Load Balancing for Multiple Controllers in SDN Based on Switches Group

Yaning Zhou, Ying Wang, Jinke Yu, Junhua Ba, Shilei Zhang
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: {zhouyaning, wangy}@bupt.edu.cn

***Abstract*—Software-Defined Networking (SDN) develops a centralized control plane to manage the whole network, but the traditional centralized control plane suffers from the issues of reliability and scalability. Although some methods solves the issues, they do not balance load among controllers. In this paper, we propose a load balancing mechanism based on switches group for multiple controllers. The mechanism not only balances the load among controllers, but also solves the load oscillation and improves time efficiency. Experiments based on floodlight show that our mechanism can improve network balancing and the time efficiency, at same time, avoid load oscillation problem among controllers.**

Keywords—Software-Defined Networking; Load balancing; Network balancing; Time efficiency; Switches group;

I. INTRODUCTION

Software-Defined Networking (SDN) is a new network technology in modern Internet. In the SDN architecture, the control plane is decoupled from the data plane by moving networking control functions from forwarding devices to a logical centralized controller. However, there are big challenges for the scalability and reliability of the single and centralized controller, due to business needs and network scales. To solve the issues, several researches[1,2,3] have implemented deployment solutions of multiple SDN controllers, however, these designs assume mapping between a switch and a controller is statically configured, making it difficult for the control plane to adapt to imbalances caused by spatial and temporal variations in traffic conditions. Under such conditions, once a controller becomes overloaded, the response time for the control plane messages may exceed required time, which will impact network performance. Therefore, it is important to handle the issue of uneven load distribution among multiple controllers.

Although existing methods based on switch migration strategy can solve the uneven load distribution in multiple controllers, some of these methods don't consider the network balancing. The network balancing directly affects network performance, thus, improving balancing is important for the network. Moreover, load oscillation problem caused by inappropriate switch migration may result in switch migration frequently, which not only increases the network overhead, but also debases network balancing, and then impact network performance. Therefore, load oscillation problem cannot be ignored in load balancing.

In addition, the time of load balancing for an overloaded controller determines when the controller functions can return to normal, and then affects the network performance. If the time could be reduce, then it can improve the network performance.

Based on the above analysis, In order to optimize network balancing and time efficiency of load balancing, we propose a load balancing scheme based on switches group for multiple-controllers architecture. In the scheme, a switches selection algorithm and a target controllers selection algorithm are put forward to solve the load oscillation problem and improve the time efficiency.

II. RELATED WORK

Numerous previous works [4,5,6] have been done in the area of load balancing based on switch migration. These methods solve the load balancing from three procedures: 1) migrated switch selection. 2) target controller selection. 3) switch migration.

The inappropriate selection of migrated switch and target controller will cause the load oscillation problem. To solve the load oscillation problem among controller, papers [4] proposed switch selection methods. However, these methods only consider the load of overloaded and target controller, which cannot make the best of controller resources. Chu Liang et al. also put forward a method to solve the load oscillation problem in paper [5], but they only consider the round-trip time in method, and omit the load of switches and controllers, which does not help improve the network balancing.

To improve the network balancing, the authors of [6] proposed ElastiCon based on the switch migration strategy. The ElastiCon periodically balances the load of controllers in pool. Although the method ensures good performance at all times irrespective of the traffic dynamics, which causes a high network overhead.

From the point of time efficiency, the methods of papers [4,5,6] choose a switch to migrate in a balancing process, which do not make the overloaded controller become normal timely, thus impact the network performance.

III. PROBLEM STATEMENT

In this section, we describe the problem of load oscillation among controllers and time efficiency in detail.

1) Load oscillation problem

Here, we give an example in Fig.1 to illustrate the problem of load oscillation. As can be seen in the figure, the packets arrival rate of controller1 is 8000pps, which exceeds the threshold value 6000pps. The number of packets received from each switch of which that controller1 is the master is 2000pps, 500pps, 500pps and 5000pps respectively. If we choose the switch with the highest packet request rate to migrate, and the packets arrival rate of target controller is 1000pps.Then it may make the target controllers become overloaded quickly. In this case, there will be a problem of load oscillation among controllers.

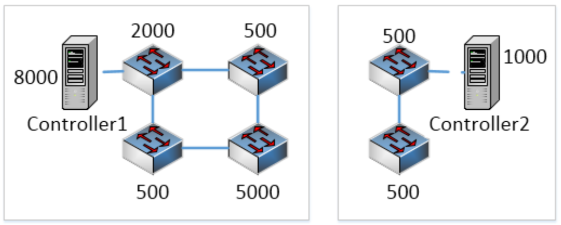

Fig.1. Examples demonstrating load oscillation problem

2) Time efficiency problem

Some existing methods select one switch to migrate in a balancing process, and if the migration don't make the overloaded controller balanced, then the overloaded controller would enter next balancing process until the controller restores normal. Under these circumstances, the overloaded controller will cost two or more balancing processes to balance the load. For example, the packet request rate of every switch in Fig.1 is 2000pps, other parameters remain unchanged. If mechanism selects a switch to migrate, we find that it will take two balancing processes until the overloaded controller become balanced, which could not balance the load quickly, and also reduce time efficiency.

IV. DESIGN AND IMPLEMENTATION

In this section, we propose methods to solve the problems proposed above. Then, we put forward the load balancing scheme based on the methods.

In this paper, we adopt logically-centralized control plane as multiple-controllers architecture. There is a super controller and multiple local controllers in the architecture. The relationship of controllers and switches is many-to-many, which is supported by OpenFlow 1.3[7] or its higher version. The super controller maintains a TCP channel with every local controller, with which super controller and local controllers exchange load information and decision results. We assume that the controllers communicate with the switches by in-band mode, which means that the control messages and data messages are transmitted via the same channel. Thus, the distance between controller and switch could be measured by the number of hops in the path from controller to switch.

A. Load Measurement

The load measurement component runs on every local controller to measure the load of controller and switches that mapped to the controller, and periodically inform load information to the super controller. In the SDN network, the load of processing the *PACKET_IN* events is generally regarded as the most significant part of the total load of controller [9]. So in this paper, we use the total arrival rate of *PACKET_IN* events to denote the load of the controller, and sending rate of *PACKET_IN* events to represent the load of switch. Since controllers have different maximum load capabilities, so we use formula (1) to quantify the current load press of a controller.

$$P_i = C_i / C_{i,max} \tag{1}$$

Where C_i denotes the current load of controller i, $C_{i,max}$ represents the maximum load capability of controller i.

B. Balance Decision

The balance decision component makes balancing decision according to the data which are informed by load measurement.

1) Switches selection

In section III, we introduce the situation of load oscillation among controllers. The reason for the situation is that the switch selection algorithm does not consider the network load status, and only choose a switch based on partial load. We also introduce the circumstances that an overloaded takes multiple balancing processes to balance the load. The reason for the circumstances is that switch selection algorithm only thinks about the load of switches, and neglects the load of overloaded controller. Thus, to shorten the balancing times to one time, we take the load of overloaded controller and switches into account in switches selection.

The procedure of switches selection algorithm is shown as Algorithm 1. Taking the overloaded controller as input. A summary of the symbols in paper is shown in TABLE I.

TABLE I

The summary of the symbols used in paper.

Symbol	Definition
S^{OVER}	The set of switches by overloaded controller control.
S^{M}	The set of switches to be migrated.
N	The number of local controllers.
S_i	The load of switch *i* in the network.
C_i	The load of controller *i* in the network.
C_{over}	The load of overloaded controller.
C_{sla}	The load of slave controller.
S_{mig}	The load of switch to be migrated.
$C_{i,max}$	The maximum load capability of controller *i*.
Sum_i	The sum of switches load in set *i*: $Sum_i = \sum_{k=1}^{m} S_k$.
$d_{i,j}$	The hops between switch *i* to controller *j*.
$D_{i,j}$	The distance between switches group *i* to controller *j* $D_{ij} = \sum_{k=1}^{m} d_{kj}$,m is the switch number in set *i*.
CR_i	The criterion set of slave controllers belong switch *i*.
Thr	The local controller threshold in the network.
TC	The set of target controllers for switches group.

The switches selection algorithm selects a switches group whose load is the nearest to the minus of two load, the load of overloaded controller and network average, which makes load of the overloaded controller be closed to the network average after migration is completed. In general, the control plane latency can be estimated with distance between switch and controller, and control plane latency controlling the nearer switch is short than the further one. Thus, to reduce the control plane latency, the algorithm chooses the furthest switches groupwith the overloaded controller to migrate when there are multiple groups whose load are closed to the *Target* value equally. For example, the overloaded controller controls 5 switches at present. The switch sending rate of *PACKET_IN* events is 700pps, 800pps, 200pps, 500pps and 3000pps respectively, and the distance between switch to the overloaded controller is 6 hops, 4 hops, 2 hops, 3 hops and 2 hops. If the *Target* value is 1500pps, then we will find that there are two groups whose sum of load are 1500pps, they are G_1 (700,800) and G_2 (800,200,500). Under the above circumstance, the algorithm will choose $G1$ because the D_{G1} is 10 hops, which is greater than 9 hops of D_{G2}.

Algorithm 1 the Switches_Selection algorithm

```
Input: controller over
Output: $S^M$
Procedure:
1:  $C_{ave} \leftarrow \frac{1}{N}\sum_{i=1}^{N} C_i$ , $n \leftarrow S^{OVER}$.length
2:  Target $\leftarrow C_{over} - C_{ave}$
3:  for k in n
4:  C=CombineAlgorithm ($S^{OVER}$,k)
5:  Add C into S
6:  end for
7:  for i in S.length
8:    calculate $Sum_i$
9:      $dif[i] \leftarrow Sum_i - Target$
10: end for
11: Select $S_i$ in S with min dif[i] into $S^E$
12: if ($S^E$.length>1)
13: for i in $S^E$.length
14:   calculate $D_{i,over}$ of $S_i$
15: end for
16:     select $S_i$ with max $D_{i,over}$
17:   end if
18: $S^M \leftarrow S^i$
```

2) *Selection of target controller for switches group*

To avoid the load oscillation among controllers and improve the network balancing, we also make the load of target controller be near to the network average load as much as possible. Furthermore, we have considered migration overhead and control plane latency in the selection. The migration overhead can be estimate with distance between controller and switch in general, so we add the distance between slave controller and switch in formula. The criterion of slave controller j for switch i is calculated by formula (2):

$$CR_{i,j} = w_1 \times |C_{sla} + S_{mig} - C_{ave}| + w_2 \times d_{i,j} \qquad (2)$$

Where C_{sla} , S_{mig}, C_{ave} and $d_{i,j}$ are explained in table I, Both w_1 and w_2 are weight coefficients, the sum of them is 1.0. The criterion is smaller, then the slave controller is more appropriate to accept the switch.

There will have a migration conflict in target controller selection for a switches group, if there are some switches whose target controller are same, which may result in the target controller become overloaded. Therefore, to prevent the occurrence of the problem, we propose a target controllers selection algorithm for switches group. The procedure of the algorithm is shown as Algorithm 2. The output of the algorithm is the target controllers set TC of switches group.

Algorithm 2 the Target Controllers Selection algorithm

```
Input: $S^M$
Output: $TC$
 Procedure:
 1:  Num1 $\leftarrow S^M$.length, Num2 $\leftarrow C^{Slave}$.length
 2:    for i in Num1
 3:      for j in Num2
 4:  Compute the weights $CR_{i,j}$ slave controller j by formula (1)
 5:        end for
 6:        sort $C^{i,Slave}$ by $CR_i$ ascending
 7:    end for
 8:  sort switch in $S^M$ by load ascending
 9:    for i in Num1
10:     for j in Num2
11:       sum=$S_i$+$C_j$
12:     if (sum/$C_{j,max}$>$C_{ave}$)
13:       continue
14:     else $TC_i$=j
15:          $C_j$ +=sum
16:     end if
17:     end for
18:   end for
```

V. Performance Evaluation

A. Experimental Environment

In this section, we implement the logically-centralized *OpenFlow* controller based on *Floodlight* [11]. We use *Cbench* [10] tool to measure the maximum rate in which flow requests are handled by controller. Besides, we choose *Mininet* [12] to emulate a network of software-based virtual *OpenFlow* switch as our experimental testbed. Here, the virtual *OpenFlow* switch adopts *Open vSwitch* [13]. It supports *OpenFlow* as a method of exporting remote access to control traffic. In our test, we use a super controller and five local controllers to deploy a logically-centralized SDN network topology. There are 20 switches, and they connect to each local controller respectively. Every local controller connects to four switches as master and eight switches as slaves. In order to simulate the different arrival rate of *PACKET_IN* message, we adjust the sending rate of flows of end-hosts by *Hping* [14] tool.

In this section, we mainly compare four schemes: in scheme I, switch-controller mapping keeps static; scheme II adopts Elasticon [6]; scheme III adopts method in [4]; scheme IV is the mechanism based on switches group in this paper. We use *Cbench* tool to measure the maximum rate in which *PACKET_IN* message are handled by Floodlight based on our physical hardware. The result is that the average rate is 12518 *PACKET_IN* messages per second. Therefore, in our experience, we set the threshold value of local controllers is 8000pps.

B. Network balancing

We compare the load balancing scheme based on the switches group with scheme III from the perspective of network balancing. We use standard deviation of controllers resource utilization to estimate network balancing, as defined by (3).

$$var = \frac{1}{M}\sum_{i=1}^{M}(u_i - \bar{u})^2 \qquad (3)$$

Where u_i represents the resource utilization of controller i, and $\bar{u}$ stands for the resource utilization average of all local controller. This metrics reflects whether there is load oscillation among controllers and network balancing.

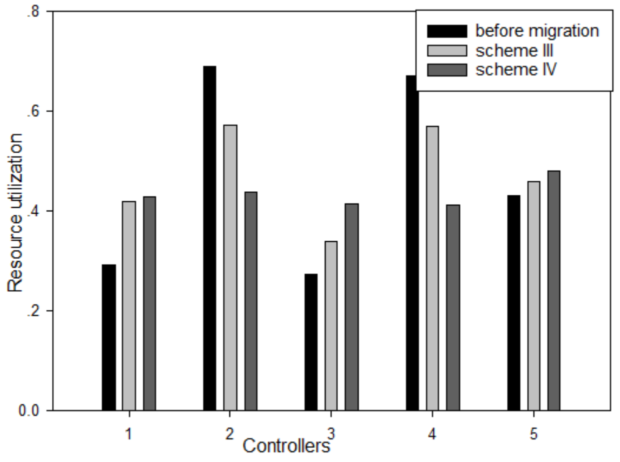

Fig.2. The utilization of controllers using different schemes

The Fig.2 shows the resource utilization of controllers before and after balancing. It can be observed that scheme IV not only can balance the overloaded controller, but also make the network balancing is higher than the scheme III.

C. Completion time

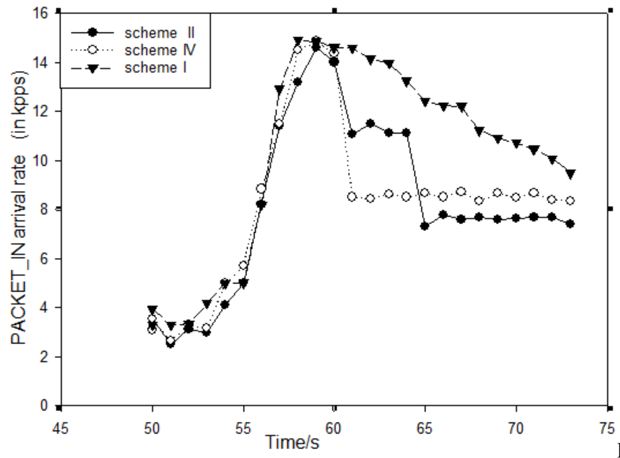

Fig.3. The completion time of load balancing with different schemes

We compare scheme IV with scheme I and II from the perspective of the completion time between overloaded and rebalancing status. In order to verify that the scheme IV can restore the overloaded controller in a balancing process, the sending rate of flows is increased by the *Hping* tool so that the load of Controller2 exceeds the threshold, and at least two switches are migrated to complete the balancing. As Fig.3 shows, the three curves represent the load of controller2 in scheme I, II and IV. From 0s to 56s, the load of controller2 is smaller than threshold value. At the time of 56s, we give commands to end-hosts which are connected to the switches of controller2 to increase the sending rate of flows. The load of controller2 exceeds its threshold between 56s and 57s. In II, it needs to take two load balancing processes, the two balancing are completed at 61s and 65s respectively. In scheme IV, two switches are migrated at the same time in a load balancing process, the migration is completed at 61s. In scheme I, the reduction of load in controller1 is a slow process, its load is normal at 71s. Therefore, the scheme of switches group can optimize the time efficiency.

VI. Conclusion

In this paper, we investigate the load balancing problem of multiple controllers in SDN, then we propose the load balancing scheme based on switches group. To solve the issue of load oscillation among controllers and improve network balancing, we put forward a switches selection algorithm and a target controller selection formula, which ensure that performance remains stable even under highly dynamic traffic conditions. In addition, to improve the time efficiency, we also design a target controllers selection algorithm for a switches group, which insures a switches group can be migrated in a balancing process. Simulations show that our approach can solve the load oscillation problem, and improve network balancing in a time efficient way.

ACKNOWLEDGMENTS

This work was supported by the National High Technology Research and Development Program of China (2013AA013502) and the National Natural Science Foundation of China (61501044).

References

[1] "OpenDaylight," http://www.opendaylight.org/.

[2] Koponen, Teemu, et al. Onix: a distributed control platform for large-scale production networks. 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings 2010:351-364.

[3] Tootoonchian, Amin, and Y. Ganjali. HyperFlow: a distributed control plane for OpenFlow. Proceedings of the 2010 internet network management conference on Research on enterprise networking 2010:3-3.

[4] Yu J, Wang Y, Pei K, et al. A load balancing mechanism for multiple SDN controllers based on load informing strategy[C]// Network Operations and Management Symposium. IEEE, 2016.

[5] Liang C, Kawashima R, Matsuo H. Scalable and Crash-Tolerant Load Balancing Based on Switch Migration for Multiple Open Flow Controllers[C]// Second International Symposium on Computing and NETWORKING. IEEE Computer Society, 2014:171-177.

[6] Dixit, Advait Abhay, et al. ElastiCon: an elastic distributed sdn controller. Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems ACM, 2014:17-28.

[7] Open Networking Foundation “OpenFlow Switch Specification Version 1.3.3 (Protocol version Ox04) ” July 27 2013.

[8] Cheng G, Chen H, Wang Z, et al. DHA: Distributed decisions on the switch migration toward a scalable SDN control plane[C]// Ifip NETWORKING Conference. IFIP, 2015.

[9] YAO G, BI J, LI Y, et al. On the capacitated controller placement problem in software defined networks [J]. IEEE Communications Letters, 2014, 18(8) : 1339 - 1342.

[10] “Cbench,” http://sourceforge.net/projects/cbench/ [Online; accessed January 17, 2015].

[11] “Floodlight,” http://www.projectfloodlight.org/floodlight/, [Online; accessed June 04, 2015]

[12] “mininet,” http://mininet.org/ [Online; accessed June 02 2015].

[13] “Open vSwitch,” http://openvswitch.org [Online; accessed September 11 2015].

[14] “Hping,” http://hping.org/ [Online; accessed December 24 2015].