Evaluation of Load Balancing Mechanism for Multiple SDN Controllers based on Load Informing Strategy

Jinnapat Indrapiromkul[1], Loc Duy Phan[2] and Bivan Alzacky Harmanto[3]

***Abstract*— Software defined network (SDN) refers to network technology that aims at flexible network management. A multiple controller scheme has been proposed to avoid availability and reliability issues of a single controller. However, performance improvement is limited due to uneven load distribution. Therefore, a load balancing mechanism was suggested [1], [2] in order to optimize the network performance. However, the mechanism can cause controllers to produce unnecessary traffic among controllers when load of controller fluctuates. This paper analyzes the mechanism using real network traffic data to investigate the existence of such scenario. The result of the simulation showed that such fluctuating behavior of controller's load does exist. However, the frequency and duration of the fluctuation was not long enough to outweigh the benefit of Load Informing Strategy.**

I. Introduction

SDN allows network configuration from an application in higher layers as opposed to traditional approaches where network infrastructure is abstracted for above layers, restraining network requirements to be expressed. This solution resonates with currently emerging dynamic computing technology e.g. cloud computing, Internet of Things, etc. that requires customization to support monitoring or boost performance of a system.

SDN segments network layer into two planes, namely data plane and control plane. Data plane takes care of actual packet forwarding mechanism, while control plane is in charge of network configuration, routing table information and deciding on network traffic. Control plane is logically centralized as it keeps track of network state as a whole in order to manage low-level datapath according to application requirements. Consequently, network performance becomes dependent on that of an SDN controller. A multiple controller scheme has been proposed to avoid availability and reliability issues of a single controller. However, performance improvement is limited due to uneven load distribution. Load balancing mechanism is therefore suggested in order to optimize network performance. For a centralized approach, the decision is made by a super controller which collects load information from every controller and sending load balancing command back to an overloaded node. In a distributed approach, each controller collects load information from others and make a local decision. [2] proposed a dynamic and adaptive algorithm for controller load balancing (DALB), which was implemented as a module in SDN controller. [1], the process of load collection latency in responding to load fluctuation. Thus, [1] has suggested load informing strategy instead of load collection, with inhibition algorithm to regulate the frequency of informing messages. With this approach, every controller have a view of other's loads, so it can make decision quickly once it is overloaded.

We suspect the technique proposed in [1] can generate unnecessary traffic between controllers when controller's load fluctuates around a load segments' border. In order to confirm the existence of such scenario, we implemented the technique and ran a simulation with real traffic data retrieved from http://www.caida.org/data/. The result of the simulation showed that such fluctuating behavior of controller's load happens infrequently, with short duration.

II. Background on Load Informing Strategy

A. Dynamic and Adaptive Load Balancing algorithm

That algorithm runs as 3 modules inside the SDN controller. The first one is the load measurement, which is responsible to calculate the load for the local controller. The second one is the load collection, in which the load information will be gathered from all the other controllers in the network. The last one is the decision maker, which is responsible for deciding whether a controller's load exceeds the threshold, then deciding on which switch from that controller is to be switched to which controller. Only the controller with highest load can migrate in order to prevent switch migration from overloading the target controller. The switch selected for migration should also not overload the controller and hence, the following formula is applied in the process of selecting switch:

$$L_{Migrate} \leq \frac{Thr_{Target} - L_{Target}}{\alpha}$$

No migration should be done if all controllers' load is balance. The balance of the network is calculated as follow

$$\rho = \frac{L_{max}}{\sum_i L_i}$$

B. Load Balancing Mechanism based on Load Informing Strategy

Based on DALB, a similar algorithm is proposed in [1] where load collection procedure is replaced with a controller actively informing its load to other controllers. Instead of requests when the overload occurs, this algorithm inform the changes in load periodically (with the period to be determined in advance). Other than that, this load informing procedure also involves the inhibition algorithm (shown in

[1]Jinnapat Indrapiromkul, 20121097, School of Computing, KAIST
[2]Loc Duy Phan, 20184174, School of Computing, KAIST
[3]Bivan Alzacky Harmanto, 20174090, School of Computing, KAIST

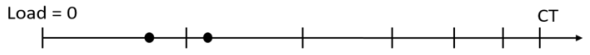

Fig. 1. Load Inhibition algorithm illustration

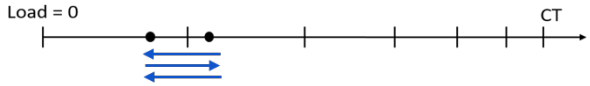

Fig. 2. Potential Problem with Load Fluctuation

II-B), to make sure that the load informing will only be done when it is necessary. To determine that, they divide the range between 0 and the controller threshold (CT) into several chunks. If the former load and the current load falls into the different intervals, then the load informing needs to be done. Otherwise we just need to adjust the information about the load for that particular controller.

Though, the condition is different between the low-load controller and the high-load controller. In the low-load controller, should it be chosen as the target controller at one moment, the amount of the new load after the migration will not have a significant impact as it still has lots of space to cope up with. However, the high-load controller will have not enough capacity should it be chosen as the target controller. To resolve this issue, the author made the length of the interval to be decreasing progressively as it reaches the threshold.

III. Load Fluctuating Problem

Load Informing Strategy [1] was proposed to resolve DALB's problem of massive traffic occurring during overload because of load collection. While we agree that this solution does resolve the problem, it has an issue of its own. Segments' size are fixed almost all the time (CT only changes when all controllers are overloaded). A controller whose load fluctuate around a segment border can cause unnecessary informing messages even if the range of load change is not large as shown in Figure 2.

This is merely an assumption and we do not know whether such scenarios happen in real world or whether the mechanism generates many unnecessary traffic when applied in the real network. Hence, in order to evaluate the mechanism and answer those questions, we will (1) implement the load balancing mechanism, and (2) evaluate the mechanism by simulating a software defined network using real traffic data.

IV. Implementation

Floodlight [5] was used as an SDN controller that supports OpenFlow [3]. Instead of 3 modules as proposed in [1], we combined Load Measurement and Load Informing modules to form our Load Collector Module (Figure 3). This module measure PACKET_IN messages per second for load measurement. Not only that, it also check from which switch each packet came from in order to know how much each switch contributes to the current load of the controller. The module contains 2 maps: one from switch to of switch and another one from controller to load of

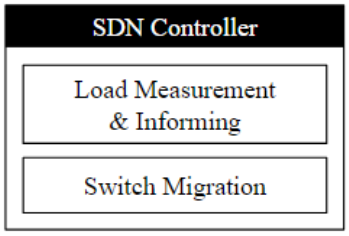

Fig. 3. Load Balancing Mechanism implemented as 2 modules in Floodlight

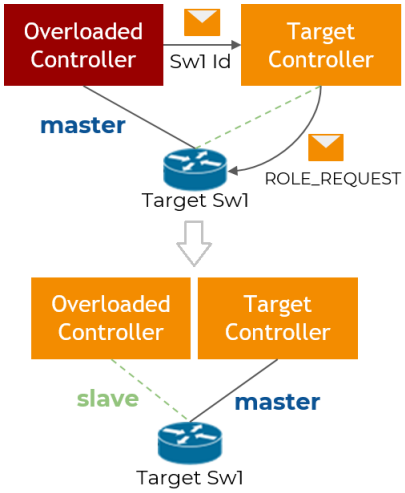

Fig. 4. Switch Migration illustration

controller. Whenever a controller informs its load to other controllers, the second map is updated accordingly. JGroups [6] is used as a communication channel between controllers so they can inform load as well as migrate switch.

When the controller load exceeds CT, the switch migration modules is triggered. Similar to proposed solutions, we only let the controller with the highest load migrate to prevent the target controller from overloading. We confirmed this using the map stored loads informed from other controllers in Load Measurement module.

Next, we choose the controller with lowest load stored in the map as the target controller. Recall that in Load Measurement we also store how much each switch contributes to the current load of the controller. Also, we assume that each switch is connected to all controller and there is no restriction on switch migration. Hence, we choose the target switch to migrate using the same formula as proposed solutions.

After that, the migrating controller will send target switch's id to the target controller via JGroups channel. The target controller, when receives the message, will send a ROLE_REQUEST packet to the switch requesting to change its role to MASTER. The switch will respond by changing the role of target switch to MASTER and role of migrating controller to SLAVE (Figure 4). However, in OpenFlow v1.3 [3], the switch will not inform the controller of the role change so the migrating controller is not aware of its role change to SLAVE. To handle this, we handle error message

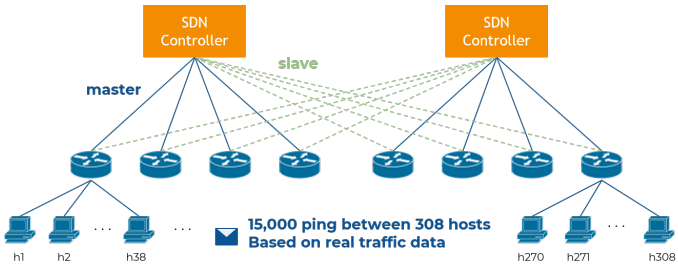

Fig. 5. Experiment setup: 2 controllers, 4 switches,308 hosts and 15000 pings

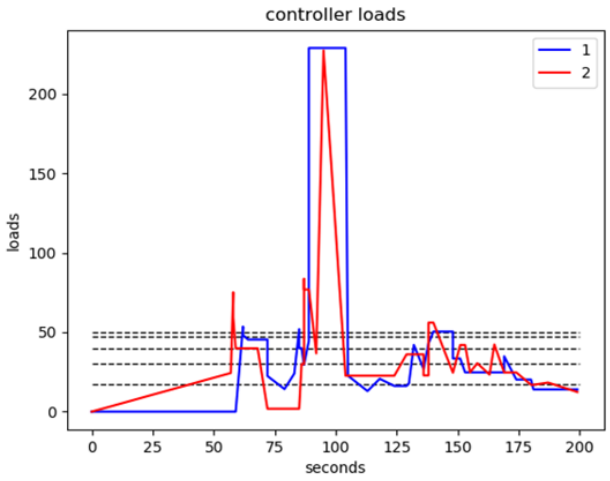

Fig. 6. Controllers' loads overtime, recorded when controller informs

from the switch with error code IS_SLAVE and change the role accordingly.

V. Experiment Setup

We used Mininet [4] to create a virtual SDN network containing 308 hosts and 8 Open VSwitch which is an OpenFlow v1.3 switch (Figure 5). In the network, one SDN switch is connected to all controllers, but only one controller acts as a master controller, and the other controllers are slaves. Only master controller is allowed to send FLOW_MOD packets and modify the flow table of a switch.

A real traffic data is obtained as pcap files through http://www.caida.org/data/, which then were filtered to contain 15,000 packets among 308 hosts. This was later used to observe the behavior of the virtual network. A Mininet script is created to simulate the traffic taking into consideration the time difference between each selected packets. We chose CT to be 50 pps and divide into 5 segments : [50, 47.5, 40, 30, 17.5, 0].

VI. Result and Discussion

Figure 6 shows the controllers' loads overtime, recorded whenever a controller informs its load. Figure 7 shows the graph within the range of controller threshold. There are many fluctuations of load in the graph. However, most of them have a long range and include more than 2 segments. The only two instances that matches our criteria can be

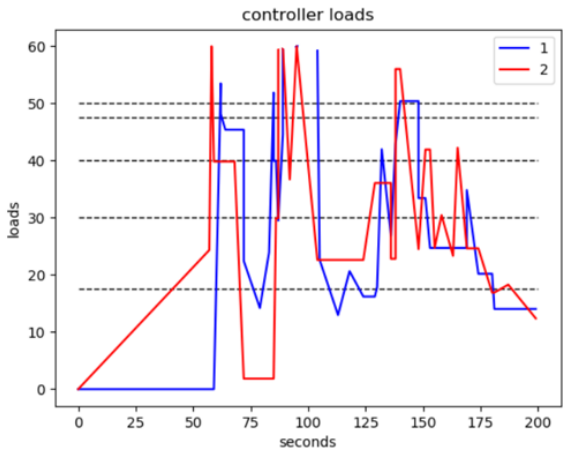

Fig. 7. Close-up view of 6

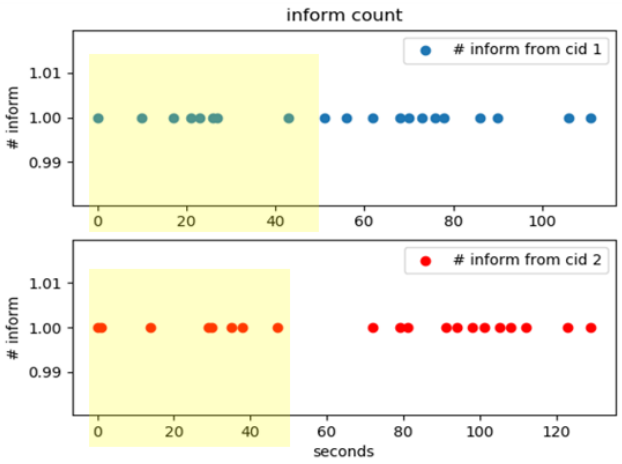

Fig. 8. Controller's informing messages over time

observed at controller 1 between 100s and 125s; and at controller 2 between 150s and 170s. The problem occurs scarcely and does not last long (i.e. crossing 2 segments only 3-4 times) so we can arguably infer that the problem is not severe and does not outweigh the benefit of load informing strategy.

VII. Threats to Validity

First of all, figure 8 shows each time each controller informs its load to other controller when its load crosses the segments. Comparing this with Figure 6, we observed that the logging message disappears after around 150. In addition, before 50s mark in Figure 8, there were some inform messages that were not reflected in Figure N, which means that some of the messages did not reach the other controllers. These are the errors that we were not able to resolve due to time limitation.

Secondly, we could not get the source code from the authors of Load Balancing mechanism based on Load Informing Strategy so we implemented the mechanism following the description in [1] as similar as we can. However, we

cannot rule out the fact that our implementations may differ from what [1] did, which can make the simulation result inaccurate.

Lastly, in our simulation, we used ping command to simulate a packet from one host to another. The command sends a packet from host 1 to host 2 and waits for another ACK packet from host 2 to host 1. This can make the simulation result again, less accurate. Instead, what we could have done is to use another command to send a UDP packet between hosts for a more precise simulation.

VIII. Conclusion

Load Fluctuating problem, which can cause distributed SDN controller to inform load unnecessarily does happen in real scenarios. However, such situations are scarce and does not outweigh the benefit of Load Informing Strategy.

In our future work, we suspect that more exposure towards other possible values of CT (Controller Threshold) and alpha in the switch selection formula may expose more about load fluctuating problem.

References

[1] Jinke Yu, Ying Wang, Keke Pei, Shujuan Zhang, Jiacong Li, "A Load Balancing Mechanism for multiple SDN Controllers based on Load Informing Strategy," in APNOMS 2016

[2] Yuanhao Zhou, Mingfa Zhu, Limin Xiao, Li Ruan, Wenbo Duan, Deguo Li, Rui Liu, "A Load Balancing Strategy for SDN Controller based on Distributed Decision," in IEEE TrustCom, 2014.

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

[4] http://mininet.org/.

[5] http://www.projectfloodlight.org/floodlight/.

[6] http://jgroups.org/.