Balancing Load on SDN Controller Under High Load

Madhukrishna Priyadarsini
School of Electrical Sciences
IIT Bhubaneswar
Email: mp18@iitbbs.ac.in

Shailesh Kumar
School of Electrical Sciences
IIT Bhubaneswar
Email: sk38@iitbbs.ac.in

Padmalochan Bera
School of Electrical Sciences
IIT Bhubaneswar
Email: plb@iitbbs.ac.in

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 that works under high load condition. There are proposed work in this field, however all of them seem to fail or becomes liability when there is high load on whole network. In this paper we present some optimization to balance load under high load condition.**

1. 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. Both centralized and single controller have some bottleneck depending upon their deployments in real life implications like uneven load distribution, when mapping between a switch and a controller. This limitation will lead to degraded network performance. In order to handle this kind of issues, we propose a novel load balancing algorithm, which addresses the limitation of both single and multiple controller in load balancing. our algorithm gives better performance under high load condition

2. Related Work

There has been several research work in this area, starting from centralized load balancing approach to distributed load balancing and later on load informative strategy for load balancing.

2.1. Centralized Load Balancing

In Centralized load balancing approach a single controller is responsible for doing load balancing, it periodically collects the loads from all other controller, informs the overloaded controller to transfer some of its load to a lightly loaded controller. The key issue in this approach is latency which includes the time elapsed since a controller gets highly loaded, to load collection done by the master controller and then sending the load migration command to the target controller and then doing load migration by the target controller to the lightly loaded controller. Load on a controller changes dynamically and our load balancing strategy lags behind the real load on the target controller.

2.2. Distributed Load Balancing

In distributed load balancing approach DALB[] for load balancing where each controller does its own load balancing. In this method there is a thresh-hold defined for each controller up to which it does not need load balancing. When load on a controller increases its thresh-hold it does load collection and then it may go for load balancing if following conditions satisfies
i) its load is highest among all controller
ii) load is not uniformly distributed, that is $\rho < 0.7$,

$$where \rho = \frac{\sum_{i=1}^{n} L_i}{n * L_{max}}$$

otherwise it would set its new thresh-hold to average of loads of all controller. If the controller goes for load balancing it just need to do the load migration, thus it saves the time spend in sending load migration command as it was in centralized approach. But still it need to do load collection which takes time before balancing the load. Second problem is when the load is uniformly distributed and it is greater than thresh-hold it does load collection and sets its thresh-hold to average load of all controllers. In this case almost half of the controllers would be doing load collection continuously, because of their load greater than thresh-hold. This would consume bandwidth available for

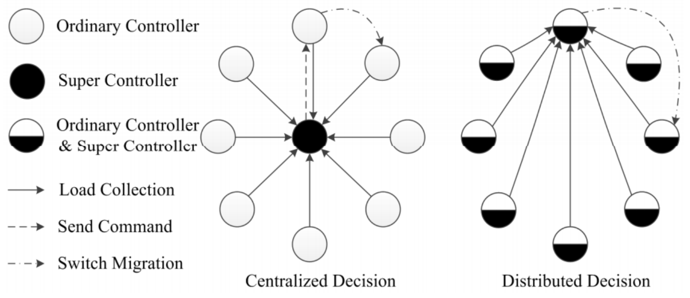

Figure 1. **Centralized and Distributed Load Balancing**

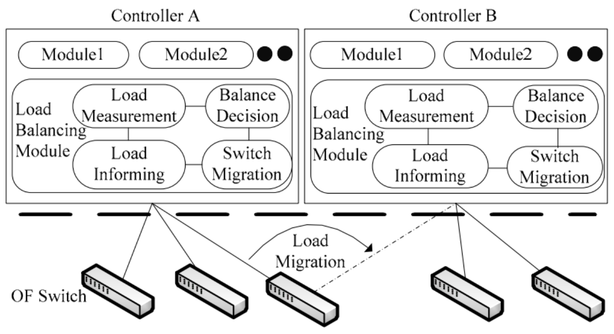

Figure 2. **Load Informative Architecture**

the data plane because of large number of controller O(n^2) message exchange between controllers.

This problem can be solved with a simple modification in the protocol. When a controller with highest load (greater than thresh-hold) does load collection and sees load is uniformly distributed, broadcasts the new thresh-hold to be set by all other controller which is equal to its own load. Thus number of control messages can be reduced from O(n^2) to O(n). But latency due to load collection still exists there.

2.3. Load Informative Strategy

In this method each controller broadcasts its own load so that any controller does not require to do the load collection thus saving time and doing load balancing as quickly as possible. The load scale from 0 to thresh-hold is segmented into many pieces (V_0 = 0, V_1, V_2, ... V_n = Thresh-hold), where segment size decreases gradually that is

$$|Vi - Vi - 1| < |Vi + 1 - Vi|$$

. Let L_{former} be the previous load that was broadcasted and $L_{current}$ is the current measured load, then load would be informed or broadcasted only if two of them lie in two different segments. Now every controller has load information about others but with some error, because it is not the current loads of controllers but informed loads. This error in load value is more if the load is low because of large segment size and less if the load is high because of small segment size. Segments sizes have been intentionally kept decreasing so that a slight deviation in load when the load is high is informed. Thus all controllers will have load of all other controller with very little deviation, if the load corresponding to that controller is high, and slight deviation if the load corresponding to that controller is low.

But there are following limitation in the above mechanism
i) If there is frequent deviation of load around any segment boundary then load would be frequently informed consuming bandwidth
ii) When the load of all controllers are near but less than the thresh-hold, load informing would take place too frequently.
iii) There is no mentioning of working of load balancing when all controller's load increases its thresh-hold value.

3. Proposed Work

This design is an extension to Load Informative Strategy, where we try to sort out its limitations as well as provide some improvements over previous design. There are four major components of this design namely Load Measurement, Load Informative, Balancing Decision, Load Migration.

3.1. Load Measurement

Load on a controller is measured using queuing model. We store the count of packets of different switches that are present in queue to be server by the controller. Let n be the queue size, (c_1, c_k, ... c_k) be the counts of packets of k switches, and r be serving rate is the rate at which packets are being served by the controller.

$$n = \sum_{i=1}^{k} c_k$$

$$r = \frac{t1 + t2}{2}$$

where t_1 and t_2 is time taken to serve the last packets.
Load of a controller is calculated as

$$L = n * r$$

Load due i^{th} switch is calculated as

$$L_{switch_i} = \frac{L * ci}{n}$$

In the previous designs DALB[] and Load Informative[] message arrival rate is used to represent load a controller. Consider the case of two controller C_1 and C_2, with message arrival rate 1000 and 500 per second. Thus C_1 is highly loaded compared to C_2. But there may be case that controllers have different hardware capabilities and hence packet service time of C_1 is 1 ms and that of C_2 is 5ms. In that case a new packet arriving at C_1 has to wait 1 second and 2.5 seconds at C_2. Hence the proposed load measurement technique can accurately measure the load, independent of underlying hardware used for the different controllers.

3.2. Load Informing Component

This component informs the load of a controller to other controllers. Assuming the following abbreviation
BT : base thresh-hold, let say 1000(millisecond)
CT : current thresh-hold, initially set to equal to BT.
$L_{informed}$: the load informed, initially set to 0.
$L_{current}$: the current load
δ : max load deviation = 10% of BT

Algorithm 1 Load Informative Algorithm

```
$L_{current}$ = GetCurrentLoad();
if $|L_{current} - L_{informed}| > \delta$ then
    BraodcastLoad($L_{current}$)
    $L_{informed} = L_{current}$
    return 0;
else
    return 0;
end if
```

3.3. Balancing Decisions

This is the main component that takes decisions of balancing load and adjusting the Current Threshold value for the whole network. It checks 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. If a heaviest overloaded controller can't do load balancing because of load being uniformly distributed, it broadcasts new Current Thresh-hold equal to its own load. This component is responsible for adjusting the Current Thresh-hold value according to situation and bringing it down to equal to Base Thresh-hold when load on network decreases.

3.3.1. Load Balancing. In the process of load balancing, another problem is inevitable. If two or more controllers exceed their Current 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 above the issue, only the heaviest overloaded controllers among overloaded controllers is allowed to do load balancing.

We take the lightest lightly loaded controller as the target controller, if its load is ¡ 70% of the Current Threshold then we go for load balancing otherwise broadcast the new Current Threshold to be equal to the heaviest loaded controller's load. If it satisfies the above condition then then we select a switch with highest load such that it satisfies following condition

$$L_{switch} < \frac{L_{overloaded} - L_{target}}{2}$$

3.3.2. Adjusting Current Threshold value. As we are dynamically increasing the Current Threshold value so as to work load balancing even in high load condition, we need to make sure to decrease it when the network gets lightly loaded and set it the base threshold value again. If the Current Threshold is greater than Base Threshold, heaviest loaded controller may reduce the Current Threshold and set it equal to max($L_{heaviest}$, Base Threshold) if

$$CurrentThreshold - L_{heaviest} > \rho || L_{heaviest} < BaseThreshold$$

where ρ is a fixed value(20% of Base Threshold)

Algorithm 2 Balancing Decision Algorithm

```
if $|L_{current}$ is heaviest then
    if $L_{heaviest} > CT$ then
        if $L_{lightest} < 0.7 * CT$ then
            Do the load-balancing
        else
            BroadcastCurrentThreshold($L_{heaviest}$)
        end if
    else
        return 0;
    end if
    if $CT > BT$ then
        if $L_{heaviest} < BT$ then
            CT = BT
        else
            if $CT - L_{heaviest} > \rho$ then
                CT = $L_{heaviest}$
            else
                return 0;
            end if
        end if
    else
        return 0;
    end if
else
    return 0;
end if
```

3.4. Load Migration

Firstly, the heaviest overloaded controller C_1 triggers switch migration by sending a switch migration request message to the target controller C_2. Then, C_2 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_2 , C_2 informs C_1 that its role change is finished. When the completion of the role-change process, controller 2 can also receive asynchronous messages from S_0. Secondly, C_1 cannot become the slave immediately from managed S_0 because there may be unfinished request at C_1 before receiving the reply for migration from C_2 So C_1 continues to interact with S_0 to complete undone work until it sends end migration to C_2. Thirdly, after receiving the end migration message, C_2 changes its role from equal to master by sending a ROLE_REQUEST message to S_{10}. And S_0 sets C_1 to slave. Finally, both controllers update the stored controller- switch mapping synchronically. The whole switch migration process is completed.

4. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.