

Article

An Adaptive State Consistency Architecture for Distributed Software-Defined Network Controllers: An Evaluation and Design Consideration

Rawan Alsheikh ^{1,*} , Etimad Fadel ¹ and Nadine Akkari ²

¹ Faculty of Computing and Information Technology, King Abdulaziz University, Abdullah Sulayman, Jeddah 21589, Saudi Arabia; eafadel@kau.edu.sa

² Computer Science and Information Technology Department, Jeddah International College, Ibn Rasheed Elfehri, Jeddah 23831, Saudi Arabia; n.akkari@jiccollege.edu.sa

* Correspondence: rsalshaikh@kau.edu.sa

Featured Application: A potential application of interest involves applications that can tolerate some level of inconsistency, such as routing or load-balancing applications.

Abstract: The Physically Distributed Logically Centralized (PDLC) software-defined network (SDN) control plane is physically dispersed across several controllers with a global network view for performance, scalability, and fault tolerance. This design, providing control applications with a global network view, necessitates network state synchronization among controllers. The amount of inter-controller synchronization can affect the performance and scalability of the system. The absence of standardized communication protocols for East-bound SDN interfaces underscores the need for high-performance communication among diverse SDN controllers to maintain consistent state exchange. An inconsistent controller's network view can significantly impact network effectiveness and application performance, particularly in dynamic networks. This survey paper offers an overview of noteworthy AI and non-AI solutions for PDLC SDN architecture in industry and academia, specifically focusing on their approaches to consistency and synchronization challenges. The suggested PDLC framework achieves an adaptive controller-to-controller synchronization rate in a dynamic network environment.

Keywords: distributed software-defined network controller; Physically Distributed Logically Centralized (PDLC); inter-controller synchronization; consistency



Citation: Alsheikh, R.; Fadel, E.; Akkari, N. An Adaptive State Consistency Architecture for Distributed Software-Defined Network Controllers: An Evaluation and Design Consideration. *Appl. Sci.* **2024**, *14*, 2627. <https://doi.org/10.3390/app14062627>

Academic Editors: Panagiotis Sarigiannidis, Thomas Lagka, Alexandros-Apostolos Boulogeorgos, Vasileios Argyriou and Pantelis Angelidis

Received: 12 February 2024

Revised: 29 February 2024

Accepted: 18 March 2024

Published: 21 March 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Software-defined networking (SDN), a rapidly growing networking management paradigm, has been investigated and enhanced, including efforts to enhance consistency and standardization in various aspects [1–5].

The distributed SDN controllers are interconnected in several ways, such as hierarchical or flat. Flat architecture can be Physically Distributed Logically Centralized or Fully Distributed [6,7].

Focusing on a Physically Distributed Logically Centralized (PDLC) SDN control plane framework, each controller is responsible for its domain by addressing the failure in its domain data plan and traffic flow congestion. These changes must be communicated across all controllers in different domains within the cluster to attain a globally consistent view in a timely manner. However, achieving such a time rigorous level of consistency while preserving good performance is a complicated task [4]. A key concern is balancing performance and consistency without altering SDN protocols [8].

Several challenges with the PDLC control of SDNs demand particular consideration, including scalability, reliability, consistency and synchronization overhead, interoperability, East–West interface implementations, and security, as depicted in Figure 1 [9,10].



Figure 1. The main challenges of Physically Distributed Logically Centralized SDN control.

Reliability and scalability, often viewed as primary concerns in centralized SDN control architecture, are equally crucial in developing a PDLC SDN architecture. Furthermore, examining consistency issues while making trade-offs for an SDN controller platform is crucial. Consistency requires a synchronization process, which can present a harmful overhead for the system [9,11,12]. We will focus on the consistency and synchronization overhead challenge.

Designing applications that run on the SDN approach on top of the distributed environment is a crucial task because of the complexity of synchronizing controllers and the network state, which will affect the performance of the applications and network efficiency [13,14]. The consistency management is handled with an inter-controller synchronization process, as per the deployed application's consistency level preferences (strong or eventual) [15].

The frequency of distributed control plane-wide state updates and the eventual resolution of state conflicts are governed by consistency levels. In loose consistency levels, synchronization overhead decreases, but the possibility of state conflicts arising increases. Contradictorily, strict consistency levels result in more frequent synchronization, resulting in more overhead in the control plane [4,13,16–19]. The resulting overhead will increase latency and affect the scalability of the system. We refer to latency (sometimes called responsiveness) as the time the system takes to reply to flow requests. Trade-offs between various consistency models and responsiveness features are present with PDLC topology [3,5,6,9,16,20].

As observed from the literature, the PDLC SDN architecture inter-controller synchronization process can be applied in static or adaptive approaches.

a. Static Inter-Controller Synchronization Approach

A static synchronization technique provides fixed, predetermined intervals (for instance, every 3–4 s), which can affect the application's performance by utilizing outdated data or irritating the network with unnecessary synchronization messages [18].

b. Adaptive Inter-Controller Synchronization Approach

The adaptive synchronization approach involves adjusting the consistency levels according to the correctness thresholds. This cost-aware approach determines the balance between two major performance metrics: the synchronization overhead and the system accuracy of the distributed control plan [19]. Additionally, it can leverage the performance of the applications that can tolerate some inconsistency.

It is important to strike a balance with adaptive synchronization to avoid excessive overhead or delays in propagating critical updates. Fine-tuning the adaptive synchronization mechanism requires careful analysis and testing to ensure optimal performance and

responsiveness based on the distributed SDN controller deployed application requirements and characteristics.

However, the challenges persist, such as optimizing SDN control techniques in complex, dynamic networks, where a control plane's configuration and forwarding mostly rely on human operation [21]. Recent distributed SDN consistency research that integrated network control with AI is much more flexible than earlier adaptive synchronization attempts since it automates policy learning in any network environment and makes no network assumptions [10].

The flexibility and robustness of the SDN network and its centralized nature can provide a global network view, driving the integration of SDN and AI concepts [10].

We review notable PDLC SDN solutions, including both AI and non-AI approaches from industry and academia. The focus of this work is on how these solutions tackle synchronization and consistency issues. We suggest a design framework that achieves an adaptive controller-to-controller synchronization rate, illustrating the design factors for a state consistency communication protocol. This will help optimize and reduce communication costs while ensuring precise and fast state synchronization amongst dispersed SDN controllers. This framework was developed following research and analysis of open problems and existing solutions.

The remainder of this paper is organized as follows: Section 2 introduces Physically Distributed Logically Centralized (PDLC) SDN architecture and components. Section 3 discusses non-AI consistency solutions for Physically Distributed but Logically Centralized SDN controllers. Section 4 presents AI-based solutions for SDN consistency synchronization. Section 5 discusses a future direction: AI-based state consistency architecture. Section 6 concludes the paper.

2. Physically Distributed Logically Centralized (PDLC) SDN Architecture and Components

Understanding the design and functionality of the Physically Distributed Logically Centralized (PDLC) SDN architecture is essential. Figure 2 illustrates the problem setting of a distributed SDN architecture. This architecture comprises three logical layers: the data layer (Layer P), the control layer (Layer S), and the application layer (Layer A). This representation shares similarities with the Onix SDN control systems [22] and related work [16]. Five key components can be found, each serving a specific purpose: the data layer, connectivity, distributed control plane instances, control logic, and database. In Figure 2, each dashed and dotted green arrow signifies a point of network state exchange within the distributed SDN system.

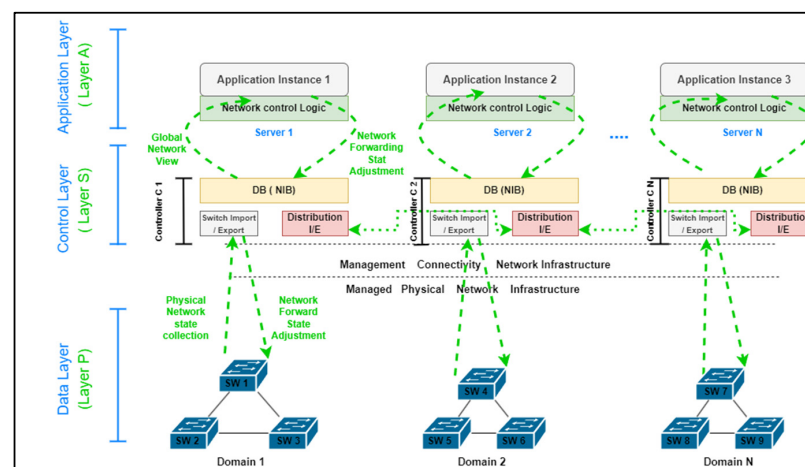


Figure 2. Distributed SDN architecture with network state distribution.

2.1. Data Layer

At the lowest layer depicted in Figure 2, network switches and routers are represented as blue boxes. These forwarding hardware devices store the state of the network data plane Forwarding Information Base (FIB), such as configured port speeds and TCAM (Ternary Content Addressable Memory) entries and the associated meta-data, including flow ports and packet counters. Additionally, they offer an interface facilitating the controller's ability to modify the network state by adjusting forwarding table entries that dictate their behavior. These network components merely require software support for this interface, such as OpenFlow, and basic connectivity. The collective set of switches directly linked to a specific SDN controller, along with all hosts connected to those switches, constitutes the domain of that controller [15,16,22].

2.2. Connectivity

The connectivity infrastructure must facilitate bidirectional communication between the control plane instances, illustrated by the green dotted arrows connecting the red boxes that represent the distribution I/E module in Figure 2. Moreover, bidirectional communication between switches and control plane instances is required, depicted by the green dashed arrows between the grey boxes representing the switch Import/Export module in Figure 2, with the capability to support convergence in case of link failure. Standard routing protocols such as OSPF or IS-IS are employed in the connectivity architecture to establish and maintain the forwarding state [22].

The eastbound interface, indicated by the red boxes in Figure 2, serves as a means for SDN controllers to communicate and exchange information among themselves. Conversely, a westbound interface facilitates information exchange between distributed legacy control systems and the SDN control plane [4,5].

In PDLC SDN architecture, controllers are interconnected horizontally, often referred to as flat connections [6,7]. However, the cost of intercommunication among controllers in a distributed SDN environment remains a significant challenge [23].

2.3. Distributed Control Plane Instances

The control plane operates as a distributed system consisting of a cluster of one or more physical servers, as depicted in Figure 2. Each server has the capability to operate multiple control plane instances, also known as the "Network Operating System" (NOS) [16,22].

The controllers within each domain serve as the SDN's core element and realize the state management (Layer S). They are responsible for facilitating programmatic network access to the control logic, allowing for the reading and writing of the network state. Additionally, the control plane instances communicate the network state to other instances within the cluster through distribution I/E, as illustrated in Figure 2. Consequently, the SDN can present an instance of the control application (Layer A) with an abstraction of the physical network state [16,22].

Each controller maintains a Network Information Base (NIB) data structure stored in the database, represented by the yellow boxes in Figure 2. That provides an application with a view of the state of the network [16].

2.4. Control Logic

The network control logic is deployed atop the controller's API, as illustrated by the green box in Figure 2. The controller only provides the primitives necessary for accessing the correct network state. The control logic defines the intended network behavior [16,22].

2.5. Database

A distributed control plane's most essential component is the Database Management System, as shown by the yellow box in Figure 2. This database serves as the repository for all intra- and inter-domain information, constituting the core of each controller. It can be leveraged as a means of information exchange among controllers, eliminating the need for

a specific communication protocol. Various SDN solutions surveyed utilize either relational (SQL) or NoSQL databases [6].

A significant challenge arises in preserving and disseminating the network state within the distributed control plane. A non-distributed database may exhaust system memory. Alternatively, the number of network events (produced by the database) or the amount of work necessary to manage them could increase to the point where they overwhelm a single control plane instance's CPU. The distributed database framework allows the control plane to scale to accommodate large networks and withstand controller and network failures [22].

3. Non-AI Consistency Solutions for PDLC SDN Controllers

Our study seeks to provide the current state of the notable PDLC SDN industry and academia solutions that can use static or adaptive synchronization approaches. We analyze how each controller solution may respond to many of the distributed environment challenges.

3.1. Static Consistency Synchronization Techniques

First, we examine some notable industry solutions for PDLC SDN controllers, such as HyperFlow [11], Onix [22], ONOS [24], ElstiCon [25], and Orion [26] (See Table 1). Each proposed solution impacts several distributed SDN controller challenges, including scalability, reliability, interoperability, consistency, synchronization overhead, Technological Readiness Level (TRL), East–West communication protocol, and implementation programming language [2,4,15,27–29]. The logical classification used in this paper was adopted from [6,9].

We consider the Technological Readiness Level (TRL) scale. It was recommended by the European Commission and published in [30]. It serves as a measure of the maturity of a certain technology. The TRL of technology must be taken into account due to the sophistication of the mechanisms we are dealing with in this study in order to minimize development efforts as much as possible [6].

Second, several works are presented by academia to solve the inconsistency problem in PDLC SDN controllers in WANs. Some work uses static consistency.

Table 2 compares some of the notable static synchronization techniques for PDLC SDN consistency in academia. It tabulates these solutions based on scalability, robustness (reliability), consistency, synchronization approach, synchronization overhead, interoperability, East–West communication protocol, and implementation programming language.

Many recent articles have examined the architecture of a large-scale or global SDN controller with a static consistency synchronization approach, including [11,31–33]. Each SDN-distributed controller platform has its own communication method. This is a major drawback of these controllers since it makes it challenging for the SDN network to share data between various domains [1].

Almadani, Beg, and Mahmoud [34] introduce the Distributed SDN control plane Framework (DSF). It is a modular design for constructing an East–West interface [34].

The proposed work synchronizes topologies by utilizing Data Distribution Service (DDS), a standardized data-centric Real-Time Publish-Subscribe (RTPS) approach. The local network state changes published by the controllers into the domain data space with a predefined structure of topic and Quality of Service (QoS) attributes [34].

The researchers have justified the aim of using a data-centric RTPS approach in the SDN control plane domain. This approach allows many communicating participants to interact in real-time and in a synchronized manner, providing secure end-to-end data connectivity. This is in contrast to the broker-based or client–server models which can cause extra processing delay, introduce a single point of failure, if a failure occurs, and limit concurrent read/write [34].

The problem with DSF is that every time an update is made, they broadcast it to every other controller using the active replication mechanism. Consequently, the broadcasting strategy lowers the network's performance [5].

Table 1. Notable Industry Solutions for Physically Distributed Logically Centralized (PDLC) SDN Controllers.

Distributed Controller	Year	Distributed Architecture	Connectivity Model	Scalability	Robustness	Consistency Model	Sync. Approach	Sync. Overhead	Interoperability	TRL	East–West Comm. Protocol	Programming Language
HyperFlow [11]	2010	Log. Centralized	Flat	Medium	Low	Eventual	Static	Medium	No	TRL3	Broker-based P2P	C++
ONIX [22]	2010	Log. Centralized	Flat	Medium/Low	Medium	Weak/Strong	Static	Medium/High	No	TRL7	ZooKeeper API	Python or C
ONOS [24]	Dec. 2014	Log. Centralized	Flat	Medium/Low	Medium	Medium	Static	Medium/High	No	TRL9	Atomix DB (RAFT Algorithm)	Java
ElastiCon [25]	2014	Log. Centralized	Mesh	Medium	Undefined	Eventual	Static	Medium	No	TRL3	DB and TCP Channel	Java
Orion [26]	2021	Log. Centralized	Hybrid	High	Undefined	Eventual	Static	Medium	No	TRL3	Publish-Subscribe DB (NIB)	C++

Table 2. Notable Static Synchronization Techniques for Physically Distributed Logically Centralized (PDLC) SDN Consistency In Academia.

Architecture	Year	Distributed Architecture	Connectivity Model	Scalability	Robustness	Consistently	Synchronization Approach	Synchronization Overhead	East–West Comm. Protocol	Interoperability	Programming Language
DSF [34]	2021	Log. Centralized	T-Model, Hierarchical, Flat	Medium	Medium	Strong	Static	High	RTPS	No	Java
WECAN [1]	2019	Log. Centralized	Flat	Low	Medium	Strong	Static	High	FLEX	No	NA
VNF-Consensus [8]	2020	Log. Centralized	Flat	Medium	Medium	Strong	Static	Low	REST	Yes	Python

In ref. [1], Haisheng Yu, Heng Qi, and Keqiu Li introduced WECAN: an efficient West–West-East control associated network for large-scale SDN systems. WECAN proposes to provide a scalable network control layer interoperable with an East–West interface. This work aims to solve the communication problem between different controllers [1].

The substantial WECAN component is the decision layer with several servers known as decision elements that are directly connected to the network. Recalling the original SDN layers, WECAN’s decision layer is a newly developed layer between the SDN controller layer and the application layer. All network control choices are made by the decision layer. WECAN creates a user-friendly and practical web control interface that can control a wide range of SDN controllers [1].

However, only two controllers may integrate with WECAN: Maestro [35] and Floodlight [5]. Moreover, WECAN stores information using a database that has ten core tables. This rigid design will limit the system’s scalability.

The work in [8] introduced VNF-Consensus: a virtual network function for maintaining a consistent distributed SDN control plane. The distributed SDN control plane can keep consistency by using NFV (Network Function Virtualization).

The solution depends on Virtual Network Function (VNF), which they call VNF-Consensus that performs Paxos [36], maintaining strong consistency across distributed control planes by synchronizing their actions. The controller will access a VNF-Consensus instance running on a host different from the controllers. VNF-Consensus will handle all decisions and make them systematically without the controller’s involvement. In this situation, an SDN controller is able to synchronize their communication and receive decisions [8].

Additionally, they deploy VNF-Consensus communication with controllers by a REST interface. Neither the switches nor the SDN protocol need to be altered to adopt the proposed technique. This approach outperforms having the controllers themselves be in charge of synchronizing and ensuring the consistency of network activities by a large margin [8].

There are other works in the literature that are focused on the static consistency for fully distributed architecture (logically and physically distributed), such as [37,38]; with these solutions, they focus on the read/write operations from the replicas and their concurrency and consistency in the distributed database system, which is out of our focus.

3.2. Adaptive Consistency Synchronization Techniques

Adaptive consistency has recently been used in several works of literature. In ref. [13], they define an adaptive controller as “One that can autonomously and dynamically tune its configuration in order to achieve a certain level of performance measured in predefined metrics and based on its requirements.”

The adaptive consistency module is a tunable consistency module that provides a configurable level of consistency. It encapsulates the distributed information complexity upon multiple controllers. Additionally, it develops a uniform interface that can alter the consistency of this information among different levels of consistency. The maintained network state information must reflect the changes in the fluctuating network environment to avoid inefficient rules in forwarding physical devices. By this model, the controllers can make collective designs and produce a single flow using measurement and calculation [13].

Table 3 compares some of the notable non-AI adaptive synchronization techniques for PDLCD SDN consistency in academia. It compares these solutions based on scalability, robustness (reliability), consistency, synchronization approach, synchronization overhead, interoperability, East–West communication protocol, and implementation programming language.

There are two types of adaptive solutions in the literature; some are focused on adaptive consistency according to a target application, such as the performance of load-balancing applications. Others are concerned with dynamically adapting the consistency level according to the deployed applications running on the distributed controllers at

runtime. Different network applications have different consistency requirements. This depends on the recorded state inconsistencies and the threshold assigned to each application preference, which is called continuous adaptive consistency.

Table 3. Notable Non-AI Adaptive Synchronization Techniques for Physically Distributed Logically Centralized (PDLC) SDN Consistency in Academia.

Architecture	Year	Distributed Architecture	Connectivity Model	Scalability	Robustness	Consistently	Synchronization Approach	Synchronization Overhead	East–West Comm. Protocol	Interoperability	Programming Language
Adaptive State Consistency for Distributed ONOS Controllers Towards Adaptive State Consistency in Distributed SDN Control Plane Adaptive Consistency for Distributed SDN Controllers	2018	Log. Centralized	Flat	Medium	Medium	Eventual	Continuous Adaptive	Medium	Events shared by the distributed core of ONOS	No (Homogeneous)	Java
	2017	Log. Centralized	Flat	High	Medium	Eventual	Continuous Adaptive	Low	Event-based	NA	NA
	2016	Log. Centralized	Flat	High	Medium	Eventual	Adaptive	Low	Not specified	Yes	Python

Continuous adaptive consistency is explored in [18,19,39], where authors aimed to minimize synchronization overheads and enhance system performance and scalability. Their works confirm that traditional static approaches to eventual consistency may not be optimal in dynamic SDN environments.

The work on adaptive state consistency for distributed Open Network Operating System (ONOS) controllers, as per [18], was compared to the static eventual consistency used in the ONOS approach. Instead of running for each controller replica, the anti-entropy process was run periodically at static constant intervals (each 3–5 s). That caused significant overhead and affected the system’s performance and scalability. They scheduled the anti-entropy process only when there was a risk to the system consistency [18].

As per [19], adaptive state consistency in the distributed SDN control plane is introduced. In this model, strict synchronization is used for operations that significantly impact the network resources, while less important changes are periodically propagated across cluster nodes. Every resource state an SDN application uses has a consistency level (e.g., topology manager and routing). Based on the experienced state convergence after a non-synchronization time has passed and the inefficiencies brought on by operations using the stale state as inputs, the consistency level is adjusted [19].

Their model provides a transition of global controller decisions into local ones by designating all controllers as rulers of all switches. This method will reduce controller-to-controller synchronization efforts in the distributed control plane. In their concept, a single controller can apply a global route configuration. That, in current controllers, requires message delivery across the entire distributed control plane to all switches on the path, where the controller’s administrative domain may span the entire network [19].

In the same idea of continuous adaptive consistency, Aslan and Matrawy [40] proposed a clustering-based consistency adaptation strategy for distributed SDN controllers. They use two online clustering techniques, sequential and incremental k-means, to map an indicator for application performance with the level of consistency that is feasible in the underlined consistency approach. Their results show that this technique can produce satisfying results in cluster numbers greater than or equal to 50 [40].

Other work is concerned with the adaptive consistency according to some target applications, such as the performance of the load-balancing or critical path-establishing applications. This type of application might be willing to use a weaker consistency model.

Aslan and Matrawy [41] argue that in order for intelligent SDN applications to take the correct action at the correct time, the network state needs to be monitored. In their work, they compare the effect of passive and active methods for collecting network states on the load-balancing applications that run on SDN controllers. Their results showed that if the network has low traffic fluctuations, the passive method acts better than the active ones. However, load-balancing applications are more resilient to the fluctuating traffic load in the active methods than passive ones. The active collection state method that they experiment with depends on switching polling periodically [41].

In ref. [13], they proposed adaptive consistency for distributed SDN controllers. They expanded the standard SDN controller design. This architecture comprises a measurement and/or prediction module, a control module, and a feedback loop. The control module would modify the system's consistency level in accordance with the performance level determined by the measurement module.

They use a load-balancing application as a case study. Every other controller tracks how many flows are assigned to it via its local domain server. The relative difference was employed as the application performance metric. The smaller the relative difference, the greater the performance. The tunable consistency module will make the synchronization duration available to the adaptation module as a configurable parameter (an indicator of consistency level). As determined by the adaptation module, the value of the synchronization period will be automatically adjusted according to the load balancer's performance [13].

3.3. Discussion and Open Issues

We present a summary of some of the notable works in industry and academia of PDLC SDN control planes that used the static or adaptive SDN consistency methodologies summarized in Tables 1–3.

Interoperability between SDN controllers is a prerequisite for these distributed controllers. However, we can notice from the comparison tables (Tables 1–3) that each SDN-distributed controller platform has its own communication method. This makes it difficult for the SDN network to share data across various domains. This highlights the East–West interface implementation challenge of PDLC SDN architecture as depicted in Figure 1.

Furthermore, communication between controllers is typically facilitated through network protocols and messaging mechanisms. As observed from the literature [6,11,15,34], there are some common approaches for enabling communication and updates between controllers. Some examples of distributed controller systems use a mixture of the following:

- **Message passing:** Controllers can exchange messages with each other to communicate updates and exchange information. This can be performed using various messaging protocols such as MQTT, AMQP, or custom protocols implemented over TCP/IP or UDP. These include ElstiCon [25], which employs message passing as their communication approach among the controllers in the pool.
- **Publish-subscribe model:** Controllers can subscribe to specific topics or events of interest and publish updates on those topics. Other controllers that are subscribed to those topics will receive the updates. This model allows for asynchronous communication and decoupling between controllers. Orion [26] utilizes a publish-subscribe approach for sharing state across its applications.
- **Remote procedure calls (RPC):** Controllers can expose remote procedures or methods that other controllers can invoke to trigger updates or request information, such as using RESTful APIs. Floodlight [42] is an open-source SDN controller that exposes RESTful APIs for remote procedure calls.
- **Shared data store:** Controllers can access a shared data store, such as a database or distributed key-value store, to read and write shared information. This shared data store can be a centralized repository where controllers can update and retrieve data. HyperFlow [11] is an SDN controller that falls under the publish-subscribe category and the “shared data store” category. It utilizes a WheelFS [43] to read and write shared information for communication and coordination among controllers. In addition to the publish-subscribe model, Orion [26] also employs a centralized, in-memory data store called the Network Information Base (NIB), which acts as a shared state repository across the Orion applications.
- **Event-based communication:** Controllers can generate and listen to events to communicate updates. Events can be published by one controller and subscribed to by others, allowing for event-driven updates and notifications. ONOS (Open Network Operating System) [24] is an SDN controller that fits into multiple communication approaches.

It supports both the “publish-subscribe model” and “event-based communication” approaches. Controllers in the ONOS framework can subscribe to specific topics or events of interest and publish updates on those topics, enabling asynchronous and event-driven communication.

The choice of communication approach depends on factors such as the system architecture, scalability requirements, latency constraints, and the nature of the communicated updates.

Although standard static eventual consistency is commonly used in modern SDN systems to achieve effective scalability, it suggests a synchronization process at fixed periods (Tables 1 and 2). It is argued that it provides no bounds on the tolerated state inconsistencies by SDN applications. This may result in utilizing stale data and disrupting the network by sending unneeded synchronization messages. This creates an overhead and affects the deployed application’s performance and scalability [18].

The common controller designs ONOS [24] and ONIX [22] utilize static consistency synchronization techniques (see Table 1). They offer APIs for choosing either a strong or eventual mode of consistency for their distributed state primitives, in an effort to address the adaptive consistency and scalability issue. However, the SDN application must have the active model of state consistency hard-coded because it does not change at runtime and cannot be deployed without knowing the precise network limitations [19].

However, ONOS [24], a cutting-edge SDN controller, uses the anti-entropy protocol to achieve eventual consistency. The anti-entropy protocol’s main idea is that controllers randomly synchronize with one another using a gossip algorithm. Recently, more concerns have been raised about the effectiveness of their protocol despite the fact that it can eventually reach consistency [44].

Additionally, the static consistency solutions introduced in the literature have a number of shortcomings. First, they are unable to handle an SDN system that is heterogeneous and created by various vendors. Second, their consistency approach is straightforward in that it ignores the trade-off between the consistency level and SDN network performance. Third, most existing solutions in the literature try to build a consistent control plane by incorporating additional features for synchronization across the control plane. The main disadvantage of this approach is the overhead on the controllers. On the other hand, avoiding the increasing overhead on controllers by letting switches synchronize their behaviors has a drawback as they need an alteration of the SDN protocol.

The use of a tunable adaptive consistency module that provides a configurable level of consistency needs to be considered. Adaptive synchronization offers several benefits, including the following:

Adaptability: By dynamically adjusting the synchronization rate, the distributed controllers can effectively respond to network environment changes. For example, during periods of high network activity or frequent events, the synchronization rate can be increased with respect to the application SLA to ensure the timely propagation of updates and better responsiveness. Conversely, during periods of relative network stability, the synchronization rate can be reduced to minimize overhead and conserve resources.

Scalability: Adaptive synchronization helps improve scalability in distributed SDN controllers. A fixed synchronization rate may become inadequate as the network grows in size or complexity, and it can irritate the network with unneeded messages. By dynamically adjusting the synchronization rate, the system can adapt to the changing demands of the network, ensuring efficient resource utilization and acceptable performance levels.

However, there are several works on adaptive consistency in the literature. It is not considered fully adaptive. It needs to be more flexible and automated to any network environment and make no network assumptions.

The studies on adaptive synchronization rates, such as [13,19], suggest a dynamic adaption of synchronization rates among controllers. This is similar to the goal of other methodologies that use DRL, such as [44,45], which suggest far more adaptability than these efforts. It makes no network assumptions and automates policy learning under any network setting.

The comparison in Tables 1–3, as well as the literature, shows that there are many ways to construct a PDLC SDN architecture; some of these methods performed better than others in some areas while falling short in others. Indeed, none of the suggested SDN controller platforms addressed all the issues needing to be solved for a successful adaptive synchronization for distributed SDN deployment.

4. AI-Based Solutions for SDN Consistency Synchronization

With the tremendous technological evolution and the increasing involvement of digital aid in everyday life creating extensive network traffic, the current network should be improved to be more intelligent, scalable, and efficient.

The lack of AI adoption in networks arises from one major challenge: the distributed nature of networks, where each router or switch has only a fractional control and view of the whole system. Hence, learning and tackling knowledge from nodes that have a partial view and operate upon a small section of the system is extremely complex [46]. For these reasons, the knowledge plane introduced in 2003 by D. Clark for the Internet [47] is not heavily deployed or prototyped in the area of the network.

The new direction toward logical centralization of control presented in the SDN paradigm will smoothly tame the complexity in the distributed system environment. In this context, the knowledge plane can benefit from different Machine Learning (ML) techniques that collect and utilize network knowledge to control and manage the network [46,48]. Many researchers, such as the authors of [47–49], support the integration of SDN and AI. Integration of AI techniques and SDN abstraction concepts can cause more adaptive network behavior [10].

ML is the backbone of artificial intelligence, and it can learn from the behavior of networks by applying network analytics. The learning algorithms that benefit from the network use the current and verified information provided by the analytical platforms. Three approaches are used in this process: supervised learning, unsupervised learning, and Reinforcement Learning (RL) [48,49].

While it could be easy to consider RL to be an unsupervised learning method because it does not require examples of appropriate behavior, RL focuses on maximizing a reward rather than trying to uncover hidden structures. As a result, it has been considered that unsupervised learning and supervised learning are the first two machine learning paradigms, RL is the third, and there may be other paradigms [50].

ML methods, especially deep learning (DL), have been utilized to handle complicated problems without explicit programming. These techniques use training data or the environment to simulate and learn network behavior [48].

DL has been used to enhance the efficiency of RL algorithms, enabling the application of RL in more complex situations. As a result, so-called deep reinforcement learning (DRL) is produced by combining DL and RL. DRL started in 2013 through Google Deep Mind [51].

A Deep Neural Network (DNN) can be used to generalize and approximate policy in RL, which is called deep reinforcement learning (DRL). DRL algorithms have recently made significant advancements in AI that are being used in various network-related sectors [49].

4.1. AI-Based SDN-Related Work

In ref. [21], they introduced the network AI architecture. The model comprises three planes: the AI plane, the control plane, and the forward plane. The generation of policies is the role of the AI plane. The AI plane benefits from SDN and monitor approaches to achieve a comprehensive view and control of the whole network [21].

In refs. [52,53], they use the knowledge plane with the other SDN planes. The researchers add a data-driven plane to SDN networks that use the network data generated from the infrastructure to permit network intelligence. Their work advises a general design of data-driven SDN and also gives inspiration for the future evolution of SDN.

The researchers in [54] proposed a data-driven intelligent future network design that integrates a big data engine in the SDN control plane. The big data engine is responsible for

processing the data, analyzing the data, and supporting decisions. Even though the network architecture in this design is constricting for addressing the future Internet content delivery, the data of the big data engine are gained from both application data and network data.

The researchers in [55] proposed a multi-agent controller to enable cognition in software-defined networks. This work enables the intelligence in SDN controllers in order to introduce a knowledge-defined network. The researchers constructed a multi-agent system SDN controller (MAS-SDN). The proposed architecture is composed of two constituents. First, the southbound interface can communicate between the SDN MAS and the network. Second is the multi-agent system, which consists of multi-agents communicating together.

In ref. [55], they justify not using SDN controllers based on ML techniques. The ML can be used to solve a specific issue like flow classification or load balancing. Moreover, this type of work needs large sets in order to train the model. Their solution aims to construct an SDN controller that can think intelligently according to the domain knowledge (rules) and the existing beliefs (conditions). Additionally, in the future, they will be able to build a full cognitive SDN controller when logic-based solutions and machine learning come together.

The researchers in [56] proposed machine learning for network resiliency and consistency. Without consistency, they argued that security could not exist. Implementing security policies on a network might be difficult when there is no way to identify any misconduct [56]. The proposed work claimed to be the first effort at an AI-based consistency verification solution. They target more resilient networks in a privacy-preserving architecture.

In their study, they use DL to categorize whether the flow-table entries of the network nodes are consistent with the controller's view of the network. The system uses a hybrid approach with both controller-level centralized processing and node-level distributed processing across the data plane [56]. The extracted features are entered into a hash process. Then, the vector output is forwarded to the controller level to be verified with the controller network view. If an inconsistency is discovered, a second layer of verification is triggered to identify the event that causes the flow-based attack [56].

In light of recent achievements in using RL methods to address challenging issues, the researchers in [44] try to answer how to get the maximum synchronization benefits among controllers to preserve a logically centralized view under eventual consistency. They introduced a DQ Scheduler: deep reinforcement learning-based controller synchronization in distributed SDN. They considered the problem of synchronization among controllers as a Markov Decision Process (MDP) by developing the Deep-Q (DQ) Scheduler. They formulate the controller policy synchronization development to maximize the MDP performance metric as an MDP that can be solved by employing RL techniques. They utilize DNN to generalize and approximate the estimation of the synchronization policy. An application of interest is inter-domain routing.

The controller synchronization frequencies directly impact the quality of implemented routing paths. They specify the synchronization budget as the maximum number of other controllers that an SDN controller can synchronize with at any given time.

Inspired by the success of the DRL technique in [44], the researchers in [45] introduced a controller synchronization framework. They called it MACS: deep reinforcement learning-based SDN controller synchronization policy design. As far as we are aware, MACS (Multi-Armed Cooperative Synchronization) is the first DRL-based SDN controller synchronization scheduler that produces fine-grained synchronization rules. It is aimed at communication and computes resource optimization [45].

They investigated the problem of controller synchronization with a restricted synchronization budget. In their work, to avoid the significant overhead created by the frequent distribution of synchronization messages between controllers, only a limited number of synchronization messages at a given time were exchanged [45]. This is in contrast to the existing implementation of the distributed controller, such as ONOS [24], where the entirety of the state information is exchanged between the synchronized controllers.

They modeled their work after Dueling Network Architectures [57] and Action Branching Architectures for DRL [58].

However, MACS [45] suggested a central controller whose only role is to set synchronization policies and regulate controller synchronization—either a standalone control unit or one of the distributed controllers that are already in use. MACS’s single-agent DRL is built on logic with centralized control, which is insufficient for distributed control in SDN [59].

The work presented by MARVEL focused on enabling controller load balancing in software-defined networks with multi-agent reinforcement learning as per [59]. MARVEL attempts the first distributed solution of the NP-Complete Software Migration Problem (SMP).

This work proposes a scheme to dynamically control the load balancing between the distributed controllers in the large SDN environment. The distributed processing characteristic of MARVEL makes it desirable for the communications and decision-making of the distributed control plane in SDN. MARVEL uses the DRL technique with each agent. Following thorough training, the MARVEL agents in the controllers can decide on the SMP control approach.

4.2. Discussion and Open Issues

Most SDN applications need to maintain and record network information because it is continuously changing within the application’s present run cycle; the maintained network state information does not reflect the present changes in the network environment. Moreover, several applications do not communicate directly with each other. Subsequently, several inefficient rules existed in forwarding physical devices. To solve this problem, the running application on the controller should make a collective design and produce a single flow. Hence, a controller that is conscious of the current network state and can make critical decisions is needed [55].

Table 4 summarizes some notable works that integrate AI and SDN architecture to create an adaptive SDN consistency. Different connectivity models are presented, including single controllers and flat models. The choice of connectivity models can impact scalability and robustness. Furthermore, the communication between controllers is the same technique used in non-AI solutions. Several programming languages and tools are used across different approaches, such as the GOAL agent programming language, Python, and frameworks like TensorFlow and Keras for deep learning implementations. The integration of AI will reduce synchronization overhead and enhance the distributed system’s scalability.

There are several adaptive consistency inter-controller synchronization works that exploit AI, such as RL and DRL, in the literature. Still, it is in its early stages and is not commonly used; it has not proven its efficiency or the correctness of its result yet, and it needs further investigation.

The majority of current studies only presume that such a logically centralized network view may be obtained with certain synchronization designs. The precise manner in which controllers should synchronize with each other, achieving optimal synchronization to reduce the performance minimization under eventual consistency requirements, is largely ignored.

However, other works, such as [44,45], suggest a central controller whose only role is to set synchronization policies and regulate controller synchronization. This centralized control is insufficient for distributed control in SDN.

Table 4. Notable AI Adaptive SDN Consistency Solutions for Distributed SDN Architecture.

Architecture	Year	Distributed Architecture	Connectivity Model	Scalability	Robustness	Consistently	Synchronization Approach	Synchronization Overhead	East–West Comm. Protocol	Interoperability	Programming Language
A Multi-agent Controller to enable Cognition in Software Defined Network	2018	Log. Centralized (single controller)	Single controller	Very low (one SDN controller)	Low	Eventual	Adaptive	NA	No (single controller)	No	GOAL agent programming language. Additionally, Prolog is used by GOAL to represent knowledge.
Machine Learning for Network Resiliency and Consistency	2019	Logically centralized (single controller)	Flat	Very low (one SDN controller)	Low	Strong consistency (for security architecture)	Adaptive	No Overhead (only one controller)	No	No between different controllers (interoperability between SDN controller and the ARS system, handling an SDN system that is heterogeneous and uses SDN controllers created by various vendors by using REST interface)	NA
DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN	2018	Logically centralized	Flat	High	Medium	Eventual	Adaptive	Low	NA	Yes	Python
MACS: Deep Reinforcement Learning-based SDN Controller Synchronization Policy Design	2019	Logically centralized	Flat	High	Medium	Eventual	Adaptive	Very Low	Through domain controllers broadcasting or receiving control plan messages that have the selected up-to-date basic information of synchronization	Yes	Python
MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning	2020	Logically Centralized	Mech	High	High	Eventual	Adaptive	Low	Such as ONOS local controller state data are sent throughout the cluster via events shared.	Yes	Python

5. Future Direction: AI-Based State Consistency Architecture

Constructing an architecture capable of achieving an adaptive controller-to-controller synchronization rate through the integration of AI techniques within a PDLC SDN framework, as illustrated in Figure 3, necessitates the inclusion of additional components or modules beyond those detailed in Section 2 and depicted in Figure 2. We need the following additional components or modules:

- **Synchronization Module:** This module implements an algorithm that dynamically adjusts controllers' synchronization rates based on network conditions and workload. This algorithm should be capable of learning and adapting over time using AI techniques. The distributed controllers work collaboratively under the guidance of the Synchronization Module to synchronize their activities.
- **AI Techniques:** Many recent studies have recommended RL for SDN networks such as [44,45,59]. There are many reasons for that, such as the abundance of data that the SDN switches can afford through the protocol of OpenFlow, and the high heterogeneity of distinct SDN domains. Moreover, SDN networks are considered complex and intricate systems. Subsequently, modeling such system accuracy is mathematically complicated. Due to the lack of restrictions on the structure or dynamicity of the network, model-free RL-based techniques are particularly appealing. They can be used in SDN networks in real-world scenarios.

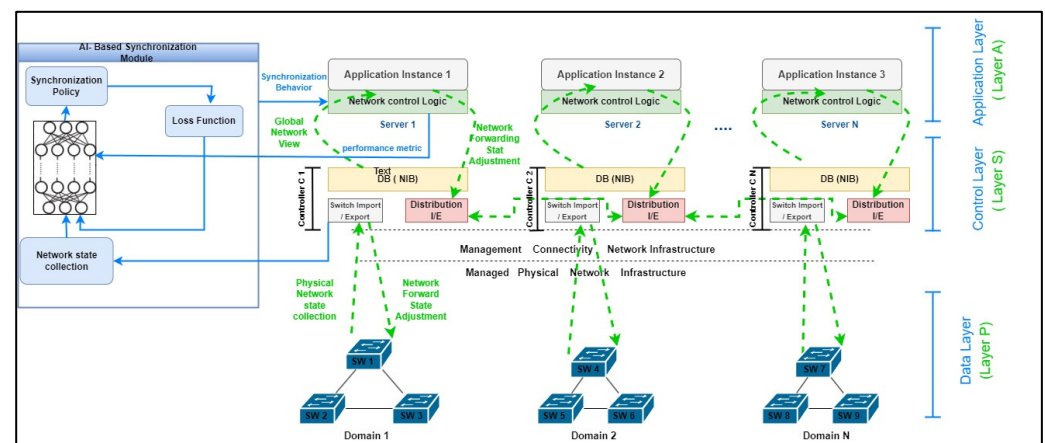


Figure 3. Intelligent Physically Distributed but Logically Centralized (PDLC) inter-controller synchronization approach.

The state-action space for the controller synchronization problem is quite huge. Therefore, the DNN value function can be utilized as an approximator for synchronization policy [44,45].

DRL combines the benefits of RL with DNN, and it can effectively generate a control action for the target system while handling enormous input state spaces. DRL techniques can be utilized to train the adaptive synchronization algorithm. DRL is able to evolve based on interactions with the training environment without the need for significant labeled data collection, in contrast to supervised learning methods like DL [59]. These techniques enable the algorithm to analyze data, make predictions, and optimize synchronization decisions based on real-time network conditions in order to achieve intelligent automated synchronization behavior.

- **Synchronization Monitoring:** Implement a mechanism to monitor the synchronization state between controllers. It can involve tracking performance metrics, latency measurements, and other relevant parameters to ensure effective coordination.
- **Network State Collection:** Incorporate mechanisms to monitor the network state, traffic patterns, and other relevant metrics. These data can be used as feedback for the

AI synchronization algorithm, allowing it to make informed decisions based on the current network conditions.

By integrating these components and modules, you can create an architecture that leverages AI techniques to achieve adaptive and dynamic controller-to-controller synchronization within a PDLC SDN environment.

The solution should be tested for scalability and resiliency to design the architecture to be more scalable and resilient, capable of accommodating a growing number of controllers and maintaining synchronization in the face of network changes or failures. A potential application of interest that can serve as a use case involves applications that can tolerate some level of inconsistency, such as critical path-establishing or load-balancing applications.

5.1. Challenges of AI-Based SDN

While the AI adaptation in SDN provides the network with additional benefits, certain challenges are handled in relation to AI-based SDN, such as the new ML techniques and data standardization [49].

5.1.1. New ML Techniques

As computer learning advances, machine learning becomes a versatile tool. Techniques like learning graphs help detect network topology and adapt to dynamic network changes, especially in deterministic scenarios. ML, using AI techniques like Convolution Neural Networks (CNNs), Q-learning, and other different deep learning techniques, can model and understand graph theory effectively. However, in non-deterministic networks, ML struggles to ensure networking and identify faults.

When the training set is appropriately represented, ML adaptation operates as intended.

Unfortunately, even with AI installation, we still require a deep understanding of the correlation between the ML model's correctness, the network's properties, and the size of the training set. In this case, we are addressing the requirement for ML to evaluate a set of SDN setups, load balancing, and traffic using new tools [49].

5.1.2. Dataset Standardization

ML techniques rely significantly on training datasets. SDN generates different data from various scenarios, which ML algorithms learn and enhance via a straightforward approach, resulting in higher-quality training datasets. This approach proves valuable in routing and VNF experiments. Using AI algorithms to generate high-quality training data in a straightforward approach makes it possible to contribute substantially to larger datasets [49].

5.2. Reinforcement Learning (RL) Challenges

5.2.1. Trade-Off between Exploration and Exploitation

The trade-off between exploration and exploitation is one of the difficulties RL faces, as opposed to other types of learning. The agent in reinforcement learning must favor actions that it has previously performed and proven to be effective in creating reward if it wants to gain a large amount of reward. However, it must also try actions that it has never chosen before. In other words, the agent must take advantage of what it has already learned to profit, but it must also explore to choose its future actions more wisely. The problem is that pursuing either exploration or exploitation exclusively would result in failure. The agent must test several things and gradually favor the ones that seem to work the best. Each action in a stochastic task must be repeatedly tested in order to obtain a reliable estimate of the predicted reward [50].

5.2.2. The Whole Problem as a Goal-Directed Agent

The explicit consideration of the entire problem of a goal-directed agent dealing with an uncertain environment is another crucial aspect of RL. This is contrary to many

techniques, which consider subproblems without considering how they could fit into the bigger picture [50].

5.2.3. Continuous Control Problems

Based on experiences demonstrated by [45] using a variety of RL approaches to solving the specified Markov Decision Process (MDP), several non-trivial issues are highlighted, deriving primarily from the two following characteristics.

First, if the problem of MDP is a “discrete control”, which means that the system’s actions or inputs are limited to a finite set of options, this will pose a barrier to employing several widely recognized and mature actor–critic techniques [60], founded on the policy gradient theorem [61]. For instance, the state of the art is DeepMind’s work on the Deep Deterministic Policy Gradient (DDPG) agent [62], designed to handle “continuous control” problems effectively where actions can vary across a continuous range.

Second, of the challenges faced in employing AI-based SDN, one is whether the state-action space in the network environment of the defined MDP is large. There are up to $2N$ different acts that a state could perform, specifically. Therefore, the action space size grows exponentially when there is more information in the network. A large action space has been demonstrated to be extremely challenging to explore and generalize from. In fact, traditional RL approaches and their modifications, effective in situations with comparatively small discrete action spaces, are inadequate to address this issue [45]. It has been traditional to save experiences in tabular form. However, due to the insufficiency of generalization for a large state-action space, this method is unfeasible for many RL problems.

6. Conclusions

Physically Distributed Logically Centralized (PDLC) SDN controllers require synchronization in their network views. Synchronization overhead adversely affects network operations requiring swift responses and can impede system scalability. Finding the optimal synchronization rate necessitates meticulous tuning and weighing trade-offs between consistency, responsiveness, and overhead.

Embracing adaptive state consistency synchronization in SDN could diminish controller state distribution overhead by eliminating unnecessary messages, thereby preserving application performance.

The flexibility and robustness of the SDN network and its centralized nature, which can provide a global network view, drive the consideration of more automatic adaptable solutions with the integration of SDN and AI concepts.

Recent research on distributed SDN consistency indicates that integrating AI into networks will automate policy learning in any network environment, eliminating the need for network assumptions, and, thus, offering greater adaptability than prior methods.

RL techniques can effectively improve SDN networking attributes. Nevertheless, when coupled with DNNs this generalizes and approximates synchronization policies, aiding in learning controller synchronization strategies based on existing knowledge.

The prevailing trend in current research presumes to attain a logically centralized network view through specific synchronization designs. Yet, there remains a significant gap in the optimal manner in which controllers should synchronize with each other, diminishing performance minimization under eventual consistency requirements. Some AI-based SDN suggests centralized learning to set synchronization policies and regulate controller synchronization. This centralized control is insufficient for distributed control in SDN.

Network-related DL applications are still in their nascent stages. Additionally, since AI network consistency applications are still in their infancy, further research is needed.

Author Contributions: Conceptualization, R.A., E.F. and N.A.; Data Curation, R.A.; Formal Analysis, R.A.; Investigation, R.A.; Methodology, R.A.; Project Administration, E.F.; Resources, R.A.; Software, R.A.; Supervision, E.F. and N.A.; Validation, E.F. and N.A.; Writing—Original Draft, R.A.; Writing—Review and Editing, R.A., E.F. and N.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Yu, H.; Qi, H.; Li, K. WECAN: An Efficient West-East Control Associated Network for Large-Scale SDN Systems. *Mob. Netw. Appl.* **2020**, *25*, 114–124. [\[CrossRef\]](#)
2. Keshari, S.K.; Kansal, V.; Kumar, S. A Systematic Review of Quality of Services (QoS) in Software Defined Networking (SDN). *Wirel. Pers. Commun.* **2021**, *116*, 2593–2614. [\[CrossRef\]](#)
3. Tadros, C.N.; Mokhtar, B.; Rizk, M.R.M. Logically Centralized-Physically Distributed Software Defined Network Controller Architecture. In Proceedings of the 2018 IEEE Global Conference on Internet of Things, GCIoT 2018, Alexandria, Egypt, 5–7 December 2018; IEEE: Piscataway, NJ, USA, 2019. [\[CrossRef\]](#)
4. Ahmad, S.; Mir, A.H. Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers. *J. Netw. Syst. Manag.* **2021**, *29*, 9. [\[CrossRef\]](#)
5. Hoang, N.T.; Nguyen, H.N.; Tran, H.A.; Souihi, S. A Novel Adaptive East–West Interface for a Heterogeneous and Distributed SDN Network. *Electronics* **2022**, *11*, 975. [\[CrossRef\]](#)
6. Espinel Sarmiento, D.; Lebre, A.; Nussbaum, L.; Chari, A. Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 256–281. [\[CrossRef\]](#)
7. Blial, O.; Ben Mamoun, M.; Benaini, R. An Overview on SDN Architectures with Multiple Controllers. *J. Comput. Netw. Commun.* **2016**, *2016*, 9396525. [\[CrossRef\]](#)
8. Venâncio, G.; Turchetti, R.C.; Camargo, E.T.; Duarte, E.P. VNF-Consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *Int. J. Netw. Manag.* **2021**, *31*, e2124. [\[CrossRef\]](#)
9. Informatique, S.; Informatique, G. *Extending SDN Control to Large-Scale Networks: Taxonomy, Challenges and Solutions*; Université Paris-Est Créteil: Créteil, France, 2021.
10. Hussein, A.; Chehab, A.; Kayssi, A.; Elhajj, I. Machine learning for network resilience: The start of a journey. In Proceedings of the 2018 5th International Conference on Software Defined Systems, Barcelona, Spain, 23–26 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 59–66. [\[CrossRef\]](#)
11. Tootoonchian, A.; Ganjali, Y. HyperFlow: A distributed control plane for OpenFlow. In Proceedings of the 2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking, INM/WREN 2010, San Jose, CA, USA, 27 April 2010.
12. Ts, O.N.F. Reference Design SDN Enabled Broadband Access. 2019. Available online: <http://www.opennetworking.org> (accessed on 10 March 2023).
13. Aslan, M.; Matrawy, A. Adaptive consistency for distributed SDN controllers. In Proceedings of the 2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks, Montreal, QC, Canada, 26–28 September 2016; pp. 150–157. [\[CrossRef\]](#)
14. Panda, A.; Scott, C.; Ghodsi, A.; Koponen, T.; Shenker, S. CAP for networks. In Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, Hong Kong, China, 16 August 2013; pp. 91–96. [\[CrossRef\]](#)
15. Oktian, Y.E.; Lee, S.G.; Lee, H.J.; Lam, J.H. Distributed SDN controller system: A survey on design choice. *Comput. Netw.* **2017**, *121*, 100–111. [\[CrossRef\]](#)
16. Levin, D.; Wundsam, A.; Heller, B.; Handigol, N.; Feldmann, A. Logically centralized? State distribution trade-offs in software defined networks. In Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks, HotSDN’12, Helsinki, Finland, 13 August 2012; pp. 1–6. [\[CrossRef\]](#)
17. Foerster, K.T.; Schmid, S.; Vissicchio, S. Survey of Consistent Software-Defined Network Updates. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1435–1461. [\[CrossRef\]](#)
18. Bannour, F.; Souihi, S.; Mellouk, A. Adaptive State Consistency for Distributed ONOS Controllers. In Proceedings of the 2018 IEEE Global Communications Conference, Abu Dhabi, United Arab Emirates, 9–13 December 2018. [\[CrossRef\]](#)
19. Sakic, E.; Sardis, F.; Guck, J.W.; Kellerer, W. Towards adaptive state consistency in distributed SDN control plane. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017. [\[CrossRef\]](#)
20. Akyildiz, I.F.; Lee, A.; Wang, P.; Luo, M.; Chou, W. A roadmap for traffic engineering in software defined networks. *Comput. Netw.* **2014**, *71*, 1–30. [\[CrossRef\]](#)
21. Yao, H.; Jiang, C.; Qian, Y. (Eds.) *Developing Networks Using Artificial Intelligence*, 1st ed.; Springer International Publishing: Cham, Switzerland, 2019; p. 8, ISBN 978-3-030-15028-0. [\[CrossRef\]](#)
22. Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T.; et al. Onix A Distributed Control Platform for Large-Scale Production Networks. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, Vancouver, BC, Canada, 4–6 October 2010; Volume 10, pp. 1–14. Available online: <http://dl.acm.org/citation.cfm?id=1924943.1924968> (accessed on 26 June 2022).

23. Alowa, A.; Fevens, T. Towards minimum inter-controller delay time in software defined networking. *Procedia Comput. Sci.* **2020**, *175*, 395–402. [\[CrossRef\]](#)
24. Open Network Operating System (ONOS). SDN Controller for SDN/NFV Solutions. In Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Los Angeles, CA, USA, 20–21 October 2014; Available online: <https://opennetworking.org/onos/> (accessed on 26 June 2022).
25. Dixi, A.; Hao, F.; Mukherjee, S.; Lakshman, T.V.; Kompella, R.R. ElastiCon: An elastic distributed SDN controller. In Proceedings of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2014, Marina del Rey, CA, USA, 20–21 October 2014; pp. 17–27. [\[CrossRef\]](#)
26. Ferguson, A.D.; Gribble, S.; Hong, C.Y.; Killian, C.; Mohsin, W.; Muehe, H.; Ong, J.; Poutievski, L.; Singh, A.; Vicisano, L.; et al. Orion: Google’s Software-Defined Networking Control Plane. In Proceedings of the 2021 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, Virtual, 12–14 April 2021; pp. 83–98.
27. Bannour, F.; Souihi, S.; Mellouk, A. Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 333–354. [\[CrossRef\]](#)
28. Hu, J.; Lin, C.; Li, X.; Huang, J. Scalability of control planes for software defined networks: Modeling and evaluation. In Proceedings of the IEEE 22nd International Symposium of Quality of Service (IWQoS), Hong Kong, China, 26–27 May 2014; pp. 147–152. [\[CrossRef\]](#)
29. Remigio Da Silva, E.; Endo, P.T.; De Queiroz Albuquerque, E. Standardization for evaluating software-defined networking controllers. In Proceedings of the 2017 8th International Conference on the Network of the Future (NOF), London, UK, 22–24 November 2017; pp. 135–137. [\[CrossRef\]](#)
30. European Commission. Technology Readiness Levels (TRL). Horizon 2020—Work Program. 2014–2015 Gen. Annex. Extr. from Part 19—Comm. Decis. C. 2014. Available online: http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415 (accessed on 26 June 2022).
31. Jain, S.; Smith, T. Googles SDN. *J. Netw. Eng.* **2013**, *5*, 3–14.
32. Hong, C.-Y.; Lee, D.; Kim, E. SDWAN: Achieving High Utilization. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM ’13, Hong Kong, China, 12–16 August 2013; ACM Press: New York, NY, USA, 2013; Volume 43, p. 15.
33. Qiu, T.; Qiao, R.; Wu, D.O. EABS: An event-aware backpressure scheduling scheme for emergency internet of things. *IEEE Trans. Mob. Comput.* **2018**, *17*, 72–84. [\[CrossRef\]](#)
34. Almadani, B.; Beg, A.; Mahmoud, A. DSF: A Distributed SDN Control Plane Framework for the East/West Interface. *IEEE Access* **2021**, *9*, 26735–26754. [\[CrossRef\]](#)
35. Cai, Z.; Cox, A.; Ng, E.T.S. Maestro: A System for Scalable OpenFlow Control. Cs.Rice.Edu.. 2011. Available online: <http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf> (accessed on 10 March 2024).
36. Rajsbaum, S. ACM SIGACT news distributed computing column 13. *ACM SIGACT News* **2003**, *34*, 53–56. [\[CrossRef\]](#)
37. Benamrane, F.; Ben Mamoun, M.; Benaini, R. An East-West interface for distributed SDN control plane: Implementation and evaluation. *Comput. Electr. Eng.* **2017**, *57*, 162–175. [\[CrossRef\]](#)
38. Adedokun, E.A.; Adekale, A. Development of a Modified East-West Interface for Distributed Control Plane Network. *Arid. Zone J. Eng. Technol. Environ.* **2019**, *15*, 242–254. Available online: www.azojete.com.ng (accessed on 12 January 2022).
39. Abdelsalam, M.A. Network Application Design Challenges and Solutions in SDN. Ph.D. Thesis, Carleton University, Ottawa, ON, Canada, 2018.
40. Aslan, M.; Matrawy, A. A Clustering-based Consistency Adaptation Strategy for Distributed SDN Controllers. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization, NetSoft 2018, Montreal, QC, Canada, 25–29 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 257–261. [\[CrossRef\]](#)
41. Aslan, M.; Matrawy, A. On the impact of network state collection on the performance of SDN applications. *IEEE Commun. Lett.* **2016**, *20*, 5–8. [\[CrossRef\]](#)
42. Floodlight Controller—Confluence. Available online: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview> (accessed on 25 June 2022).
43. Stribling, J.; Sovran, Y.; Zhang, I.; Pretzer, X.; Li, J.; Kaashoek, M.F.; Morris, R.T. Flexible, wide-area storage for distributed systems with wheelies. In Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, Boston, MA, USA, 22–24 April 2009; USENIX Association: Berkeley, CA, USA, 2009; pp. 43–58.
44. Zhang, Z.; Ma, L.; Poularakis, K.; Leung, K.P.; Wu, L. DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; IEEE: Piscataway, NJ, USA, 2019. [\[CrossRef\]](#)
45. Zhang, Z.; Ma, L.; Poularakis, K.; Leung, K.K.; Tucker, J.; Swami, A. MACS: Deep reinforcement learning based SDN controller synchronization policy design. In Proceedings of the 2019 IEEE 27th International Conference on Network Protocols, Chicago, IL, USA, 7–10 October 2019; IEEE: Piscataway, NJ, USA, 2019. [\[CrossRef\]](#)
46. Mestres, A.; Rodriguez-Natal, A.; Carner, J.; Barlet-Ros, P.; Alarcón, E.; Solé, M.; Muntés-Mulero, V.; Meyer, D.; Barkai, S.; Hibbett, M.J.; et al. Knowledge-defined networking. *Comput. Commun. Rev.* **2017**, *47*, 2–10. [\[CrossRef\]](#)
47. Clark, D.D.; Partridge, C.; Christopher Ramming, J.; Wroclawski, J.T. A Knowledge Plane for the Internet. *Comput. Commun. Rev.* **2003**, *33*, 3–10. [\[CrossRef\]](#)

48. Aouedi, O.; Piamrat, K.; Parrein, B. Intelligent Traffic Management in Next-Generation Networks. *Futur. Internet* **2022**, *14*, 44. [\[CrossRef\]](#)
49. Mohammad, S.; Shankar, K.; Chanti, Y. AI Based SDN Technology Integration with their Challenges and Opportunities. *Asian J. Comput. Sci. Technol.* **2019**, *8*, 165–169.
50. Andrew, A.M. Reinforcement Learning: An Introduction. *Kybernetes* **1998**, *27*, 1093–1096. [\[CrossRef\]](#)
51. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
52. Li, Y.; Su, X.; Ding, A.Y.; Lindgren, A.; Liu, X.; Prehofer, C.; Riekk, J.; Rahmani, R.; Tarkoma, S.; Hui, P. Enhancing the Internet of Things with Knowledge-Driven Software-Defined Networking Technology: Future Perspectives. *Sensors* **2020**, *20*, 3459. [\[CrossRef\]](#) [\[PubMed\]](#)
53. Huang, H.; Yin, H.; Min, G.; Jiang, H.; Zhang, J.; Wu, Y. Data-Driven Information Plane in Software-Defined Networking. *IEEE Commun. Mag.* **2017**, *55*, 218–224. [\[CrossRef\]](#)
54. Fang, C.; Guo, S.; Wang, Z.; Huang, H.; Yao, H.; Liu, Y. Data-driven intelligent future network: Architecture, use cases, and challenges. *IEEE Commun. Mag.* **2019**, *57*, 34–40. [\[CrossRef\]](#)
55. Chemalamarri, V.D.; Braun, R.; Lipman, J.; Abolhasan, M. A Multi-agent Controller to enable Cognition in Software Defined Networks. In Proceedings of the 28th International Telecommunication Networks and Application Conference, ITNAC 2018, Sydney, Australia, 21–23 November 2018; IEEE: Piscataway, NJ, USA, 2019. [\[CrossRef\]](#)
56. Hussein, A.; Salman, O.; Chehab, A.; Elhajj, I.; Kayssi, A. Machine learning for network resiliency and consistency. In Proceedings of the 2019 6th International Conference on Software Defined Systems, SDS 2019, Rome, Italy, 10–13 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 146–153. [\[CrossRef\]](#)
57. Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; De Frcitas, N. Dueling Network Architectures for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York, NY, USA, 20–22 June 2016; Volume 4, pp. 2939–2947.
58. Tavakoli, A.; Pardo, F.; Kormushev, P. Action branching architectures for deep reinforcement learning. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 4131–4138. [\[CrossRef\]](#)
59. Sun, P.; Guo, Z.; Wang, G.; Lan, J.; Hu, Y. MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Comput. Netw.* **2020**, *177*, 107230. [\[CrossRef\]](#)
60. Konda, V.R.; Tsitsiklis, J.N. *Actor-Critic Algorithms*; Laboratory for Information and Decision Systems, Massachusetts Institute of Technology: Cambridge, MA, USA, 2001.
61. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **2000**, *12*, 1057–1063.
62. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. In Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016.

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 MDPI and/or the editor(s). MDPI 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.