

Design and Implementation of an Automated Spine-and-Leaf Data Center using Infrastructure as Code

Khaled Taisir Elgdamsi^{1*}, Shahed Giuma Ashour²

^{1,2}Electrical and Electronic Department, Faculty of Engineering, University of Tripoli, Libya

تصميم وتنفيذ مركز بيانات مؤتمت قائم على معمارية العمود الفقري والأوراق باستخدام البنية التحتية كبرمجيات

خالد تيسير الغدامسي^{1*}، شهد غومة عاشور²
^{1,2}قسم الهندسة الكهربائية والإلكترونية، كلية الهندسة، جامعة طرابلس، ليبيا

*Corresponding author: k.elgdamsi@uot.edu.ly

Received: July 19, 2026

Accepted: September 30, 2026

Published: October 14, 2026

Abstract:

Modern data centers require fast and reliable networks to handle massive amounts of information; thus, traditional three-tier designs are rendered inefficient for heavy east-to-west traffic. To address this, the Spine-and-Leaf architecture has been adopted to provide faster, and highly predictable connectivity. Because the manual configuration of these network switches is slow and prone to human error, an Infrastructure as Code (IaC) approach was utilized to completely automate a modern data center network. A scalable Spine-and-Leaf network was designed in a virtual lab using Arista vEOS switches. A physical underlay was established using OSPF and IS-IS, and a virtual overlay network was created utilizing MP-BGP EVPN and VXLAN technologies. The entire network was provisioned using automation tools, specifically Ansible playbooks and Python scripts. Following successful deployment, key network metrics, including latency, throughput, and jitter, were measured. A comparative analysis was performed between a baseline 2-leaf topology and a scaled 4-leaf topology, proving the automated network can scale smoothly to handle increased traffic without losing performance. Finally, a self-healing automation system was successfully developed. A custom dashboard was engineered to continuously monitor network health in real-time, executing remediation scripts to fix failures within sixty seconds without human intervention.

Keywords: Spine-and-Leaf, Infrastructure as Code, Network Automation, MP-BGP EVPN, VXLAN, OSPF, IS-IS, Self-Healing.

الملخص

تتطلب مراكز البيانات الحديثة شبكات سريعة وموثوقة للتعامل مع كميات هائلة من المعلومات؛ وبالتالي أصبحت التصميمات التقليدية ثلاثية الطبقات غير فعالة لحركة مرور البيانات الكثيفة بين الخوادم. ولمعالجة ذلك، تم اعتماد بنية (spine and leaf) لتوفير اتصال أسرع وقابلية عالية للتنبؤ. ونظراً لأن الإعداد اليدوي للشبكة يعد بطيئاً وعرضة للأخطاء البشرية، فقد تم استخدام نهج البنية التحتية كبرمجة (Infrastructure as Code) لبناء شبكة مركز بيانات حديثة بالكامل. تم تصميم وبناء شبكة قابلة للتطوير في بيئة معملية افتراضية باستخدام محولات Arista vEOS.

تم إنشاء طبقة أساسية مادية قوية باستخدام بروتوكولات التوجيه OSPF و IS-IS، وتم إنشاء شبكة افتراضية مرنة فوق هذه الطبقة المادية باستخدام تقنيات MP-BGP EVPN و VXLAN. للقضاء على الأخطاء اليدوية وتوفير الوقت، تم إعداد الشبكة بأكملها باستخدام أدوات برمجية، وتحديد ملفات Ansible و Python. بعد بناء الشبكة بنجاح، أجريت مقارنة بين بنية أساسية مكونة من محولين وبنية موسعة مكونة من 4 محولات.

أخيراً، تم بنجاح تطوير نظام آلي ذاتي الإصلاح لضمان مرونة الشبكة. تم تصميم لوحة تحكم مخصصة لمراقبة أداء الشبكة باستمرار. في حالة حدوث عطل مفاجئ، يتم اكتشاف المشكلة تلقائياً بواسطة النظام، ثم يتم تنفيذ النصوص البرمجية لمعالجة المشكلة في غضون ستين ثانية دون الحاجة إلى أي تدخل بشري.

الكلمات المفتاحية: Spine-and-Leaf, البنية التحتية كبرمجة, أتمتة الشبكات, MP-BGP EVPN, VXLAN, OSPF, IS-IS, الإصلاح الذاتي.

1. INTRODUCTION

Data centers are considered the heart of modern computing, hosting the applications, services, and data that power daily operations [4]. Historically, data center networks were built utilizing a traditional three-tier design consisting of core, aggregation, and access layers. However, as network traffic grew especially the traffic moving between servers within the data center, known as east-to-west traffic this traditional design began to cause bottlenecks, delays, and unequal paths [4]. To solve these performance issues, Spine-and-Leaf architecture was widely adopted by the networking industry [1]. This modern design connects every leaf switch to every spine switch, providing a direct, fast, and predictable path between any two servers.

To make this physical network even more flexible, advanced virtualization technologies such as VXLAN and MP-BGP EVPN are currently utilized [1], [4]. These protocols create a flexible overlay network on top of a solid physical underlay network, which is driven by routing protocols such as OSPF or IS-IS.

While these technologies are powerful, configuring them manually across dozens of switches is slow, repetitive, and highly prone to human error [2]. To overcome these limitations, Infrastructure as Code (IaC) and network automation are adopted [3]. Through the utilization of tools such as Ansible and Python, scripts are written to automatically configure devices, verify connections, and monitor network health [1], [3].

The primary goal of this research is to implement, configure, and automate a modern data center network. Furthermore, network stability is ensured by enforcing a golden configuration state, wherein unauthorized modifications are automatically detected and rolled back. This shift from manual configuration to automated deployment and continuous state remediation is considered essential for the construction of the scalable, self-healing networks required in the contemporary telecommunications and IT industries [2].

2. LITERATURE REVIEW

In recent advancements within data center network management, an increasing focus is placed on the replacement of manual configuration with robust automation. While the theoretical benefits of network automation are highlighted by previous studies, its practical application is advanced by this research through a comprehensive Infrastructure as Code (IaC) strategy. The foundational context and related research driving this transition are summarized as follows:

- A framework integrating tools such as Python scripting with advanced networking was developed by Kahagalage et al. [1], demonstrating that provisioning times and manual misconfigurations are significantly reduced when MP-BGP EVPN and VXLAN are implemented over a simulated Spine-and-Leaf architecture.
- Building upon the necessity for scalable operations, it is emphasized by W. Rafique and L. Velasco [2] that manual configuration methods are no longer adequate.
- The transition to automated, high-performance fabrics is heavily supported by industry standards. For example, the agentless architecture of Red Hat's Ansible is highlighted as a primary method for consistent multi-vendor automation [3], while the implementation of encapsulation protocols such as VXLAN is supported by Cisco Systems [4].

While the theoretical benefits of network automation are highlighted by previous studies, the practical application in this paper advances beyond the existing literature. A comprehensive IaC strategy is implemented to deploy EVPN-VXLAN Spine-and-Leaf architecture utilizing Arista vEOS switches within PNETLab. Furthermore, an empirical performance analysis of latency, throughput, and jitter is provided, wherein a baseline two-leaf topology is compared to a scaled four-leaf topology. Most notably, a novel autonomous, self-healing mechanism is introduced, failures are detected and resolved without human intervention, and an automated rollback method to rectify configuration drift against a golden state is pioneered, consequently bridging the gap between automated provisioning and dynamic network resilience.

3. MODERN DATA CENTER ARCHITECTURE

Fundamental networking models are utilized to standardize communication and ensure interoperability across diverse computing systems [5]. Historically, enterprise networks and early data centers were built upon a hierarchical three-tier model consisting of core, distribution, and access layers [6]. Within this legacy design, the Spanning Tree Protocol (STP) is utilized to prevent network bridging loops; however, redundant links are inherently blocked by this protocol, effectively wasting up to half of the available physical bandwidth [7]. As internal server-to-server communication, defined as East-West traffic, became dominant, the limitations of the three-tier model were exposed [8]. This dramatic shift in traffic patterns, driven by advanced server virtualization and distributed computing workloads, required continuous, high-speed lateral data exchange [8]. Consequently, the legacy reliance on STP created severe performance bottlenecks, as the forced oversubscription and disabled redundant links could not sustain these massive internal flows [7]. Ultimately, the structural inability of the three-tier hierarchy to provide scalable, non-blocking pathways necessitated a complete architectural paradigm shift to support modern telecommunication demands [6].

3.1 Principles of the Spine-and-Leaf Topology

To overcome the restrictions of traditional hierarchical models, the spine-and-leaf architecture is widely adopted in contemporary data center environments [6]. This strictly two-tier architecture comprises a centralized spine layer and a distributed leaf layer [5]. A fundamental structural rule is enforced wherein every leaf switch maintains a physical connection to every spine switch, while spine switches are strictly prohibited from interconnecting with one another, as demonstrated in Figure (1) [7].

By extending Layer 3 routing down to the edge, traditional Layer 2 spanning tree protocols are entirely eliminated, ensuring that all physical interconnects remain active and unblocked [8]. Consequently, advanced routing mechanisms deploy Equal-Cost Multi-Path (ECMP) forwarding to simultaneously load-balance traffic across all available paths [9]. This deterministic design guarantees a highly predictable, two-hop maximum transmission distance between any two end nodes, providing the robust scalability and performance required for modern, programmable network infrastructures [10].

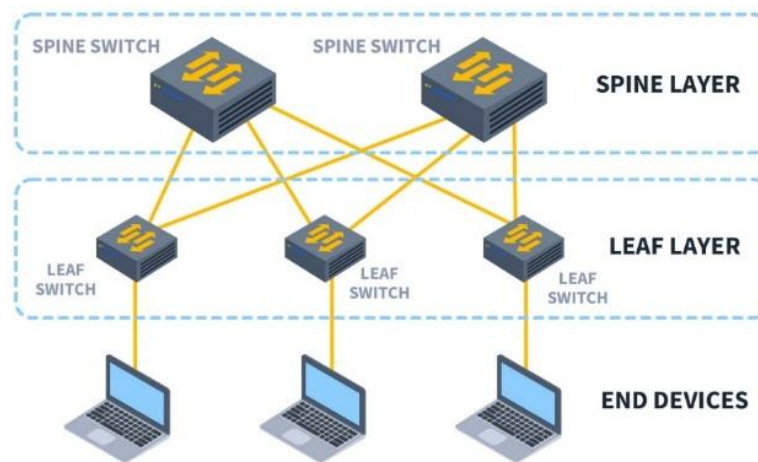


Figure 1: Logical representation of a spine-and-leaf topology.

3.2 NETWORK OPERATIONAL PLANES

To facilitate advanced programmability and dynamic routing capabilities, the internal architecture of modern network switches is logically partitioned into three distinct operational planes [8], [10]:

- **The Control Plane:** This logical entity is utilized by routing protocols to dynamically discover network topologies, mathematically compute optimal forwarding paths, and construct the overarching routing information base [8].
- **The Data Plane:** Responsible for the physical transit of user payloads, this plane leverages the deterministic forwarding tables derived directly from the control plane to switch and route encapsulated packets at hardware speeds [8].
- **The Management Plane:** A secure, programmatic interface is exposed by this plane, enabling the network infrastructure to be provisioned, orchestrated, and continuously audited by centralized automation servers [10].

4. DATA CENTER NETWORKING AND PROTOCOLS

To support the high-speed requirements of a spine-and-leaf data center while guaranteeing strict multi-tenant isolation, the network architecture is fundamentally decoupled into a robust physical underlay and a virtualized logical overlay [11], [12].

4.1 Underlay Network Architecture (OSPF vs. IS-IS)

The primary function of the underlay network is to provide reliable, loop-free IP reachability between all physical switches [13]. By enforcing IP routing across every physical interconnection, legacy Layer 2 loop-prevention protocols are completely eliminated. Two primary link-state interior gateway protocols are evaluated for these underlay operations:

- **Open Shortest Path First (OSPF):** A link-state routing protocol wherein Link-State Advertisements (LSAs) and the Shortest Path First (SPF) algorithm are utilized to mathematically compute optimal forwarding paths. OSPF operates over the network layer and is encapsulated directly inside IP packets [13].
- **Intermediate System to Intermediate System (IS-IS):** A highly scalable link-state protocol that operates directly over the data link layer. Network reachability information is shared using extensible Type-Length-Value (TLV) fields, rendering the protocol highly flexible and inherently isolated from IP-based attack vectors [13].

A comprehensive architectural and functional comparison between OSPF and IS-IS is briefed in Table 1.

Table 1: Comparison of OSPF and IS-IS routing protocols.

Feature	OSPF	IS-IS
OSI Layer	Network Layer (Runs over IP)	Data Link Layer (Runs over Layer 2)
Routing Algorithm	Shortest Path First (SPF)	Shortest Path First (SPF)
Database Packets	Link-State Advertisements (LSAs)	Link State Protocol Data Units (LSPs)
Protocol Flexibility	Low (Needs a new version for IPv6)	High (Uses TLV for easy updates)
Security	Vulnerable to IP-based attacks	Safe from IP attacks

4.2 Overlay Network Virtualization (VXLAN Data Plane)

To overcome the restrictive 4,094 network isolation limit inherent to traditional 802.1Q VLANs, the Virtual eXtensible Local Area Network (VXLAN) standard is deployed [11]. VXLAN functions as a highly scalable data plane virtualization technology, utilizing a Media Access Control in User Datagram Protocol (MAC-in-UDP) encapsulation technique, as illustrated in Figure (2) [11]. By incorporating an extended 24-bit Virtual Network Identifier (VNI) within the header, the logical segmentation capacity of the data center is exponentially increased to support approximately 16.7 million unique tenant networks [11]. The physical switching devices responsible for dynamically executing this encapsulation and decapsulation process at the leaf edge are defined as VXLAN Tunnel Endpoints (VTEPs) [10], [11].

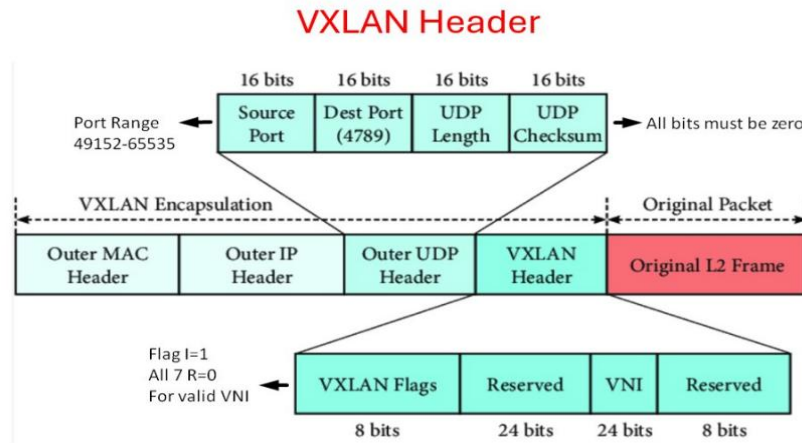


Figure 2: The VXLAN MAC-in-UDP encapsulation header structure.

4.3 Overlay Control Plane (MP-BGP EVPN)

While basic VXLAN inherently relies on inefficient data plane multicast flooding to resolve remote MAC addresses [11], this overhead is entirely eliminated through the deployment of an Ethernet Virtual Private Network (EVPN). EVPN functions as an advanced extension of the Multiprotocol Border Gateway Protocol (MP-BGP), specifically engineered to serve as a robust control plane for VXLAN fabrics [12]. By utilizing BGP update messages, the forwarding tables of all participating leaf switches are proactively populated with reachability information prior to the transmission of any user payloads [12].

To guarantee massive scalability and reduce peering complexity, the spine switches are architected to operate as BGP Route Reflectors [9]. Consequently, when a new local MAC address is learned by a leaf switch, the routing update is transmitted exclusively to the spine layer. The spine subsequently reflects this information to all other leaf nodes, thereby significantly minimizing control plane overhead and ensuring highly efficient MAC address distribution [9], [12].

5. INFRASTRUCTURE AS CODE AND PROGRAMMABILITY

Traditionally, network infrastructure was provisioned manually via device-specific Command-Line Interfaces (CLIs) [10]. However, this imperative approach is highly susceptible to human error and lacks the agility required to support the rapid scaling and dynamic workload shifts of modern data center environments [2], [10].

To resolve these operational bottlenecks, Infrastructure as Code (IaC) paradigms have been widely adopted across the industry [13]. By representing complex network topologies as machine-readable definition files, IaC fundamentally transitions network engineering from a fragile imperative process to a robust, automated declarative model [10], [13]. This programmatic approach ensures that configurations are highly consistent, easily replicable, and capable of being pushed to dozens of switches simultaneously without manual intervention..

5.1 Ansible Automation Architecture

Ansible is deployed as the primary automation engine to provision the network underlay [13]. Operating on an agentless architecture, proprietary software is not required to be installed on the target network devices [3]. Instead, standard Secure Shell (SSH) connections are established over an out-of-band management network [10]. Configurations are written in human-readable YAML "playbooks" [3]. These playbooks declaratively map the desired state of the physical interfaces, point-to-point subnets, and routing protocols, allowing identical configurations to be pushed uniformly across multiple device groups [10], [13].

5.2 Python and the Netmiko Framework

While declarative playbooks efficiently handle static provisioning, high-level programmatic frameworks are often required for complex, iterative overlay deployments [10]. The Python programming language is heavily utilized for this purpose, specifically leveraging the Netmiko library to interact with network infrastructure [10], [13]. Netmiko effectively abstracts the multi-vendor complexities associated with establishing and managing secure SSH terminal sessions across diverse operating systems [10].

Within this programmatic framework, parameterized Python scripts are engineered to iterate through relational data dictionaries. This approach allows the automation server to programmatically push complex logical

configurations such as Symmetric Integrated Routing and Bridging (IRB) instances, Virtual Routing and Forwarding (VRF) mappings, and localized BGP peering consistently and rapidly across the entire data center fabric [13].

5.3 Automated Task Scheduling

To guarantee continuous operational availability and facilitate dynamic self-healing mechanisms, network automation scripts must be executed systematically without requiring manual intervention [1], [2]. To achieve this autonomous execution, the Cron daemon is utilized within the centralized Linux environment as a highly reliable, time-based job scheduler [10], [13].

Scheduled Cron jobs are configured on the automation server to automatically trigger comprehensive health-check and programmatic remediation scripts at strictly defined intervals. This seamless integration of time-based scheduling and responsive scripting yields a fully self-sustaining, closed-loop network architecture capable of autonomous fault resolution [1], [13].

6. VIRTUAL EXPERIMENTAL ENVIRONMENT

To construct, automate, and empirically evaluate the proposed network topology without the financial constraints of physical hardware, a comprehensive virtualized emulation environment was established. The underlying hardware infrastructure was hosted locally on a dedicated workstation equipped with a multi-core processor and 16 GB of Random Access Memory (RAM).

The software foundation was constructed utilizing VMware Workstation Pro as the primary hypervisor. Within this hypervisor, the PNETLab emulation platform was deployed to host the individual network nodes. The data center switching fabric was constructed utilizing Arista vEOS images, which provide execution and feature parity with physical hardware for highly accurate representation.

Finally, a centralized Ubuntu Linux virtual machine was deployed to serve as the out-of-band automation controller. This centralized server was pre-configured with Python 3, the Netmiko library, and the Ansible engine to fully orchestrate and manage the programmable fabric [10].

7. METHODOLOGY AND SYSTEM IMPLEMENTATION

The methodology for deploying automated fabric followed a strict, phased approach to ensure configuration accuracy and protocol stability. The complete sequence of these implementation steps, from the initial virtual setup to the final automated health checks, is illustrated in the project workflow detailed in Figure (3).

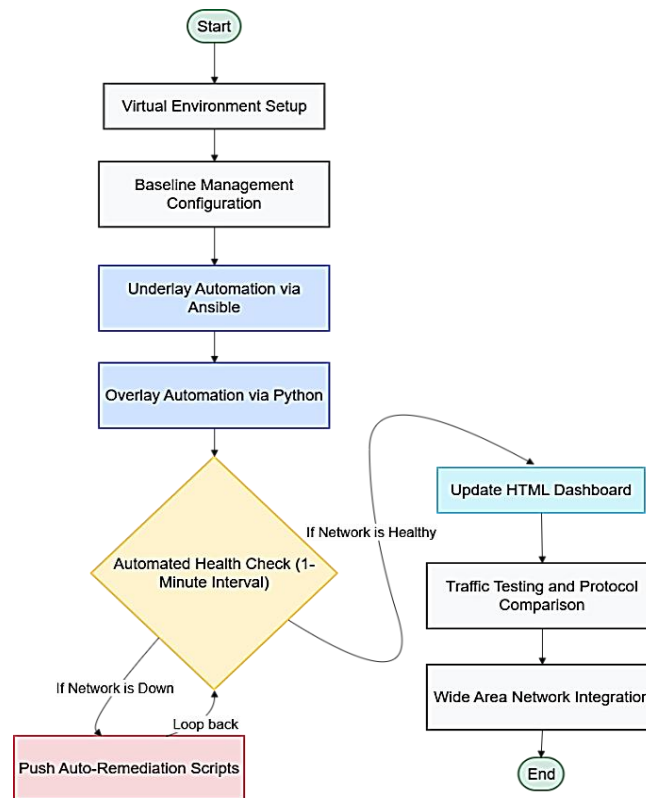


Figure 3: Project implementation and testing workflow.

7.1 Virtual Environment Setup and IP Addressing

To construct and empirically evaluate the proposed network topology without the financial constraints of physical hardware, a comprehensive emulation environment was established utilizing VMware Workstation and the PNETLab platform. The data center switching fabric was deployed utilizing Arista vEOS images to ensure execution and feature parity with physical hardware switches.

To enable the centralized Linux server to push automation scripts to the fabric, an out-of-band management network was established with static IPv4 addresses assigned to the mgmt1 interfaces of all spine and leaf switches. Concurrently, unique loopback interfaces were assigned to identify each device within the routing control plane and to serve as the Virtual Tunnel Endpoints (VTEPs) for the logical overlay. This consolidated foundational addressing scheme is detailed in Table (2).

Table 2: Out-of-Band Management and Control Plane Loopback Addressing.

Device Type	Device Name	Management IP Address	Loopback Address
Spine	Spine 1	10.245.20.11 /24	1.1.1.1 /32
Spine	Spine 2	10.245.20.12 /24	1.1.1.2 /32
Leaf	Leaf 1	10.245.20.13 /24	2.2.2.1 /32
Leaf	Leaf 2	10.245.20.14 /24	2.2.2.2 /32
Leaf	Leaf 3	10.245.20.15 /24	2.2.2.3 /32
Leaf	Leaf 4	10.245.20.16 /24	2.2.2.4 /32

Finally, specific Virtual Local Area Networks (VLANs) and their corresponding Anycast Gateway subnets were provisioned across the leaf switches to establish multi-tenant isolation boundaries as outlined in Table (3).

Table 3: Tenant VLAN Distribution.

Device Name	Assigned VLANs	Gateway Subnet
Leaf 1	VLAN 10	10.0.10.254 /24
Leaf 2	VLAN 10, VLAN 20	10.0.10.254 /24, 10.0.20.254 /24
Leaf 3	VLAN 10, VLAN 20	10.0.10.254 /24, 10.0.20.254 /24
Leaf 4	VLAN 20	10.0.20.254 /24

7.2 Network Topology Design

The physical architecture was constructed utilizing a non-blocking Clos spine-and-leaf topology. This strictly two-tier design ensures that every leaf switch maintains a direct physical connection to every spine switch, while interconnects between switches of the same tier are strictly prohibited. This deterministic framework guarantees a maximum two-hop transmission distance between any two end hosts, optimizing the fabric for high-bandwidth east-to-west data center traffic.

While the initial baseline provisioning and validation were conducted on a two-leaf framework, the physical fabric was ultimately expanded to a four-leaf design. This scaling was executed to accurately measure routing protocol stability, link convergence, and control plane overhead under expanded physical conditions. The complete logical and physical layout of this scaled architecture is illustrated in Figure (4).

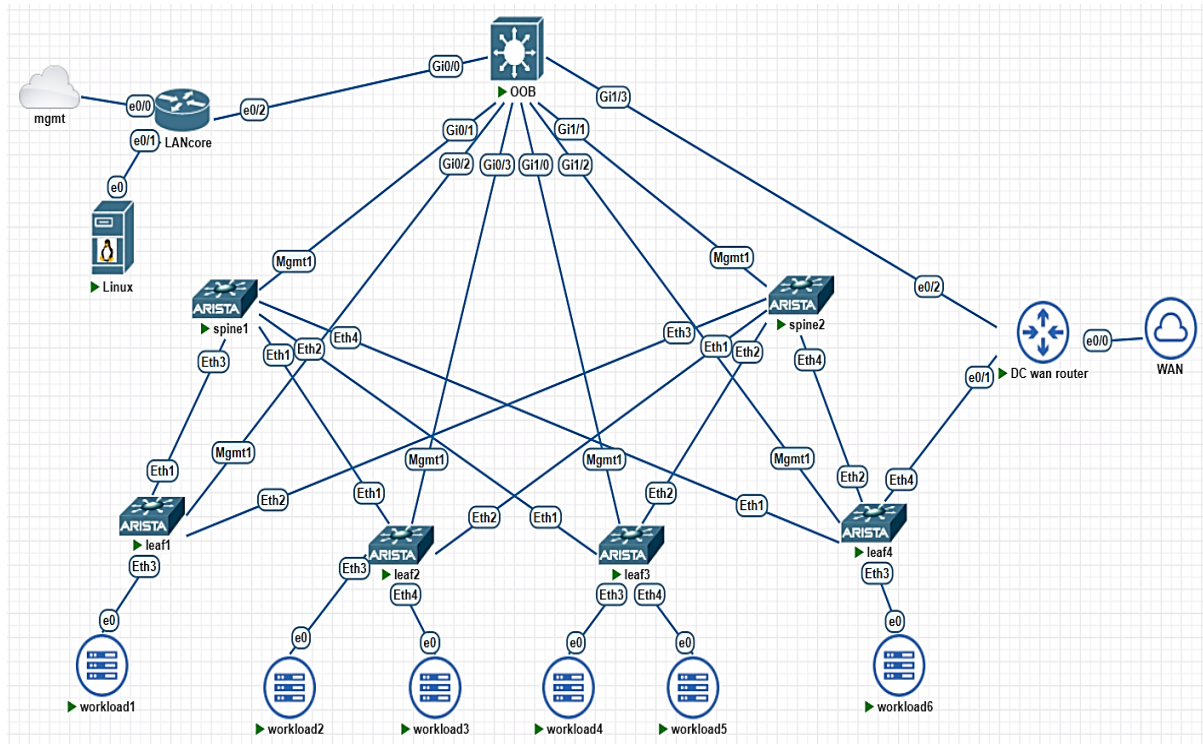


Figure 4: Scaled 4-Leaf Spine-and-Leaf Data Center Topology.

7.3 Automated Underlay Provisioning

Once the management network and physical topology were established, the underlay routing infrastructure was provisioned utilizing the Ansible automation engine. Ansible's agentless, push-based architecture enabled the rapid, uniform deployment of baseline configurations across all spine and leaf nodes simultaneously.

To comprehensively evaluate underlay performance, both OSPF and IS-IS routing protocols were sequentially deployed and tested.

To optimize link-state adjacency formation and eliminate the unnecessary election of a Designated Router (DR) within the OSPF environment, all physical inter-switch connections were strictly configured as point-to-point network types. The declarative YAML logic utilized within the Ansible playbooks to uniformly push these physical interface parameters is detailed in the excerpt below -name : optimize physical interfaces for OSPF

```
arista.eos.eos_config:
```

```
lines:
```

- no shutdown
- ip ospf area 0
- ip ospf network point-to-point
- ospf mtu-ignore

```
parents: "interface {{ item.name }}"
```

```
loop: "{{ p2p_interfaces }}"
```

By utilizing this Infrastructure as Code (IaC) methodology, the manual, device-by-device configuration of individual interfaces was entirely eliminated. This programmatic approach guaranteed strict configuration consistency and successfully mitigated the risk of human error across the scalable underlay fabric.

7.4 Automated Overlay Provisioning

To overcome the multi-tenant limitations of traditional VLANs, VXLAN data plane and a MP-BGP EVPN control plane were deployed.

While declarative playbooks efficiently managed the static underlay, the programmatic complexities of generating Symmetric Integrated Routing and Bridging (IRB) tables and Virtual Routing and Forwarding (VRF) instances required a more dynamic approach. Consequently, a Python automation framework utilizing the Netmiko library was engineered. By iterating through relational data dictionaries, the Python scripts dynamically mapped tenant VLANs to their corresponding Virtual Network Identifiers (VNIs) and generated the necessary Anycast Gateway configurations.

The following script excerpt demonstrates the iterative logic utilized to push the EVPN and VXLAN parameters across the leaf nodes:

```
commands.extend([
    f"vrf instance {vrf_name}",
    f"ip routing vrf {vrf_name}",
    f"ip virtual-router mac-address {anycast_mac}",
    f"vlan {transit_vlan}",
    f"interface Vlan{transit_vlan}",
    f"vrf {vrf_name}",
    "interface Vxlan1",
    "vxlan source-interface Loopback0",
    "vxlan udp-port 4789",
    f"vxlan vrf {vrf_name} vni {transit_vni}"])
net_connect.send_config_set(commands)
```

This programmatic methodology enabled the rapid and consistent instantiation of isolated tenant environments across the entire data center fabric without manual intervention.

7.5 External Gateway and WAN Integration

To facilitate North-South traffic and enable communication between the internal data center workloads and external networks, a dedicated Wide Area Network (WAN) router was integrated into topology. This external gateway was configured to bridge the isolated MP-BGP EVPN control plane with external routing domains.

Network Address Translation (NAT) was implemented on the WAN router to securely map internal tenant IP addresses to external, publicly routable IP addresses. This implementation step ensured that the automated fabric could seamlessly route traffic to and from external segments without compromising the internal multi-tenant isolation or security boundaries established by the VXLAN overlay.

7.6 Closed-Loop Telemetry and Dashboard Implementation

To guarantee continuous operational availability and enforce dynamic self-healing mechanisms, a closed-loop telemetry system was engineered. A dedicated Python auditing script (`network_health.py`) was developed to perform continuous network health and configuration state audits across the switching fabric.

This script was scheduled as a system background process utilizing the Linux Cron daemon, configured to execute precisely at one-minute intervals. During execution, the script securely polls the active operational state of every Arista switch and parses the running configuration against a predefined, verified "golden state" file (`golden_config.cfg`).

If an unauthorized configuration drift or a routing protocol failure is detected, an automated remediation function is instantly triggered. The script forces an automated configuration replacement, seamlessly restoring the verified network state within sixty seconds without requiring manual human intervention.

To provide real-time observability of this automated process, a dynamic HTML monitoring dashboard was deployed. This custom interface translates the raw Python telemetry data into a color-coded visual status of the entire data center fabric. Healthy nodes maintaining perfect state parity are visually highlighted, while nodes

actively undergoing automated remediation are dynamically flagged for operational awareness. real-time monitoring dashboard is depicted in Figure (5).

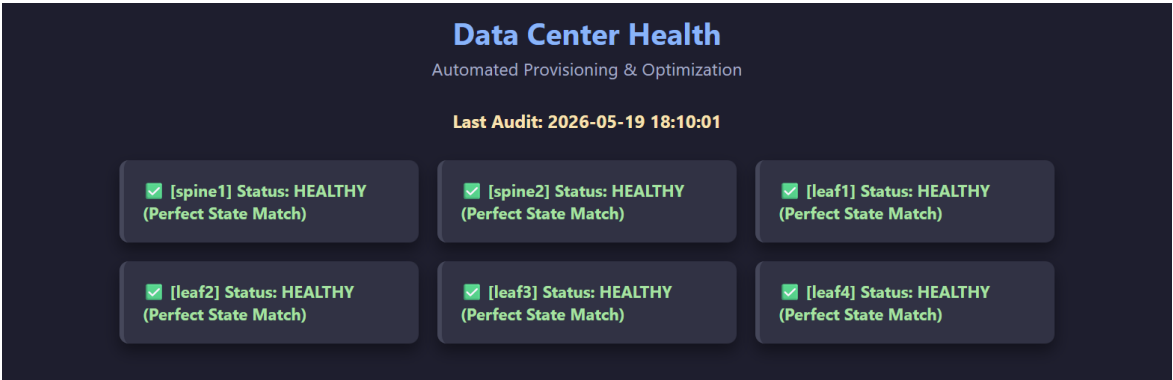


Figure 5: Real-Time Data Center Health and Telemetry Dashboard.

8. RESULTS AND DISCUSSION

Following the successful programmatic deployment of the Spine-and-Leaf fabric, rigorous empirical testing was conducted to validate the overall performance of the architecture and determine the optimal routing protocol for the data center underlay. Data plane performance was quantified utilizing standard network measurement utilities. Internet Control Message Protocol (ICMP) requests were utilized to measure baseline network latency, while the iperf3 utility was deployed to generate a sustained 1.00 Mbps UDP stream, allowing for the precise measurement of throughput capacity, jitter, and packet loss under an active network load. In the baseline two-leaf topology, the performance metrics indicated that IS-IS maintained a slight latency advantage over OSPF. However, when the network fabric was scaled to a four-leaf architecture, a critical divergence in reliability was recorded.

8.1 Baseline Data Plane Performance

Initial performance evaluations were conducted on the baseline two-leaf topology to establish a control benchmark. A comparative visual analysis of the data plane metrics specifically latency, throughput, and jitter between the OSPF and IS-IS underlay routing protocols is illustrated in the figures below.

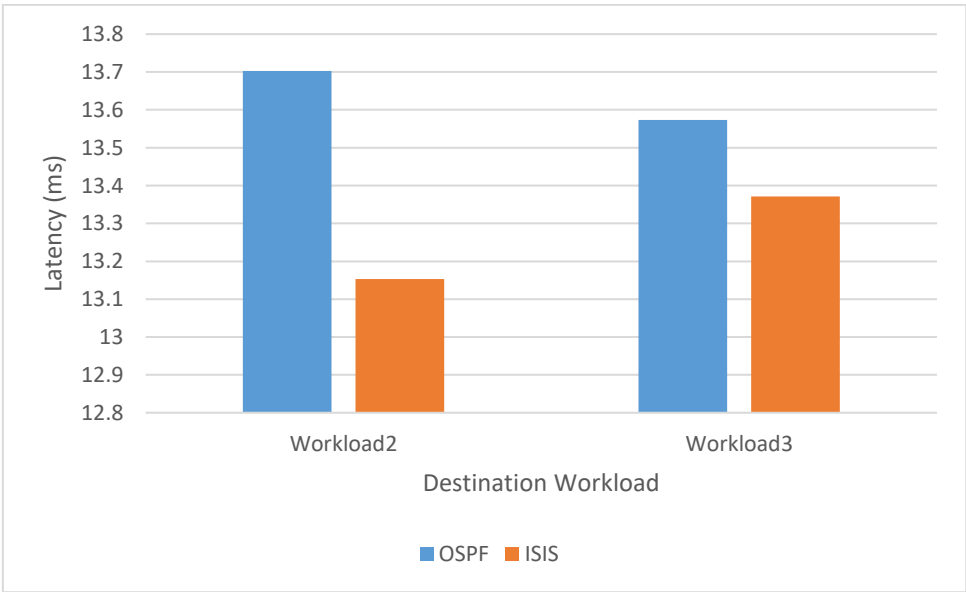


Figure 6: Latency Comparison in Baseline 2-Leaf Topology.

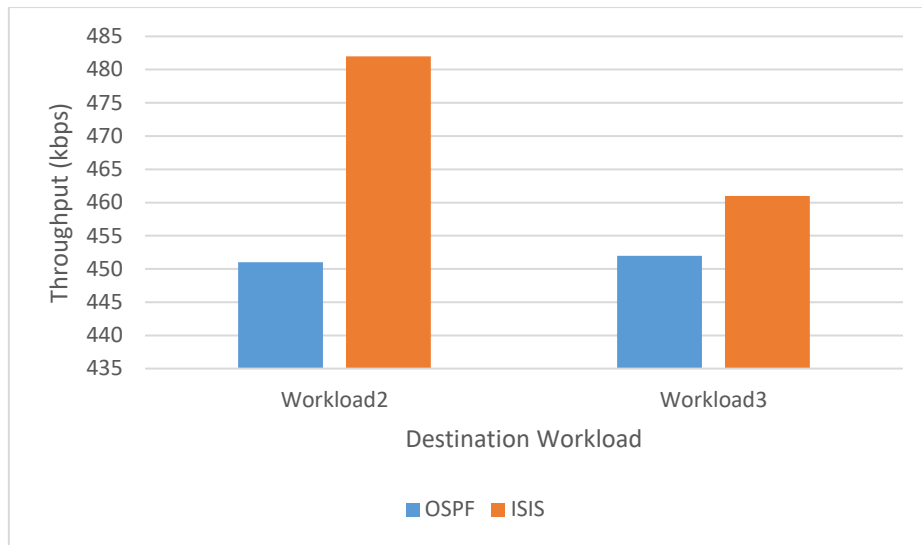


Figure 7: Throughput Comparison in Baseline 2-Leaf Topology.

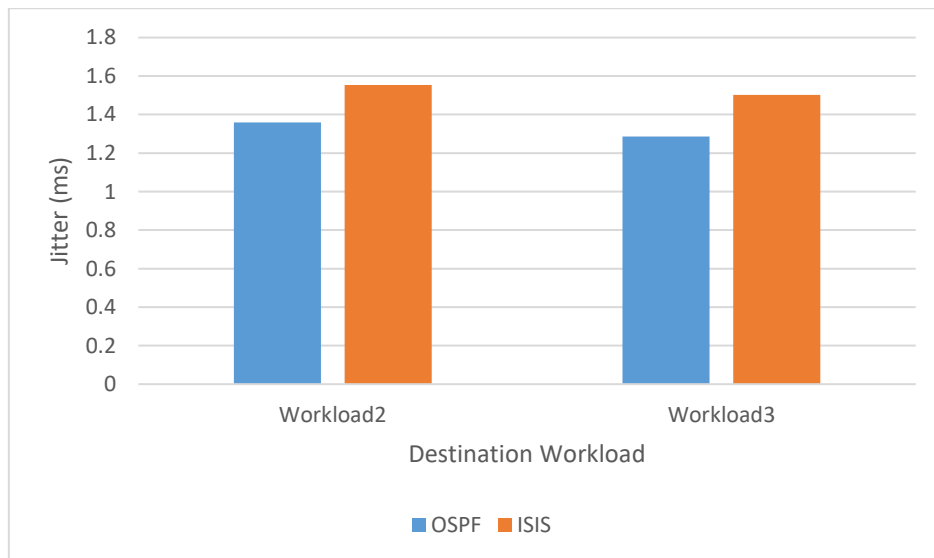


Figure 8: Jitter Comparison in Baseline 2-Leaf Topology.

As depicted in the charts above, within this restricted physical footprint, both protocols successfully maintained a zero percent packet loss rate under the sustained 1.00 Mbps UDP stream. The empirical data visually indicates that IS-IS maintained a negligible latency advantage and marginally higher throughput compared to OSPF. Conversely, OSPF demonstrated superior stability regarding transmission delay variation, recording consistently lower jitter across the evaluated workloads. Overall, both protocols demonstrated highly efficient and comparable performance within the unscaled, baseline environment.

8.2 Scaled Topology Performance

To evaluate the routing protocols under realistic expansion constraints, the network fabric was scaled to a four-leaf architecture. The performance metrics recorded under the sustained UDP control stream within this expanded environment are presented in the following figures.

When the fabric was expanded to the four-leaf architecture, a critical divergence in protocol reliability emerged. As illustrated in the scaled performance charts, the IS-IS underlay experienced noticeable performance degradation and intermittent packet loss. Conversely, the OSPF underlay remained entirely stable. OSPF successfully maintained a zero percent packet drop rate across all measured scaled nodes, demonstrating vastly superior resilience under expanded physical conditions.

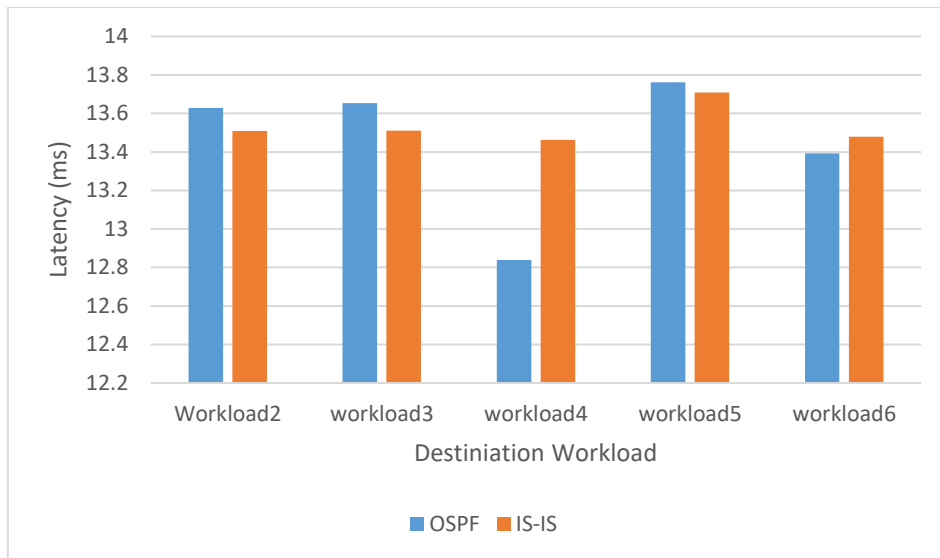


Figure 9: Latency Comparison in Scaled 4-Leaf Topology.

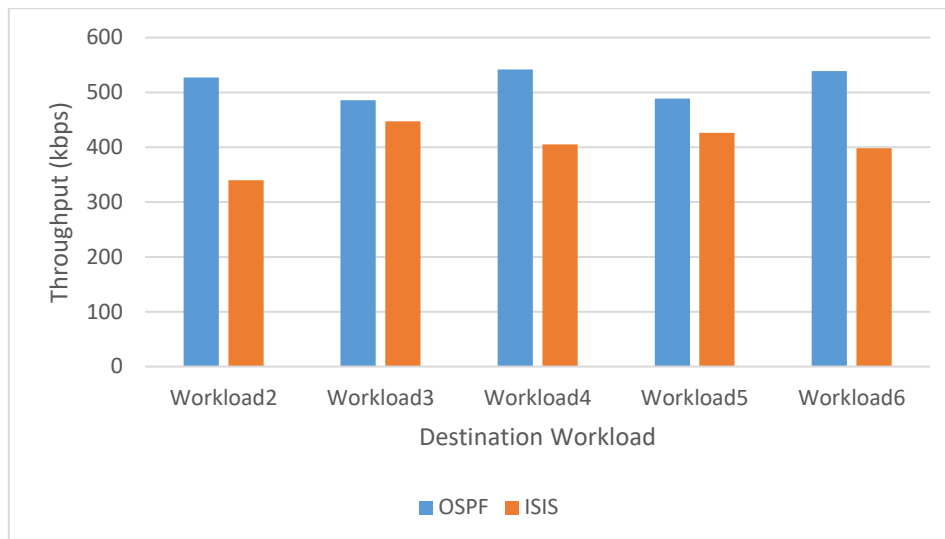


Figure 10: Throughput Comparison in Scaled 4-Leaf Topology.

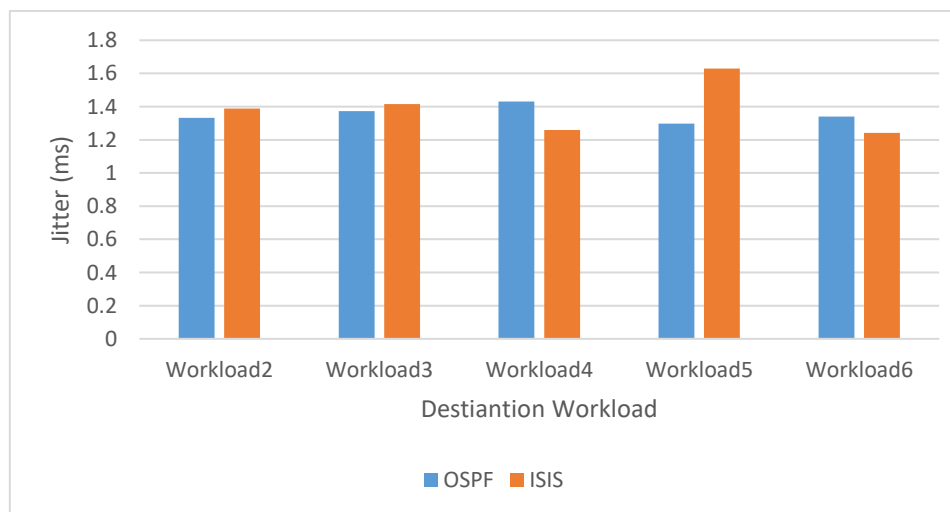


Figure 11: Jitter and Packet Loss in Scaled 4-Leaf Topology.

While the empirical data provides an accurate comparative baseline for control plane behavior, it is critical to note that the absolute data plane metrics specifically latency, throughput, and jitter are constrained by the software emulation environment. In a production data center deployment, packet forwarding, routing lookups, and VXLAN encapsulation are processed entirely in hardware by specialized Application-Specific Integrated Circuits (ASICs). These custom silicon chips enable non-blocking, line-rate throughput with microsecond or even nanosecond latency. Because this topology was deployed utilizing virtualized images relying on CPU-based software switching, the performance metrics do not reflect the hardware-accelerated capabilities of physical Arista switches. However, because the underlying virtual compute resources were distributed equally across all nodes, the relative performance differences observed between the protocols remain analytically valid.

8.3 Control Plane Convergence Analysis

Beyond raw data plane metrics, the efficiency of the underlay routing protocol is heavily dependent on its ability to rapidly recalculate routing tables during a link failure. During simulated link failure scenarios within the scaled fabric, OSPF demonstrated significantly superior control plane convergence. OSPF successfully recalculated the topology and restored full reachability in approximately 0.2 seconds. In contrast, IS-IS exhibited a noticeably slower convergence time, requiring approximately 1.0 second to restore traffic flow.

The performance degradation and packet loss observed with IS-IS in the expanded topology can be directly attributed to its handling of Maximum Transmission Unit (MTU) parameters. IS-IS routers actively pad their Hello Protocol Data Units (PDUs) to the maximum interface MTU to ensure two-way communication paths can support full-sized frames. Within a virtualized environment utilizing VXLAN encapsulation which inherently adds 50 bytes of overhead to the original Ethernet frame this strict MTU padding often leads to frame fragmentation or dropped Hello packets if the MTU is not perfectly synchronized across all virtual switches and hypervisor interfaces.

Because OSPF natively handles link-state advertisements without strict MTU padding constraints by default, it proved significantly more resilient to the encapsulation overhead introduced by the VXLAN overlay. Consequently, based on its 0.2-second convergence time, superior jitter stability, and 0% packet loss under scaled loads, OSPF was definitively selected as the optimal underlay routing protocol for this automated Spine-and-Leaf architecture.

9. CONCLUSION AND RECOMMENDATIONS

This research concludes that manual network configuration is inadequate for the rapid scalability demanded by modern data centers. By implementing Infrastructure as Code (IaC) principles utilizing Ansible and Python, the complex configurations for the physical underlay and logical overlay were successfully abstracted. This declarative methodology drastically reduced provisioning time, eliminated human error, and ensured strict configuration parity across the switching fabric.

The deployment of a deterministic, two-tier Spine-and-Leaf architecture optimized the fabric for heavy east-to-west traffic by guaranteeing a two-hop maximum pathway. Furthermore, integrating VXLAN with an MP-BGP EVPN control plane enabled massively scalable, secure multi-tenancy while completely eliminating inefficient data plane flooding. Through rigorous comparative analysis, OSPF was validated as the optimal underlay routing protocol, exhibiting sub-second convergence and zero packet loss under scaled workloads. Ultimately, the development of a closed-loop telemetry system featuring 60-second automated audits, configuration drift detection, and a real-time dashboard proved that programmatic automation is a fundamental necessity for engineering resilient, self-healing infrastructures.

Recommendations for Future Deployments:-

To further enhance the scalability and intelligence of this automated architecture in production environments, the following advancements are recommended:

- **Multi-Site Redundancy:** Expand the MP-BGP EVPN control plane across geographically dispersed sites utilizing Data Center Interconnect (DCI) technologies to achieve active-active disaster recovery.
- **AI-Driven Predictive Telemetry:** Transition to a proactive remediation model by feeding telemetry data into Machine Learning (ML) algorithms capable of detecting micro-anomalies and predicting hardware degradation.
- **NetDevOps CI/CD Integration:** Integrate automation scripts into version control systems (e.g., Git) and CI/CD pipelines to automatically validate proposed network changes in a digital twin before production deployment.

- **Cloud-Native Orchestration:** Interface the fabric directly with orchestration platforms (e.g., Kubernetes) via APIs to dynamically provision isolated network segments exactly when new containerized workloads are instantiated.

ACKNOWLEDGEMENTS

The authors acknowledge this research has never been published in any scientific journal and bears the literary and professional responsibility for its scientific content.

DECLARATION OF CONFLICTING INTERESTS

The authors declared no potential conflicts of consequence with respect to the research, authorship, and/or publication of this article.

FUNDING

This authors received no financial support for the research, authorship, and/or publication of this article.

Compliance with ethical standards

Disclosure of conflict of interest

The authors declare that they have no conflict of interest.

References

- [1] K. S. R. Kahagalage, K. C. Yakupitiya, H. P. N. Warnasooriya, and S. Joonos, "Automated Network Provisioning and Optimization in Data Center Environments Using Custom Open-Source Tools with Advanced Network Technologies - Proof of Concept," School of Computing, Kingston University and ESOF Uni, 2024. [Accessed: Mar. 28, 2026].
- [2] W. Rafique and L. Velasco, "Machine Learning for Network Automation: Overview, Architecture, and Applications," IEEE Communications Surveys & Tutorials, vol. 20, no. 4, pp. 3360-3403, 2018. [Accessed: Mar. 28, 2026].
- [3] Red Hat, "Ansible Documentation," 2021. [Online]. Available: <https://docs.ansible.com>. [Accessed: Mar. 28, 2026].
- [4] Cisco Systems, "Networking Technologies for Data Centers," 2021. [Online]. Available: <https://www.cisco.com>. [Accessed: Mar. 28, 2026].
- [5] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 8th ed. Pearson, 2020.
- [6] J. Zhang, M. Xia, et al., "A Survey on Data Center Network Architectures," IEEE Communications Surveys and Tutorials, vol. 18, no. 1, pp. 264-282, 2016.
- [7] W. Odom, CCNA 200-301 Official Cert Guide, Volume 1, Cisco Press, 2019.
- [8] J. Doyle and S. Carroll, Routing TCP/IP, Volume 1, 2nd ed. Cisco Press, 2005.
- [9] S. Halabi, Internet Routing Architectures, 2nd ed. Cisco Press, 2000.
- [10] J. Edelman, C. S. Lowe, and M. Oswalt, Network Programmability and Automation, 1st ed. O'Reilly Media, 2018.
- [11] M. Mahalingam et al., "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348, 2014. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7348>. [Accessed: 10-Mar-2026].
- [12] A. Sajassi et al., "BGP MPLS-Based Ethernet VPN," RFC 7432, 2015. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7432>. [Accessed: 10-Mar-2026].
- [13] O. A. M. Ali, "Network Automation with Ansible and Python for Large-Scale Data Centers," SSRN Electronic Journal, 2024. [Online]. Available: <https://ssrn.com/abstract=5189881>. [Accessed: 18-Mar-2026].
- [14] ITPrice, "Cisco and Arista Global Price List and Order Guide - Data Center Infrastructure," 2026. [Online]. Available: <https://www.itprice.com/cisco-price-list/7280R.html>. [Accessed: Mar. 28, 2026].
- [15] Router-Switch, "Enterprise Networking and Server Price Catalog - Dell PowerEdge and Arista Series," 2026. [Online]. Available: <https://www.router-switch.com/price-list.html>. [Accessed: Mar. 28, 2026].

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of **AJAPAS** and/or the editor(s). **AJAPAS** and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.