Computer Networks 121 (2017) 100–111

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Distributed SDN controller system: A survey on design choice

Yustus Eko Oktian, SangGon Lee*, HoonJae Lee, JunHuy Lam

Department of Ubiquitous IT, Dongseo University, Busan, South Korea

ARTICLE INFO

Article history:
Received 24 September 2015
Revised 31 January 2017
Accepted 10 April 2017
Available online 11 April 2017

Keywords:
Distributed SDN controller
Survey
Design choice
Pros and cons

ABSTRACT

Although within SDN community, the notion of logically centralized network control is well understood and agreed upon, many different approaches exist on how one should deliver such a logically centralized view to multiple distributed controller instances. In this paper, we survey and investigate those approaches. We discovered that we can classify the methods into several design choices that are trending among SDN adopters. Each design choice may influence several SDN issues such as scalability, robustness, consistency, and privacy. Thus, we further analyze the pros and cons of each model regarding these matters. We concluded that each design begets some characteristics. One may excel in resolving one issue but perform poorly in another. We also presented the design choices to build distributed controller that is scalable, robust, consistent and secure.

1. Introduction

Open Networking Foundation (ONF) [1] provisioned Software Defined Networking (SDN) architecture with a notion of a separation between the control plane and the data plane in network devices. ONF then joins all the segregated control planes into a new instance called SDN controller, which has a role in supervising the separated data planes. This new notion introduces a new network infrastructure that consists of three layers: infrastructure layer (i.e. network devices), controller layer (i.e. SDN controllers), and application layer (i.e. SDN applications). Among those three layers, two communication channels exist to tie each of them together.

- *The Southbound API* is the communication channel that resides between the infrastructure layer and the controller layer. This API comprises the protocols that dictate the way SDN controller directs the switch (e.g. collecting network statistics from the switch or inserting forwarding rules into the switch). OpenFlow [2] is arguably the most popular Southbound API for SDN. Thus, in this paper, we limit our scope to cover only SDN solutions with OpenFlow.
- *The Northbound API* is the communication channel that lives between the controller layer and the application layer. By using this API, SDN applications can take part in governing the network through the SDN controller. Contrary to the Southbound API, there is still no standardized protocol for this API.

SDN architecture applies a top-down logically centralized network control [1]. In this case, SDN controllers (together with the SDN applications installed) can centrally orchestrate all of the switches in the network. By having a centralized control, this design drives several benefits such as giving the ability to manage the network efficiently and react to the dynamic events quickly.

There are two implementation variants to reach the logical centralization in SDN: single and distributed SDN controller. In single SDN controller, only one machine of SDN controller exists in the network. That machine effortlessly leads all of the switches in a centralized manner since they are connected to the same instance. Many early SDN adopters are using this variant. Unfortunately, this variant has two principal issues.

- *Scalability Issue*. In general, the scalability covers the network ability to scale and control a tremendous amount of traffic. In SDN, the scalability reflects the capacity of the SDN controller in handling multiple forwarding path requests from switches. SDN controller has limited resource when handling an immense amount of requests. To solve this issue, researchers try to limit the number of forwarding path request sent to the SDN controller [3,4]. This strategy, however, adds the intelligence back to the switch thus violating the concept of SDN.
- *Robustness Issue*. A single SDN controller has a single point of failure problem. When a SDN controller fails, switches will miss their ability to forward new packets, and eventually, the entire network will break. To solve this issue, researchers can configure the switch as an OpenFlow-hybrid [5]. The switch will transform from the OpenFlow switch to the traditional switch (i.e. non-SDN switch mode) when it fails to connect to the SDN controller.

* Corresponding author.
E-mail addresses: yustus.oktian@gmail.com (Y.E. Oktian), nok60@dongseo.ac.kr (S. Lee), hjlee@dongseo.ac.kr (H. Lee), timljh@gmail.com (J. Lam).

http://dx.doi.org/10.1016/j.comnet.2017.04.038
1389-1286/

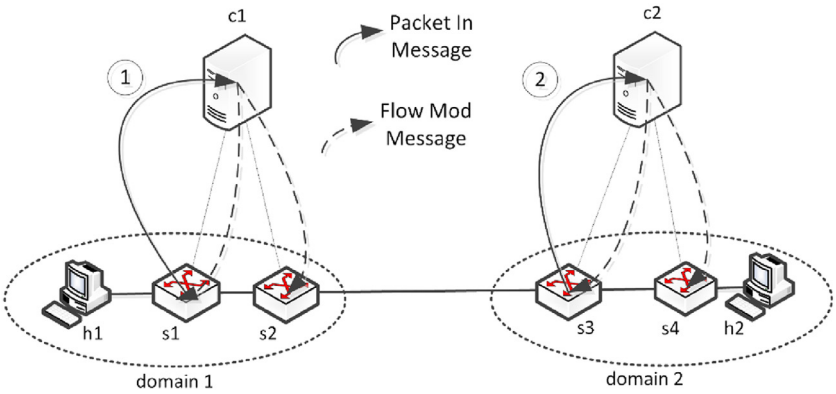

Fig. 1. An example of multiple SDN controllers works independently in routing packets across multiple domains. The Packet-In message is sent to ask the forwarding path from the SDN controller. The Flow-Mod message is forwarded to install the forwarding path to the switches.

1.1. Distributed SDN controller

As a second variant, distributed SDN controller promises to resolve both issues exist in the single SDN controller. The general idea is by forming multiple controllers, which can share the load in the network equally. Furthermore, one controller can take over another controller when it crashes. In detail, these procedures are sophisticated, which may require some technology adoptions from the distributed system.

Moreover, the distributed SDN controller architecture is not only about owning multiple controllers. Consider the following example as depicted in Fig. 1. The network topology consists of two SDN controllers. Each of them manages part of the network called the network domain. *c1* manages switches inside domain 1, and *c2* manages switches inside domain 2. Now suppose h1 requires sending packets to *h2*. When the packets reach *s1, s1* requests for a forwarding path from *c1* through OpenFlow Packet-In message. After obtaining the forwarding path, it forwards the packets to *s2*. The same schemes happen when the packets arrive at *s3. s3* demands for a forwarding path (i.e. again through Packet-In message) from *c2* then eventually the packets arrive at *h2*. In this example, there are multiple SDN controllers deployed in the network, but they work independently for each network domain. Therefore, we see two forwarding path requests happen (in *s1* and *s3*), and this does not represent distributed SDN controller architecture.

Now consider that if *c1* and *c2* are sharing the same logic in a logically centralized manner such that when new packets arrive at *s1*, both *c1* and *c2* can directly install forwarding paths in all corresponding switches. In this scenario, a single forwarding path request (i.e. happen in *s1*) can directly deliver the packets to the destination as if all the switches connect to a single SDN controller. This example illustrates the terms of *physically distributed but logically centralized SDN controller*, which we knew as distributed SDN controller.

1.2. Roadmaps

Distributed SDN controller has grown rapidly in the past couple years. Since there is no standard in constructing distributed SDN controller, a mixture of distributed SDN controller projects exist in the literature. In this paper, we survey and examine those projects. We can classify that distributed SDN adopters embrace several design trends in their architecture regarding the switch to controller connection strategy, network information distribution strategy, controller coordination strategy, and whether the controller employs in-band or out-of-band connection.

The remainder of the paper is organized as follows. We discuss the property of distributed SDN controller in Section 2 and present our design choice survey in Section 3. In Section 4, we analyze pros and cons of each design choices in solving SDN major issues such as scalability, failure, consistency and privacy. In Section 5, we discuss the mapping of several distributed SDN controller project regarding their design choice and address several related works that deliver several considerations and solutions in solving SDN issues. Lastly, we conclude in Section 6.

2. Property of distributed SDN controllers

In this section, we discuss the property of distributed SDN controller, items that distributed SDN controller should have to control the network in a logically centralized manner.

2.1. Network domains

In a distributed SDN controller, a single SDN controller controls a chunk of the wide-area network called domain. We define a domain of an SDN controller to be the set of all switches directly connected to the same controller and all hosts attached to this switch. For instance, in Fig. 1, *h1, s1*, and *s2* are in the same domain 1 because all of them are in control of controller *c1*.

There is a second definition of domain in distributed SDN controller. Network operators may split the wide-area network into multiple domains based on the group that is responsible for that particular network. We call this domain as the administrative domain, which is an area where the same organization administers the network. For instance, in Fig. 1, if the same company operates *c1* and *c2*, then all hosts and switches in the figure are in the same administrative domain.

2.2. Local network state

Local network state is a representation of the current state of the network in a domain that is stored locally in the SDN controller. This state helps the SDN controller to be conscious of every event that happens in the network such as host join/leave and link up/down. Lin et al. [6] classify the network state into two categories: static network state and dynamic network state.

Static Network State. The static network state comprises network information that is not modified/updated frequently. It may include the following:

- *Reachability*. This information informs whether an address in the network is reachable. They may include some of the network events such as hosts leave/join, devices up/down, links up/down.

- *Topology*. A network typically consists of nodes (e.g. switches, routers, firewalls, and hosts), links, link bandwidths, port throughputs and link connections. Based on this information, SDN controller can form a network topology.
- *Network Service Capabilities*. A network domain may have specific network policy such as Service Level Agreement (SLA) or Generic Routing Encapsulation (GRE).
- *Quality of Service (QoS) parameter*. A pre-defined QoS parameter such as latency, packet loss rate, availability, throughput, time delay variation, and cost.

Dynamic Network State. The dynamic network state comprises the network information that is modified/updated frequently. The dynamic aspect of the network mainly includes the flow table of the switches, the real-time bandwidth utilization, and the flow paths (forwarding paths) in the network. All of these attributes are likely to change over time because of the dynamic nature of the network.

To construct a local network state, an SDN controller must periodically query both static and dynamic network information from the network, which can be obtained from two different sources.

- *Network policies given by network operators*. Network operators can take part in managing the network through the SDN applications or any Northbound API that the controller provides. For instance, the operators can inject proactive flow rules that justify the Access Control List (ACL) policy.
- *Network statistic from network devices*. SDN controllers can collect network statistics directly from network devices through OpenFlow protocol [5]. For instance, the controllers can periodically query for port statistics and flow statistics to acquire detail information regarding the congested traffics or to identify when a device or port is up/down.

2.3. Global network-wide state

A single SDN controller has a limited vision of the whole network. For example, back in Fig. 1, *c1 is* only able to see *s1, s2*, and *h1* exist in the network. Meanwhile, the presence of *s3, s4*, and *h2* are not visible to *c1*. The SDN controller can only centrally administer the network when it can view the entire network. Therefore, the SDN controllers also need to have a global network state, which is a representation of the state of the network across multiple domains.

To construct a global network state, each controller in multiple domains must distribute parts of their local network state to other controllers. Afterward, SDN controllers merge their local network state with the others and build the global network state. There is no standard governs what information should be exchanged among SDN controllers. However, in general, the controller can share the following knowledge as summarized in Table 1.

- *Local Network State*. SDN controller can share any static or dynamic network information from their domain.
- *Cross-controller Event*. Some events are not part of the local network state events but they are useful for the sake of distributed SDN controllers. For instance, the controller up/down event is helpful to identify failed SDN controller. SDN application-specific event such as the one studied in Kandoo [7] is valuable to route traffics across multiple domains.
- *Inventory*. This information covers the inventory of a particular SDN controller that is useful for other controllers. For instance, knowing the list of installed SDN applications and the list of connected devices in an SDN controller can be beneficial to fasten the SDN controller failover mechanism [8].

Table 1
Classification of sharable network information.

Category	Examples
Local Network State (*Static*)	Reachability (*host join/leave, link up/down, device up/down*), topology (*node, links, link bandwidth, port throughput, link connection*), capabilities (*SLA, GRE*), QoS parameter (*jitter, latency, packet loss rate, time delay variation, cost, throughput*)
Local Network State (*Dynamic*)	flow tables, flow rules, real-time bandwidth utilization, flow paths
Cross-controller Event	controller up/down, SDN application spesific information
Inventory	list of installed SDN applications, list of connected switches

2.4. East/Westbound API

The East/Westbound API is the communication channel that exists solely for distributed SDN controllers. This API provides the connection between multiple SDN controllers for coordination purpose. For instance, SDN controllers can share their local network state through this API to construct a global network state. Another example, this API is used for the SDN controller failover mechanism. When one of the SDN controllers fails in the cluster, other controllers must be ready to take over the role of the failed controller. Lastly, the cluster leader can also use this API to do a load balancing mechanism among the SDN controllers. When the leader can equally distribute the weight of the network among the SDN controllers, the network can gain the best performance and efficiency.

3. Distributed SDN controller design choice

Distributed SDN controller has grown rapidly resulted in lots of projects are available in the literature. Furthermore, the fact that there is no standard in fulfilling distributed SDN controller allows the SDN adopters to implement a variety of distributed SDN controller solution tweaked arbitrarily to meet their needs. However, we have surveyed the literature and observed four design trends that SDN adopters choose during their implementation. The choice of each design may have some impacts on the quality of distributed SDN controller in scalability, robustness, consistency and privacy factor. We will explain them in the next section. Meanwhile, in this section, we explore each design choices in more details.

3.1. Switch to controller connection strategy

The first design choice is regarding how switches connect to SDN controllers. In early OpenFlow version, switches can only attach to one controller. Furthermore, that link is static, meaning that operators have to configure the switch manually when it needs to attach to a new controller. The distributed SDN controller, in another hand, require a dynamic connection between switches to controllers. The dynamic connection helps the controller to move a switch from one controller to another controller during a failover or load balancing process. Fortunately, there are two options to deploy such flexible switch to controller connection, using the IP alias connection or OpenFlow Master/Slave connection.

3.1.1. IP alias connection

Yazici et al. [9] harness the IP aliasing technique, which is a technique to allocate multiple IP addresses to a single network interface, to create a unique dynamic switch to controller connection strategy. By using this technique, the switch can statically connect

to a single SDN controller at any given time. However, it can still reconnect to another controller dynamically without any reconfiguration required in the switch.

In their work, operators statistically configure each switch to connect to only one SDN controller at any given time. For instance, a switch with the IP address *s1* connects to an SDN controller with IP alias address *p1*. Hence, their static IP mapping is (*s1, p1*). Meanwhile, a switch with IP address *s2* connects to a controller with IP alias address *p2* result in an IP mapping of (*s2, p2*). The cluster leader stores and maintains this mapping information as a pool of IP address mapping such as (*s1, p1*), (*s2, p2*), (*s3, p3*).

At any given time, the cluster leader can dynamically assign the IP alias address from the pools to any controller. Suppose we have a controller with IP alias address *p2* currently connecting to a switch with IP address *s2* (*s2, p2*). This controller fails, and the switch *s2* needs to connect to another controller to continue operation. The cluster leader assigns the IP alias address *p2* to a new controller. After the change is applied, the switch *s2* automatically connects to that new controller without any reconfiguration needed in the switch.

3.1.2. OpenFlow master/slave connection

In general, SDN controller has two access when governing the switch: read and write access. Read access gives the controller the ability to read the state of the switch. For instance, collecting the Flow Table information from the switch. Write access, in another hand, gives the controller the ability to write/modify the state of the switch. For example, modifying the contents of Flow Table in the switch.

OpenFlow version 1.2 [10] introduces three new roles for SDN controllers. Each of roles determines the switch access privilege given to the controller. The controller with Master role holds the highest privilege, which enables the controller to have read and write access to the switch. Similar to the Master role, the Equal role also owns the same read and write access. Meanwhile, the Slave role has only read access to the switch. By using these three roles, the switch now can connect to two or more controllers simultaneously. However, the switch can only attach to one controller with the Master role at any given time. No limitation applies to Equal and Slave role regarding the number of connection.

A cluster leader can change the role of a controller dynamically, enabling the switch to migrate from one controller to another as well. Suppose switch *s1* connects to both Master role controller *c1* and Slave role controller *c2*. At a given time, *c1* fails resulted in *s1* loses its primary controller. The cluster leader can migrate *s1* from *c1* to *c2* by swapping the role of *c1* and *c2*. In the new configuration, *s1* connects to Master role *c2* and Slave role *c1*. The last but not least, operators can utilize OF-Config protocol [11] to dynamically configure the switch connection to the controller when needed.

3.2. Network information distribution strategy

Theoretically, a single controller can centrally control the entire switch because it has a vision of the whole network. Therefore, to provide a logically centralized control in distributed SDN environment, there must be at least one controller in the cluster that has the global network state. Hence, sharing and distributing the local network information among SDN controllers is inevitable. Our survey of the literature summarizes two design choices. Each option utilizes a different method to distribute the network information among SDN controllers.

- *Hierarchical model*. One or some (but not all) SDN controllers in the cluster have the global network state.
- *Flat model*. All of the SDN controllers in the cluster have the global network state.

3.2.1. Hierarchical model

The hierarchical model is also known as vertical architecture in ALTO [12] because of the hierarchical controls that run from top to bottom among SDN controllers. Fig. 2(a) illustrates this model. Two local SDN controllers maintain the switch in each domain. Meanwhile, the root controller manages the coordination between local controllers. The global network state, which is stored in the datastore, is only available in the root controller. Thus, the local controller must first query network information from the root controller before it can execute any inter-domain operation. Due to this notion, this model is also known as client and server model, where the root controller acts as a server and local controllers as clients. Other important details regarding this model.

- Operators can replicate the role of a centralized root SDN controller in multiple controllers to mitigate the single point of failure issue.
- Each of the local SDN controllers in the cluster does not have an East/Westbound API connection to other local controllers. Instead, every local SDN controller only has a link to the centralized root SDN controller.
- The root SDN controller stores and maintains an up-to-date global network-wide state. Meanwhile, local SDN controllers keep only their respective local network state.
- The root controller must govern the operation of the subordinate local controllers to provide connectivity throughout the entire network [7].

IRIS use case example. IRIS [13] is one of the distributed SDN controller project that adopts the hierarchical model. During network runtime, IRIS local SDN controllers collaboratively upload the network information they want to share to the IRIS root SDN controller periodically. This way the root controller can sustain an up-to-date global network state. Complementary, the local controllers can also inquire the network information they need from the root SDN controller.

Fig. 3 depicts a network topology with IRIS controller implementation. Suppose *host #1* needs to send packets to *host #2*, which located in a different administrative domain from *host #1*. Before sending the IP packets, *host #1* sends the ARP packet to look for the MAC address of *host #2* to forward the packets. The switch forwards the header of the ARP packet to the *local controller #1* during the flow setup. Since the controller does not have any information regarding *host #2*, it tries to query the required information from the higher tier, the *root controller #1*. However, the *root controller #1* also does not have any information regarding *host #2*. Thus, it tries to query from another higher tier controller, the *root controller #3*, which holds the overall network information of the network. The *root controller #3* then instructs the subordinate controllers, which are affected by this flow setup, to forward the ARP packet through *Router A, B*, and *C*. After the flow setup finishes, the switch can deliver the IP packets directly to the destination, *host #2*.

3.2.2. Flat model

The flat model is also known as horizontal architecture in ALTO [12] since there is no hierarchical control among SDN controllers in this model. Fig. 2(b) illustrates this principle. Three SDN controllers control switches in their domain and all of them have a datastore to store and maintain the global network state. All of the SDN controllers also own East/Westbound API connection to other controllers. Thus, they can contact and notify each other directly.

Before an SDN controller can construct the global network state, it must first receive the local network state from every SDN controller in the cluster. Also, any changes happen inside a domain of a controller must be shared to other controllers so that they

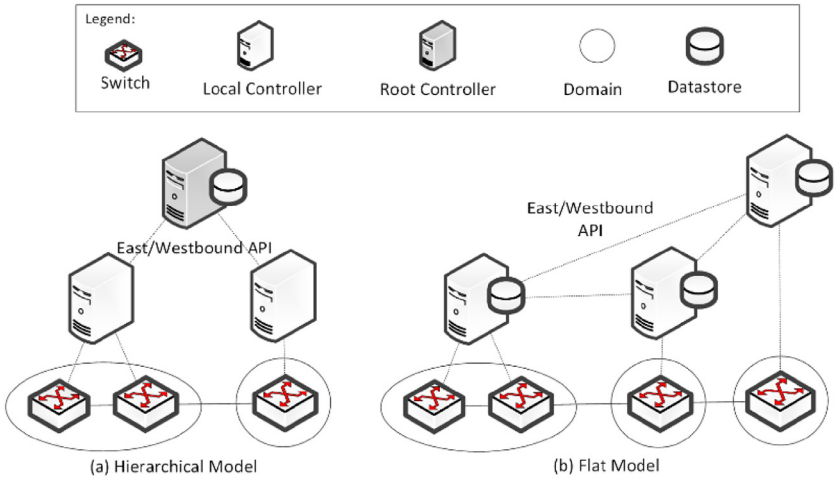

Fig. 2. An illustration of the hierarchical model and flat model.

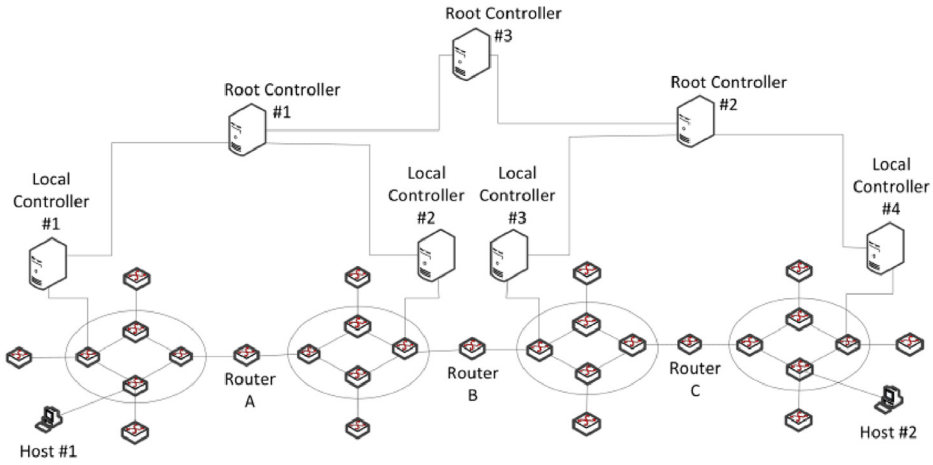

Fig. 3. An example of IRIS hierarchical model network topology.

can update their global network state to reflect the changes. When the cluster able to perform every network information distribution correctly, all SDN controllers should have the same global network state. Due to this notion, operators also call this model as a peer-to-peer model since every controller can reach directly to other controllers during network information distribution. Others may also call this model as replicated state machine model because the cluster replicates all local network states from SDN controllers to one another.

There are two ways to acquire the network information from other SDN controllers.

- *Polling*. Each SDN controller can periodically request for a new network information from other controllers in the cluster. For instance, an SDN controller request for a new switch information every 10 minutes. It will execute the request periodically even when there is no update happen in the other controller. Thus, it may receive the same network information as the last one. Therefore, this method may not be efficient.
- *Publish and Subscribe*. Each SDN controller can publish/subscribe the network information from other controllers in the cluster. For instance, an SDN controller *c1* needs an update of switch information from its neighboring controller *c2*. The controller *c1* can subscribe for a switch information from *c2*. In this case, *c1* acts as a subscriber and *c2* acts as a publisher. Later, *c2* will notify *c1* when there is a change regarding the switch information in its domain. In this scenario, the controller *c2* will issue an update only when there is a change. Therefore, this method is more efficient.

ONOS use case example.

ONOS use case example. ONOS is one of distributed SDN controllers that adopts the flat model and achieve a logically centralized SDN controller through a replicated state machine model. In ONS 2015 conference [14], Madan Jampani discusses that there are two challenges arise in this model: how to ensure ordering between events that are broadcasted arbitrarily from multiple controllers and how to handle failure that can happen during the event broadcast transmissions.

To solve ordering problem, ONOS puts the logical timestamp into the event. In every case of network changes, the events are timestamped in the source controller (i.e. where the network changes happen) before they are broadcasted to the cluster. Upon receiving events, the controller first examines the timestamp. If it is a stale event, then the controller drops it. Furthermore, event messages are structured in the (*switch_id, term_number, event_number*) format based on Raft [15] to achieve excellent event ordering mechanism. The *switch_id* is the unique switch identifier. The *term_number* gets incremented when the controller failures and leader re-election happen such that the switch changes mastership from one controller to another. Meanwhile, the *event_number* increments when particular events occur in the switch. With these three additional parameters, the controller can determine the order of the incident. For instances, when the con-

troller receive two events (*S1,2,2*) and (*S1,2,1*), then it knows that (*S1,2,1*) happens before (*S1,2,2*).

To solve event failure problem, ONOS periodically synchronize with the other controllers and exchanges metadata of the current network state. When the controller identifies that the received metadata contains the later timestamp than its own, it requests for the most recent network state from the source controller. In this way, the controller can achieve better consistency and solves the broadcast transmission failure.

3.3. Controller coordination strategy

The third design choice is regarding who can program the network. Multiple SDN controllers exist in the network may cause network inconsistency. Consider the following example. Controller A requires to update forwarding path Y to a switch under controller C domain. Meanwhile, controller B wants to delete forwarding path Y from the same switch. Without coordination, controller C has difficulties in determining which operation to execute. SDN controllers can apply a consensus algorithm [16] to reach agreements. In distributed systems, there are two ways to gain consensus: leaderless and leader-based approach. The same scenario applies for distributed SDN controller.

3.3.1. Leaderless approach

The first approach is also known as the symmetric approach, in which there is no leader/ruler among SDN controllers. Each controller has the same role/privilege and can directly contact other controllers to program the network arbitrarily. One example from the literature that adopts this approach is HyperFlow [17]. When HyperFlow controller needs to install forwarding paths to the switch belongs to other controllers, it serializes and deserializes OpenFlow message to the destination controller. There is no leader during this operation.

3.3.2. Leader-based approach

The second method is also known as the asymmetric approach. It displays the existence of a cluster leader to govern multiple SDN controllers in the cluster. The job of the leader is to ensure that the majority of the controllers behave the same as if it is a single SDN controller. Therefore, the priority task of a leader is to prevent the race condition and network partition from occurring in the cluster. Most of the time, the leader receive network operation demands such as inserting/deleting forwarding paths from an SDN controller. Before the cluster commits the request, the leader has to approve it by assuring that the demand passes the consensus, Thus, by design, the controller leader has higher privilege.

To prevent race condition and inconsistency in the network, the cluster must ensure that only one leader is managing the network at all time. Depending on the network distribution strategy in Section 3.2, the cluster may implement a different method to achieve that goal.

- *Hierarchical model*. Because only the root controller owns and manages the global network state for the cluster, it has to acts as a leader for the subordinate controllers. The role of the root controller as a leader is by design and inevitable.
- *Flat model*. Since each SDN controller owns the global network-wide state, the cluster must elect a leader using a leader election algorithm [15]. The cluster can conduct the election process in a straightforward way. One SDN controller initiates the leader election by voting for itself to be a leader candidate and broadcast the vote to other controllers in the cluster. In case the non-leader candidate SDN controller obtain multiple votes, the controller picks one with the lowest machine ID. In other words, the vote with the lowest machine ID wins. Finally, the vote will be committed when the majority of the controllers in the cluster agree on the same controller to become a leader.

One thing to be noted, in a leader-based environment, each SDN controller can talk to the cluster leader but not to the other SDN controllers. The leader is the only instance that can communicate to the rest of controllers in the cluster. Thus, every message or network event going to a particular SDN controller has to go through the cluster leader before that event message is executed in the destination controller [11].

3.4. In-band vs. out-of-band connection strategy

OpenFlow specification [5] states that network operators can set up the Southbound API with two types of connections: in-band or out-of-band connection. This choice applies to any management connection. Thus, operators can also choose whether to use in-band or out-of-band to configure the East/Westbound API.

- *Out-of-band connection*. The out-of-band connection requires a dedicated link for management traffics. In the case of Southbound API, the link contains the switch coordination messages from the SDN controller such as OpenFlow, OF-Config, and OVSDB protocol. Meanwhile, in the case of East/Westbound API, the link comprises the controller coordination messages such as leader election, network information distribution and controller failover mechanism.
- *In-band connection*. The in-band connection does not require a dedicated link. Instead, it utilizes existing data links (i.e. the links that transfer data traffics) to transfer management traffics. Thus, the management and data traffics are in the same link.

4. Distributed SDN controller design analysis

Each design choice presented in the previous section affects several distributed SDN controller issues such as scalability, failure, consistency and privacy. In this section, we elaborate those implications by analyzing pros and cons of each design in regards to the issues above. Our analysis is based on our study of the literature that we surveyed. We give a plus (+) point when we find an advantage from a particular design choice. Otherwise, we give a minus (−) point when we find disadvantages or flaws.

4.1. The fundamental issues

Scalability. Since there is a constraint on the number of forwarding path requests that one SDN controller can handle, it limits the performance of the network. Distributed SDN controller arises to tackle this problem by establishing multiple SDN controllers on the network and managing it cooperatively. The presence of multiple SDN controllers can boost the performance of the network since the controllers can distribute the load of the network among them.

Failure. The single point of failure issue exists in the single centralized SDN controller architecture is a nuisance. When the controller fails, the entire network will also fail. By presenting multiple SDN controllers in the network, one controller can take over other controllers when one of them fails.

Consistency. To act as physical distributed but logically centralized SDN controller, multiple SDN controllers must beget the same global network state at all time. When there is an inconsistency in that global network state, the network can be falsely partitioned into multiple states, which may result in false network operation. Therefore, having a consistent global network state is inevitable.

Table 2
Pros (+) and cons (−) of switch to controller connection strategy.

Parameter	Master/Slave	IP Alias
Scalability		
Master/Slave has a reliable switch migration protocol on the fly [18] that guarantees safety and liveness.	+	−
Failure		
Both designs support the controller failover mechanism.	+	+
Consistency		
In Master/Slave, a scenario may put the network state in an inconsistent state when two or more SDN controllers has the same write access to the same switch.	−	+

Table 3
Pros (+) and cons (−) of network information distribution strategy.

Parameter	Hierarchical	Flat
Scalability		
The result from a study in [19] informs that the hierarchical model is more scalable.	+	−
Failure		
The flat model is easier to maintain, any controller in the cluster can simply take over a failed controller.	−	+
The flat model has a robust failover mechanism [20].	−	+
Privacy		
The hierarchical model can implement a network abstraction in a straightforward fashion.	+	−
Consistency		
The hierarchical model tends to be more consistent since less controller maintain the global network state.	+	−

Privacy. The privacy issue is a unique issue regarding sharing network information across multiple SDN controllers. Sometimes, exposing the local network state to other SDN controllers is not desirable. Consider the following example, SDN controller *c1* shares its local network state to SDN controller *c2*, which is governed by other network organization. Sharing *c1*'s local network state to *c2* means revealing the secret of the network information owned by its organization. Therefore, this action is undesirable. To solve this issue, the terms of network information abstraction arises. The network abstraction allows SDN controllers to share only the aggregated information of their local network state to the other controllers. The final output of the aggregation will be a high-level network information or network summarization to acts as a resource for other SDN controllers to construct the global network state.

4.2. Analysis of switch to controller connection strategy

Table 2 depicts the pros and cons of OpenFlow Master/Slave Connection and IP Alias Connection design. In the following paragraphs, we elaborate in more detail.

Scalability. OpenFlow Master/Slave design has a novel load balancing technique using a switch migration on the fly protocol [18]. This technique can migrate the switch from a densely loaded controller to the idle one, with the intention that the cluster can distribute the load of the network equally among multiple SDN controllers in the cluster. Thus, the network becomes more scalable and efficient.

The switch migration on the fly protocol is largely based on OpenFlow Master/Slave roles as well as OpenFlow messages. Thus, the cluster cannot perform the same protocol in the IP alias design. Furthermore, this protocol can guarantee the safety and liveness property during the switch migration. There is no packet loss or duplication happens on the network when this protocol is working.

We have to mention that the IP Alias Connection design also provides a mechanism to do a switch migration on the fly, similar to the Master/Slave design. However, there is still no safety and liveness property guarantees in that switch migration. Thus, Master/Slave design has an advantage here.

Failure. SDN controllers can use both designs in implementing SDN controller failover mechanism since both designs support the dynamic switch to controller connection.

- *In IP alias design*, the cluster can give the IP alias of the failed SDN controller to the new controller to take over the failed controller. Afterward, switches will be connected to the new controller automatically.
- *In Master/Slave design*, when an SDN controller fails, the cluster has to check if that SDN controller owns any Master roles to the underlying switches. When the Master role is found, the cluster has to assign the new Master role to the other controllers.

Consistency. The OpenFlow protocol [10] states that both Master and Equal role can have the same write/read access to the switch. Also, the switch can connect to multiple SDN controllers with the Equal role. Thus, a hole for network inconsistency may rise in this design. For instance, two inconsistent SDN controllers (one with a Master role and another one with an Equal role) can insert two conflicting Flow Rules to the same switch.

We cannot recreate this scenario in IP alias design since only one SDN controller can connect to one switch at any given time. Thus, this design gives a clearer thought to the clusters which can control a particular switch during network runtime. Therefore, IP alias design has better consistency.

4.3. Analysis of network information distribution strategy

Table 3 depicts the pros and cons of the hierarchical and flat model. In the following paragraphs, we elaborate in more detail.

Scalability. Hu et al. [19] study a metric of scalability for SDN controllers in four distinct controller architectures:

- Single centralized SDN controller.
- Distributed SDN controllers with flat architecture (local view). In this category, each SDN controller only has its local network state. The controller also has the local network state from its neighboring controller but it is abstracted into a single node.
- Distributed SDN controllers with flat architecture (global view). In this category, each SDN controller holds the global network-wide state. Thus, this category is the same as the flat model in the previous section.
- Distributed SDN controllers with hierarchical architecture. This category is similar to the hierarchical model in the previous section.

In their study, they measure the scalability of SDN controller in processing the flow initiation request. Based on their result, single controller obviously has the worst scalability points. Meanwhile, the flat distributed SDN controller with a local view has the best scalability points. The hierarchical model competes in the 2nd place with a subtle gap. Interestingly, the flat distributed SDN controller with the global view suffers heavily in their experiment (with a long gap between the hierarchical result). Based on this result, the hierarchical model has an advantage.

Failure. In the hierarchical model, the number of the root SDN controllers should be kept minimal. A higher number of root controllers may cause the network less efficient in the coordination perspective [21]. This notion disadvantage the hierarchical model in solving failure issue. The lesser number of root SDN controllers means lesser SDN controllers are available to take over the failed controller.

Furthermore, in the hierarchical model, SDN controllers are divided into two roles (root and local) with different capabilities. Because of this differences, the local SDN controller may not be able to take over the root controller and vice versa. The root controller may only take over the root controller, and the local controller takes over the local controller.

As a comparison, the SDN controller failover mechanism in the flat model is more straightforward. Since all the SDN controllers are similar in the form of replicated state machines, any SDN controller can take over any failed controller in the network. Furthermore, with replicated state machine, one can construct a robust controller failover mechanism that considers the safety and correctness properties in the switch and the SDN controller during the failover procedure [20].

Privacy. The way SDN controllers handle the network abstraction differs depending on the strategy.

- *In the hierarchical model*, higher tier network organizations should take the role of top SDN controllers (e.g. root controllers). Meanwhile, lower tier organizations take the role of the subordinate local SDN controllers [13]. All of the local SDN controllers can share the abstracted local network state to the root SDN controller. Then, the root controller can form an abstracted global network-wide state. With this abstracted version of the global network-wide state, root controllers do not know details regarding the local network state owned by the local controllers. Furthermore, each of the local controllers also does not know any information relating to the network of other local controllers. Thus, a win-win solution is settled.
- *In the flat model*, the cluster has to find a way to share abstracted network state across all SDN controllers. One solution is using a network abstraction table [6], which is a table that maps the value of real network state to the abstracted one. Each SDN controller has to maintain its network abstraction table and share abstracted local network state to other SDN controllers. Then, each of them can combine this information and form an abstracted global network-wide state, just like the one in a hierarchical model. The SDN controller has to update the network abstraction table regularly just like the local network state.

Based on the simplicity, the hierarchical model wins because the abstraction can be done simply by design. Thus, the network abstraction table is not required. Therefore, the hierarchical model has an advantage.

Consistency. In the hierarchical model, only root controllers hold the possession of the global network-wide view. Meanwhile, in the flat model, all SDN controllers maintains the global network-wide state. Thus, the number of SDN controllers with the global network-wide state is fewer in the hierarchical model compared to the flat model. With a few of SDN controllers owns the global network-wide state means that it is apparent to maintain its consistency.

4.4. Analysis of controller coordination strategy

Table 4 depicts the pros and cons of the leaderless and leader-based approach. In the following paragraphs, we elaborate in more detail.

Scalability. The leader-based approach is obviously slower than the leaderless approach since any network agreements have to be evaluated by the leader before they can be committed to the network. However, the leader-based approach turns out to be more scalable in a large nodes scenario [15].

Table 4
Pros (+) and cons (−) of controller coordination strategy.

Parameter	Leaderless	Leader-based
Scalability		
Leader-based is more scalable in practice [15].	−	+
Failure		
Additional processing: the cluster needs to execute leader election procedure when the leader fails.	+	−
Consistency		
Leaderless tends to be inconsistent.	−	+

Table 5
Pros (+) and cons (−) of in-band vs. out-of-band connection strategy.

Parameter	In-band	Out-of-band
Scalability		
Out-of-band is more stable compared to in-band, result in better performance.	−	+
Privacy		
Out-of-band is more secure and private compared to in-band.	−	+
Consistency		
In-band in some conditions is less likely to be partitioned [23].	+	−

Failure. In leaderless, the cluster has to do a new leader election process when the leader fails, which demands additional time and complexity. In contrast, when an SDN controller fails in a leaderless environment, the cluster only need to remove the existence of the failed controller from the voting boards. Thus, the leaderless approach may generate a faster controller failover mechanism.

Consistency. The leaderless approach does not have a leader. Therefore, there is a higher probability that inconsistency will happen in this environment. Also, the leaderless approach performs terribly during a network partition. Suppose there is a network partitioned occur in the network such that one network operation cannot get majority votes from SDN controllers. The leaderless approach cannot execute that operation since it does not pass consensus. Thus, the operation may be suspended or falsely executed [22]. The existence of a leader here may help in sustaining the same order within the partitioned network.

4.5. Analysis of in-band vs. out-of-band connection strategy

Table 5 depicts the pros and cons of the in-band and out-of-band connection strategy. In the following paragraphs, we elaborate in more detail.

Scalability. The distributed SDN controller with out-of-band configuration can deliver better scalability because the management links are more durable compared to the in-band configuration. In the in-band configuration, the load of the data traffics influences the management traffics thus degrading the performance of the SDN controller. When the data traffics is congested, the SDN controller may draw less bandwidth for their management traffics.

Privacy. Since the management traffics is shared with the data traffics in the in-band configuration, Attackers may steal or tamper any management messages that traverse through the link.

Consistency. Panda et al. [23] study the consistency of distributed SDN controller and find that the out-of-band connection may provide lower resistance when the distributed SDN controller suffers from network partitions. They build this finding based on

a scenario when the SDN controller cannot update their view of the network topology. They also suggest implementing the hybrid combination of out-of-band and in-band configuration, which can tackle the network partition better.

5. Discussion

In this section, we discuss multiple distributed SDN controllers and several related works that we have surveyed from the literature. We start our discussion by classifying existing distributed SDN controller project in regards to the design choice that we have presented in the previous section. Then we discuss several studies that are related to the fundamental issues of distributed SDN controllers such as scalability, failure, consistency, and privacy. Several projects offer valuable solutions in resolving these matters.

Classification. Table 6 depicts the classification of existing distributed SDN controller projects based on their design choices. We omit the classification for in-band vs. out-of-band connection strategy because most of the projects in the literature do not specify their implementation regarding this option. Based on our classification, we can learn several trends such as:

- The hierarchical model is most likely to adopt leader-based approach with the root SDN controller act as the leader for the subordinate SDN controller.
- Master/Slave connection is administered by the OpenFlow standard [10]. Therefore, most of the projects pick this mechanism over the non-standard IP Pool alias technique.
- Most of the distributed SDN controller projects are the modified version of the single SDN controller version. For example, ONIX [24] are based on NOX [25], DISCO [26] are based on Floodlight [27] with inter-domain modules.
- Most of the projects use the combination of OpenFlow Master/Slave, flat model, and leader-based design. Production-level SDN controllers such as OpenDaylight [28], ONOS [29], and ONIX [24] uses this combination. Thus, this is a popular combination.
- There is no standard, which governs the use of datastore and East/Westbound API in distributed SDN controller system. This issue results in a variety of datastore and East/Westbound API technology used by distributed SDN controllers. Thus, it is challenging to make two different SDN controller projects to work together since most probably they do not fit one another.

Scalability. Heller et al. [32] discuss the controller placement problem to determine how many controllers that we need and where should they be placed. According to their analysis, one controller is often sufficient to meet the response time requirements of the widely WAN implementation. However, we can gain more benefits when deploying more SDN controllers. For instance, in the same study, Heller et al. [32] analyze that deploying three controllers can reduce the average latency of the controller by half.

Jiménez et al. [33] had demonstrated that the number of controllers chosen and their placement alters the performance of the network. When we incrementally increase the number of SDN controllers, at some point, it tends to be costly than to be effective. Thus, the best result happens if the network can run a near 100% utilization with a small number of SDN controllers.

Elasticon [18] try to achieve 100% utilization with a novel idea that enables the cluster to add or remove SDN controllers on the fly based on the load of the network. The cluster maintains a congestion threshold value. When the traffics pass the congestion threshold, the cluster can add a new controller to join the cluster and help managing the network. After the new controller joins, the cluster must assign switches to that controller. Furthermore, the cluster can also remove SDN controllers from the cluster when the traffics declines below the threshold.

Table 6
Classification of existing distributed SDN controller project considering the distributed SDN controller design choices.

Distributed SDN Controllers Projects	Related SDN Projects	*Switch to Controller Connection Strategy*		*Network Information Distribution Strategy*		*Controller Coordination Strategy*		Datastore	East/Westbound API
		Master/Slave	IP Alias	Hierarchical	Flat	Leaderless	Leader-based		
East/West Bridge [6]	Floodlight [27], NOX [25]	–	–	–	✓	✓	–	Cassandra	Custom BGP
Elasticon [18]	–	✓	–	–	✓	–	✓	Hazelcast	Custom Protocol
DISCO [26]	Floodlight [27]	–	–	–	✓	✓	–	Extended DB	AMQP
HyperFlow [17]	NOX [25]	–	–	–	✓	✓	–	WheelFS	publish/subscribe messages
OpenDaylight [28]	–	✓	–	–	✓	–	✓	InMemory Datastore	Akka, Raft
ONOS [29]	–	✓	–	–	✓	–	✓	Raft Log	Raft
D-SDN [8]	–	✓	–	✓	–	–	✓	–	Custom MC-MC, MC-SC, and SC-SC protocol
IRIS [13]	Floodlight [27], Beacon [30]	✓	–	✓	–	–	✓	MongoDB	Custom Protocol
ONIX [24]	NOX [25]	–	–	–	✓	–	✓	DHT, NIB	Zookeeper
Kandoo [7]	–	✓	–	✓	–	–	✓	–	Messaging Channel
Controlling SDN [9]	Beacon [30]	–	✓	–	✓	–	✓	–	JGroups
FlowBroker [21]	Floodlight [27]	–	–	✓	–	–	✓	–	Broker protocol
Ravana [20]	Ryu [31]	✓	–	–	✓	–	✓	InMemory Log	Ravana protocol

Failure. Ravana controller [20] is a novel SDN controller that focus on resolving failure issues in distributed SDN controller environment. It has a goal to provide a failover mechanism that considers the total event ordering, exactly-once events, exactly-once commands, and transparency. It is also the only SDN controller to date, which can guarantee both the correctness and consistency of the failover mechanism even when the candidate of replacement controller fails during the controller failover mechanism. The Ravana protocol is the key to this success. It is a novel two-phase replication protocol, which includes additional buffers in the switch and the controller. Ravana utilizes those buffers to guarantee correctness. It buffers each event or commands before sending them to the respective switch or controller.

The correctness guarantee comes with a performance price. Based on their measurements. Ravana gains up to 31% throughput overhead during events processing. Ravana also has the average failover time of 75 ms, which includes 40 ms to detect the failure and elect a new leader, 25 ms to catch up with the old master, and 10 ms to assign a new role on the switch. Unfortunately, there are not many failover time measurements in distributed SDN controller. Thus, we cannot compare these metrics with the other projects.

Consistency. Levin et al. [34] found that there is a trade-off between the responsiveness and the consistency state of the network. A strongly consistent network state will lead to high overhead while a weak consistent network state will lead to high performance but less correct network operation. Furthermore, sometimes the consistency level is required on demand by the SDN applications depending on their current network tasks. Therefore, the SDN controller should offer a choice whether a particular network operation needs to be conducted in strong or weak consistency environment. There are two possibilities to reach this solution:

- SDN Controller has to provide two APIs, one for strong consistency and one for the weak one. For instance, in ONOS [29], SDN application can use *ConsistentMap* API for stronger consistency requirements and use *EventuallyConsistentMap* for weaker consistency.
- The second solution only works in a hierarchical model. The cluster can install SDN application that requires stronger consistency in the root controller where the global network-wide state resides. Meanwhile, the SDN application that only needs weak consistency can be installed in the local SDN controllers.

Based on the Consistency, Availability and Partition Tolerance (CAP) theorem, one can only support two of the three properties [35,36]. This theorem is crucial because the distributed SDN controller utilizes the datastore to store the global network state. The datastore has different flavors of consistency. Some datastore such as Cassandra [37]. MongoDB [38], and Dynamo [39] prefer AP. The other datastore such as Zookeeper [40], Chubby [41], Spanner [42] prefer CP. When the network operator implements this datastore, the distributed SDN controller most likely will inherit the datastore characteristic (i.e. whether it is AP or CP).

Privacy. Network operators own the Service Level Agreement (SLA), which is negotiable through multiple administrative domains in the network. This SLA governs how the SDN controllers are performing the inter-domain related operation. Sometimes the cluster must construct the network abstraction based on this SLA. For instance, operators can realize the routing SLA and create a virtual path based on whether the packets can compromise with low bandwidth and high latency. Lin et al. [6] present excellent examples of the network abstraction table that fit the SLA requirements. The network abstraction table maps the physical links from the non-abstracted topology with the virtual links from the abstracted topology based on the SLA. The SDN controller must keep this network abstraction table up-to-date. When the SLA changes, the network abstraction table must be ready to reflect the changes.

6. Summary and conclusion

Since there is no standard governs the implementation of distributed SDN, many SDN adopters develop a variety of distributed SDN controller version to meet their needs. However, our survey finds that there are several design trends, which the SDN adopters may choose in developing the distributed SDN controller. Our analysis shows that each design choice influences the fundamental issues such as scalability, failure, consistency and privacy. We can take several key points from our analysis.

First, there is no perfect design choice. Each design has its characteristic. They may excel in solving one issue but suffer from the others. For instance, the flat model is excellent in controlling failure problems but has several disadvantages in scalability, privacy, and consistency. Second, we can also sum and combine pros and cons from each design to construct a guideline in designing a future distributed SDN controller. When one wants to build a *most scalable* distributed controller, then consider building with a combination of Master/Slave, hierarchical, leader-based and out-of-band design. Similarly, one can also generate other combinations such as the *most robust* controller with Master/Slave, flat and leaderless combination. The SDN controller with the *best privacy* is the combination of the hierarchical and out-of-band. Lastly, the controller with the *best consistency* is the combination of IP alias, hierarchical, leader-based and in-band design.

Several related works are capable of discovering and solving some fundamental issues of distributed SDN controller. By combining those works with our previous analysis, one can set up a desired distributed SDN controller with a custom trade-off among scalability, robustness, consistency and privacy.

Acknowledgment

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant Number: 2014R1A1A2060021).

References

[1] ONF, Software-Defined Networking: The New Norm for Networks. [White Paper], 2012. July 27, 2015, Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.
[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2008) 69–74.
[3] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: scaling flow management for high-performance networks, in: ACM SIGCOMM Comput. Commun. Rev. 41, 2011, pp. 254–265.
[4] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, ACM SIGCOMM Comput. Commun. Rev. 40 (2010) 351.
[5] ONF, 2011. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf.
[6] P. Lin, J. Bi, Y. Wang, East-west bridge for sdn network peering, Front. Internet Technol. (2013) 170–181.
[7] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 2012, pp. 19–24.
[8] M.A. Santos, B.A. Nunes, K. Obraczka, T. Turletti, B.T. de Oliveira, C.B. Margi, Decentralizing SDN's control plane, in: IEEE 39th Conference on Local Computer Networks (LCN), 2014, 2014, pp. 402–405.
[9] V. Yazici, M.O. Sunay, and A.O. Ercan, "Controlling a software-defined network via distributed controllers," *arXiv preprint arXiv:1401.7651*, 2014.
[10] ONF. (2011, September 8). *OpenFlow Switch Specification Version 1.2*. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf.
[11] ONF. (2014, September 8). *OpenFlow Management and Configuration Protocol version 1.2*. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.

[12] H. Xie, T. Tsou, D. Lopez, H. Yin, and V. Gurbani, "Use cases for alto with software defined networks," *Working Draft, IETF Secretariat, Internet-Draft draft-xie-alto-sdn-extension-use-cases-01. txt*, 2012.
[13] B. Lee, S.H. Park, J. Shin, S. Yang, IRIS: the Openflow-based recursive SDN controller, in: Advanced Communication Technology (ICACT), 2014 16th International Conference on, 2014, pp. 1227–1231.
[14] Open Network Operating System (ONOS) (September 8). *ONS Developer Track*. Available: http://onosproject.org/news-events/events/.
[15] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: Proc. USENIX Annual Technical Conference, 2014, pp. 305–320.
[16] L. Lamport, Paxos made simple, ACM Sigact News 32 (2001) 18–25.
[17] A. Tootoonchian, Y. Ganjali, HyperFlow: a distributed control plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, 2010 https://www.usenix.org/legacy/event/inmwren10/tech/.
[18] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 7–12.
[19] J. Hu, C. Lin, X. Li, J. Huang, Scalability of control planes for Software defined networks: Modeling and evaluation, in: Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of, 2014, pp. 147–152.
[20] N. Katta, H. Zhang, M. Freedman, J. Rexford, Ravana: controller fault-tolerance in software-defined networking, Presented at the SOSR 2015, 2015.
[21] D. Marconett, S. Yoo, FlowBroker: a software-defined network controller architecture for multi-domain brokering and reputation, J. Netw. Syst. Manage. 23 (2014) 328–359.
[22] A. Akella, A. Krishnamurthy, A highly available software defined fabric, in: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, 2014, p. 21.
[23] A. Panda, C. Scott, A. Ghodsi, T. Koponen, S. Shenker, Cap for networks, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, HongKong, 2013, pp. 91–96.
[24] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, et al., Onix: a distributed control platform for large-scale production networks, in: OSDI, 2010, pp. 1–6.
[25] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, et al., NOX: towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2008) 105.
[26] K. Phemius, M. Bouet, J. Leguay, Disco: Distributed multi-domain sdn controllers, in: Network Operations and Management Symposium (NOMS), 2014, IEEE, 2014, pp. 1–4.
[27] Big Switch Networks (July 27). *Floodlight*. Available: http://www.projectfloodlight.org/floodlight/.
[28] OpenDaylight (ODL) (July 30). *OpenDaylight*. Available: https://www.opendaylight.org/.
[29] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, et al., ONOS: towards an open, distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, 2014, pp. 1–6.
[30] D. Erickson, The beacon openflow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HongKong, 2013, pp. 13–18.
[31] Ryu (July 27). *Ryu*. Available: http://osrg.github.io/ryu/.
[32] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 2012, pp. 7–12.
[33] C. Cervello-Pastor, A.J. Garcia, On the controller placement for designing a distributed SDN control layer, in: Networking Conference, 2014, IFIP, 2014, pp. 1–9.
[34] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 2012, pp. 1–6.
[35] E.A. Brewer, Towards robust distributed systems, PODC, 2000.
[36] S. Gilbert, N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News 33 (2002) 51–59.
[37] D. Featherston, Cassandra: principles and application, International Conference on Computing, Engineering and Information, University of Illinois at Urbana-Champaign, 2010.
[38] MongoDB (September 1). *MongoDB*. Available: https://www.mongodb.org/.
[39] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, et al., Dynamo: amazon's highly available key-value store, ACM SIGOPS Operating Syst. Rev. 41 (6) (2007) 205–220.
[40] P. Hunt, M. Konar, F.P. Junqueira, B. Reed, ZooKeeper: wait-free coordination for internet-scale systems, in: USENIX Annual Technical Conference, 2010, p. 9.
[41] M. Burrows, The Chubby lock service for loosely-coupled distributed systems, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006, pp. 335–350.
[42] J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J.J. Furman, et al., Spanner: Google's globally distributed database, ACM Trans. Comput. Syst. (TOCS) 31 (2013) 8.

Yustus Eko Oktian is a Master student at Dongseo University, Busan, Korea. He received his bachelor degree in Electrical Engineering from Petra Christian University in 2013, Surabaya, Indonesia. His research interests are on the topics of Software Defined Networking, network security, and web-based application.

Sang-Gon Lee received his B.Eng., M.Eng., and Ph.D. degrees in electronics engineering from Kyungpook National University, Rep of Korea, in 1986, 1988, and 1993, respectively. He is a professor in the Division of Computer Engineering, Dongseo University, Busan, Republic of Korea. He was a visiting scholar at QUT, Australia, from August 2003 to July 2004 and at the University of Alabama at Huntsville, USA, from July 2012 to Jun 2013. His research areas include information security, network security, wireless mesh/sensor networks, and the future Internet.

Hoon-Jae Lee received the B.S., M.S. and Ph.D. degree in Electrical Engineering from Kyungpook national university in 1985, 1987 and 1998, respectively. He had been engaged in the research on cryptography and network security at Agency for Defense Development from 1987 to 1998. Since 2002 he has been working for Department of Computer Engineering of Dongseo University as an associate professor, and now he is a full professor. His current research interests are in security communication system, side-channel attack, USN & RFID security. He is a member of the Korea institute of Information security and cryptology, IEEE Computer Society, IEEE Information Theory Society and etc.

JunHuy Lam was born in Kuala Lumpur, Malaysia, in 1986. He received the B.E. degree in computer engineering from the Multimedia University, Cyberjaya, Malaysia, in 2010, and the M.Sc. degree in ubiquitous IT from Dongseo University, Busan, South Korea, in 2012. In 2012, he joined GHL Systems Berhad, Kuala Lumpur, Malaysia as Software Engineer, and involved in the software development of the credit card and contact-less card payment systems. Since 2014, he joined the Department of Ubiquitous IT, Dongseo University as a Ph.D. candidate. His current research interests include software defined network, algorithm, and network security.