A Load Balancing Mechanism for multiple SDN Controllers based on Load Informing Strategy

Jinke Yu, Ying Wang, Keke Pei, Shujuan Zhang, Jiacong Li
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: {zjjyjk, wangy}@bupt.edu.cn

***Abstract*—Software defined networking (SDN) is currently regarded as one of the most promising paradigms of future Internet. Although the availability and scalability that a single and centralized controller suffers from could be alleviated by using multiple controllers, there lacks a flexible mechanism to balance load among controllers. This paper proposes a load balancing mechanism based on a load informing strategy for multiple distributed controllers. With the mechanism, a controller can make load balancing decision locally as rapidly as possible. Experiments based on floodlight show that our mechanism can balance the load of each controller dynamically and reduce the time of load balancing.**

Keywords—Future Internet; Load balancing; Load informing; Software defined network (SDN)

I. INTRODUCTION

Software defined networking (SDN) has emerged as a new and promising paradigm shifting from traditional network to the Future Internet to offer programmability and easier management[1]. In SDN, a centralized control plane brings many benefits such as controlling the network by a central node and abstracting the underlying network infrastructure from the applications. However, the single and centralized controller imposes potential issues of scalability and reliability. Hence some research works have designed the deployments of multiple controllers to avoid this bottleneck. Although these solutions can settle the issue, one key limitation is inevitable for these solutions: it is hard for the control plane to make an adaptation to uneven load distribution, when the mapping between a switch and a controller is statically configured. The limitation will lead to degraded network performance. So it is important to handle the issue.

Currently, the research work about load balancing decisions of multiple controllers can be divided into two categories: the centralized decision and the distributed decision. For the centralized decision [2, 3, 4], there are two essential processes: one is collecting load information of all local controllers and the other is sending load balancing commands to the local overloaded controller. Due to these two processes, the time efficiency of load balancing is not high. For the distributed decision [5], every controller can make balance decision locally, so the process of sending load balancing commands can be omitted. However, the existing distributed decision method need to collect load information of other controllers before an overloaded controller can make decision. This process extends the completion time of load balancing.

In order to make an overloaded controller balanced as quickly as possible, we propose a load balancing mechanism. It adopts the distributed decision. Meanwhile, it is based on a load informing strategy, namely every controller periodically actively informs its load information. And it also processes and stores load information informed by other controllers. Moreover, for reducing the overhead of communication and processing messages caused by the strategy, an inhibition algorithm is put forward to lower the frequency of informing.

II. RELATED WORK

The load balancing decisions of multiple controllers has been studied in [2, 3, 4, 5], which can be divided into two categories: the centralized decision [2, 3, 4] and the distributed decision [5].

The authors of [2, 3] propose ElastiCon, which includes a logically centralized load adapter responsible for balancing load among controllers in a controller pool. In the deployment of [4], there is a coordinator controller to be responsible for maintaining a global controller load info table. According to the table, the controller decides whether to balance the load among controllers. The centralized decision [2, 3, 4] can settle the issue of uneven load distribution among controllers. However, they two essential processes, namely collecting load information of all controllers and sending load balancing commands to the overloaded controller. These two processes extend the time of load balancing.

The authors of [5] propose DALB, which allows every controller can make load balancing decision locally. For an overloaded controller, before it makes balance decisions locally, it will collect load information of all other controllers. However, due to the process of collecting, the time efficiency of load balancing is not high.

To reduce the delay caused by these processes, we put forward a load balancing mechanism. Firstly, the mechanism allows controllers can make decisions locally. Secondly, different from the above process of collecting in DALB [8], we propose a load informing strategy: each controller periodically actively reports its load information to other controllers. It also handles and stores load information informed by other controllers. So an overloaded controller no longer collects all other controllers' load information before make decisions locally. Thirdly, an inhibition algorithm is proposed to lower the frequency of load informing for reducing the processing and communication overhead caused by the informing strategy.

This work was supported by National Natural Science Foundation of China (61501044).

III. ARCHITECTURE OVERVIEW

The architecture of our proposed distributed decision is shown in Fig. 1. In this architecture, the relationship of controllers and switches is many-to-many, which is supported by OpenFlow 1.3 [6] or its higher version. The communication and coordination of control plane is based on JGroups [7]. Our proposed load balancing mechanism is running as a module of each SDN controller in this architecture, named as load balancing module. This module includes four components: (1) load measurement is used for measuring load metrics and judging whether the load of a controller exceeds the threshold, (2) load informing is responsible for a local controller sending its load information to other controllers, (3) balance decision is in charge of making load balancing decisions, (4) switch migration is liable for shifting the selected switch to balance the load among controllers.

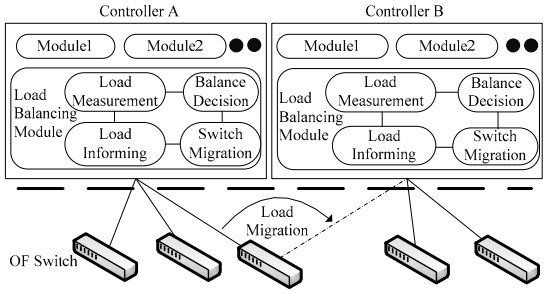

Fig. 1. The architecture of the proposed distributed decision

The load balancing module on each controller cooperates with one another to balance the controller load by running the above four components. Firstly, the load measurement component periodically measures load metrics and checks whether the controller's load exceeds the predefined threshold. Secondly, if the load is under the threshold, the load informing component will judge whether the controller needs to inform its load information to other controllers. If not, the balance decision component makes load balancing decision locally, like selecting appropriate switches to migrate and choosing light-loaded controllers as target controllers to accept selected switches. Thirdly, the target controller determines whether to accept these switches. If so, these switches are migrated to their target controllers. Finally, after completing the migration, the controller updates its load information and reports it to other controllers via the load informing component.

IV. DESIGN AND IMPLEMENTATION

A. Load Measurement component

The load measurement component runs on each controller to periodically measure load information. In this paper, we choose two metrics for load balancing decision. They are the average message arrival rate (I) from each switch and the round-trip time (R) from each switch to controller. CPU is typically the throughput bottleneck of a controller, and the CPU load is roughly in proportion to the message arrival rate. As a result, we calculate the total message arrival rate to represent the load of a controller. Besides, since the round-trip time is an important factor to evaluate the performance of control path, it will be considered as one parameter when selecting the target controller. Here, we assume that the connection between a controller and a switch is in the in-band mode, so the round-trip time R could be measured by the number of hops from the controller to switch.

B. Load Informing component

With the proposed load informing strategy, each controller can periodically actively reports its load information to other controllers. And it also handles and stores the load information from others. While the periodical active load informing can decrease the decision delay, it also causes additional processing and communication overhead in the control plane. Especially, when the current load value does not change much compared to the last value, reporting it to other controllers is a redundant operation. To reduce these overheads, we put forward an inhibition algorithm to reduce the frequency of load information notification. This algorithm is outlined in Algorithm 1.

Algorithm 1 the inhibition algorithm

Input:

$\boldsymbol{L_{Current}}$: Current load value

$\boldsymbol{L_{Former}}$: Former load value

$\{\boldsymbol{V_0 = 0, V_1, V_2, \dots, V_7, \dots, V_n = Threshold}\}$, and $\boldsymbol{V_m - V_{m-1} < V_{m-1} - V_{m-2}, m \geq 2, n \geq 7}$, n is an odd number: the number of segments

Output:

True or **False**: Informing load information

1: Value = **False**
2: if($\boldsymbol{L_{Current} < Threshold}$) then
3: for ($\mathbf{i}: \mathbf{1 \rightarrow n-1}$)
4: if($\boldsymbol{L_{Current} \geq V_i}$&&$\boldsymbol{L_{Former} < V_i}$)||($\boldsymbol{L_{Current} < V_i}$&&$\boldsymbol{L_{Former} \geq V_i}$))
5: Value = **True**
6: break
7: end if
8: end for
9: $\boldsymbol{L_{Former} = L_{Current}}$
10: end if
11: return Value

In this algorithm, the current load value of a controller collected by the load measurement component is denoted as $\boldsymbol{L_{Current}}$ and the load value measured last time is denoted as $\boldsymbol{L_{Former}}$. Meanwhile, we divide the scope from 0 to the threshold value into several segments, such as the segment of V_1 to V_2. When $\boldsymbol{L_{Current}}$ and $\boldsymbol{L_{Former}}$ are in the same segment, namely: $V_i \leq \boldsymbol{L_{Former}, L_{Current}} \leq V_{i+1}$, a controller will not inform its load information to other controllers. So the controller does not always report its load information after periodically collecting load metrics. However, such inhibition algorithm may lead to a deviation between the current load value and the load value stored by other controllers. If the load of a controller is low, when the controller is selected as the target controller to accept some load of a high-load controller, the deviation has no significant impact on this controller because the controller has enough capacity to accept the migrated load. However if the load of a controller is high, the deviation can cause the controller become a new heavy-loaded

controller after accepting the migrated load. In order to avoid such situation, we make the lengths of the segments decrease progressively, namely: $\boldsymbol{V_m - V_{m-1} < V_{m-1} - V_{m-2}, m \geq 2}$. In such way, a controller whose load is close to its threshold will release its load information more frequently to reduce the deviation.

C. Balance Decision component

The balance decision component firstly judges whether an overloaded controller is the heaviest overloaded controller among all controllers. Then it decides which switches should be selected to be migrated and which controllers should be selected as the target controllers to accept the chosen switches.

1) *The heaviest overloaded controller judgment*

In the process of load balancing, another problem is inevitable. If two or more controllers exceed their load thresholds, they will take migration operation simultaneously. If these overloaded controllers select one same controller as the target controller to accept the chosen switches, the controller may become a new overloaded controller. To settle the above issue, we come up with selecting the heaviest overloaded controllers among overloaded controllers to be balanced during each load balance cycle. For the selection of the heaviest overloaded controller, we propose the following formula (1) named the overload proportion formula. When the overloaded proportion of a controller is the biggest, the controller is regarded as the heaviest overload controller. If there are several such controllers, we determine that the controller with the maximum current load is the heaviest. If these two values are still equal for several controllers, we will choose one of them randomly.

$$P_{overLoad} = \frac{L_{Current} - Thr}{Thr} \quad (1)$$

$L_{Current}$ denotes the current load value of a controller and Thr is the load threshold of the controller. $P_{overLoad}$ is the overload proportion of an overloaded controller.

2) *Switch selection*

After selecting the heaviest overload controller, we choose switches to be migrated. From part A, we can obtain the average message arrival rate (I) from each switch. The bigger the average message arrival rate is, the switch brings more load to its controller. In order to release the load of the overloaded controller as fast as possible, we preferentially select the switch with high message arrival rate. If one selected switch with high arrival rate can reduce the load of the controller to be under the threshold, the switch selection is finished. If not, the above switch groups with another switch with high arrival rate as the migrated switches and so on. So we sort the switches controlled by the overloaded controller in descending order with their message arrival rate.

$$G_{Switch} = \text{sort}\{I_{ID}\} \quad (2)$$

I_{ID} is the average message arrival rate of the switch with ID. G_{Switch} is a switch set sorted by I in descending order.

When choosing the switch, we also define a constraint formula (3). We ensure that the migrated load $L_{Migrate}$ is not more than $1/\alpha$ of the difference between load threshold Thr_{Target} of the target controller and the current load value L_{Target}.

$$L_{Migrate} \leq \frac{Thr_{Target} - L_{Target}}{\alpha} \quad (3)$$

We search the sorted switch set G_{Switch} to find one switch group satisfying constraint (3) to be migrated.

3) *Target controller selection*

A switch can connect to one master controller and several slave controllers. For each switch in the selected switch group, there is a possible situation that multiple slave controllers can accept it. It is desired that the lightest-load controller among these slave controllers is selected as the target controller to accept the switch. However, when the round-trip time between the target controller and the selected switch is the biggest among those from the selected switch to other slave controllers, the performance of control path is not high. So we consider both the load condition of a slave controller and the round-trip time from the selected switch to the controller, when we choose a target controller. The selection formula is as follow:

$$C_{Target} = w_1 \times \left(Thr_{Target} - L_{Target}\right) - w_2 \times R \quad (4)$$

R denotes the round-trip time. Both w_1 and w_2 are weight coefficients, and the sum of them is 1.0. C_{Target} is regarded as a criterion for selecting the target controller. The slave controller with the largest C_{Target} will be chosen as the target controller. If there are several such controllers, we will choose one of them randomly.

D. Switch migration component

When two or more controllers become overloaded at the same time, because of the load informing strategy, each high-load controller may judge itself the heaviest controller before receiving load messages of other controllers. And they may choose the same target controller. This may lead to migration conflict and the overload of the target controller. To avoid such situation, during a switch migration cycle, a target controller only accepts overloaded controllers' one switch migration request. When the target controller accepts a migration request, the process of the switch migration is as follow.

Firstly, the heaviest overloaded controller (C_A) triggers switch migration by sending a switch migration request message to the target controller (C_B) through JGroups. Then, C_B sends a ROLE_REQUEST message to the selected switch (S_0) for changing its role to equal. After S_0 replies the ROLE_REPLY message to C_B, C_B informs C_A that its role change is finished. When the completion of the role-change process, controller B can also receive asynchronous messages from S_0. Secondly, C_A cannot become the slave immediately from managed S_0 because there may be unfinished request at C_A before receiving the reply for migration from C_B. So C_A continues to interact with S_0 to complete undone work until it sends "end migration" to C_B. Thirdly, after receiving the end migration message, C_B changes its role from equal to master by sending a ROLE_REQUEST message to S_0. And S_0 sets C_A to slave. Finally, both controllers update the stored controller-

switch mapping synchronically. The whole switch migration process is completed.

V. EVALUATION

In this section, we implement the distributed OpenFlow controller based on Floodlight [8]. Our proposed load balancing mechanism runs as one of its modules. We choose Mininet [9] to emulate a network of software-based virtual OpenFlow switch as our experimental testbed. In our test, we use two controller nodes to deploy a distributed SDN network. We configure 4 switches to connect controller A as master and controller B as slave, meanwhile, another 4 switches to connect controller B as master and controller A as slave.

A. Throughput

We use Cbench [10] tool to measure the maximum rate in which Packet-In messages are handled by Floodlight based on our physical hardware. The result is an average rate of 12758 Packet-In messages per second (pps). In order to execute load balancing, at one time, we injected 5000 pps to controller A and 16000 pps to controller B. we plot the throughput of our proposed mechanism as illustrated in Fig. 2. Compared with our method, we also measure and plot the throughput when the switch-controller mapping keeps static.

As Fig. 2 shows, in static mapping model, the total average throughput of controllers is lower than that in the proposed mechanism under our workload (the total Packet-In messages injected by us per second). That because the variance of Packet-In requests leads to load imbalance among controller A and controller B, controller B is overloaded and Packet-In messages begin loss due to buffer overflow. In proposed mechanism, when controller B overloads, load balancing is triggered. Controller B dynamically shifts partial load to controller A. The load, imposed on controller A, is still below the processing capability of controller A, so the total throughput of controllers gets increased.

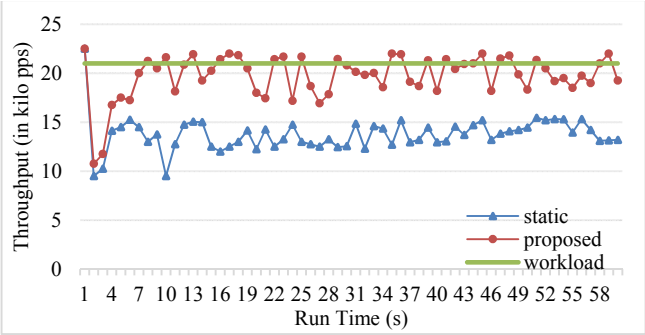

Fig. 2. Distributed Controllers' Throughput

B. Completion time

When we evaluate the completion time of our proposed mechanism at a load balancing cycle, we set the threshold values of controller A and controller B to 10000 pps and 11000 pps respectively. Once controller A or controller B exceeds the respective threshold, it needs to shift partial load to the other controller. We plot the result in Fig. 3.

As Fig. 3 shows, from 0s to 45s, the load of both controllers is smaller than the respective threshold values. At the time of 50s, we increase the Packet-In messages arrival rate of controller B. Then the load balancing module detects the load of controller B exceeds its threshold and makes a decision. So the selected switch is migrated to controller A. At 55s, the load of controller B comes down and controller A comes up. The load balancing is completed within 5s. This completion time is acceptable.

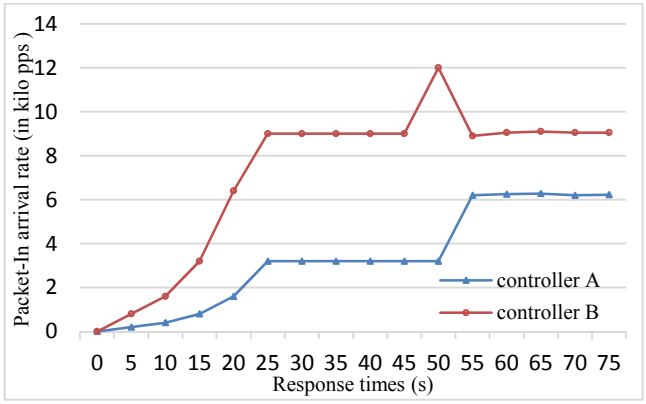

Fig. 3. Load balancing's completion time

VI. CONCLUSION AND FUTURE WORK

The uneven load distribution is an inevitable issue in deployment solutions of multiple controllers. To settle this issue, we propose a mechanism based on load informing strategy to balance the load among controllers and reduce the time of balancing. The results of evaluation demonstrated that our mechanism can achieve the above two aims. In the future, we will continue with the optimization of our proposed load balancing mechanism, with the focus on optimizing Load Informing component and Balance Decision component. In addition, we intend to implement our load balancing mechanism among multiple heterogeneous SDN controllers.

REFERENCES

[1] "The open networking foundation," https://www.opennetworking.org/, [Online; accessed May 16, 2015].

[2] Dixit, Advait, et al. "Towards an Elastic Distributed SDN Controller." Acm Sigcomm Computer Communication Review 43.4(2013):7-12.

[3] 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.

[4] Liang, Chu, R. Kawashima, and H. Matsuo. "Scalable and Crash-Tolerant Load Balancing Based on Switch Migration for Multiple Open Flow Controllers." Second International Symposium on Computing and Networking (CANDAR'14) 2014:171 - 177.

[5] Zhou, Yuanhao, et al. "A Load Balancing Strategy of SDN Controller Based on Distributed Decision." Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on IEEE, 2014:851-856.

[6] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.3 (Protocol version Ox04)," July 27, 2013.

[7] "JGroups," http://jgroups.org/, [Online; accessed January 02, 2016]

[8] "Floodlight," http://www.projectfloodlight.org/floodlight/, [Online; accessed June 04, 2015].

[9] "mininet," http://mininet.org/, [Online; accessed June 02, 2015].

[10] "Cbench," http://sourceforge.net/projects/cbench/, [Online; accessed January 17, 2016].