Dynamic Application Load Balancing in Distributed SDN Controller

Kenji HIKICHI, Toshio SOUMIYA, and Akiko YAMADA
Fujitsu Laboratories Ltd.
Kawasaki, Japan
hikichi.kenji@jp.fujitsu.com

***Abstract*— To deploy Software-Defined Networking (SDN) in a large-scale network, cluster-based distributed SDN controllers were proposed, which enable performance scaling by simply adding a new server in the cluster. However, loads caused by many control messages from many kinds of SDN applications have not been widely studied so far. In this paper, we show an architecture of a distributed SDN controller where all kinds of applications are running on all of the servers to share the loads caused by the applications. We propose a dynamic application load balancing method which calculates the weight of round robin scheduling of an external load balancer to distribute requests submitted to the applications among the servers. Experimental results show the method increases the performance of the controller by reducing inter-server messages.**

Keywords—SDN; distributed SDN controller; OpenFlow; distributed sytem; load balancing; network virtualization

I. INTRODUCTION

To overcome the limitation of programmability in traditional networks, many studies have been carried out on Software-Defined Networking (SDN) [1] in recent years. Prior to SDN, network control logics such as path computation algorithms were tightly coupled with distributed hardware switches, which prevented agile adaption to customer requirements and consolidation with other IT services. On the contrary, OpenFlow/SDN [2] decouples network control logics from hardware switches, and concentrates them at centralized SDN controllers. SDN controllers manage switches by handling southbound protocols such as OpenFlow and maintaining network topology and network states. They also provide an application programming interface (API) for retrieving network information and controlling switches, with which network control logics are implemented in SDN applications. Since the applications are software, flexible network controls which follow customer requirements can be realized more quickly than with traditional networks.

Since SDN is centralized architecture, SDN controllers are required to scale performance when varying the number of switches and control messages. Cluster-based distributed SDN controllers are proposed to address this problem [3], [4], [5]. The distributed controllers form a cluster which consists of several servers, called "c-nodes" in this paper. Each c-node shares topology information and manages a separate subset of switches, which enables performance scaling by simply adding or removing c-nodes. In this paper, our study is based on the open-source distributed SDN controller ONOS [5].

In addition to scalability with a varying number of switches, SDN controllers are also required to scale performance when many control messages from many kinds of applications are issued. This requirement is necessary when SDN is deployed in a wide-area network (WAN). In WAN areas, many customers use the same network, and they require their traffic to be controlled by their own policy such as best-effort service, guaranteed quality of service and multipath routing. We consider that these policies should be implemented as their SDN applications; consequently many kinds of applications will use the same SDN controller. However, the loads caused by many applications in this context have not been studied sufficiently in previous works. In the architecture of ONOS, all kinds of applications are running on all c-nodes, and the loads of applications can be shared among c-nodes. However, the performance will be degraded due to increase of inter-c-node messages unless requests to the applications are assigned among c-nodes properly.

In this paper, we propose a **dynamic application load balancing** method**.** This method works together with an external server load balancer, and calculates the weight parameter of round robin scheduling in the load balancer to distribute requests among c-nodes. We developed an algorithm for calculating the parameter to reduce inter-c-node messages. We also show an architecture of the SDN controller called a **multi-policy controller**, which makes multiple customer applications control the network independently. The dynamic application load balancing method works effectively with this architecture.

The rest of this paper is organized as follows. Related works on distributed controllers are shown in Section II. An architecture of the multi-policy controller and dynamic application load balancing function is described in Section III. The algorithm for dynamic application load balancing is proposed in Section IV. The method and results of evaluation of the performance of the multi-policy controller when varying the number of policies, and the performance of the proposed algorithm is described in Section V. We conclude this paper by showing the limitations of our system and future works in Section VI.

II. RELATED WORKS

There are several related works on distributed SDN controllers. In the OpenFlow specification, switch configuration such as flow table entry can be modified only by the master c-node, and the master c-node should be assigned to equalize the number of messages to/from switches among c-nodes. Load balancing of the master c-node is proposed in [3] and [6] to increase the scalability when varying the number of switches. The architecture of a loosely coupled SDN controller is proposed in [4], where an SDN application and SDN controller are separated with a messaging system. In this architecture, a method to increase the scalability of the messaging system is proposed, which consists of multiple message queues. On the contrary, ONOS [5] is classified into a tightly coupled architecture, where both the SDN application and SDN controller function are located in the same c-node. ONOS also has a monolithic property, which means all c-nodes in the cluster have the same components including SDN applications. However, these previous distributed controllers have not previously considered the loads caused by many control messages from many kinds of applications sufficiently.

III. ARCHITECTURE

In this section, we explain the architecture of the proposed system, which is based on ONOS. We show the architecture of the multi-policy controller followed by dynamic application load balancing based on the architecture.

A. Multi-policy controller

Implementing customer policy as applications enables a flexible and agile change of network control logics following customer requirement. To make customer applications control their network independently, the SDN controller is required to isolate them from each other. We realize this by slicing physical network resources into several virtual networks. Customer applications are confined into their virtual networks (Figure 1). To describe how the isolation is realized, we explain the architecture of the system shown in Figure 2. In this figure, c-nodes in an SDN controller cluster have the following components. Note that among the components, "physical network manager" and "virtual network manager" are components of ONOS, on the other hand the "app-network binding function" are extensions by us.

The **physical network manager** handles southbound protocols such as OpenFlow to manage physical network resources, which includes switches, links and ports, it also sends control messages to change the behavior of the switches. The relationship between a switch and its master c-node is maintained in this component.

The **virtual network manager** defines virtual networks which consist of virtual switches, virtual ports and virtual links on a physical network. Virtual ports are associated with one of the ports in the physical network and virtual links are associated with tunnels realized by labels such as VLAN or MPLS.

Both the physical and virtual network managers are distributed components, which means that the same information about the physical and virtual network topology and their states are shared among c-nodes [5].

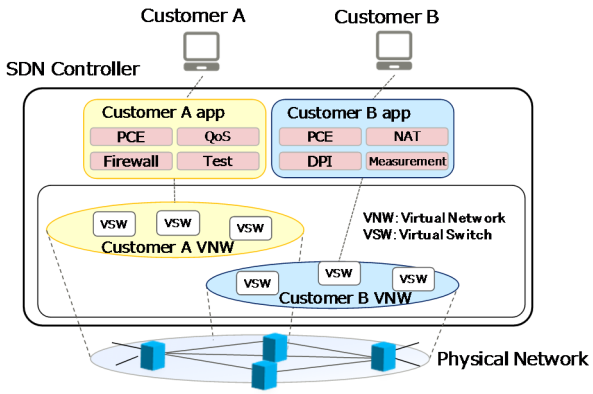

Figure 1 Logical view of multi-policy controller

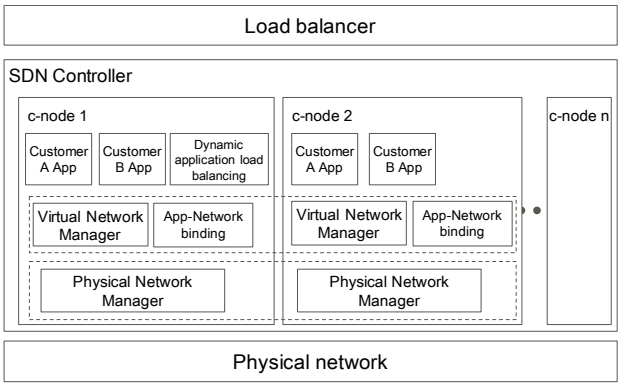

Figure 2 Architecture of the proposed system

SDN applications (apps) are software which receives requests from load balancers, and they issue several control messages applied to the switches. Several kinds of applications are deployed in the SDN controller, and instances of all applications are running on all c-nodes. We assume that apps have "transparency property", which means the same control messages are issued with the same request wherever c-node apps work. This property is realized by the distributed nature of the physical and virtual network managers.

The **app-network binding function** confines SDN applications into one of the virtual networks. "Confine" here means that the apps can only view and control the resources in the associated virtual network through the virtual resource manager.

Isolation among apps is achieved by the app-network binding function and the virtual network manager. API calls issued by confined apps are redirected to the virtual network manager instead of the physical network manager by the app-network binding function. The virtual network manager provides the same API as that of the physical network manager, and it returns the information about the associated virtual network as a response to the API call for retrieving network information. Moreover, when the apps issue a Flow Table Modification Message [1] which changes the forwarding rule of

a virtual switch, the virtual network manager converts it to the message for the physical switch whose port are associated with the virtual switch. By adding the push-VLAN or push-MPLS action to the converted message, control messages issued by different apps are isolated from each other.

B. Dynamic application load balancing

With regard to the dynamic application load balancing, the system has the following two components which are an extension of ONOS developed by us.

The **load balancer** is located between the SDN controller and external clients to distribute requests to apps among the c-nodes. The load balancer can simply distribute requests among c-nodes without maintaining any states associated with requests owing to the transparency property of apps, since the load balancer has separate queues for each app, and requests to the apps are assigned to one of the c-nodes with weighted round robin scheduling for each queue.

The **dynamic application load balancing function** is running on one of the c-nodes. It collects the measurement results from all c-nodes which includes the number of requests to the apps and control messages issued by the apps, and it computes an optimal weight parameter of round robin scheduling which is configured to the load balancer.

The key point of the dynamic application load balancing is to change the weight parameter dynamically based on the measurement results. The details of the algorithm and measurement are explained in the next section. The multi-policy controller was explained in this section, and we will show the evaluation results of it when varying the number of policies in section V.

IV. DYNAMIC APPLICATION LOAD BALANCING

In this section, we start by explaining the issues of load balancing of requests to apps followed by the description of our load balancing algorithm. For simplicity, in this section and later, we assume that SDN apps are path computation functions which process path setup requests between two endpoints and issue several control messages applied to the switches on the calculated path and confirm that all control messages are installed.

A. Issues of load balancing

In this subsection, we point out that the load balancing algorithm should take inter-c-node messages into consideration. To explain it, the detailed process of the SDN controller is shown after a path setup request is received by one of the c-nodes denoted by (a) in Figure 3. The app calculates a path between two endpoints and issues control messages for virtual switches (b). The messages are converted to control messages for physical switches by the virtual network manager (c). The physical network manager finds the master c-node of the destined switch of the control message. If the master c-node is the c-node itself, the physical network manager sends the message directly to the switch (d), otherwise the message is sent to the switch via other c-nodes denoted by (e) and (f). The former message is called a "direct message", and the latter one is called

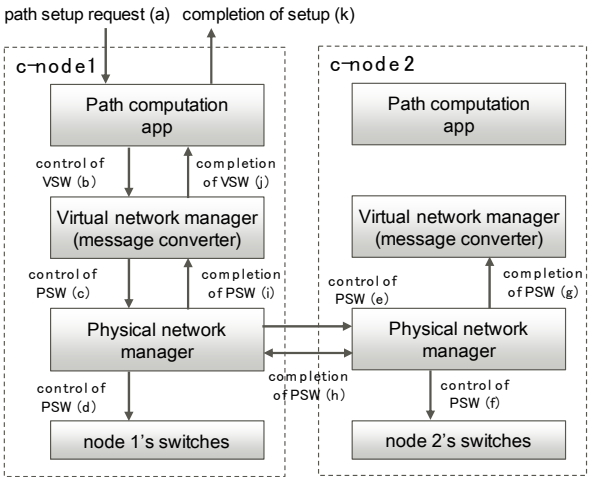

Figure 3 Detailed process of the SDN controller
(VSW: virtual switch, PSW: physical switch)

an "inter-c-node message". The physical network manager checks that the message installation is completed, and issues the completion message. The message is sent to all virtual network managers among the c-nodes denoted by (g), (h) and (i). The virtual network manager converts the message to that of the associated virtual switch (j), and the app issues a path setup completion message after all messages of switches on the path are received (k).

From this process, the tasks of c-nodes can be classified into the following two kinds of tasks:

CPU-bound task: path calculation in the app and message conversion in the virtual network manager

I/O-bound task: sending/receiving messages to/from switches or other c-nodes

To increase scalability of the SDN controller by adding the c-nodes, the load of these tasks should be balanced among c-nodes. CPU-bound tasks can be balanced by distributing requests with round robin scheduling in which the weight parameter is determined by the c-node processing performance. Regarding I/O-bound tasks a master c-node assignment algorithm to equalize the number of messages between c-nodes and switches was proposed in a previous work [3]. In addition, inter-c-node messages should be reduced because they are

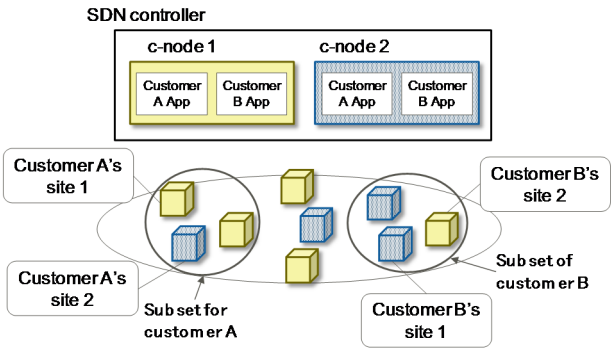

Figure 4 Example of master c-node assignment
(The color and pattern of switches shows master c-node)

distributed SDN controller-specific overheads which means they don't exist in single controller architecture. As described in (d), (e) and (f) in Figure 3, the amount of inter-c-node messages depends on the c-node where the app works, which is determined by the load balancer. That's why the load balancing algorithm should take inter-c-node messages into consideration.

B. Algorithm

To reduce inter-c-node messages, we exploit the idea that customer apps tend to control a subset of switches. We believe this characteristic is common in WAN areas; for example, a customer app tends to control the switches around the customer sites shown in Figure 4. Moreover, when a certain c-node is the dominant master of the subset, which means that the c-node is the master of many switches, the inter-c-node messages are likely to be reduced by directing requests to the app on the c-node. For example, "Customer A app" tends to control the subset of switches labeled "Subset for customer A" in Figure 4, and c-node 1 is the dominant master in the subset. In this case, inter-c-node messages issued by the app are likely to be reduced by directing the request to c-node 1. Note that here it is assumed for simplicity that the subset is pre-defined and the number of messages sent to each switch is equal, but the following method does not assume this.

However, the load of c-nodes cannot be balanced if requests are concentrated on a certain c-node. To avoid this we determine a weight parameter to maximize the amount of direct messages with keeping equality of the total number of requests for all kinds of apps. This means the number of requests for a certain kind of app is not balanced among c-nodes; however, it is balanced for all kinds of apps. We formalized this process with linear programming as follows:

Step 1 The number of requests received by apps and control messages issued by the apps are measured for all c-nodes at every fixed time, then we calculate the following values:

a_{ij}: The ratio of "direct messages" to all control messages issued by app i on c-node j.

r_i The amount of total requests received by app i.

Step 2 The dynamic application load balancing function collects a_{ij} and r_i from all c-nodes, then solves the following linear programming problem:

$$Maximize \sum_{1\le i\le m, 1\le j\le n} a_{ij}x_{ij} \text{ (the number of direct messages)}$$

$$\text{s.t.} \sum_{1\le i\le m} x_{ij} - c \le \delta, \sum_{1\le i\le m} x_{ij} - c \ge \delta, \text{for each c} - \text{node } j$$

(c-nodes' load should be balanced)

$$and \sum_{1\le j\le n} x_{ij} = r_i, \text{ for each application } i$$

(all requests should be assigned)

where

x_{ij} (variable): The number of requests to app i which are assigned to c-node j.

m: The number of applications.

n: The number of c-nodes in the controller.

δ: Parameter to relax load equalization.

c: C-node's capacity calculated as $c = \frac{\sum_i r_i}{n}$

Step 3 The function calculates the weight parameter from the result of Step 2 using the following equation, and configures it to the load balancer.

$$w_{ij} = \frac{x_{ij}}{\sum_j x_{ij}}$$

where w_{ij} is the normalized weight of c-node j for round robin scheduling of requests to app i.

V. Evaluation

In this section, the method and results of evaluation of the proposed system with emulated OpenFlow switches are shown. We conducted the following three experiments:

E1 Evaluation with varying number of policies

E2 Evaluation of a simple load balancing

E3 Evaluation of the dynamic load balancing method

E1 is intended to evaluate the multi-policy controller. E2 and E3 are comparative evaluations between load balancing algorithms without and with taking inter-c-node messages into consideration. The term "simple load balancing" here means that the load balancer uses round robin scheduling where the weight parameter is determined only by c-node processing performance. We start by explaining the evaluation setup followed by the details of each evaluation.

A. Evaluation setup

The SDN controller is built on ONOS 1.3 and we extend it by implementing the virtual network manager, the app-network binding and the dynamic application load balancing function. The physical network is built on emulated OpenFlow switches. Since the switches work as a component of c-nodes, communication overheads between the switch and c-node are relatively low compared to those between c-nodes. The number of master c-nodes are balanced equally among c-nodes in advance. The topology of all virtual networks is the same as that of the physical network.

Elapsed time is measured from submitting a set of path setup requests to confirming that all paths are installed in the switches. The requests are submitted by the other app, which works on all c-nodes to eliminate the influence of client and load-balancer performance. For simplicity, the requests satisfied the following properties:

- Endpoints of the path setup request are different ports of the same switch, consequently apps generate only one control message per request.

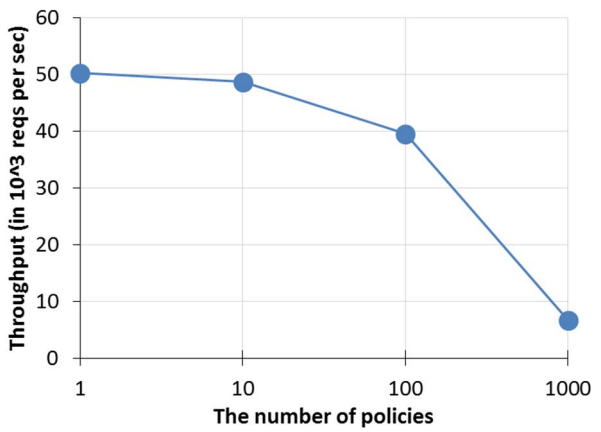

Figure 5 Performance when varying number of policies

- The destined switches of the requests are equally balanced among all switches.

The servers for all c-nodes have CPU of 4 core Intel Xeon CPU E31240 @ 3.30 GHz, and 8 GB of memory is assigned to the instance of ONOS. All c-nodes are connected to the same layer 2 switch with 1 Gbps link. The number of switches is 15, and the number of path requests is 900,000. In all figures from this point, we show the throughput of the system, which is the number of path requests divided by the measured elapse time.

B. E1: Evaluation with varying number of polices

This evaluation is intended to evaluate the virtual network manager and app-network binding function, which are used to realize the multi-policy controller. In this evaluation the SDN controller consists of only one c-node. We measure throughput while varying the number of policies and corresponding applications and virtual networks. The requests are equally submitted among all of the applications.

Figure 5 shows that the throughput degrades when increasing the number of policies. We found that this degradation was caused by the overheads of handling completion messages denoted by (i) in Figure 3. This problem heavily depends on the current implementation of the virtual network manager and app-network binding function which create the same number of instances of the component to handle the completion messages as the number of virtual networks. The completion message from the physical network manager is broadcasted to all of the instances, and the instances use a lot of time to filter irrelevant messages. In the evaluation later, we evaluate the system with one virtual network and one kind of app to avoid this performance degradation.

C. E2: Evaluation of simple load balancing

In this evaluation, we measured the throughput of the SDN controller with the simple load balancing where the weight parameter of the round robin scheduling is determined only by c-node processing performance. The number of working c-nodes varies from one to the total number of c-nodes, which is three or five. Since all servers for c-nodes have the same processing performance, the weight parameters of all working c-nodes are the same.

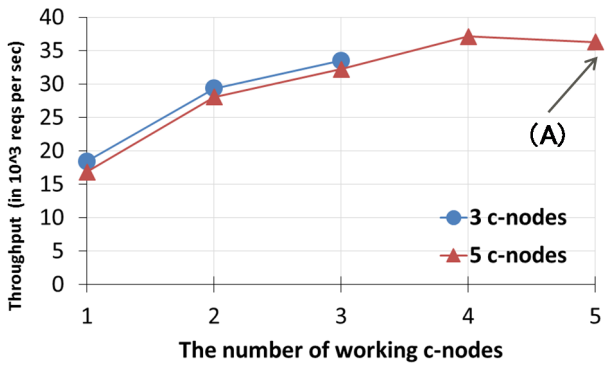

Figure 6 Performance when varying the number of working c-nodes

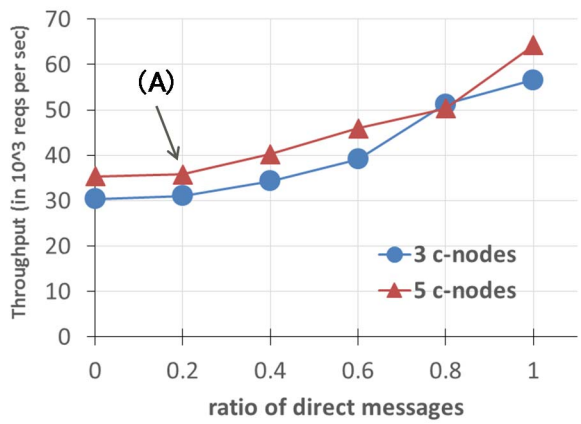

Figure 7 Performance when varying ratio of direct messages

Figure 6 shows that the result of the evaluation with three and five c-nodes clusters. From this figure, we can find that the throughput is increasing when increasing the number of working c-nodes in both cases. It indicates the loads of CPU-bound tasks are equally balanced among c-nodes and the load for every c-node is decreasing. However, the increasing amount of the throughput by adding one c-node becomes lower when increasing the number of working c-nodes. This result indicates that the influence of I/O-bound tasks becomes relatively larger than CPU-bound tasks. We consider the reasons of this is that inter-c-nodes messages don't decrease when increasing the number of running c-nodes with the simple load balancing.

D. E3: Evaluation of dynamic load balancing

We evaluated the improvement of the throughput with the dynamic load balancing. As described in Section IV this function works to maximize direct messages, we measured throughput when varying the ratio of it. To change the ratio, the path setup requests are intentionally created by sniffing the mapping information between virtual and physical switches and the master c-nodes. For example, to configure the ratio to 1.0, requests are always directed to the master c-node of the destined physical switch of the requests.

Figure 7 shows that the result of this evaluation with three and five c-nodes clusters. In this evaluation, apps are running on all c-nodes, this result can be compared with that of E2. For example, the result of the number of working c-nodes of "5" on "5 c-nodes" line in Figure 6 is the same as that of the ratio of $0.2(= 1/5)$ of "5 c-nodes" line in Figure 7, where both points are denoted by (A). From this figure, we can find that the throughput becomes larger when increasing the ratio of direct messages. The improvement of the ratio by our algorithm is depends on characteristics of applications, network topology, customer behavior, etc., however, this result indicates our algorithm shows better performance than the simple load balancing by reducing inter-c-node messages.

VI. CONCLUSION

In this paper, we described an architecture of the distributed SDN controller which can handle multiple customer apps, and can work together with external server load balancers. We proposed an algorithm to optimize the weight parameter of the load balancer dynamically to reduce inter-c-node messages. Experimental results show that our method increases the performance of the controller. However, there are still possibilities to make the algorithm better. For example, this algorithm assumes that the loads of processing one request are the same among all kinds of apps. This assumption should be removed to deploy SDN in realistic and practical environments. Moreover, evaluation on how many inter-c-nodes messages can be reduced by our algorithm is required. It is still necessary to improve performance and scalability of the distributed SDN controller to deal with multiple policies. For example, an architecture of handling notification messages from the physical network manager should be improved so as not to degrade the performance when increasing the number of virtual networks. We plan to improve the performance of the SDN controller by addressing these problems.

REFERENCES

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," ONF White Paper April 13, 2012.

[2] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.4 (Protocol version 0x04)," March 27, 2014.

[3] K. Hikichi, S. Shimizu, A. Yamada, and T Somiya, "Study on Scalability for Distributed SDN Controller," Technical Report of 13th IEICE NV, Mar 17, 2015 (in Japanese).

[4] S. Shimizu, A. Yamada, and T Somiya, "Study on Scalable Messaging System for Distributed SDN Controller," IEICE Tech. Rep., vol. 113, no. 472, NS2013-212, pp. 207-212, March 2014 (in Japanese).

[5] P. Berde et al. "ONOS: Towards an Open, Distributed SDN OS," In Proc. of ACM HotSDN 2014, Aug. 2014.

[6] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Oct. 2013.