A Load Balancing Model based on Load-aware for Distributed Controllers

Fengjun Shang, Wenjuan Gong

College of Compute Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China

E-Mail: Shangfj@cupt.edu.cn

Keywords: Software-Defined Networking; Load-aware; Load Balancing; Failure Recovery

Abstract. Software-Defined Networking (SDN) is a new paradigm that decouples the control plane from data plane. But with the continuous expansion of the network scale, a single controller faces with scalability issues and may trigger a single point of failure. It is important to take into account the scalability of the control plane and how to balance the load of multiple controllers. In this paper, we study and analyzed the architecture model of multiple controllers and then proposed a load balancing model based on load-aware for distributed controllers which can flexibly adjust the flow-requests handled by each controller based on the flow-request, and resolved the single point of failure through fast recovery of failure.

Introduction

Recently with the continuous expanding of network scale and the increasing of network interconnection equipment and classes, traditional Internet faces many challenges such as controllability of management、scalability of scale and validation of innovative experiments[1]. Software-Defined Networking (SDN) architecture is currently seen as one of the most promising paradigm where the network control is decoupled from data plane. Control plane is consisted of a logically centralized controller which is responsible for making packet forwarding decisions and providing a programmatic interface to the application plane. Data plane, which is responsible for forwarding packets, is composed of physical devices. In the SDN architecture, the data plane is controlled by the control plane through a well-defined API. SDN has already some concrete implementations. OpenFlow [2] is an open standard protocol, specifically designed for the SDN networks, that allows the communication between the control and data planes and permits the manipulation of the latter. At the beginning, the OpenFlow protocol was developed at the Stanford University around 2008 for enabling researchers to run experimental protocols in the campus networks, but now has received wide attention from academia and industry.

In SDN architecture based on OpenFlow, the OpenFlow switches send flow-requests to the controller via the OpenFlow protocol over a secure channel and the controller sets packet forwarding rules to switches via a global view of the network. The centralized control function of SDN has reduced the complexity of network management and configuration, but with the continuous expanding of network scale and increasing demand for services, the centralized controller needs to deal with massive flow-requests from switches and the scalability problem of control plane is becoming more and more serious [3]. The paper [4] points out that the NOX controller can handle 30K flow every second and each flow needs an average of 10ms [5], however the data center including 100 switches can generate 10M flow-requests every second which exceed the capability of controller [6]. So researchers has proposed many architecture models of distributed controller. In 2010 the paper [3] proposed a distribution of control plane called HyperFlow. HyperFlow is logically centralized but physically distributed. By passively synchronizing network-wide views of OpenFlow controllers, the switch is connected to the nearest controller, thus minimizing the control plane response time to data plane requests. The paper [7] present Onix, a distributed system based on control plane. Control planes written within Onix operate on a global view of the network, and use basic state distribution primitives provided by the platform. Onix provides a general API for control plane implementations, while allowing them to make their own trade-offs among consistency, durability, and scalability. In 2012, the paper [8] designed a high scalable mechanism named MSDN to balance the data flow initialization requests and then those requests were parallel processed with a share global network view. In the same year, Yannan Hu proposed BalanceFlow [9], a controller load balancing architecture for OpenFlow networks with by utilizing CONTROLLER X action extension for OpenFlow switches. The "super controller" can flexibly tune the flow-requests handled by each controller, without introducing unacceptable propagation latencies architecture.

The Load Balancing Model based on Load-aware for Distributed Controllers

BalanceFlow architecture [9] used "super controller" to run flow-requests partition when controller load imbalance is detected. Once the "super controller" has a fault, the flow-requests partition function would be disable. So to resolve the above problem, we present the load balancing model based on load-aware for distributed controllers. In the model, each switch connects with multiple controllers but at any moment is controlled by only one controller. And one controller can manage more than one switch. All controllers are of the same performance without "super controller", so the model resolves the problem of single point failure, improves the stability of the network; meanwhile we propose load balancing algorithm based on load-aware random assignment that ensures that the network does not trigger a larger tilt when partitions the flow-requests. Besides in order to detect the failure of the controller, the controller will send heartbeat packets to each other. If a controller dos not sent a heartbeat packet in the specified time, then the other controllers deem that the controller has failed and will immediately run the load balancing algorithm to partition the flow-requests. The architecture model is shown in Fig.1.

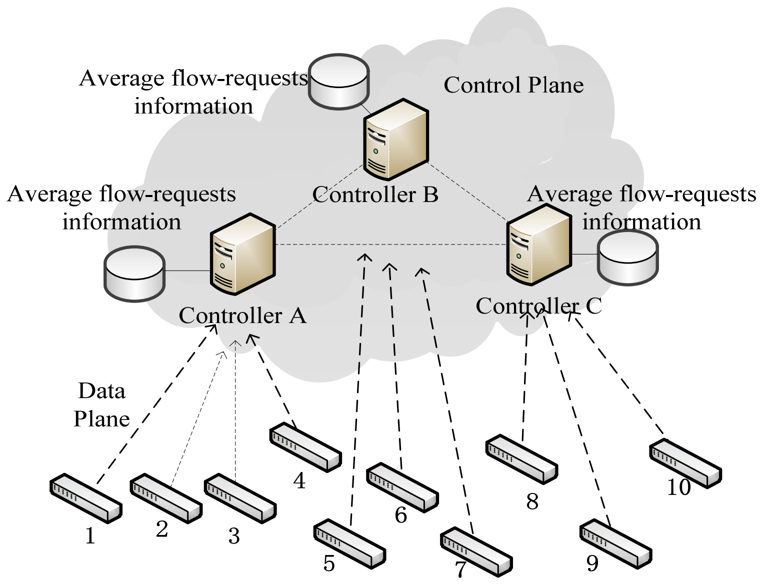

Average flow-requests information
Control Plane
Average flow-requests information
Controller B
Average flow-requests information
Controller A
Controller C
Data Plane
1
2
3
4
5
6
7
8
9
10

Fig.1 The Load Balancing Model Based on Load-aware For Distributed Controllers

Load Information Awareness.

In the load balancing model presented by this paper, the load information of the controller is mapped to the average number of flow-requests handled by it. The load information awareness points out that each controller can aware the load status autonomously through calculating the ratio of average number of flow-requests to the total number of flow-requests in the network. And using the flow-requests based on switch pairs is viable since the total number of switch pairs is usually limited with less storage space.

So every controller maintains an $N*N$ order matrix Q_{avg} which represents the load information of controller, where N is the number of switches in the network. The element in the ith row, jth column denotes the average number of flow-requests from switch i to switch j. When a controller receives a flow-request, it checks the destination address and source address and locates the corresponding egress switch for that flow and the relevant element in the matrix is updated periodically. The average number of flow-requests from switch i to switch j is calculated using the following Eq. 1.

$$q_{avg}(i,j)=(1-\delta)q_{pre}(i,j)+\delta p(i,j) \tag{1}$$

Where δ is a weighted coefficient, $q_{pre}(i,j)$ is the average flow-requests number of previous cycle and $p(i,j)$ is the number of flow-requests from switch i to switch j in a certain cycle.

Load Balancing Algorithm based on Load-aware Random Assignment.

This paper proposes the load balancing algorithm based on load-aware random assignment where all controllers publish average flow-requests information to each other periodically, calculate the total number of flow-requests. If the average number of flow-requests handled by any controller contributes more than some threshold of the total flow-requests rate in the network, the controller will aware that the load is imbalanced. Then the controller of load imbalance runs the load balancing algorithm which reallocates flow-requests to multiple controllers via allocation probability.

Related Concept Description.

(1) load degree of switch pair to controller

Firstly we suppose the distributed controller model has k controllers ($k \geq 2$), $L(s)$ is the ratio the average number of flow-requests of controller s to the total number of flow-requests Q_s in the network R_{total}, calculated by Eq. 2.

$$L(s) = Q_s / R_{total} \qquad s=1,2,\cdots,k \tag{2}$$

So when $L(s)$ exceeds the given threshold, controller s runs the load balancing algorithm. On this basis, we propose the concept of the load degree of switch pair to controller $L_s(i,j)$, which denotes the load of switch pair (i, j) to controller s, using the Eq. 3.

$$L_s(i,j)=(Q_s+p(i,j))/R_{total} \qquad i,j=1,2,\cdots,n \tag{3}$$

Where $p(i,j)$ is the flow-requests that is about to be handled by the controller s. The algorithm assumes that every controller has same performance and select average number of flow-requests and the flow-requests which is about to be handled to measure the load of switch pair (i, j) to controller.

(2) load weight and load difference

We define $W_s(i,j)$ as the load weight of switch pair (i, j) to the controller s, using Eq. 3 to calculate:

$$W_s(i,j)=(1-\beta)L_s(i,j)+\beta d(i,s)/d_{avg} \qquad i,j=1,2,\cdots,n, s=1,2,\cdots,k \tag{3}$$

Where β is a constant parameter, adjusting the weights between controllers' load and propagation latencies. When setting β too small, we ignore the benefit of reducing latencies. When setting β too large, however, we do not require the load to be balanced. We empirically found that β between 0.075 and 0.15 gives a good result. d_{avg} is the average node-to-controller latencies.

Then we define $\Delta W_s(i,j)$ as the load difference of switch pair (i, j) to the controller s, which denotes the difference between the sum of all load weights of switch pair (i, j) to controllers and the load weight of switch pair (i, j) to the controller s, calculated by Eq. 4.

$$\Delta W_s(i,j)=\sum_{c=0}^{k}W_c(i,j)-W_s(i,j) \qquad i,j=1,2,\cdots,n \quad c=1,2,\cdots,k \tag{4}$$

(3) allocation probability of switch pair to controller

We define $P_s(i,j)$ as the allocation probability of switch pair (i, j) to the controller s. $P_s(i,j)$ denotes that the probability of flow-requests from switch pair (i, j) allocated to the controller s, calculated according to Eq. 5.

$$P_s(i,j) = \Delta W_s(i,j) / \sum_{c=0}^{k} \Delta W_c(i,j) \qquad s = 1,2,\cdots,k \qquad \sum_{s=0}^{k} P_s(i,j) = 1 \tag{5}$$

In order to further reduce the probability of the occurrence of tilt in the network when traffic bursts, we introduce random probability to divide the flow-requests. Through judging the random number [0, 1] falls on which section of the allocation probability of switch pairs to controllers, select the corresponding controller.

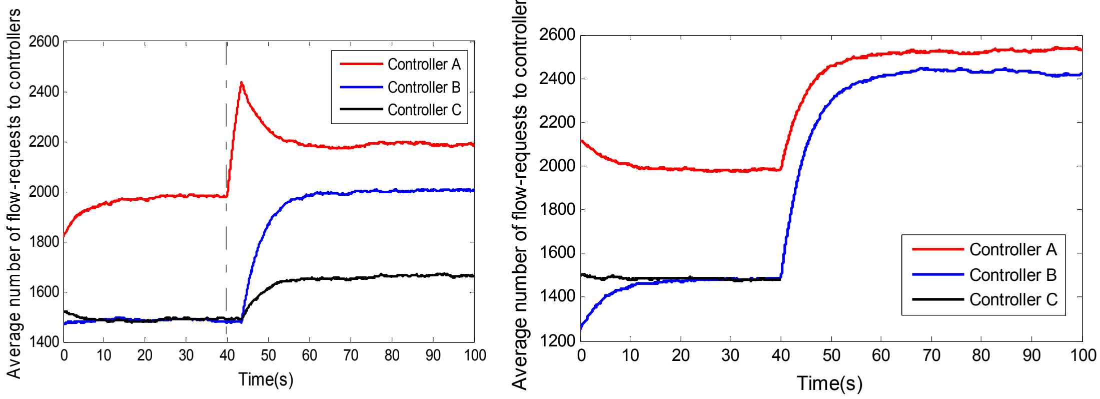
Procedure of Load Balancing Algorithm based on Load-aware Random Assignment.

After introducing the related concept of algorithm proposed by this paper, the complete operation process of load balancing algorithm based on load-aware random assignment is as follow:

1) Initializing the network state, divide the switches of the controller management.
2) Setting the controller load imbalance threshold $(\varepsilon - 0.1) * k = 1$.
3) Controllers update the respective average flow-requests matrix every cycle T_m.
4) Controllers send the respective average flow-request information to each other every cycle T. Calculate $L(s)$ of each controller. When $L(s) \succ \varepsilon$, the controller occurs load imbalance, runs the load balancing algorithm, then skips to step 5; when $L(s) \le \varepsilon$, the distributed controller system is in a state of balancing and skips to step 3.
5) Calculate allocation probability of switch pair (i, j) to the controller s, $P_s(i,j)$. Then repartition flow-requests from switch pair (i, j) to specified controller according to $P_s(i,j)$ via random probability.

Evaluation

In order to verify the feasibility of our distributed controller model, we built an experimental environment. We evaluate our model on Abilene topology, which contains 10 nodes and 13 links. In our experimental, each switch connects ten hosts and three controllers (denotes as A, B and C for simplicity in illustrating) are deployed in the network. We set T_m to 10ms and T to 1s and the controller load imbalance threshold is calculated according the number of controllers. The initial status of the network is shown in Fig.1. The default behavior of each switch is to forward its flow-requests to the nearest controller to achieve quick response, so controller A, B and C will receive the flow-requests from 4, 3 and 3 switches respectively. At time 0 second, we assume that the network has been in a basic steady status. Each host starts to send a packet flow to a randomly-chosen host. After that a host randomly waits between 1 and 10ms before sending a new flow. At time 40 seconds, we intentionally increase the load of controller A (by raising the rate of sending new flows which are handled by controller A). We repeated the test 20 times, and Fig.2 (a) plots the average load of the three controllers over time when all controllers are running normally, Fig.2 (b) plots the average load of the three controllers over time when some controller malfunctions.



(a) All controllers are running normally (b) some controller is broken

Fig.2 the average number of flow-requests handled by three controllers

According Fig.2, we can see, that as the hosts start sending traffic, the average flow-requests received by each controller ramp up, the division of load is relatively close to the 4:3:3. In Fig.2 (a),After 20 senconds, as expected, the load on controller A increases draamatically, and at time 44 second the load of controller A exceeds the controller load imbalance theshold 45%, so controller A runs the load balancing algorithm to reallocate the flow-requests. And then the load of each controller is well under imbalance triggering threshold. In Fig.2 (b), at time 40 second, the controller C is broken, so after 3 cycles of updating, the load balancing is invoked immediately and reallocate flow-requests to controller A and B. Then the load of the rest of controllers is well under imbalance triggering threshold.

Conclusions

This paper analyzes the scalability problem of SDN architecture and studies the architecture models of multiple controllers. And then propose a load balancing model based on load-aware for distributed controllers which can runs the load balancing algorithm to reallocate the load of each controller. Our experiment shows that the model can flexibly adjust the load of each controller and succeed in avoiding the single point of failure and guarantee that the function of load balancing algorithm is run normally whether controller has fault or not.

Acknowledgements

This work was supported in part by the Chongqing Natural Science Foundation under Grant No.cstc2012jjA40038, the Chongqing Basic and Frontier Research Project under Grant No.cstc2013jcyjA40023, the Ministry of Industry and Information Technology for the special fundes of Development of the Internet of things(2012-583) and the Special Foundation for Young Scientists of Chongqing(NO.cstc2014kjrc-qnrc40002).

References

[1]Zhang C K, Yong C, Tang H Y, et al. State-of-the-art survey on software-defined networking (SDN) [J]. Journal of Software, 2015.

[2] Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling Innovation in Campus Networks[J]. Acm Sigcomm Computer Communication Review, 2008, 38(2):69-74.

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

[4] Gude N, Koponen T, Pettit J, et al. NOX: Towards an operating system for networks[J]. Acm Sigcomm Computer Communication Review, 2008, 38(3):105-110.

[5] Tavakoli A, Casado M, Koponen T, et al. Applying NOX to the Data center[J]. Hotnets, 2009.

[6] Curtis A R, Mogul J C, Tourrilhes J, et al. Devoflow: Scaling flow management for high-performance networks[J]. Acm Sigcomm, 2011, 41(4):254-265.

[7] Koponen T, Casado M, Gude N, et al. Onix: A Distributed Control Platform for Large-scale Production Networks.[J]. Proc Osdi, 2010:351-364.

[8] LIN Pingping, BI Jun, HU Hongyu, JIANG Xiaoke. MSDN:a Mechanism for Scalable Intra-domain Control Plane in SDN[J].Journal of Chinese Computer Systems,2013,34(9):1969-1974.

[9] Hu Y, Wang W, Gong X, et al. BalanceFlow: Controller load balancing for OpenFlow networks[C]// Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on. IEEE, 2012:780-785.