An Efficient Load Adjustment for Balancing Multiple Controllers in Reliable SDN Systems

Kai-Yu Wang * Shang-Juh Kao * and Ming-Tsung Kao †

Department of Computer Science and Engineering, National Chung Hsing University, Taiwan

E-mail: g105056104@mail.nchu.edu.tw

Abstract

Software-defined networks (SDNs) are flexible for use in network traffic routing determination because they separate control and data planes. OpenFlow—the de facto protocol for SDNs—requires a dedicated SDN controller. When a network's coverage increases, a system based on a single SDN controller can experience a severe bottleneck. Using multiple controllers in an SDN may solve the scalability problem. In this paper, to achieve load balancing among controllers, we proposed a load adjustment mechanism for application to each controller. The proposed mechanism comprises three logical components: a load collector, load balancer, and switch migrater. Cbench was used to generate traffic to simulate various loads for each switch, and the experimental results confirmed that the data collection procedure and adjustment of global and local loading were efficient.

Keywords—OpenFlow, SDN, Multiple controllers, Load balance

I. INTRODUCTION

The software-defined network (SDN) model based on OpenFlow [1] is one of the most promising paradigms among network developments. However, in southbound SDN interfaces, use of a single SDN controller may result in a bottleneck when the system is presented with large volumes of incoming packets. Accordingly, scalability and a single point of failure represent potential challenges to SDN usage [2]. Network traffic is dynamic; therefore, efficient adjustment of traffic forwarding paths to evenly distribute, if not balance, traffic loads among various network paths and controllers requires investigation. In this paper, a multicontroller architecture is proposed.

A centralized or distributed approach may be adopted to balance loads among multiple controllers. The centralized approach involves a dedicated super controller equipped to collect the loading statuses of all other controllers and coordinate traffic distribution. When the super controller fails or is overloaded, the entire network succumbs to a single point of failure; this is the same problem that occurs when a single controller is used. By contrast, in the distributed approach, every controller acts as a super controller; loading information is exchanged among all controllers and each controller registers the loading statuses of the others. Hence, a heavily loaded controller can efficiently redirect incoming packets. In this study, we adopted a distributed approach to balance load among multiple controllers. In the proposed model, the reliability problem is solved through the use of multiple controls and the distributed approach. If a controller suddenly fails, the system can be recovered through redirection of all communications of the failed controller to normal operational controllers.

II. RELATED STUDIES

In a centralized approach to network load balancing, a super controller is used to balance the loads of all controllers, as in the system designs proposed in BalanceFlow [3] and ElastiCon [4]. These proposed systems both involve a logically centralized load adapter that balances loading among controllers in a controller pool. However, collecting load information and sending load-balancing commands may burden the super controller. If the super controller fails, packets can be lost or delayed. In the distributed approach, every SDN controller collects the other controllers' loading and locally decides subsequent actions. Thus, each controller acts as a subordinate and super controller. The distributed architecture guarantees scalability and availability of distributed SDN controllers [5]. Hence, in the proposed model, we adopted the distributed approach to manage multiple controllers. Despite the advantages of the distributed system, the following challenges required resolution.

(1) The bandwidth between controllers adds extra load to information delivery. In the distributed approach, all controllers must promptly collect information from the other controllers, and the extra load from the bandwidth may interfere with efficiency. This problem was mentioned in a previous study [6], but a relevant simulation was not presented.

(2) Load migration among controllers is the most critical function in the SDN system. The system network environment and traffic intervals vary, resulting in dynamic load changes among switches and controllers. Therefore, for the proposed model, an efficient load-balancing algorithm was developed to address the complex network environment.

(3) No studies of multiple controller systems have addressed the reliability problem. If one controller fails, another controller must assume the failed controller's tasks, otherwise the switches associated with the failed controller do not function and network traffic is negatively influenced. To avoid a single point of failure, we integrated a failure recovery mechanism into the proposed system.

III. LOADING ADJUSTMENT SYSTEM DESIGN

To balance load distribution among multiple controllers, overall loading information must be processed, and the number of application flows that require redirection must be determined. When a controller is overloaded, the system initiates balancing operations and shifts packet flows to a controller with a lighter load. Hence, the load adjustment system in the proposed SDN network system relies upon coordination among all controllers. In the proposed system, three logical components were included in each controller, namely a load collector, load balancer, and switch migrater, as shown in Fig.1.

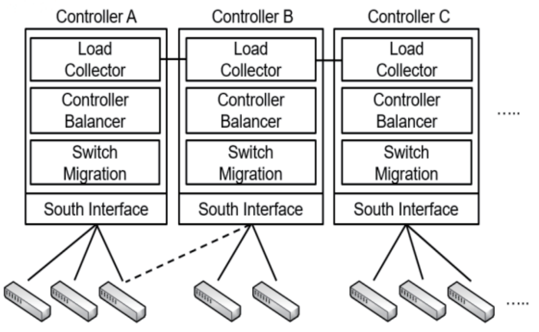

Fig.1 Architecture of the Distributed SDN System

The load collector in each controller periodically activates to collect loading statuses from other controllers and share its loading information. Each controller records its identification, loading information, associated switches, and most recent update time. A loading threshold (T) comprises 70% of controller bandwidth. A 0–70 interval measuring loading capacity is divided into three subintervals according to 4:6 ratios, that is, 70 to 60, 60 to 42, and 42 to 0. When a loading value crosses a subinterval boundary, the controller notifies its neighboring controllers of its loading change. Loading is classified according to the following scheme.

Algorithm 1 The Classify Algorithm

```
1:   Input:
2:       $L_{current}$: current load value
3:       $L_{former}$: former load value
4:       $V_0$=0, $V_1$=42, $V_2$=60, $V_3$=70
5:   Output:
6:       True or False means informing or not
7:     Value = False
       if ($L_{current}$<threshold)
8:       if ($L_{former}$<42 && $L_{current}$>42)
9:               Value = True
10:              break
11:         end if
12:         if ( ($L_{former}$>42 &&$L_{former}$<60) &&
     ($L_{current}$<42 ||$L_{current}$>60) )
13:              Value = True
14:              break
15:         end if
16:         if ( ($L_{former}$>60 && $L_{former}$<70) &&
     ($L_{current}$<60) )
17:              Valus = True
18:              break
19:         end if
20:    end if
21:    return Value
```

Once the load of a controller exceeds the threshold, T, the controller balancer determines the controller with the lightest load as the target to share the overloaded controller's load. Because all controllers share the overall loading information, the determination for an alternative load route is simple. Ideally, the primary task of the controller balancer is to rearrange loads so that all loadings remain in the first or second interval. The balancing algorithm is outlined as follows.

Algorithm 2: The Balancing Algorithm

```
1:    Input:
2:            $L_M$:migrated controller (threshold > 70%)
3:            $L_T$: target controller //the lightest controller
4:            M_switch=sort {$switch_1$, $switch_2$,...$switch_n$},
      switch will be sort from small to large
5:    Output:
6:            the target controller and $M_n$
7:    M= ($L_M$-$L_T$)/2
8:    if ($M_1$>M)
9:              not suitable switch for migration
10:   end if
11:   else
12:     for (i: 1 to n)
13:         If ($L_T$+$M_i$) >threshold
14:             $M_n$=$M_{i-1}$
15:             Return $M_n$
16:         end if
17:   end if
```

If the load of a controller exceeds the threshold, the balancing procedure is activated. First, switches are sorted in order of increasing load. Subsequently, the switch with the largest load is chosen for migration to the controller with the lightest load. The proposed balancing scheme is greedy to offload as much as possible from the overloaded controller. The balancing scheme is also conservative because the possibility of multiple controllers being designated to receive additional loads or multiple switches to be distributed is not considered.

Controller synchronization is a concern related to selection of and load redistribution to the lightest loaded controller. To avoid simultaneous migration assignment to the same controller, the balancer module processes the information of one controller at a time. One method for resolving the multiple migration problem is to defer the operation of the load balancer. When an overloaded controller registers another overloaded controller on its load record, the controller's subsequent actions may be delayed for a brief period such as 5 seconds. After this delay, the loading status can be reassessed before the balancing process is restarted.

When source and target controllers are determined, the switch migrator of the source controller notifies its associated switch to redirect the forwarding path to the target controller. After completing the migration, the load collector continues to monitor the overall loading status. The operating flow of the proposed system is depicted in Fig.2.

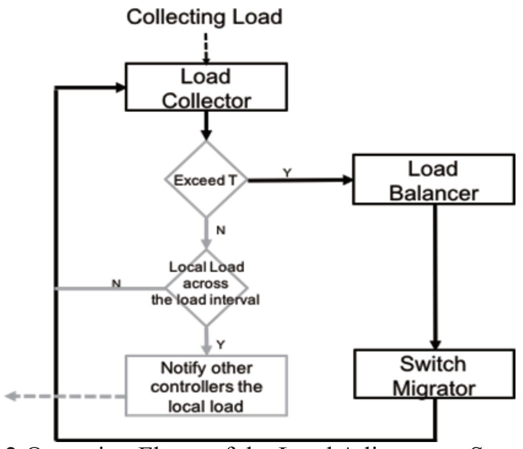
Collecting Load
Load Collector
Exceed T
Y
Load Balancer
N
Local Load across the load interval
N
Y
Notify other controllers the local load
Switch Migrator

Fig.2 Operating Flows of the Load Adjustment System

ISBN 978-1-5386-4342-6

In addition to load balancing, the proposed system provides a standby mechanism to enhance system reliability. We designated a standby controller to ensure that smooth whole-system recovery is possible if a controller fails. Communications among the controllers during the notification and migration phases enable the controllers to register the survival status of peer controllers. A peer controller that registers an out-of-service controller may rewrite forwarding rules so that loads directed to the orphaned switch may be redirected to a candidate controller; this mechanism is similar to that of the load balancer.

IV. SIMULATION AND EVALUATION

The experimental environment for testing the load adjustment system was developed using ONOS, Cbench, and Mininet. ONOS is a Java-based OpenFlow controller. We used Cbench to benchmark the performance of OpenFlow controllers. Cbench also generated incoming packet requests for OpenFlow switches. Mininet was adopted to emulate SDN network topology. Two simple experiments were conducted to evaluate the proposed mechanism's balancing effect and failover recovery. The topology in Linux Ubuntu 14.04 is illustrated in Fig.3.

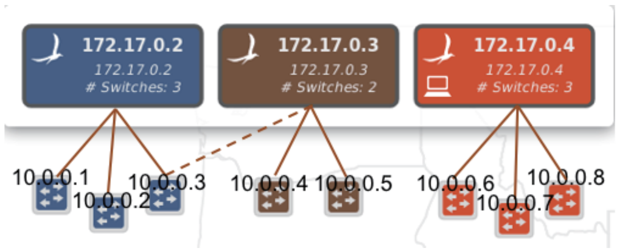

Fig.3 Simulation environment

As shown in Fig.4, at 90 seconds, the load of controller C crossed the threshold. Based on the balancing algorithm results, the lightest controller, B, was chosen for load sharing. The traffic from switch 1, which was associated with controller C, was redirected to controller B, and at 100 seconds, the loads of controller B and C were approximate. At 280 seconds, controllers A and B were overloaded, but the loading figures for peer controllers did not meet the balancing criteria; therefore, both controllers maintained their overloads. From 430 to 460 seconds, the same switch was migrated back and forth between controllers A and C, indicating that load balancing may involve a tradeoff of bandwidth conscription.

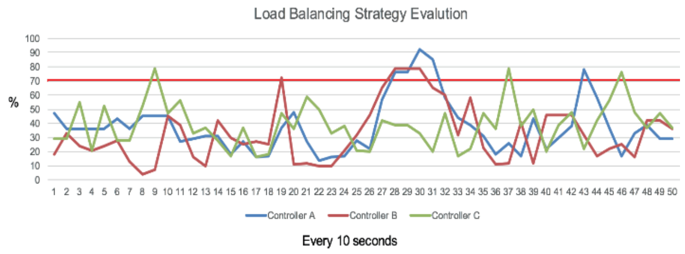

Fig.4 Load Balancing Strategy Evaluation

We used the request and reply feature of Internet Control Message Protocol to test the proposed system's capacity for fail recovery. To initiate this test, we intentionally shut down controller A. As represented in Fig. 5, ping requests 23–27 elicited no response. At ping 28, the failed switch responded because the standby controller accounted for the associated switches. When the load collector in a peer controller registered that controller A had been shut down, it notified the switch migrator to send outbound packet flow to the switches associated with the broken controller.

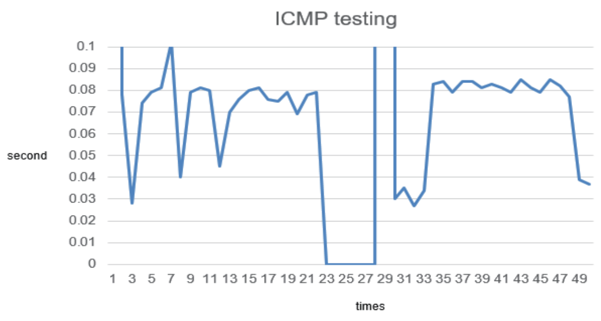

Fig.5 Failover Testing

V. CONCLUSIONS

This paper presents a method for efficient adjustment of traffic flows to achieve load balance among multiple controllers in SDN systems. Using an in-band environment, that is, a design in which all switches connect to one controller through a dedicated switch, we developed a load adjustment mechanism comprising a load collector, load balancer, and switch migrater. In this mechanism, each controller launches its loading data to the other controllers through a process designed to prevent as much unnecessary communication as possible. When the controllers have registered global load status, overloaded controllers appropriately reroute the traffic of their associated switches to controllers with lighter loads. An experimental system was used to demonstrate and test the balancing function of the proposed adjustment mechanism. ONOS, Cbench, and Mininet were used for simulations. The experimental results confirmed that the standby controller solved the reliability problem; recovery after single controller failure was successful.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14-76, 2015.

[2] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *Journal of Computer Networks and Communications,* vol. 2016, 2016.

[3] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Balanceflow: controller load balancing for openflow networks," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, 2012, vol. 2, pp. 780-785: IEEE.

[4] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Architectures for Networking and Communications Systems (ANCS), 2014 ACM/IEEE Symposium on*, 2014, pp. 17-27: IEEE.

[5] Y. Zhou *et al.*, "A load balancing strategy of sdn controller based on distributed decision," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, 2014, pp. 851-856: IEEE.

ISBN 978-1-5386-4342-6

[6] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, 2016, pp. 1-4: IEEE.
[7] *ONOS SDN controller*. Available: https://wiki.onosproject.org/
[8] *Cbench*. Available: https://github.com/mininet/oflops/tree/master/cbench
[9] *Mininet*. Available: http://mininet.org/

ISBN 978-1-5386-4342-6