Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2016, Article ID 9396525, 8 pages
http://dx.doi.org/10.1155/2016/9396525

*Review Article*

# An Overview on SDN Architectures with Multiple Controllers

**Othmane Blial, Mouad Ben Mamoun, and Redouane Benaini**

*LRI, Faculty of Sciences, Mohammed V University in Rabat, Avenue des Nations Unies, Agdal, 10000 Rabat, Morocco*

Correspondence should be addressed to Othmane Blial; blial.othmane@gmail.com

Received 9 December 2015; Revised 15 March 2016; Accepted 14 April 2016

Academic Editor: Eduardo da Silva

Software-defined networking offers several benefits for networking by separating the control plane from the data plane. However, networks' scalability, reliability, and availability remain as a big issue. Accordingly, multicontroller architectures are important for SDN-enabled networks. This paper gives a comprehensive overview of SDN multicontroller architectures. It presents SDN and its main instantiation OpenFlow. Then, it explains in detail the differences between multiple types of multicontroller architectures, like the distribution method and the communication system. Furthermore, it provides already implemented and under research examples of multicontroller architectures by describing their design, their communication process, and their performance results.

## 1. Introduction

Unlike traditional networks, software-defined networking (SDN) [1] separates the control from the data plane in network devices, like switches and routers. This new concept suggests the use of a centralized controller that determines the behavior of all forwarding components in the network.

Southbound interfaces permit communication between the control plane and the data plane, while northbound interfaces provide enormous possibilities for networking programmability, like creating applications that can automate all networking tasks. Consequently, SDN will enhance creativity, as well as innovation, in the domain of networking.

Three critical requirements are not achievable in an SDN-enabled centralized network, which was the main tendency for early proposed SDN architectures, using just one controller: first, efficiency that is not enough established with just one centralized controller, second, scalability that is one of the most issues that pushes network architects to consider the idea of multicontrollers, and, third, high availability, which has two items, redundancy and security. Redundancy is one of the most significant aspects of any design. One controller could fail anytime and, for this reason, abandon the network without its control plane. Security is considered an important item. If an attacker compromises the controller, subsequently it loses the entire management over the network. Clearly, if we have multiple controllers, we can certainly minimize the issue, because they will team up to identify that another one is misbehaving and for that reason separate the attacker from the network.

All these arguments push network designers to think seriously about integrating multicontroller architectures in their designs, and several works have been proposed in this context. Therefore, we were motivated to write a comprehensive overview that explains in detail different aspects and characteristics related to distributed architectures in a software-defined network and clarifies some notions that can be confused, as the difference between logically and physically distributed architectures.

Many papers have done surveys and overviews about SDN; for example, the authors of [1] explained in detail almost all notions and concepts related to SDN. They mention multicontrollers when they talk about the scalability issue. They also provide a table that distinguishes the difference between centralized and distributed controllers; however, they do not give more information about the distribution method. Another survey [2] discusses SDN by explaining its features and clarifying in detail its layers. It mentions multicontrollers briefly when it talks about methods to enhance the control layer performance. There is another interesting survey [3] about SDN that presents chronologically its development. Their authors mention multicontrollers, by providing in a paragraph the difference between centralized and distributed controllers. To the best of our knowledge, we have found only

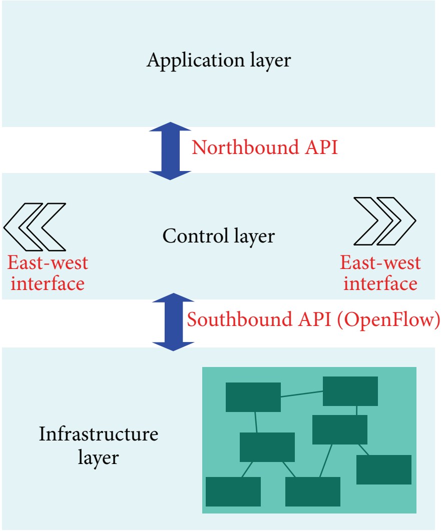


Figure 1: SDN architecture.

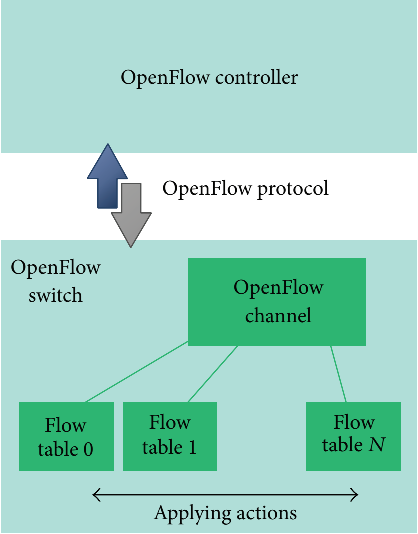


Figure 2: OpenFlow architecture.

one paper [4] that presents a specific survey about the control plane. Nevertheless, it does not give enough information about the implementations of multicontrollers it provides.

The remainder of this paper is organized as follows: we provide a review of the architecture of SDN and its main implementation OpenFlow in Section 2. Further, we present a thorough explanation about the characteristics and related subjects of the distribution of multicontroller designs in Section 3. In Sections 4 and 5, we present some of the proposed multicontrollers by explaining their architectures, classifying them by distribution method, while mentioning the performance results of each model. Finally, in Section 6, we give a conclusion.

## 2. SDN Architecture

In this section, we will present a review of the architecture of SDN and OpenFlow, its main implementation, as shown in Figures 1 and 2, respectively.

*2.1. SDN Architecture.* An SDN architecture contains six major components.

First is the management plane, which is a set of network applications that manage the control logic of a software-defined network. Rather than using a command line interface, SDN-enabled networks use programmability to give flexibility and easiness to the task of implementing new applications and services, such as routing, load balancing, policy enforcement, or a custom application from a service provider. It also allows orchestration and automation of the network via existing APIs [1].

Second is the control plane that is the most intelligent and important layer of an SDN architecture. It contains one or various controllers that forward the different types of rules and policies to the infrastructure layer through the southbound interface [1].

Third, the data plane, also known as the infrastructure layer, represents the forwarding devices on the network (routers, switches, load balancers, etc.). It uses the southbound APIs to interact with the control plane by receiving the forwarding rules and policies to apply them to the corresponding devices [1].

Fourth, the northbound interfaces that permit communication between the control layer and the management layer are mainly a set of open source application programming interfaces (APIs) [1].

Fifth, the east-west interfaces, which are not yet standardized, allow communication between the multiple controllers. They use a system of notification and messaging or a distributed routing protocol like BGP and OSPF.

Sixth, the southbound interfaces allow interaction between the control plane and the data plane, which can be defined summarily as protocols that permit the controller to push policies to the forwarding plane. The OpenFlow protocol is the most widely accepted and implemented southbound API for SDN-enabled networks.

OpenFlow is normalized by the Open Networking Foundation (ONF) [5], backed by the leaders of IT industry like Facebook, Cisco, Google, HP, and others. For this reason, understanding the OpenFlow architecture is important to grasp the notion of SDN, which we are going to present in the next subsection. Before that, we should realize that OpenFlow is just an instantiation of SDN, as there are many existing and under development southbound APIs, for instance, OpFlex [6], which distributes some of the complexity of managing the network to the infrastructure layer to improve the scalability. On the other hand, ForCES [7] proposes a flexible method to ameliorate the management of traditional networks without using a logically centralized controller, while ROFL [8], which relies on OpenFlow, provides an API for software developers to enable full development of new applications [1].

*2.2. OpenFlow Architecture.* In an OpenFlow-enabled network, flow can be represented as a transmission control protocol (TCP) connection. Flows can also be packets with a matching MAC address, an IP address, a VLAN tag, or a switch port [6].

The OpenFlow switch has one or more flow tables. A flow table is a set of flow entries. A flow entry is used to match and process packets. It consists of many matching fields to match packets, a set of encounters to track packets, and instructions to apply [9]. The OpenFlow switch uses an OpenFlow channel to communicate with the OpenFlow controller [9].

The OpenFlow channel is a secure channel between the OpenFlow switch and the OpenFlow controller. It permits communication by allowing the control plane to send instructions, receive requests, or exchange information. All messages are encrypted, using transport layer security (TLS) [9]. The OpenFlow channel has three types of messages. The controller/switch message is initiated by the controller and may not require a response from the switch. The asynchronous message informs the controller about a packet arrival, a switch state change, or an error. The symmetric message can be sent in both directions for other purposes [9].

The OpenFlow controller handles flow tables inside the switch by adding and removing flow entries. It uses the OpenFlow channel to send and receive information [9]. It can be considered as an operating system that serves the whole network.

The OpenFlow protocol is the southbound interface that permits communication between the OpenFlow controller and the OpenFlow switch via the OpenFlow channel [9].

## 3. Architectures of a Multicontroller Software-Defined Network

A multicontroller architecture is a set of controllers working together to achieve some level of performance and scalability. In a software-defined network, multicontroller architectures can have different aspects and characteristics that we are going to discover in the next paragraphs, the differences between logically or physically centralized and distributed architectures, and flat and hierarchical designs. Then, we will try to describe some aspects, like elasticity, controller placement, and communication intercontrollers.

*3.1. Physically Centralized versus Physically Distributed.* In a software-defined network, there are two types of architectures, physically centralized and physically distributed. When SDN appeared, the main tendency was to have a physically centralized controller; however, due to some issues, like the single point of failure and the scalability problem, network experts suggest physically distributed designs.

It is important to understand that we are talking about a multicontroller architecture, just in the case of the physically distributed network. Thus, in the next sections and paragraphs, we will always assume that the controllers are always physically distributed.

Controllers like Beacon [10] and NOX [11] have used multithreading techniques to split a single controller logically to increase its performance. In this case, it is so obvious, since we have a single controller, that we are not talking about a multicontroller architecture.

A physically distributed network can differ on various levels, like how to place controllers and also which type of communication to use among them.

In the next paragraphs, we will explain the different related subjects to a physically distributed architecture.

*3.2. Logically Distributed versus Logically Centralized.* A physically distributed architecture can be either logically centralized or logically distributed.

Logically centralized means that we take advantage of the concept of a multicontroller design, but at the same time, we always consider that we have a single controller. In other words, we take the charge, and we distribute it among the multiple controllers; however, for the underlying layer, it is like there is just one controller that commands the whole network. Another idea was proposed [3] before implementing multiple controllers, which is installing replicated controllers to remove the single point of failure. Nevertheless, this method has many disadvantages such as using passive controllers that will be active, just in case the main controller fails. In a logically centralized architecture, all the controllers have the same responsibilities, and they split the charge equally. They are always aware of every change in the network, and they share the same information instantly, thanks to the network synchronization.

In a logically distributed architecture, the controllers are physically and logically distributed. Additionally, every controller has just a view of the domain it is responsible for, and it can take decisions for it, unlike a logically centralized design, where each controller makes a decision based on the global network view.

In a word, a logically centralized architecture stays near to the initial tendency of SDN, which is using a single controller, or a multicore controller to improve the performance.

On the other hand, a logically distributed architecture goes away from the first tendency of SDN, by making several controllers have several responsibilities inside the network.

*3.3. Flat Architecture versus Hierarchical Architecture.* In the majority of papers relevant to our context, we have found that multicontroller architecture can follow a flat or a hierarchical design.

In a flat or horizontal architecture, the controllers are positioned horizontally on one single level. In other words, the control plane consists of just one layer, and each controller has the same responsibilities at the same time and has a partial view of its network.

In a hierarchical or vertical architecture, the controllers are positioned vertically. They are portioned among multiple levels, which means that the control plane has several layers, generally two or three. The controllers have different responsibilities, and they can take decisions based on a partial view of the network.

These two methods have many advantages and disadvantages; for example, both of these approaches can improve the switch/controller latency in comparison to a single controller architecture or a multicore architecture. In flat design, the network provides more resilience to failures. However, the task of managing controllers becomes harder. On the other hand, a hierarchical design gives a simpler way to manage the controllers, but the problem of a single point of failure remains, because of the upper layer of the control plane. To explain more this final idea, in a hierarchical architecture, we usually have about three layers. Each layer contains a type of controllers. Typically, the bottom layer contains the local controllers, while the upper layer contains one root controller, which means that we have the problem of a single point of failure, even if it concerns just one layer of the control plane.

*3.4. Dynamic Architecture versus Static Architecture.* A logically centralized architecture can be dynamic or static.

In a dynamic or an elastic architecture, the links and the positions between the controllers, as well as the switches, are changeable, which makes the network flexible.

In a static architecture, the links and the positions between the controllers and also the switches are unchangeable, which gives more stability and less overhead to the network in comparison to a dynamic architecture.

*3.5. Intercontrollers Communication.* Communication intercontrollers are the method used to allow exchanging information among the multiple controllers of a software-defined network.

For instance, the "publish and subscribe" messaging paradigm works as follows: in this system, a set of switches subscribe to a particular controller and each controller does the same. After that, controllers publish information between each other to build a global network view.

Another example is a system of notifications exchange. Each controller will send information to its neighbors about its local state to create a global network view.

Previous examples can be more suitable for centralized architectures, while distributed architectures are more likely to implement well-known distributed routing protocols, like BGP, OSPF, and IS-IS.

Building a global network view is always connected to the notion of consistency. This last mentioned one can be either weak, which implies that updates between controllers take a period to be fully applied, or strong that signifies that the multiple controllers read the updates at the same time, which affects the performance of the network positively [1].

*3.6. Placement Problem in Multicontroller Software-Defined Architecture.* The number of used controllers and their positions in distributed network architectures will certainly impact the overall performance of the control plane, which is a significant challenge for network designers. A research paper [12] discusses this problem deeply, called the placement problem of controllers. It tries to solve the placement problem of controllers in WAN networks by improving the delay between a controller and a switch, as well as between two controllers, in order to minimize the response time and enhance the ability of the network to interact more quickly.

The authors also demonstrated that a way to determine how many controllers to use and where to place them is to control three factors: first, the desired reaction limits, notably the latency requirements, second, the metric choices, such as availability, fairness of state of the network, processing, and bandwidth, and, third, the network topology.

They also found that surprisingly a single controller is enough to meet response time requirements in a medium-size network.

## 4. Logically Centralized Architectures

In this section, we will present examples of logically centralized multicontroller architectures.

*4.1. ONIX.* ONIX [13] is a distributed control plane that contains a cluster of one or more physical servers; each one may run multiple ONIX instances.

To understand how ONIX works, we should grasp its role in the network and the utility of its API.

A network controlled by ONIX has four components: first, the physical infrastructure, which includes all the network switches and routers, and other network devices, such as load balancers and firewalls. ONIX interacts with the physical infrastructure by reading and writing the state controlling of each element, for example, the forwarding table entries, second, the connectivity infrastructure, which is the communication between the physical network and ONIX, third, the control logic that relies on the top of ONIX's API. It controls the desired network behavior, and, fourth, ONIX, which is responsible for giving the control logic programmatic access to the network.

The ONIX API is a useful API developed for network control. It allows control applications to read and write the state of any element in the network. It is a data model that represents the entire network infrastructure, with each network element corresponding to one or more data objects. The control logic, already defined, can read the current state of each object. Each copy of the network state of an object that is related to a network element is stored in the Network Information Base, NIB. This NIB is a graph that contains all the network entities. Also, network applications are implemented by reading and writing to the NIB, while ONIX distributes the NIB data between multiple running instances.

NIB has a collection of network entities. Each one has key-value pairs. Based on these pairs, each entity has a set of attributes. The NIB provides multiple methods for the control logic to access the network entities. It has the complete control over the state of an entity, because it maintains an index for each one of them.

The NIB uses a system of notifications. For example, when it receives the notification "Query," it means that the NIB needs to find one or multiple network entities, while when it receives "Create," it means that it must create a new entity.

ONIX provides three methods to improve the scalability of its network. First is by partitioning the network logically, in other words, by distributing the workload on multiple ONIX instances. Second, ONIX can allow multiple nodes to show up as a single node in the upper layer, which is called aggregation. Third, ONIX allows data state applications that can be used to improve the consistency and the durability of the network.

ONIX provides a scenario where we can experience the scalability of the network: a network with a modest number of switches that can be easily managed by a single ONIX instance. The authors found that the control logic can record all forwarding information from the switches. Also, it can coordinate all the data and share them on the multiple instances.

The main results of the evaluation study of ONIX have found that, thanks to the partitioning process, ONIX can partition the workload over multiple ONIX instances. So, in case there is an overhead inside an ONIX instance, already assigned switches can be reassigned to another ONIX instance.

*4.2. HyperFlow.* HyperFlow [14] is an application developed on the top of the NOX controller, to enable logically centralized multicontroller architectures.

The HyperFlow-based network contains three parts: a control layer, a forwarding layer, and an application layer. The control layer contains multiple NOX controllers that are working cooperatively. In the forwarding layer, the switches are connected to the nearest controller. However, a switch can be reassigned to another controller in case of failure.

To propagate information in the control plane, HyperFlow uses a "publish/subscribe" messaging paradigm. This system aims to provide a guaranteed event delivery. It is also responsible for keeping the ordering of events published by the same controller. Also, it minimizes the traffic required for intercontrollers to have less overhead.

This "publish/subscribe" system runs on the top of WheelFS [15], a distributed file system that delivers flexible wide area storage for distributed application. It permits the applications to have more control over the control plane.

In a HyperFlow-based network, we find three channels to permit interaction between the different components: the data channel, the control channel, and the controller channel.

The controllers publish and subscribe to all of them. OpenFlow commands are published only on the controller channel, which is also used to prevent failures inside the network. The data and the control channel are mainly used to execute the publish/subscribe system to permit communication intercontrollers.

A large number of network events request only some types of services, like routing. The global network view is not affected by the changing order of arriving events or those that target the same switch. In some cases, when the network is not able to identify the events that might change the network's state, HyperFlow can implement state synchronization among applications running on the top of the controllers to resolve the problem.

The authors have found that HyperFlow-based controllers can operate more smoothly under heavy load synchronization and keep minimal latency in comparison to NOX controllers.

They also found that HyperFlow can keep an acceptable amount of consistency among controllers for some 1000 arriving events per second, for instance, 1000 events that include switch and host connecting and disconnecting the network and a link state change. Nevertheless, we noticed a disadvantage, which is the added delay when a controller converges or reaches network synchronization, which increases the response time.

*4.3. ONOS.* ONOS [16] provides two prototypes of a software-defined multicontroller model, which differ in many aspects.

The first prototype has three characteristics: the global network view, the scalability, and the fault tolerance. This prototype keeps a global network view by gathering switch, port, and link information.

The network view has three components: Titan [17] (a graph database), Cassandra [18] (a key-value store), and Blueprints [19] (a graph API to expose network state to the application layer).

ONOS can add supplementary instances to distribute the workload on the control plane when it is scaling out.

ONOS can reassign a task to another instance to prevent failures.

The results of the evaluation study of ONOS prototype 1 showed that ONOS can control hundreds of switches and hosts. Moreover, ONOS can add dynamically and efficiently switches and instances and deal instantly with network failures.

ONOS presents a decent level of consistency and integrity, because it uses Titan that maintains the graphs' structural integrity and Casandra, which has a high level of consistency.

The first problem of prototype 1 is excessive data store operations. In other words, the task of mapping data, from the Titan graph to Cassandra, results in a significant number of data store operations, which slows the network. The second problem is the lack of notification and messaging system, which is essential for the proper communication between the controllers.

The second prototype focuses on improving the performance of the first prototype, while keeping the global network view consistent. Since the main problem of the first prototype was an excessive data store operation, in prototype 2, the authors will try to solve this issue, following two complementary approaches. The first one concerns making remote operations as fast as possible, while the second approach focuses on reducing the number of remote operations.

Following the first approach, they implemented the Titan/Cassandra system with a Blueprints graph implementation on the top of a data store called: RAMCloud [20], which has a low latency of order of 15–30 $\mu$s.

Following the second approach, they created a cache topology system. This way some of the remote data store operations are stored in the memory cache. Likewise, they can reduce the number of data storage operations globally in the system.

To remove the problem of notifications intercontrollers, the authors adopted a communication system based on

Hazelcast [21]. These communications will go through some channels installed at the top of all instances of the control plane.

*4.4. DISCO.* DISCO [22] has two parts: an intradomain and an interdomain.

The intradomain part is responsible for monitoring the network and managing the flow prioritization. It contains a set of modules that dynamically handle the multiple network issues, like broken links, high latency, and bandwidth. The most important module is the Extended Database, which is the central component where all controllers store their information. Next is the Monitor Manager Module, which is responsible for gathering information, such as flow throughput, and calculating the one-way latency and the packet loss rate. Doing these operations periodically, the controller sends a current view of the link and the network devices' performances to the Extended Database Module. Additionally, the Events Processing Module keeps track of the variation or the saturation events while the Path Computation Module computes routes for flows, from a source to a destination inside the control plane. Finally, the Service Manager Module handles the network SLAs. An SLA is a service-level agreement, which is a contract that documents what customers have requested from a service provider.

The interdomain part provides the communication among the multiple controllers and has two modules.

First is the Messenger Module that builds channels between neighboring controllers to share information with the link state and the host presence. Communications use in general a well-known protocol as OSPF or BGP. The authors have chosen a protocol called AMQP [23], which provides routing, messaging with orientation, and prioritized querying. The Messenger only helps local agents to exchange information, but it does not support communication for wide area networks.

The Agents Module contains four main agents, starting with the Connectivity Agent that shares information about connecting controllers and their neighboring information. Next is the Monitoring Agent, which periodically sends information about latency and bandwidth that are available for the network to all the connected devices. Then it is the Reachability Agent that advertises the presence of a device in the network to become reachable by all the other devices. And finally, the Reservation Agent reserves update requests of the network, including capability requirements.

The evaluation study of DISCO followed three use cases. The first one was a scenario that puts in the challenge of self-adaptation of the control plane in a case of failure. The results have shown that, after the Monitoring Agent discovers the failure, the Connectivity and the Reachability Modules take in charge the task of failure recovery, working together with the Messenger.

The second scenario showed how DISCO is helpful in resource reservation, thanks to the Service Manager and Reservation Agents.

The third scenario shows how DISCO can manage to migrate a virtual machine, from one domain to another inside a DISCO architecture, with low latency and high reachability.

*4.5. ELASTICON.* ELASTICON [24] has an elastic architecture. It has a cluster of autonomous controllers that share the workload to provide a global network view for the management layer. This global network view is built by the Distributed Data Store Module. Likewise, every controller has a TCP channel, connecting it to a neighboring controller to ensure exchanging messages intercontrollers and switch migration.

At the physical layer, which contains switches, each one is connected to multiple controllers where one of them is the master, and the others are slaves. Each controller has the Core Controller Module that takes in charge all the responsibilities of a centralized controller. It also gives the controller the ability to negotiate with other controllers to choose the master of the topology. The primary feature of ELASTICON is elasticity, which is represented in this case as the switch migration procedure.

In the switch migration process, controllers can be added or removed according to some predefined thresholds, which represent the traffic load of the network. The load balancing is performed periodically and independently of the traffic load.

For its evaluation study, ELASTICON used an enhanced Mininet Testbed [25], which emulates a network of Open v-Switches [26]. The experimental results show that adding controller nodes increases the throughput almost linearly, and also the throughput reduces when they restrict the controllers to two cores. Additionally, it proves that the response time increases marginally, up to the point when there is a higher packet generation rate when ELASTICON has a higher number of nodes. Also, the study shows that the load balancing via switch migration can improve the performance.

The evaluation process also indicates that the switch migration process takes about 20 ms, which proves the speed of the process.

## 5. Logically Distributed Architectures

*5.1. KANDOO.* KANDOO [27] is a logically distributed controller with a hierarchical design of two layers.

The lower layer contains local controllers, where each one controls its subdomain, while the upper layer contains the root controller, which leads all the lower layer.

The local controllers only reply to events that are previously subscribed by the root controllers. Therefore, if the developers want to deploy new applications, they need to configure the root controller manually to permit it to subscribe to the new application. So, this controller is not a zero configuration framework.

KANDOO can coexist with other controllers on the same network and be customized to the specified needs of the network.

KANDOO's authors did a performance study using different applications in an emulated environment by presenting the results obtained about the elephant flow detection problem.

This elephant flow is a large continuous flow over a network link that decreases the total bandwidth after a certain amount of time.

KANDOO has two types of deployed applications, App detect, which is running on the top of all the local controllers, while App reroute is installed only on the root controller. App detect fires one flow per second and reports a flow as an elephant, if it has sent more than 1 MB of data. The final results show that KANDOO scales significantly better than a traditional OpenFlow network, while solving the elephant flow problem increasingly.

However, the study has shown no information about workload and performance of local controllers, comparing to a standard OpenFlow topology.

*5.2. ORION.* ORION [28] is a hybrid hierarchical control plane, which is a mix of flat and hierarchical architectures. It tries to combine the benefits of both designs and put them all together in a hybrid structure to avoid the issues that each architecture separately faces, such as the superlinear computational complexity growth, caused by flat architectures when the network scales to a large one, and the path stretch problem of hierarchical designs.

A network controller by ORION has three layers: the physical layer, which contains all the physical devices, such as OpenFlow switches; the bottom layer of the control plane that includes the area controllers, which handle collecting physical device and link information, as well as dealing with intra-area requests and updates. This layer also has a significant task in the network, which is abstracting the network view and sending it to the management layer of the control plane, and, finally, the upper layer, which contains the domain controllers. A distributed protocol synchronizes the information of interdomain controllers to keep a global network view for the application layer. Abstracting views from the bottom to the upper layer can reduce the problem of computational complexity in large scale networks.

ORION relies on multiple modules to operate. First, the OpenFlow Base Module, which handles OpenFlow-related tasks, such as collecting information from switches and forwarding them to the control plane; second, the Host Management Module, which gathers the host information in the bottom layer using ARP packets and deals with problems, resolving unknown MAC addresses; third, the Link Discovery Module that manages collecting information on the multiple areas; next, the Topology Management Module, which abstracts the infrastructure layer's topology and sends it to the bottom layer and then abstracts the bottom layer's topology and sends it to the upper layer; and finally, the Storage Module, which stores information regarding hosts, switches, links, and other parts of the network.

ORION interdomain controllers' communication relies on the Horizontal Communication Module that synchronizes the information among domain controllers to build a global network view, while interarea and domain controllers' communication relies on the Vertical Communication Module, which is a set of TCP connections that permit area controllers to send the abstracted topology of the infrastructure layer and request information from the domain controller when a host in some domain wants to reach a particular host in another domain. This discussion leads us to talk about the Routing Management Module, which controls all the routing tasks, using the Dijkstra algorithm.

ORION has made a theoretical and an experimental evaluation to test the performance of its control plane.

On one hand, the theoretical evaluation shows that the computing time of ORION has a linear growth, which is much lower than the traditional Dijkstra routing algorithm.

On the other hand, the experimental evaluation tried to verify the feasibility and the effectiveness of ORION. The study, made using Mininet, demonstrated that when the number of areas increases, the delay time also increases gradually. Additionally, ORION has low overhead.

Before the conclusion of this paper, we would like to discuss this section and the previous section, which provide various examples of multicontroller distributed architectures. On one hand, in Section 4, we have presented some proposals of physically distributed but logically centralized designs, like ONIX and HyperFlow, which are more suitable for datacenter and enterprise networks, and in many cases, they do not need a distributed protocol to ensure communication intercontrollers, and they have strong consistency. On the other hand, in this section, we have provided some examples of physically and logically distributed designs, which are more suitable for WAN networks. They are more likely to use a distributed protocol like BGP, and they usually have weak consistency. Finally, we think that there has been significant work concerning logically centralized architectures contrary to logically distributed architectures, which present many future research issues, like finding new methods to improve the global consistency, or developing standardized east-west interfaces to connect between different types of controllers.

## 6. Conclusion

Software-defined networking is based on the idea of splitting the control plane and the forwarding plane and, following that, centralizing the whole control plane in one single controller that manages the entire network.

However, throughout the years, the academia and the industry in the networking field started to realize that the future of SDN relies on distributed architectures, because centralized architectures do not fulfill the needs of efficiency, scalability, and availability. In this paper, we tried to provide a comprehensive overview of SDN multicontroller architectures by explaining their characteristics and presenting in detail different examples of implemented and under research architectures and solutions.

Network researchers and designers will have to deal with many problems that distributed architectures face to enhance a multicontroller network, like developing an efficient communication process, creating an adequate network design, or integrating new applications into the northbound interface that support multiple controllers.

## Competing Interests

The authors declare that they have no competing interests.

## References

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] W. Xia, Y. Wen, C. Heng Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[4] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: a survey," *Computer Communications*, vol. 67, pp. 1–10, 2015.

[5] Open Networking Foundation (ONF), https://www.opennetworking.org/.

[6] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "OpFlex Control Protocol," Internet Draft, Internet Engineering Task Force, 2014, http://tools.ietf.org/html/draft-smith-opflex-00.

[7] A. Doria, J. H. Salim, R. Haas et al., *Forwarding and Control Element Separation (ForCES) Protocol Specification*, Internet Engineering Task Force, 2010, http://www.ietf.org/rfc/rfc5810.txt.

[8] M. Sune, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis, "An OpenFlow implementation for network processors," *Defined Netw*, p. 2, 2014.

[9] B. Pfaff, B. Lantz, B. Heller et al., *OpenFlow Switch Specification*, 2012.

[10] D. Erickson, "The Beacon OpenFlow controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 13–18, ACM, New York, NY, USA, 2013.

[11] N. Gude, T. Koponen, J. Pettit et al., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[12] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, ACM, Helsinki, Finland, 2012.

[13] M. T. Koponen, M. Casado, N. Gude et al., "Onix: a distributed control platform for largescale production networks," in *Proceedings of USENIX Operating Systems Design and Implementation (OSDI '10)*, Vancouver, Canada, 2010.

[14] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking (INM/WREN '10)*, Berkeley, Calif, USA, 2010.

[15] J. Stribling, Y. Sovran, I. Zhang et al., "Flexible, wide-area storage for distributed systems with wheelfs," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, Boston, Mass, USA, April 2009.

[16] U. Krishnaswamy, P. Berde, J. Hart et al., "ONOS: an open source distributed SDN OS," 2013, http://www.slideshare.net/umeshkrishnaswamy/open-network-operating-system.

[17] Titan Distributed Graph Database, http://thinkaurelius.github.io/titan/.

[18] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, 2010.

[19] Tinkerpop. Blueprints, http://blueprints.tinkerpop.com/.

[20] J. Ousterhout, M. Rosenblum, S. M. Rumble et al., "The case for RAMClouds: scalable high-performance storage entirely in DRAM," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 92–105, 2010.

[21] Hazelcast Project, http://www.hazelcast.org/.

[22] K. Phemius, M. Bouet, and J. Leguay, "DISCO: distributed multi-domain SDN controllers," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–4, Kraków, Poland, May 2014.

[23] AMQP, an advanced message queuing protocol that can be found on https://www.amqp.org/.

[24] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 7–12, ACM, Hong Kong, 2013.

[25] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX '10)*, article 19, ACM, 2010.

[26] "Open vswitch," http://openvswitch.org.

[27] S. H. Yeganeh, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 19–24, Helsinki, Finland, August 2012.

[28] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, "Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks," in *Proceedings of the 22nd IEEE International Conference on Network Protocols (ICNP '14)*, pp. 569–576, Raleigh, NC, USA, October 2014.

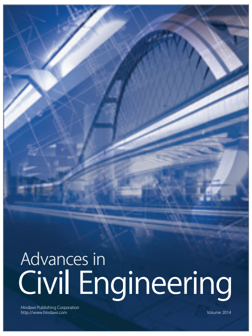
Journal of
Engineering

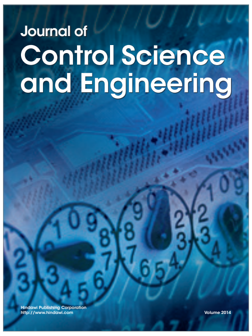
The Scientific World Journal

International Journal of
Rotating Machinery

Journal of
Sensors

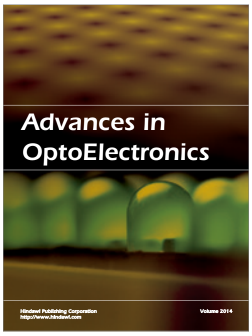
International Journal of
Distributed Sensor Networks


Advances in
Civil Engineering


Journal of
Control Science and Engineering

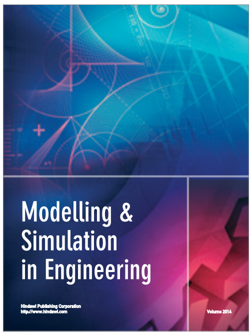
Journal of
Robotics

Journal of
Electrical and Computer Engineering

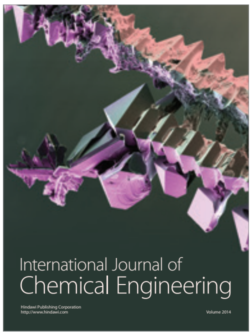

Advances in
OptoElectronics

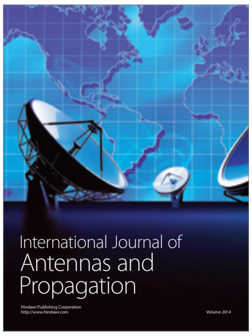
VLSI Design

International Journal of
Navigation and Observation

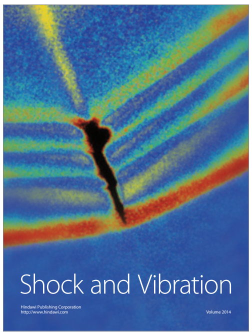

Modelling &
Simulation
in Engineering

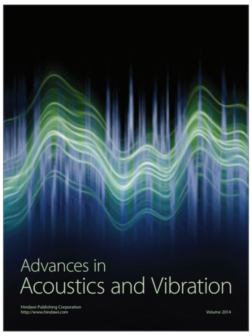
International Journal of
Aerospace Engineering


International Journal of
Chemical Engineering


International Journal of
Antennas and Propagation

Active and Passive
Electronic Components


Shock and Vibration


Advances in
Acoustics and Vibration