Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 00 (2015) 1–15
Computer Communi-cations

Control Plane of Software Defined Networks: A Survey

Junjie Xie[a], Deke Guo[a,*], Zhiyao Hu[a], Ting Qu[a], Pin Lv[b]

[a]*Science and Technology on Information Systems Engineering Laboratory*
National University of Defense Technology, Changsha Hunan 410073, China
[b]*School of Computer, Electronics and Information, Guangxi University*
Nanning, Guangxi 530003, China

Abstract

Software Defined Networking (SDN) has been proposed to solve ossifications of Internet. The main motivation of SDN is to separate the control plane and data plane, enabling a centralized control. In this way, the network infrastructure becomes an open and standardized resource. Hence, it can be managed and utilized in a more efficient way. The controller is the key infrastructure in the SDN and provides programming interfaces to the entire network. Then, various applications can be written to perform management tasks and offer new functionalities on the controller. In this survey, we present many essential research issues about the controller, and especially focus on the control architecture, performance, scalability, placement, interface and security. The aim of this paper is to provide an up-to-date view to the SDN controller.

Keywords:
Software-defined Networks, Controller, Control plane, Survey

1. Introduction

The Internet has become extremely difficult to develop both in terms of its physical infrastructure as well as its protocols and performance. Moreover, as current and emerging Internet applications and services become increasingly more complex and demanding, it is imperative that the Internet be able to evolve to meet these new challenges. Additionally, computer networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes, and many complex protocols implemented on them. Network operators have to manually transform high-level policies into low-level configuration commands with access to very limited tools while adapting to changing network conditions. Moreover, network devices are usually vertically integrated black boxes [1]. As a result, network management and performance tuning are quite challenging and thus error-prone. To solve these Internet ossifications, Software Defined Networking (SDN) has been proposed and achieves substantial attentions from both academia and industry.

The main advantage of SDN is the separation of control plane and data plane, which enables the centralized control. SDN aims to dramatically simplify the network management and enable the innovations through the programmability of networks. In SDN, the network management is logically centralized in the control plane consisting of one or multiple controllers, which host many control applications. Network devices in the data plane just perform packet forwarding and other advanced packet processing functions. Those network devices can be programmed by

*guodeke@gmail.com

applications via some open northbound and southbound interfaces, e.g., OpenFlow [2]. Actually, the SDN originates from the programmable network and the decoupled control logic [1].

The Open Signaling (OPENSIG) working group [3] has dedicated to make ATM, Internet and mobile networks more open, extensible and programmable since 1995. They note that the separation of control software from the communication hardware is necessary, but challenging to be realized. The basic idea behind such proposals is to access those network hardware via open and programmable interfaces. In the mid 1990s, the Active Networks [4] initiatively proposed that the network infrastructure should be programmable for customizing network services. One approach is to develop the user-programmable switches, each of which possesses the inband data transfer and out-of-band management channels. Another approach is called capsules, which refers to program fragments. They can be carried in users' messages and then be interpreted and executed by network devices. However, the active network is not widely deployed in practice, due to security, performance, and other practical issues [5]. In 2004, the 4D project [6] advocated a clean-slate redesign of the control and management architecture, which emphasizes the separation between the routing decision and the protocols governing the interaction between network elements. The 4D project is the first one to provide network control mechanism with a global view. It is generally believed that the 4D project is the beginning of the SDN.

The SDN includes the data plane and the control plane, where the controller is the essential component to improve the control plane. Because the controller provides the programmatic interfaces to the entire network, many applications can perform management tasks and offer new functionalities on the controller. The switches in the data plane only forward received flows, according to given rules derived from the controller. The controller is responsible to maintain the global viewpoint of the whole network and imposes control constraints on each flow by running a set of user-defined control applications. If the controller fails or becomes the performance bottleneck, the network will lose the advantages of SDN. Currently, many efforts have been done on the architecture of the SDN controller, such as NOX [7], Maestro [8], Beacon [9], etc. However, many challenging issues about the SDN controller have not been well addressed.

In this paper, we survey the up-to-date research issues about the SDN controller, so as to plot the mainstream and emerging area of the SDN controller. Currently, we consider that research issues involved in the controller mainly fall into the following aspects.

- To efficiently manage and operate the network, user-defined applications are designed and deployed at the controller.
- The architecture of controller heavily affects the performance of SDN. Many different architectures have been proposed recently, such as the multi-core controller, the logically centralized controller and the completely distributed controller.
- The single controller exhibits the limitations of performance and scalability. Meanwhile, the placement problem of distributed controllers also affects the network performance.
- The mainstream interfaces associated with the controller are essential components to connect the users and network devices so as to realize the SDN.
- The controller is the core of SDN. If it is attacked and become undependable, the entire network would be destroyed.

Section 2 introduces the framework of software-defined networks. Section 3 surveys the category of network architecture. Section 4 covers the performance and scalability issues of SDN controller. Section 5 surveys mainstream interfaces of the SDN controller. Section 6 focuses on the security issue of the SDN controller. Finally, Section 7 discusses some potential research directions of the SDN controller.

2. Framework of software-defined networks

As shown in Figure 1, SDN includes three layers, i.e., the infrastructure layer, the control layer and the application layer. We can see that the controller layer manages the underlying physical network through the southbound API. The most notable is that OpenFlow [2] supported by the Open Network Foundation (ONF) is the mainstream southbound

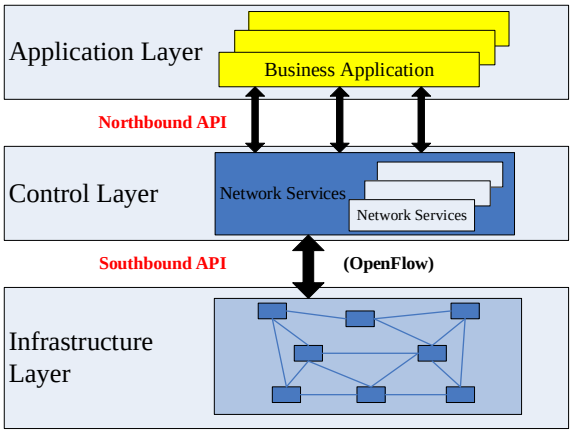

Figure 1. Three-layer framework of software defined networks.

API. OpenFlow shows the core idea of SDN that the control plane separates from the data plane. However, OpenFlow is not purely equal to SDN, and there are also some other southbound API, such as I2RS [10] and OpFlex [11]. Meanwhile, the controller layer naturally supports the application layer, where many applications are deployed at this layer, via the northbound API. That is, the controller layer acts as the core component of SDN. Additionally, all controllers at the control layer require an east-west interface as a bridge to implement the synchronization and negotiation functions.

The infrastructure layer consists of SDN switches shown in Figure 1. When a new flow reaches a SDN switch, for that flow, the SDN switch will send a route request to the controller. The controller calculates a routing path for that flow on the basis of the global view, and then delivers the forwarding rule of that routing path to all involved switches through a secure channel. When those SDN switches receive the forwarding rules, they will update their flow tables. They then forward the received flow according to the corresponding flow rules derived from the controller, and this is a reactive manner. Meanwhile, the SDN can also work with proactive flows. For example, DIFANE [12] adds proactivity to the control policies and distributes rules to authority switches. These authority switches store the mandatory rules and can directly forward packets without the controller. DIFANE can be easily implemented with today's flow-based switches. Additionally, if we want to schedule the flow for different purposes, we can develop relevant applications supported by the controller through the northbound API. There are various applications running in the controller to manage and operate the whole network.

Due to the inherent advantages, the SDN has been introduced to many networks, such as the Internet, datacenter networks, and enterprise networks. Although it traditionally motivates to address the complex routing problem in the flow control in networks, many other applications can be easily implemented in SDN, such as firewalls [13], load balance [14], access control [15], NAT, etc.

Moreover, Google had several years' experience in operating Wide Area Network (WAN) across its data centers. The utilization of such WAN is only 30 – 40% on average. To address the issue, the project B4 [16] enhances the WAN utilization to near 100% by using the SDN principle and the OpenFlow protocol. More precisely, B4 introduced the SDN-based traffic engineering into Google's WAN across data centers. Consequently, it can dynamically allocate the inter-datacenter bandwidth among traffics and fully exploit the network capacity.

Additionally, many efforts have been done on the software-defined wireless network. Some applications, such as Odin [17], have been realized at the SDN controller. The traditionally enterprise wireless local area network (WLAN) can be strengthened as a software-defined WLAN after the introduction of Odin. Consequently, it can provide a wide range of functionalities and services. Actually, there are increasing number of new applications realized and deployed in the software-defined WLAN.

In summary, SDN controller achieves and maintains a global view of the whole network, through which users can develop more applications to improve the network performance and resources utilization. Because the controller undertakes massive compute and storage tasks, it may become the bottleneck of the entire network. Meanwhile, it is well known that the control architecture will affect the performance and scalability of controller. To improve the control architecture, many efforts have been done, which will be introduced in the next section.

3. Network architecture category

Start from the 4D project [6], the centralized control architecture is proposed for software-defined networks to enable continuous innovations in the network control and management. At the beginning, Ethane [18] adopts a single controller to manage the entire enterprise network. Ethane reports that a single controller could manage over 10,000 machines. It, however, may be restricted within some Internet topologies. Due to the capacity limitation of a single controller and the large amount of flows, one controller is insufficient to control the entire network. To improve the scalability and performance, some novel control architectures are proposed, such as Maestro [8], Onix [19] and Kandoo [20]. Such architectures fall into two categories, the single-control plane and the multiple-control plane. For the first category, some researchers believe that one controller is enough and the problem is how to enhance its performance. This kind of control plane is referred to as the single-control plane. Maestro [8], for example, is a preventative single-control plane. For the second category, multiple controllers are used to cooperatively manage the network, such as HyperFlow [21] and DISCO [22].

Currently, the multiple-control plane has two different implementation methods. First, these controllers synchronize their local views about the network with each other. Consequently, all controllers maintain a global and consistent network view for making an optimal decision. Such control planes are called the logically centralized but physically distributed control planes, such as ONOS [23]. On the contrary, it is not necessary for each controller to pursue the global network view. That is, a local view of the whole network is usually sufficient to achieve given functions of software-defined networks. Each controller makes decisions only using its local view. The kind of control plane is referred to as the completely distributed control plane. Moreover, the hierarchical control plane is a special case of the completely distributed control plane, such as Kandoo [20]. In the remainder of this section, we will discuss such control planes in detail. Before the discussion, we firstly give a brief introduction to mainstream controllers as Table 1.

Table 1. A brief comparison of mainstream controllers.

Controller	Category	Southbound Interface	Platform	Development Team
Beacon	single controller	OpenFlow	Win/Mac/Linux	Stanford
McNettle	single controller	OpenFlow	Linux	Yale University
Onix	logically centralized	OpenFlow	Linux	Nicira
DISCO	logically centralized	OpenFlow	Win/Mac/Linux	Thales
ONOS	logically centralized	OpenFlow/Others	Win/Mac/Linux	ON.Lab
OpenDaylight	logically centralized	OpenFlow/Others	Win/Mac/Linux	Linux Foundation
OpenContrail	logically centralized	OpenFlow/Others	Linux	Juniper

3.1. The multi-core controller

It is true that large amount of flows are injected into various networks. If the principle of SDN is introduced into such networks, one basic issue is how the single controller can generate the optimal routing solutions for such flows in time. If the controller does not have sufficient capacity to handle such flow requests, it will become a bottleneck of the entire network. For this reason, some proposals are presented to enhance the capacity of the single controller, such as Beacon [9] and McNettle [24].

Beacon [9] provides a framework for controlling network devices using the OpenFlow protocol, and designes a set of built-in applications for enabling various control functions. Beacon achieves surprisingly high performance via utilizing multi-core. Moreover, the performance is able to scale linearly with the number of processing cores. For example, it can handle 12.8 million flow requests per second with 12 cores.

The simplicity of a logical centralized controller, however, comes at the cost of limited scalability in the control plane. To address this problem, McNettle [24] is designed as an extensible control system. It can easily extend the controller to integrate a multi-core processor by writing handlers and background programs in a high-level programming language. Using a single controller with 46 cores, McNettle [24] can serve up to 5000 switches while achieving the throughput of over 14 million flows per second.

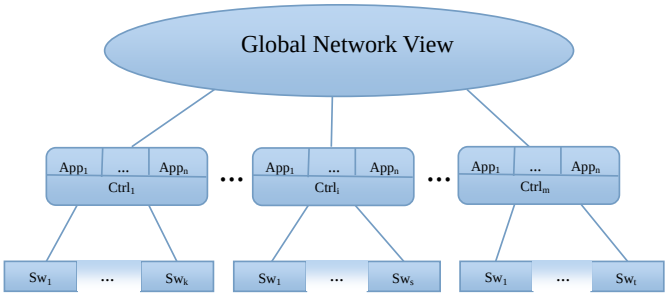

Figure 2. An example of the logically centralized but physically distributed control architecture.

Maestro [8] provides a simple single-thread programming model for application programmers. Moreover, it can support and manage the parallelism. It utilizes some optimization techniques of parallelism to expand the throughput of a controller. Maestro can be successful to deal with 600 thousand flow requests per second using an eight-core controller.

Although the aforementioned methods can improve the capacity of a single controller, one controller is indeed insufficient in practical networks, such as the Internet and large-scale data centers. For software-defined networks, the controller needs to calculate the routing path of each new flow. Moreover, the waiting time from sending a flow request to allocating its routing path by the controller should be not too long. In the case of the Internet, due to the broad geographical distribution, a single controller is not suitable no matter where it is deployed. Here, remote transmission of the flow request consumes additional time. In data centers, the limited capacity of one controller cannot adapt to the increasing scale of inter-network. Therefore, the introduction of multiple controllers is an inevitable and reasonable solution.

3.2. The logically centralized controller

As aforementioned, the SDN starts with an omniscient controller, such as NOX [7], to manage the entire network. Although multi-core controller exhibits superior performance than traditional single controller, it is confronted with many obstacles, such as the single point of failure and limited scalability. To address these issues, some proposals about distributed controllers are proposed recently. Meanwhile, such controllers have to share information with each other to build a consistent view of the entire network. This type of control plane is referred to as the logically centralized and physically distributed control plane, such as ONOS [23] and OpenContrail [25]. Meanwhile, Onix [19] and HyperFlow [21], are also two representatives of this kind of control plane.

An example of the logically centralized but physically distributed control plane is demonstrated in Figure 2. We can see that these controllers can share a network-wide view from the Figure 2. To maintain a global view of the entire network and to make a global optimal decision, these physically distributed controllers must synchronize its states with others. When the local view of one controller changes, the controller will synchronize the updated information to other controllers. In conclusion, for a logically centralized control plane, controllers need to exchange information with each other. Some researchers focus on the dedicated methods of information exchange or state synchronization. Consider that, the information exchange among controllers consumes many network resources. It is critical to reduce the resultant network overload, while keeping the information consistent for the logically centralized control plane.

In traditional networks, every function, such as routing, must build their own state distribution, element discovery, and failure discovery mechanism. Due to the lack of a common control plane, the development of flexible, reliable and feature-rich network control planes has been significantly hindered. To solve the problem, Onix [19] is proposed as a distributed system. Onix focuses on the problem of providing generic distributed state management APIs, which enable the programmer to program their control logic. Based on the Onix, the control plane operates on a global view of the network. Therefore, it is a logically centralized but physically distributed control plane. Additionally, it is also the platform to manage large-scale data center networks, such as SEATTLE, VL2 and Portland.

HyperFlow [21] provides a logical centralized control, which consists of many distributed controllers and exhibits good scalability. It has been implemented as an application for NOX [21]. In reality, network operators can deploy any number of controllers on demand. Through propagating events that affect the controller's state, HyperFlow can enable all controllers to achieve the network-wide view by passively synchronizing network-wide views of OpenFlow

controllers. Since each controller has the global view, HyperFlow minimizes the response time of the control plane through local decision making at each individual controller.

DISCO [22] is another open and extensible SDN control plane, which copes with the distributed and heterogeneous characteristics of wide area networks and modern overlay networks. Each DISCO controller manages its own network domain and communicates with other controllers, through a lightweight control channel, to ensure those end-to-end network services. Consequently, each DISCO controller can construct a logically centralized control plane. The control plane also provides some classic functionalities, such as the traffic engineering and end-point migration. Moreover, the control plane can be adapted to dynamic network topologies and is resilient to attacks and disruptions.

Since the centralized controller has not provided the required levels of availability and responsiveness, to solve the problem, Canini et al. have studied a distributed and robust control plane, which enables concurrent and robust policy implementation [26]. They introduced a formal model of SDN under fault-prone, concurrent control. Then they formulated the problem of consistent composition of concurrent network policy updates and discussed different protocols to solve the consistent policy composition problem.

Currently, there are two famous SDN control plane architectures, ONOS [27] and OpenDaylight [28], with multiple contributors. Open Network Operating System (ONOS) [23] provides a global network view to applications, which is logically centralized even though it is physically distributed across multiple servers. It adopts a distributed architecture for high availability and scale-out. Meanwhile, ONOS abstracts device characteristics so that the core operating system does not have to be aware of the particular protocol being used to control a device. ONOS follows in the footsteps of previous closed source distributed SDN controllers such as Onix [19], but ONOS has been released as an open source project which the SDN community can examine, evaluate, extend and contribute to as desired [27].

OpenDaylight [28] allows for the network to be logically (and/or physically) split into different slices or tenants with parts of the controller, modules, explicitly dedicated to one or a subset of these slices. This includes allowing the controller to present different views of the controller depending on which slice the caller is from. Meanwhile, Opendaylight builds consistent clustering that gives fine-grained redundancy and scale out while insuring network consistency. However, there lack introductions of multiple controller instances in the official website [28]. It is noteworthy that this controller is implemented strictly in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

Additionally, the OpenContrail Controller [25] is a logically centralized but physically distributed SDN controller that is responsible for providing the management, control, and analytics functions of the virtualized network. OpenContrail is a network virtualization platform for the cloud. The OpenContrail system consists of two main components: the OpenContrail Controller and the OpenContrail vRouter. Meanwhile, OpenContrail is an extensible system that can be used for multiple networking use cases but there are two primary drivers of the architecture: cloud networking and network function virtualization (NFV) in Service Provider Network.

When the mapping between a switch and a controller is statically configured, there will exhibit uneven load distribution among the controllers. To solve the problem, Dixit et al. [29] have proposed the ElastiCon, an elastic distributed controller architecture, which can dynamically increase or decrease the number of controllers according to the change of traffics. After deploying the ElastiCon, the load of controllers can be shifted across controllers. The authors also introduce a novel switch migration protocol for enabling such load shifting, which conforms to the OpenFlow standard.

3.3. The completely distributed controller

The introduction of SDN brings network designers freedom to refactor the network control plane. One core benefit of SDN is the centralized control plane, where the controller is responsible to control flows and manage network resources based on the global view of the entire network. However, to maintain a global view, controllers have to synchronize states with each other. As the state of the whole network frequently changes, synchronization may lead to the network overload. Moreover, Levin et al. have found that the inconsistent control state of the SDN significantly degrades the performance of many applications [30]. Therefore, control plane state and logic must inevitably be physically distributed. It is also understood that fully physically centralized control is inadequate, because it limits the reliability and scalability of the control plane. Some proposals begin to construct the local network view. Levin et al. also demonstrate that a controller can derive a non-optimal but good flow path, even though all controllers do not exchange their local network views with each other [30]. Tam et al. [31] and Schmid et al. [32] have also done some efforts in this field.

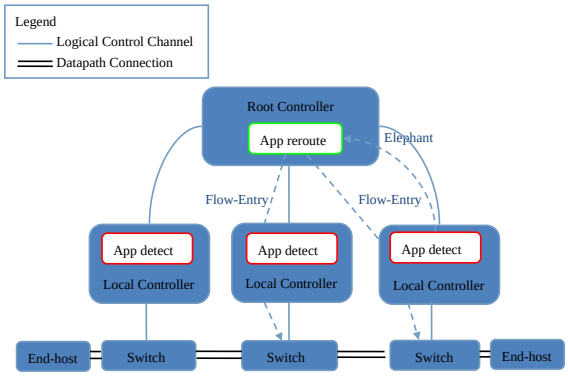

Figure 3. An example of the hierarchical control architecture.

To resolve the scalability problem of the centralized control plane in large-scale data centers, Tam et al. [31] have proposed to utilize multiple independent controllers, each of which only controls a subnet of the entire network. Such controllers can be treated as a single centralized controller. However, none of them achieves the complete information of the data center network. Moreover, given a request, it can ensure that at least one controller can response to it at any time. An inevitable drawback for this method is the case that each controller cannot find the optimal solution for each flow.

For the distributed control plane, each controller manages its local domain. Schmid et al. [32] aims to achieve the better routing decision at each controller via some improved local algorithms. Existing local algorithms can be utilized to develop efficient coordination protocols, through which each controller only cooperates with its neighboring controllers. Although the existing distributed algorithms can be used, there are also demands for the dedicated distributed algorithm for software-defined networks.

Kandoo [20] is another representative design of control plane, which has a hierarchical architecture. Kandoo employs two layers of controllers, as shown in Figure 3. The observations indicate that the size of flow in the data plane is different. The flow resulting in large volume of data transmission is called the elephant flow. Such flows can heavily affect the load of underlying networks. However, the number of the elephant flow is few. When Kandoo detects an elephant flow, the root controller will calculate an optimal path for the elephant flow, according to the network-wide view. By contrast, there are large number of small flows. Those controllers at the bottom layer will deal with those small flows by their local views. Consequently, these bottom controllers can handle most of frequent events and effectively reduce the load of the root controller. Kandoo enables network operators to deploy local controllers on demand and relieve the load of controller at the top layer, which is the only potential bottleneck in terms of scalability.

4. The performance and scalability of the controller

4.1. The performance of the controller

Given a controller, its performance means that how many flow requests the controller can handle per second and how fast the controller can respond to each flow request. NOX, a popular network controller, can handle around 30k flow requests per second while maintaining a sub-10ms flow install time [33]. To improve the performance of NOX, Tootoonchian et al. have added some well-known optimization techniques to the NOX controller, such as I/O batching and multi-threaded successor [34]. Such optimization strategies can significantly improve the performance of NOX. For example, on an eight-core machine with 2GHz CPUs, NOX can handle 1.6 million requests per second with an average response time of 2ms.

Although there are numerous efforts to enhance the performance of the controller, it still cannot meet the high network demands, such as the eruptive flow requests and the low respond delay. Recent measurements of some production environments suggest that the performance of a single controller is still far from sufficient. For example, Kandula et al. find that a cluster of 1500 servers receives 100k flows per second on average [35]. Benson et al. [9] report that a network with 100 switches can result in 10M flow arrivals per second in the worst case. In addition, the 10ms flow setup delay, resulting from a SDN controller, would add 10% delay to the large number of short-lived flows

in such a network. Therefore, the single controller cannot satisfy the network demand in terms of both capacity and response time.

The gap between limited performance of controller and heavy demand of network flows has motivated researchers to address perceived architectural inefficiencies (e.g., [12] [36]). Owing to enabling flow-level control, OpenFlow can simplify network and traffic management in enterprise and data center networks. However, OpenFlow's current design cannot meet the needs of high-performance networks. Consider that, Curtis et al. design DevoFlow [36], a modification of the OpenFlow. DevoFlow breaks the coupling between control and global visibility and just maintains a useful amount of visibility. Therefore, DevoFlow can handle most microflows in the data plane. Meanwhile, DevoFlow uses 10−53 times fewer flow table entries at each switch on average, and results in 10−42 times fewer control messages.

Meanwhile, Yu et al. demonstrate that a flow that goes through a long path needs less time than going through the controller [12]. It means that the time from a SDN switch sends a flow request to the SDN switch gets the forwarding rule is longer than the flow is directly forwarded in the data plane, although the route for the flow may not be the optimal route. Therefore, to speed up the flow of processing and reduce the load of controllers, they propose the DIFANE [12] that add the proactivity to SDN. DIFANE relegates the controller to the simpler task of partitioning these rules over the switches. By storing necessary rules into intermediate switches in advance, DIFANE can keep all traffic in the data plane. Moreover, DIFANE can handle wildcard rules efficiently and react quickly to network dynamics such as policy changes, topology changes and host mobility.

4.2. The scalability of the controller

The scalability challenge of control plane in the SDN is inherently similar to those issues in traditional networks. Yeganeh et al. think that SDN should either eliminate the design complexity of the control plane or make it more scalable [37]. To address the scalability of a single controller, the major solution is the improvement of the controller's performance with increasing the number of CPU cores linearly. McNettle [24] is a highly scalable SDN control framework, which executes on shared-memory multi-core servers and provides a simple programming model for controller developers. OpenDaylight [28] improves the scalability of control plane by supporting the OSGi (Open Service Gateway Initiative) framework. The OSGi framework is used for applications that run in the same address space as the controller. The business logic and algorithms reside in the applications. Owing to more applications, the OpenDaylight enhances the scalability of control plane. Although such proposals can improve the controller' performance at some extent, they are still not able to accelerate the response to flow requests.

The aforementioned logically centralized control plane usually consists of lots of distributed controllers. This not only improves the number of flow requests it can handle per second but also reduces the response time to each flow request. There are many efforts to design the logically centralized but physically distributed controllers, such as [19] [21] [29]. Achieving the distributed core, the network operator can add servers incrementally to ONOS, without disruption, as needed for additional control plane capacity. The ONOS instances work together to create what appears to the rest of the network and applications as a single platform [27]. Applications and network devices do not have to know if they are working with a single instance or with multiple instances of ONOS. This feature makes ONOS scalable that one can scale ONOS capacity seamlessly.

Meanwhile, distributed controllers need to maintain a global consistent network view to achieve the logically centralized control plane. The controllers share information through the state synchronization mechanism. Therefore, for improving the scalability, the essential work aims to reduce the overload of state synchronization and keep the information consistent among controllers. Existing state synchronization schemes for multiple controllers are based on periodic synchronization. Note that the state synchronization mechanism among controllers often causes undesirable situations, such as the forwarding-loop problem. To address these issues, a Load Variance-based Synchronization (LVS) mechanism is proposed in [38]. LVS-based schemes conduct effective state synchronization among controllers only when the load of a specific server or domain exceeds a certain threshold. Therefore, the LVS can effectively reduce the synchronization overhead among controllers and eliminate the forwarding-loop problem.

Although achieving the global network view incurs some challenging issues, they can be handled by related local algorithms [32] in distributed control plane. Moreover, each controller only needs to communicate with their local neighbors, within a given number of hops from it. Accordingly, we can reduce the load of controllers and realize the load balance among them. Due to such efficient mechanisms proposed in [32], the scalability of the logically centralized controller can be considerably improved.

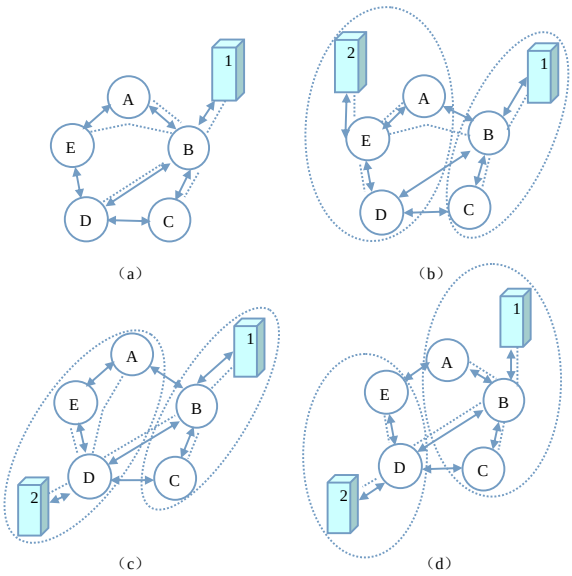

Figure 4. Options about controller placement in a five-node SDN.

4.3. The placement problem of controllers

The distributed control plane leads to an open problem, i.e., how many controllers are required. The number of controllers and their locations have direct impact on the performance of a software-defined network. Heller et al. [39] has formally characterize the placement problem of controllers in the wide-area networks. The delay between controllers or controller-switch will lead to long response time, and then influence their ability to respond to network events. The authors consider the optimal controller placement problem, which tries to minimize the propagation delay between controllers or controller-switch in the WAN. Meanwhile, they report that the latency from every node to a single controller can meet the response-time goals of existing technologies in many medium-size networks. However, with the expansion of the network, only one controller is not enough to manage the whole network. Therefore, multiple controllers are used cooperatively to control the whole network. This paper leaves many open problems to be solved in the future.

Consider that commercial controllers are scalable on the basis of capacity, many proposed control planes do not consider the scalability. Jimenez et al. [40] define the principles for designing a scalable control plane from the aspect of the controller placement problem. They use an algorithm called k-Critica to find the minimum number of controllers as well as their locations to build a robust control network topology, which disposes failures and balances the loads among lots of designated controllers.

Additionally, the authors in [41] find that the number of controllers and their locations can affect the reliability of SDNs. Figure 4 uses an example to demonstrate the effect of deploying one or two controllers in a software-defined network consisting of 5 switches. Those solid lines represent real physical links and the dotted lines denote the shortest paths between a pair of switches or from a switch to a controller. Every controller just controls several switches and each switch is controlled by one controller.

It is clear that one controller will cause the single-point-of-failure and Figures 4(b), 4(c), 4(d) are more reliable than Figure 4(a). As shown in Figure 4(b), if the physical link between switches A and E breaks, the communication path between switch A and its controller and the links between two controllers will break down. However, if the same fault happens in Figure 4(c), the communication path between two controllers will not be affected. Consequently, Figure 4(c) is more reliable than Figure 4(b). Figure 4(c) and Figure 4(b) demonstrate that the location of controllers will affect the reliability of SDN. Moreover, we find that Figure 4(d) is more reliable than Figure 4(c) due to the following reason. When the link between switches D and E fails, only switch E in Figure 4(d) would not communicate with its controller, i.e., Controller 2. However, both switches E and A would not communicate with the Controller 2 in Figure 4(c). It means that the control range of each controller also influences the reliability of SDN. Hu et al. [41] propose a metric, called the expected percentage of the control path loss, to measure the influence of the controller placement

on the reliability of SDN. Finally, the authors utilize the simulated annealing algorithm to optimize the locations of controllers.

However, the aforementioned methods focus on static networks and would not support dynamic networks. Bari et al. [42] propose a framework for dynamically deploying multiple controllers in WAN. According to the network condition, the number of required active and inactive controllers and their locations could be identified. The evaluation results show that this method could effectively decrease the set up time of a flow.

5. The major interfaces of controller

The major interfaces of controller are essential enabling components of a software-defined network. Firstly, the controller is required to provide the southbound interface to manage the underlying network. Secondly, the northbound interface is vital to directly support various networking applications, which have dedicated demands on the underlying network. Such interfaces also can simplify the design and implementation of innovative networking applications and services. Lastly, for the distributed controllers, the east-west bridge is needed to enable efficient communication among controllers and improve the reliability of control plane.

5.1. The southbound interface

As mentioned above, the separation between the control plane and the data plane in SDN results into the network programming, which enables network managers to take charge of the entire network. Moreover, many network-wide applications, such as monitoring data flow or balancing switch load, is required to program the underlying network. Thus, the setting up of switches and optimizing the network management all need a southbound interface for establishing the channel between controllers and switches.

OpenFlow is a widely used southbound interface and protocol[2]. It requires that each Ethernet switch is equipped with an internal flow-table. OpenFlow protocol establishes a secure channel between controllers and switches. Through this channel, each switch can forward a flow request to a controller, and the controller delivers the generated rules for that request to involved switches. After receiving control rule from controller, each switch updates its flow table. Additionally, OpenFlow allows researchers to run experiments on heterogeneous switches. Meanwhile, vendors do not have to expose the internal details of switches and can add OpenFlow to their switch products.

NETCONF [43] is a managing protocol to modify the configuration of network devices. It allows network devices to expose an API, through which extensible configuration data could be transmitted and retrieved. Although the protocol simplifies the device's reconfiguration and acts as a building block, there is not any separation of the data and control plane in this protocol. Thus, a network with NETCONF is not fully programmable.

Meanwhile, ForCES [44] is another famous protocol for communication between the controller and forwarding components within a network element since 2003. Although ForCES shares some common goals with SDN, their initiatives are different in many aspects. For ForCES, the internal architecture of each network device is redefined such that the control element is separated from the forwarding element. The combination of them still exhibits as a single network element to the outside world. Other researchers also try to combine additional forwarding hardware with third-party control within a single network device. ForCES is a parallel approach to software-defined networking and is under development by the IETF Forwarding and Control Element Separation Working Group.

Additionally, a high level commanding network programming language is needed to simply and efficiently manage switches on the controller. Compared with other languages, it should have a simple-structured syntax for an imperative network programming and needs only a few types of statements to construct interface rules. Programs written by these commands can construct and install rules into switches. Finally, controllers can add rules to flow tables and manage the switches.

5.2. The northbound interface

The northbound interface can help the application developers to manage and program the network. However, the existing programming languages usually use the low level abstraction supplied by the underlying hardware; hence, they fail to provide support for modular programming. Moreover, the intuitive network policies in the SDN domains are inherently dynamic and stateful. Current configuration languages are also not expressive enough to capture these policies.

Frenetic [45] is a high-level language for programming network switches. Meanwhile, it provides a declarative query language for classifying and aggregating network traffic. It also delivers a functional library for describing high-level packet-forwarding policies. Meanwhile, Frenetic facilitates the modular reasoning and activates the code reuse. Such important properties are enabled by Frenetics novel runtime system, which manages all of the details related to installing, uninstalling, and querying low-level packet-processing rules on physical switches.

Pyretic [46] is a programming platform, which raises the abstraction level and enables the creation of modular software. Consequently, it allows programmers to create sophisticated SDN applications and develop dependent modules applications. Management polices are further expressed as abstract functions. In addition, multiple policies could be grouped together using one of several policy composition operators, such as the parallel composition and the sequential composition.

Additionally, Procera [47] is a SDN control architecture, which supports a declarative policy language based on the functional reactive programming. It is further extended to express those high-level network policies and temporal queries over event streams, which occur frequently in network policies. Meanwhile, we must ensure that those rules installed on one task should not affect other tasks. Monsanto et al. [48] recommend a set of new abstractions to develop an application with multiple independent modules to jointly manage the network. Furthermore, to simplify the SDN programming, Maple [49] utilizes a standard programming language to decide the behaviors of the entire network. At the same time, it also provides a programmer-defined, centralized policy, which runs on every packet entering a network. Hence, it is obliviously employed to translate a high-level policy into SDN rules on individual switches. Maple includes a highly-efficient multicore scheduler and a novel tracing runtime optimizer. The scheduler can efficiently scale to controllers with 40+ cores. The optimizer can automatically record reusable policy decisions and keep flow tables of the switches up-to-date.

There exists the resource competing problem when multiple controllers work together. To address this problem, the corybantic system [50] is designed, which consists of a coordinator at a typical SDN controller. The coordinator utilizes many independent modules, each of which manages different aspects of the network. The coordinator has the responsibility to deal with the conflicts among modules. Every module seeks to optimize one or more objective functions. The coordination among those modules can maximize the overall value resulting from the controllers' decisions.

Currently, software-defined networks still lack the standard of northbound interfaces. ONF has a North Bound Interface Working Group (NBI-WG) [51], which is dedicated to define and subsequently standardize various Northbound API Interfaces (NBIs) for SDN Controller. Firstly, the goal of this group is to provide extensible, stable, and portable NBI APIs to controllers, network services, and application developers. Secondly, they want to increase the portability of software designed to interact with SDN controllers. This will be done by defining multiple APIs at differing levels of abstraction to allow network behavior to be more programmable. The last but not least target is to ensure that controller vendors are free to innovate, using API extensions, within their own designs. Such efforts will accelerate the SDN innovation.

5.3. The east-west bridge

Actually, large-scale datacenter networks and enterprise networks are always partitioned into many sub-networks, each of which is controlled by a different controller. The Internet is also managed by owners of different domains. The fact limits the use of the centralized control across those domains, each of which usually deploys a controller. Each of such controllers should have the global network view to determine the next domain hop for those flows across domains. Hence, controllers are required to exchange reachability and topology information between the inter-domain networks. However, these controllers cannot communicate with each other directly without the help of given interfaces, as shown in Figure 5. To solve this problem, Lin et al. [52] propose a new network view exchange mechanism.

Lin et al. design a high-performance mechanism for heterogeneous SDN domains to exchange network view for enterprises, data centers, and intra-domain networks [52]. Meanwhile, considering the network privacy, they propose WBridge [52] to abstract the physical network to a virtual network view. They also evaluate several related solutions in different SDN domains. Moreover, Lin et al. show a west-east bridge mechanism for different SDN administrative domains to cooperate with each other [53]. They design a peer-to-peer exchange mechanism of network information.

Considering different SDN domains, what network information should be exchanged and how such information can be efficiently exchanged among inter-domain SDN peers are two essential problems. To achieve a resilient peer-

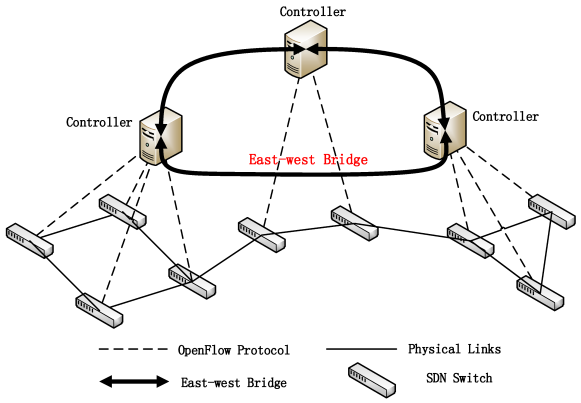

Figure 5. East-west bridge for different SDN domains.

to-peer control plane over heterogeneous SDN domains, Lin et al. propose a maximum connection degree based connection algorithm [53]. To address the privacy issue, they propose to virtualize the SDN network view, and only exchange the virtualized network view to construct the relative global network view.

As shown in Figure 5, the east-west bridge enables the exchange of individual network views among different controllers. Through the information exchange, each controller can have a global view of the entire network. After receiving the first packet of each new flow, the controller will compute an optimal routing path for that flow, according to the global view.

Additionally, ONOS [27] is deployed as a service on a cluster of servers, and the same ONOS software runs on each server. Each ONOS instance manages a subsection, and the state information local to the subsection is disseminated across the cluster as events. The events are generated in the store, and are shared with all of the nodes in a cluster via distributed mechanisms built into the various services' distributed stores. Using high speed messaging in a publish/subscribe model, ONOS instances can quickly inform other instances of updates. The distributed core of ONOS provides messaging, state management and leader election services to instances or between them. As a result, multiple instances behave as a single logical entity.

6. The security of controller

It is generally known that the networking programmability and the global control plane are two distinguished capabilities of software-defined networks. Such capabilities indeed bring some new security problems. There exists three security parts, the controllers, the communication channels among controllers, and the channels between the controllers and switches. Currently, less effort has been done to deal with the security problem of SDN. Moreover, the data integrity and confidentiality among controllers also lack sufficient attention. Kreutz et al. [54] developed a secure and dependable control platform, which still does not completely solve the security problem of SDN.

The security problem in enterprise networks differs from that on the Internet. SANE [55] provides a protection architecture for enterprises networks. It can manage the networks through the central control and authenticate all network elements to grant the enterprise security. Ethane [18] further improves the SANE. Using the centralized controller, Ethane manages the routing and admittance of flows and couples simple flow-based Ethernet switches. Ethane and SANE are designed to enable secure communication between the control plane and the data plane.

Moreover, a switch may suffer the inconsistency between a new OpenFlow rule and an existing rule. FortNOX [56] introduces a security kernel that can detect the potential rule conflicts. It can insert security rules into switches with different priorities. At the same time, to prevent faulty rules from the switch, VeriFlow [57] provides a safety layer between the controller and the switches to verify the network invariants. Moreover, it can complete the verification within hundreds of microseconds when new rules are generated. Veriflow is the first tool to check network-wide invariants in real time. They can also guarantee the communication security between the control plane and the data plane.

Furthermore, a data center usually needs a load balancer to ensure the access balance among servers. Compared to dedicate load-balancing devices, SDN brings an alternative approach that deploys a load-balanced application

at the controller. The controller installs packet-handling rules into involved switches inside the data center, which finally navigates traffic to dedicated servers in a load-balancing manner. The controller can utilize wildcard rules as a more scalable solution [14]. Wang et al. design corresponding algorithms to compute concise wildcard rules, which achieve a designed traffic distribution and are resilient to the changes of load-balanced policies without disrupting existing connections.

Additionally, FROSCO [58] provides a new application development framework, which allows developers to develop and deploy secure applications. Moreover, authors contribute the source code to the SDN community. Note that FROSCO focuses on ensuring the security of controller. Currently, less efforts have been done on the secure communication among controllers.

7. Future research issues of the controller

The SDN brings the possibility of various network innovations, but lacks uniform definitions and standard implantation in reality. Jarschel et al. make a thorough analysis on the SDN definition [59]. Many essential issues of the controller, however, need to be well addressed so as to improve the development and usages of SDN.

(1) The operators manage an underlying network through the controller, which is the key component of a software-defined network. To better manage networks, more abstractions, frameworks and programming languages need to be studied for deploying more distinguished control applications on the controller.

(2) The distributed control plane is a reasonable solution to the performance, scalability and topology issues of the control plane. However, maintaining a global network view is very difficult due to the dynamic network behaviors. For this reason, efficient methods need to be extensively studied for keeping the global consistent network view and reducing the resulting communication. Meanwhile, the design methods of completely distributed control plane still lack in-depth study.

(3) Currently, there are more and more applications to manage the whole network. It is impractical to deploy all applications into each controller. Therefore, the deployment problem of applications will need more research.

(4) The policies from different applications will be converted into flow rules on the control plane. There might be conflicts among rules. Meanwhile, to multiple applications that will deal with a same flow, it is need to compound the rules from these applications. Therefore, the rules compound and the conflicts avoiding are worth doing more works.

(5) Although there are many works about the scalability of controllers, the problem has not been well solved. Only increasing the number of controllers is not efficient to resolve the scalability of controllers. The locations and the number of controllers can also heavily affect the network performance. Actually, it is essential to research the coordination among controllers. Moreover, using the least number of controllers and finding their most appropriate locations still need more efforts.

(6) The northbound, southbound, and east-west interfaces need to be further developed. The northbound interface contributes to develop more applications to better use and manage the network. The southbound interface enables the controller to efficiently manage the underlying network. The east-west bridge makes the different SDN domains to efficiently communicate with each other. More and outstanding interfaces will significantly enhance the development of SDN.

(7) The security of the controller is always an important problem. If the controller is successfully attacked, the functionality of the software-defined network will be destroyed. To ensure the controllers' security, it is necessary to develop more security-aware control applications. Meanwhile, more techniques are required to ensure the secure communications among controllers and that between the controller and switches.

Acknowledgements

This work is partially supported by the National Basic Research Program (973 program) under Grant No. 2014CB347800, the NSFC under Grant No. 61422214, the Program for New Century Excellent Talents in University, and the Preliminary Research Funding of NUDT under Grant No. JC10-05-02.

References

[1] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, Communications Surveys Tutorials, IEEE 16 (3) (2014) 1617–1634.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: Enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[3] A. Campbell, I. Katzela, K. Miki, J. Vicente, Open signaling for atm, internet and mobile networks (opensig'98), ACM SIGCOMM Computer Communication Review 29 (1) (1999) 97–108.

[4] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, G. Minden, A survey of active network research, Communications Magazine,IEEE 35 (1) (1997) 80–86.

[5] J. Moore, S. Nettles, Towards practical programmable packets, in: Proc. IEEE INFOCOM, Anchorage, Alaska, 2001.

[6] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4d approach to network control and management, ACM SIGCOMM Computer Communication Review 35 (5) (2005) 41–54.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: Towards an operating system for networks, ACM SIGCOMM Computer Communication Review 38 (3) (2008) 105–110.

[8] Z. Cai, A. L. Cox, T. S. E. Ng, Maestro: A system for scalable openflow control, Tech. Rep. TR10-08.

[9] D. Erickson, The beacon openflow controller, in: Proc. ACM HotSDN, Hong Kong, China, 2013.

[10] IETF, Datatracker, Interface to the routing system (i2rs), `http://datatracker.ietf.org/wg/i2rs/` (2014).

[11] Cisco, Opflex: An open policy protocol, `http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html` (2014).

[12] M. Yu, J. Rexford, M. J. Freedman, J. Wang, Scalable flow-based networking with difane, ACM SIGCOMM Computer Communication Review 40 (4) (2010) 351–362.

[13] H. Hu, G.-J. Ahn, W. Han, Z. Zhao, Towards a reliable sdn firewall, in: Proc. USENIX Open Networking Summit, Santa Clara, CA, 2014.

[14] R. Wang, D. Butnariu, J. Rexford, Openflow-based server load balancing gone wild, in: Proc. USENIX Hot-ICE, Boston, MA, 2011.

[15] A. K. Nayak, A. Reimers, N. Feamster, R. Clark, Resonance: dynamic access control for enterprise networks, in: Proc. ACM WREN, Spain, Barcelona, 2009.

[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined wan, in: Proc. ACM SIGCOMM, Hong Kong, China, 2013.

[17] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, T. Vazao, Towards programmable enterprise wlans with odin, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.

[18] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking control of the enterprise, ACM SIGCOMM Computer Communication Review 37 (4) (2007) 1–12.

[19] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al., Onix: A distributed control platform for large-scale production networks., in: Proc. USENIX OSDI, Vancouver, BC, Canada, 2010.

[20] S. H. Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.

[21] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for openflow, in: Proc. USENIX INM/WREN, SAN JOSE,CA, 2010.

[22] K. Phemius, M. Bouet, J. Leguay, Disco: Distributed multi-domain sdn controllers, in: Proc. IEEE/IFIP NOMS, Krakow, Poland, 2014.

[23] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al., Onos: towards an open, distributed sdn os, in: Proc. ACM HotSDN, Chicago, IL, USA, 2014.

[24] A. Voellmy, J. Wang, Scalable software defined network controllers, ACM SIGCOMM Computer Communication Review 42 (4) (2012) 289–290.

[25] Opencontrail, `http://www.opencontrail.org/` (2014).

[26] M. Canini, P. Kuznetsov, D. Levin, S. Schmid, A distributed and robust sdn control plane for transactional network updates, in: Proc. IEEE INFOCOM, HongKong, 2015.

[27] Onos, `http://onosproject.org/` (2014).

[28] Opendaylight, `http://www.opendaylight.org/` (2014).

[29] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed sdn controller, in: Proc. ACM HotSDN, Hong Kong, China, 2013.

[30] D. Levin, A. Wundsam, N. Heller, Brandon aand Handigol, A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.

[31] A.-W. Tam, K. Xi, H. J. Chao, Use of devolved controllers in data center networks, in: Proc. INFOCOM WKSHPS, IEEE, Shanghai, China, 2011.

[32] S. Schmid, J. Suomela, Exploiting locality in distributed sdn control, in: Proc. ACM HotSDN, Hong Kong, China, 2013.

[33] A. Tavakoli, M. Casado, T. Koponen, S. Shenker, Applying nox to the datacenter, in: Proc. ACM HotNets, New York City, NY, USA, 2009.

[34] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proc. USENIX Hot-ICE, San Jose, CA, 2012.

[35] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: Measurements & analysis, in: Proc. ACM SIGCOMM IMC, Chicago, Illinois, USA, 2009.

[36] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 254–265.

[37] S. H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, Communications Magazine, IEEE 51 (2) (2013) 136–141.

[38] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H. J. Chao, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Computer Networks 68 (0) (2014) 95 – 109.

[39] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.
[40] Y. Jimenez, C. Cervello-Pastor, A. J. Garcia, On the controller placement for designing a distributed sdn control layer, in: Proc. Networking Conference, 2014 IFIP, Trondheim, Norway.
[41] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, On reliability-optimized controller placement for software-defined networks, Communications, China 11 (2) (2014) 38–54.
[42] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks., in: Proc. CNSM, Zurich,Switzerland, 2013.
[43] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network configuration protocol (netconf) (June 2011).
[44] A. Doria, J. H. Salim, W. Wang, L. Dong, R. Gopal, Forwarding and control element separation (forces) protocol specification (March 2010).
[45] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: A network programming language, SIGPLAN Not. 46 (9) (2011) 279–291.
[46] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker, Modular sdn programming with pyretic, USENIX 38 (5) (2013) 128–134.
[47] A. Voellmy, H. Kim, N. Feamster, Procera: A language for high-level reactive network control, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.
[48] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al., Composing software defined networks., in: Proc. USENIX NSDI, Lombard, IL, 2013.
[49] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, P. Hudak, Maple: Simplifying sdn programming using algorithmic policies, ACM SIGCOMM Computer Communication Review 43 (4) (2013) 87–98.
[50] A. AuYoung, S. Banerjee, J. Lee, J. C. Mogul, J. Mudigonda, L. Popa, P. Sharma, Y. Turner, Corybantic: Towards the modular composition of sdn control programs, in: Proc. ACM HotNets, College Park, MD, 2013.
[51] ONF, Open networking foundation north bound interface working group (nbi-wg) charter final version: V 1.1, `https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf` (2014).
[52] P. Lin, J. Bi, Y. Wang, East-west bridge for sdn network peering, Frontiers in Internet Technologies 401 (2013) 170–181.
[53] P. Lin, J. Bi, Z. Chen, W. Yangyang, H. Hu, A. Xu, We-bridge: West-east bridge for sdn inter-domain network peering, in: Proc. IEEE INFOCOM WKSHPS, Toronto, ON, Canada, 2014.
[54] D. Kreutz, F. M. Ramos, P. Verissimo, Towards secure and dependable software-defined networks, in: Proc. ACM HotSDN, Hong Kong, China, 2013.
[55] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, S. Shenker, Sane: A protection architecture for enterprise networks., in: Proc. Usenix Security, Vancouver, BC, Canada, 2006.
[56] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, G. Gu, A security enforcement kernel for openflow networks, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.
[57] A. Khurshid, W. Zhou, M. Caesar, P. B. Godfrey, Veriflow: Verifying network-wide invariants in real time, in: Proc. ACM HotSDN, Helsinki, Finland, 2012.
[58] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, M. Tyson, Fresco: Modular composable security services for software-defined networks., in: Proc. The Internet Society NDSS, San Diego, CA United States, 2013.
[59] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, W. Kellerer, Interfaces, attributes, and use cases: A compass for sdn, Communications Magazine, IEEE 52 (6) (2014) 210–217.