A Novel Adaptive East–West Interface for a Heterogeneous and Distributed SDN Network

Nam-Thang Hoang, Hai-Nam Nguyen, Hai-Anh Tran, Sami Souihi

electronics

MDPI

Article

A Novel Adaptive East–West Interface for a Heterogeneous and Distributed SDN Network

Nam-Thang Hoang [1], Hai-Nam Nguyen [1,2], Hai-Anh Tran [1,*] and Sami Souihi [3]

[1] School of Information and Communication Technology (SOICT), Hanoi University of Science and Technology (HUST), Hanoi 10000, Vietnam; thanghn@huce.edu.vn (N.-T.H.); namhust98@gmail.com (H.-N.N.)
[2] Texpect Laboratory, Texpect Company, 75020 Paris, France
[3] LISSI-TincNET, University of Paris Est Creteil (UPEC), 94400 Vitry-sur-Seine, France; sami.souihi@u-pec.fr
* Correspondence: anhth@soict.hust.edu.vn; Tel.: +84-915825046

Abstract: In the years since its initiation, the software-defined network paradigm has evolved into a distinguished networking technology by causing a revolution in separating the control logic from physical devices and centralizing software-based controllers. Despite its indisputable benefits compared with the traditional network, the SDN raises the challenge of scalability with its physically centralized control. The only potential solution is to transform it into physically distributed SDN control. However, this solution requires the interoperability between SDN controllers, and the consistency of network state being distributed across these controllers. Although some east–west interfaces that help SDN controllers exchange network information have been released, they reveal several drawbacks. First, they cannot support a heterogeneous SDN system where SDN controllers are developed by different providers. Secondly, their consistency solution is simple in disregarding the tradeoff between the consistency level and the performance of SDN networks. This paper proposes an east–west interface, called SINA, to provide the interoperability of a heterogeneous and distributed SDN network. In addition, a novel reinforcement-learning-based consistency algorithm is introduced for an adaptive, quorum-based replication mechanism. The experimental results showed that SINA successfully connects heterogeneous and distributed SDN domains and balances the consistency and network performance.

Keywords: SDN; heterogeneous and distributed SDN control; scalability; interoperability; quorum consistency; adaptive consistency; reinforcement learning; Q-Learning; ONOS controller; Faucet controller; OpenDaylight controller

Citation: 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. https://doi.org/10.3390/electronics11070975

Academic Editor: Riccardo Sisto

Received: 10 February 2022
Accepted: 16 March 2022
Published: 22 March 2022

1. Introduction

The inception of software-defined network (SDN) is considered an auspicious revolution in network technology by dividing the control plane from the data plane of the classical network [1]. It leverages the logically centralized controller provided by a worldwide network view to manage the system. SDN allows technological breakthroughs by relocating the burden of network management to software-based networks. Besides its numerous advantages, such as centralized network management, operating costs savings, complete enterprise administration, hardware savings, coarse security, and lower capital expenditures, the SDN architecture is facing a challenge of scalability and reliability because of its physically centralized controller [2,3]. A single physical SDN controller encounters the bottleneck problem due to many physical devices in the data plane. Additionally, the physically centralized SDN control cannot support different demands of enormous network implementation (e.g., service provider networks or data centers). In other words, the SDN architecture is recommended to be implemented with the physically distributed and logically centralized control [4–9]. When required, a physically distributed SDN control can scale physical controllers up by adding more controllers. In addition, the system reliability is also enhanced because the network is now fault-tolerant with backup controllers whenever controller failures occur. Consequently, the physically distributed SDN control is

indispensable to address the scalability and reliability problem. Such control plane needs the interoperability between distributed controllers and the consistency of network state.

The interoperability of a distributed SDN network is the ability of participant domains controllers to exchange information and coordinate seamless operation. In a general SDN architecture, besides the northbound interface and southbound interface for exchanging information with the application plane and the data plane, respectively, the east–westbound interface is needed for interoperability between distributed controllers. However, contrary to the widespread popularity of the standardization of southbound interface OpenFlow, the east–westbound interface has not received the necessary attention from the research community to offer interoperability between SDN controllers. This is explained by the fact that, today, when the upgrade from the legacy network to SDN is still relatively small, the need for interoperability of SDN networks is not yet necessary. However, this interoperability is the survival of the internet for the foreseeable future as SDN increasingly proves its advantages [10]. It is undeniable that the efforts of some research groups have yielded some results, see [11–13]. However, they have just resolved the interoperability problem for a homogeneous SDN network with the same controller type provided by a vendor. Recently, two east–west interfaces [14,15] were proposed for heterogeneous SDN networks with different types of controllers. Nevertheless, they applied the active replication mechanism in broadcasting every new update information to all other controllers. As a consequence, the broadcasting method degrades network performance.

Consistency for the network state of a distributed SDN network is to ensure that each domain's information about the global system is consistent. The centralized consistency mechanism is deployed in many studies [16,17], with a centralized server to collect all the updated information from the whole system. However, this method has a few disadvantages, including the following: (1) scalability—it is difficult to scale the number of SDN clusters due to the limitation of the centralized server's performance; (2) single point of failure—if that centralized server crashed, it would lead to the collapse of the entire system. In our paper, we choose a completely different approach in using a distributed SDN system without using any centralized server. The distributed SDN controllers apply two replication methods to ensure the consistency of network topology information of the whole system. Our approach can address both mentioned problems. With the physically distributed SDN design, some significant consistency challenges have been raised in recent years [18,19]. As proven in [20] with the CAP theorem, an SDN network cannot satisfy all the three factors: partition tolerance (P), high availability (A), and consistency (C). Concretely, a network consisting of many partitions with high availability (A and P) causes a weak level of consistency. Consequently, this state of staleness affects the correctness of applications. On the contrary, a system with a strong consistency (C and P) results in a lack of network availability. Unfortunately, the majority of research on the network state consistency for the distributed SDN network has only focused on the guarantee of the strong consistency model [21–23]. The latter ensures that all SDN controller surrogates always have the most updated network state. Unfortunately, that leads to significant degradation of network performance. The problems mentioned above motivated us to address the two issues: (1) interoperability of a distributed and heterogeneous SDN network, and (2) network state consistency in considering the tradeoff between the consistency level and the network performance. This paper proposed an east–west interface called SINA, which helps heterogeneous SDN controllers update network states. Moreover, we also introduced in SINA an adaptive, quorum-based replication mechanism for an eventual consistency model that ensures an acceptable consistency level, while keeping some network metrics below some predefined thresholds. The choice of replication method depends on the nature and needs of the system. For example, the active replication method is appropriate if the system needs to ensure strong consistency without worrying about performance waste with the broadcasting mechanism. On the contrary, if strong consistency is not essential and the system needs to save performance for message broadcasting, then it has to choose the quorum-based replication method. The experiments are divided into two phases:

SINA east–west interface validation and *SINA with adaptive consistency model*. The former demonstrates the correctness of the SINA framework in applying the most straightforward consistency policy, the active replication, based on the broadcasting mechanism. The latter is performed to evaluate SINA in applying the quorum-based replication mechanism using the Q-Learning algorithm. The experimental results obtained in both phases showed that SINA could guarantee an acceptable consistency level while maintaining good network performance metrics, such as read delay, write delay, and overhead. The organization of this paper is described as follows. Section 2 presents a brief overview of distributed SDN networks and a short survey on existing east–west interfaces and consistency mechanisms for a distributed SDN network. Section 3 describes in detail the proposed east–west interface SINA. The proposed adaptive consistency algorithm is illustrated in Section 4. The experiments are analyzed in Section 5. Section 6 gives the conclusion and perspectives.

2. Overview of Distributed SDN Network and Related Work

The distributed SDN architecture is illustrated in Figure 1. The SDN network is divided into several domains in the data layer, and a controller manages each one. All controllers constitute a joint heterogeneous controllers' cluster in the control layer. These controllers, provided by different vendors (e.g., ONOS, OpenDaylight, Faucet, etc.), communicate through an east–west interface. Besides the need to help controllers in this SDN cluster easily understand each other, the control layer also requires a consistency mechanism so that all controllers share the same knowledge about the network state. In other words, they must possess the identical local replication of the whole network topology and the device/link state. Any state change in any controller (e.g., the appearance of a new device/link, the failure of a device/link, etc.) must be disseminated to other controllers according to a consistency algorithm. This section provides a brief survey of existing east–west interfaces and their consistency mechanisms.

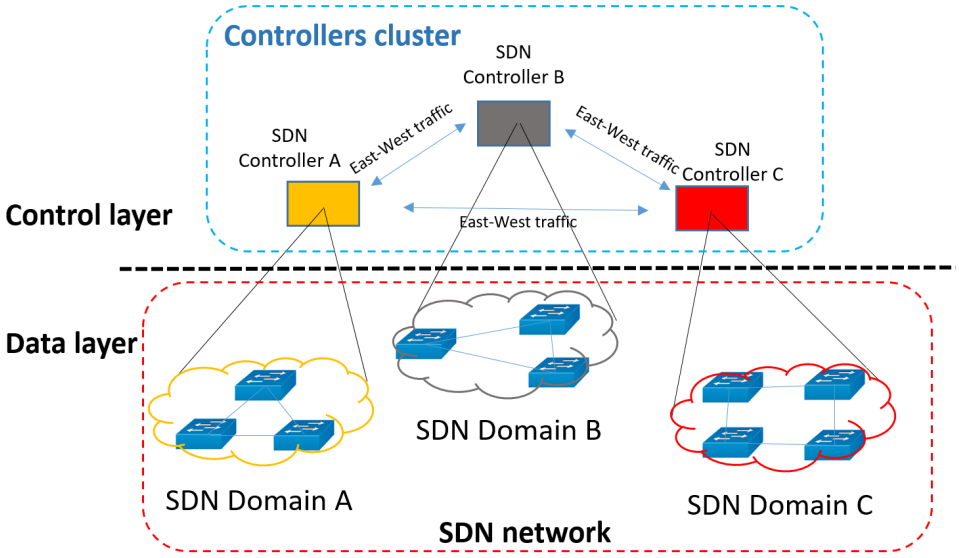

Figure 1. Distributed SDN architecture.

Almadani et al. [14] proposed a distributed SDN control framework, called DSF, also for an east–west interface. They have already tested DSF with two control platforms: ONOS and Floodlight. The problem of DSF is that its current version works only on Java-based control platforms, although the authors explain that it can run on multiple platforms. For other problems, DSF is not a distributed network, and they always need a centralized controller to collect information from other ones or centralized shared data space.

Benamrane et al. [24] proposed an east–west interface, called communication interface for distributed control plane (CIDC), that allows synchronization. CIDC's performance is

evaluated in emulated topologies with two services: firewall and load balancing. It uses three communication modes: notification, service, and full. However, CIDC is implemented only for the communication between Floodlight-based and ODL-based controllers. In [12], Adekokun et al. presented a modified version of CIDC, called mCIDC, that is deployed using Floodlight controller to ensure the communication between controllers of a WAN network. mCIDC consists of four main modules: data updater, data collector, consumer, and producer. The consistency model of mCIDC is based on publish/subscribe model, where every update in each controller is propagated to others. Netty framework is used to decrease the resource consumption and applied two communication modes: notification and full. Both CIDC and mCIDC have the same disadvantage of applicability across different controller types. In addition, these two interfaces applied the strong consistency model to update every change of each controller, leading to performance degradation.

In [15], Yu et al. proposed an east–west interface, called WECAN, consisting of three modules: the controller selection management (CSM) module, the cross-domain management (CDM) module, and the network information collection (NIC) module. The role of the NIC module is to provide a domain-wide network state by gathering network data from a different set of controllers. The CDM module provides a global network state by accumulating domain information from other domains. Finally, the CSM module chooses the most efficient controller for each network flow based on the domain-wide and entire network state. The experimental results showed that WECAN had improved some network parameters (e.g., latency, throughput) compared with a single controller–controlled network. However, WECAN can be integrated into only two controllers: Floodlight and Maestro.

The widely known controller platforms (e.g., NOX, ONOS, ONIX, OpenDaylight, Faucet) have also implemented east–west communication techniques between SDN controllers. Hyperflow [25], an application of the NOX platform [26], is a scattered-event-driven control layer for OpenFlow. Each controller has a HyperFlow controller application instance that publishes the update state events of the system using a publish/subscribe system. Afterwards, other controllers apply all the published events to regenerate the state. In other words, it aims to synchronize the view of all controllers based on an event dissemination system applying publish/subscribe mechanism. However, Hyperflow uses the broker-based distributed file system WheelFS [27] that caused the problem of network performance degradation. The open-source platform ONOS uses a distributed system framework, called Atomix [28], to manage SDN clusters. Atomix, considered an event-driven reactive Java framework, also encounters the problem of scalability, leading to system performance degeneration with the growth of the network scale. In [13], a tool named ICONA (inter-cluster ONOS network application) is proposed to help ONOS to operate in an individual supervisory WAN network plan. It allows an individual ONOS cluster to be coordinated with numerous other clusters of different administrative domains. However, ICONA works only with the ONOS controller, and it applied the active replication method to guarantee strong consistency. In the ONIX platform [29], a data structure—network information base (NIB)—is used to store network state information. Using ONIX APIS, the state information can be replicated to all other controllers for every state update. The performance limitation makes ONIX appropriate only for network states with slow changes. Besides, ONIX has as strict a requirement the necessity of a global view in all SDN controllers. The essential replication in real time of all persistent transactional databases across all controllers makes ONIX not applicable to an unstable large-scale network. The OpenDaylight platform [30] applied Akka framework [31] for network state synchronization between clusters of SDN controllers. The Akka framework has performance limitations because of large-scale clusters' huge flow or control packets. In the Faucet framework, the management of communication between SDN controllers is handled by Prometheus [32]. This latter allows each controller to share information from the local network or monitor other Faucet controllers' network information. InfluxDB is used to propagate network information between different controllers through a shared database. For Kandoo [33], the controllers are distributed in a two-level hierarchy archi-

tecture that consists of a root and multiple local controllers. The disadvantage of Kandoo is that communication between controllers in a tier is not allowed. Additionally, the implementation of second-tier services that demands a global view of the system is limited. ElastiCon [34] creates a scalable controller pool that becomes larger or smaller according to the traffic conditions. A load of tasks is dynamically allocated to controllers.

The main characteristics of mentioned SDN controller platforms are summarized in Table 1.

Table 1. Characteristics of the SDN controller platforms.

	Architecture		Design Type		Scalability Level	Consistency Level			Language
	Physically Centralized	Physically Distributed	Flat	Hierarchical		Strong	Eventual	Weak	
Hyperflow		✓	✓		High		✓		C++
POX	✓				Weak	✓			Python
NOX	✓				Weak	✓			Python
ONOS		✓	✓		Very high	✓		✓	Java
ONIX		✓	✓		Very high	✓		✓	Python C
OpenDaylight		✓	✓		Very high	✓			Java
Faucet		✓	✓		High	✓			Python
Kandoo		✓		✓	Very high		✓		Python, C++, C

3. Proposed East–West Interface for a Heterogeneous and Distributed SDN Network

The proposed SINA (**S**DN **I**nter-Domains **N**etwork **A**pplication) is an east–west interface to enable network state sharing between heterogeneous SDN domains. Concretely, every change/update of the network state of a domain is captured and then disseminated to all other SDN domains through our SINA RESTful APIs. The source code of SINA is open and available on GitLab upon access request [35].

SINA consists of two main components (Figure 2):

- *SINA Listener* component: Every network state update in the data plane is transmitted to the Core Service management module as an event. *SINA Listener* component, similar to a daemon process, aims to listen and capture these events in the core services of each controller. Then, they are disseminated to other SDN controllers.
- *SINA API* component: It is a set of definitions and protocols used to manage the network state database. SINA API allows controllers to make their resources accessible. For example, as shown in Figure 2, after capturing an event from the core service, controller one notifies it to controller two by calling the SINA API component. The latter is responsible for maintaining and updating the local network state database.

The SINA framework is fully customizable. Concretely, the notification policy can be adapted according to the nature of the information, the criticality level, or the network conditions. In addition, to secure the information transmission, data transfer authorization based on OAuth2 protocol (open authorization) (https://oauth.net/2/ accessed on 20 July 2021) can be activated. It can support different tokens to provide multiple authorization levels and dynamically adapt them according to the requirements. In order to prove the openness of SINA, we deploy it in three different widely used SDN controllers: ONOS [36], OpenDaylight [30], and Faucet [37]. Figure 3 illustrates the implementation of SINA in these three controllers as well as different interactions. The component *SINA Listener* is continuously listening to any event generated by the core service component. After an event is captured, *SINA Listener* calls *SINA API* component of other SDN controllers to update the information. The IP addresses of the other controllers are configured in the JSON file *IPList.json*.

In the inception phase, the proposed SINA is temporarily deployed with the active replication mechanism to guarantee strong consistency based on the broadcasting method. Explicitly, after an update occurs in a controller, it will be incontinently broadcasted to

all other controllers. The objective of this first temporary deployment is to validate the correctness of the application's operations. After, in the second phase (Section 4), we develop an adaptive, quorum-based replication algorithm to balance the consistency level and the network system performance.

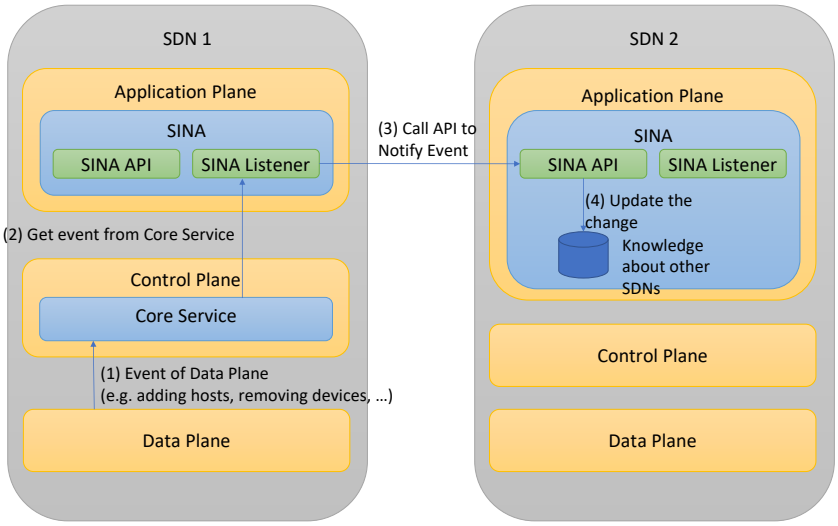

Figure 2. General architecture of SINA.

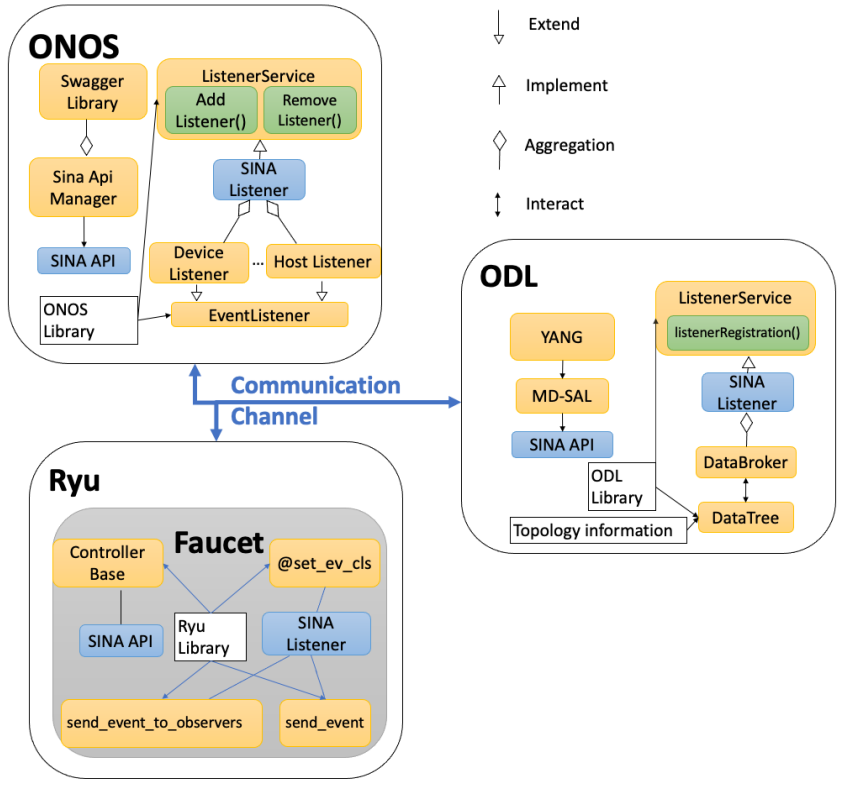

Figure 3. SINA implementations.

The different implementations of the SINA application in three controllers are described as follows (Figure 3).

3.1. Development of SINA's Components in ONOS

An application on ONOS usually consists of two main components *@active* and *@Deactive* that are used to perform actions in the running state and the stopped/deleted state,

respectively, (Figure 3). An ONOS application can execute functions and activate services. Therefore, to build a *Listener component*, we create it as a service activated by the *@active* component. ONOS offers several predefined functions for listening events, and new custom listener complements as an implementation of *ListenerService* class. The latter is an abstraction of the service that offers asynchronous notification capabilities. It contains two declared methods: `addListener()` and `removeListener()`. The SINA Listener is deployed by implementing this interface and these two functions to capture events from the ONOS core service.

For the *SINA API component*, the idea is also to create a service with additional APIs. Using the Swagger framework simplifies the creation of APIs in ONOS. Concretely, an interface called *ApiService* is created. Then, we create the *SinaApiManager* class that implements the ApiService interface by defining all its methods. The *SinaApiManager* class extends the *BaseResource* class so that the Swagger framework can implement the defined APIs.

3.2. Development of SINA on OpenDaylight

As shown in Figure 3, every application of ODL consists of two main functions: `init()` and `close()` (similar to @active and @Deactive in ONOS). There is a difference between ODL and ONOS regarding data management. In fact, in ONOS, services can capture events or get switch information from the core service. On the contrary, it is impossible for ODL services because the data is stored in an object called a *dataTree*. The operations, such as altering, collecting, editing, deleting, and capturing events, are all implemented by an object called a *DataBroker*.

For API development, ODL uses a model-driven service adaptation layer (MD-SAL). The latter utilizes YANG (Yet Another Next Generation) [38] as a modeling language to define the interfaces and the data to be exposed. So, to implement the SINA API, we define the operations in a supported file, named *sina.yang*, and register it in the method `init()`.

3.3. Development of SINA on Faucet

The Faucet, inherited from the Ryu controller [39], is essentially different from the ONOS and ODL. It is impossible to create an application separately from the Faucet because it is already an application on the Ryu controller. Any application developed outside of Faucet can capture only Ryu events/information. Hence, we integrate our source code directly into Faucet's source code (Figure 3).

For the *Listener component*, the functions `send_event()` and `send_event_to_observers()` are used. They are defined by the Ryu controller and can be used to add a new listener to the Faucet's listener list. In addition, *Listener component* used `set_ev_cls` to obtain events from the core service. For example: `set_ev_cls(dpset.EventDP)` contains the event handlers of the DataPath.

Faucet/Ryu implements a library based on WSGI (Web Server Gateway Interface) that facilitates the API building. It is a specification that describes how a web server connects to web applications and how these applications can process requests. According to the *SINA API* component, we defined WSGI context in a Faucet class. In addition, a new class is also created to define the supported operation.

3.4. Openness of SINA for Other SDN Controllers

According to the openness of SINA, besides the three controllers ONOS, OpenDaylight, and Faucet, SINA is ready to be implemented into all other SDN controllers because it uses two types of controllers library: Listening (SINA listener component) and API creating library (SINA API component) packages. Developers can leverage that every SDN controller uses these two types of libraries to implement SINA into that controller. For example, Floodlight [40,41], a widely known Java-based controller, has three library packages for Listener—*SwitchListener*, *DeviceListener*, and *MessageListener*—to receive notifications whenever there is any update of switch, devices (end-host), and message. Concerning the data translation, SINA has a broker module in charge of converting the network state

information in the API component. After receiving data from other SDN controllers, this broker module extracts the necessary data (switch, host, link, port, etc.) and saves the data in a fixed form. This means that this data has the same form in different SDN controllers and platforms. In addition, since the information transmitted is only in JSON or raw text, this broker module can be customized easily.

4. Proposed RL-Based Consistency Algorithm

As mentioned above, the initial phase of SINA follows the active replication, where any update information is propagated to all other controllers. Unfortunately, this broadcast-based solution is not appropriate for large-scale distributed systems because it consumes a lot of network bandwidth and degrades other quality of service factors (delay, loss rate, etc.). Therefore, we are addressing the problem of how to ensure an acceptable consistency level while guaranteeing good QoS metrics. That motivates us to propose an adaptive, quorum-based algorithm to consider the compromise between consistency level and network system performance. The objective is to maximize the consistency metrics represented by staleness values, such as *version staleness* and *time staleness*, while continuously keeping the QoS metrics below some predefined thresholds.

Among consistency protocols, the replicated write protocols are appreciated for their flexibility of sending updates to multiple replicas instead of only one. Replicated write protocols consist of two types: *active replication* and *quorum-based* protocol. The main difference to distinguish these two protocols is that the former uses the broadcasting mechanism by forwarding the update to all replicas, while the latter, on the contrary, forwards the update to only some replicas (the write–quorum). According to the reading operation, for the *active replication* protocol, a replica reads its local data item. Otherwise, for the *quorum-based* protocol, a replica performs the read operation on a set of other replicas (the number of replicas of this set is read–quorum).

The proposed consistency algorithm is described explicitly in the following subsections.

4.1. Quorum-Inspired Consistency Adaptation Strategy for SINA

The consistency strategy can be modeled by the quorum-based replication mechanism. Let N, N_R, and N_W be the total number of controllers, the read–quorum value, and the write–quorum value, respectively. Then, every time a network state update occurs in an SDN domain, the controller forwards it to N_W other controllers. Similarly, every time a controller performs a read operation, it will pull the network state information from N_R other controllers. Thus, as explained above, there are two types of consistency guarantees.

- Strong consistency: The *strict quorums* require that the sum of write–quorum and read–quorum must be greater than the total number of controllers: $N_W + N_R > N$. This consistency model guarantees that the read operation always obtains the latest data item value. However, it is costly in terms of network performance due to the high synchronization overheads.
- Eventual consistency: The *partial quorums* allow the sum of write–quorum and read–quorum not need to overlap the total number of replicas: $N_W + N_R \leq N$.

Partial quorum-based systems guarantee only the eventually consistent network information, resulting in inconsistent information returned to the applications. This issue is not tolerable for some applications. Therefore, we apply the PBS model in [42] to analyze the returned network data, evaluate the consistency level, and control the balance between consistency level and network parameters representing QoS.

Based on these notions, this paper proposes an adaptive, quorum-based consistency model for SINA using partial quorums. The expected benefits are good network performance, scalability, and an acceptable consistency level. The key to the problem lies in choosing the pair of values N_R and N_W to optimize network performance and consistency levels. The network performance is based on the network metrics obtained when running an application in the application plane, such as the read/write delay and the synchroniza-

tion overhead. The continuous consistency model monitors the consistency metrics (i.e., version and time staleness metrics) to meet the consistency requirement consistently.

4.2. Proposed Reinforcement Learning-Based Consistency Model for SINA

We propose an adaptive, quorum-based algorithm to consider the compromise between consistency level and network system performance. The objective is to maximize the consistency metrics represented by staleness values such as version staleness and time staleness while continuously keeping the QoS metrics below predefined thresholds. That motivates us to apply reinforcement learning (RL) to optimize two parameters N_R and N_W, so that both the network performance and the consistency level are guaranteed. RL approach uses the Q-Learning algorithm [43] (Figure 4) that is an algorithm for specific problems by applying the RL approach [44] with its basic concept of Markov decision process (MDP). Based on the experience and rewards, an RL agent can learn control strategies. The MDP model is illustrated by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with $\mathcal{S}$ and $\mathcal{A}$ are set of states and actions, respectively. The transition model $\mathcal{P}(s'|s,a)$ is the probability of switching to state s' after performing action a at state s. The obtained reward is $\mathcal{R}(s,a,s')$ after executing a at state s and switching to s'. At state s, the selection action a is qualified by the Q-value $Q(s,a)$.

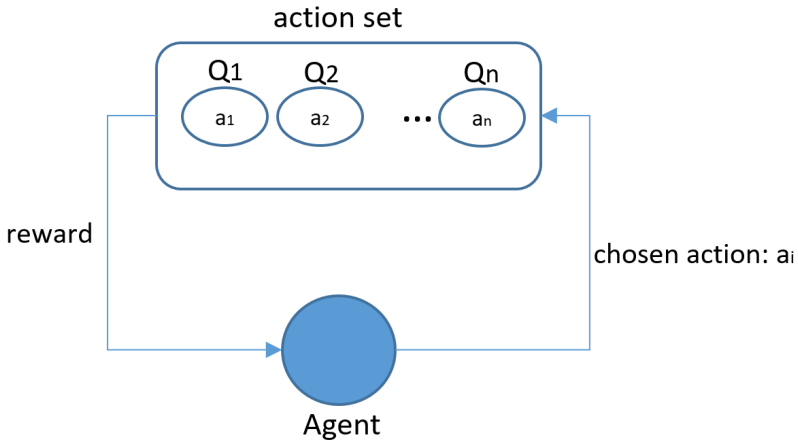

Figure 4. Q-Learning algorithm.

In order to solve an MDP, an optimal policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is needed. The π policy determines the action corresponding to the current state to maximize the cumulative rewards. The Q-fixed point (Bellman equation) needs to be calculated:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q^*(s',a') \quad (1)$$

where $r(s,a)$ represents the expected instantaneous reward of the chosen action a. γ represents the discount factor.

Based on RL, the Q-Learning algorithm [43] is proposed to estimate the optimum action-value function in Equation (1). The agent uses the returned reward r to update the Q-values with the following update equation:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')] \quad (2)$$

where α is the learning rate.

The characteristic of Q-Learning in directly approximating the optimum action-value function with the current policy free motivates us to model the selection problem of the pair of values N_R and N_W as the Q-Learning context.

Our selection of the Q-Learning's action set in Algorithm 1 is inspired by the TCP Vegas, the congestion avoidance mechanism of TCP [45]. That is explained by the fact that the essence of TCP Vegas is to dynamically control the window size according to

the network's congestion level. For our problem, the values of N_R and N_W are adjusted automatically according to the consistency level. If the consistency level is in a downtrend, the values of N_R and N_W will be increased. Otherwise, if the consistency level is an uptrend or acceptable, the values of N_R and N_W will be decreased.

Algorithm 1: Reinforcement learning consistency algorithm for SINA.

Input: R, ϵ, α, $QValues[]$
Output: Arrays $N_R[]$ and $N_W[]$
Initialize the array $QValues[]$ of 6 update actions of N_R and N_W by Equations (6) and (7)
for $t \leftarrow 1$ ***to*** R **do**
  Choose an update action with ϵ probability for the highest Q-Value and $(1-\epsilon)$ probability for a random action
  Append the chosen N_R and N_W to two the arrays $N_R[]$ and $N_W[]$
  Calculate the reward value by Equation (3)
  Update $QValues[]$ by Equation (2)
end
return Arrays $N_R[]$ and $N_W[]$

The reward is calculated by Equation (3).

$$Reward_t = \begin{cases} ThV_{st} - V_{st} & if\, WD < ThW \& RD < ThR \\ -ThV_{st} & if\, WD > ThW \text{or} RD > ThR \end{cases} \tag{3}$$

where WD and RD are the read delay and write delay metrics, respectively. The former is the period from when the request is sent to N_R controllers to when these controllers' results are returned. The latter is the period to disseminate topology update messages to N_W controllers. The ThW and ThR are the delay thresholds for the write and read operations. They are considered tolerable thresholds of the latency of write and read operations. These two thresholds are analyzed in detail through experiments in Section 5.2.3. V_{st} and ThV_{st} are the metrics of version staleness and its threshold, respectively. The version staleness is determined by the version difference between the real version of the local database and the one at the moment where the controller performs a read operation [42,46].

We define the variable DIFF for the adjustment of N_R and N_W. DIFF is calculated by Equation (4).

$$\text{DIFF} = N * \left(1 - \frac{\text{CL}}{\text{CLbase}}\right) \tag{4}$$

where CLbase is the largest value of the CL values observed in the last 10 times. The value CL representing the consistency level is calculated by Equation (5).

$$\text{CL} = \frac{\pi}{\Pi} \tag{5}$$

where π is the number of times that a replica has read the latest value in the last 10 s. Π is the total number of requests in the last 10 s.

The N_R and N_W are updated with Equations (6) and (7). The two thresholds α and β are pre-fixed through experiments (Section 5).

$$N_R(t+\delta t) = \begin{cases} N_R(t) - 1 & if\, \text{DIFF} < \alpha \\ N_R(t) & if\, \alpha < \text{DIFF} < \beta \\ N_R(t) + 1 & if\, \text{DIFF} > \beta \end{cases} \tag{6}$$

$$N_W(t+\delta t) = \begin{cases} N_W(t) - 1 & if\, \text{DIFF} < \alpha \\ N_W(t) & if\, \alpha < \text{DIFF} < \beta \\ N_W(t) + 1 & if\, \text{DIFF} > \beta \end{cases} \tag{7}$$

For the selection mechanism of the Q-Learning algorithm, we applied the ϵ-greedy policy [47]. The latter chooses with a probability of $(1 - \epsilon)$ the value of N_W or N_R with the highest Q-value and a probability of ϵ a random value of N_W or N_R in the current state.

The proposed algorithm is expected to guarantee acceptable values of version staleness and time staleness while keeping the metrics WD and RD below the two thresholds ThW and ThD, respectively.

4.3. Proposed Adaptive Architecture

The proposed architecture is illustrated in Figure 5. The *SINA API* component contains the module "*Network state updating*" that is used to continuously listen and receive the propagation information from other controllers (flow 1). Then, it updates the local network state database representing the global view on the whole network of the current controller (flow 2). To assess the performance of the proposed consistency model, we deployed an application for file transfer (the module *running application*) with an adaptive routing mechanism that finds the lowest cost path from the sender to the receiver machine. This application extracts the network topology information from the local database to perform the routing algorithm (flow 3). The SINA Listener component consists of two modules: *network monitoring* and *quorum-based replication*. The former always listens for the update events from the core service of the control plane (flow 6), updates the local database (flow 5), then sends up to the module *quorum-based replication* for the propagation. The latter is the core module of the SINA architecture. Since reads or writes operations are a decisive factor influencing the application's consistency level and the network performance, the selection operation for the two values of N_W and N_R plays a crucial role in the consistency model. Therefore, the *quorum-based replication* module is modeled as a Q-Learning model inspired by the TCP Vegas algorithm as described in Section 4.2. After choosing the two values N_W and N_R, the module *quorum-based replication* propagates the update information to N_W controllers, or sends the network state request to N_R controllers (flow 8).

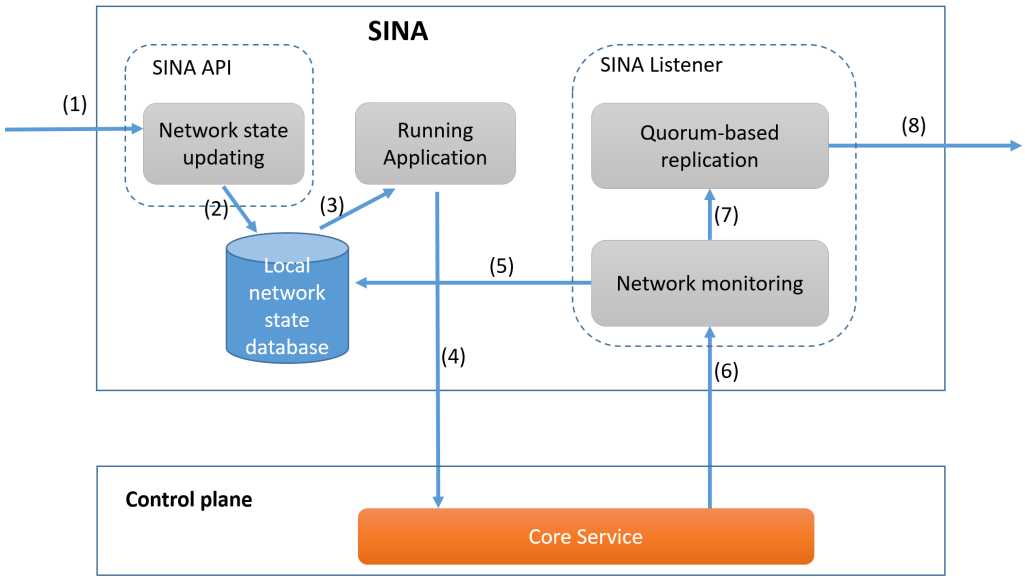

Figure 5. SINA architecture with the adaptive, quorum-based replication module. The numbers (1)–(8) represent flow ID.

5. Performance Evaluation

In this work, the performance evaluation is divided into two phases:

- *SINA east–west interface validation*: In the first step, we aim to examine the correctness of the SINA framework in applying the most straightforward consistency policy, the active replication, based on the broadcasting mechanism. Every update in each

controller is sent to all other controllers. We also compare SINA's performance with other east–west interfaces of three widely used frameworks: ONOS, OpenDaylight, and Faucet.

- *SINA with adaptive consistency model*: In the second phase, the performance of the SINA architecture in Figure 5 is evaluated in applying the quorum-based replication mechanism using the Q-Learning algorithm.

5.1. SINA East–West Interface Validation

5.1.1. Testbed Setup

The used network topologies in our testbed are taken from *TopologyZoo* (http://www.topology-zoo.org/, accessed on 18 June 2021), an in-progress project aiming to compile network topology data from around the world. There are over two hundred and fifty networks in the "Zoo" in different graph formats for statistical analysis, plotting, or other network research topics.

In order to study the performance for different network scales, four network topologies are selected from the *TopologyZoo.*

- AARNet is initially established between the University of Melbourne in Australia, the Commonwealth Scientific and Industrial Research Organization (CSIRO) in all Australian state capitals and the Australian National University in Canberra. AARNet consists of 13 network switches and hosts.
- Abilene network generated by the Internet2 community with 11 network switches and hosts.
- AMRES network is the national research and education network of Serbia. It has 21 network switches and hosts.
- NORDU network, located in Nordic countries, consists of 5 network switches and their hosts.

These four network topologies are emulated by the tool *mininet* [48]. The network topologies between SDN controllers are fully-connected. All virtual machines hosting 442 controllers are connected through the bridged networking method. We also tested the latency between two machines: the obtained RTT result is about 0.03 ms. Consequently, it does not affect the experimental results. Each switch is connected to only one controller. The experimentation scenario aims to evaluate the communication between multiple SDN controllers. The network is built with many SDN controllers (from 6 to 12) in each scenario. According to the links' characteristics, the throughput, latency, and loss rate are set to 10 Mb/s, 10 ms, and 2%, respectively. We use only HELLO messages for network traffic.

The experimental setup is physically deployed on the server PowerEdge R740 with 32 CPUs Intel(R) silver 4216 2.10 GHz and 256 GB of memory. We create 13 Linux virtual machines on this physical server to deploy emulated network topology and SDN controllers as follows:

- One virtual machine is used to emulate the network topology in using *mininet*.
- 12 remaining virtual machines are used to implement SDN controllers separately. Concretely, each one is launched in a virtual machine. Thus, we spent 12 virtual machines for the largest experiment where 12 controllers were deployed.

The three evaluation parameters are described as follows.

- *Data transmission (bits)*: For each change of network topology, the SDN controller broadcasts the update information to others.
- *Latency (second)*: The period between the sending of the notifications and the receipt of the last acknowledgment from the destination controllers.
- *Overhead (%)*: The ratio between the controller packets to the total packets.

Each experiment is repeated 100 times. The confidence interval is concretely shown in our obtained results, keeping the confidence level at 95%.

5.1.2. Experimental Results

In order to prove the scalability and the interoperability of SINA, the communication between multiple controllers is evaluated in varying the number of controllers in a testbed from six to twelve in four different topologies. As shown in Table 2, the experimentation scenario is divided into three sub-scenarios of 6, 9, and 12 controllers. Each sub-scenario is divided into four schemes (4 rows for each sub-scenario in Table 2): in the first one, three types of controllers (Faucet, ONOS, and ODL) are implemented in using SINA. For the three remaining schemes, all controllers are implemented with one type. For example, in the sub-scenario of 6 controllers, the first scheme is to implement six different controllers: 2 Faucet, 2 ONOS, and 2 ODL. The three remaining schemes are implemented with 6 Faucet, 6 ONOS, and 6 ODL. As mentioned above, the confidence interval (the values of the upper limit, lower limit, and mean) is shown in Table 2 after launching each experiment 100 times with the confidence level of 95%. The launching scenario is replicated strictly for each experiment's run. The idea is to analyze the impact of the SINA application. The obtained results show a common feature: the larger the topology, the higher the data transmission value. Similarly, the larger the topology, the higher the latency value obtained because lots of data is exchanged between the controllers, causing the time-consuming. The average confidence intervals of Data transmission and Latency are acceptable with the values of 2412.2 (bit) and 0.36 (s), respectively. We also note that in most scenarios, the use of SINA application dramatically improves performance in terms of both latency and data transmission. However, in some cases, Faucet's solution's scheme attains better results (e.g., scheme of 9 Faucet controllers with the *Abilene* topology). To sum up, SINA has reduced 3.8% and 2.4% of data transmission and latency, respectively.

Table 2. Communication between multiple controllers. The blue and bold numbers are the best values in networks with the same number of controllers. Similarly, the red and bold numbers are the worst values. Explaination for abbreviations: x is an integer number. Pr: Parameter; DT: Data Transmission (bit); Ltc: Latency (second); Low. Lim.: Lower limit of the confidence interval Up. Lim.: Upper limit of the confidence interval; x diff.: x/3 ONOS, x/3 ODL, and x/3 Faucet (using SINA). x ONOS Controllers; x ODL Controllers; x Faucet Controllers: not using SINA.

Network	Pr	Nordu 1989			Abilene			Aarnet			Amres		
		Low. Lim.	Up. Lim.	Mean	Low. Lim.	Up. Lim.	Mean	Low. Lim.	Up. Lim.	Mean	Low. Lim.	Up. Lim.	Mean
6 diff.	DT	2981.3	4251.3	**3616.3**	7372.1	9145.5	**8258.8**	9186.6	12,936	**11,061.3**	17,887.4	19,061	18,474.2
	Ltc	0.011	0.051	**0.031**	0.03	0.078	**0.054**	0.044	0.066	0.055	0.081	0.111	0.096
6 ONOS	DT	3075.2	4764.6	3919.9	9181	9945.8	9563.4	10,710.6	13,448	12,079.3	13,107	17,680	**15,393.5**
	Ltc	0.029	0.047	0.038	0.033	0.087	**0.06**	0.073	0.085	0.079	0.063	0.167	0.115
6 ODL	DT	2301.2	6014.2	**4157.7**	9633.8	10,072.6	9853.2	12,503.6	14,572	**13,537.8**	18,321.6	22,623	**20,472.3**
	Ltc	0.043	0.081	**0.062**	0.028	0.078	0.053	0.087	0.105	**0.096**	0.103	0.145	**0.124**
6 Faucet	DT	3532	4113.8	3822.9	9716.6	10,258.2	**9987.4**	10,509.2	13,991	12,250.1	16,108.6	22,271	19,189.8
	Ltc	0.013	0.051	0.032	0.022	0.084	0.053	0.025	0.071	**0.048**	0.081	0.095	**0.088**
9 diff.	DT	3214.5	5004.3	**4109.4**	11,995.5	13,825.3	**12,910.4**	10,487.6	15,964	**13,225.8**	22,114	24,676	**23,395**
	Ltc	0.034	0.042	**0.038**	0.034	0.088	0.061	0.056	0.084	**0.07**	0.12	0.136	**0.128**
9 ONOS	DT	5204.8	8107.6	**6656.2**	11,610.2	12,056.8	11,833.5	12,636.8	13,253	12,944.9	17,453	19,472	**18,462.5**
	Ltc	0.019	0.063	0.041	0.052	0.076	**0.064**	0.067	0.137	0.102	0.124	0.162	**0.143**
9 ODL	DT	3793.6	5384.8	4589.2	10,489.8	12,534.8	11,512.3	13,621	15,726	**14,673.5**	22,144.4	23,362	22,753.2
	Ltc	0.049	0.077	**0.063**	0.049	0.077	0.063	0.107	0.143	**0.125**	0.093	0.131	0.112
9 Faucet	DT	3930.2	6152.4	5041.3	9680.4	10,245.6	**9963**	11,229.6	15,404	13,316.8	18,595.6	21,983	20,289.3
	Ltc	0.023	0.055	0.039	0.0445	0.0535	**0.049**	0.049	0.067	0.058	0.101	0.159	0.13
12 diff.	DT	2441.9	6014.3	**4228.1**	9916.9	14,352.3	**12,134.6**	12,226	13,234	12,730	18,922	20,701	**19,811.5**
	Ltc	0.082	0.092	0.087	0.077	0.091	**0.084**	0.125	0.141	0.133	0.105	0.177	**0.141**
12 ONOS	DT	3558.8	6825.4	5192.1	12,694	13,894.2	13,294.1	11,902.6	13,280	12,591.3	16,710.6	19,784	**18,247.3**
	Ltc	0.07	0.11	**0.09**	0.05	0.132	0.091	0.117	0.135	0.126	0.175	0.193	**0.184**
12 ODL	DT	2725	6279.4	4502.2	13,039.5	15,437.1	**14,238.3**	10,348.4	12,148	**11,248.2**	16,884.2	21,364	19,124.1
	Ltc	0.057	0.085	0.071	0.059	0.143	**0.101**	0.157	0.225	**0.191**	0.1508	0.1712	0.161
12 Faucet	DT	6158.7	7425.1	**6791.9**	8765.8	12,440	10,602.9	13,756.6	14,084	**13,920.3**	17,272.4	21,876	19,574.2
	Ltc	0.017	0.047	**0.032**	0.077	0.093	0.085	0.065	0.097	**0.081**	0.14	0.158	0.149

Figure 6 illustrates the latency and the overhead results in using the "Amres" topology, the largest one. The confidence intervals are acceptable for each experiment, keeping the confidence level of 95%. The results showed that the legacy Faucet solution obtains the best results in terms of overhead and latency because the communication solution using the Prometheus tool is optimized for the data exchanges between Faucet controllers. Concretely, Faucet has reduced the overhead and latency of other controllers by 62.4% and 10.1%, respectively. Regarding the two controllers, ONOS and ODL, the communication protocols implemented in both cases generate more overhead than the SINA application. Concretely, SINA improves the overhead and latency of ONOS and ODL by 58.1% and 14.5%, respectively. Concerning the latency, the SINA application outperforms existing solutions in most cases, although there are no constraints related to the heterogeneity of the controllers for other approaches. Thus, the obtained preliminary results prove the scalability and interoperability of SINA.

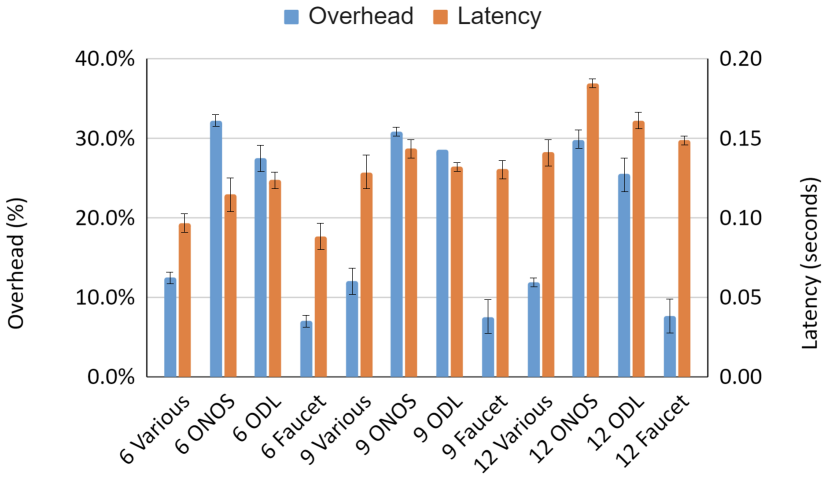

Figure 6. Latency and Overhead Analyses.

5.2. SINA with Adaptive Consistency Model

For this second phase of performance evaluation, the SINA is implemented in applying the adaptive, quorum-based replication mechanism with the Q-Learning algorithm described in Section 4.

5.2.1. Testbed Setup

A large-scale topology, called *Cogent*, has been chosen among the topologies from *Topology Zoo* to assess the efficiency of the proposed adaptive consistency model. The Cogent topology, located in USA and Europe, consists of 176 switches and 21 hosts. We use the mininet tool to emulate the Cogent topology and divide it into 18 separated SDN domains, each of which is managed by a dedicated controller (ONOS, OpenDaylight, or Faucet). Among these 18 domains, each of 17 domains has ten switches; six remaining switches are located in the last one. Each domain of 15 domains has one host; the remaining domains have two hosts. Each switch is connected to only one controller. Thus, the network topology between SDN controllers is fully connected. Concerning the links' characteristics, the throughput, latency, and loss are set to 10 Mb/s, 10 ms, and 2%, respectively. The HELLO messages are used for network traffic.

The open-source *Tcpreplay* tool [49] is used to generate the flows with any data bit-rate. This tool is usually used to edit and replay a previously captured network traffic with a fixed sending bit rate. It helps us generate a few initial data flows and replay them many times after.

In order to emulate the dynamic network environment, the constant changes of the network topology are based on the classical 2-State Markov Model of Gilbert [50]. All SDN switches and links follow the 2-State Markov Model with *ON* and *OFF* states. The former represents the state where the switch/link works normally. Otherwise, the latter represents the state where the switch/link is broken. The state transition is shown in Figure 7.

The values of transition probability p and r are determined by Equations (8) and (9).

$$p = P(q_t = \text{OFF} | q_{t-1} = \text{ON}) \tag{8}$$

$$r = P(q_t = \text{ON} | q_{t-1} = \text{OFF}) \tag{9}$$

where q_t is the state at time t.

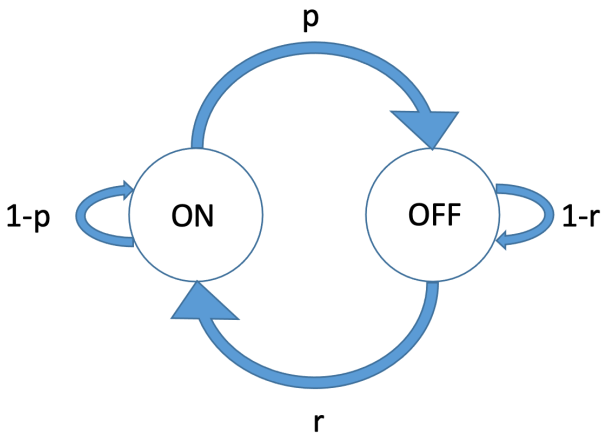

Figure 7. State transition of switches and links based on 2-State Markov Model of Gilbert.

Based on the experiments, both values p and r are fixed to 50%.

The experimental setup is also physically deployed on the mentioned server PowerEdge R740. On this physical server, 19 Linux virtual machines are created to host one VM for emulating network topology (with *mininet*) and 18 VM for separated SDN controllers that manage 18 SDN domains.

5.2.2. Performance Metrics

For the performance metrics, two types of metrics are used as follows:

- QoS-related metrics: Concerning the QoS-related metrics, we are based on the overhead and the delays. Concretely, they are *WriteOverhead*, *ReadOverhead*, *WriteDelay*, and *ReadDelay*. The *WriteOverhead* and *ReadOverhead* are the numbers of the inter-controller packets through TCP port 8787 for the read and the write commands, respectively. The *WriteDelay* is the latency the system takes to disseminate the message of topology update to N_W controllers. The *ReadDelay* is the sum of the latency for sending topology requests to N_R controllers and the latency for receiving all topology data from these N_R requested controllers.
- Consistency-related metrics: Two staleness metrics are used: the time-based staleness (*t_staleness*) and the version-based staleness (*v_staleness*) [42,46]. The former is calculated by the period between the last time the local topology database is updated and the moment the controller performs a read operation on its local topology database. The latter is determined by the version difference between the real version of the local database and the one at the moment where the controller performs a read operation.

Similar to the first phase of *SINA east–west interface validation*, each experiment is repeated 100 times. The running scenario for each run is exactly replicated for different tests. The confidence interval is shown in the following obtained results in keeping the confidence level of 95%.

5.2.3. Experimental Results

For the first step, the two parameters N_W and N_R are fixed to the values of 18 and 1, respectively. In other words, we reevaluate our proposal with the active replication method in satisfying strong consistency condition: $N_W + N_R > N$ (the number of controllers in our experiment, N, is 18). The obtained results for *WriteDelay* and *ReadDelay* are illustrated in Figure 8. The result showed that the curve of *ReadDelay* is always stable with a small value of 2 (ms) because the N_R is fixed to the value of 1. In other words, this short and stable period is explained by the fact that each controller requests the topology data from only one controller. Meanwhile, the *WriteDelay* value is gradually decreasing from 95 down to 70 (ms). Its stability is also reflected in decreasing the confidence interval from 7 down to 1.9 (ms).

From the following experiments, the Q-Learning is performed with the parameters $\gamma = 0.8$ and $\alpha = 0.6$. In the beginning, the initial values of N_R and N_W are 4 and 14, respectively. The value of the write threshold is fixed to 50 (ms). Three scenarios are launched with three different values of the Read-threshold: 10 ms, 30 ms, and 50 ms. The value ϵ of the selection mechanism ϵ-greedy is fixed to 10%. We used the threshold in order to control the bounds of the write delay, read delay, and version staleness. The following results will demonstrate the capacity of the bound control.

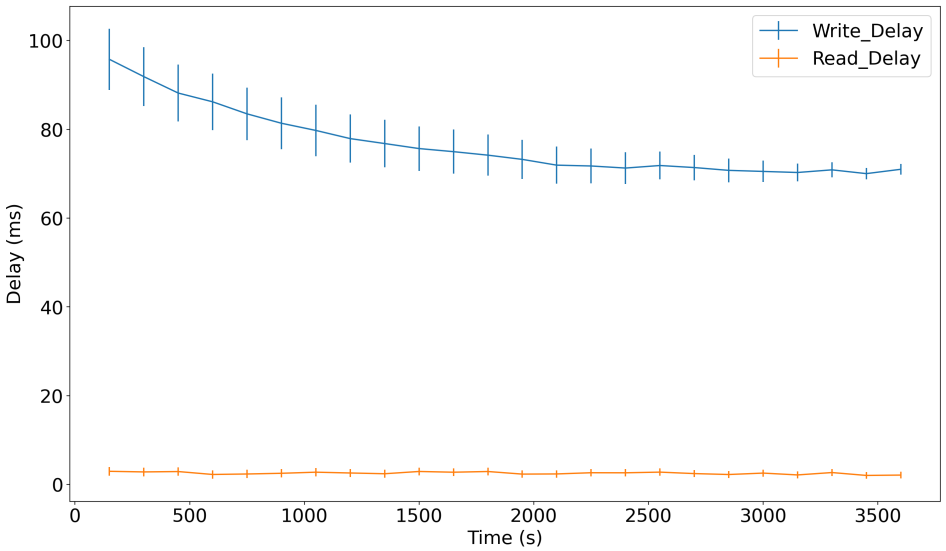

Figure 8. ReadDelay and WriteDelay with fixed values of $N_W = 18$ and $N_R = 1$.

As shown in Figure 9, although the adaptive selection of the value N_R increases the read delay compared to the active replication method above, these delay values do not exceed the fixed thresholds 10 ms, 20 ms, and 30 ms. After about 1200 s of initialization, the delay variation related to the three thresholds 10 ms, 30 ms, and 50 ms become stable under the correspondent threshold values. Thereby, this result showed that the proposed algorithm had established a stable delay for the read operation. Similarly, the adaptive selection of the value N_W of the proposed algorithm decreases the write delay. As a result, the write delay in three cases is always less than the fixed write threshold value of 50 ms. Concretely, the convergence values of the mean value of write delay for three read threshold's cases—10 ms, 30 ms, and 50 ms—are 45.2 s, 40 s, and 38.3 s, respectively. This obtained result also showed that the more the read threshold is increased, the more the write delay value is reduced with a fixed write threshold.

The obtained results in Figure 9 showed that the proposed algorithm guarantees a bound control for the two metrics read and write delay. That is explained by the value of the reward in Equation (3), where we assign a terrible value if the two read and write values exceed the two thresholds.

The version staleness and time staleness are illustrated in Figure 10. After the 1500 s, the value of version staleness becomes stable. Concretely, the three cases of read thresholds 10 ms, 30 ms, and 50 ms have the convergent mean values of 0.4, 0.11, and 0.3, respectively. Significantly, the version staleness metric is always less than the fixed version staleness threshold (0.5). Similar to the previous case, the version staleness value is also bounded. The curve corresponding to the read threshold of 30 ms attains the smallest range value, showing the importance of selecting the read threshold value. An appropriate selection of read-threshold gives a good consistency level in terms of version staleness. Concerning the obtained results of time staleness, after the first 1500 ms with a high fluctuation, the value of time staleness returns to a stable state. In three cases of read thresholds 10 ms, 30 ms, and 50 ms, the convergent mean values of time staleness are 2641, 2412, and 2502, respectively.

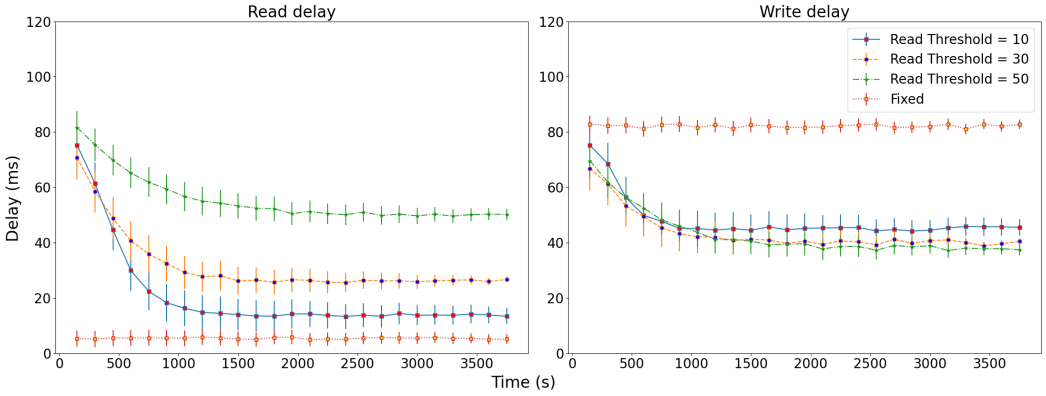

Figure 9. ReadDelay and WriteDelay with different values of ThR.

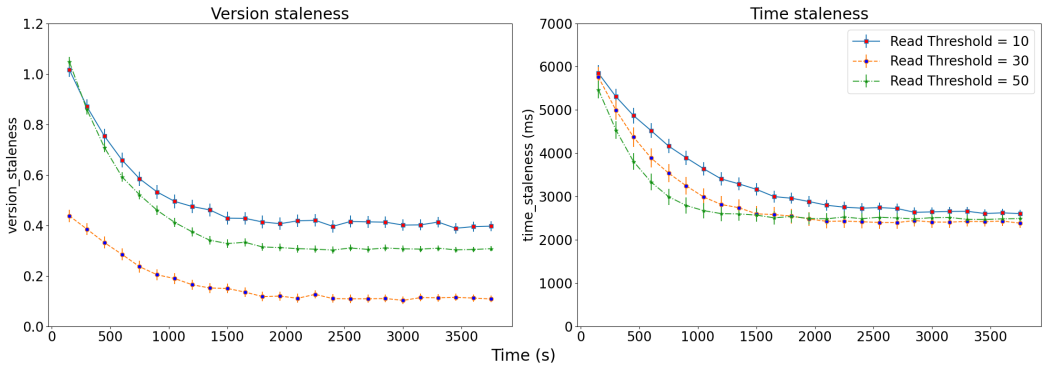

Figure 10. Version staleness and time staleness with different values of ThR.

According to the overhead result, as shown in Figure 11, the fixed case always obtains the highest overhead value of 4950 (Bytes/s) because its sum of two values N_W and N_R is greater than the other ones. After about 1500 s, the three curves corresponding to read thresholds of 10 ms, 30 ms, and 50 ms become gradually stable with the convergent mean values of 2670, 2415, and 2500, respectively.

The obtained experimental results showed that the proposed solution could maintain an acceptable consistency level (metrics of version staleness and time staleness) while continuously controlling the bound of different network metrics such as read delay, write delay, and overhead.

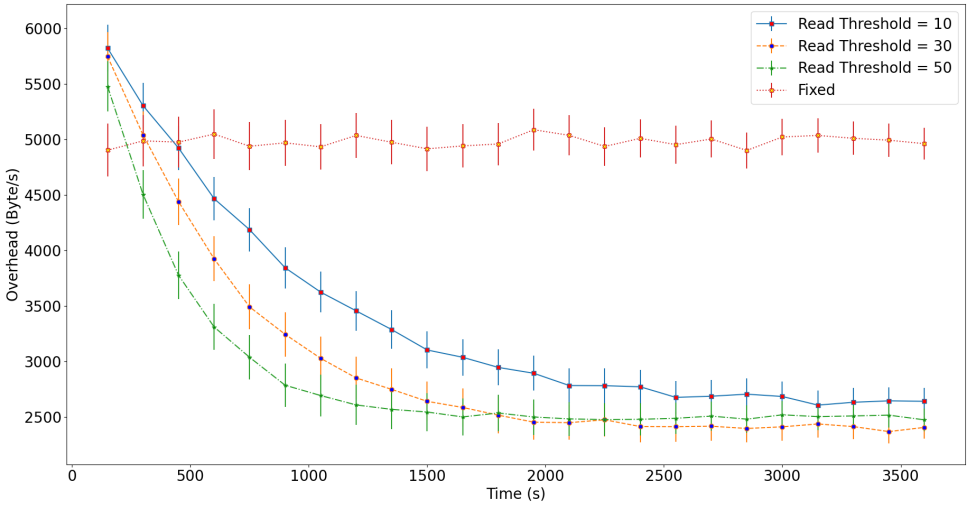

Figure 11. Overhead with different values of *ThR*.

6. Conclusions

This paper proposes an east–west interface called SINA to provide interoperability and scalability for heterogeneous and distributed SDN networks. In the first step, since we aim to validate its operations in a large-scale SDN network, SINA is implemented with active replication to maintain the highest consistency level. The experiments showed that SINA attains outstanding results compared to other well-known related interfaces regarding reading/writing latency and overhead. However, despite its high consistency level, the active replication has revealed its weaknesses in the excessive use of system resources (bandwidth, processing capacity, etc.). That motivated us to propose an adaptive and continuous consistency model in applying the quorum-based replication method in the application layer of SINA. This Q-Learning-based algorithm, inspired by the Vegas strategy—a congestion avoidance solution for TCP—transformed SINA into a more scalable and innovative replication policy. As a result, the experiments show that SINA can connect heterogeneous and distributed SDN domains. In addition, with the proposed adaptive, quorum-based replication, SINA maintains an acceptable consistency level while providing impressive network metrics such as read delay, write delay, and overhead. For future work, we will perform other replication methods for comparison to have more experiments with other consistency mechanisms.

Author Contributions: Conceptualization, H.-A.T. and S.S.; Data curation, H.-N.N.; Formal analysis, N.-T.H.; Methodology, H.-A.T. and S.S.; Project administration, H.-A.T.; Software, H.-N.N.; Supervision, H.-A.T. and S.S.; Validation, N.-T.H. and H.-N.N.; Visualization, H.-N.N.; Writing—original draft, N.-T.H. and H.-A.T.; Writing—review & editing, H.-A.T. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

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

References

1. Prajapati, A.; Sakadasariya, A.; Patel, J. Software defined network: Future of networking. In Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January2018; pp. 1351–1354.
2. Mokhtar, H.; Di, X.; Zhou, Y.; Hassan, A.; Ma, Z.; Musa, S. Multiple-level threshold load balancing in distributed SDN controllers. *Comput. Netw.* **2021**, *198*, 108369. [CrossRef]

3. Midha, S.; Tripathi, K. Extended Security in Heterogeneous Distributed SDN Architecture. In *Advances in Communication and Computational Technology*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 991–1002.
4. Vizarreta, P.; Trivedi, K.; Mendiratta, V.; Kellerer, W.; Mas–Machuca, C. DASON: Dependability Assessment Framework for Imperfect Distributed SDN Implementations. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 652–667. [CrossRef]
5. Li, M.; Wang, X.; Tong, H.; Liu, T.; Tian, Y. SPARC: Towards a scalable distributed control plane architecture for protocol-oblivious SDN networks. In Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–9.
6. Beiruti, M.A.; Ganjali, Y. Load migration in distributed sdn controllers. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9.
7. Li, Z.; Hu, Y.; Hu, T.; Wei, P. Dynamic SDN controller association mechanism based on flow characteristics. *IEEE Access* **2019**, *7*, 92661–92671. [CrossRef]
8. Sood, K.; Karmakar, K.K.; Yu, S.; Varadharajan, V.; Pokhrel, S.R.; Xiang, Y. Alleviating Heterogeneity in SDN-IoT Networks to Maintain QoS and Enhance Security. *IEEE Internet Things J.* **2019**, *7*, 5964–5975. [CrossRef]
9. Moeyersons, J.; Maenhaut, P.J.; Turck, F.; Volckaert, B. Pluggable SDN framework for managing heterogeneous SDN networks. *Int. J. Netw. Manag.* **2020**, *30*, e2087. [CrossRef]
10. Prasad, J.R.; Bendale, S.P.; Prasad, R.S. Semantic Internet of Things (IoT) Interoperability Using Software Defined Network (SDN) and Network Function Virtualization (NFV). In *Semantic IoT: Theory and Applications. Studies in Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2021; Volume 941, pp. 399–415.
11. Brockelsby, W.; Dutta, R. Traffic Analysis in Support of Hybrid SDN Campus Architectures for Enhanced Cybersecurity. In Proceedings of the 2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 1–4 March 2021; pp. 41–48.
12. Adedokun, E.; Adekale, A. Development of a Modified East–West Interface for Distributed Control Plane Network. *Arid. Zone J. Eng. Technol. Environ.* **2019**, *15*, 242–252.
13. Gerola, M.; Lucrezia, F.; Santuari, M.; Salvadori, E.; Ventre, P.L.; Salsano, S.; Campanella, M. Icona: A peer-to-peer approach for software defined wide area networks using onos. In Proceedings of the 2016 Fifth European Workshop on Software-Defined Networks (EWSDN), Den Haag, The Netherlands, 10–11 October 2016; pp. 37–42.
14. 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]
15. 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]
16. Jin, X.; Gossels, J.; Rexford, J.; Walker, D. CoVisor: A Compositional Hypervisor for Software-Defined Networks. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, CA, USA, 4–6 May 2015; pp. 87–101.
17. Yang, G.; Yu, B.y.; Jin, H.; Yoo, C. Libera for programmable network virtualization. *IEEE Commun. Mag.* **2020**, *58*, 38–44. [CrossRef]
18. 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]
19. Gao, K.; Nojima, T.; Yu, H.; Yang, Y.R. Trident: Toward Distributed Reactive SDN Programming With Consistent Updates. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1322–1334. [CrossRef]
20. Panda, A.; Scott, C.; Ghodsi, A.; Koponen, T.; Shenker, S. Cap for networks. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 91–96.
21. Botelho, F.; Ribeiro, T.A.; Ferreira, P.; Ramos, F.M.; Bessani, A. Design and implementation of a consistent data store for a distributed SDN control plane. In Proceedings of the 2016 12th European Dependable Computing Conference (EDCC), Gothenburg, Sweden, 5–9 September 2016; pp. 169–180.
22. 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.
23. Zhang, T.; Giaccone, P.; Bianco, A.; De Domenico, S. The role of the inter-controller consensus in the placement of distributed SDN controllers. *Comput. Commun.* **2017**, *113*, 1–13. [CrossRef]
24. Benamrane, F.; Benaini, R. An East–West interface for distributed SDN control plane: Implementation and evaluation. *Comput. Electr. Eng.* **2017**, *57*, 162–175. [CrossRef]
25. Tootoonchian, A.; Ganjali, Y. Hyperflow: A distributed control plane for openflow. In Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, San Jose, CA, USA, 27 April 2010; Volume 3.
26. Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 105–110. [CrossRef]
27. Stribling, J.; Sovran, Y.; Zhang, I.; Pretzer, X.; Li, J.; Kaashoek, M.F.; Morris, R.T. Flexible, Wide-Area Storage for Distributed Systems with WheelFS. In Proceedings of the NSDI, Boston, MA, USA, 22–24 April 2009; Volume 9, pp. 43–58.
28. ONF. Atomix: A Reactive Java Framework for Building Fault-Tolerant Distributed Systems. Available online: https://atomix.io (accessed on 15 February 2021).

29. 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 OSDI, Vancouver, BC, Canada, 4–6 October 2010; Volume 10, pp. 1–6.
30. Medved, J.; Varga, R.; Tkacik, A.; Gray, K. OpenDaylight: Towards a model-driven sdn controller architecture. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, Australia, 19 June 2014; pp. 1–6.
31. Akka: Build Powerful Reactive, Concurrent, and Distributed Applications More Easily. Available online: https://www.lightbend.com (accessed on 6 April 2021).
32. Prometheus. Prometheus—Monitoring System & Time Series Database. Available online: https://prometheus.io/ (accessed on 17 May 2021).
33. Hassas Yeganeh, S.; Ganjali, Y. Kandoo: A framework for efficient and scalable offloading of control applications. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012; pp. 19–24.
34. Dixit, A.; Hao, F.; Mukherjee, S.; Lakshman, T.; Kompella, R.R. ElastiCon; an elastic distributed SDN controller. In Proceedings of the 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Marina del Rey, CA, USA, 20–21 October 2014; pp. 17–27.
35. Nguyen, H.N.; Tran, H.A.; Souihi, S. SINA: An SDN Inter-Clusters' Network Application. Available online: https://gitlab.com/souihi/sina (accessed on 3 May 2021).
36. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: Towards an open, distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 1–6.
37. Bailey, J.; Stuart, S. Faucet: Deploying SDN in the enterprise. *Commun. ACM* **2016**, *60*, 45–49. [CrossRef]
38. Bjorklund, M. YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF); IETF RFC 2010. Available online: https://www.rfc-editor.org/info/rfc6020 (accessed on 9 February 2022).
39. Asadollahi, S.; Goswami, B.; Sameer, M. Ryu controller's scalability experiment on software defined networks. In Proceedings of the 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), Bangalore, India, 1–2 February 2018; pp. 1–5.
40. Mohammed, S.H.; Jasim, A.D. Evaluation of firewall and load balance in fat-tree topology based on floodlight controller. *Indones. J. Electr. Eng. Comput. Sci.* **2020**, *17*, 1157–1164. [CrossRef]
41. Khan, M.A.; Goswami, B.; Asadollahi, S. Data visualization of software-defined networks during load balancing experiment using floodlight controller. In *Data Visualization*; Springer: Singapore, 2020; pp. 161–179.
42. Bailis, P.; Venkataraman, S.; Franklin, M.J.; Hellerstein, J.M.; Stoica, I. Probabilistically bounded staleness for practical partial quorums. *arXiv* **2012**, arXiv:1204.6082.
43. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
44. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
45. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of the Conference on Communications Architectures, Protocols and Applications, London, UK, 31 August–2 September 1994; pp. 24–35.
46. Bailis, P.; Venkataraman, S.; Franklin, M.J.; Hellerstein, J.M.; Stoica, I. Quantifying eventual consistency with PBS. *VLDB J.* **2014**, *23*, 279–302. [CrossRef]
47. Gomes, E.R.; Kowalczyk, R. Dynamic analysis of multiagent q-learning with epsilon-greedy exploration. In Proceedings of the 26th International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009.
48. Jawaharan, R.; Mohan, P.M.; Das, T.; Gurusamy, M. Empirical evaluation of sdn controllers using mininet/wireshark and comparison with cbench. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–2.
49. Klassen, F. TcpReplay. Available online: https://tcpreplay.appneta.com/ (accessed on 19 July 2021).
50. Haßlinger, G.; Hohlfeld, O. The Gilbert-Elliott model for packet loss in real time services on the Internet. In Proceedings of the 14th GI/ITG Conference-Measurement, Modelling and Evalutation of Computer and Communication Systems, Dortmund, Germany, 31 March–2 April 2008; pp. 1–15.