2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications

A Load Balancing Strategy for SDN Controller based on Distributed Decision

Yuanhao Zhou[1], Mingfa Zhu[1], Limin Xiao[1], Li Ruan[1], Wenbo Duan[1], Deguo Li[1], Rui Liu[1]
[1]School of Computer Science and Engineering, Beihang University, Beijing, China, 100191
{zhouyh, atomd, lideguo, lr}@cse.buaa.edu.cn
{zhumf, xiaolm, ruanli*}@buaa.edu.cn

Mingming Zhu[2]
[2]Huawei Technologies Co. LTD.
Beijing, China, 100085
tinny.zhu@huawei.com

***Abstract*—Software-Defined Networking (SDN), enabled by OpenFlow, represents a paradigm shift from traditional network to the future Internet. Replicate or distributed controllers have been proposed to address the issues of availability and scalability that a centralized controller suffers from. However, it lacks a flexible mechanism to balance load among distributed controllers. To address this problem, this paper presents DALB, a dynamic and adaptive algorithm for controller load balancing totally based on distributed architecture, without any centralized component. This algorithm is running as a module of SDN controller. On one hand, it adopts an adjustable load collection threshold so as to reduce the overhead of exchanging messages for load collection, and on the other hand it can make policy and election locality in order to reduce the decision delay caused by network transmission. In this paper, we build the prototype system on floodlight to demonstrate our design and test the performance of our algorithm.**

Keywords—OpenFlow; Load Balancing; Distributed Decision; Software-Defined Networking(SDN)

I. INTRODUCTION

Software-Defined Networking (SDN) based on OpenFlow [1] is currently seen as one of the most promising paradigm shift from traditional network to the future Internet by enabling programmability, easier management and faster innovation [1, 2, 3]. Unlike traditional network, the control plane is decoupled from data plane in SDN architecture. A centralized control plane, which is called SDN controller, brings many benefits such as programming the network by applications without concerning for the underlying network infrastructure, controlling the network from a central node and so on. However, the centralized controller imposes potential issues of scalability and availability. Thence researchers propose logically centralized, but physically distributed SDN controllers to tackle these problems.

There are a few papers on how to build and implement distributed SDN controllers, such as Onix [4], HyperFlow [5] and OpenDaylight [6]. They focus their attention on the necessary components to achieve distributed control plane and provide a global view of the network topology for upper applications. Although we can improve the scalability and availability with the help of multiple controllers, another issue is inevitable. How to maintain the map between a switch and a controller in case of one of the controllers is not overwhelmed? Even if we deploy the switches and controllers very carefully, it's difficult for controllers to adapt to changeful traffic load. This problem could be explained from another perspective. Real networks show the characteristics from two aspects: temporal and spatial [7]. For instance, from the temporal angle, there may be less traffic during the night. However, there could be a large scale of flows generated by applications of routing calculation in a very short time. Besides, from the point of spatial, applications running on different controllers possibly compute and generate different numbers of flows, and some switches can get lots of flows compared to other domains of the network. So it's essential for us to balance load dynamically among distributed controller cluster instead of static network configuration. Overloaded controller should be detected and high-load switches mapped to this controller ought to be smoothly migrated to the under-load controllers so as to improve resource utilization of distributed controller cluster.

To address the problem above, in this paper, we propose a dynamic and adaptive algorithm (DALB) for SDN controller. This algorithm is running as a module of SDN controller and the controllers in distributed environment can cooperate with each other to keep load balancing. Specifically, our algorithm contains three important features:

- DALB is totally based on distributed architecture, without any centralized component. Meanwhile, it is running as a module of SDN controller. With the help of our algorithm, each controller can collect its own load, adjust load collection threshold, gather other controllers' load, make decision and election, migrate switches and so on. As a result, our algorithm avoids single-point problem, providing scalability and availability.
- DALB is running on every SDN controller and it allows controller make load balancing policy locality, which reduces the decision delay caused by network transmission. Besides, DALB adopts an adjustable load collection threshold in order to decrease the overhead of exchanging messages in controller cluster.

978-1-4799-6513-7/14 $31.00 © 2014 IEEE
DOI 10.1109/TrustCom.2014.112

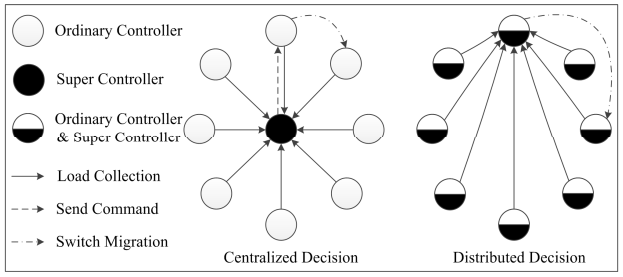

Fig. 1. Centralized Decision and Distributed Decision

- DALB can make load decision based on the collected information locality and elect high-load switches to migrate to an under-load SDN controller to realize load balancing among distributed controllers.

The rest of this paper is organized as follows. Section II reviews relevant research related to SDN controller load balancing. Section III outlines the architecture of DALB and we describe its components in Section IV. Section V illustrates the verification environment and analysis of experiment results. Section VI is our conclusion to the algorithm for SDN controller load balancing and it introduces future work of our project.

II. Related Work and Motivation

We can deploy switches and correspondent distributed controllers as reasonable as possible. However, real networks may exhibit dynamic changes of traffic condition because of temporal and spatial characteristics. Routing calculation, asymmetric flows arrival rate, along with other factors can result in load imbalance among distributed controllers. The usual method to solve uneven load problem can be divided into the following two categories.

A. Load Balancing Algorithm based on Centralized Decision

One way to solve uneven load problem among distributed controllers is that deploying a super controller, which is responsible for balancing the load of all controllers. This measure is taken by BalanceFlow [8]. Typically, as Fig. 1 shows, the centralized decision controller node collects the load information of other controllers and then decides whether a load balancing action should be launched or not. This is a good way to monitor the load change of all controllers from a global view. However, this algorithm may own two limitations. (1) The performance of a centralized node is limited by memory, CPU power and bandwidth. Besides, a centralized node collects load information periodically and it exchanges lots of messages frequently with other controllers, which will lead to performance reduction of the whole system. Moreover, if the central node collapses, the whole load balancing strategy is down. This goes against availability among distributed SDN controllers. (2) As Fig. 1 exhibits, each load balancing action need two network transmissions: one for collecting load and the other is for sending commands. In this situation, the aggregated load information may be past due and the command is lag behind the real load condition.

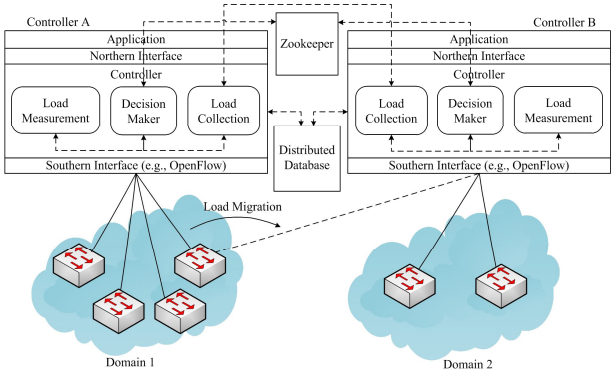

Fig. 2. Architecture of Load Balancing for Distributed SDN Controller

B. Load Balancing Algorithm based on Distributed Decision

As Fig. 1 shows, the algorithm based on distributed decision allows every SDN controller collect other controllers' load, make its own decision locality. In other words, every distributed controller acts as not only ordinary controller but also super controller. By this way, we will get two advantages. (1) This algorithm is totally based on distributed architecture, which guarantees the scalability and availability of distributed SDN controllers. (2) This algorithm just needs one network transmission for gathering load and doesn't need push command to other controllers. As a result, the decision delay will be reduced in this case.

In addition, in order to avoid all controllers collect the load information in cluster too frequently, leading to large numbers of messages transmitted for load collection in SDN environment, we adopt a dynamic and adaptive threshold to prevent frequent load collection. In short, load balancing of distributed SDN controllers is a key issue in academic community, and the algorithm based on distributed decision is firstly adopted to address this issue in this paper.

III. System Overview

The key components in our load balancing strategy for distributed SDN controller design are shown in Fig. 2. It consists of the following four parts:

- The domains of SDN network refer to different sets of the physical network infrastructure such as OpenFlow switches and links that carry the data and control plane traffic. A switch can connect to multiple controllers simultaneously, which is supported by OpenFlow 1.2 or OpenFlow 1.3. Although a switch can connect to multiple controllers at the same time, only one controller is active and acts as the master and the others are slaves.
- Each distributed SDN controller manages a domain of SDN network and they communicate with each other by message. DALB is running as a module of controllers. It consists of three components. (1) Load Measurement is responsible for collecting load information of local controller. (2) Load Collection is used for aggregating other controllers' load information. (3) Decision maker

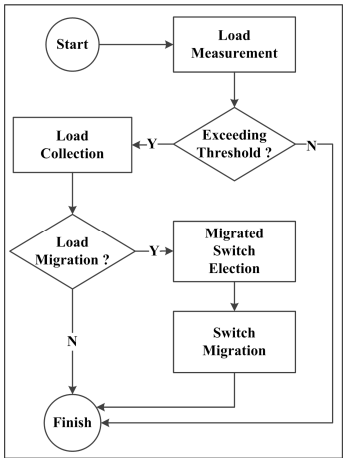

Fig. 3. Flowchart of DALB

is in charge of estimating whether controller load exceeds the threshold, making election to balance load among controllers, adjusting the new threshold and migrating switch to other controllers.

- The distributed database glues the cluster of controllers to provide a logically centralized controller for upper applications. It stores all switches' specific information (i.e., link information, topology information).
- Zookeeper [9] is used for coordinating controllers' operation.

DALB presented in this article runs on each controller and they cooperate with each other to keep the load balance among all controllers. Fig. 3 illustrates the flowchart of DALB and their relationship with each other.

When DALB is running as a module in controller, the controller will periodically collect its own load information. Then it will check whether the load is beyond the threshold. If the load exceeds the threshold, the controller will gather other controllers' load information. After aggregating all load information from the controller cluster, this high load controller will make migration decision and select which switch will be migrated to a low load controller. With the help of zookeeper, the selected switch will be migrated to the objective controller. At last, the threshold will be adjusted to a new value.

In the following sections, we will explain the components of DALB in detail.

IV. DESIGN AND IMPLEMENTATION

In this section, we will describe every components of DALB in detail, including Load Measurement, Load Collection and Decision Maker. Moreover, we will present the algorithms which guarantee the effectiveness of our load balancing strategy for distributed SDN controllers.

A. Load Measurement

A load measurement module runs on the controller to report load statistics, including the number of flow table entries (N), the average message arrival rate from each switch (F) and the round trip time from each switch to controller (R). TABLE I illustrates the purpose of each collecting load statistics.

TABLE I. LOAD MEASUREMENT

Load Statistics	Purpose
The number of flow table entries	Controller Election
The average message arrival rate	Threshold Estimation and Controller Election
The round trip time	Controller Election

It's easy for us to understand that the load of controller is in proportion to the message numbers per unit time sent from the switches. According to OpenFlow protocol, there are different kinds of requests sent to controller such as Hello message, Echo message, Packet-in message and so on. However, Packet-in messages occupy the largest ratio. As a result, we count the average packet-in message arrival rate to represent the load of a controller.

In addition, when a controller is overloaded, we need to make election so as to decide which switch ought to migrate to another controller. The election condition is integrated of the number of flow table entries, the average message arrival rate and the round trip time. With a large scale of flow table entries, the controller manages a very huge flow table and the load of a controller will be high. Bigger average message arrival rate of a switch, it means this switch brings more load to the controller. Besides, the round trip time is also an influence factor. If a controller is overloaded, we elect a switch to migrate according to the following formula based on the three factors above.

$$c_{Load} = w_1 * N + w_2 * F + w_3 * R \quad (1)$$

In the formula, w_1, w_2, w_3 are weight coefficients and the sum of them is 1.0. N, F, R represent the three factors talked above. We can compute the load of each switch bringing to the controller and decided which one should be migrated.

B. Load Collection

As mentioned above, we census the average packet-in message arrival rate as the load of a controller. In order to make the right load balancing decisions, DALB need to collect the other controllers' load information of the whole system. However, frequently sending or receiving messages may decrease the performance of the whole system. In order to address the issue above, a load collection threshold (denoted with CT, short for controller threshold) is adopted in DALB. The threshold CT is stored in the distributed database and each controller node can read from the database. We set the initial value of CT is 1000 for the reason that our prototype is based on Floodlight. 1000 means the packet-in arrival rate is 1000 messages per second and this value is used by Floodlight [11] to control the flows.

However, if all distributed controller nodes are overloaded (which means the load of each controller exceeds CT), setting the CT equal to 1000 will result in the communication

overhead is n^2 (n represents the number of controllers). This will also degrade the performance of the distributed SDN controllers. In order to tackle this problem, an adaptive load collection threshold adjusting algorithm named AdaptiveCT is adopted. In such a case, the communication overhead will be $\frac{1}{2} n^2$ in the worst case. AdaptiveCT is presented in Algorithm 1. The idea of AdaptiveCT is adjusting the value of CT to the average load of all controllers when the whole cluster is overloaded.

Algorithm 1 AdaptiveCT

Input:

$\{L_1, \ldots, L_n\}$: List of all controllers' load

Output:

CT: New load collection threshold

1: $\delta = \frac{1}{n} \sum_{i=1}^{n} (L_i)$

2: ICT = 1000

3: if $\{L_1, \ldots, L_n\}$ is not complete **or** $\delta \leq$ ICT **then**

4: CT = 1000

5: else if $\delta >$ ICT **then**

6: CT = δ

7: end if

8: return CT

We implement the load collection module named LoadCollect with JGroups [10]. JGroups is a toolkit for reliable messaging. It can be used to create clusters whose nodes can send messages to each other. When a controller's load is greater than CT, it will send messages to request load information from the other controllers. After gathering all nodes' load, it will make migration decision. At last, this controller will update CT with AdaptiveCT algorithm.

C. Decision Maker

Decision maker plays an important role in DALB because it decides whether adopting load migration operation. In order to realize an effective distributed decision mechanism. We need to consider two problems.

First, DALB running on each controller need not only take its own load information into consideration, but also give consideration to the whole system. One controller's load has been talked about in Load Measurement. We use the packet-in arrival rate to measure the load of a controller. In the view of the whole system, we define load balancing rate ρ in the following formula.

$$\rho = \frac{\frac{1}{n} \sum_{i=1}^{n} \{L_i\}}{\max_{i=1}^{n} \{L_i\}} \quad (2)$$

$\{L_1, \ldots, L_n\}$ represents the load list of the whole system (including the overloaded controller's own load). It's obvious that the codomain of ρ is between 0 and 1. If ρ is close to 1, it means the load is evenly distributed. In our prototype, we set the initial value of ρ 0.7. If a controller's calculation result of

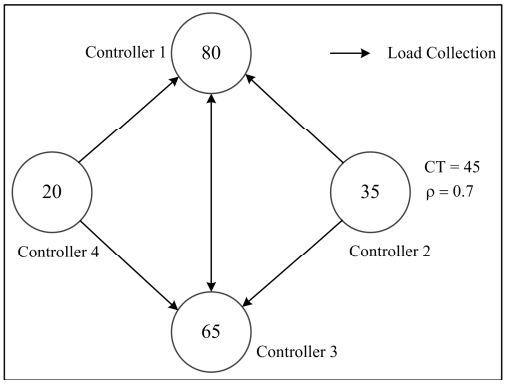

Fig. 4. Example of Decision Maker Module

ρ is smaller than 0.7, it means that this controller need to take load migration operation. In contrast, if the calculation result is bigger than 0.7, it means this controller's load is well-distributed in view of the whole system.

We introduce ρ to consider the load of the whole system. However, another problem is inevitable. If there are two controllers' load exceeding the threshold and their calculation results of ρ are all smaller than 0.7, it will result in two controllers taking migration operation at the same time and they may migrate their switch to the same target controller, which will result in the target controller becomes a new overloaded controller. To address this issue, we add a restrictive condition when making migration decision. If the overloaded controller's load is not the maximum value in the load list of all controllers, it can't take migration decision. In other words, if an overloaded controller wants to migrate its switch, it needs to satisfy two conditions: (1) Its ρ is smaller than 0.7. (2) Its load is the biggest in the load list of all controllers. By this way, we can avoid multiple controllers migrate switches at the same time.

In order to explain how the decision maker module works, we take a simple example. Fig. 4 presents an example of decision maker module. The load of controller 1 and controller 3 is greater than CT, so they need to collect the other controllers' load information. Then ρ is computed according to (2) and the value is 0.625. Controller 1 and controller 3 have the same ρ and the value is smaller than 0.7. Because the load of controller 1 is the biggest, so controller 1 will launch load migration and controller 3 won't. In the next load balancing cycle, controller 3 will be the overloaded node but the ρ is 0.77 (greater than 0.7). As a result, controller 3 doesn't need to make migration decision. Therefore, it is an effective distributed load balancing decision mechanism by means of limited ρ and demanding the maximum load.

D. Migrated Switch Election and Switch Migration

In this section, we will discuss how to select a switch for migration and the process of switch migration.

According to (1), we can know that N, F and R are the factors that switches affect controllers' load. We assign w_1, w_2, w_3 to 0.1, 0.8 and 0.1. Because in stable LANs, the flow tables

are like routing tables and it's not frequent to modify them. The round trip time is also very small in LANs. Therefore, we set w_1 and w_3 very small values.

The goal of DALB is balancing the load among controllers as far as possible. In an ideal condition, the reduced load of an overloaded controller had better equal the increased load of the target controller. However, it is impossible for us to achieve this. We can conclude that if the migrated switch's load equals or less than half of the minus of two controllers, the high load controller's load may be decreased. So we choose the migrated switch with the following formula.

$$L_{Migrate} \leq \frac{L_{Overloaded} - L_{Target}}{2} \quad (3)$$

In the migrated switch selection algorithm, we don't design very complex election algorithm. We choose a high load switch according to (3). After the migration, we also need make sure the load of the overloaded controller is less than CT. With these conditions, we think the migration election is successful.

In Section III, we talk about that a switch can connect to multiple controllers at the same time. However, only one controller is active and acts as the master, the others are slaves. With the help of Zookeeper, we can change the role of master node and slave node. We can realize switch migration with this method. In order to ensure the high available during the migration, we use HA-TCP to guarantee the messages (sent from migrating switch to controller) will not be lost.

E. Dynamic and adaptive load balancing algorithm

The whole algorithm of DALB is shown in Algorithm 2. The first step is that the controller takes load measurement and judge whether its value exceeds the threshold. If not, there is no need for load balancing. Otherwise it means this controller is overloaded and it sends message to collect load information from the other controllers. Then this controller will compute the maximum value of load list and the load balancing rate. If the rate is bigger than ρ or the load is not the max one, there is no need for load balancing. If not, it means this controller needs to migrate switch in order to balance load. This controller will call SwitchElection function and get the migrated switch's dpid (which identifies the switch). Then with the help of Zookeeper and HA-TCP, the chosen switch will be migrated to a low load controller. If this process is successful, it will return 1. Otherwise, it will throw exception. At last, the value of CT will be updated and written back to the distributed database.

Algorithm 2 DALB algorithm

Input:
$\{L_1, \ldots, L_n\}$: List of all controllers' load
CT: Load measurement threshold
ρ: Load balancing rate
Output:
0: no need for SDN controller load balancing
1: SDN controller load balancing success
-1: SDN controller load balancing fail

```
1: L_C = LoadMeasurement()
2: if L_C < CT then
3:    return 0
4: else
5:    {L_1,...,L_n} = LoadCollect()
6:    L_max = max({L_1,...,L_n})
7:    rate = (1/n Σ_{i=1}^{n} {L_i}) / L_max
8:    if rate > ρ or L_max > L_C then
9:       return 0
10:   else
11:      dpid = SwitchElection()
12:      SwitchMigration(dpid)
13:      if success then
14:         return 1
15:      else
16:         Throw Exception
17:         return -1
18:      end if
19:   end if
20:   CT = AdaptiveCT({L_1,...,L_n})
21:end if
```

V. IMPLEMENTATION AND EVALUATION

We implement the distributed SDN controller based on Floodlight. Floodlight is a Java-based OpenFlow controller forked from the Beacon SDN controller developed by Stanford. This controller is an open-source Software Apache-licensed, supported by a community of developers. It offers a modular architecture and it's easy to extend and enhance. We add the Load Measurement module, Load Collection module and Decision Maker module. We make use of Cassandra [12] as the distributed database in our system. Cassandra is a distributed NoSQL database. It supports scalability and high availability. We store switches' specific information, CT and ρ in Cassandra. In order to evaluate the controller's performance, we use Cbench [13] software. Cbench is a performance measurement tool designed for benchmarking OpenFlow controllers. It can simulate packet-in requests of OpenFlow switches. Besides, we use mininet [14] to emulator SDN network topology.

A. Distributed SDN controller throughput

First, we use Cbench to test our distributed SDN controllers based on Floodlight. In the benchmarks, we configure Cbench with 16 tests, each lasting 10 seconds, the number of switches is 32, the cooldown and warmup loop are 1. Our prototype is deployed on X86 server, which contain 4 physical cores from Intel Xeon 5160 processors. The operating system is 64-bit Ubuntu 12.04 OS.

The throughput of our distributed SDN controller is shown in Fig. 5. As Fig. 5 illustrates, when the number of controller nodes becomes double, the throughput doesn't become twice. During the test, Cbench will emulate a lot of packet-in messages sent to controller. This will trigger our load balancing

module and this will affect the performance of controller. Therefore, the throughput will slightly decrease.

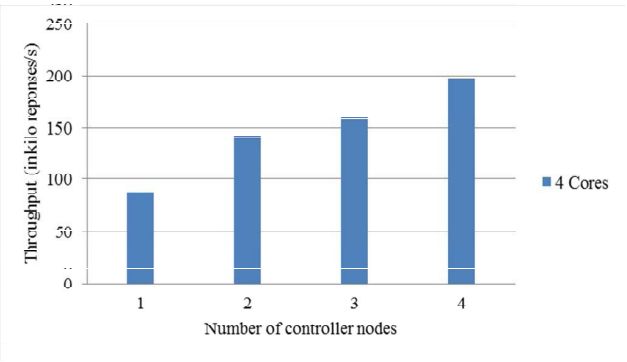

Fig. 5. Distributed Controllers' Throughput

B. *Load Balancing Strategy Evaluation*

We use two controller instances to deploy a distributed SDN network. It's hard for us to emulate the change of packet-in arrival rate, so we use different numbers of switches to emulate the load. Each switch sends packet-in message continuously. Controller A connects 8 switches and controller B connects 11 switches. We assign CT's initial value to 1200. The DALB algorithm test result is shown in Fig. 6.

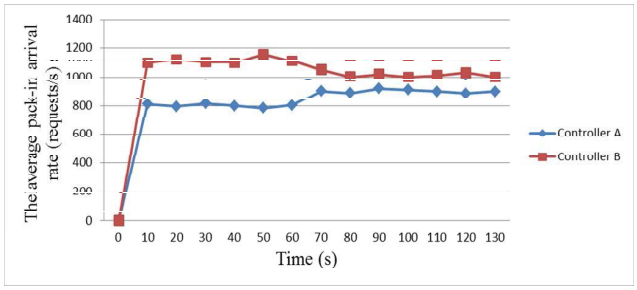

Fig. 6. DALB Test Result

As Fig. 6 shows, during the time between 0s and 60s, the load of controller A and B is smaller than CT. At the time of 70s, we run our program to modify the CT value in Cassandra to 1000. Then the load balancing module will detect the load of controller B exceeds the threshold CT and will run DALB algorithm. So a switch needs to be migrated to controller A. At last, the load will be balanced in the whole system. Meanwhile, the AdaptiveCT algorithm will be called and the calculation result is 950, which smaller than CT. As a result, the CT's value is still 1000 according to AdaptiveCT.

VI. CONCLUSION AND FUTURE WORK

In this paper, we put forward a load balancing strategy named DALB for SDN controller based on distributed decision. We describe every component of DALB and test the performance of our distributed SDN controllers and test the DALB algorithm function. In the future, we will focus our attention on the testing our algorithm in more detail. Besides, if we choose different values of CT and ρ, different performance of our algorithm will display. It's very important for us to do research on how to select the value of CT and ρ in order to get the optimal performance.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G.Parulkar, et al., "Openflow: enabling innovation in campus networks," SIGCOMM CCR, 2008.

[2] M. Casado, M. J. Freedman, and S. Shenker, "Ethane: Taking Control of the Enterprise," in ACM SIGCOMM, 2007.

[3] A. Greenberg, G. Hjalmtysson, D. A. Maltz, et al., "A clean slate 4D approach to network control and management," in SIGCOMM CCR, 2005.

[4] T. Koponen et al., "Onix: A Distributed Control Platform for Large-scale Production Networks," in OSDI, 2010.

[5] A. Tootoonchian and Y. ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in INM/WREN, 2010.

[6] OpenDaylight. http://www.opendaylight.org/[M].

[7] Dixit A, Hao F, Mukherjee S, et al., "Towards and Elastic Distributed Sdn Controller," in ACM SIGCOMM, 2013.

[8] Yannan Hu, Wendong Wang, Xiangyang Gong, et al., "Balanceflow: Controller Load Balancing For Openflow Networks," in IEEE CCIS, 2012.

[9] Hunt P, Konar M, Junqueira F P, et al., Zookeeper: wait-free coordination for internet-scale systems[c], Proceedings of the 2010 USENIX conference on USENIX annual technical conference. 2010, 8: 11-11.

[10] JGroups. http://www.jgroups.org/[M].

[11] Floodlight. http://www.projectfloodlight.org/floodlight[M].

[12] Cassandra. http://cassandra.apache.org/[M].

[13] Cbench. http://sourceforge.net/apps/trac/cbench/[M].

[14] Mininet. http://mininet.org/[M].