



PDF Download
3759023.3759117.pdf
06 January 2026
Total Citations: 0
Total Downloads: 211

Latest updates: <https://dl.acm.org/doi/10.1145/3759023.3759117>

RESEARCH-ARTICLE

Towards a Heterogeneous Distributed SDN Control Plane System: A comparative Review of Open Network Operating System and Open Daylight

NTSHUXEKO MAKONDO, The Council for Scientific and Industrial Research, Pretoria, Gauteng, South Africa

TOPSIDE E MATHONSI, Tshwane University of Technology, Pretoria, Gauteng, South Africa

WANDILE THABANI MNYANDU, The Council for Scientific and Industrial Research, Pretoria, Gauteng, South Africa

HLABISHI ISAAC KOBO, The Council for Scientific and Industrial Research, Pretoria, Gauteng, South Africa

DANIEL DU PLESSIS, University of Johannesburg, Johannesburg, Gauteng, South Africa

Open Access Support provided by:

Tshwane University of Technology

The Council for Scientific and Industrial Research

University of Johannesburg

Published: 26 November 2025

Citation in BibTeX format

icABCD 2025: 2025 International
Conference on Artificial Intelligence,
Big Data, Computing and Data
Communication Systems
November 26 - 27, 2025
Cape Town, South Africa

Towards a Heterogeneous Distributed SDN Control Plane System: A comparative Review of Open Network Operating System and Open Daylight

Ntshuxeko Makondo
NextGen Enterprises and Institutions
Council for Scientific and Industrial
Research (CSIR)
Pretoria, Gauteng, South Africa
NMakondo2@csir.co.za

Topside Mathonsi
Department of Information
Technology
Tshwane University of Technology
Pretoria, South Africa
MathonsiTE@tut.ac.za

Wandile Mnyandu
NextGen Enterprises and Institutions
Council for Scientific and Industrial
Research (CSIR)
Pretoria, South Africa
WMnyandu@csir.co.za

Hlabishi Kobo
NextGen Enterprises and Institutions
Council for Scientific and Industrial
Research (CSIR)
Pretoria, South Africa
HKobo@csir.co.za

Daniel Du Plessis
Academy of Computer Science and
Software Engineering
University of Johannesburg
Johannesburg, South Africa
Danieldp@uj.ac.za

Abstract

The evolution of software-defined networking (SDN) has transitioned from centralized control plane architectures to distributed control plane models, enhancing scalability, reliability, and fault tolerance. While central controllers often face challenges in managing extensive networks, distributed control architectures provide superior performance and flexibility. Initially, homogeneous distributed architectures were prevalent; however, the demand for heterogeneous distributed control planes where controllers interact with diverse systems and components has increased, necessitating advanced coordination and interoperability. This paper compares two widely adopted open-source distributed controllers, the Open Network Operating System (ONOS) and OpenDaylight (ODL), focusing on their methodologies for distributed clustering. It further explores the roles of Atomix and Akka (now Apache Pekko) distributed frameworks in ONOS and ODL respectively, discussing their impact on performance and scalability. Furthermore, the paper examines the data consistency models used by ONOS and ODL to manage network state and synchronization. Finally, it offers recommendation for enhancing the interoperability and performance of heterogeneous distributed control planes in large-scale and diverse network deployments.

CCS Concepts

• **Computer systems organization** → **Distributed architectures**; *Dependable and fault-tolerant systems and networks*; • **Networks** → *Network architectures*.

Keywords

ONOS, ODL, Distributed frameworks, Clustering, Consistency, Heterogeneous distributed Control Plane.

ACM Reference Format:

Ntshuxeko Makondo, Topside Mathonsi, Wandile Mnyandu, Hlabishi Kobo, and Daniel Du Plessis. 2025. Towards a Heterogeneous Distributed SDN Control Plane System: A comparative Review of Open Network Operating System and Open Daylight. In *2025 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD 2025)*, November 26–27, 2025, Cape Town, South Africa. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3759023.3759117>

1 INTRODUCTION

Software-defined networking (SDN) has revolutionized network management by decoupling the control plane from the data plane, allowing for centralized decision-making, dynamic network configuration, and enhanced programmability [30]. In traditional SDN architecture, a single centralized controller manages network operations [64]. While suitable for small-scale networks, this approach becomes inefficient as networks grow in size and complexity [7]. This architecture often leads to performance bottlenecks, scalability challenges, and reliability issues due to the risk of controller failure [7]. In the event of an SDN controller failure, SDN switches such as the Open vSwitches (OVS) would be isolated, as depicted in Figure 1. The recent works in [31, 32] serve as examples of centralized SDN architectures, where a single controller manages the entire network [7]. To overcome these limitations, researchers and industry practitioners have turned to distributed SDN controller architectures, where control plane functions are distributed across multiple interconnected controllers, forming a highly available and resilient cluster [7].

SDN controller clustering enables seamless inter-controller synchronization, reduces latency, provides controller redundancy, and ensures consistent network state management, making it suitable



This work is licensed under a Creative Commons Attribution 4.0 International License.
icABCD 2025, Cape Town, South Africa
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1427-6/25/11
<https://doi.org/10.1145/3759023.3759117>

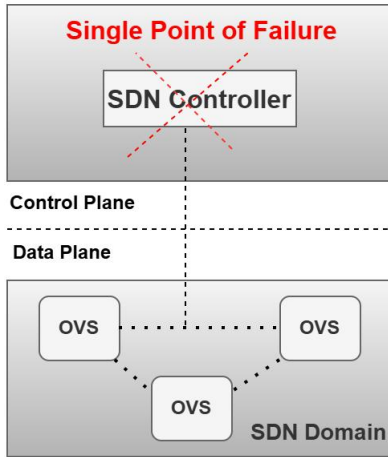


Figure 1: Centralised SDN Controller

for large-scale and mission-critical SDN deployments [37]. A common distributed SDN deployment is that of a homogeneous distributed control plane system, where controllers from a single vendor form a uniform cluster, as it has reduced complexity [64]. In this setup, controllers use East-to-West Interface (EWi) Application Programming Interfaces (APIs) to synchronise and coordinate their operations for global network management as depicted in Figure 2 [64]. EWi API facilitates communication between SDN controllers in a distributed control plane system [29]. This setup takes advantage of consensus algorithms to ensure consistent network view, state synchronization, and dynamic load balancing, enhanced scalability, fault tolerance, and overall network efficiency [65]. Naturally, the homogeneous architecture suffers from vendor lock-in, making it hard to explore new technologies and innovations from other vendors due to incompatibility issues [64]. In contrast to the homogeneous architectures are the heterogeneous counterparts. In this network architecture, SDN controllers may be from different vendors, opening opportunities for business innovation while potentially imposing interoperability issues [61]. Authors in [5, 33, 34] emphasized that Open Daylight (ODL) [41] and Open Network Operating System (ONOS) [39], are the most widely adopted open-source distributed controllers in industry and academia. Júnior [21] claimed that ONOS is well-suited for multi-domain Wide Area Networks (WANs) and service provider networks. While Rahim et al. [46] highlighted that ODL is more oriented toward data centre environments, focusing on the integration of legacy infrastructures with SDN. Although they specialise in different deployment cases, both projects are committed to achieving scalability, high performance, and high availability in their control planes [48] making them worthy candidates for investigating heterogeneous SDN control plane architectures.

However, maintaining consistency, scalability, reliability and coordination among heterogeneous controllers remains a significant challenge, particularly in large-scale deployments such as Software-Defined Wide Area Networks (SD-WAN) [3, 60]. This complexity arises from the fact that controllers may operate using different

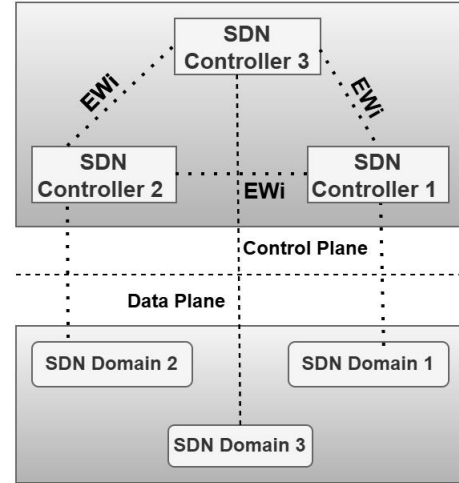


Figure 2: Distributed SDN Control Plane

distributed frameworks, consistency models, and data models, leading to potential state synchronization issues, latency overhead, and interoperability constraints [64]. Fortunately, ONOS and ODL both use Yet Another Next Generation (YANG) as a data model which allows the controllers to define a common understanding of the underlying network elements, configurations, and their states [57]. This ensures consistency in how network resources are represented and managed across the control plane since there is no need for complex translations or mappings between different data models. Nonetheless, ONOS and ODL use different communication mechanisms for their EWi APIs, with ONOS using the Atomix framework and ODL using the Akka framework [27]. The lack of a standard unified EWi API results in inconsistencies in policy enforcement, flow rule propagation, and network-wide decision-making, necessitating a standardized cross-platform EWi API to ensure seamless operation [64].

Therefore, this work aims to review and compare ODL and ONOS distributed frameworks to assist in deriving principles for a controller-independent EWi API design. Furthermore, this paper investigates the underlying consistency models, analysing their approaches to network state synchronization, fault tolerance, and scalability in large-scale SDN deployments. This is followed by a critical evaluation of the strengths and limitations of these models within complex network infrastructures, identifying key challenges in maintaining consistency and interoperability. Finally, we propose a strategic framework for implementing a heterogeneous multi-SDN control architecture that integrates both ODL and ONOS, fostering seamless coordination, cross-platform interoperability, and efficient network management in dynamic and large-scale SDN environments. The contribution of this paper can be summarized as follows.

- This paper presents an overview of the widely adopted distributed SDN controllers.
- This paper further provides insights into the distributed frameworks for these controllers.

- The consistency of these controllers is detailed, focusing on the Raft consensus algorithm.
- Lastly, this paper delves into the interoperability of the heterogeneous control plane comprising the ONOS and ODL controllers.

The remainder of this paper is organized as follows. Section 2 provides an overview of ONOS and ODL. Section 3 describes the ODL and ONOS distributed frameworks. Section 4 discusses consistency in SDN distributed controllers. Section 5 explores interoperability between ONOS and ODL. Section 6 presents recommendations, including comparative scenarios and key findings. Finally, Section 7 concludes the paper and outlines directions for future work.

2 Overview of ONOS and ODL

ONOS is a distributed control system built on the Floodlight platform [36] and is designed to achieve high scalability, optimal performance, and robust availability. This controller is developed in Java and licensed under the Apache 2.0 License [66]. In addition, ONOS offers multiple interfaces for interaction, including REST APIs, a Command-Line Interface (CLI), and a web-Based Graphical User Interface (GUI). The architecture of ONOS follows a tiered model as seen in Figure 3, where the distributed core is responsible for separating control and data plane functions while maintaining a logically centralized perspective of the network [12, 29]. The core interacts with network elements through a Southbound API, which supports protocols such as OpenFlow and NETCONF, while the Northbound API enables integration with higher-level applications and services [55]. In addition, ONOS utilizes a modular framework using OSGi-based dynamic applications to handle device-specific operations efficiently [55].

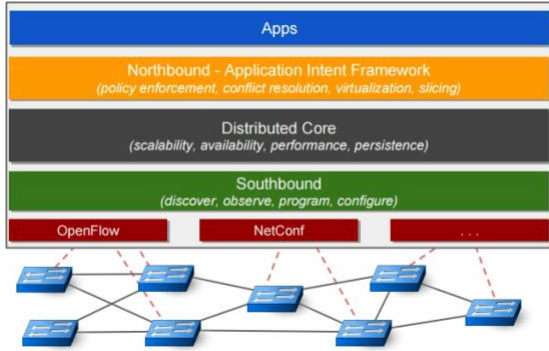


Figure 3: ONOS architecture [12]

While ONOS is a leading SDN controller [34], ODL remains its primary competitor. ODL, designed as a distributed controller and implemented in Java, adopts a model-driven SDN architecture and leverages YANG [9] as its primary modelling language to define system and application components [53]. ODL controller integrates key technologies, such as OSGi for dynamic module management and Karaf as an application container for deployment [53]. Initially, ODL implemented the Service Abstraction Layer (SAL) using an API-Driven architecture (AD-SAL), which required coding SAL

APIs and adaptation functionality for routing service requests between consumers and providers [16]. Although AD-SAL simplified element-level complexity, it faced scalability challenges due to the need for repetitive coding when adding new plugins.

In response to these challenges, ODL transitioned to a more scalable and flexible architecture, the Model-Driven Service Abstraction Layer (MD-SAL) [35]. MD-SAL uses model objects to represent service and network states, which are decoupled from specific protocol implementations, enhancing scalability, and flexibility [16]. A key feature of MD-SAL is its extensibility as middleware, inspired by message-bus architectures [11].

In addition, MD-SAL provides messaging and data storage functionality based on user-defined data and interface models, enabling inter-application communication [47]. This architecture supports various transport and payload formats such as binary, XML, and JSON, and leverages YANG for both interface and data definitions, establishing a messaging and data-centric runtime environment tailored to YANG-modelled services [10].

Furthermore, MD-SAL offers two types of APIs: MD-SAL Binding, which generates APIs and classes from YANG models, ensuring compile-time safety, and MD-SAL DOM (Document Object Model), which represents data in a flexible DOM-like structure [47]. The DOM APIs provide greater flexibility but at the expense of compile-time safety. Additionally, ODL facilitates external interactions through model-driven protocols like NETCONF, supporting XML-based remote procedure calls and data operations, and RESTCONF, which provides a REST-like API for managing YANG-modelled data in XML or JSON formats [47]. These features collectively enhance ODL’s scalability, flexibility, and interoperability within SDN architectures.

Even though both ONOS and ODL are designed for SDN environments, ONOS is optimized for large-scale, distributed deployments due to its scalability and high availability [48]. On the other hand, ODL prioritizes flexibility and extensive protocol support, making it well-suited for diverse networking applications and multi-protocol interoperability [48]. Table 1 presents a comparative analysis of the key features of ONOS, and ODL, as outlined in the study by Mamushiane et al. [33].

Table 1: Feature-based Comparison between ONOS and ODL

Feature	ONOS	ODL
Southbound, Interfaces	OpenFlow, NETCONF	OpenFlow, NETCONF, YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP, OFCONFIG
REST API	Yes	Yes
OS Support	Linux, Windows, MAC	Linux, Windows, MAC
Virtualization	Mininet and OVS	Mininet and OVS
Distributed	Yes	Yes
Programming, Language	Java	Java

3 Overview of SDN Distributed Control Plane

Recent studies have highlighted the growing interest in physically distributed control plane architectures as a promising solution to the limitations inherent in centralized SDN architectures [7]. In distributed SDN control plane systems, coordination among controllers is crucial to avoid inconsistencies [38]. Two main strategies for achieving consensus are the leaderless approach and the leader-based approach. In the leaderless approach, all controllers are equal and communicate directly with each other [38]. Hyper Flow [56] is an example where controllers serialize and deserialize messages without a designated leader.

On the other hand, in the leader-based approach, one controller is designated as the leader to ensure that decisions, such as adding or deleting forwarding paths, are agreed upon by multiple controllers before being executed [38]. This model can be hierarchical, where the root controller acts as the leader, or flat, where a leader is elected through a voting process. In leader-based models, controllers communicate with the leader for decision-making, ensuring consistent and conflict-free network management [38]. The distributed SDN control plane approach is further divided into three models: flat, hierarchical, and hybrid topologies [2]. Among these, the flat and hierarchical architectures are the most prevalent due to their scalability and efficient resource management.

3.1 Flat Control Plane Architecture

The flat network architecture is the predominant approach in distributed SDN control plane designs due to its scalability and efficiency in managing large-scale networks [2].

In this model, multiple controllers are responsible for distinct network segments while maintaining inter-controller communication through dedicated EWi for domain-specific information exchange as illustrated in Figure 4. The network view can be either global, where controllers have full visibility of the entire network, or local, where each controller manages only its designated segment, abstracting neighbouring networks as logical nodes [23].

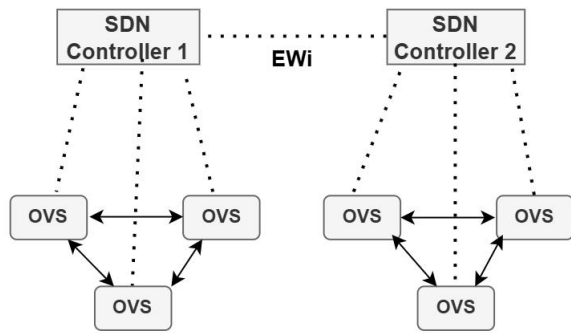


Figure 4: Flat Control Plane Architecture

3.2 The Hierarchical Control Plane

In this architecture, controllers are structured in multiple tiers, with local controllers managing region-specific tasks that do not require a global network view, while a root controller oversees applications that need comprehensive network visibility [2], as depicted

in Figure 5. This model differs from the flat architecture, where all controllers share a complete network view. In contrast, lower-tier controllers in a hierarchical setup operate with localized visibility, delegating broader network decisions to the higher-tier controller. This structure enhances efficiency by distributing frequent, localized operations to lower-tier controllers while assigning complex, network-wide tasks to the root controller [23].

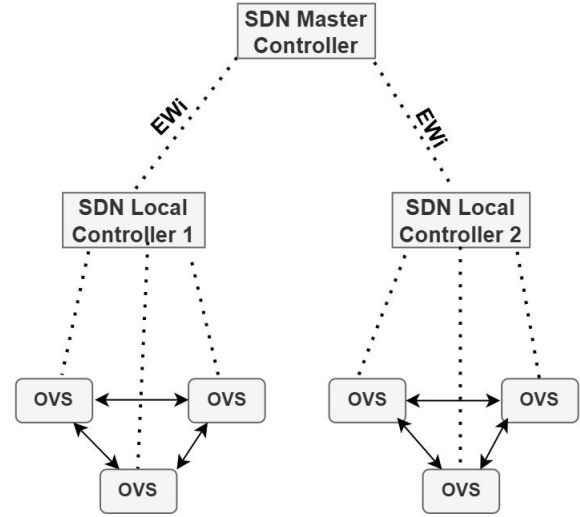


Figure 5: The Hierarchical Control Plane

3.3 Distributed Frameworks

This section provides the distributed frameworks for both ONOS and ODL controllers. A distributed framework is a mechanism used to coordinate and synchronise communications and operations between multiple peers.

3.3.1 Atomix framework for ONOS. In ONOS, clustering enables multiple controller instances to function as a single unified distributed system. Initially, Hazelcast was leveraged as a distributed data platform, facilitating state replication and distribution across nodes in the network [27]. Hazelcast supports both eventual and strong consistency using Best Effort and Anti-entropy as consensus algorithms [26]. However, its distribution mechanism posed challenges during network partitions, resulting in the "splitting brain" problem, as described by Kobo et al. [27]. This problem can lead to several parts of the system working in an inconsistent state, jeopardizing the reliability of the network.

To address this, ONOS adopted the Atomix framework from version 1.4 [6]. Atomix utilizes the Raft consensus algorithm [40] for distributed coordination and consensus across cluster nodes. Raft is a leader-based distributed consensus algorithm that ensures secure, sequential updates across systems with multiple active replicas [50]. This shift to Atomix enhanced coordination and fault tolerance, decoupling cluster management, service discovery, and persistent data storage from ONOS instances, improving scalability and reliability. In ONOS 1.14 (codenamed Owl), Atomix's functionality

was moved to a separate cluster, allowing for greater flexibility and independent management of these critical functions.

Atomix requires an odd number of control plane nodes (e.g., 3, 5, or 7) for the Raft algorithm to function effectively, as depicted in Figure 6. The framework guarantees strong consistency, ensuring that all controllers share the same state, even during network partitions.

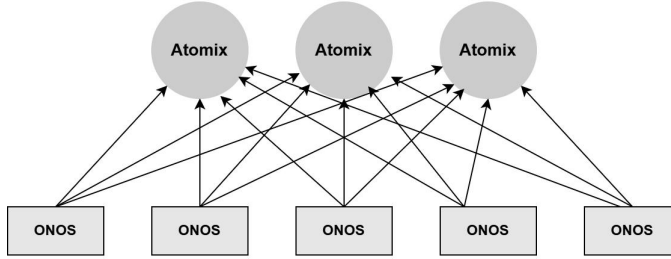


Figure 6: ONOS/Atomix Clustering

To establish an ONOS cluster, the platform relies on Atomix’s distributed primitives, which are essential for coordinating and managing data across a distributed system [58]. These primitives are fundamental data structures and synchronization mechanisms that enable multiple nodes to collaborate efficiently in tasks such as data storage, consensus, synchronization, and messaging, ensuring scalability, high availability, and fault tolerance [22]. Atomix provides various distributed primitives, including the EventuallyConsistentMap for eventual consistency, the ConsistentMap for strong consistency through Raft-based consensus, and additional primitives like DistributedSet, DistributedQueue, and AtomicCounter [58]. These primitives are replicated across the system and managed by Raft-based partition groups, each of which elects a leader to coordinate operations within the partition.

The management of these distributed primitives is facilitated through partition groups, introduced in Atomix 3.0 [22]. A partition group consists of multiple shards, each responsible for handling a subset of distributed primitives. Each shard implements a specific distributed systems protocol, such as Raft or primary backup, to ensure consistency and fault tolerance within its data subset [59]. For example, a distributed key-value store can be partitioned into multiple shards, with each shard managing a portion of the keys and values. These shards work together to ensure the smooth operation of the distributed system by adhering to the required protocols.

Distributed primitives are stored within partition groups, and they can operate across any partition group, irrespective of the protocol used [59]. In ONOS, the system is typically configured with a Raft partition group, enabling it to efficiently manage distributed primitives while ensuring consistency and high availability across the cluster. Through this mechanism, ONOS can effectively manage state, facilitate communication, and maintain the reliability of its distributed architecture [59].

3.3.2 Akka framework for ODL. ODL’s MD-SAL leverages Akka clustering for managing the distributed state of the cluster [8]. In contrast to ONOS, which migrated from Hazelcast to Atomix with a Raft-based consensus approach, ODL has consistently utilized

Akka’s actor-based model for coordination and communication [18]. At its core, ODL utilizes a distributed data store that is segmented into three primary shards: inventory, topology, and toaster [24]. The inventory shard stores the flow rules for network switches, ensuring consistent traffic management across the network, while the topology shard maintains an up-to-date view of the network’s structure [24]. The toaster shard is responsible for handling Remote Procedure Calls (RPCs) that facilitate automation and service orchestration within the cluster [24]. Initially, each shard is assigned to a specific controller node and is subsequently replicated across multiple nodes to prevent data loss in the event of a failure. Similarly to ONOS, ODL employs the Raft consensus algorithm, which coordinates the synchronization of state updates throughout the cluster to maintain strong consistency across the replicated shards [8]. Furthermore, the Akka facilitates peer-to-peer communication through Akka remoting and ensures fault tolerance via clustering, where nodes form a distributed system with an elected leader [8]. Lastly, the Akka persistence maintains state consistency across failures without altering internal states, enhancing redundancy as depicted in Figure 7.

When a node intends to join an Akka cluster, it initiates a structured four-way handshake defined by the gossip protocol. The node sends an InitJoin message to the seed nodes, which serve as the preconfigured contact points for new nodes. A seed node confirms the request by replying with an InitJoinAck message. In addition, the node sends a Join message and successfully joins the cluster when it receives a Welcome message from the seed node. The gossip protocol replicates the current state of the cluster across all nodes and maintains the cluster’s persistence. To achieve robust fault tolerance in ODL clustering, the system requires at least three controllers [20]. Additionally, nodes exchange heartbeat messages periodically and expect a HeartbeatRsp within a specified timeout to verify that each node remains active.

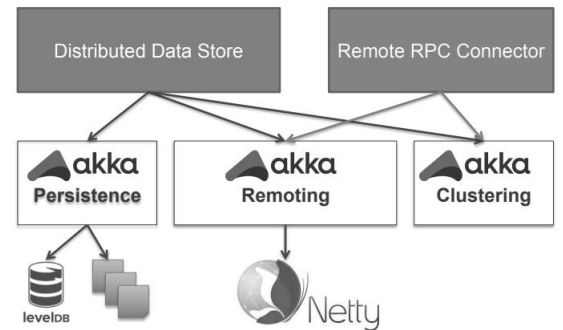


Figure 7: ODL Distributed Architecture [17]

Starting with ODL 2025.03 Titanium, ODL replaced Akka with Apache Pekko due to licensing restrictions imposed by Akka’s transition to a Business Source License (BSL) [42]. Apache Pekko, an open-source fork under the Apache 2.0 license, supports long-term sustainability and community-driven development, aligning with industry efforts to uphold open and scalable distributed architectures [1].

4 Consistency in ONOS AND ODL Controllers

Unlike physically centralized SDN architectures, distributed SDN controller platforms encounter significant challenges related to maintaining consistency [7]. In the SDN distributed control plane, consistency refers to maintaining a stable, up-to-date, and unified network-wide view, ensuring uniform policy updates across the network, and providing coherent fault tolerance [33]. Data consistency models are fundamental to distributed systems as they dictate how data converges across various nodes. Achieving an optimal balance between consistency and performance in the SDN distributed control plane, particularly when dealing with network partitions, is a complex task [7].

According to the CAP theorem applied to networks, it is generally not possible for SDN control plane to simultaneously achieve Consistency (C), high Availability (A), and Partition Tolerance (P) [43]. In this paper, the CAP theorem can be interpreted as follows: Consistency refers to the condition where all controllers, always maintain an identical network state, ensuring that the network remains symmetric. Availability means that all nodes in the network remain accessible and operational, preventing any downtime. Finally, Partition Tolerance ensures that the network continues to operate effectively, even in the event of a node failure or network partition, demonstrating the system's resilience during disruptions [26].

In the presence of network partitions, opting for a weaker consistency level in exchange for higher Availability (AP) results in stale state, which can cause incorrect application behavior. Conversely, maintaining a strong consistency level to ensure the correct enforcement of Network Policies (CP) comes at the cost of network availability. There are two primary consistency models: eventual consistency and strong consistency [27]. The former relies on gossip protocols like anti-entropy and rumormongering, while the latter uses consensus algorithms such as Raft and Paxos. The selection of a consistency model depends on the application's requirements and the nature of the data [27]. While achieving consistency is relatively simple in a centralized control architecture, where a single controller oversees the entire network, it becomes more complex in a distributed control setup where multiple controllers must coordinate and share the same network state.

In both ONOS/Atomix and ODL/Akka clustering platforms, strong consistency is preferred as it ensures that all controllers maintain an identical network state, which is crucial for managing network policies and fault tolerance. The strong consistency is a data consistency model in distributed systems that ensures all replicas always reflect the most recent update [25]. Strong consistency is implemented using leader-based consensus protocols like Raft [40] and Paxos [44], with Raft [40] being the common choice for both controllers. In this model, each replica in a cluster assumes one of three roles: leader, follower, or candidate as illustrated in Figure 8 [17]. When a client initiates a state update, the receiving replica forwards the request to the current cluster leader, which orders and serializes the updates, ensuring that the data store maintains a consistent state across the replicas [17]. The updates are then propagated to the followers and committed only after a majority (quorum) of replicas agree on the change [51]. Upon initialization, a node defaults to the Follower state, where it passively operates

by responding to requests from Leaders and Candidates without initiating any actions [17]. The Leader assumes primary responsibility for processing client requests and ensuring state replication across the cluster. If a client interacts with a Follower, the request is redirected to the Leader for handling. The Candidate state emerges during the leader election process, where nodes actively seek votes to establish a new Leader in the event of a leadership failure or transition [17].

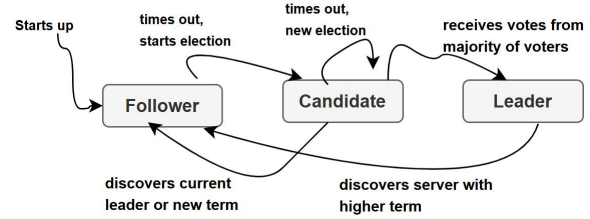


Figure 8: Raft Consensus Algorithm.

This consensus mechanism ensures consistency by preventing divergent views of the network state but it introduces overhead [51]. The round-trip communication to the leader for state confirmation and the requirement for most followers to agree on the update lead to added latency and lower responsiveness. The quorum-based consensus has a failure tolerance of $F = [C/2 - 1]$ in a cluster of C controllers, meaning that if too many controllers fail, consistency cannot be guaranteed [14, 43]. While Raft ensures correctness by serializing state updates, it leads to slower response times and limited availability, as the system must wait for consensus to be reached. In the best case, the cluster operates at the speed of the leader, but if the leader is slow or unavailable, the system's performance can be significantly impacted by the lag of the slowest follower [51]. Raft also ensures consensus among replicated data, even if a minority of instances fail [15]. It relies on heartbeat intervals and election timeouts, where the minimum timeout should be significantly higher than the network's broadcast time to avoid unnecessary leader elections [15]. Proper timeout configuration is crucial for maintaining system availability and preventing disruptions.

In contrast, the eventual consistency data model allows replicas to diverge temporarily but guarantees eventual convergence to the same state once all updates have been propagated and synchronized [49]. In the eventual consistency model, state updates can be handled locally at each replica without waiting for confirmation from a leader or quorum [51]. This means that applications can proceed with operations without blocking, leading to faster performance and higher availability. In ONOS, eventual consistency is implemented through an update-push distribution process and anti-entropy, where replicas regularly compare their local states and exchange updates to synchronize with each other over time [51].

These synchronization models introduce delays such as blocking time (the time taken for data to propagate) and read time (the time it takes to retrieve data). While strong consistency results in longer reaction times due to the need for full synchronization, eventual consistency provides quicker reactions, albeit with the

risk of temporary inconsistencies. The choice between these models plays a crucial role in the overall performance, reliability, and responsiveness of the SDN controller system [54].

Table 2: Comparison between Consistency Models

Communication Scenario	Strong Consistency	Eventual Consistency
Data Structure	ConsistentMap	EventuallyConsistentMap
Algorithm	Raft, Paxos	Anti-entropy, rumormongering
Update Period	Instantly	Periodically (every 5ms)

5 Plane Interoperability

A centralized SDN control plane faces significant scalability and reliability challenges, especially in large or geographically distributed networks [23]. As the network grows, a single controller struggles to manage increasing traffic and flow requests, leading to higher latency and reduced performance [19]. Additionally, a single controller creates a single point of failure, meaning that if it fails, all connected switches lose management capabilities, which affects service availability and exposes the network to potential security threats such as denial-of-service attacks [7]. To address these limitations, a multi-controller architecture distributes control across several controllers, reducing latency by positioning controllers closer to their respective switches while improving fault tolerance through redundancy [19].

A homogeneous SDN control plane is a form of multi-controller architecture consists of multiple controllers of the same type, often from a single vendor, ensuring seamless integration and consistent data models [64]. This homogeneity simplifies network management, facilitates efficient controller synchronization, and reduces the complexity of inter-controller communication. Homogeneous deployments also benefit from easier policy enforcement, as all controllers adhere to the same operational framework [64]. However, this approach introduces significant drawbacks, particularly in scalability, security, and vendor lock-in [64]. In addition, a vulnerability in one controller affects all others, leading to homogeneous controller common-mode failures where attackers can exploit a single flaw to compromise the entire control plane. Moreover, relying on a single vendor restricts flexibility, limiting the adoption of diverse data models and network innovations [64]. To overcome these challenges, the heterogeneous SDN control plane emerged, integrating multiple controllers from different vendors and architectures as seen in Figure 9, each optimized for specific network functions [7]. This approach enhances scalability and fault tolerance by eliminating single points of failure while improving security by ensuring that vulnerabilities in one controller do not compromise the entire network [7]. Additionally, this heterogeneity mitigates vendor lock-in, allowing network operators to adopt open standards and interoperable solutions [64]. However, this architecture brings inter-operability challenges in SDN distributed control plane systems [64]. This issue is further complicated by the lack of standardized inter-controller communication, especially for EWi [64].

Despite ongoing standardization efforts, the rapid and effective development of SDN EWi faces multiple obstacles, one of the most critical being the heterogeneity of data models used by different SDN controllers. Each vendor often adopts proprietary data models, making it difficult to establish a uniform representation of state and configuration data across controllers. Even though YANG has emerged as a promising standard, integrating it into diverse SDN architectures remains a challenge due to variations in internal structures and legacy systems. This heterogeneity complicates data exchange, increases development overhead, and hampers efforts to achieve seamless interoperability in multi-vendor, multi-domain environments. One of the earliest approaches to enabling cross-domain SDN interoperability was proposed by Lin et al. [28] with the east-west bridge (EWBridge). This high-performance mechanism facilitated controller communication across different platforms using JSON while ensuring network privacy by abstracting physical infrastructure behind a virtualized network. EWBridge has been deployed in large-scale networks such as CERNET, Internet2, and CSTN (CSTNET). Following this, Dixit et al. [13] introduced FlowBricks, a framework for integrating heterogeneous controllers within a unified control plane. By allowing diverse SDN controllers to contribute specialized services such as Virtual Private Network (VPN) and firewalls, FlowBricks sought to enhance network flexibility.

Implemented as a Floodlight module, it aimed to optimize service composition. Similarly, Yu et al. [62] proposed the Zebra architecture, which improved scalability and network efficiency through its dual-module structure: the Heterogeneous Controller Management (HCM) module for controller coordination and the Domain Relationships Management (DRM) module for inter-domain communication. Qi et al. [45] further addressed the challenge of heterogeneous controller integration by introducing a controller management system that creates a unified network view. Their approach, tested on a fat-tree-based data centre network with Ryu, Floodlight, and Pox controllers, introduced a standardized API framework to mitigate inconsistencies across different platforms.

More recent solutions have continued to refine EWIs for distributed SDN control. Yu et al. [63] introduced WECAN, which optimizes controller selection and cross-domain communication using a three-module structure for state aggregation, coordination, and efficient flow management, leading to improved network performance. Almadani et al. [4] proposed Distributed SDN control plane Framework (DSF), an adaptive EWi framework that synchronizes topologies across heterogeneous SDN controllers using a standardized communication protocol. However, DSF currently supports only Java-based control platforms. Hoang et al. [18] introduced SDN Inter-Domains Network Application (SINA), an EWi designed to enable interoperability in decentralized SDN systems. SINA initially used active replication to ensure consistency but later incorporated a Q-learning-based quorum replication strategy to improve scalability and resource efficiency. The literature has proven that there is an urgent need to develop a standardized EWi API that is universally applicable across heterogeneous SDN control planes.

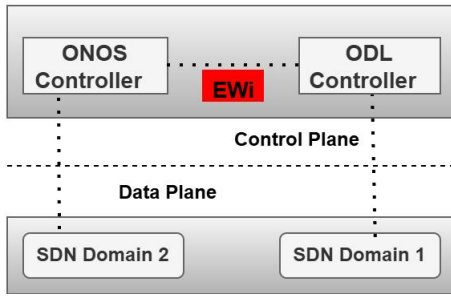


Figure 9: Heterogeneous control plane architecture.

6 Recommendations

This paper recommends developing a unified API for the coordination of ONOS and ODL. There is a urgent need to find a way to connect them without doing major rewrites of their core codebases. Both controllers already share common features which can be exploited for this endeavour as highlighted throughout this paper, such as:

- Shared consensus mechanism based on Raft.
- Standardized messaging protocols based on RESTCONF for inter-controller communication.
- Strong consistency across controller instances but implemented differently.
- Unified data modelling layer using YANG to ensure protocol compatibility.

The important question here is where the unified API is going to be implemented. Whether directly on both the controllers, on an external middleware layer or a hybrid approach where minor extensions are implemented on each controller while offloading most logic to a middleware layer. Considering the complexity of modifying two well-established SDN controllers, an external middleware layer makes the most sense for a practical starting point. This option minimizes changes to ONOS and ODL and allows independent development of both controllers. Additionally, this approach has the potential to support SDN controllers other than ONOS and ODL. In the next subsection, the paper recommends an external middleware layer for a unified SDN middleware (USM).

6.1 Unified SDN Middleware Architecture

This section presents the key components of the recommended USM as seen in Figure 10.

- **Unified API Gateway:** Exposes API endpoints to clients such as the Middleware User Interface (MUI) and network administrators to programmatically interact with controllers from a centralised location. The structure of the data exchanged by this API is defined using YANG models which are exposed using JSON or XML to align with RESTCONF conventions. This ensures that ONOS's flat key-value updates and ODL's hierarchical YANG trees are normalized.
- **Translation Layer:** Translates unified API calls into controller-specific requests. It consists of adapters for each controller which interface with their RESTful APIs. For example, it maps key-value pairs to ConsistentMap operations

via ONOS/Atomix and converts flat key-value pairs into ODL/Akka YANG-based DataBroker transactions. To prevent tampering with message exchanges, controllers should offer a secure version of TLS (such as v1.2) or a TLS comparable protocol for communication between the application/data layers and the control layer [52].

- **State Synchronisation Module:** Ensures a consistent network state across ONOS and ODL using the controllers RAFT APIs i.e. ONOS/Atomix and ODL/Akka. This module may be implemented as an API endpoint that controllers use to share topology information, flow rules, and track shared network state.
- **Monitoring Module:** Collects real-time metrics to optimize performance and detect inconsistencies using existing ONOS and ODL RESTful APIs. Consistency can be confirmed via Raft logs from both controllers.
- **Service Discovery Module:** Tracks and registers all ONOS and ODL instances dynamically using their RESTful API ports.
- **Middleware User Interface:** Uses Unified API Gateway to provide a human-friendly network management console.

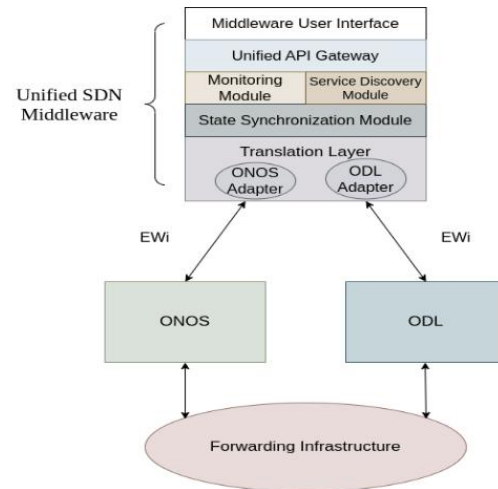


Figure 10: A Unified SDN middleware

All controller instances connect to the USM which may be deployed as a container or VM alongside the controllers. Controllers communicate via the USM and is responsible for delegating tasks to all connected controllers via their adapters. The USM should also be responsible for mediating conflict between the ONOS and ODL clusters. In this case it becomes inherent that the USM should implement a message queue to handle simultaneous requests from multiple controllers.

7 Conclusions and Future Work

This paper presented a review of ONOS and ODL, the two widely adopted distributed SDN controllers. These controllers seek to address the scalability, consistency, and reliability issues that are faced by the centralised controllers. In this paper, the authors have identified several pressing challenges within the heterogeneous SDN

control plane, such as consistency and interoperability, which offer promising avenues for future research. Despite these challenges, a heterogeneous control plane enables greater adaptability, resilience, and scalability, making it a compelling choice for modern, dynamic SDN environments. Our observations indicate that current SDN controllers often suffer from a lack of standardized EWi that connects the ONOS and ODL controllers. In the future, the authors seek to deploy a distributed control plane consisting of these two disparate controllers using a unified API for coordination and synchronisation of the network view.

Acknowledgments

To Robert, for the bagels and explaining CMYK and color spaces.

The authors wish to acknowledge the use of Writefull Paraphrasing Tool in the writing of this paper. This tool was used to assist with improving the language in the paper. The paper remains an accurate representation of the authors' underlying work and novel intellectual contributions.

References

- [1] 2025. *Migration from Akka to Apache Pekko*. <https://pekko.apache.org/docs/pekko/current/project/migration-guides.html> Accessed 06 March, 2025.
- [2] A. Abuarqoub. 2020. A Review of the Control Plane Scalability Approaches in Software Defined Networking. *Future Internet* 12, 3 (2020), 49. <https://www.mdpi.com/1999-5903/12/3/49>.
- [3] S. Ahmad and A. H. Mir. 2021. Scalability, consistency, reliability and security in SDN controllers: a survey of diverse SDN controllers. *Journal of Network and Systems Management* 29 (2021), 1–59.
- [4] B. Almadani, A. Beg, and A. Mahmoud. 2021. DSF: A distributed SDN control plane framework for the east/west interface. *IEEE Access* 9 (2021), 26735–26754.
- [5] A. Azzouni, M. Nguyen, and G. Pujolle. 2019. Topology discovery performance evaluation of OpenDaylight and ONOS controllers. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 285–291.
- [6] F. Bannour, S. Souihi, and A. Mellouk. 2018. Adaptive state consistency for distributed ONOS controllers. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.
- [7] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. 2018. Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Communications Surveys & Tutorials* 20, 1 (2018), 333–354. doi:10.1109/COMST.2017.2782482
- [8] C. Bhatt, V. Sihag, G. Choudhary, P. V. Astillo, and I. You. 2021. A multi-controller authentication approach for SDN. In *2021 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 1–4.
- [9] M. Bjorklund. 2010. *YANG—a data modeling language for the network configuration protocol (NETCONF)*. Technical Report 2070-1721.
- [10] C. D. Cajas and D. O. Budanov. 2020. SDN applications and plugins in the opendaylight controller. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 9–13.
- [11] G. Castellano et al. 2019. A model-based abstraction layer for heterogeneous SDN applications. *International Journal of Communication Systems* 32, 17 (2019), e3989.
- [12] H. M. S. da Silva. 2022. Fairness Centred Framework for the ONOS SDN Controller.
- [13] A. Dixit, K. Kogan, and P. Eugster. 2014. Composing heterogeneous SDN controllers with flowbricks. In *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 287–292.
- [14] S. Gilbert and N. Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News* 33, 2 (2002), 51–59.
- [15] R. Hanmer, L. Jagadeesan, V. Mendiratta, and H. Zhang. 2018. Friend or foe: Strong consistency vs. overload in high-availability distributed systems and SDN. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 59–64.
- [16] A. Hemid. 2017. Facilitation of The OpenDaylight Architecture. In *The 4th Computer Science Conference for University of Bonn Students*.
- [17] E. Hernandez and M. G. Alberto. 2016. Implementation and Performance of a SDN Cluster-Controller Based on the OpenDayLight Framework. Unpublished manuscript or conference paper without formal proceedings.
- [18] N.-T. Hoang, H.-N. Nguyen, H.-A. Tran, and S. Souihi. 2022. A novel adaptive east–west interface for a heterogeneous and distributed SDN network. *Electronics* 11, 7 (2022), 975.
- [19] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan. 2018. Multi-controller based software-defined networking: A survey. *IEEE Access* 6 (2018), 15980–15996.
- [20] S. A. Iqbal. 2018. Green software defined distributed data center model. Unpublished manuscript or non-peer-reviewed work.
- [21] L. D. P. Júnior. 2019. *Towards a software defined network based multi-domain architecture for the internet of things*. Ph. D. Dissertation. ISCTE-Instituto Universitario de Lisboa (Portugal).
- [22] X. Kang. 2023. Optimization of Distributed SDN Controller Placement Considering East-West Interface Overhead. Unpublished manuscript or non-peer-reviewed work.
- [23] M. Karakus and A. Duresi. 2017. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks* 112 (2017), 279–293.
- [24] M. Karatisoglou, K. Choumas, and T. Korakis. 2019. Controller placement for minimum control traffic in OpenDaylight clustering. In *2019 IEEE 2nd 5G World Forum (5GWF)*. IEEE, 353–358.
- [25] B. Kemme, A. Schiper, G. Ramalingam, and M. Shapiro. 2014. Dagstuhl seminar review: Consistency in distributed systems. *ACM SIGACT News* 45, 1 (2014), 67–89.
- [26] H. I. Kobo. 2018. *An efficient distributed control system for software-defined wireless sensor networks*. Ph. D. Dissertation. University of Pretoria (South Africa).
- [27] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke. 2019. Efficient controller placement and reelection mechanism in distributed control system for software defined wireless sensor networks. *Transactions on Emerging Telecommunications Technologies* 30, 6 (2019), e3588.
- [28] P. Lin, J. Bi, and Y. Wang. 2013. East-west bridge for SDN network peering. In *Frontiers in Internet Technologies: Second CCF Internet Conference of China, ICoC 2013, Zhangjiajie, China, July 10, 2013, Revised Selected Papers*. Springer, 170–181.
- [29] F. Lucrezia. 2018. Network Infrastructures for Highly Distributed Cloud-Computing.
- [30] Ntshuxeko Makondo, Hlabishi I. Kobo, and Topsyde E. Mathonsi. 2023. The Latest Developments in Software Defined Networking: Adoption Rate and Challenges. In *2023 IEEE AFRICON*. IEEE, 1c–6. doi:10.1109/AFRICON55910.2023.10293567
- [31] N. Makondo, H. I. Kobo, T. E. Mathonsi, and D. Du Plessis. 2024. Implementing an Efficient Architecture for Latency Optimisation in Smart Farming. *IEEE Access* 12 (2024), 140502–140526.
- [32] N. Makondo, H. I. Kobo, T. E. Mathonsi, D. Du Plessis, T. M. Makhosa, and L. Mamushiane. 2024. An efficient architecture for latency optimisation in 5G using Edge Computing for uRLLC use cases. In *2024 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*. IEEE, 1–7.
- [33] L. Mamushiane, A. Lysko, and S. Dlamini. 2018. A comparative evaluation of the performance of popular SDN controllers. In *2018 Wireless Days (WD)*. IEEE, 54–59.
- [34] L. Mamushiane and T. Shoji. 2021. A QoS-based evaluation of SDN controllers: ONOS and OpenDayLight. In *2021 IST-Africa Conference (IST-Africa)*. IEEE, 1–10.
- [35] J. Medved, R. Varga, A. Tkacik, and K. Gray. 2014. OpenDaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE international symposium on a world of wireless, mobile and multimedia networks 2014*. IEEE, 1–6.
- [36] L. V. Morales, A. F. Murillo, and S. J. Rueda. 2015. Extending the floodlight controller. In *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE, 126–133.
- [37] M. Nkosi, A. Lysko, L. Ravhuanzwo, T. Nandeni, and A. Engelberent. 2016. Classification of SDN distributed controller approaches: a brief overview. In *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE, 342–344.
- [38] Y. E. Oktian, S. Lee, H. Lee, and J. Lam. 2017. Distributed SDN controller system: A survey on design choice. *Computer Networks* 121 (2017), 100–111.
- [39] Open Networking Foundation (ONF). [n. d.]. Open Network Operating System (ONOS). <https://opennetworking.org/onos/>. Accessed: 2025-03-03.
- [40] D. Ongaro and J. Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 305–319.
- [41] OpenDaylight. [n. d.]. Automating networks of any size & scale. <https://www.opendaylight.org/>. Accessed: 2025-03-03.
- [42] OpenDayLight. 2025. *Setting Up Clustering*. <https://docs.opendaylight.org/en/latest/getting-started-guide/clustering.html#akka-to-pekko-migration> Accessed 06 March, 2025.
- [43] Arjun Panda, Colin Scott, Ali Ghodsi, Teemu Koponen, and Scott Shenker. 2013. Cap for networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 91–96.
- [44] M. Poke, T. Hoefler, and C. W. Glass. 2016. Allconcur: Leaderless concurrent atomic broadcast (extended version). *arXiv preprint arXiv:1608.05866* (2016).
- [45] H. Qi, K. Li, H. Qi, and K. Li. 2016. Management system of heterogeneous software-defined networking controllers. In *Software Defined Networking Applications in Distributed Datacenters*. 57–65.

- [46] J. A. Rahim, R. Nordin, and O. A. Amodu. 2024. Open-Source Software Defined Networking Controllers: State-of-the-Art, Challenges and Solutions for Future Network Providers. *Computers, Materials and Continua* 80, 1 (2024), 747–800.
- [47] B. Ribes Garcia. 2015. OpenDaylight SDN controller platform.
- [48] L. V. Ruchel, R. C. Turchetti, and E. T. de Camargo. 2022. Evaluation of the robustness of SDN controllers ONOS and ODL. *Computer Networks* 219 (2022), 109403.
- [49] Y. Saito and M. Shapiro. 2005. Optimistic replication. *ACM Computing Surveys (CSUR)* 37, 1 (2005), 42–81.
- [50] E. Sakic and W. Kellerer. 2017. Response time and availability study of RAFT consensus in distributed SDN control plane. *IEEE Transactions on Network and Service Management* 15, 1 (2017), 304–318.
- [51] E. Sakic and W. Kellerer. 2018. Impact of adaptive consistency on distributed SDN applications: An empirical study. *IEEE Journal on Selected Areas in Communications* 36, 12 (2018), 2702–2715.
- [52] S. Scott-Hayward. 2015. Design and deployment of secure, robust, and resilient SDN controllers. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 1–5.
- [53] J. Singh. 2023. Deployment of SDN Controller and configuring Cisco devices using NETCONF.
- [54] D. Smyth, D. O'Shea, V. Cionca, and S. McSweeney. 2019. Attacking distributed software-defined networks by leveraging network state consistency. *Computer Networks* 156 (2019), 9–19.
- [55] R. Subhashini. 2023. Review of the SDN Architecture with Various API Controllers. In *2023 International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMK-MATE)*. IEEE, 1–6.
- [56] A. Tootoonchian and Y. Ganjali. 2010. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, Vol. 3.
- [57] R. Vilalta et al. 2021. Experimental evaluation of control and monitoring protocols for optical SDN networks and equipment [Invited Tutorial]. *Journal of Optical Communications and Networking* 13, 8 (2021), D1–D12.
- [58] P. Vizarreta, K. Trivedi, V. Mendiratta, W. Kellerer, and C. Mas-Machuca. 2020. DASON: Dependability assessment framework for imperfect distributed SDN implementations. *IEEE Transactions on Network and Service Management* 17, 2 (2020), 652–667.
- [59] K. Xingyuan, K. Takahashi, C. Nakasan, K. Ichikawa, and H. Iida. 2023. Multi-Objective Optimization of Controller Placement in Distributed ONOS Networks. In *2023 Eleventh International Symposium on Computing and Networking (CANDAR)*. IEEE, 76–85.
- [60] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu. 2019. Software-defined wide area network (SD-WAN): Architecture, advances and opportunities. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–9.
- [61] P. Yi, T. Hu, Y. Hu, J. Lan, Z. Zhang, and Z. Li. 2021. SQHCP: Secure-aware and QoS-guaranteed heterogeneous controller placement for software-defined networking. *Computer Networks* 185 (2021), 107740.
- [62] H. Yu, K. Li, H. Qi, W. Li, and X. Tao. 2015. Zebra: An east-west control framework for SDN controllers. In *2015 44th International Conference on Parallel Processing*. IEEE, 610–618.
- [63] H. Yu, H. Qi, and K. Li. 2020. WECAN: an efficient west-east control associated network for large-scale SDN systems. *Mobile Networks and Applications* 25, 1 (2020), 114–124.
- [64] Muhammed Nura Yusuf, Kamalrulnizam Bin Abu Bakar, Babangida Isyaku, and Fadhil Mukhlif. 2023. Distributed Controller Placement in Software-Defined Networks with Consistency and Interoperability Problems. *Journal of Electrical and Computer Engineering* 2023, 1 (2023), 6466996. doi:10.1155/2023/6466996
- [65] B. Zhang, X. Wang, and M. Huang. 2018. Adaptive consistency strategy of multiple controllers in SDN. *IEEE Access* 6 (2018), 78640–78649.
- [66] A. A. Zopellaro Soares et al. 2021. SDN-based teleprotection and control power systems: A study of available controllers and their suitability. *International Journal of Network Management* 31, 3 (2021), e2112.