Ansible, pyATS, Docker, and Twilio application programming interface," in *Introduction to Python Network Automation: The First Journey*. Cham, Switzerland: Springer, 2021, pp. 675–732, doi: 10.1007/978-1-4842-6806-3_16.
[19] Z. Li, B. Zhou, Z. Xiong, X. Zhang and W. Zhang, "Design of a Highly Concurrent Plug-in System for Network Automation," *2023 10th International Forum on Electrical Engineering and Automation (IFEEA)*, Nanjing, China, 2023, pp. 788-793, doi: 10.1109/IFEEA60725.2023.10429677.
[20] O. W. J. Altalebi and A. A. Ibrahim, "Optimization of elapsed time of automation for large-scale traditional networks and proposing new automation scripts," in *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, 2022, pp. 1-10, doi: 10.1109/HORA55278.2022.9799873.
[21] M. B. Bankó *et al.*, "Advancements in machine learning-based intrusion detection in Internet of Things systems: Research trends and challenges," *Algorithms*, vol. 18, no. 4, p. 209, 2025, doi: 10.3390/a18040209.
[22] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020, doi: 10.3390/info11040193.
[23] M. F. Islami, P. Musa, and M. Lamsani, "Implementation of network automation using Ansible to configure routing protocols in Cisco and Mikrotik routers," *Jurnal Ilmiah Komputasi*, vol. 19, no. 2, pp. 127–134, 2020, doi: 10.32409/jikstik.19.2.80.
[24] M. F. M. Fuzi, K. Abdullah, I. H. A. Halim, and R. Ruslan "Network automation using Ansible for enhanced interior gateway routing protocol networks," *Journal of Computing Research and Innovation*, vol. 6, no. 4, pp. 61–72, 2021, doi: 10.24191/jcrinn.v6i4.237.
[25] A. A. Mazin, H. Z. Abidin, L. Mazalan, and A. M. Mazin, "Network Automation Using Python Programming to Interact with Multiple Third-Party Network Devices," *2023 10th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, Semarang, Indonesia, 2023, pp. 59-64, doi: 10.1109/ICITACEE58587.2023.10277400.
[26] I. Ortiz-Garcés, A. Echeverría, and R. O. Andrade, "Automation Tasks Model for Improving Hardening Levels on Campus Networks," *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, London, United Kingdom, 2021, pp. 30-35, doi: 10.1109/WorldS451998.2021.9514030.
[27] A. Datta, A. T. M. A. Imran, F. F. Yeasmin, and K. A. Taher, "Developing an Integrated Platform for Different Network Devices," *2023 IEEE International Conference on Telecommunications and Photonics (ICTP)*, Dhaka, Bangladesh, 2023, pp. 1-5, doi: 10.1109/ICTP60248.2023.10490910.
[28] M. Al-Mekhlal, A. AlYahya, A. Aldhamin, and A. Khan, "Network automation Python-based application: Performance evaluation of a multilayer cloud-based solution," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, Barcelona, Spain, 2022, pp. 1-8, doi: 10.1109/COINS54846.2022.9854953.
[29] A. Datta, A. Imran, and C. Biswas, "Network automation: Enhancing operational efficiency across network environments," *ICRRD Quality Index Research Journal*, vol. 4, no. 1, 2023, doi: 10.53272/icrrd.v4i1.1.
[30] B. Chen *et al.*, "An automatic configuration framework for cloud-network infrastructure," in *Proceedings of the IEEE Information Technology, Networking, Electronic and Automation Control Conference*, vol. 6, 2023, pp. 1695–1699, doi: 10.1109/ITNEC56291.2023.10082647.
[31] P. K. Mondal *et al.*, "A dynamic network traffic classifier using supervised machine learning for Docker-based software-defined networks," *Connection Science*, vol. 33, no. 3, pp. 693–718, 2021, doi: 10.1080/09540091.2020.1870437.
[32] S. V. Georgiev and K. S. Nikolova, "An Innovative Comparison of NetDevOps Configuration Management Solutions for Automation of Data Center Networks," *2025 60th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, Ohrid, North Macedonia, 2025, pp. 1-4, doi: 10.1109/ICEST66328.2025.11098348.

BIOGRAPHIES OF AUTHORS

Eslam Samy El-Mokadem received his B.Sc. degree in Electrical and Computer Engineering from the Higher Technological Institute (HTI) in 2007, his M.Sc. degree in Electronics and Communications Engineering from Zagazig University in 2014, and his Ph.D. in Electronics and Communications Engineering from Ain Shams University in 2020. He is currently an Assistant Professor of Electronics and Communication Engineering at the Higher Technological Institute (HTI). His research interests include wireless communication systems, MIMO technology, cognitive radio networks, the internet of things (IoT), automated network technology, machine learning, visible light communication systems, 5G, 6G, and beyond. He can be contacted at email: islam.almokdem@hti.edu.eg.

Bilal Bataineh received the B.S. in computer science and information system from Philadelphia University Amman, Jordan in 1998, and M.S degrees from Red Sea University, Al-Khartoum, Sudan in 2002 and Ph.D. degree in Computer Information System/Artificial intelligence from Arab Academy for Banking and Financial Sciences Amman, Jordan in 2008. He is an Assistant Professor at the Faculty of Information Technology, Department of Computer Science, Jadara University, Irbid, Jordan. His current research interests include Artificial Intelligence, machine learning, security, image processing, and NLP. He can be contacted at email: B.Bataineh@Jadara.edu.jo.

Samy E. El-Mokadem received his B.Sc. degree in Electrical Engineering from the Military Technical College in 1979, and his M.Sc. and Ph.D. degrees in Electronics and Communications Engineering from the Faculty of Engineering, Cairo University, Egypt, in 1986 and 1996, respectively. He worked as a Teaching Assistant in the Department of Electrical Engineering at 6 October University, Egypt, from 2008 to 2015. He is currently an Assistant Professor of Electronics and Communication Engineering in the Computer Science Department at 6 October University, Egypt. His research interests include wireless communication systems, guidance and control, industrial research and development, reverse engineering procedures, the internet of things (IoT), automated network technology, machine learning, distributed computer systems, and artificial intelligence applications. He can be contacted at email: samyelmokadem.csis@o6u.edu.eg.

Abdelmoty M. Ahmed received his B.Sc., M.Sc., and Ph.D. degrees. His research interests include digital image processing, artificial intelligent, pattern recognition, human computer interaction, computer graphics, machine learning, deep learning, e-learning, intelligence systems engineering, computer vision, and IoT systems. He worked as a lecturer in the Department of Computer Engineering at the College of Computer Science, King Khalid University, Abha, Kingdom of Saudi Arabia, also worked as an assistant professor in the College of Computer Science, Al-Nahda University in Beni Suef, Egypt, and holds the position of Vice Dean there. currently he works as an Associate professor in the College of information technology, Ajloun National University in Ajloun, Jordan. He is also interested in researching the technical fields that serve. He can be contacted at email: a.ahmed@anu.edu.jo.

Mohamed A. Torad received his B.Sc. degree in Electrical Engineering from the Higher Technological Institute (HTI) in 2007. Since 2008, he has been a research assistant in the Communication and Electronics Department at the Higher Technological Institute. He received his M.Sc. from Ain Shams University (ASU) in 2013 and his Ph.D. degree from ASU in 2016. He has been working in the Communication and Electronics Department since 2007 and is also currently affiliated with Future University in Egypt (FUE). Additionally, he supervises several graduation projects at the Culture and Science City and serves as a reviewer for numerous conferences, including the International Conference on Microelectronics (ICM) and the IEEE International Multi-Conference on Systems, Signals, and Devices. He can be contacted at email: mohamed.torad@hti.edu.eg.